CHAPTER 7

Queues

Figure 7.1

Some queue operations

Operation	Queue after operation
	Front ←
`queue.createQueue()`	
`queue.enqueue(5)`	5
`queue.enqueue(2)`	5 2
`queue.enqueue(7)`	5 2 7
`queueFront = queue.peek()`	5 2 7 (`queueFront` is 5)
`queueFront = queue.dequeue()`	5 2 7 (`queueFront` is 5)
`queueFront = queue.dequeue()`	2 7 (`queueFront` is 2)

Figure 7.2

The results of inserting a string into both a queue and a stack

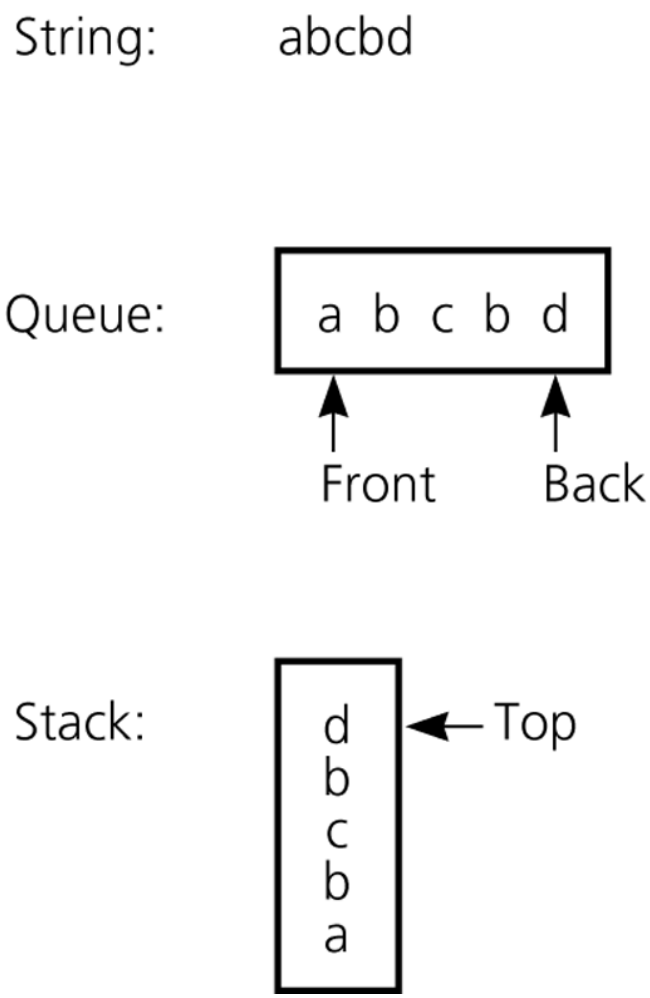

Figure 7.3a

A reference-based implementation of a queue: a) a linear linked list with two external references; b) a circular linear linked list with one external reference

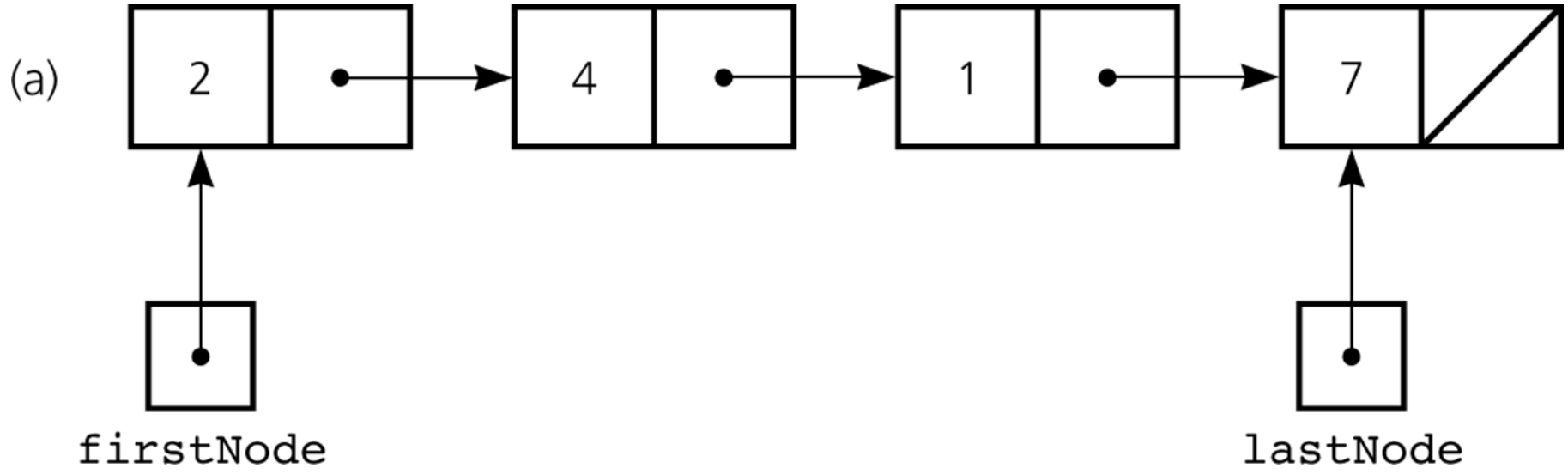

Figure 7.3b

A reference-based implementation of a queue: a) a linear linked list with two external references; b) a circular linear linked list with one external reference

