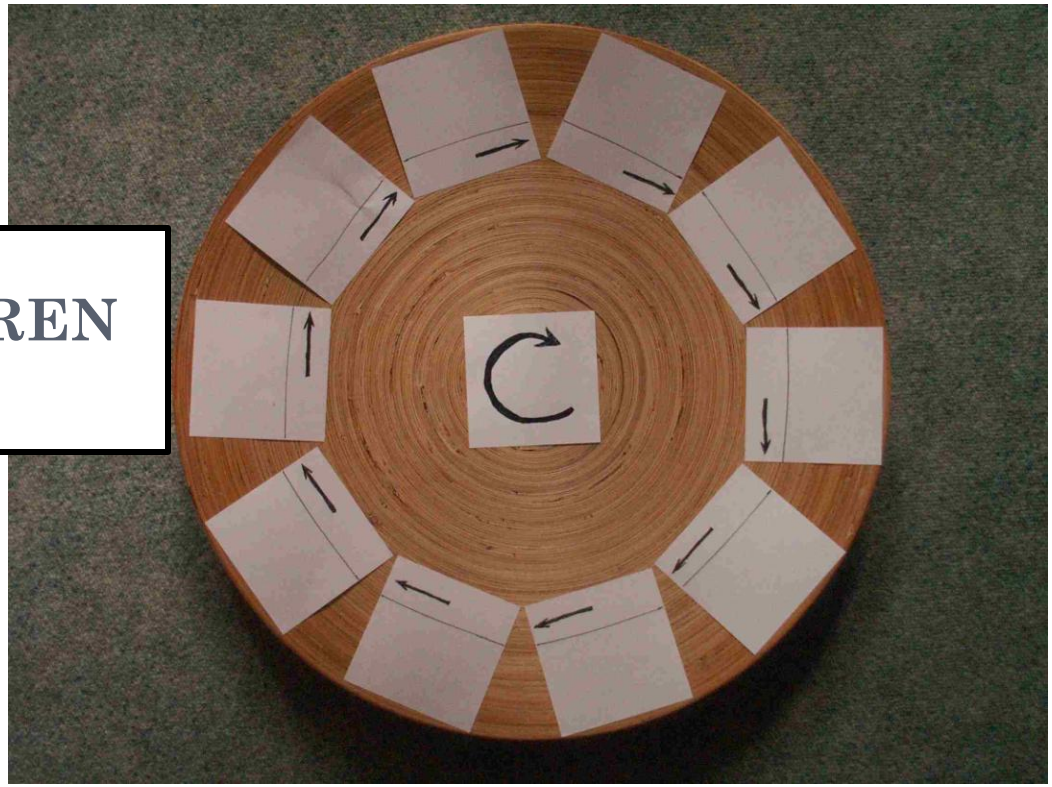
Datastructuren
Lijsten



Dr. D.P. Huijsmans

12 sept 2012

Universiteit Leiden, LIACS

Een analogie voor abstracte datatypen (ADT)

- ADT:
- architectuur + functionaliteit
  - Woon/werk elementen, adressering
  - Mogelijkheden voor bewoning/opslag, verkeer
  - Nieuwbouw + aanmaak voorzieningen
  - Sloop + vrijgeven perceel
- Woon/werkverkeer:
- architectuur + gebruiksmogelijkheden
  - Woonblok(ken), straatnamen,adressen
  - Vrijstaande huizen, rijtjeshuizen, flats
  - Verkeer, auto (pointer), bussen (buffers)
  - Voorzieningen: nieuwbouw, verhuizing, uit/thuis, postbezorging, verbouw, baan plus reizen, sloop

EENVOUDIGE WOON,WERKARCHITECTUREN

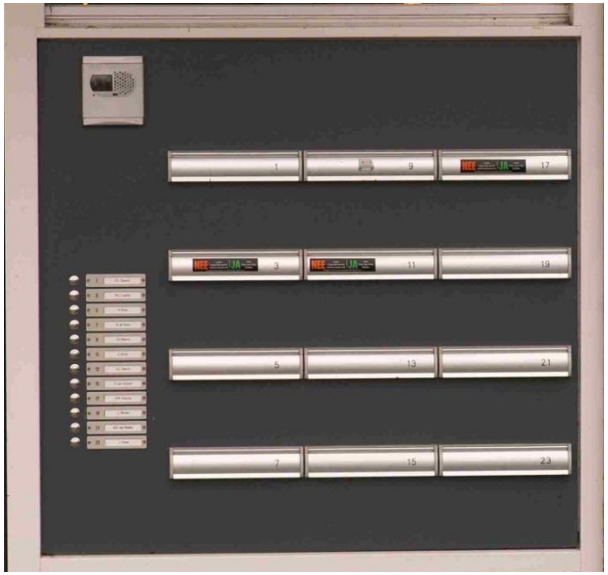
Byte array

Losse variabelen

2-byte array (upper & lower byte per element)

2D en pointer architectuur analogiën

2D data array



1D array van data elementen met pointers

4x3 array met indexen en waardes

Eisen aan ADT

- Complete beschrijving van zowel:
- Architectuur:
  - (alle benodigde data geheugen elementen
  - hulptellers en pointer voorzieningen)
- Functionaliteit (publiek gebruik)
  - (hoe van deze architectuur gebruik gemaakt kan worden: operaties)
- Creatie/Annihilatie:
  - (reserveren van geheugen structuur en vrijgeven ervan na afloop)
- Randvoorwaarden (publiek gebruik):
  - precondities (geldige invoer en huidige situatie voldoet aan?)
  - postcondities (operatie geslaagd? -> resultaten voldoen aan)

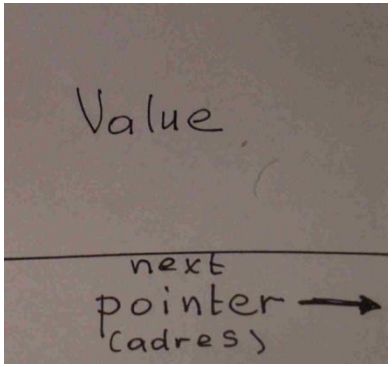
Verschillende soorten lijsten(rijen/queue/array)

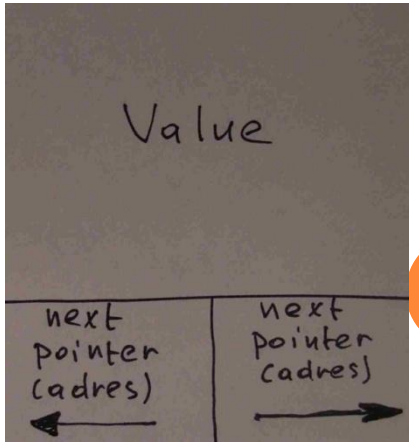
- 1 dimensionaal:
  - Lijst: random access
  - Geordende lijst (gesorteerd): priority queue
  - Circulaire lijst: buffer, FIFO queue
- 2 dimensionaal:
  - Matrix
- N dimensionaal:
  - Serialisatie
  - (Array Mapping Function)

Lijst/Rij/Queue implementaties

- **Array** implementatie (index(en))
  Circulair array (modulo size)

- **verbonden lijst** implementatie
- (single linked list)
- Extra next pointer veld
- Circulaire variant
- **Dubbel verbonden lijst**
- implementatie
- (double linked list)
- 2 extra pointer velden
  - *Links/rechts, vorige/volgende*
  - *Circulaire variant*





ADT_1D_ARRAY SPECIFICATIE

- **Elementen**: alle elementen zijn van hetzelfde opgegeven (eventueel samengestelde) datatype
- **Structuur**: lineaire index waarvan de waardes 1 op 1 corresponderen met de afzonderlijke elementen van elke 1D array waarde.
- **Type** Array: array Indextype of ComponentType
- (b.v. Maand: array[1..12] of String; met Maand[3]:=”Maart”;)
- **Domein**: Index range met bijbehorende elementwaarden
- (b.v. [1.12] met [“Januari”…”December”])
- **Operaties**:
- Als A en B gelijksoortige array’s, c een variabele van type element
- Array element kopieren: c :=A[i];
- Array element wijzigen: A[i] :=c;
- Array kopieren: A := B;

ADT_2D_ARRAY SPECIFICATIE

- **Elementen**: alle elementen zijn van hetzelfde opgegeven (eventueel samengestelde) datatype
- **Structuur**: 2 lineaire indexen waarvan het Carthesisch product i,j 1 op 1 correspondeert met de afzonderlijke elementen van elke 2D array waarde.
- **Type** Array: array [Indextype1,Indextype2] of ComponentType
- (b.v. FotoHelderheid[50,256] is 126)
- **Domein**: Carthesische product van Index ranges met toegestane elementwaarden
- (b.v. i,j [0..500,0..750] met FotoHelderheid [0..255])
- **Operaties**:
- Als A en B gelijksoortige array's, c een variabele van type element
- Array element kopieren: c :=A[i,j];
- Array element wijzigen: A[i,j] :=c;
- Array kopieren: A := B;

