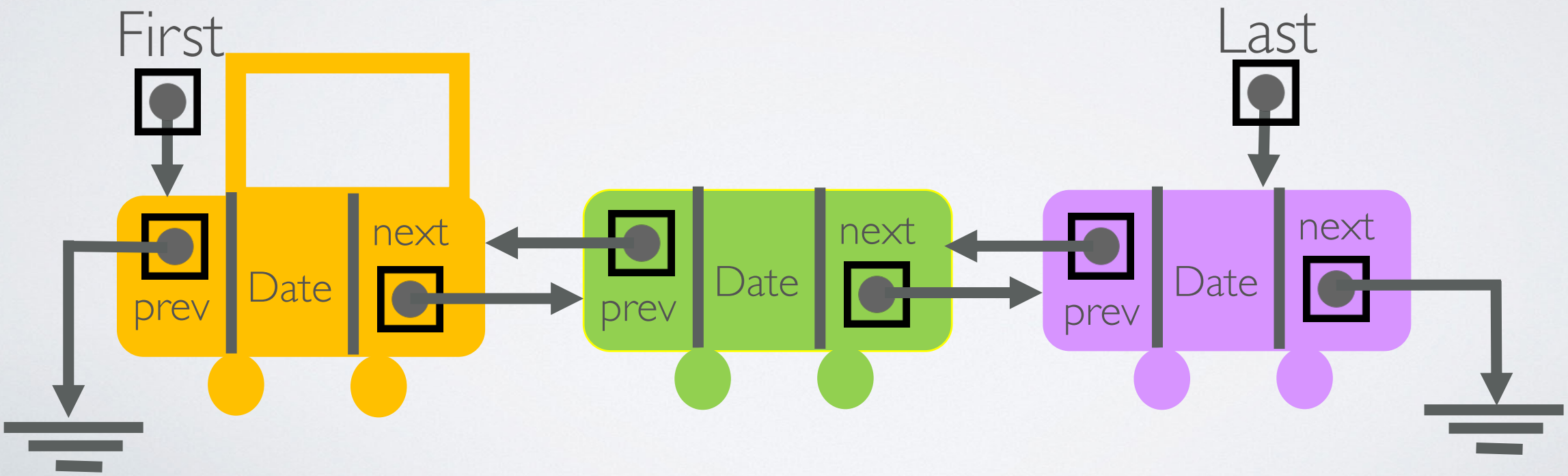
UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

SDA CURS 2:

LISTA DUBLU INLANTUITA, LISTA SIMPLU INLANTUITA CIRCULARA, STIVA, COADA

LISTA DUBLU INLANTUITA

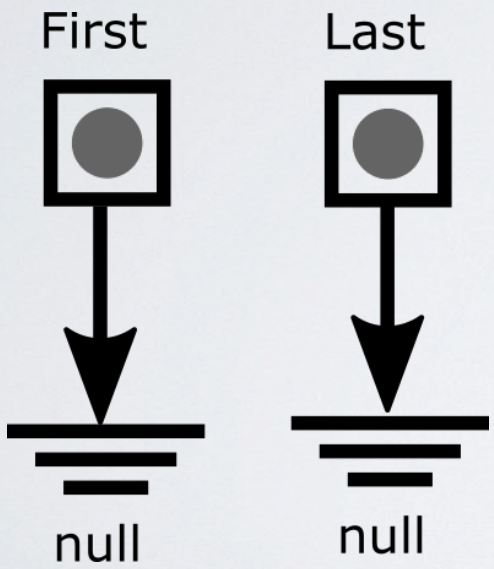
- Tip special de lista in care informatia de legatura a fiecarui element cuprinde atat adresa elementului *precedent* (**`prev`**) din lista cat si adresa elementului *urmator* (**`next`**) – inlantuire bidirectionala



LISTA DUBLU INLANTUITA – OPERATII

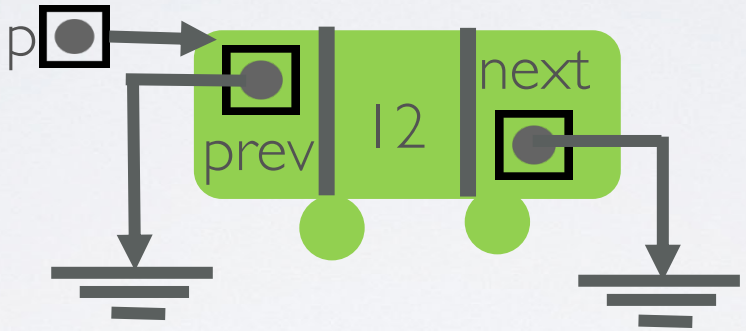
UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

create_empty:
creaza o lista goala



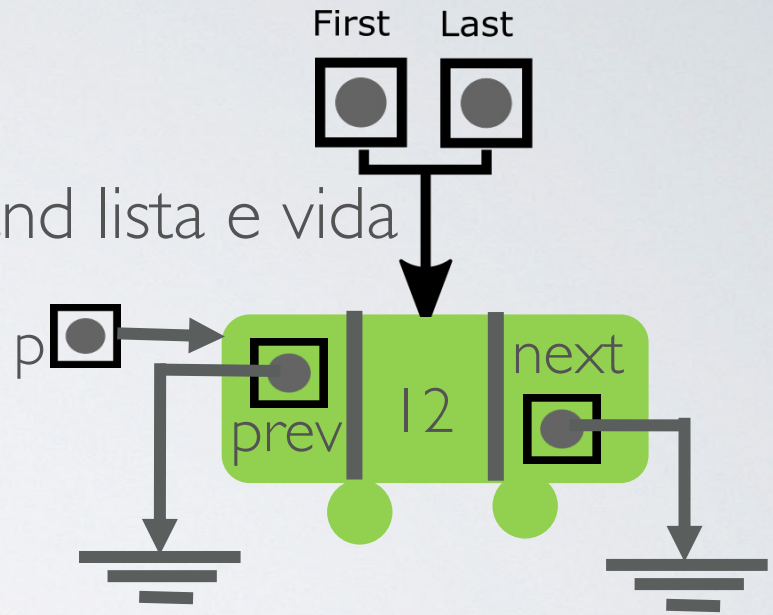

```
NodeDL *first = NULL, *last = NULL;
```

insert_first - *cream elementul*




```
NodeDL *p = (NodeDL *)malloc(sizeof(NodeDL));
p->key  = 12;
p->next = NULL;
p->prev = NULL;
```

cand lista e vida




```
if (first == NULL)
{
    first = last = p;
}
```

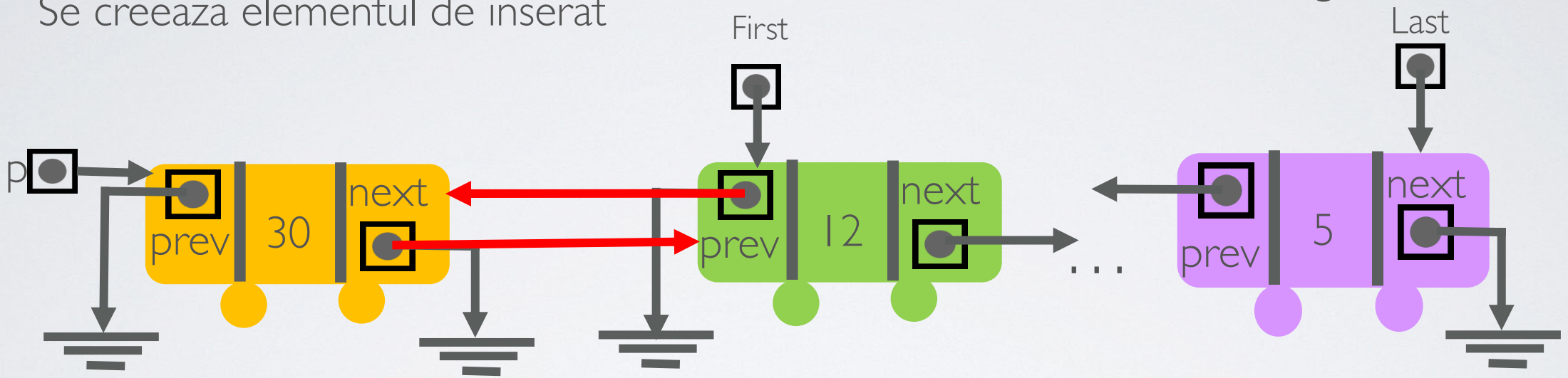
LISTA DUBLU INLANTUITA – INSERT_FIRST

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

insertFirst – cand lista are elemente

Se creeaza elementul de inserat

Se actualizeaza legaturile din lista.



