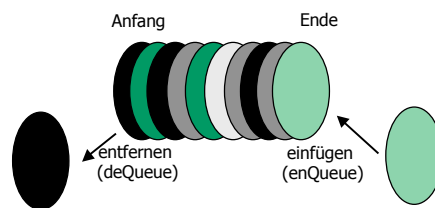




## Queue



- deutsch: Warteschlange, Puffer
- abstrakte Datenstruktur, Container-Datentyp
- Elemente können eingefügt und wieder entfernt werden
- direkter Zugriff nur auf das zuerst eingefügte (*least recently added*) Element (daher: FIFO = *first in first out*)



## Operationen



- `enqueue` Füge ein neues Objekt in die Warteschlange ein.
- `dequeue` Lösche und gebe aus das Objekt, das zuerst eingefügt wurde.
- `isEmpty` Ist die Warteschlange leer?

```
q = Queue()
q.enqueue("This")
q.enqueue("is")
q.enqueue("a")
print q.remove()
q.enqueue("test.")
while not q.isEmpty() :
    print q.dequeue()
```

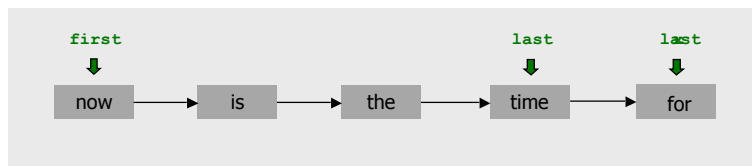
A simple queue client



## Implementierung als verkettete Liste



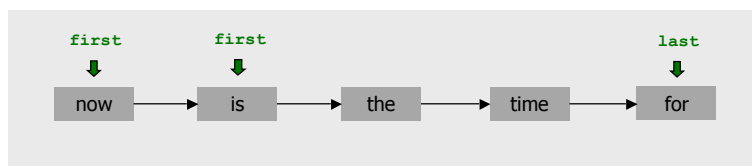
### enqueue



```
x = List();  
x.item = "for";  
last.next = x;  
last = x;
```



### dequeue



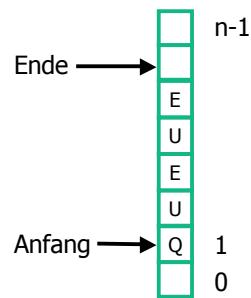
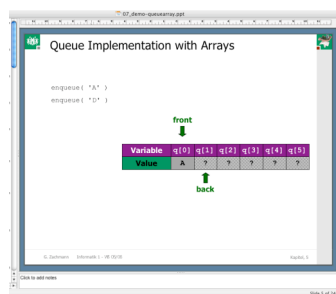
```
val now  
val = first.item;  
first = first.next;  
return val;
```



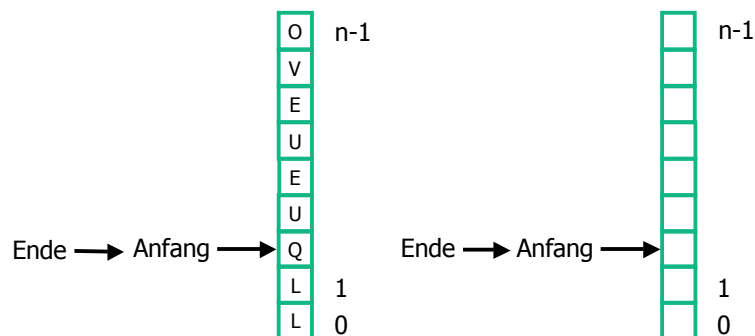
## Implementierung mit Array



- begrenzter Speicherplatz: Array mit  $n$  Elementen
- zwei Zeiger: auf Anfang und Ende
- zyklischer Zugriff auf Elemente
  - erreicht ein Zeiger beim Inkrementieren den Wert  $n$ , wird er auf 0 zurückgesetzt