Serialisatie
AMF = Array Mapping Function

- Serialisatie van multi-dimensinale index naar 1D geheugenopslag m.b.v. AMF
- AMF = Array Mapping Function
- B.v. Het 1D adres, waar een 2D byte array element A[i,j] met rij i ε [0..5] en kolom j ε [0..3] opgeslagen wordt, is bij rij voor rij serialisatie
- 1D adres A[i,j]=c0+ i*rijlengte +j =c0+6i+j
- Waarbij c0 het begin adres is waar de compiler het array A laat beginnen.
- Bij kolom voor kolom serialisatie
- 1D adres A[i,j]=c0 +j*kolomlengte+i=c0+4j+i
- Als elk array element een 4 byte integer zou zijn dan wordt dit c0+j*kolomlengte*elementbreedte+i*elementbreedte=c0+16j+4i

Serialisatie voorbeeld

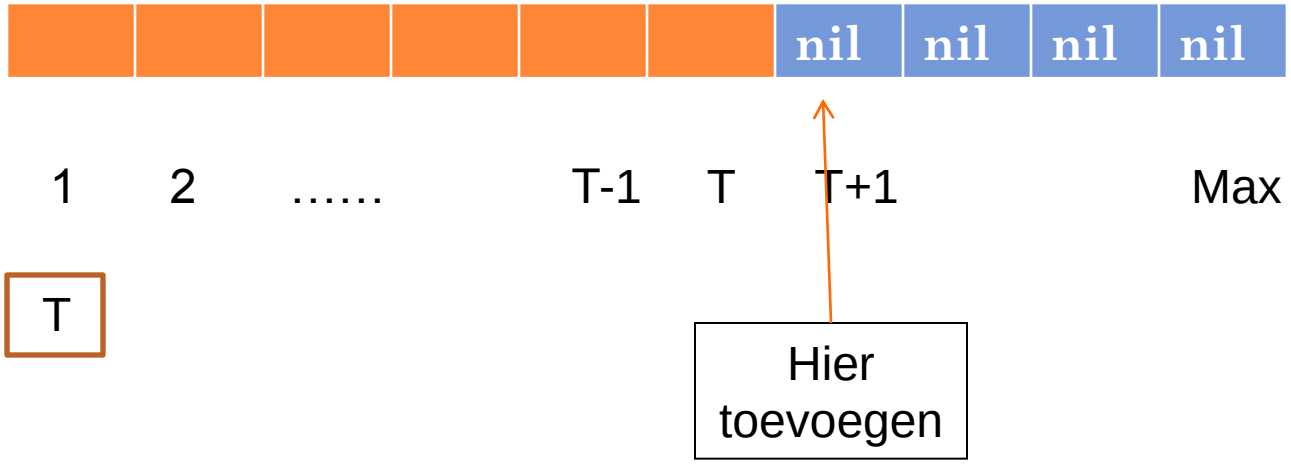
- Van het 2D byte array A[i,j] met i=[0..5] en j=[0..3] kunnen de 6*4=24 elementen met c0=0 en c1=4 afgebeeld worden op [0..24] opgeteld bij het adres (in wezen c0) waar de compiler array A laat beginnen.

A	j	0	1	2	3
i					
0		0	1	2	3
1		4	5	6	7
2		8	9	10	11
3		12	13	14	15
4		16	17	18	19
5		20	21	22	23

Multidim. Array voorbeelden

- kD array
- Voorbeeld 2D:
  - Grijswaarden foto
  - Google Earth
- Voorbeeld 3D:
  - Kleurenfoto (RGB lagen of matrix RGB tripletten)
- Voxelmodel (3D reconstructie x,y,z locaties):
  - Serial sections groep Fons Verbeek
  - Digitale videoscene (2D,t)
- Voorbeeld 4D:
  - Vormverandering in de tijd (3D,t):
    - embryonale ontwikkeling (groep Verbeek)

Lineaire rij (ongesorteerd,linksaangeschoven)



OPERATIES OP EEN ONGESORTEERDE 1D TOEGANGSRIJ

- Array met vaste lengte A[1..Max] beschikbaar
- Links aangeschoven rij, ongesorteerd (gevulde plaatsen, gevolgd door lege)
- Teller voor aantal gevuld
- Operaties bij LIFO gebruik:
- Aansluiten: als teller<Max; Teller->Teller+1; elem->A[Teller];
- Beurt (LIFO):
- als Teller>0; Beurt= A[Teller];A[Teller]=nil;Teller->Teller-1]
- Rij leeg? Y als Teller=0, anders N
- Rij vol? Y als Teller=Max, anders N
- Initialisatie/vrijmaken:
- A[i]=nil voor i ε [1..Max] (lege rij); Teller=0

1D tijdgesorteerde rij als FIFO rij

- Achteraan toevoegen (teller nu head pointer)
- Vooraan afhalen (tail pointer)

Randvoorwaarden:

Tail mag head niet inhalen

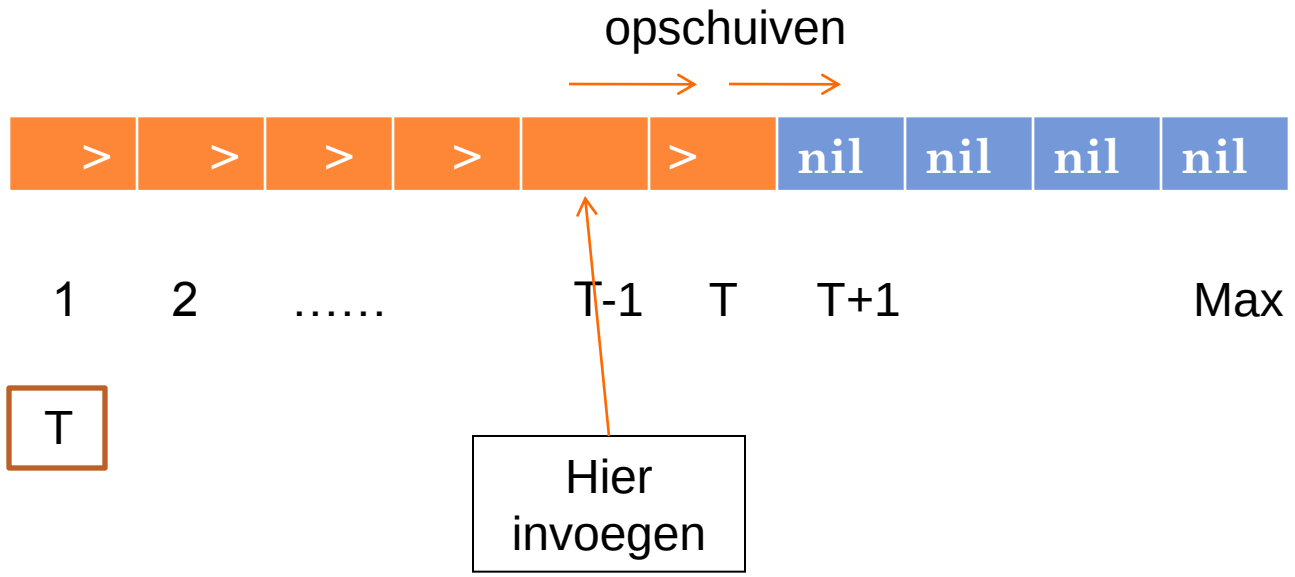
Tekortkoming:

Wachtrij loopt tegen eind array aan

Oplossing: bij lezen 1 element rest rij terugschuiven? Complexiteit?

Betere oplossing met behoud O(1) complexiteit?

LINEAIRE RIJ (GESORTEERD,LINKSAANGESCHOVEN)



Operaties op een gesorteerde 1D rij

- Door sorteervolgorde wordt invoegen complexer:
- Zoek invoegplek. Dom O(N) slim O(logN)
- Schuif alle elementen daar voorbij 1 op. O(N)
- Voeg element in. O(1)

