# Cursul nr. 4

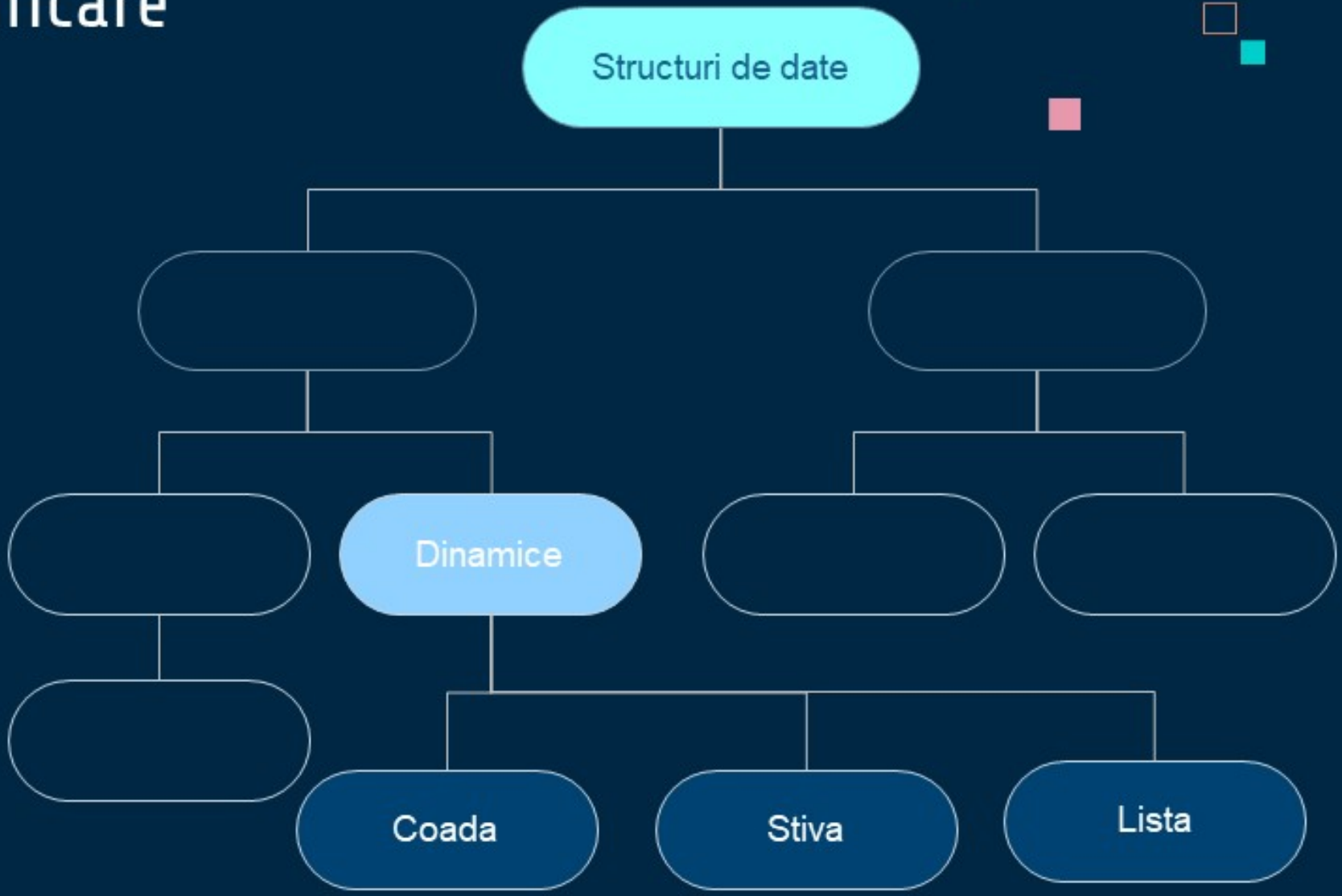
# STRUCTURI DE DATE

## dinamice

## Liste simplu, dublu inlantuit, circulare

Lector dr. Dorin IORDACHE

# Clasificare



# Agenda

01

Lista simplu inlantuit

02

Lista dublu inlantuit

03

Lista circulara

# Tipuri abstracte de date

Tip abstract de date = set de date asupra căruia se pot efectua operații specifice

Nivel: utilizare

Exemple: lista (list), stiva (stack), coada (queue), movila (heap), dicționar (dictionary), tabel de dispersie (hash table), arbori de căutare

Reprezentare abstractă = modalitate de organizare a elementelor setului de date care permite efectuarea eficientă a operațiilor specifice

Nivel: proiectare

Exemple: structură liniară, structură arborescentă

Implementare = specificarea structurii de date folosind tipuri de date și construcții specifice unui limbaj de programare