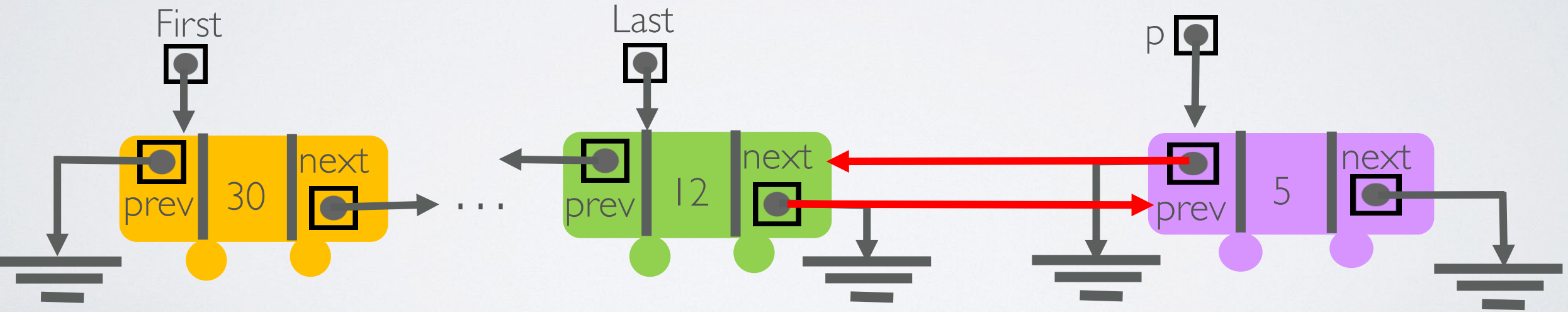

```
NodeDL *p = (NodeDL *)malloc(sizeof(NodeDL));
p->key  = 30;
p->next = NULL;
p->prev = NULL;
```

```
if (first != NULL)
{   /* list is not empty */
    p->next     = first;
    first->prev = p;
    first       = p;
}
```

LISTA DUBLU INLANTUITA INSERT_LAST

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

2. Inserare la final (append):
   - Lista goala - ca si la insertFirst
   - Lista nu e goala

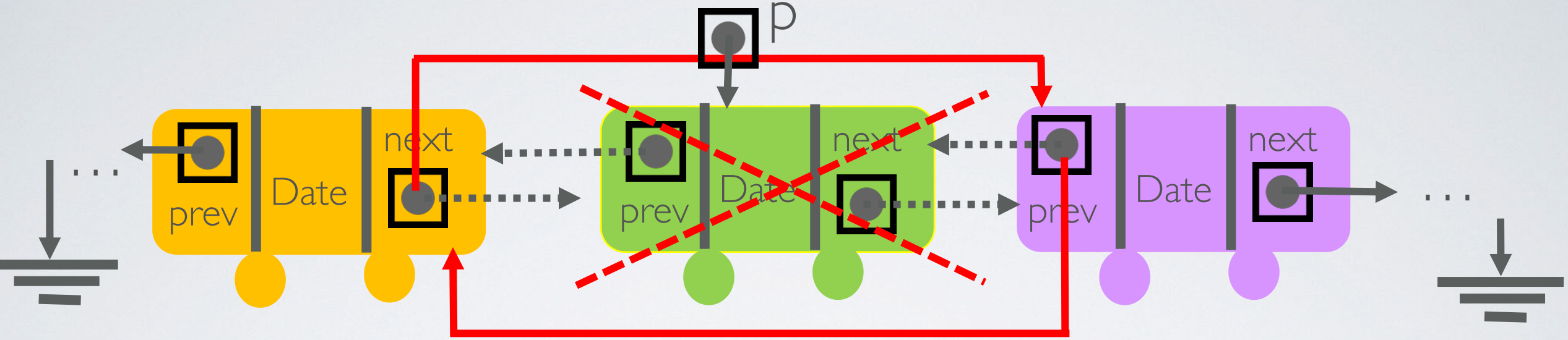


LISTA DUBLU INLANTUITA
INSERT IN INTERIORUL LISTEI

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

3. Inserare in lista ordonata, la pozitia k, inainte/dupa o anumita cheie – cazuri posibile:
   - Lista goala: ca si mai inainte
   - Lista cu elemente:
     - Inainte de primul nod (i.e. inserare la inceput)
     - Dupa ultimul nod (i.e. append)
     - **In interiorul listei**
       - trebuie traversata lista, si stabilit un pointer (*de ce nu 2?*):
         - nodul curent (va deveni ***`next`*** pt nodul inserat)
         - nodul inserat va deveni ***`prev`*** pentru nodul curent
   - **Exercitiu!!!!**

LISTA DUBLU INLANTUITA DELETE_KEY (CAZUL GENERAL)

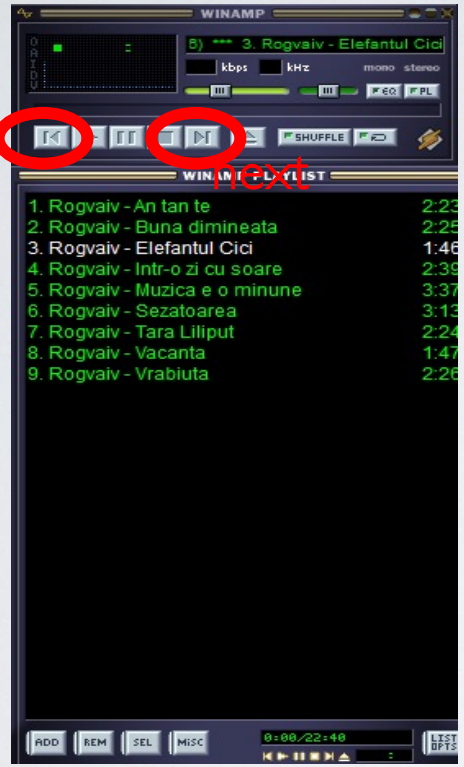


Cazuri speciale?

Pseudocod:

```
LIST-DELETE(L,p)
  if p.prev != NIL
      p.prev.next = p.next
  else L.first = p.next
  if p.next != NIL
      p.next.prev = p.prev
  else L.last = p.prev
```

LISTA DUBLU INLANTUITA – UTILITATE



- Player de muzica cu butoane de next si previous
- Cache-ul din browsere, cu functionalitatea de *BACK-FORWARD* pe pagini
- Functionalitatea de *UNDO-REDO*
- Sisteme de operare – *planner* de fire de executie

LISTA SIMPLU VS DUBLU INLANTUITA EFICIENTA

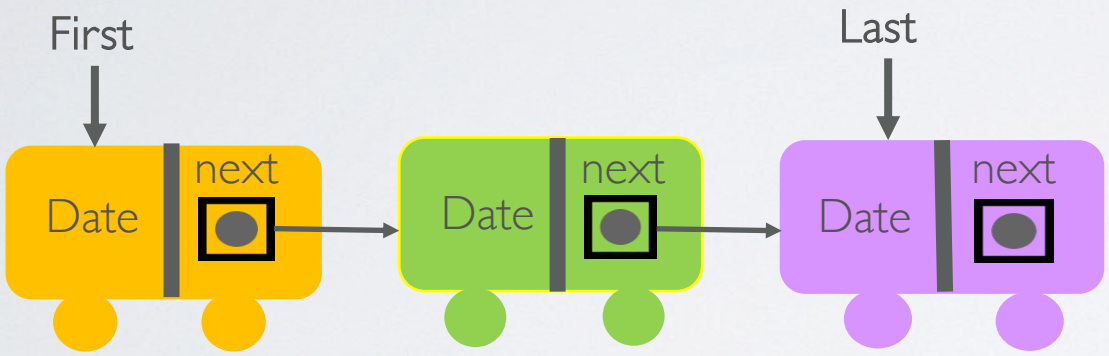
UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Operatie	Lista simplu inlantuita, cu first si last	Lista dublu inlantuita, cu first si last
Inserare la inceput	O(1)	O(1)
Inserare la sfarsit	O(1)	O(1)
Inserare in interiorul listei (adresa cunoscuta)	O(n)	O(1)
Cautare dupa cheie	O(n)	O(n)
Stergere de la inceput	O(1)	O(1)
Stergere de la sfarsit	O(n)	O(1)
Stergere din interior (adresa nod data)	O(n)	O(1)