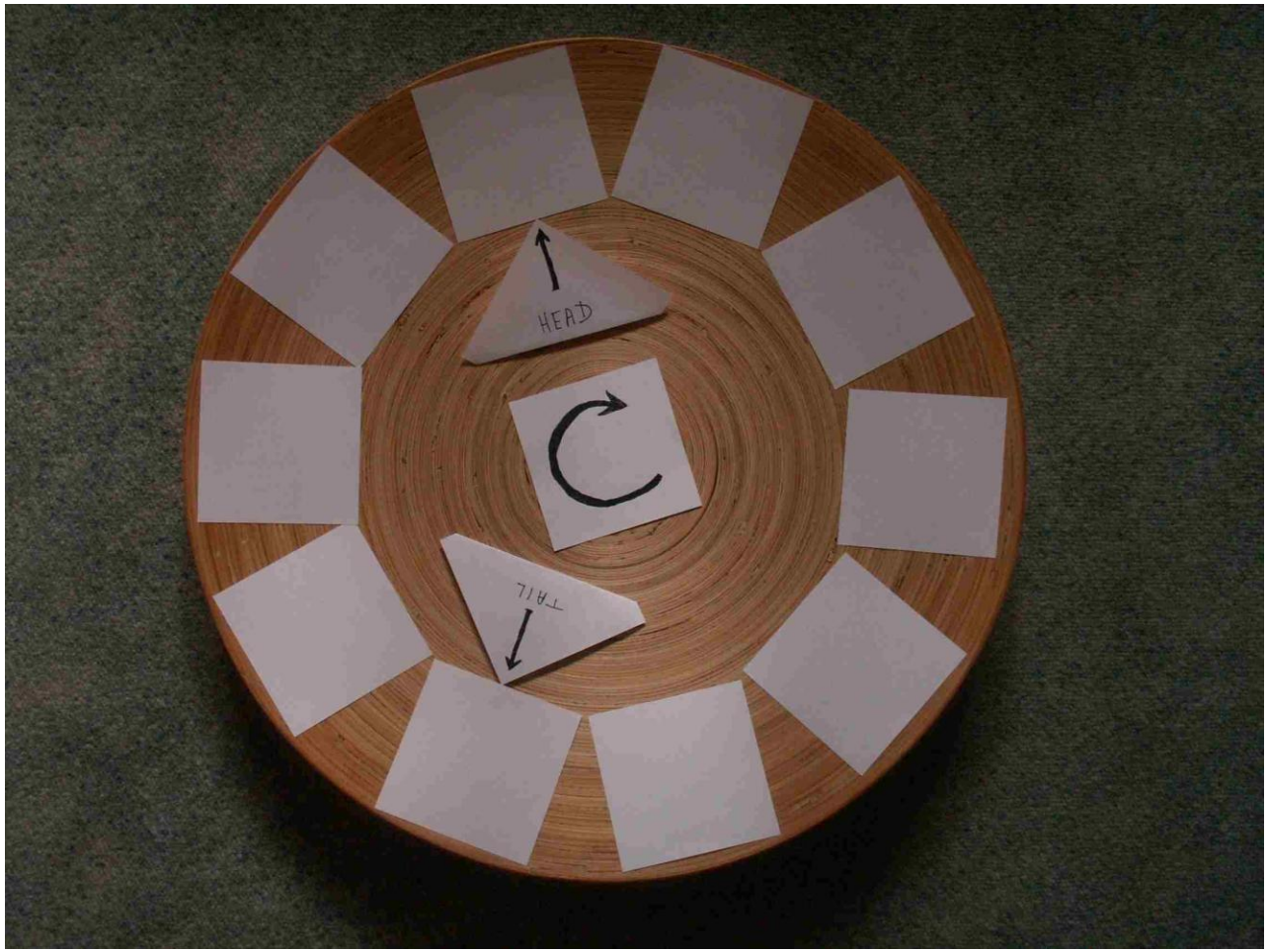
- Hoe voorkomen we het moeten opschuiven van reeksen data waarden? (duurste onderdeel)

Circulair 1D array en buffering

- Een eenvoudige manier om een lijst/rij geimplementeerd als 1D array cyclisch te doorlopen (FIFO queue/buffer b.v.) is m.b.v. modulo adressering:
- Als A domein [0..max] heeft kan oplopende teller T modulo (max+1) steeds opnieuw array A doorlopen
- Met een aparte head- en tail-teller die elkaar niet mogen inhalen, kan dan m.b.v. een 1D array een buffer of FIFO queue gerealiseerd worden.

ADT_FIFOQUEUE/COMMUNICATIE BUFFER

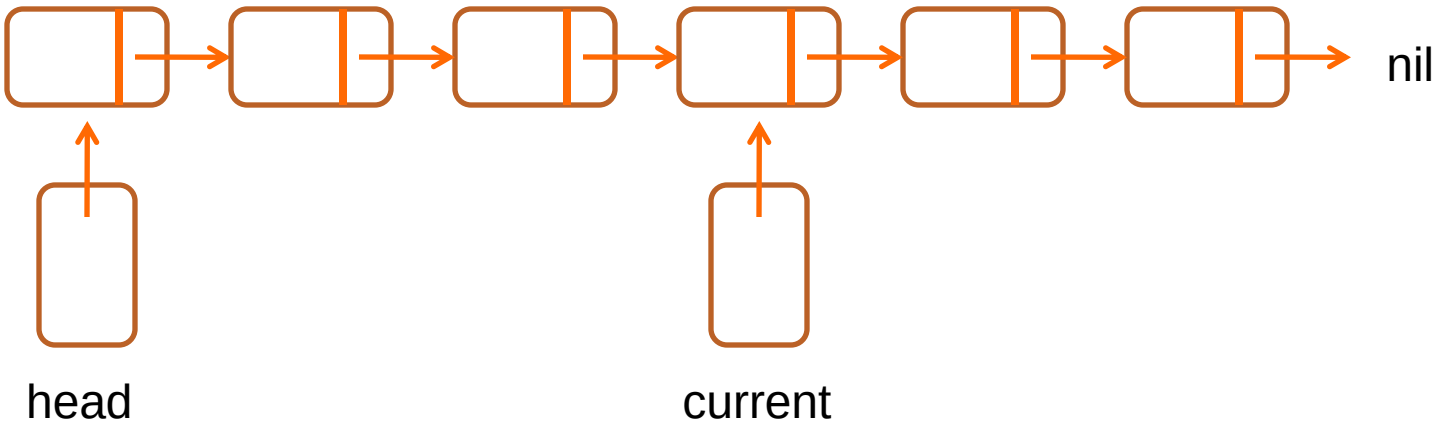
Voor specificatie en array implementatie zie los blad



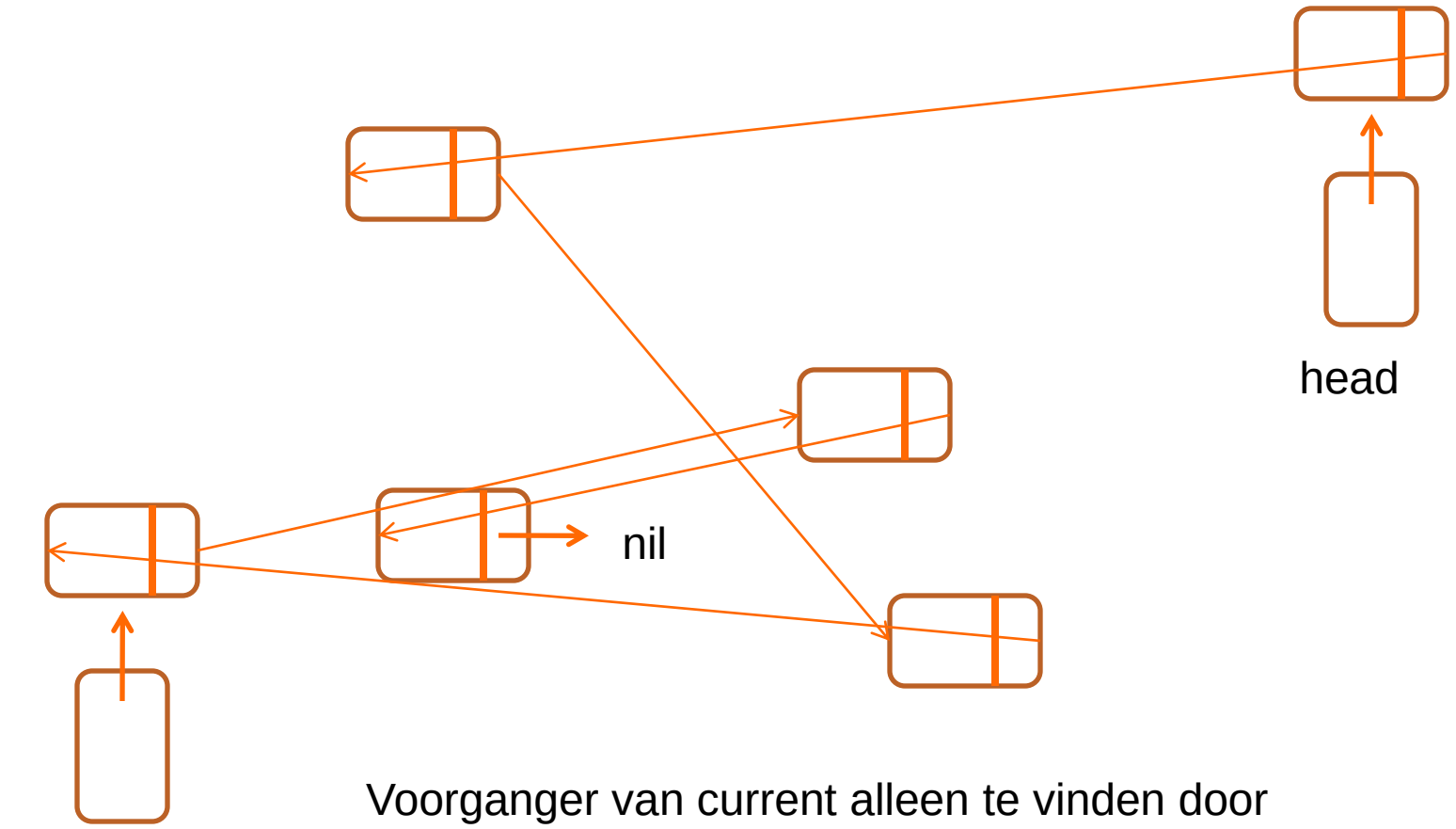
Single linked list

- Voordeel van array implementatie:
  - random access O(1)
- Nadeel van array implementatie:
  - Array misschien maar deels gebruikt
  - Array misschien te klein
- Linked list implementatie:
  - geheugen vrijmaken per nieuw bijkomend element
  - Voordeel: niet snel te klein (tot hele geheugen op)
    - vrijgegeven als niet meer nodig
  - Nadeel:
    - extra next_pointer nodig per element
    - Niet meer random access, maar lijst aflopen via pointers

