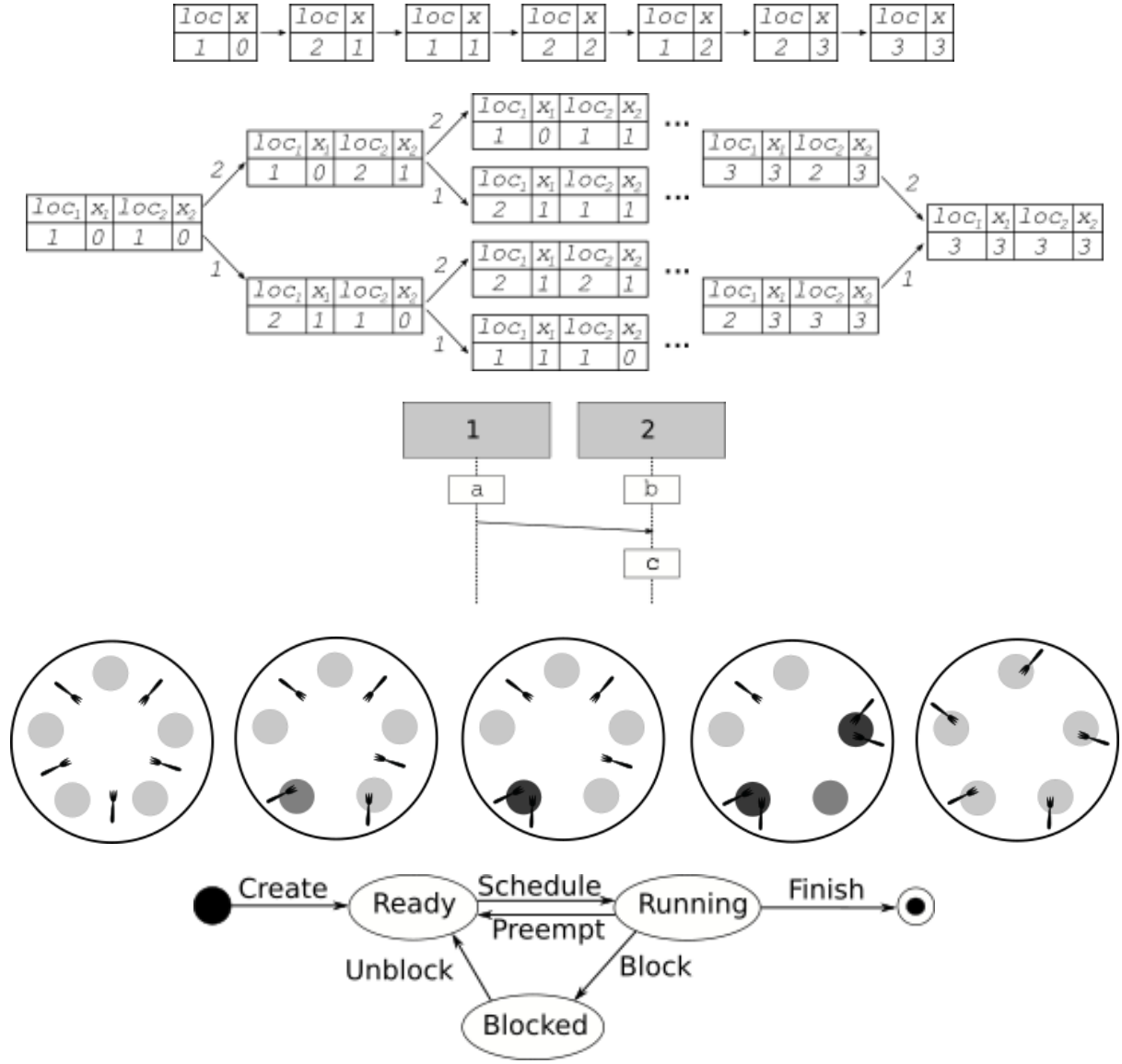
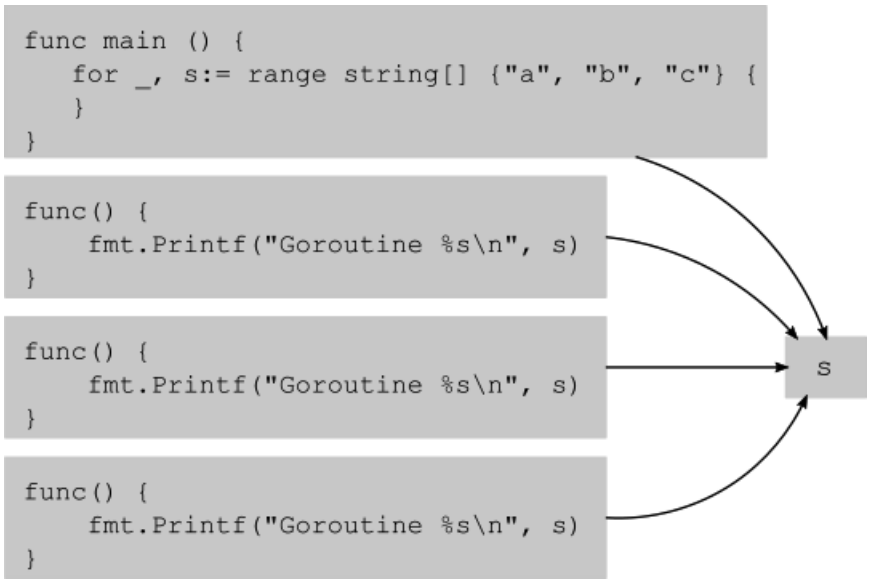
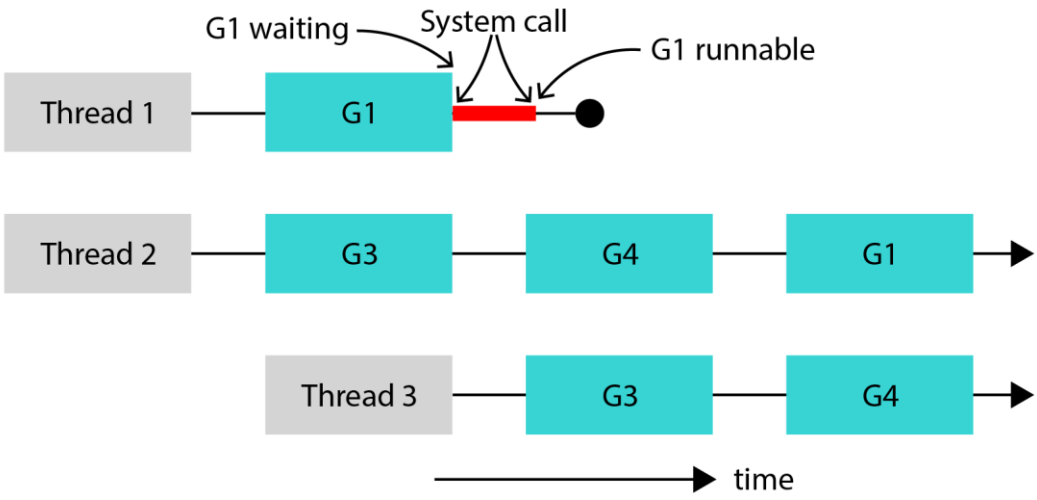