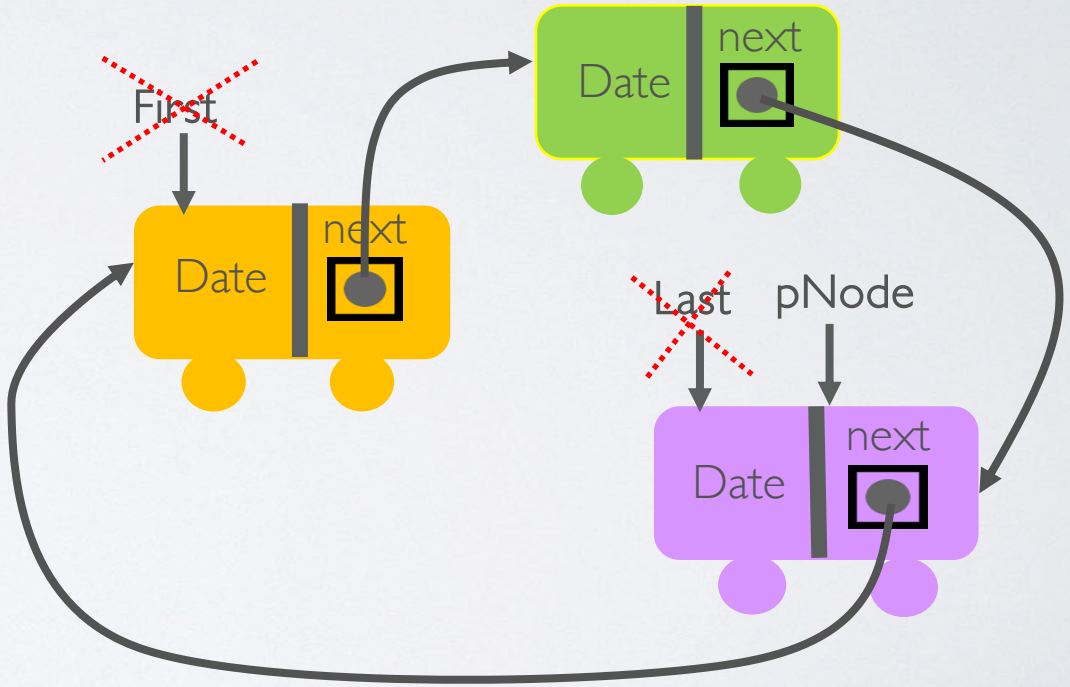
LISTA CIRCULARA

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Lista simplu/dublu inlantuita cu proprietatea ca primul element din lista urmeaza dupa ultimul element (pointerul next al ultimului element pointeaza catre primul element).



Lista simplu inlantuita



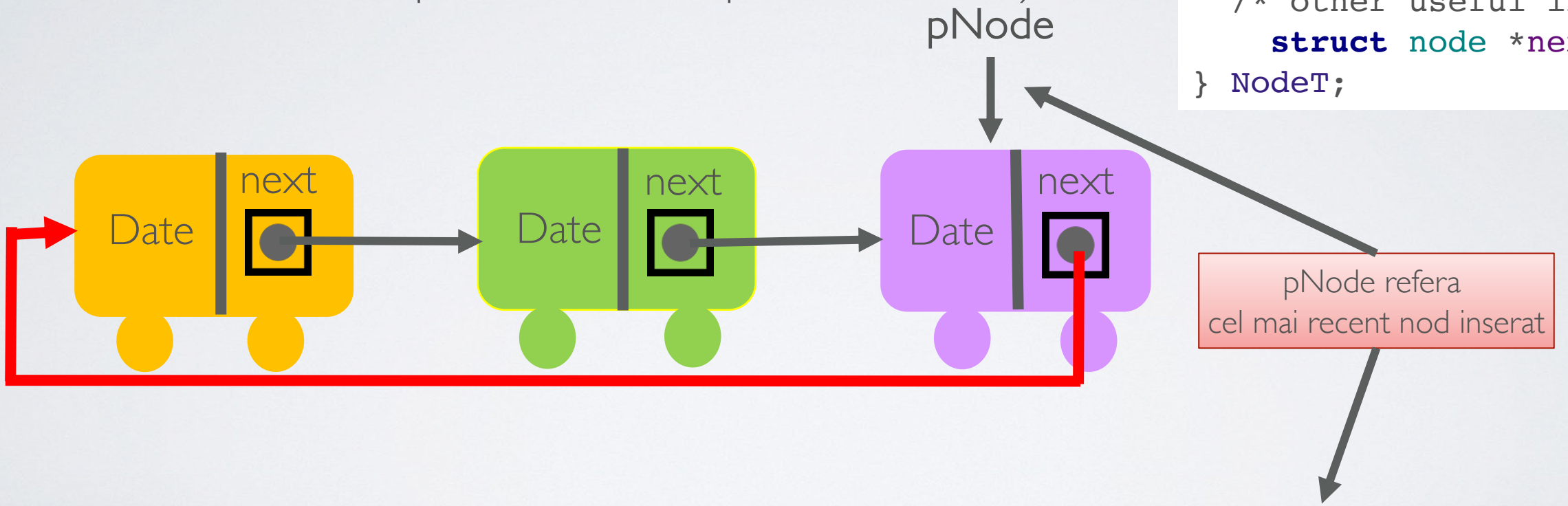
Lista simplu inlantuita circulara

LISTA CIRCULARA

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Lista simplu/dublu inlantuita cu proprietatea ca primul element din lista urmeaza dupa ultimul element (pointerul next al ultimului element pointeaza catre primul element).

```
typedef struct node {
    int key;
  /* other useful info */
    struct node *next;
} NodeT;
```



*Se foloseste un singur pointer **pNode** pentru a indica un element din lista – e.g. **cel mai recent nod inserat***

LISTA CIRCULARA - OPERATII

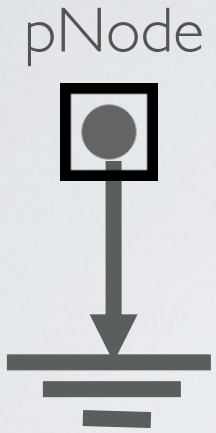
UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Cautarea unui element (dupa cheie)
- Inserarea unui element
  - Pe **prima**/ultima pozitie in raport cu cel mai recent nod inserat (**inainte** / dupa pNode)
  - Dupa un anumit element
  - …
- Stergerea unui element
  - Primul/ultimul (inainte / dupa pNode)
  - Dupa cheie
- Exercitii !

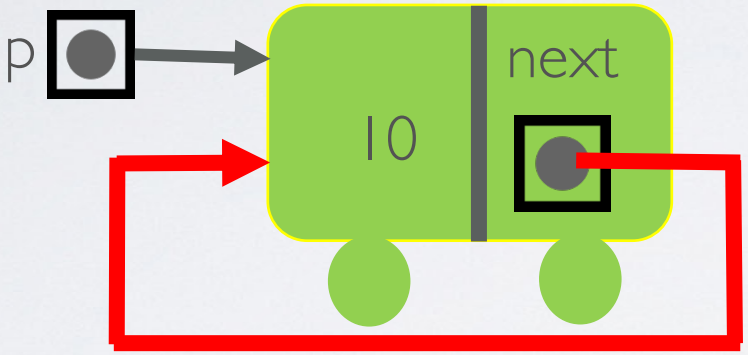
LISTA SIMPLU INLANTUITA CIRCULARA - OPERATII

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

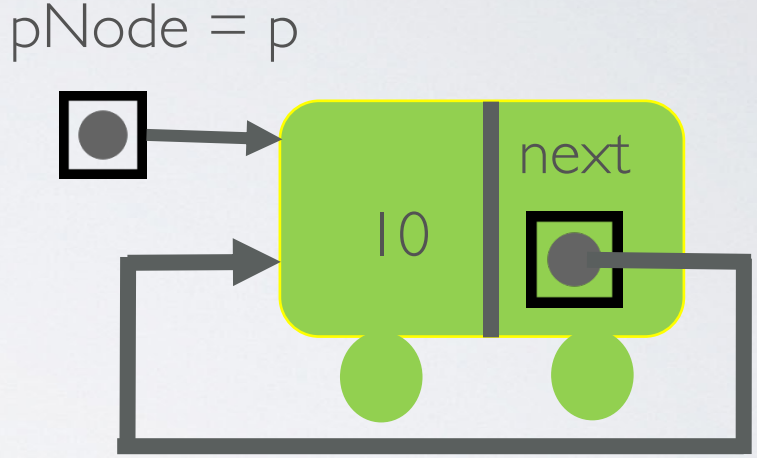
insert_first:




```
NodeT *pNode = NULL;
```




```
NodeT *p = (NodeT *) malloc(sizeof(NodeT));
p->key = 10;
p->next = p;
```




```
if (pNode == NULL)
{   /* empty list */
    pNode = p;
}
else
{   /* list is not empty */
    p->next = pNode->next;
    pNode->next = p;
    pNode = p; /* pNode points to most recently
added element*/
}
```