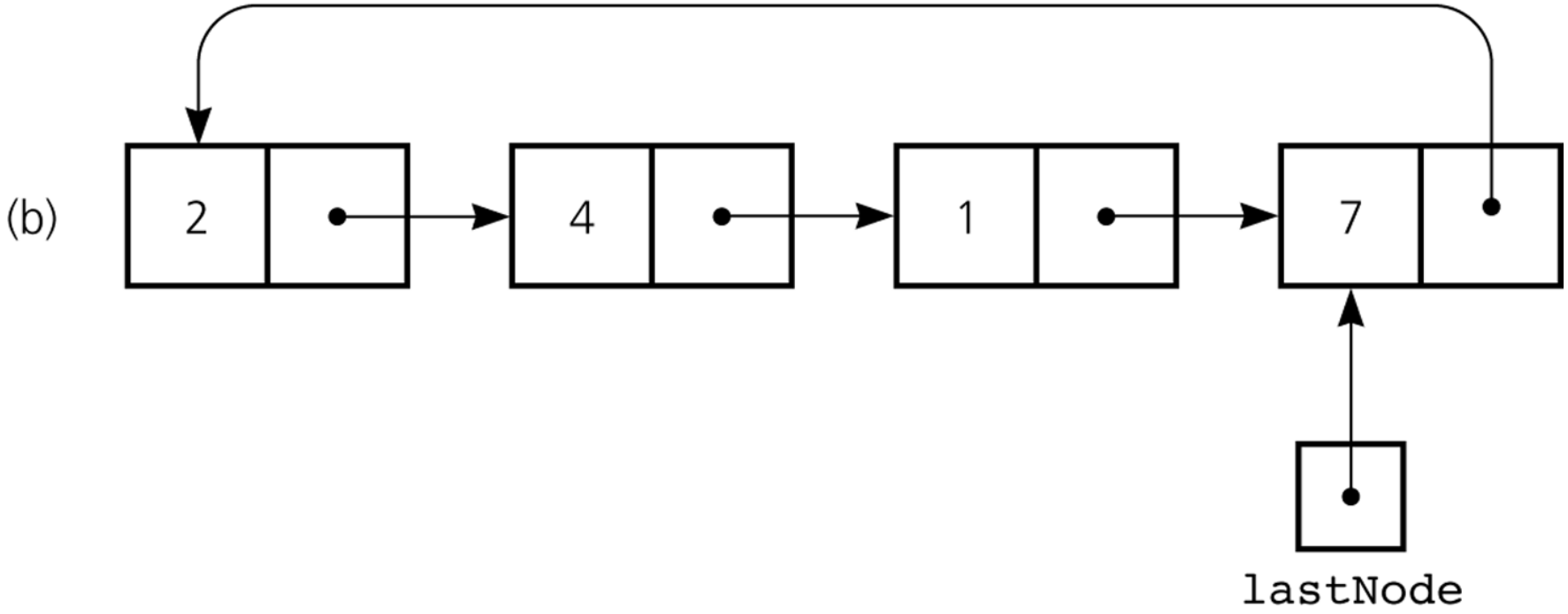

Figure 7.4

Inserting an item into a nonempty queue

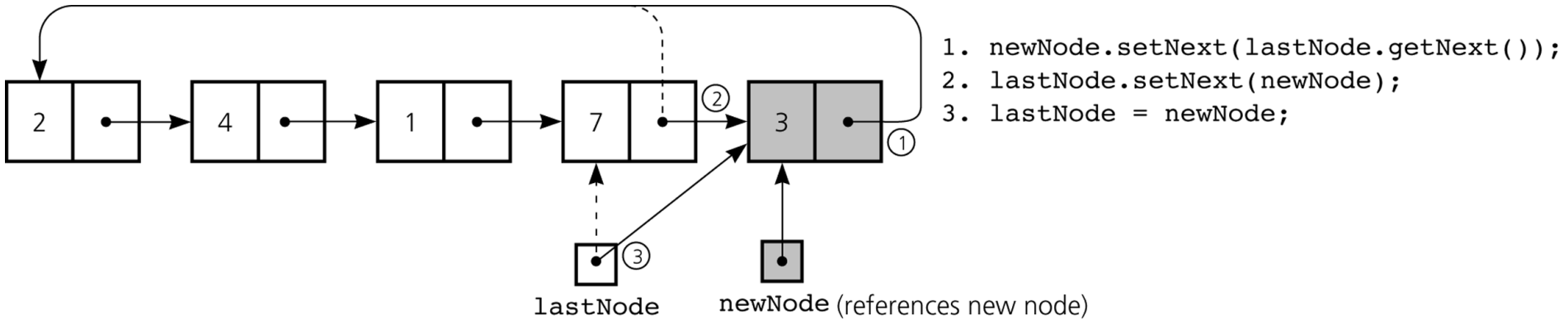

Figure 7.5

Inserting an item into an empty queue: a) before insertion; b) after insertion

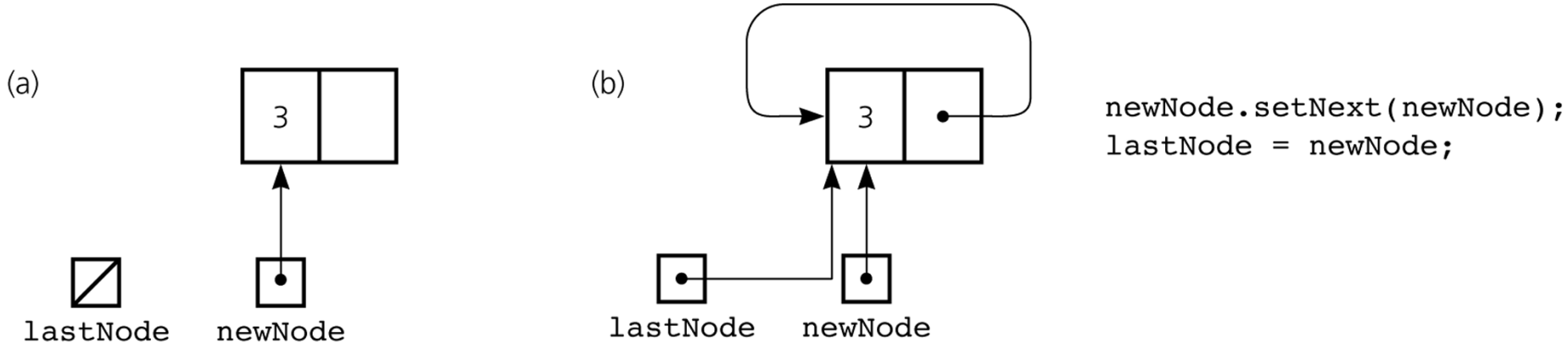

Figure 7.6

Deleting an item from a queue of more than one item

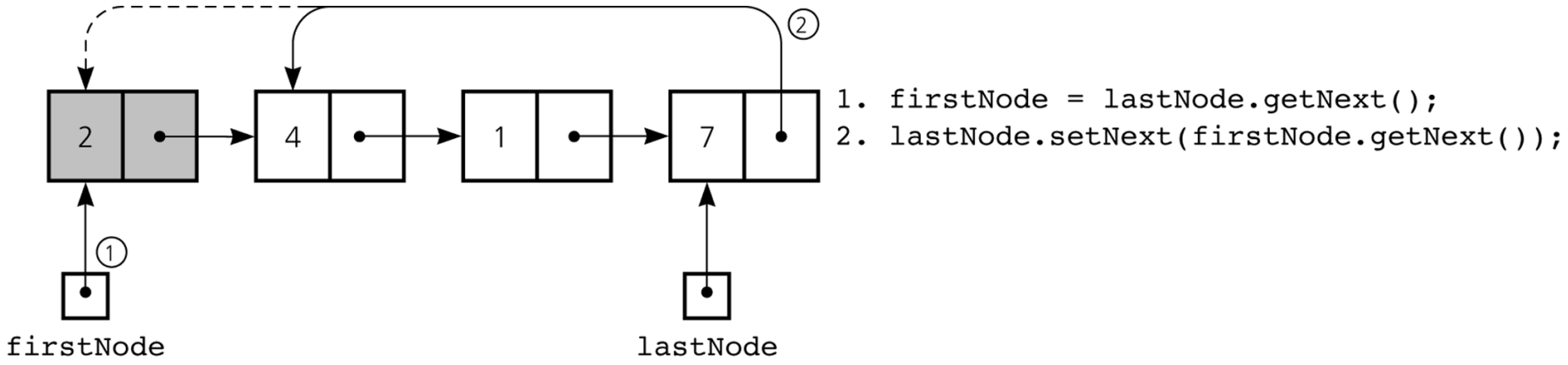

Figure 7.7

a) A naive array-based implementation of a queue; b) rightward drift can cause the queue to appear full