- **Anfang == Ende** → entweder ist Queue voll, oder leer



## Implementierung in Python mit Liste

```
class Queue :
    def __init__( self ) :
        self.first = self.last = None
    class List:
        # nested class
        item = None      # satellite data
        next = None     # "pointer"
    def isEmpty(self): # Methode in Queue
        return first == None
    def enqueue(self, item):
        x = List()
        x.item = item
        x.next = None
        if self.isEmpty():
            self.first = x
        else:
            self.last.next = x
            self.last = x
    def dequeue(self):
        val = self.first.item
        self.first = self.first.next # unlink first item
        return val
```

- Bemerkung: Die Queue muß nicht homogen sein! (im Gegensatz zu den einfachen, analogen Implementierungen in Java/C++)
  - Da schon Liste (und Array) nicht homogen sein müssen
- Frage: stimmt **dequeue()** auch für den Fall, daß Liste genau 1 Element enthält ?
  
- Generelle Regel für Datenstrukturen-Entwurf: checke die "Ausnahmen"!! (Randfälle, *boundary cases*)
  - Stimmt die Funktion für den Fall, daß 0 oder 1 Element vorhanden ist?
  - Was passiert, wenn Cursor am Ende oder auf None steht?
  - ...



## Anwendungen (nur Beispiele)



- In Programmen: alle Arten von Daten-Puffern
  - Dispensing requests on a shared resource (printer, processor)
  - Asynchronous data transfer (file IO, pipes, sockets)
    - man kann mehrere Elemente auf einen Schlag hinzufügen (z.B. Teil einer Datei von Festplatte)
    - danach kann man einzeln auf die Elemente zugreifen
  - Data buffers (MP3 player, portable CD player, Tastatur)
- Simulation
  - von Fertigungsprozessen: Objekte auf Förderbändern verhalten sich wie in einer Warteschlange
  - Wartezeiten bei McDonalds oder Call-Center, oder Verkehr vor Tunnel, oder ...



## Exkurs: Poisson-Prozeß



- In der Natur viele Prozesse mit folgenden Eigenschaften:
  - Zeitpunkt des Ereignisses zufällig und unabhängig voneinander, z.B.:
    - Radioaktives Material
    - Kunden kommen an Warteschlange an
    - HTML-Request an einem WWW-Server
    - Unfälle an einer Kreuzung
- Zugrunde liegende Annahme:
  - Seien  $T_i$  der Zeitpunkt des  $i$ -ten Ereignisses, seien  $X_i = T_i - T_{i-1}$  die "Zwischenzeiten" (*inter-arrival times*)
  - Der Prozeß verhält sich nach dem  $k$ -ten Ereignis (für beliebiges  $k$ ) genau wie am Anfang
  - Die  $X_i$  müssen alle identisch und unabhängig voneinander verteilt sein (*i.i.d = independent and identically distributed*)



## Verteilung der Inter-Arrival Times



- Fundamentale Annahme: der Prozeß ist "gedächtnislos", d.h.,
  - Falls Ereignis noch nicht nach Zeitraum  $s$  eingetreten, dann ist Wahrscheinlichkeit dafür, daß Ereignis bis Zeit  $s+t$  eintritt genauso hoch wie W.keit, daß Ereignis bis Zeit  $t$  eintritt

$$\forall s, t > 0 : P[X > t + s \mid X > s] = P[X > t]$$

- Man kann zeigen (o.Bew.):

$$P[X > t + s] = P[X > t] \cdot P[X > s]$$

- Die einzige Funktion, die diesen Prozeß beschreiben kann, ist

$$P[X > t] = e^{-rt}, \quad t \geq 0$$

wobei  $r$  eine Skalierung ist.