LISTA CIRCULARA - UTILITATE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Utila pentru implementarea unei cozi
  - Mentinem pointer catre ultimul nod inserat (*tail*); *head* va fi nodul care urmeaza dupa *tail*
- Liste circulare dublu inlantuite se utilizeaza in implementarea unor structuri complexe (e.g. Fibonacci Heaps)

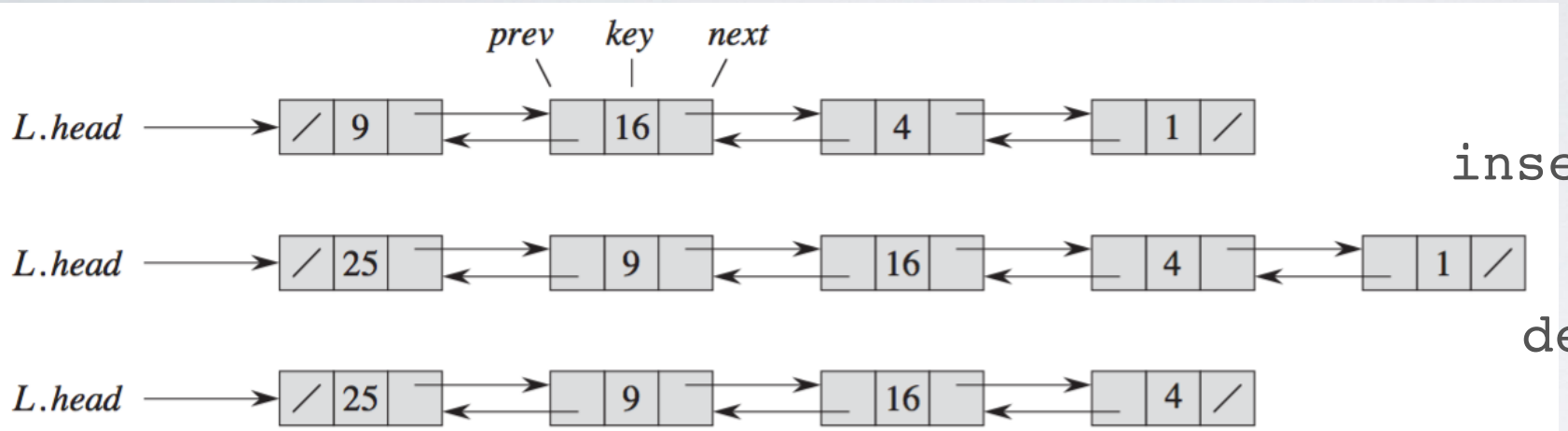
LISTE INLANTUITE CU SANTINELA

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- O **santinela** este un element de lista *dummy* care ne permite sa eliminam necesitatea de a trata cazurile speciale (NULL).
  - nu contine informatie
  - contine toate campurile celorlalte elemente de lista (i.e. legaturile **`prev`** si **`next`**)
  - oricand avem in cod o referinta catre NULL, se inlocuieste cu referinta catre santinela.
  - lista dublu inlantuita -> lista dublu inlantuita circulara (daca utilizam santinela)

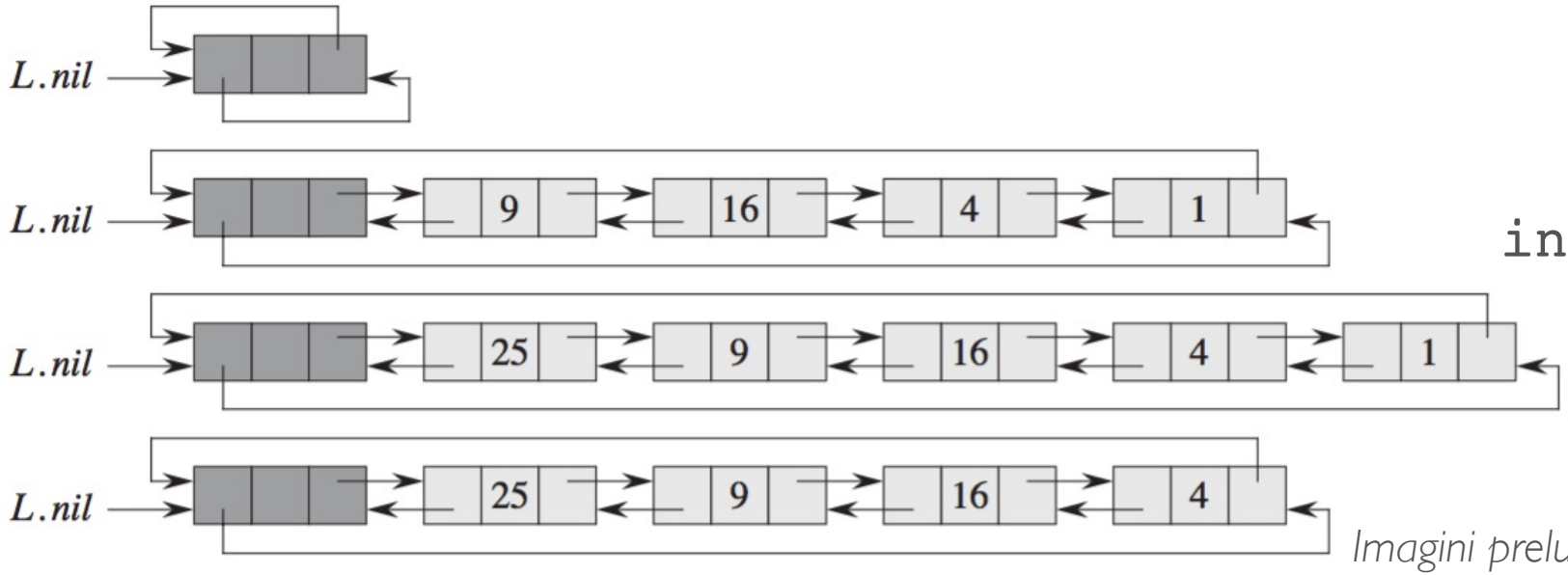
LISTA DUBLU INLANTUITA - FARA/CU SANTINELA

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA




```
insert_first(L, 25)
```

```
delete(L, 1)
```



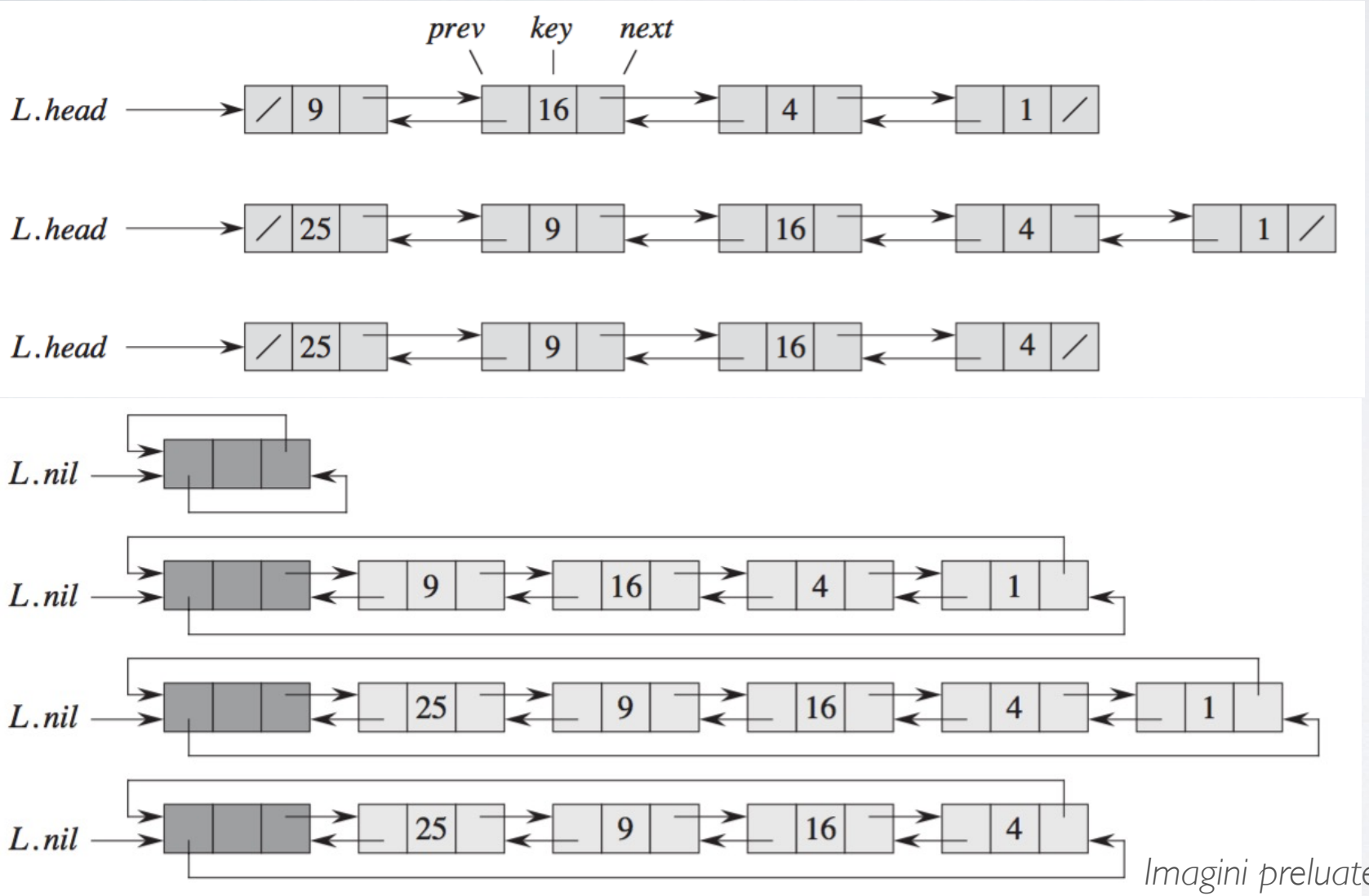

```
insert_first(L, 25)
```

```
delete(L, 1)
```