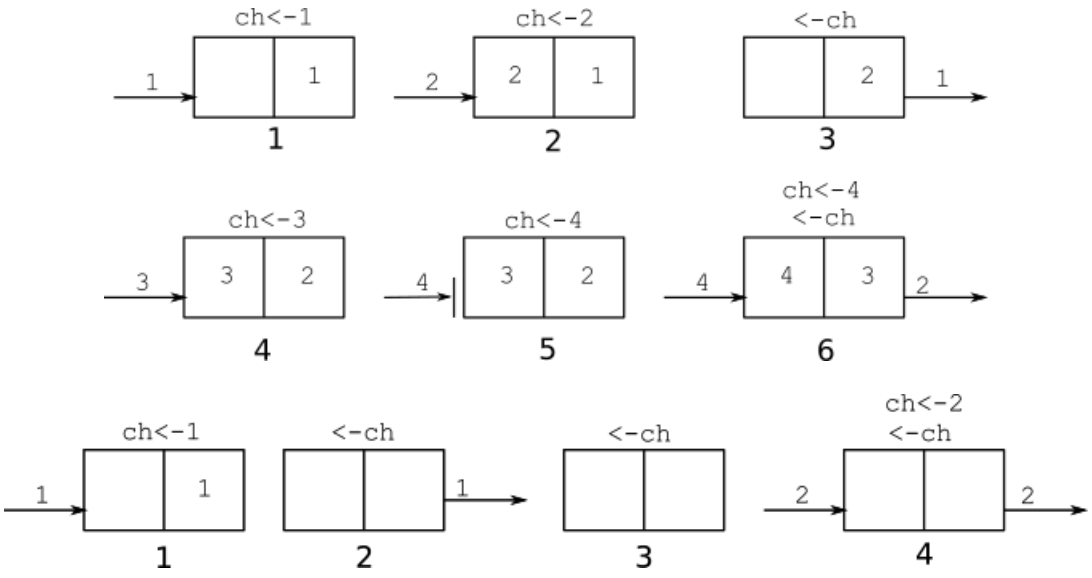
Chapter 1: Concurrency: A High-level Overview

