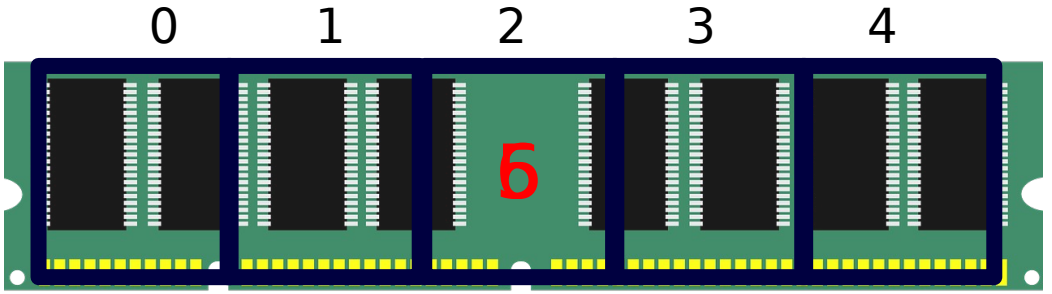
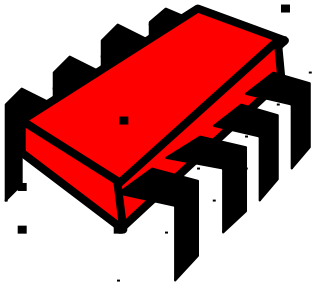
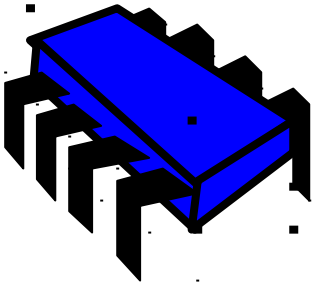
Concurrent Counting using Compare-and-Swap Registers

Pankaj Khanchandani, Roger Wattenhofer
ETH Zurich





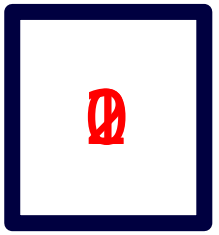
R_2.write(5)



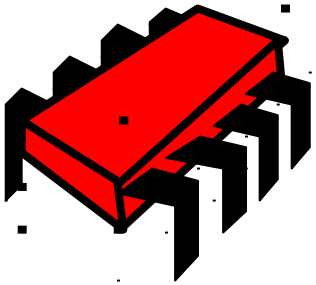
v = R_2.read()

v’ = v + 1

success = R_2.CAS(v, v’)

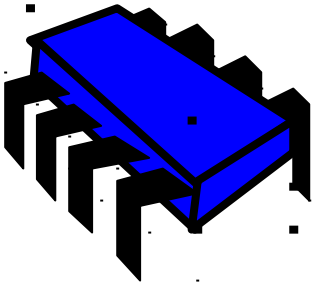


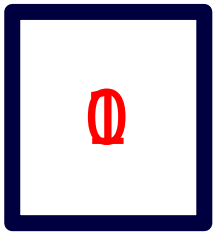
C

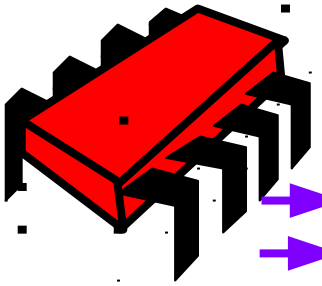


C.inc()

C.inc()
v = C.read()

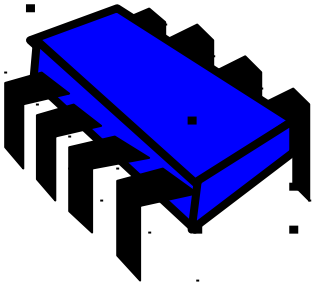




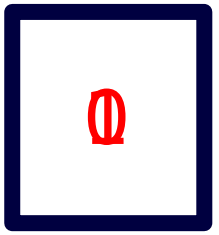


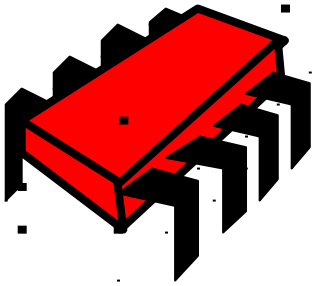
v = R.read()
R.write(v + 1)