Imagini preluate din Th. Cormen – "Introduction to algorithms"

LISTA DUBLU INLANTUITA - FARA/CU SANTINELA

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA




```
delete(L,p)
   if p.prev != NIL
         p.prev.next = p.next
   else L.head = p.next
   if p.next != NIL
         p.next.prev = p.prev
```

```
delete_s(L,p)
   p.prev.next = p.next
   p.next.prev = p.prev
```

Imagini preluate din Th. Cormen – "Introduction to algorithms"

STIVA

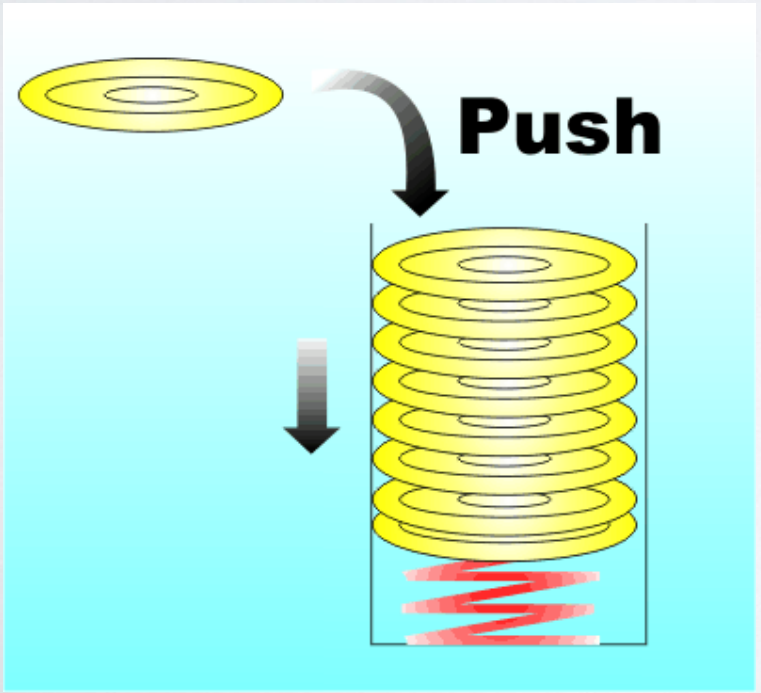
- o colectie de elemente cu politica de acces (inserare/stergere) de tip *LIFO* (*Last-In-First-Out*)
- elementele sunt inserate astfel incat, la orice moment, doar cel mai recent element inserat poate fi eliminat

STIVA
OPERATII FUNDAMENTALE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

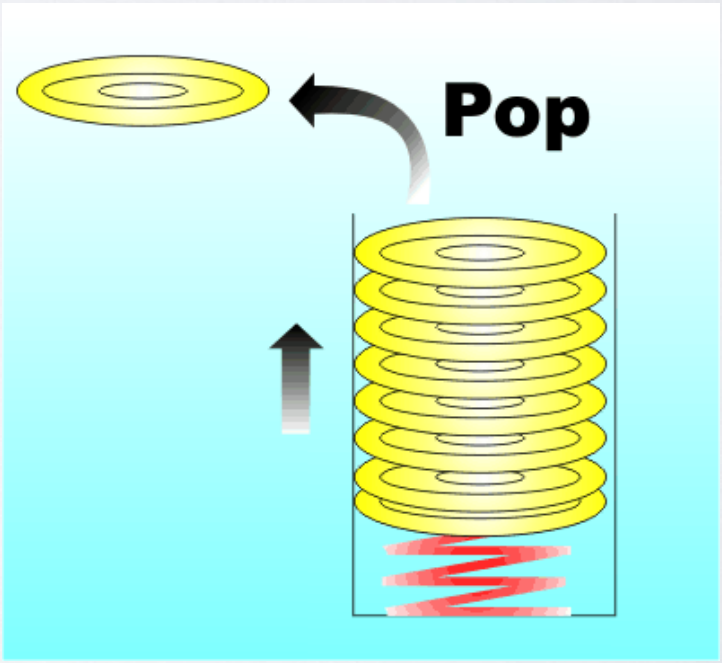
push(x): insereaza elementul *x in varful* stivei (en. *top, stack pointer*)

- Intrare: ElementStiva; Iesire: nimic



pop(): Sterge elementul aflat in varful stivei, si il returneaza; daca stiva e goala, mesaj de eroare

- Intrare: nimic; Iesire: ElementStiva



STIVA
OPERATII ADITIONALE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Utile dar nu fundamentale:
  - size(): returneaza numarul de elemente din stiva
    - Intrare: nimic; Iesire: intreg
  - isEmpty(): semnaleaza daca stiva este goala
    - Intrare: nimic; Iesire: boolean
  - top(): returneaza elementul din varful stivei, fara a-l sterge; daca stiva este goala, mesaj de eroare.
    - Intrare: nimic; Iesire: ElementStiva

STIVA: IMPLEMENTARE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Pseudocod:

```
PUSH(S,p)
   S.top = S.top + 1
  S.[S.top] = p

POP(S)
   if S.top ==0
            error "underflow"
   else S.top = S.top - 1
   return S[S.top +1]
```

- Folosim o *lista inlantuita*
  - Simplu sau dublu inlantuita?
  - De cate referinte avem nevoie pentru a implementa eficient operatiile? (i.e. first si/sau last)
- Folosim un *sir* (si mai ce?)
- Overflow?

Sursa: Th. Cormen – "Introduction to Algorithms"