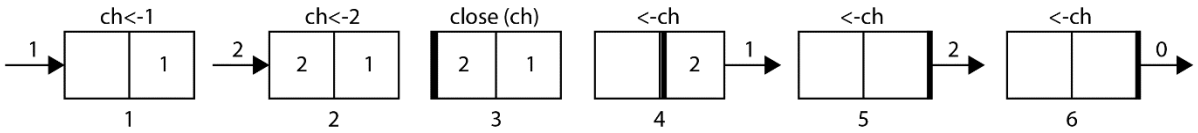


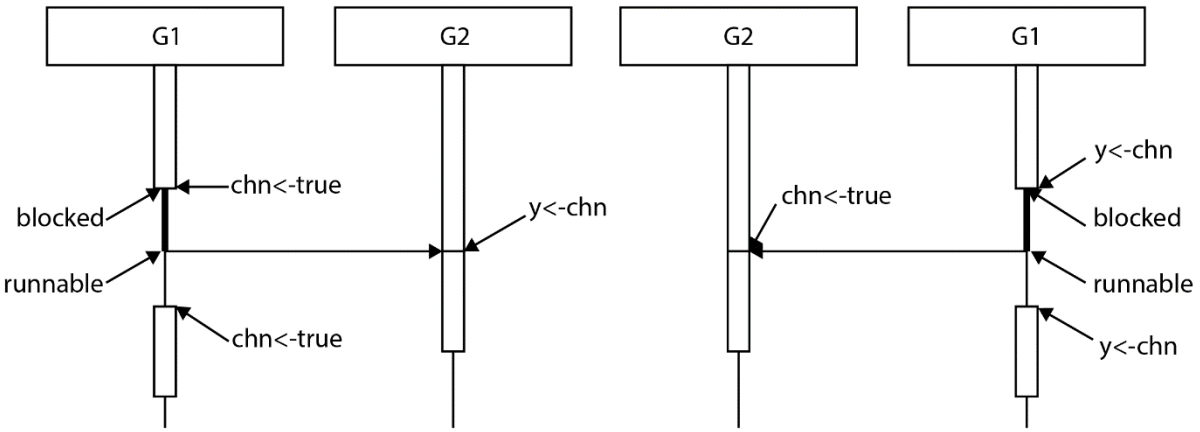
Chapter 2: Go Concurrency Primitives



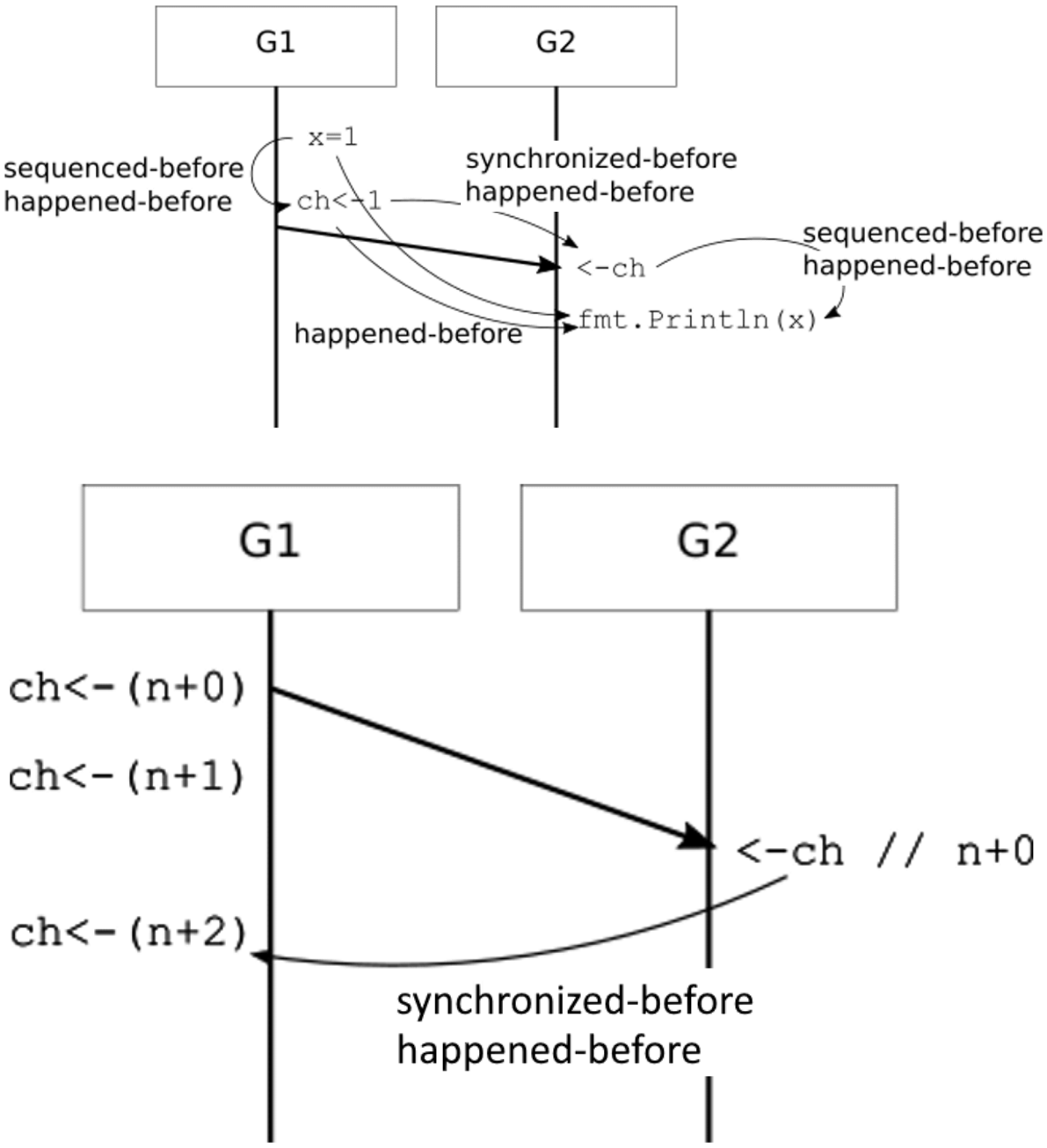




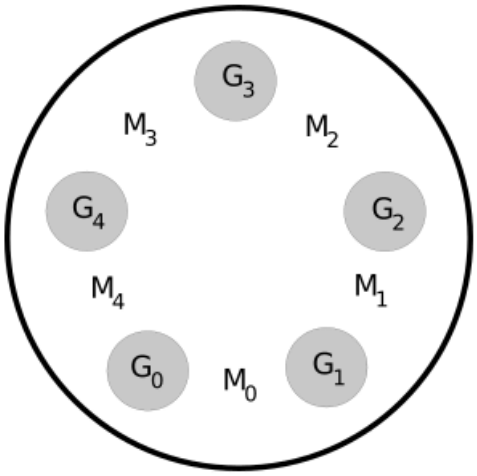