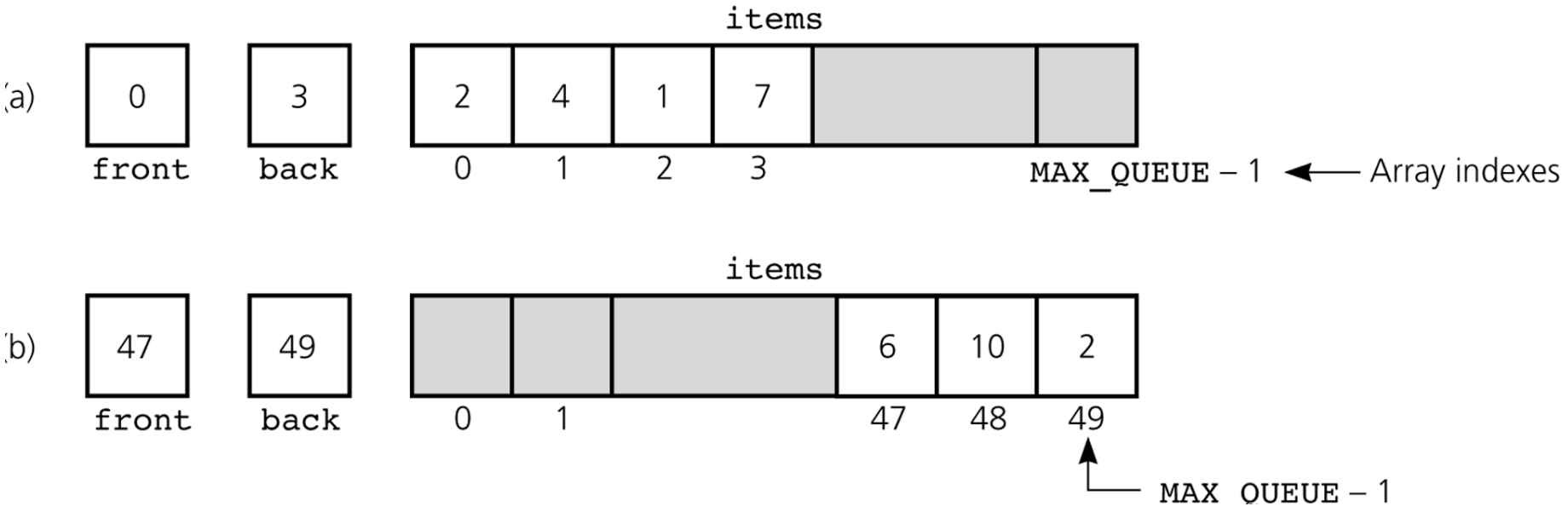

Figure 7.8

A circular implementation of a queue

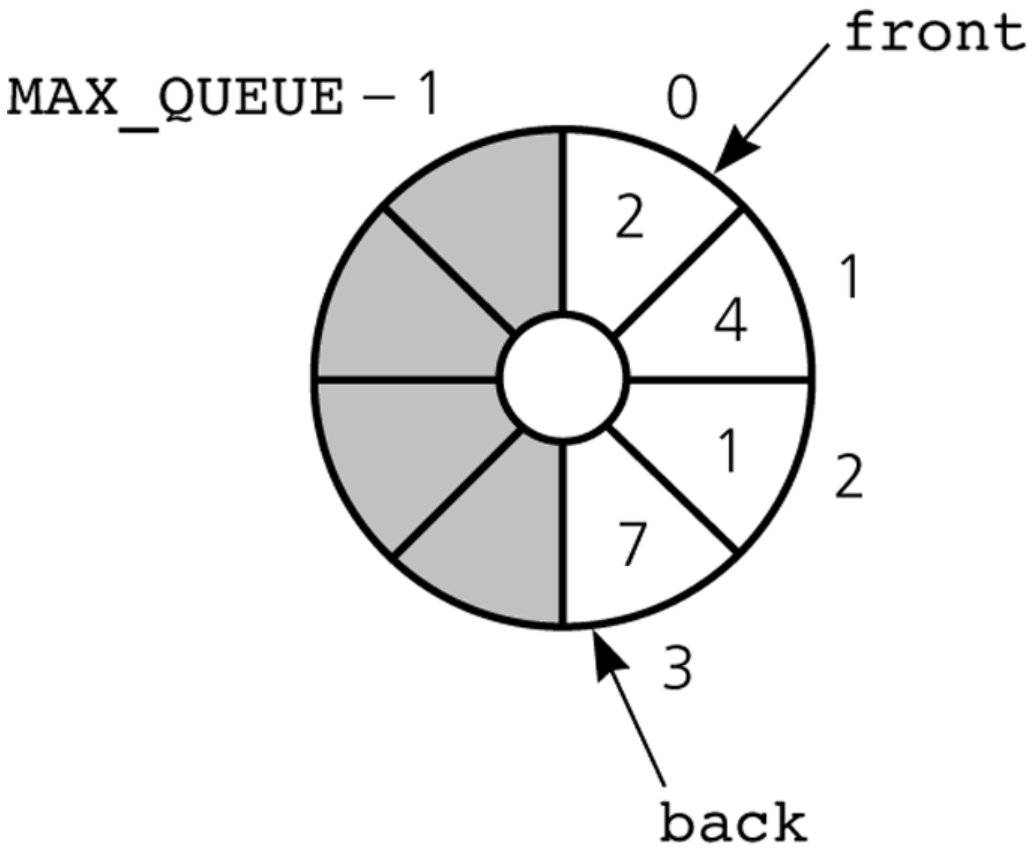

Figure 7.9

The effect of some operations of the queue in Figure 7-8

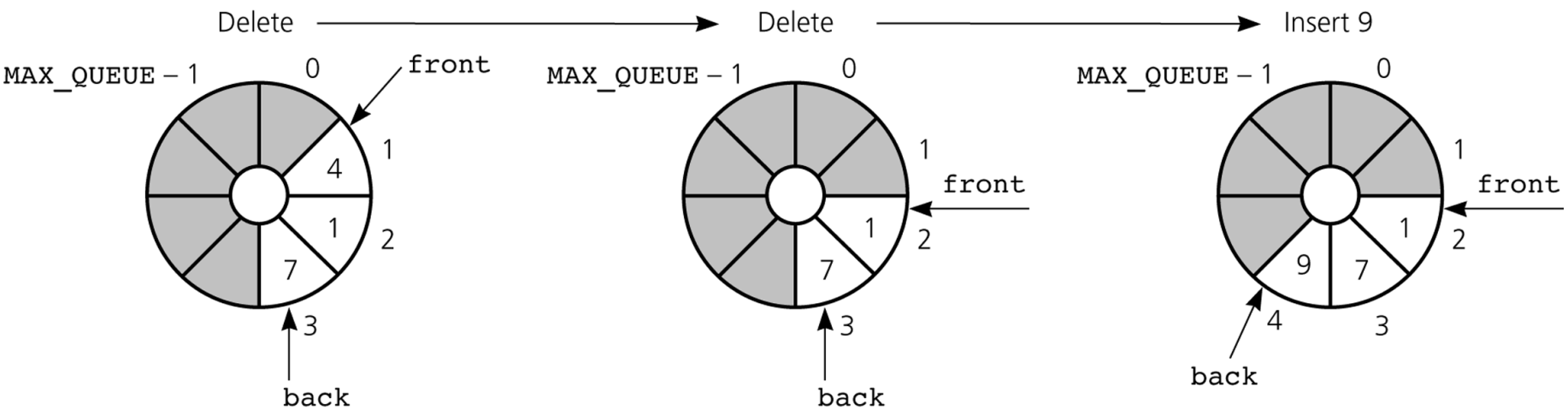

Figure 7.10a

a) ***front*** passes ***back*** when the queue becomes empty