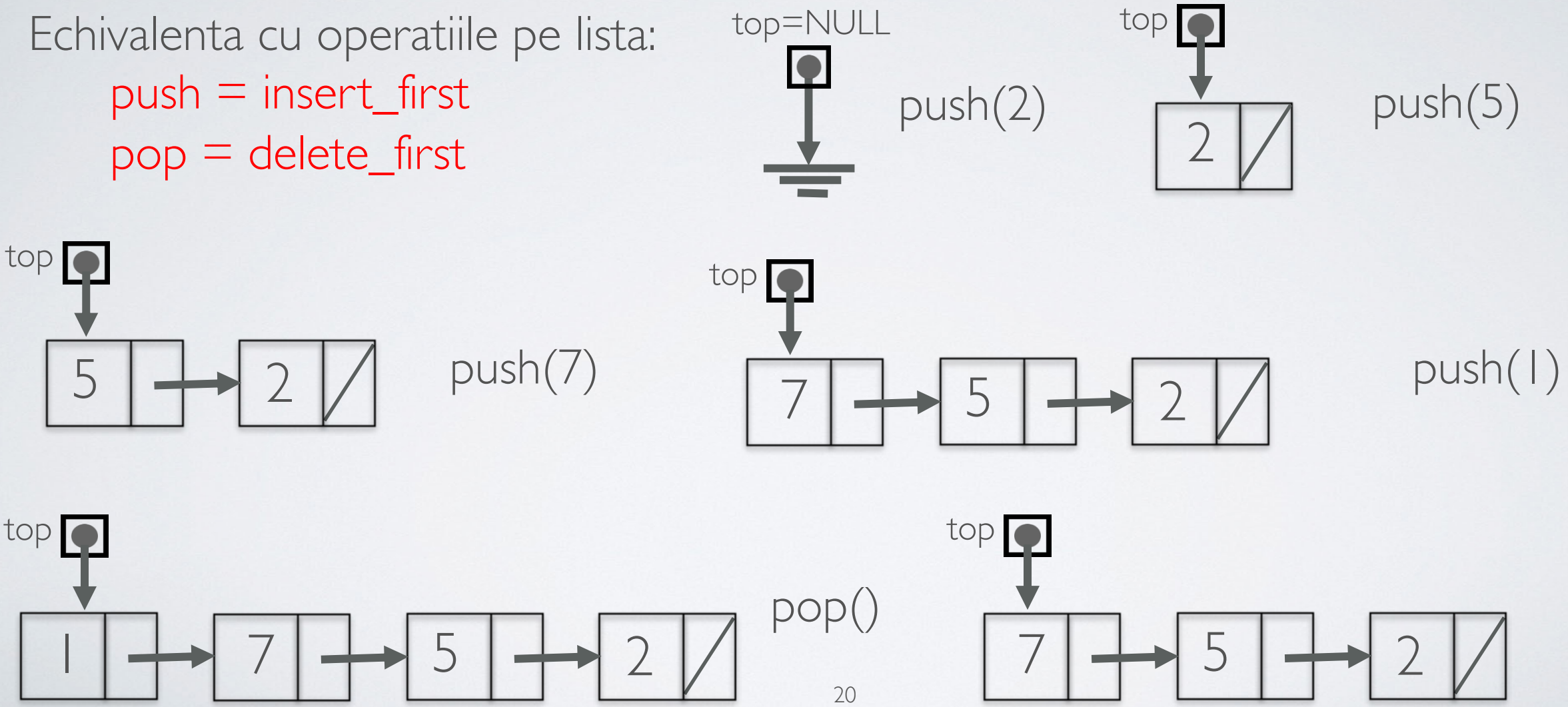
STIVA: IMPLEMENTARE CU LISTA SIMPLU INLANTUITA

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Echivalenta cu operatiile pe lista:

push = insert_first

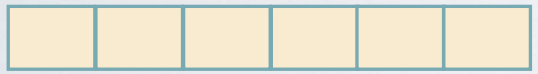
pop = delete_first



STIVA: IMPLEMENTARE CU SIR

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Definim un vector cu o capacitate data. Initial stiva e goala (i.e. Top = 0).

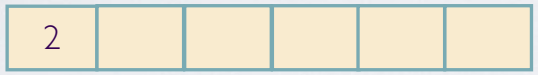


Top = 0;
Capacity = 6;

```
//push(x)
top ++;
stiva[top] = x;
```

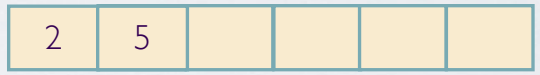
```
//pop
top--;
```

push(2)



Top = 1;
Capacity = 6;

push(5)



Top = 2;
Capacity = 6;

push(7)

Top = 3;
Capacity = 6;

push(1)

Top = 4
Capacity = 6;

pop()

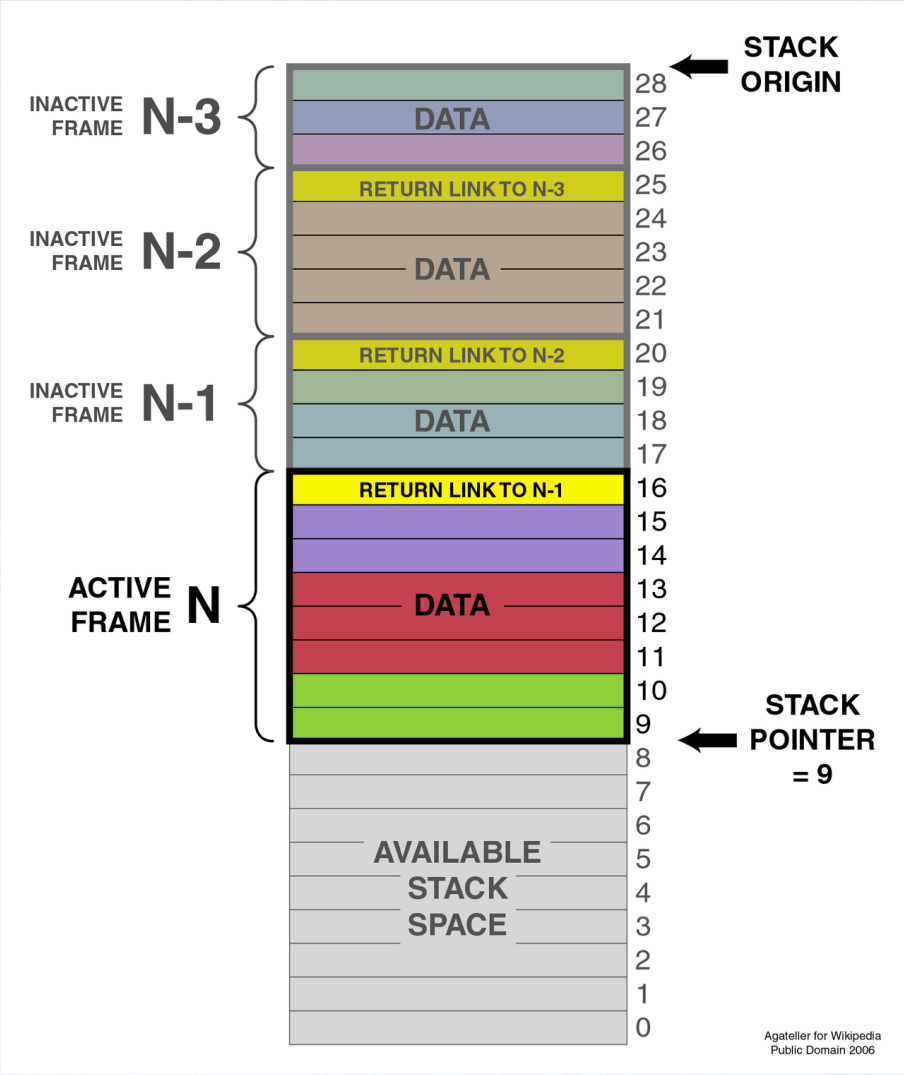
Top = 3;
Capacity = 6;

push(21)

Top = 4;
Capacity = 6;

STIVA - UTILITATE

- Stiva apare in programe: *call stack*
  - Structura de date care stocheaza informatia despre subrutinele active ale unui program.
  - Mecanism cheie in apelul/iesirea functii/proceduri
  - Formata din *stack frames*
- Recursivitatea - stiva
  - Apel: push *stack frame*
  - Iesire: pop *stack frame*
  - *Stack frame*
    - argumente functie
    - variabile locale
    - adresa de iesire
- Evaluarea expresiilor aritmetice/logice
- Natural Language Processing
  - *Parsing* sintactic



COADA

Colectie de elemente cu politica de acces (inserare/stergere) de tip *FIFO* (*First-In-First-Out*)

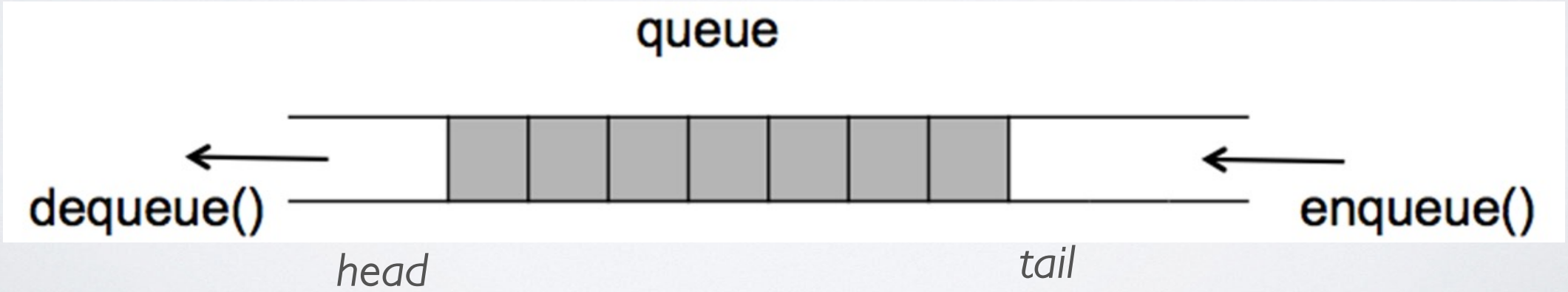
- Elementele sunt inserate astfel incat, la orice moment, elementul care a fost inserat la cel mai indepartat moment poate fi eliminat (cel mai *vechi*)
- Inserare: la ***sfarsit*** (tail, rear) **-** *"enqueue"*
- Stergere: de la ***inceput*** (head, front) **-** *"dequeue"*

https://www.tutorialspoint.com/data_structures_algorithms/images/queue_example.jpg