Single linked list conceptueel lineair



Single linked list implementatie at random



Voorganger van current alleen te vinden door aflopen verbonden lijst vanaf head
Nummer vanaf head is lineaire index

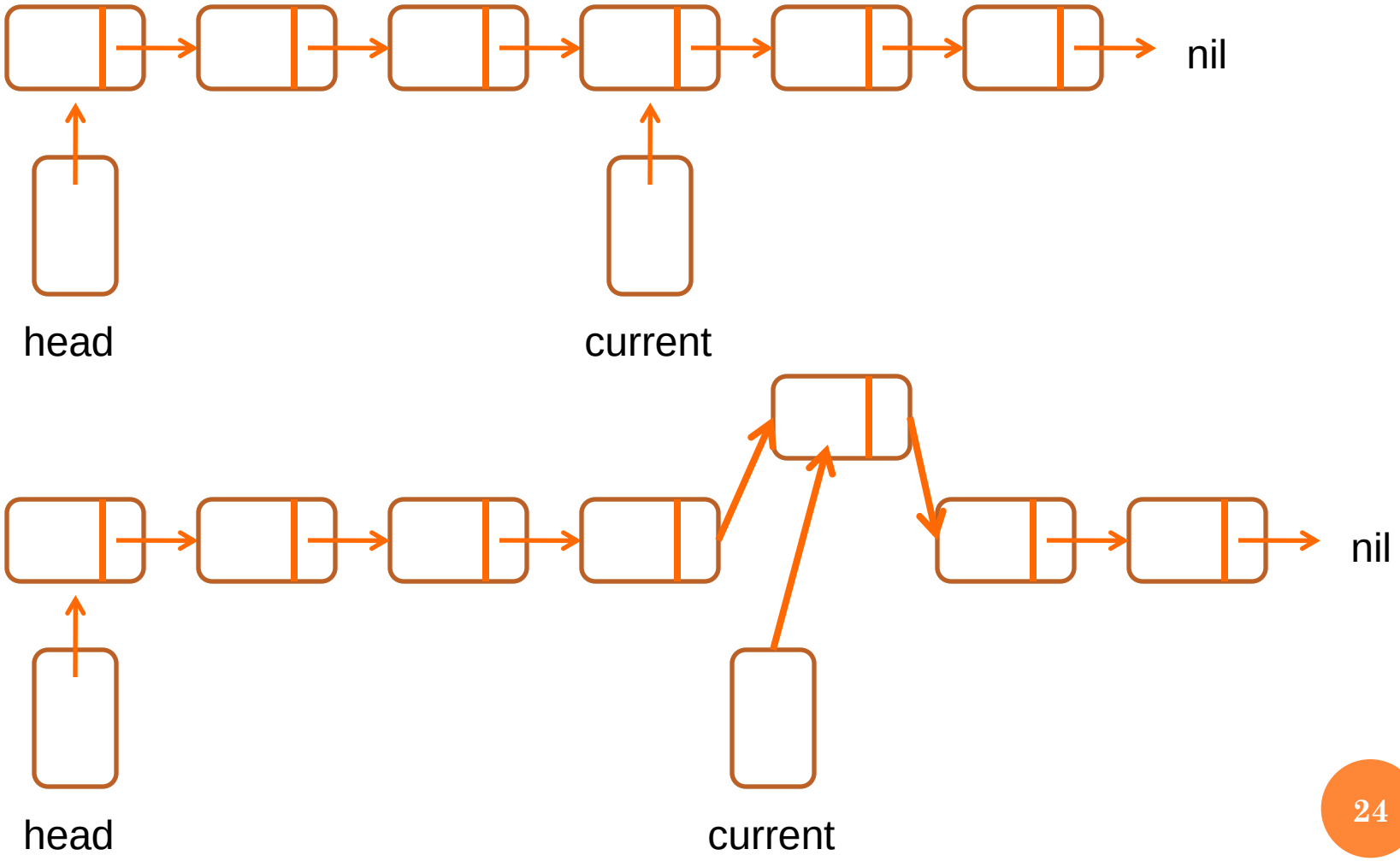
ADT Single linked list

- Specificatie zie apart vel
- Wel/niet ook een teller voor aantal knopen?
- Hoe ziet een lege linked list eruit?

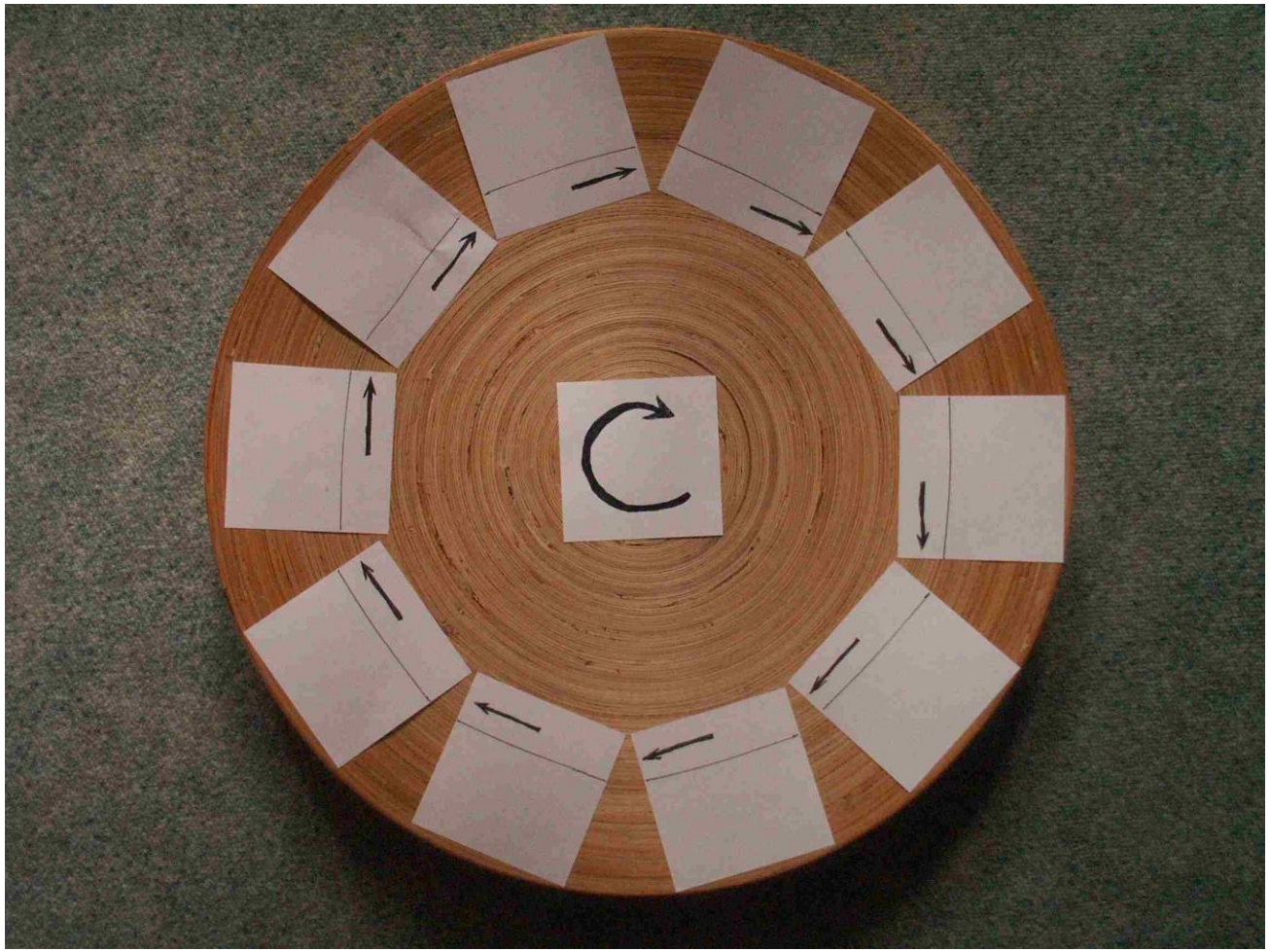
- Wat mist in deze specificatie?