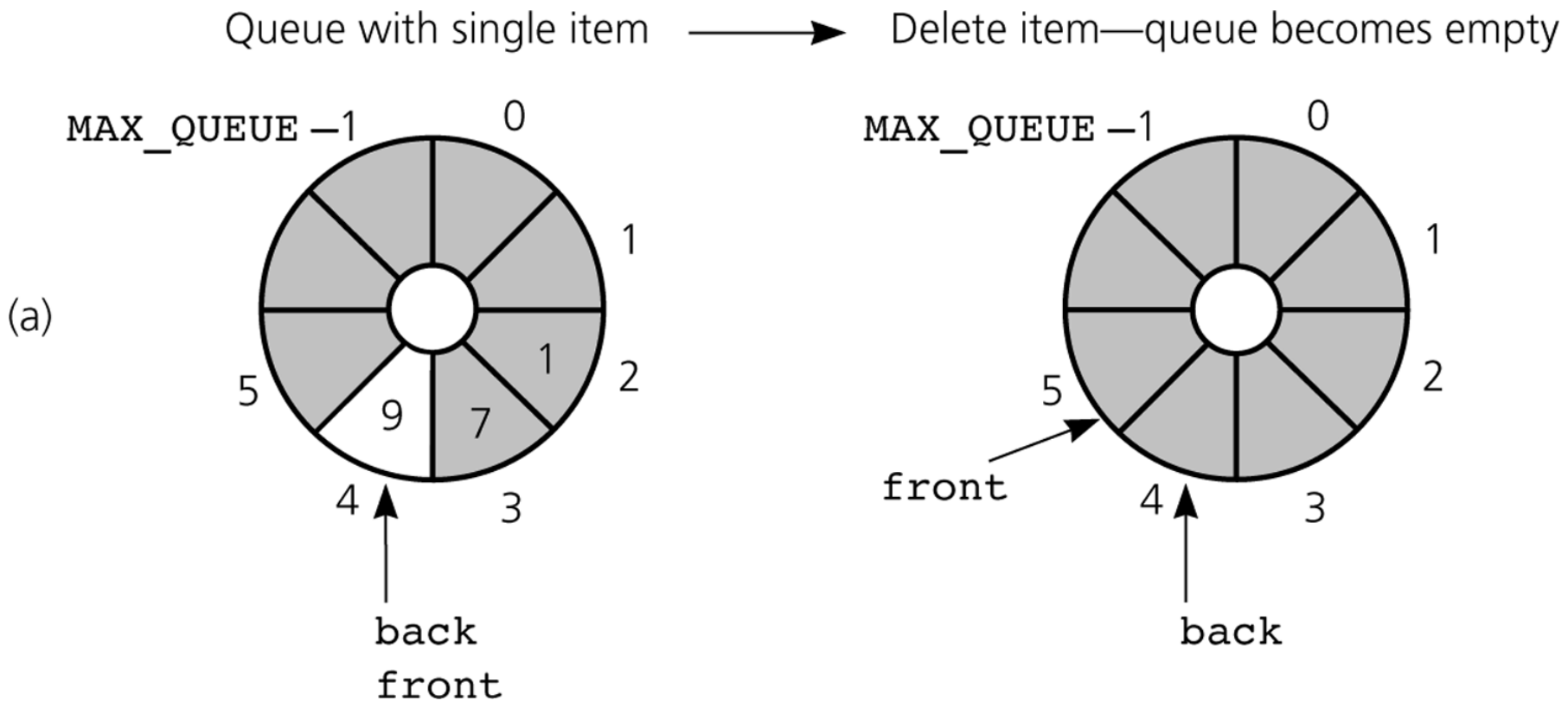

Figure 7.10b

b) ***back*** catches up to ***front*** when the queue becomes full

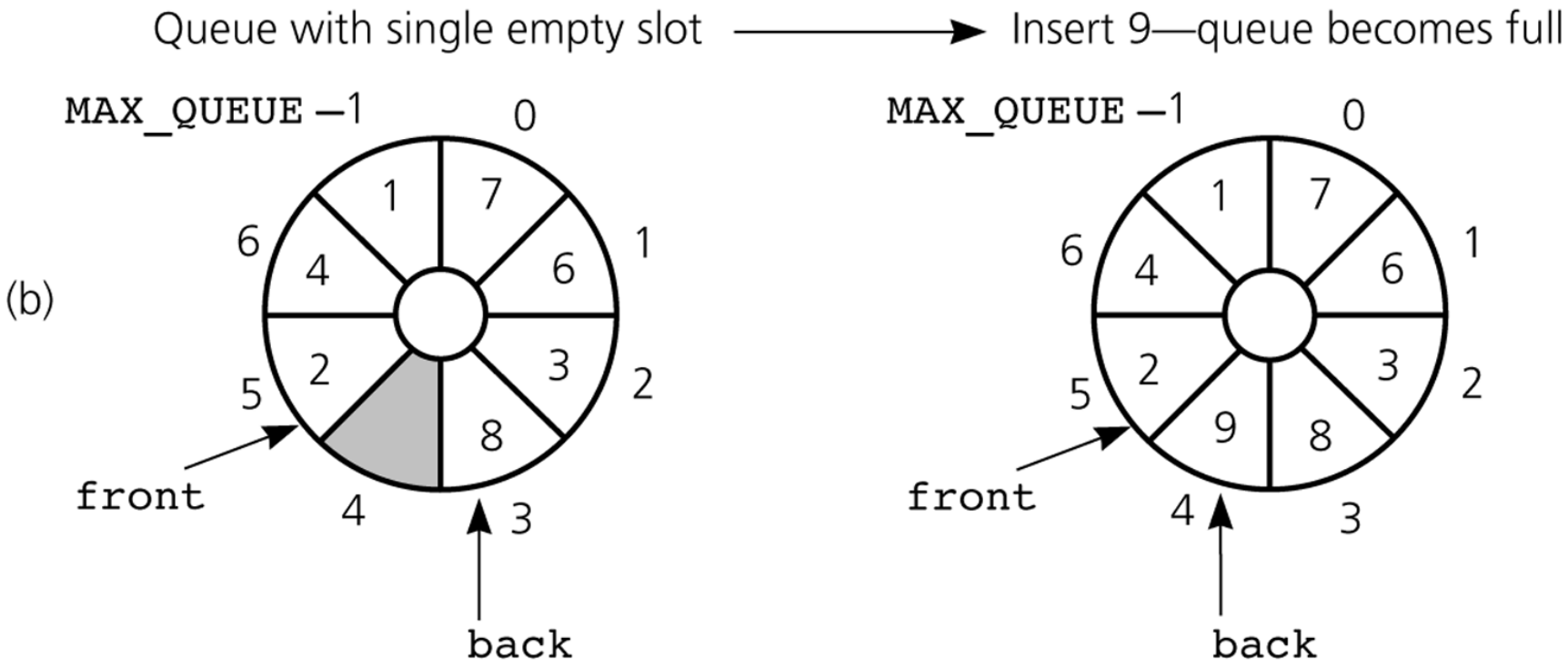

Figure 7.11

A more efficient circular implementation: a) a full queue; b) an empty queue

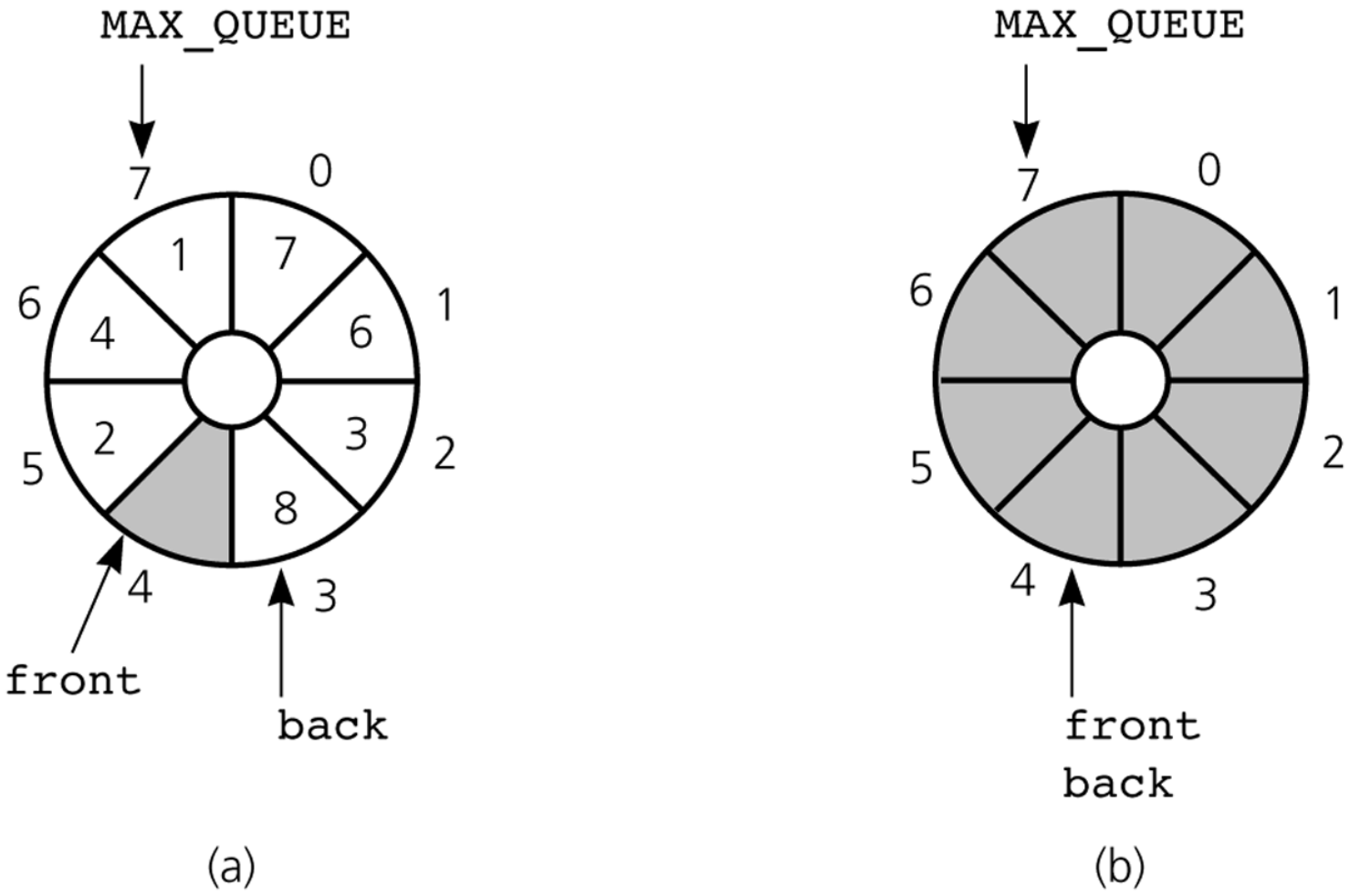

Figure 7.12