COADA: OPERATII FUNDAMENTALE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Operatii fundamentale:
  - enqueue(elem): Insereaza elementul *elem* la sfarsitul cozii
    - Intrare: ElementCoada; Iesire: nimic
  - dequeue(): Sterge elementul aflat la inceputul cozii si il returneaza; eroare daca nu exista elemente in coada
    - Intrare: nimic; Iesire: ElementCoada



http://www.cs.cmu.edu/~mrmiller/15-121/Homework/hw8/queue.png

COADA: OPERATII ADITIONALE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Operatii utile dar nu fundamentale:

- size(): returneaza numarul de elemente din coada
  - Intrare: nimic; Iesire: intreg
- isEmpty(): semnaleaza daca coada este goala
  - Intrare: nimic; Iesire: boolean
- front(): returneaza elementul din varful cozii, fara a-l sterge; daca coada este goala, mesaj de eroare.
  - Intrare: nimic; Iesire: ElementCoada

COADA: IMPLEMENTARE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Cum implementam o coada ?
  - Folosim o lista inlantuita
    - Simplu/dublu?
    - Cu first si last?
  - Cu sir

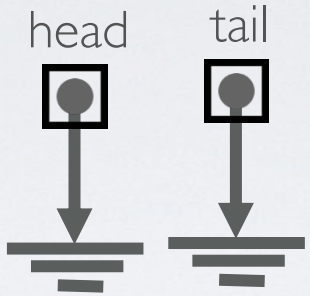
COADA: IMPLEMENTARE CU LISTE SIMPLU INLANTUITE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

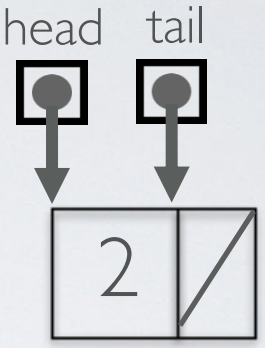
Echivalenta cu operatiile pe lista:

enqueue = insert_last

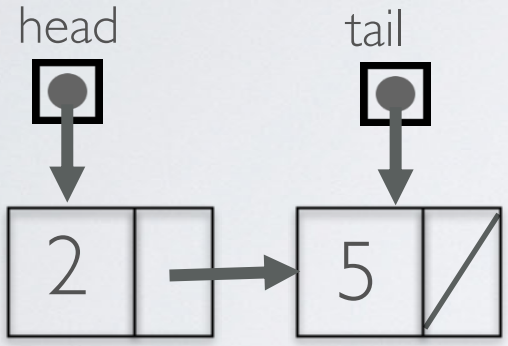
dequeue = delete_first



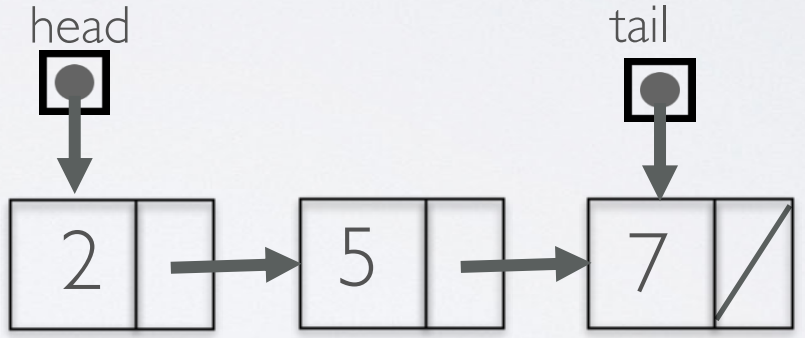
enqueue(2)



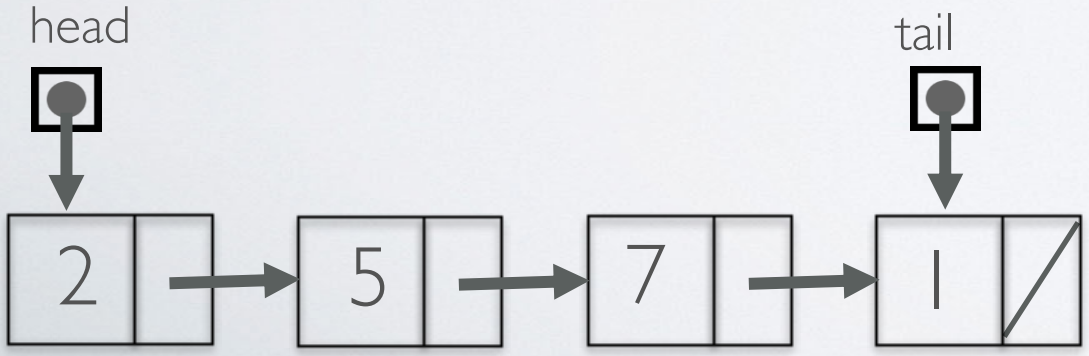
enqueue(5)



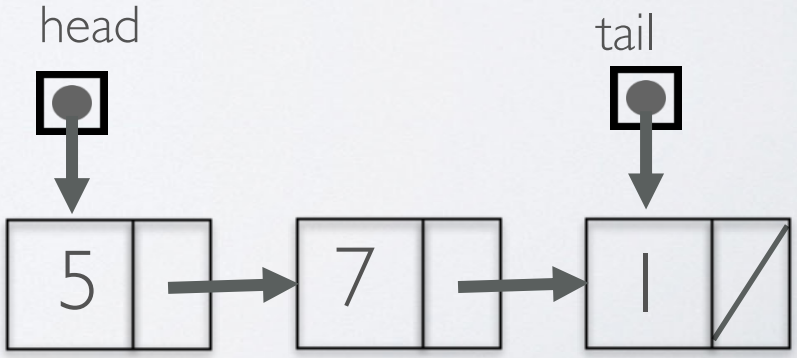
enqueue(7)



enqueue(1)



dequeue()



COADA: IMPLEMENTARE CU SIR

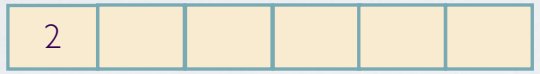
UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Definim un vector cu o capacitate data.

1 2 3 4 5 6

head= 0; tail= 0;
Size = 0;
Capacity = 6;

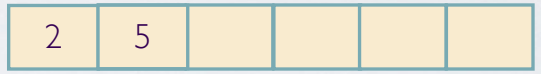
enqueue(2)



1 2 3 4 5 6

head= 1; tail= 1;
Size = 1;
Capacity = 6;

enqueue(5)



1 2 3 4 5 6

head= 1; tail= 2;
Size = 2;
Capacity = 6;

enqueue(7)

1 2 3 4 5 6

head= 1; tail= 3;
Size = 3;
Capacity = 6;

enqueue:

Pas 1 – Verificam daca coada este plina (size == capacity)

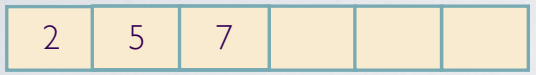
Pas 2 – Daca coada este plina trimitem mesaj de overflow si iesim.

Pas 3 – Daca coada nu este plina incrementam ultim ca sa "pointeze" la urmatorul spatiu liber (tail ++);

Pas 4 – Adaugam elementul pe pozitia lui ultim (coada[tail] = x;)

Pas 5 – Returnam success !

COADA: IMPLEMENTARE CU SIR



head= 1; tail= 3;
Size = 3;
Capacity = 6;

dequeue()

head= 2; tail= 3;
Size = 2;
Capacity = 6;

enqueue(1)

head= 2; tail= 4;
Size = 3;
Capacity = 6;

dequeue()

head= 3; tail= 4;
Size = 2;
Capacity = 6;

dequeue:
Pas1 – Verificam daca coada este goala(size == 0)
Pas 2 – Daca coada este goala trimitem mesaj de underflow si iesim.
Pas 3 – Daca coada nu este goala incrementam prim ca sa "pointeze" la urmatorul spatiu liber (head ++);
Pas 4 – Returnam success !

enqueue(20)

head= 3; tail= 5;
Size = 3;
Capacity = 6;

enqueue(17)

head= 3; tail= 6;
Size = 4;
Capacity = 6;

enqueue(30) – ce se intampla?