- Implementatie komt nog

Single linked list insert



Circulair verbonden lijst



Lege circulair verbonden lijst

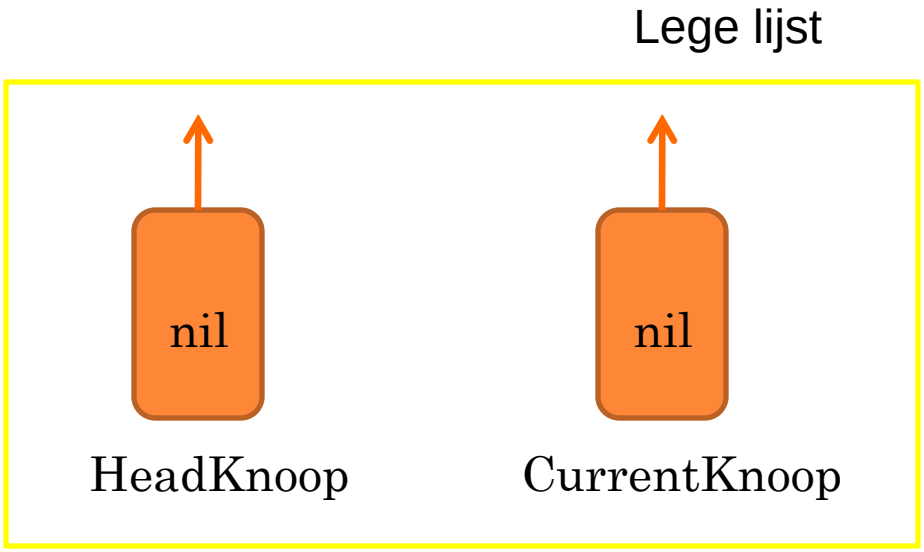
- Alleen Head- en Current-pointer waarde nil

Circulair verbonden lijst met 1 knoop

- Head- en Current-pointer wijzen beide naar de ene knoop
- Rechts-pointer ook

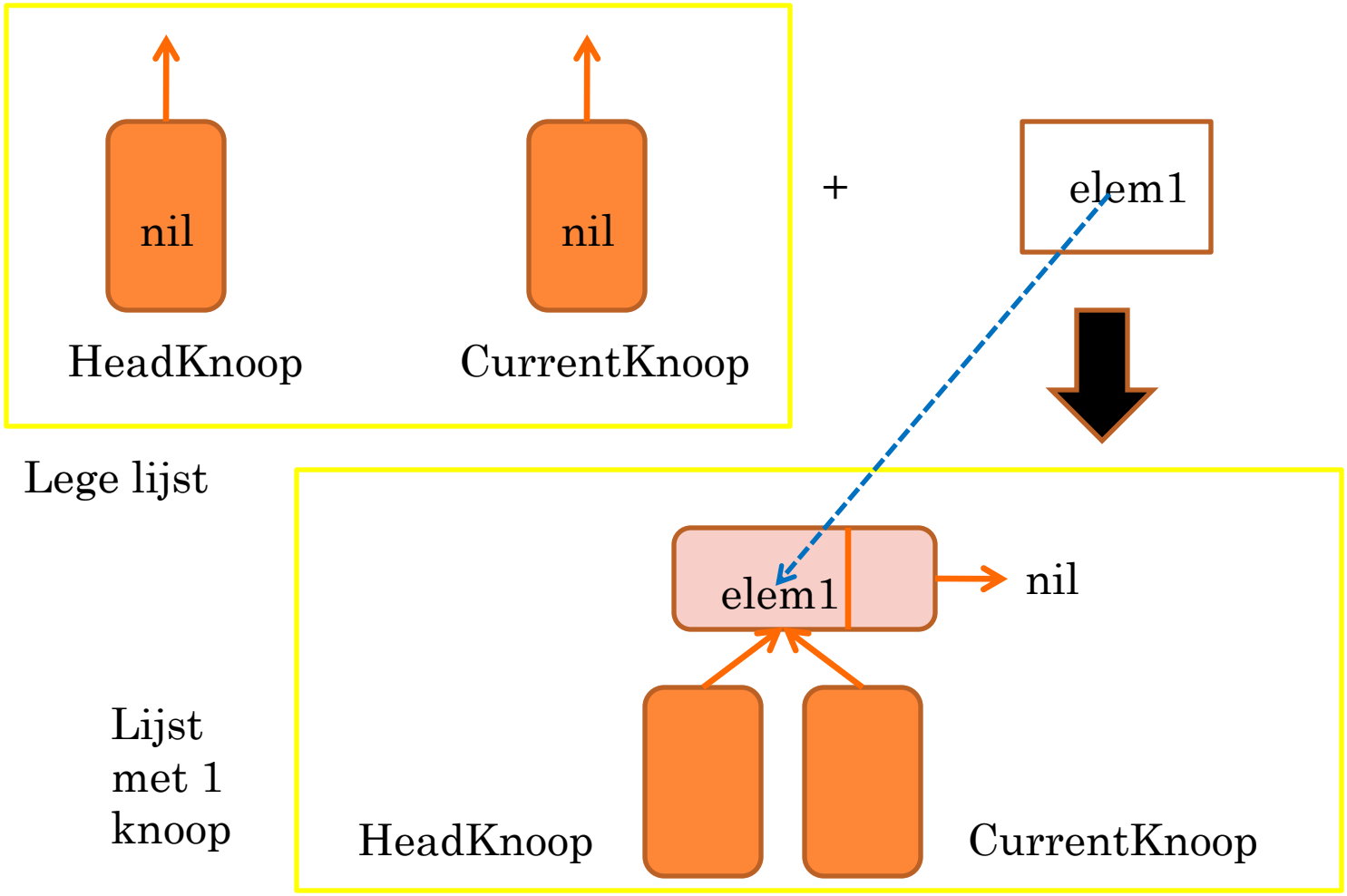
Single Linked List complete specificatie

- Zie uitgedeelde beschrijving
- Na Create lege lijst:
- **Create(var SLL:SingleLinkedList)**

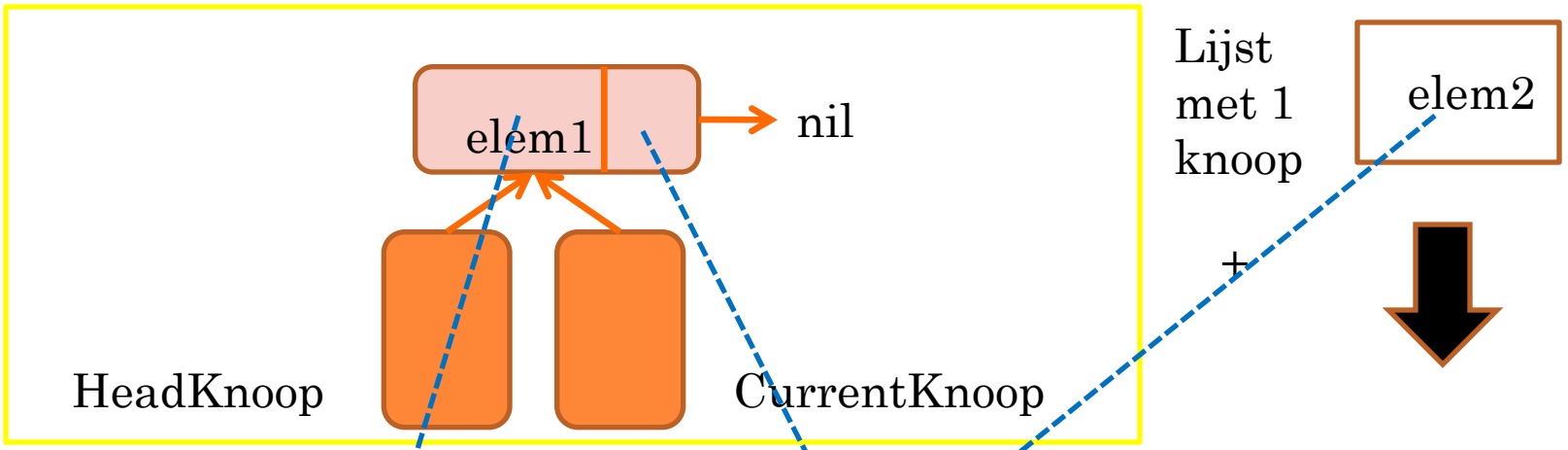


Single Linked List
toevoegen eerste element

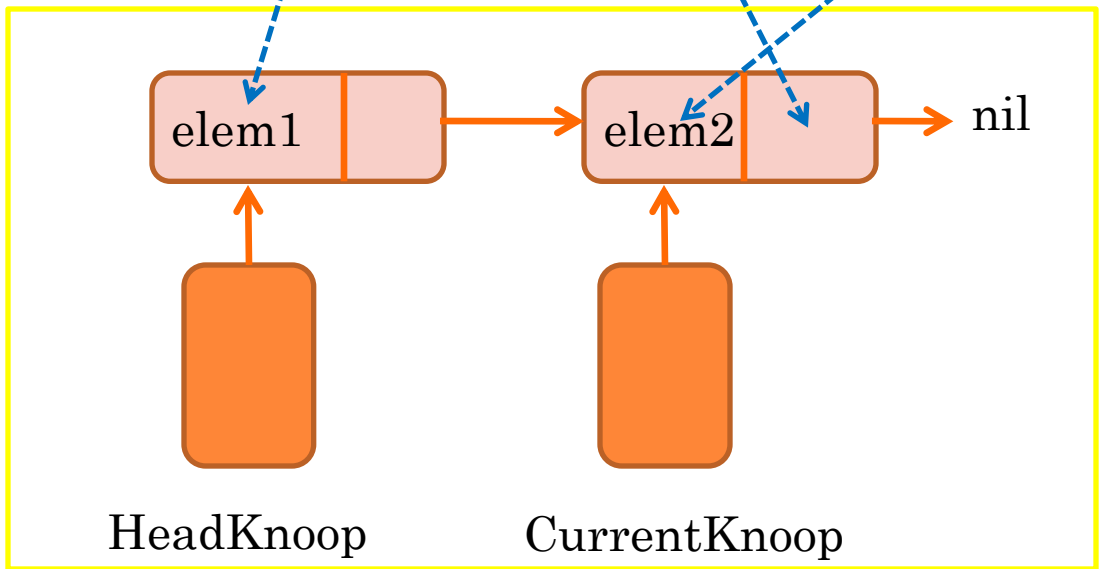
- **InsertThisOne**(SSL:SinglelinkedList;elem1:DataType)