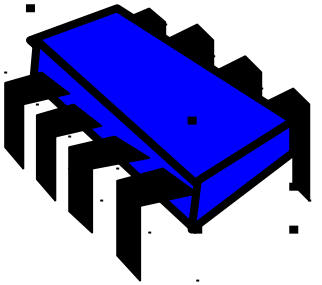
v = R.read()
R.write(v + 1)



NOT LINEARIZABLE





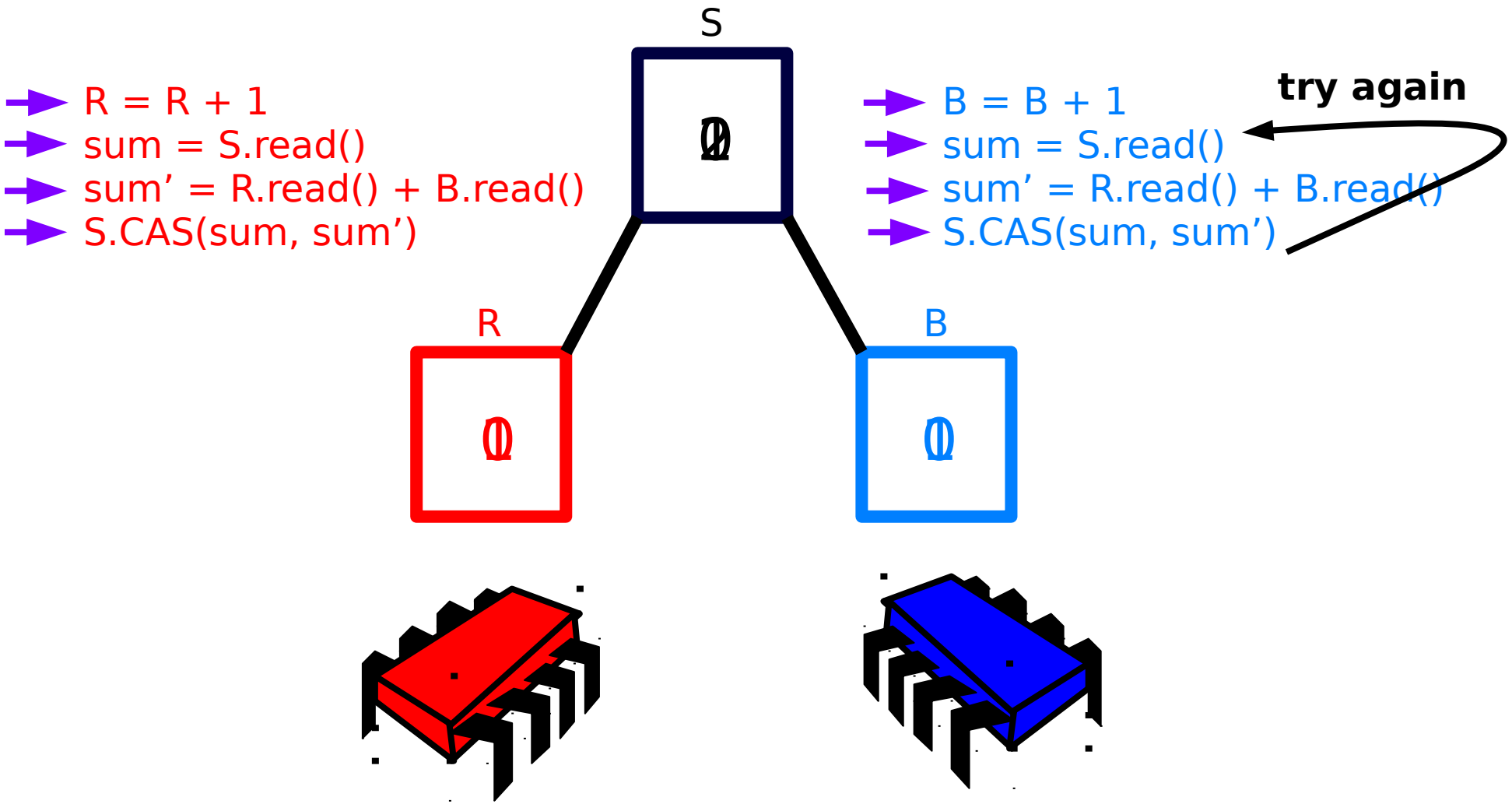


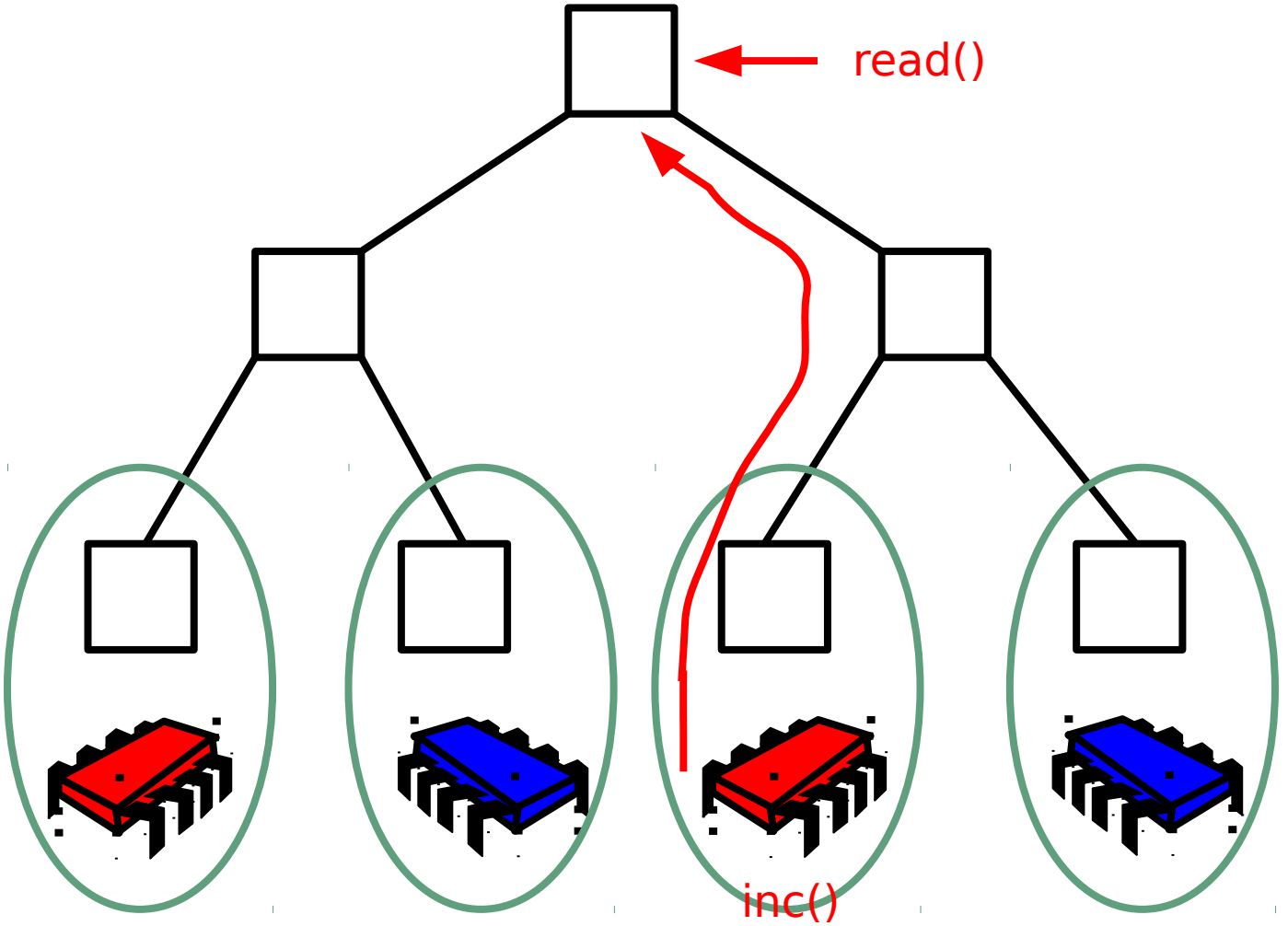
```
while(true) {
    v = R.read()
    if(R.CAS(v, v + 1)){
        break
    }
}
```

```
while(true) {
    v = R.read()
    if(R.CAS(v, v + 1)){
        break
    }
}
```

NOT WAIT-FREE

Linearizable
Wait-free
Shared Counter


S
0
R = R + 1
sum = S.read()
sum’ = R.read() + B.read()
S.CAS(sum, sum’)
B = B + 1
sum = S.read()
sum’ = R.read() + B.read()
S.CAS(sum, sum’)
try again
R
0
B
0


read()
inc()

Simple O(log n) implementation
Uses O(n) space
Ellen et al. Fetch-and-increment uses $O(n^2)$

inc(x)?
dec()?
fetch-and-increment() in O(n) space?