An implementation that uses the ADT list

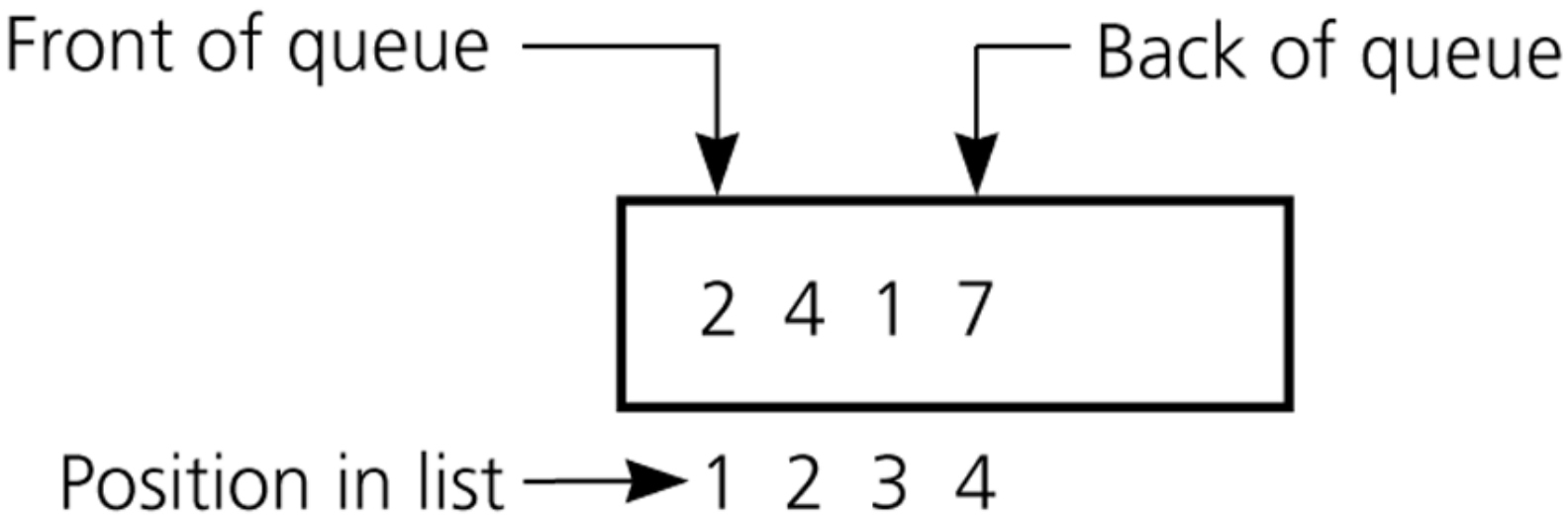

Figure 7.13a and 7.13b

A blank line at at time a) 0; b) 12

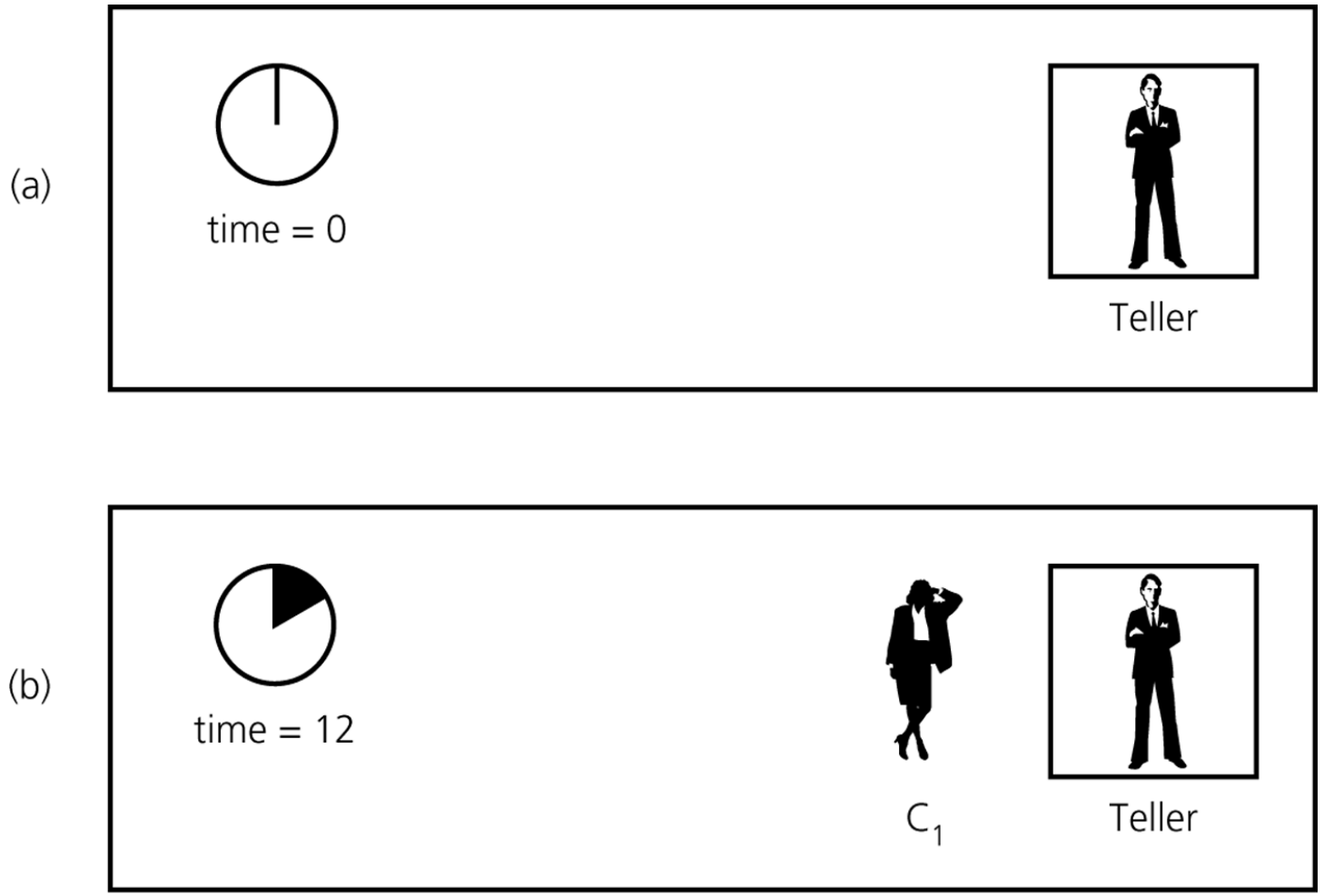

Figure 7.13c and 7.13d

A blank line at at time c) 20; d) 38

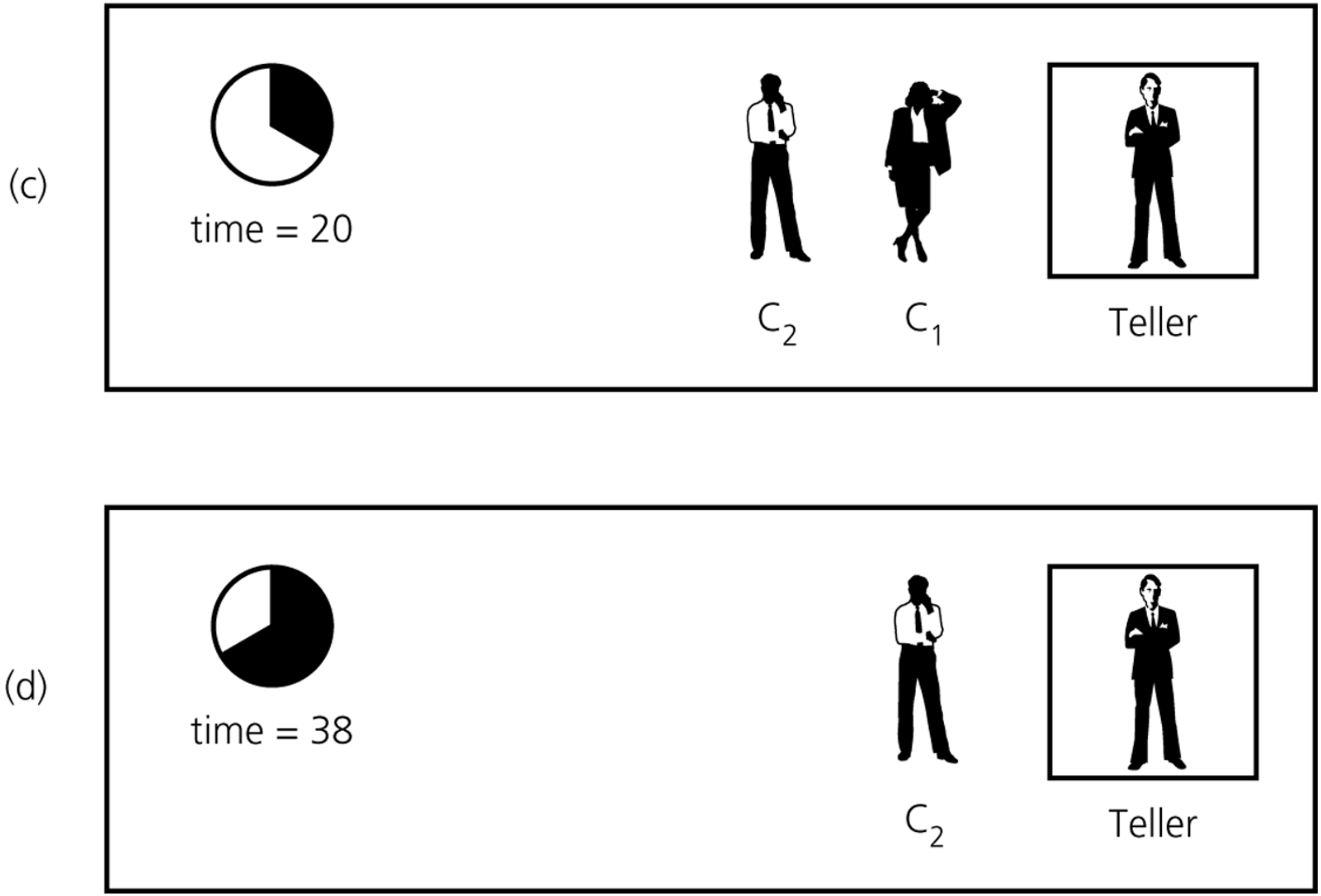

Figure 7.14

A typical instance of the event list