ch<-1
ch<-2
close (ch)
<-ch
<-ch
<-ch
1
2
1
2
2
1
2
1
2
1
2
0
1
2
3
4
5
6

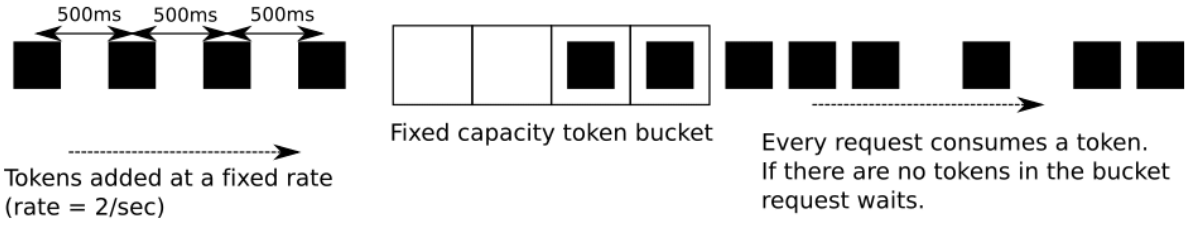

G1
G2
G2
G1
chn<-true
blocked
runnable
chn<-true
y<-chn
chn<-true
y<-chn
blocked
runnable
y<-chn