COADA CIRCULARA

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Structura liniara in care operatiile sunt efectuate conform principiului FIFO dar ultimul element este conectat la primul element si formeaza astfel o structura circulara.
- Poate fi implementata folosind:
  - Liste simplu/dublu inlantuite
  - Vectori
- Utilitate
  - *Round Robin* – algoritm de planificare pentru resursele CPU

COADA:
IMPLEMENTARE CU SIR – V2

head= 1; tail= 4;
Capacity = 6;

dequeue()

head= 2; tail= 4;
Capacity = 6;

enqueue(1)

head= 2; tail= 5;
Capacity = 6;

dequeue()

head= 3; tail= 5;
Capacity = 6;

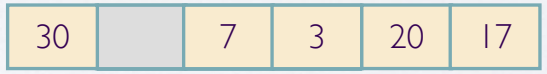
enqueue(20)

head= 3; tail= 6;
Capacity = 6;

enqueue(17)

head= 3; tail= 1;
Capacity = 6;

enqueue(30)



head= 3; tail= 2;
Capacity = 6;

head – adresa/pozitia primului element din coada
tail – adresa/pozitia unde ar trebui inserat urmatorul element la enqueue
size – nu e necesar; ce implica lipsa campului size?

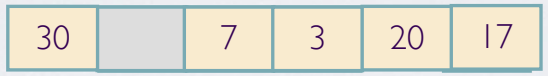
COADA *CIRCULARA*: IMPLEMENTARE CU SIR – V2

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Pseudocod:

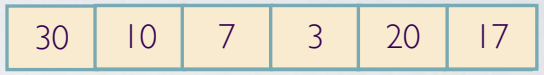
```
ENQUEUE(Q, p)
  Q[Q.tail] = p
  if Q.tail == Q.capacity
        Q.tail = 1
  else Q.tail = Q.tail + 1

DEQUEUE()
  p = Q[Q.head]
  if Q.head == Q.capacity
        Q.head = 1
  else Q.head = Q.head + 1
  return p
```

enqueue(30)



head= 3; tail= 2;
Capacity = 6;

enqueue(10)



head=3; tail= 3;
Capacity = 6;

Overflow??
Is this possible?

Codul nu trateaza overflow/underflow; Tema: fix it!

Sursa: Th. Cormen – "Introduction to Algorithms"

COADA – UTILITATE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Scheduler (e.g. in sistemul de operare), pentru mentinerea ordinii de acces la resursele partajate
  - Printer scheduling
  - CPU scheduling (coada circulara)
  - Disk scheduling
- Transfer asincron de date
  - Buffere I/O, *pipes*, citire/scriere in fisiere

BIBLIOGRAFIE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Th. Cormen et al – "Introduction to Algorithms", 3rd ed: sect. 10.1, 10.2

- https://www.doc.ic.ac.uk/~ar3/lectures/ProgrammingII/L3/Unit3Stacks&QueuesTwoUp.pdf

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

EXERCITII

TEORIE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

1. O literă înseamnă push iar un asterisc (*) reprezintă o operație pop() pentru o stivă. Se dă secvența de operații de mai jos realizate pe o stivă inițial vidă. Scrieți ce elemente vor fi returnate de operațiile pop().

E A S * Y * Q U E * * * S T * * * I O * N * * *

2. O literă înseamnă enqueue() iar un asterisc (*) reprezintă o operație dequeue() pentru o coadă. Se dă secvența de operații de mai jos realizate pe o coadă inițial vidă. Scrieți ce elemente vor fi returnate de operațiile dequeue().

E A S * Y * Q U E * * * S T * * * I O * N * * *

3. Considerați o stivă pe care se execută operații de push() și pop(). Operația pop() returnează și afișează valoarea nodului eliminat din stivă. Operația push(x) pune un nod cu valoarea x in stivă. Stim ca se executa in ordine operatii push(0), push(1), push(2) … push(9) dar intre aceste operatii pot aparea si apeluri de pop().
Scrieti care e secventa de operatii push(), pop() pentru fiecare din variantele de mai jos, si identificati daca exista vreo varianta care nu este valida (nu se poate afisa niciodata).

(a) 4 3 2 1 0 9 8 7 6 5
(b) 4 6 8 7 5 3 2 9 0 1
(c) 2 5 6 7 4 8 9 3 1 0
(d) 4 3 2 1 0 5 6 7 8 9

A. EXERCITII – LISTA SIMPLU INLANTUITA

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

1. Se citesc n cuvinte de la tastatura. Sa se numere frecventa de aparitie a fiecarui caracter si sa se stocheze caracterele intr-o lista **simplu inlantuita** – vezi cursul 1. Sa se implementeze urmatoarele:

- operatia insertLast, insertFirst (curs1), deleteFirst, deleteLast
- afisareLista (curs1)
- Se citeste un cuvant de la tastatura; Sa se caute in lista fiecare caracter din cuvantul citit si sa se decrementeze frecventa de aparitie a acestuia. Daca frecventa este 0 sa se stearga nodul respectiv.

A. EXERCITII – LISTA DUBLU INLANTUITA

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

2. Se citesc n cuvinte de la tastatura. Sa se numere frecventa de aparitie a fiecarui caracter si sa se stocheze caracterele intr-o lista **dublu inlantuita**. Sa se implementeze urmatoarele:

- operatia insertLast, insertFirst (curs 1), deleteFirst, deleteLast
- afisareLista (afisare incepand cu primul element, afisare incepand cu ultimul element)
- Se citeste un cuvant de la tastatura; Sa se caute in lista fiecare caracter din cuvantul citit si sa se decrementeze frecventa de aparitie a acestuia. Daca frecventa este 0 sa se stearga nodul respectiv.
- Folositi laboratorul 3 pentru rezolvare !

A. EXERCITII – LISTA DUBLU INLANTUITA

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

3. Se citesc n numere intregi. Stocati aceste numere intr-o listă dublu înlanțuită sortată crescător. Să se implementeze urmatoarele:

- operatia insertSorted(DList *d, int x) --> inserează valoarea x în lista dublu înlănțuită astfel încât după inserare lista rămâne ordonată crescător.

- afisareLista (afisare incepand cu primul element, afisare incepand cu ultimul element)

4. Se dă o listă dublu înlănțuită ordonată crescător (considerați lista de la exercițiul 3).
Găsiți perechi de noduri din listă a căror sumă a valorilor este egala cu s, unde s se citește.
Exemplu: 1(head) <-> 2 <-> 4 <-> 5 <-> 6 <-> 8 <-> 9 (tail)

X = 7 Output: (6, 1), (5,2)

A. EXERCITII – STIVA SI COADA IMPLEMENTARE CU LISTE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

5. Se citeste un sir de n numere intregi de la tastatura.

- Folosind functile implementate pe lista (insert, delete) sa se insereze numerele citite intr-o structura de tip **stiva**.
- Afisati continutul cozii dupa fiecare inserare.
- Stergeti primele 2 elemente din stiva.

6. Se citeste un sir de n numere intregi de la tastatura.

- Folosind functile implementate pe lista (insert, delete) sa se insereze numerele citite intr-o structura de tip **coada**.

A. EXERCITII – STIVA, COADA IMPLEMENTATA CU VECTORI

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

I1. Se citesc n numere reale. Stocati aceste numere intr-o structura de tip **stiva implementata folosind vectori.**

- Operatii: init, push, pop

I2. Se citesc n numere reale. Stocati aceste numere intr-o structura de tip **coada implementata folosind vectori.**

- Operatii: init, enqueue(), dequeue()

A. EXERCITII – LISTA CIRCULARA

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

13. Se citeste un sir de n numere intregi. Implementati urmatoarele operatii pe o lista simplu inlantuita circulara:

- Creare lista;
- Inserati primele n/2 numere ca pe prima pozitie in lista circulara;
- Inserati restul elementelor pe ultima pozitie in lista circulara;
- Cititi un numar si implementati o fuctie de cautare a acestuia.Returnati nodul care contine numarul.
- Daca numarul citit apare in lista, stergeti nodul care contine numarul citit.

14. Se da o lista simplu inlantuita circulara. Implementati o functie care imparte lista in jumatate (genereaza doua liste simplu inlantuite circulare, prima avand prima jumatate din elementele listei originale, a doua avand restul elementelor) – sursa

15. Implementati o coada circulara, folosind o lista dublu inlantuita (operatii: init, enqueue, dequeue)

A. EXERCITII - COMPLEXITATE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

16. Analizati complexitatea operatiilor implementate la probleme