Single Linked List
toevoegen tweede element

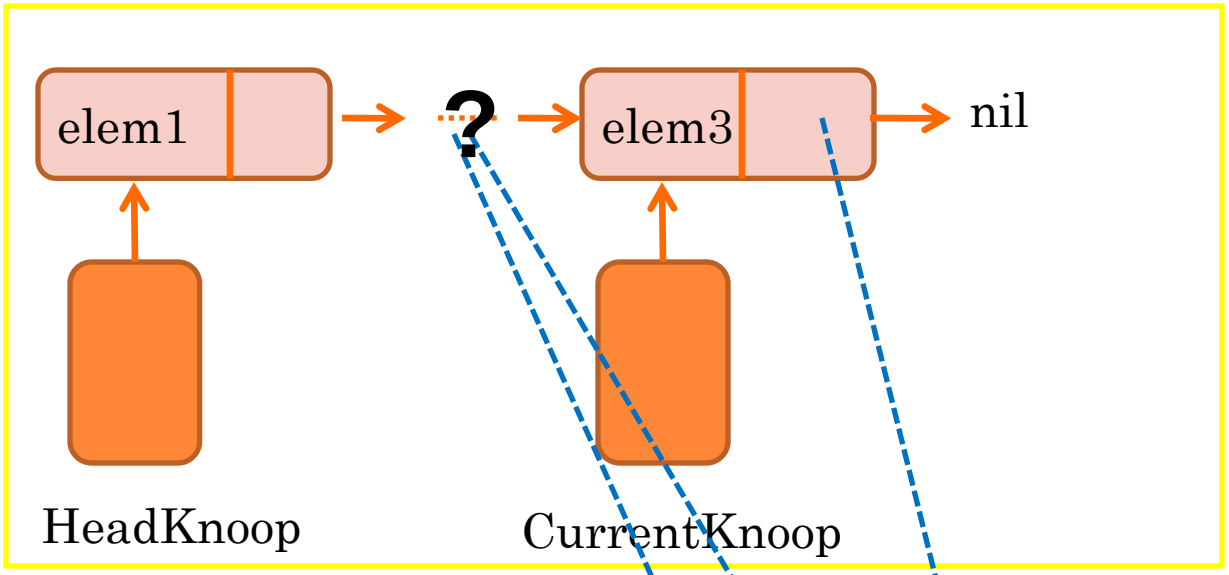


InsertThisOne(SSL:SinglelinkedList;elem2:DataType)



Lijst
met 2
knopen

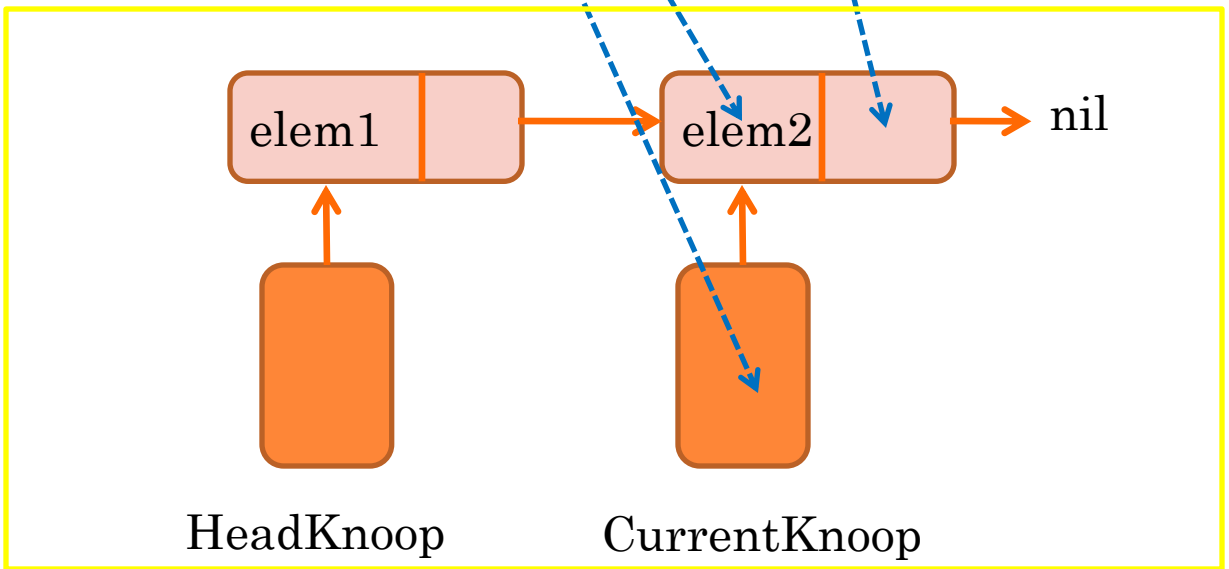
Single Linked List weghalen element



Lijst met 3 knopen

elem3

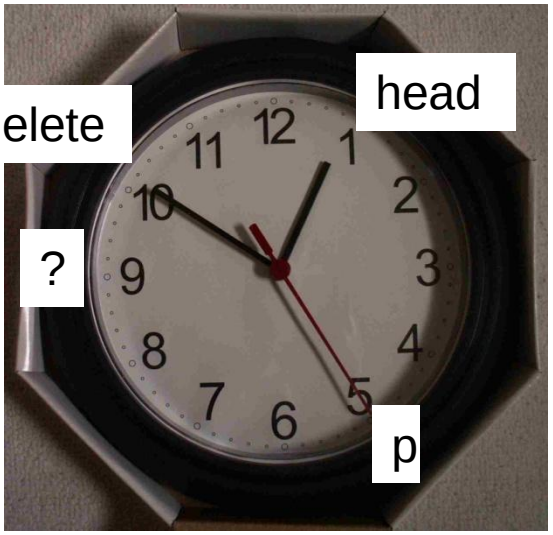
DeleteThisOne(SSL:SinglelinkedList)



Lijst met 2 knopen

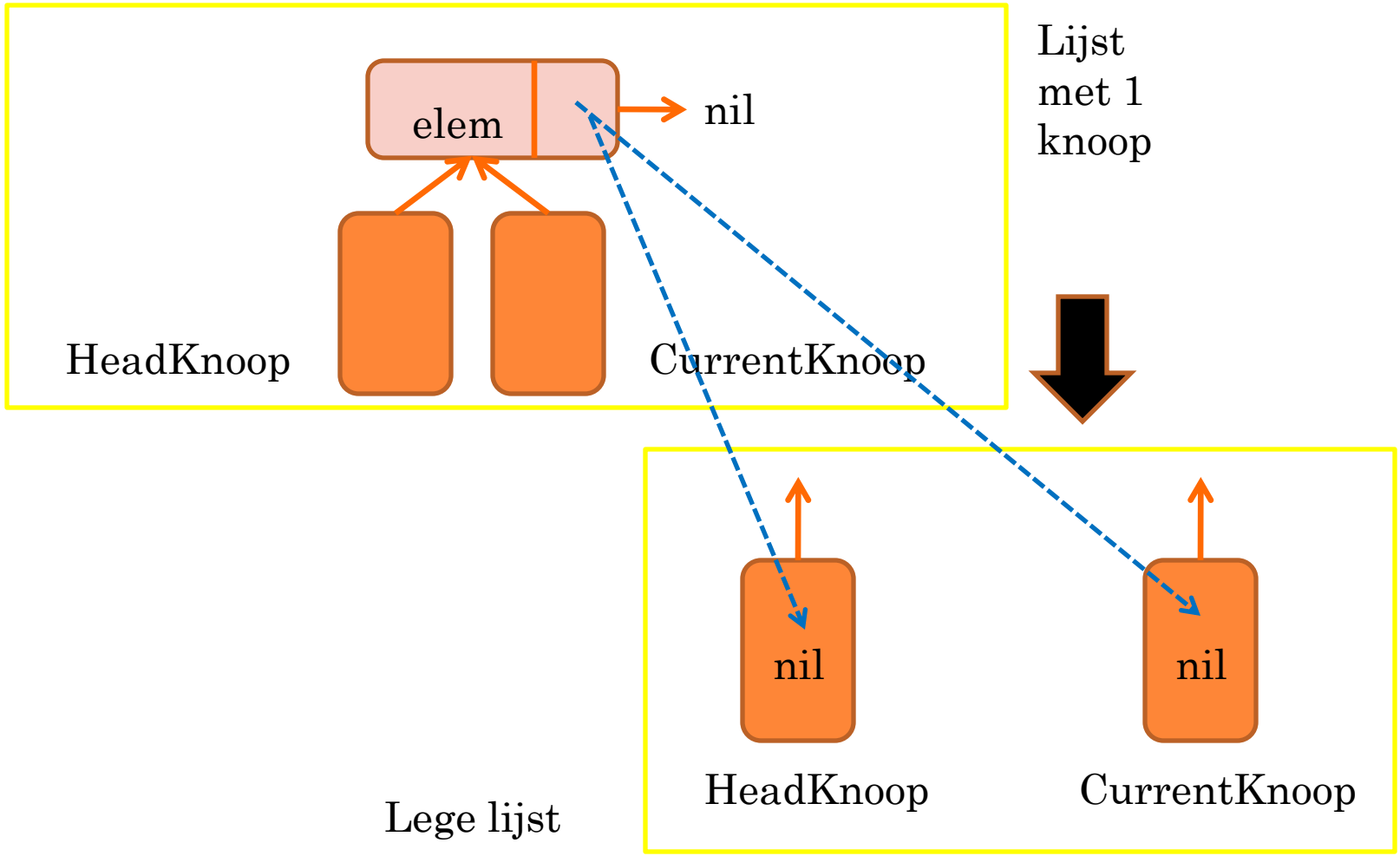
Weghalen knoop: voorganger?

