Synchronization
mutex
semaphore
monitor
lock
unlock

data structure

lock
update shared var

lock
do-something
unlock

lock
examine shared
unlock

Semaphore

activity
≤ k threads
at once

signal
up
form p
wait

Eg. Windows protocol in networking: send ACK signal after
receiving packet

\[ S = \text{semaphore}(k) \]

activity ()
  s. wait
  do something
  s. signal()

bounded buffer

\[ \boxed{1 \ldots 1} \leq k \text{ items} \]

it is similar to how a pipe works

put -> Go get

\[ s_1: \text{track empty slots} \]
\[ s_2: \text{track available items} \]

\[ S_1 = \text{semaphore}(k) \]
\[ S_2 = \text{semaphore}(0) \text{ wait (block) until someone acts on it} \]

"Signal" calls are stored
- similar to the green/red light in an intersection

? \[ \rightarrow \] \[ \rightarrow \] \[ \rightarrow \] \[ \rightarrow \] \[ \rightarrow \] \[ \rightarrow \] ok

ok

\[ \text{signal (green light)} \]
\[ \text{is already here} \]
put

wait(s1)

put entry in
signal(s2)

get

wait(s2)

g get it

signal(s1)

head

insert

= tail

head

remove

empty:

full locations: > tail, but < head

→ unlike with semaphores, with monitors there is no state associated →... 

seeks atomic operations

view from other thread

current thread

→ go

→ go

→ go

→ block

→ block
with semaphores, we need release, then block
we can’t do block, then release

queue = semaphore (0)
mutex = semaphore (1)
Nwaiting = 0

wait (mutex)
tmp = Nwaiting++
if Nwaiting = 2
    Nwaiting = 0
    signal (mutex)
else
    signal (g)

if tmp = 2
    sleep (g)
else
    wait (g)

Monitors

monitor:
    condition objects C1, C2
    wait (C1)
    signal (C1)
    broadcast (C1)

if you signal a condition that has no one waiting on it,
you missed – because it does not store state (doesn’t
keep any signal counter)

method:
    state 1
    wait ---- leaves monitor
    state 2 ---- can find it in a different state
monitor robot:

    bool am_first = true
    condition C

do_rendezvous():
    if am_first
        wait(C)
    else
        signal(C)

incorrect: no thread will signal; they both see am_first=true

monitor robot:

    bool am_first = T
    condition C

do_rendezvous():
    if am_first
        wait(C)
        am_first = false
    else
        signal(C)
        am_first = true

incorrect:

monitor robot:

    customer # = 0
    condition C
    do_rendezvous():
        customer # ++
if is odd (cust #1)
input (c1)
print customer woke up
else
signal c