- Die Zeit  $X$  zwischen zwei Ereignissen gehorcht dieser Dichtefunktion

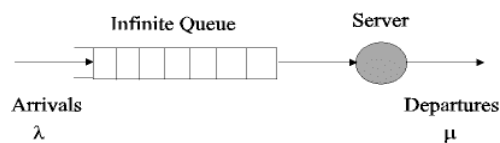
$$f(t) = P[X = t] = re^{-rt}, \quad t \geq 0$$



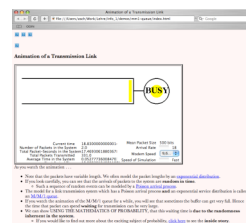
## M/M/1 Queuing Model



- Customers arrive at rate of  $\lambda$  per minute.
- Customers are serviced at rate of  $\mu$  per minute.
- Use *Poisson process* to model arrivals and departures.
- Wichtigstes Warteschlangenmodell



- How long does a customer wait in queue?





## M/M/1 Queue: Implementation



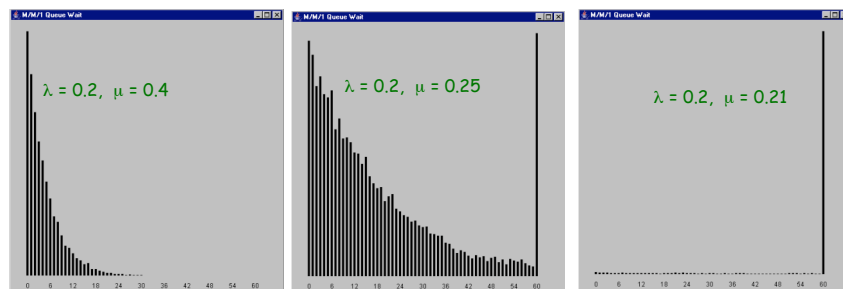
```
while true :  
    if nextArrival < nextDeparture : arrival  
        customer = Customer()  
        customer.arrive( nextArrival )  
        q.enqueue( customer ) ↙ amount of time until next arrival  
        nextArrival += exponential( lambda )  
    else :  
        if not q.isEmpty() : departure  
            hist.addDataPoint( math.min(60, q.length()) )  
            customer = q.dequeue()  
            customer.depart( nextDeparture )  
            nextDeparture += exponential( mu )  
            ↙ amount of time until next departure
```



## M/M/1 Queue Analysis



Wenn Abarbeitungsrate gegen Ankunftsrate geht, warten immer mehr Kunden  $\geq 60$  Min., d.h., die Queue explodiert immer häufiger



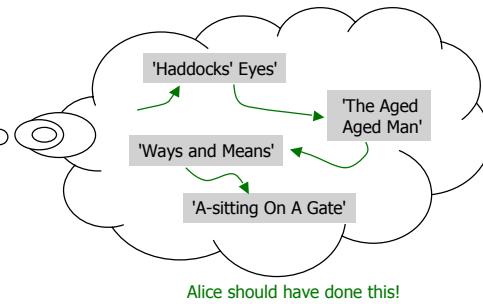
("As service rate approaches arrival rate, service goes to  $h^{***}$ .")



## Folgerung



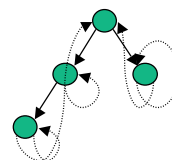
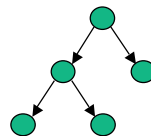
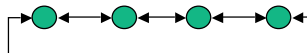
- *Sequential allocation*: unterstützt Indizierung, feste Größe.
- *Linked allocation*: variable Größe, unterstützt sequentiellen Zugriff.
- Verkettete Strukturen sind eine zentrale Datenstruktur und -abstraktion.



## Vergleich von Varianten verketteter Strukturen



- Linked list.
- Circular linked list.
- Doubly linked list.
- Binary tree.
- Patricia tries.







- Stacks und Warteschlangen sind fundamentale ADTs.
  - Implementation als Verkettete Liste.
  - Arrayimplementation.
  - Verschiedene Performanceeigenschaften.
  
- Viele Anwendungen.
  - Taschenrechner.
  - Drucker und PostScript language.
  - Arithmetische Ausdrücke.
  - Funktionimplementation im Compiler.
  - Web browsing.
  - . . .