- Zoek vanaf HeadKnoop naar knoop die naar huidige (te deleten) knoop wijst
- While loop met p vanaf
  - Headpointer tot
  - Knoop waarvan NextKnoop
  - CurrentKnoop is
- Dure operatie O(N)
- Snellere procedure O(1) als:
- Lijst dubbelverbonden, dan ook direct pointer naar voorganger beschikbaar



Single Linked List
weghalen laatste element

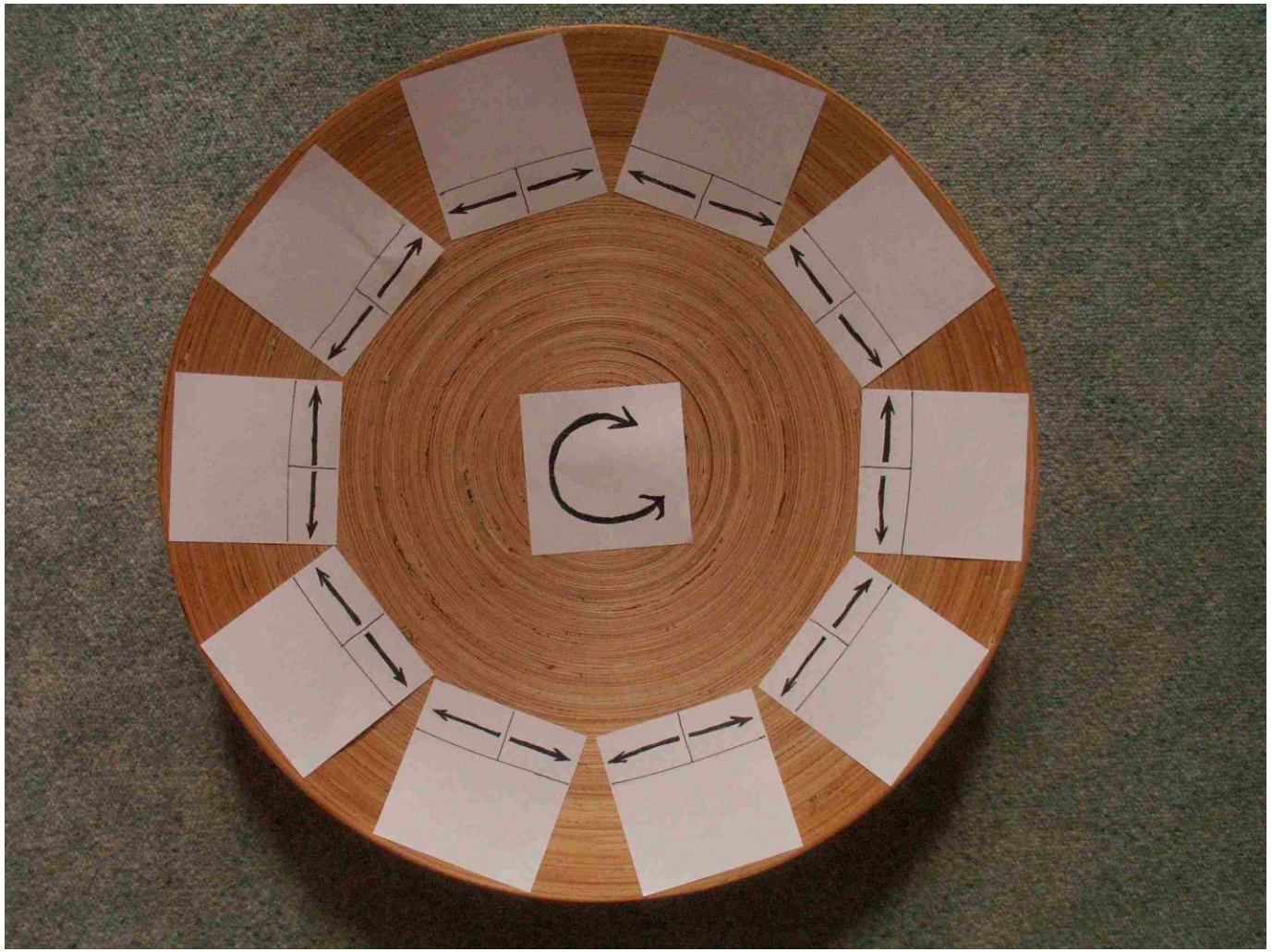
- **DeleteThisOne**(SSL:SinglelinkedList)



Mogelijk extra operaties ADT_SLL

- Extra teller voor aantal in lijst:
  - Moet bijgehouden worden tijdens inserts,deletes
- Extra operatie om i-de knoop in lijst vanaf Head tot CurrentKnoop te maken:
  - Dure operatie O(N) vanaf HeadKnoop lijst doorlopen;
  - Resultaat wordt CurrentKnoop
- Extra operatie om te zoeken naar knoop met bepaalde Data-Element waarde:
  - Dure operatie O(N) vanaf HeadKnoop zoeken naar eerste knoop met Data_Element = gezochte waarde;
  - Resultaat Y-> wordt CurrentKnoop

Circulair dubbel verbonden lijst



Lege circulair dubbel verbonden lijst

- Alleen Head- en Current-pointer waarde nil

Circulair dubbel verbonden lijst met 1 knoop

- Head- en Current-pointer wijzen beide naar deze ene knoop
- Links- en Rechts-pointer van deze knoop ook