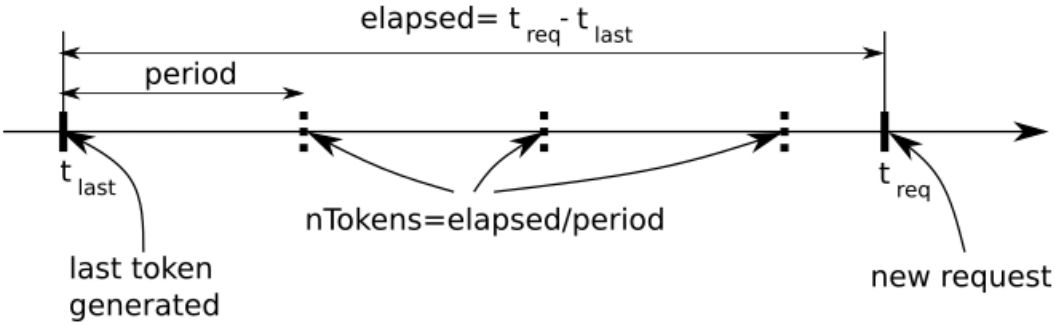
Chapter 3: The Go memory model

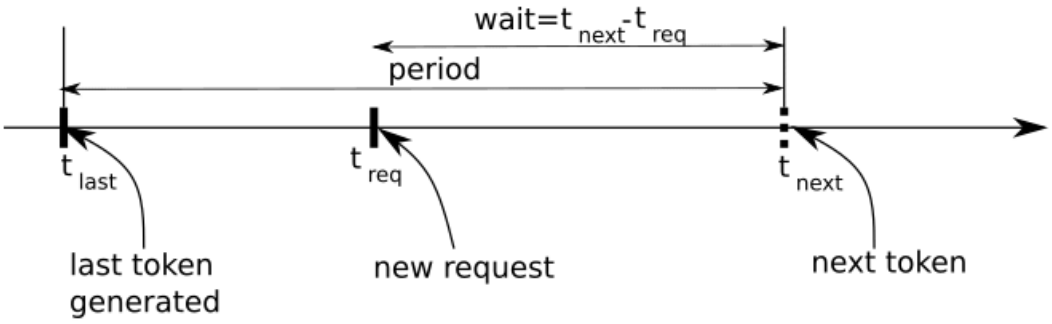


Chapter 4: Some well-known concurrency problems

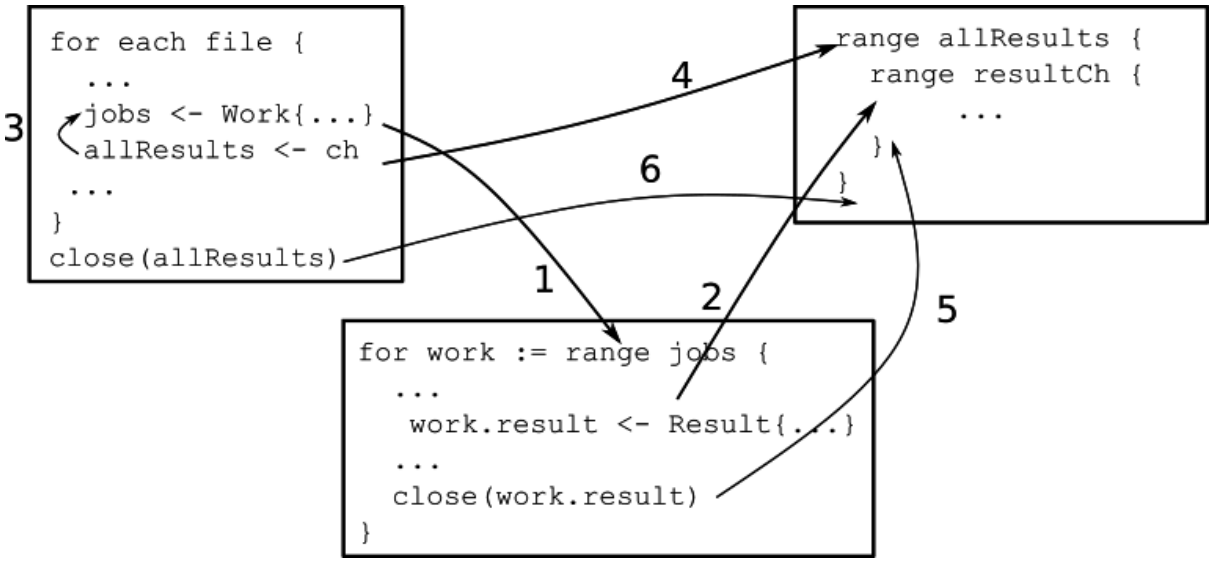


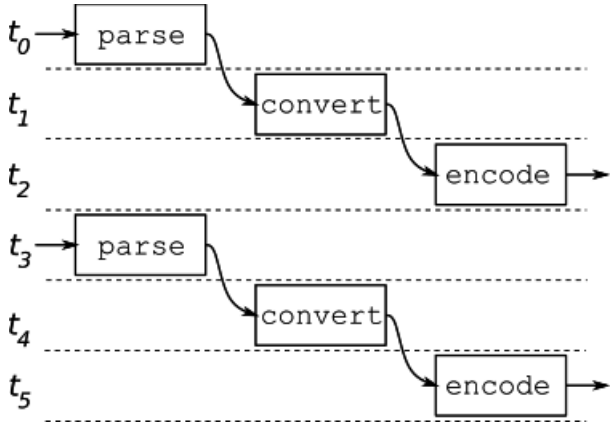


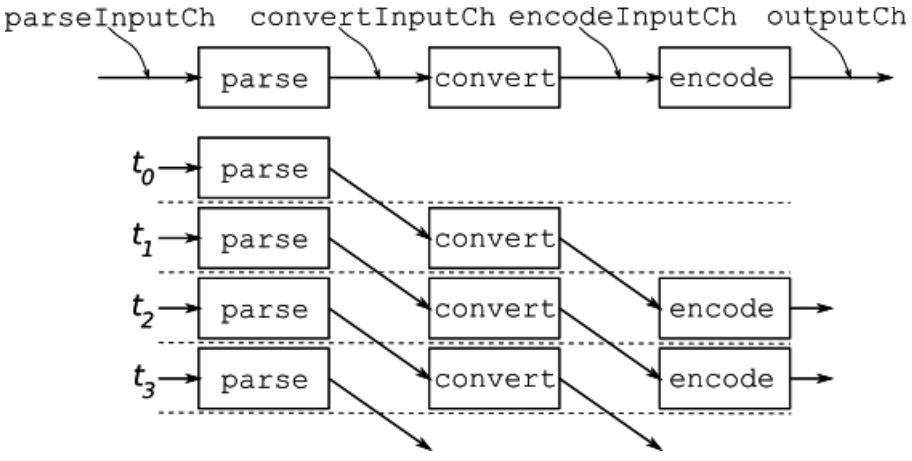