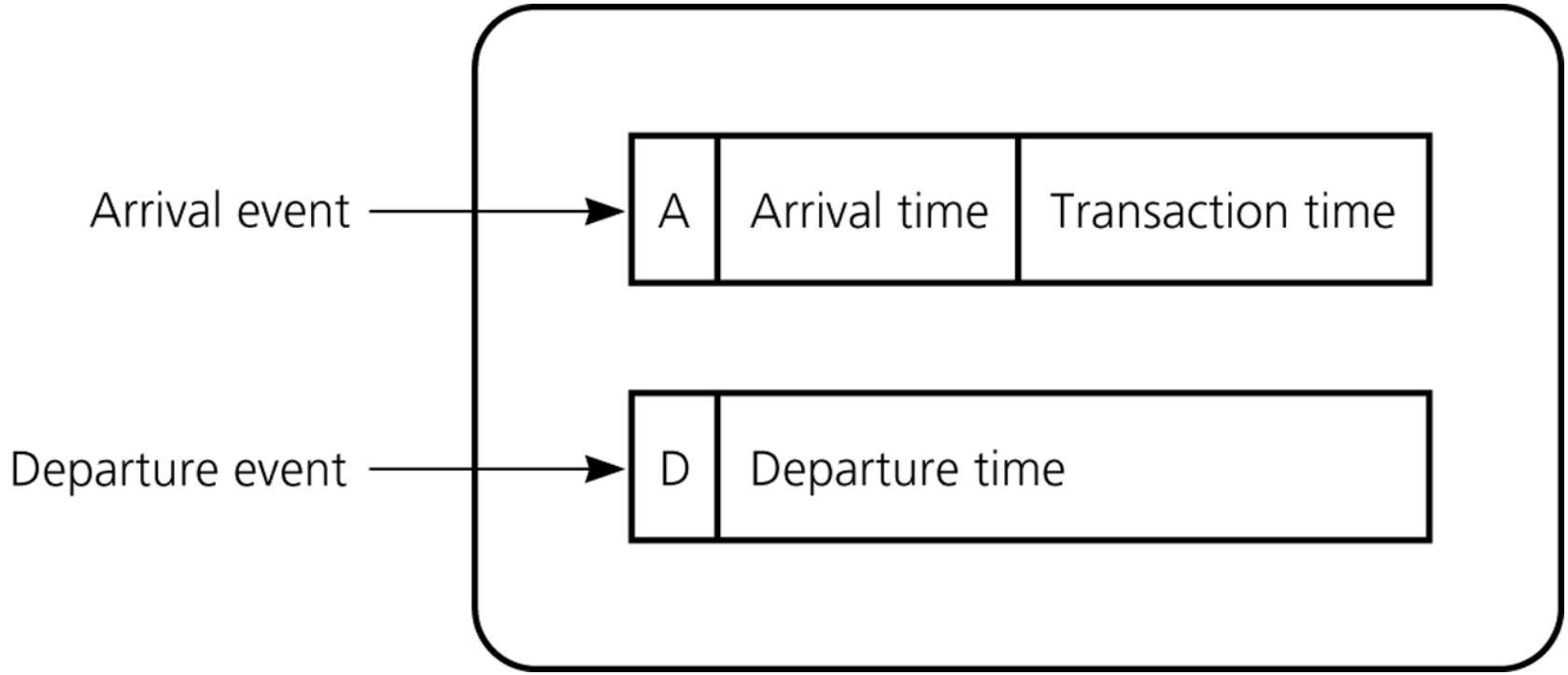

Figure 7.15

A Partial trace of the bank simulation algorithm for the data

20 5 23 2
22 4 30 3

Time	Action	`bankQueue` (front to back)	`eventList` (beginning to end)
0	Read file, place event in `eventList`	*(empty)*	A 20 5
20	Update `eventList` and `bankQueue`: Customer 1 enters bank	20 5	*(empty)*
	Customer 1 begins transaction, create departure event	20 5	D 25
	Read file, place event in `eventList`	20 5	A 22 4 D 25
22	Update `eventList`and `bankQueue`: Customer 2 enters bank	20 5 22 4	D 25
	Read file, place event in `eventList`	20 5 22 4	A 23 2 D 25
23	Update `eventList` and `bankQueue`: Customer 3 enters bank	20 5 22 4 23 2	D 25
	Read file, place event in `eventList`	20 5 22 4 23 2	D 25 A 30 3
25	Update `eventList` and `bankQueue`: Customer 1 departs	22 4 23 2	A 30 3
	Customer 2 begins transaction, create departure event	22 4 23 2	D 29 A 30 3

Self-Test Exercise 6 asks you to complete this trace.