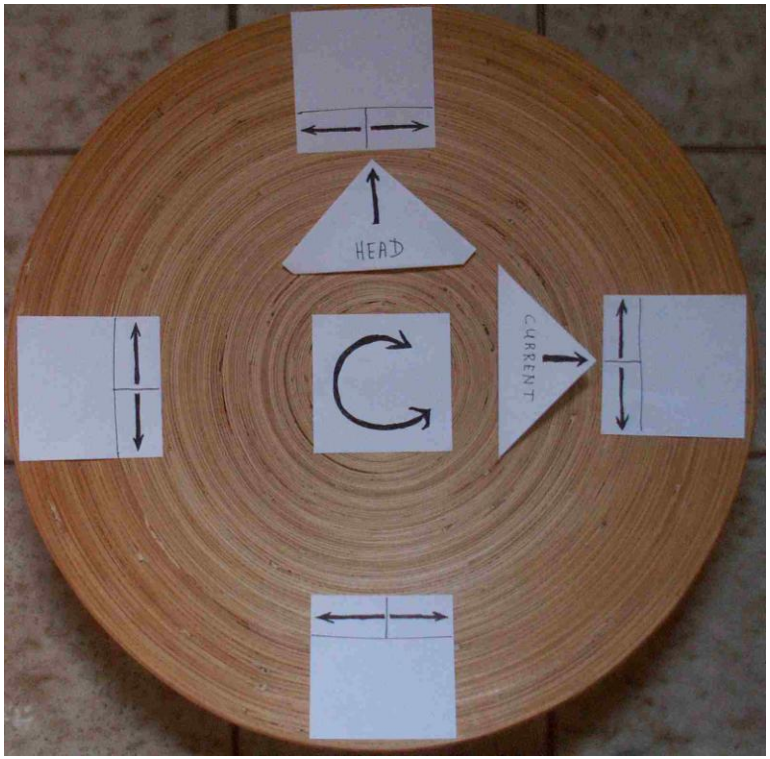
Dubbel voordeel circulaire lijsten

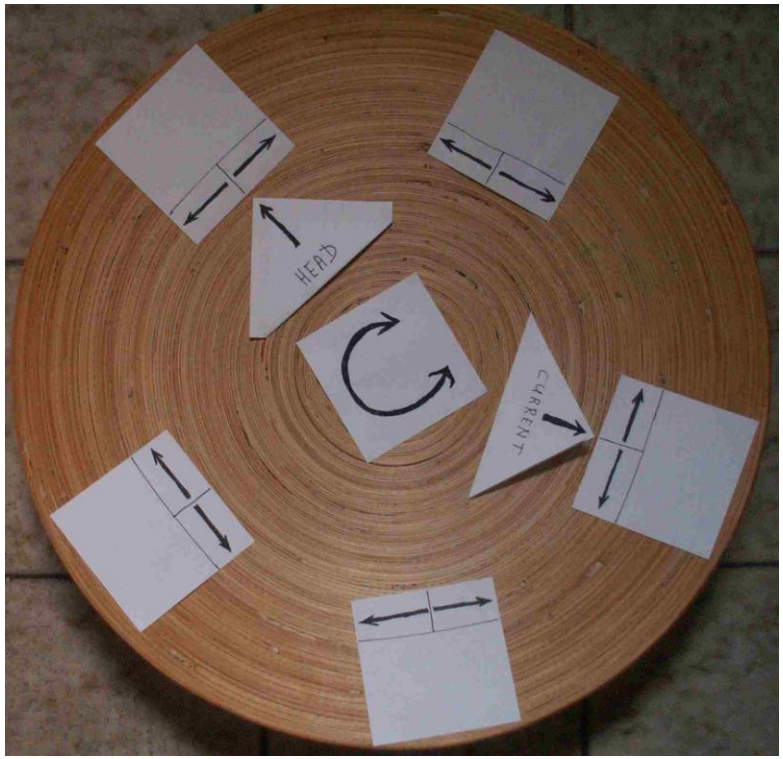
- Dubbel verbonden en tevens circulaire lijsten:

Operatie complexiteit:	SCLL	DCLL
FindFirst	O(1)	O(1)
FindNext	O(1)	O(1)
FindPrevious	**O(N)**	O(1)
RetrieveThisOne	O(1)	O(1)
InsertAfterThisOne	O(1)	O(1)
InsertBeforeThisOne	**O(N)**	O(1)
DeleteThisOne	**O(N)**	O(1)
UpdateThisOne	O(1)	O(1)
Empty	O(1)	O(1)
Last (Head-1)	**O(N)**	O(1)

Linked list versie
dubbel verbonden circulaire lijst

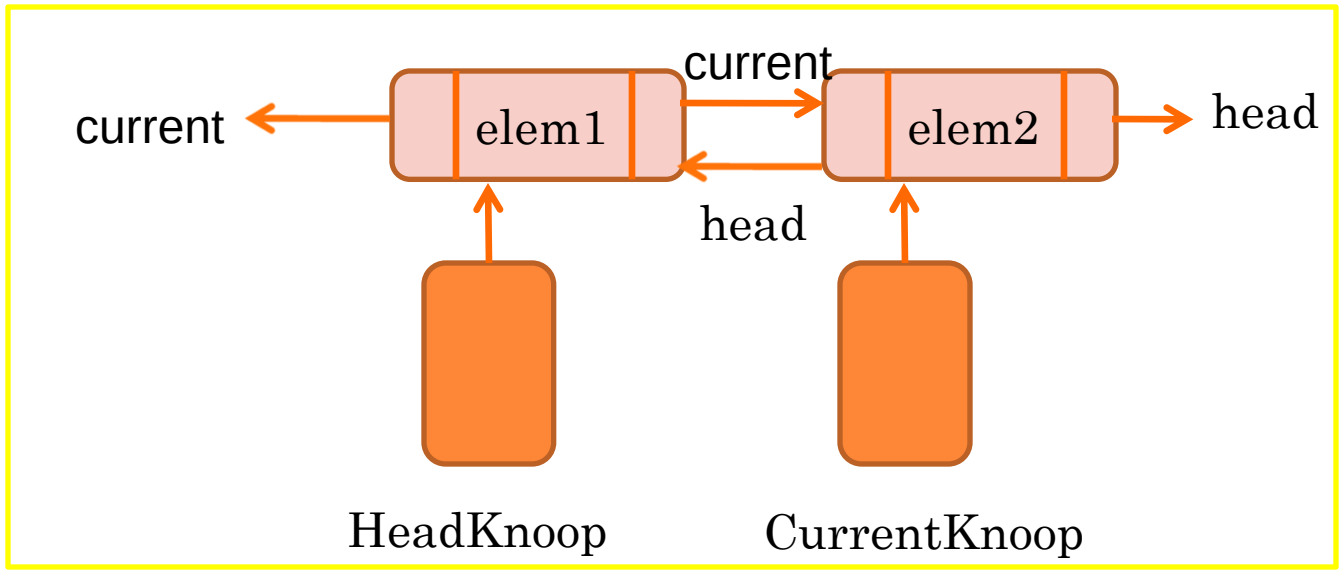
- Zie apart vel voor ADT_DCLL specificatie en linked list implementatie
- Wordt ook wel een RING genoemd





ADT_DCLL DELETE BIJ LAATSTE KNO(O)P(EN)

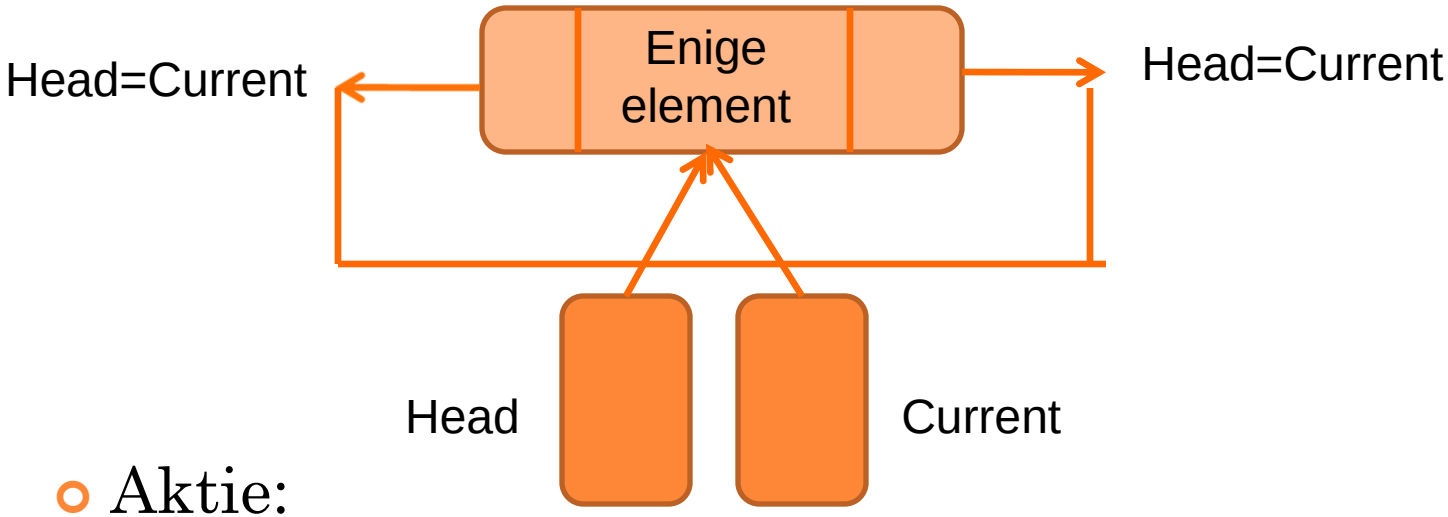
- Als de buren links en rechts gelijk zijn:



- Kunnen er nog twee knopen in de lijst staan

ADT_DCLL DELETE LAATSTE KNOOP

- Herkenning zonder teller aantal in lijst:
  - CurrentKnoop^.RightKnoop=CurrentKnoop^.LeftKnoop
  - En CurrentKnoop^.RightKnoop=CurrentKnoop



- Aktie:
  - Dispose(CurrentKnoop)
  - HeadKnoop:=nil
  - CurrentKnoop:=nil

ADT en data in een file

- Gegevens opgeslagen in een file
- Sequentie: [0..Max) geindexeerde data elementen met regelmatig oplopende index die tot ∞ door zou kunnen lopen, maar in de praktijk ergens eindigt door eindigheid beschikbaar geheugen.
- Afhankelijk van hoe gegevens in de file zijn gestructureerd, kunnen er I/O functies omheen geschreven worden:
- Simpelste file opzet (Unix, Pascal): byte-stream
- Ingewikkelder opzet: variabele lengte records met vaste lengte voor index-key

ADT_ByteStreamFile Magneetband analogon

- Data niet verder gestructureerd dan dat het een lineair geordende rij bytes [0..Max) is in een benoemde filenaam binnen een hierarchische directory en waarop de volgende file functies zijn gespecificeerd:
  - Create (lege) File met bepaalde naam in (sub)dir
  - Delete File
  - Open File voor lezen en/of schrijven (Pascal of!)
  - Sluit File
  - Skip n bytes op File (voorwaarts/achterwaarts)
    - (niet in Pascal!)
  - Lees m bytes van File -> buffer (Pascal 1 byte!)
  - Schrijf k bytes van buffer -> File (Pascal 1 byte!)

Magneetband analogon voor records (vaste of variabele lengte)

- In plaats van een serie bytes kan een magneetband ook gedacht worden als een sequentie van records:
- - van vaste lengte (steeds m bytes/record lezen/schrijven)

- - van variabele lengte (steeds zoveel bytes lezen totdat EOR byte code bereikt is)