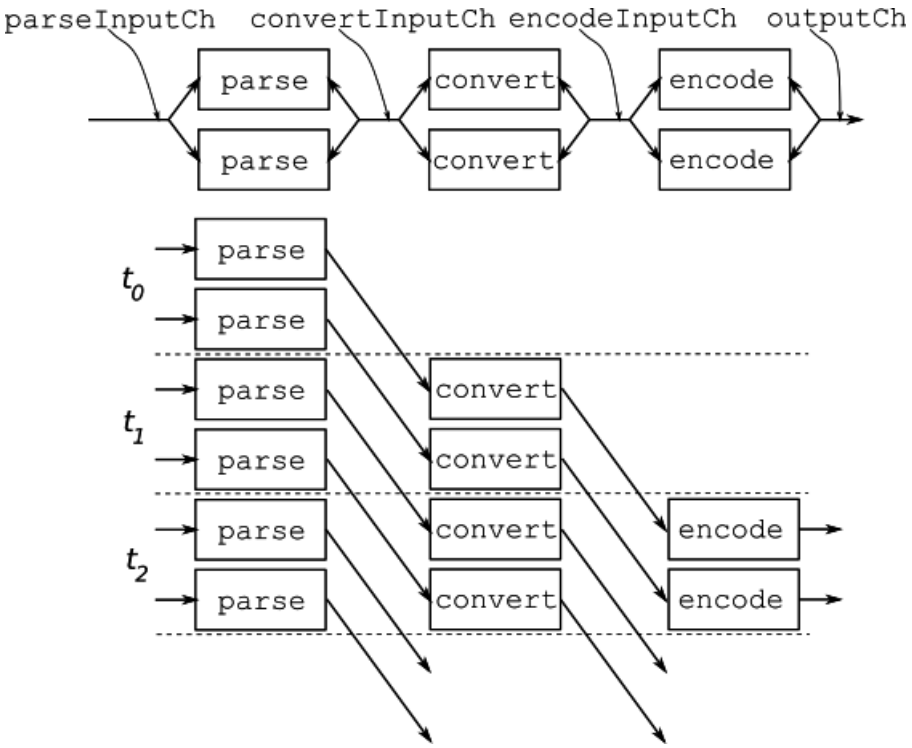


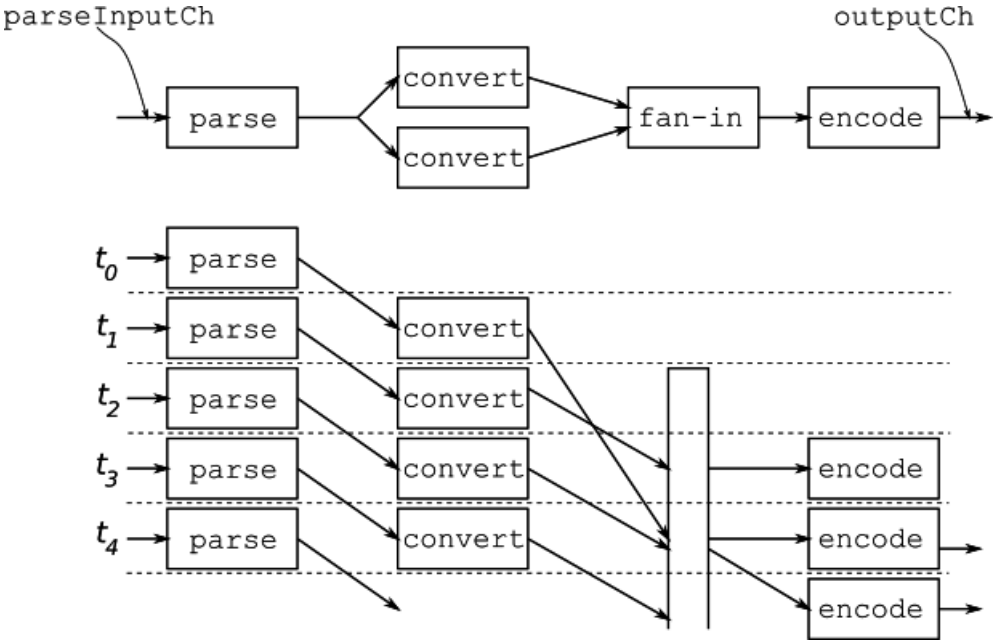
Chapter 5: Worker pools and pipelines

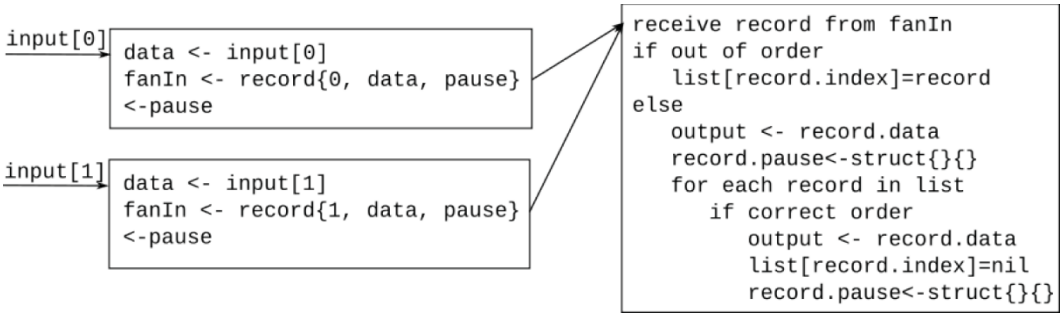


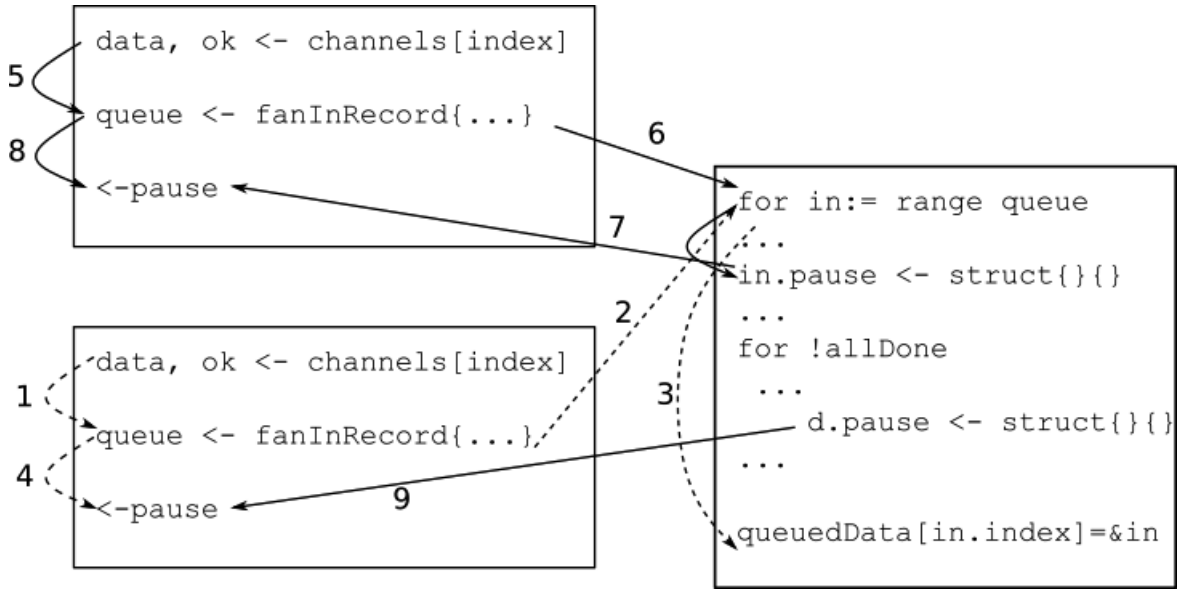





parseInputCh
convertInputCh
encodeInputCh
outputCh
parse
parse
convert
convert
encode
encode
t0
t1
t2
parse
parse
parse
parse
parse
parse
convert
convert
convert
convert
encode
encode


parseInputCh
outputCh
parse
convert
convert
fan-in
encode
t0
t1
t2
t3
t4
parse
parse
parse
parse
parse
convert
convert
convert
convert
encode
encode
encode

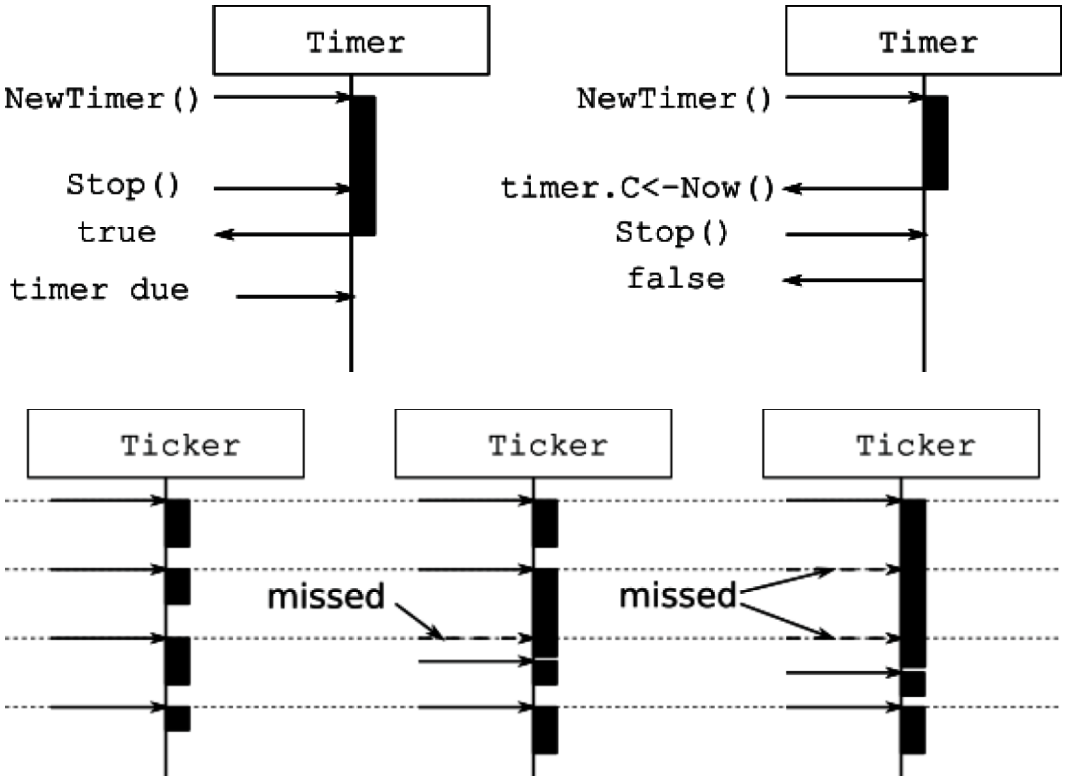

input[0]
data <- input[0]
fanIn <- record{0, data, pause}
<-pause
input[1]
data <- input[1]
fanIn <- record{1, data, pause}
<-pause
receive record from fanIn
if out of order
list[record.index]=record
else
output <- record.data
record.pause<-struct{}{}
for each record in list
if correct order
output <- record.data
list[record.index]=nil
record.pause<-struct{}{}


data, ok <- channels[index]
queue <- fanInRecord{...}
<-pause
data, ok <- channels[index]
queue <- fanInRecord{...}
<-pause
for in:= range queue
...
in.pause <- struct{}{}
...
for !allDone
...
d.pause <- struct{}{}
...
queuedData[in.index]=&in
1
2
3
4
5
6
7
8
9

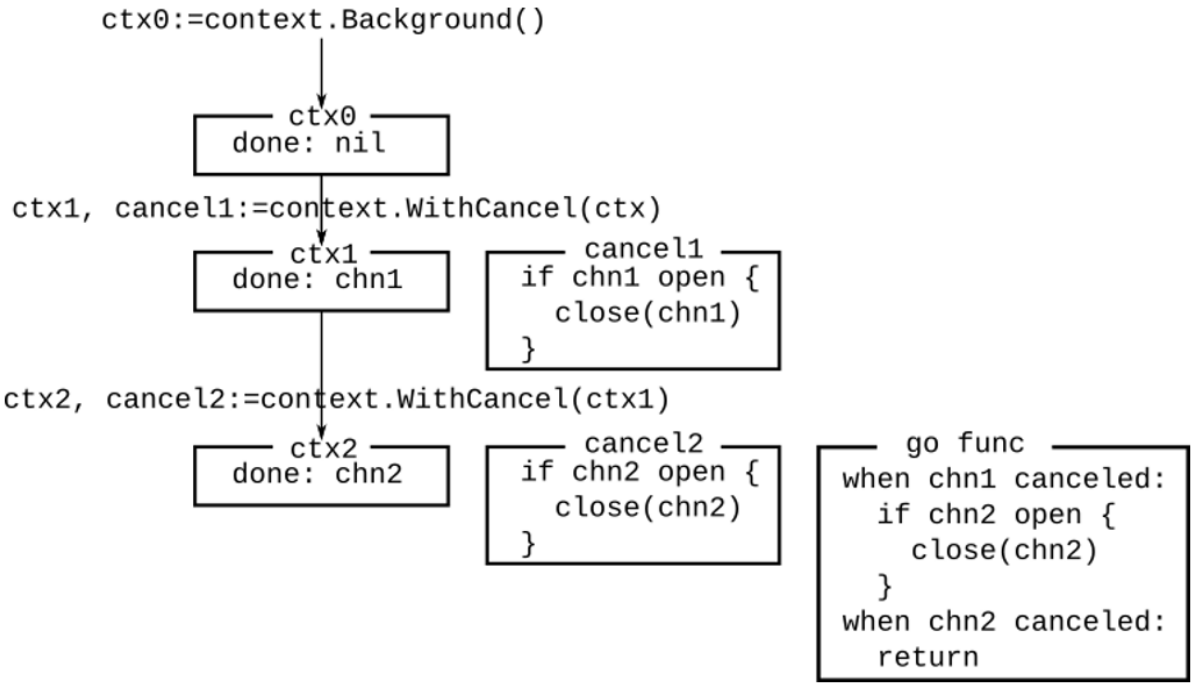
Chapter 6: Error handling

No-images…

Chapter 7: Timers and Tickers



Chapter 8: Handling requests concurrently



Chapter 9: Atomic memory operations

No-images…

Chapter 10: Troubleshooting Concurrency Issues

No-images…