Nivel: implementare

Exemple: tablouri (arrays), liste înlănțuite (linked lists)

# Lista simplu inlantuit

01

# Tip abstract de date: lista

Tip abstract de date = set de date asupra căruia se pot efectua operații specifice Nivel:

utilizare

Exemple: lista (list), stiva (stack), coada (queue), movila (heap), dicționar (dictionary), tabel de dispersie (hash table), arbori de căutare

Reprezentare abstractă = modalitate de organizare a elementelor setului de date care permite efectuarea eficientă a operațiilor specifice

Nivel: proiectare

Exemple: structură liniară, structură arborescentă

Implementare = specificarea structurii de date folosind tipuri de date și construcții specifice unui limbaj de programare

Nivel: implementare

Exemple: tablouri (arrays), liste înlănțuite (linked lists)

# Lista simplu inlantuit

Tip abstract de date = set de date asupra căruia se pot efectua operații specifice

Nivel: utilizare Exemplu: lista

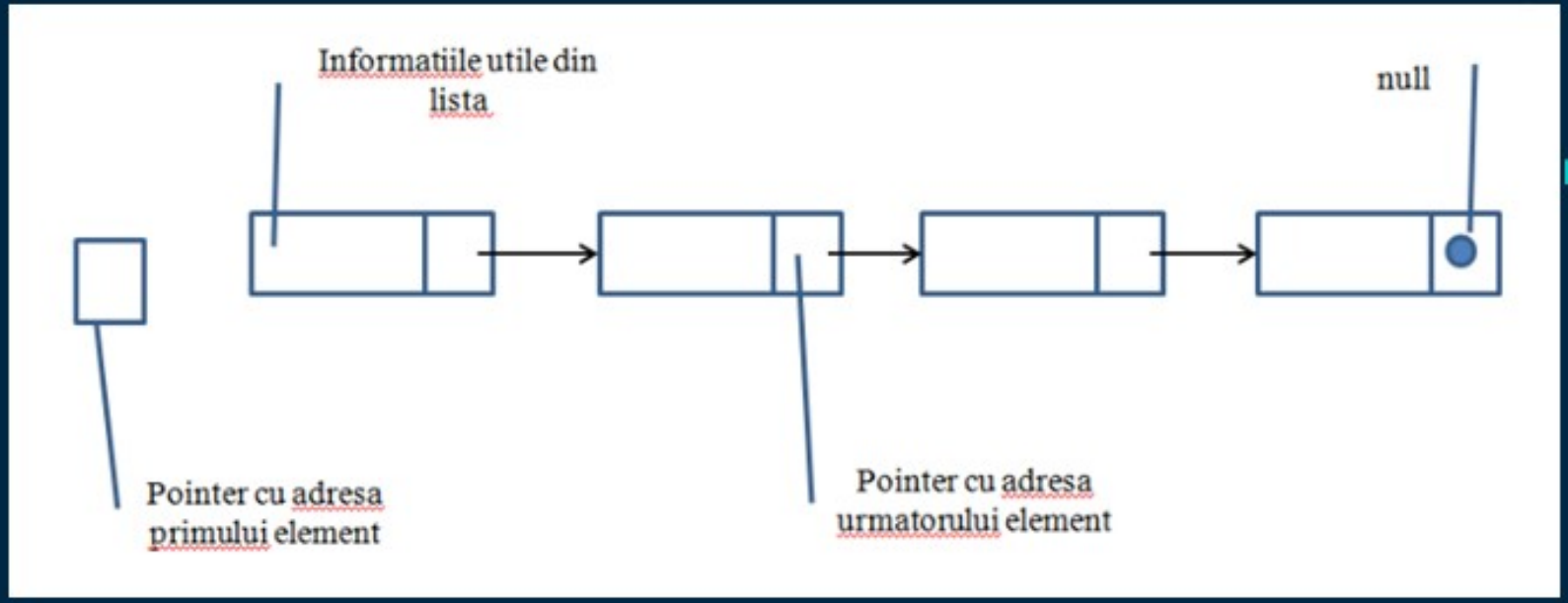
Operații specifice listelor: **interogare = căutarea unui element**

- După poziție
  - Elementul aflat pe o anumită poziție
  - Elementul următor/ anterior unui element specificat (element curent)
- După valoare
  - Elementul care conține o valoare specificată
  - Elementul care conține cea mai mică/ mare valoare
  - Elementul care conține o valoare specificată prin poziția relativă în raport cu alte valori (ex: al treilea element în ordine crescătoare, elementul median etc)

Caz particular: parcurgerea tuturor elementelor într-o anumită ordine

# Lista simplu inlantuit este o structura de date dinamice.

lista nu este alocata ca bloc omogen de memorie, ci ca elemente separate de memorie

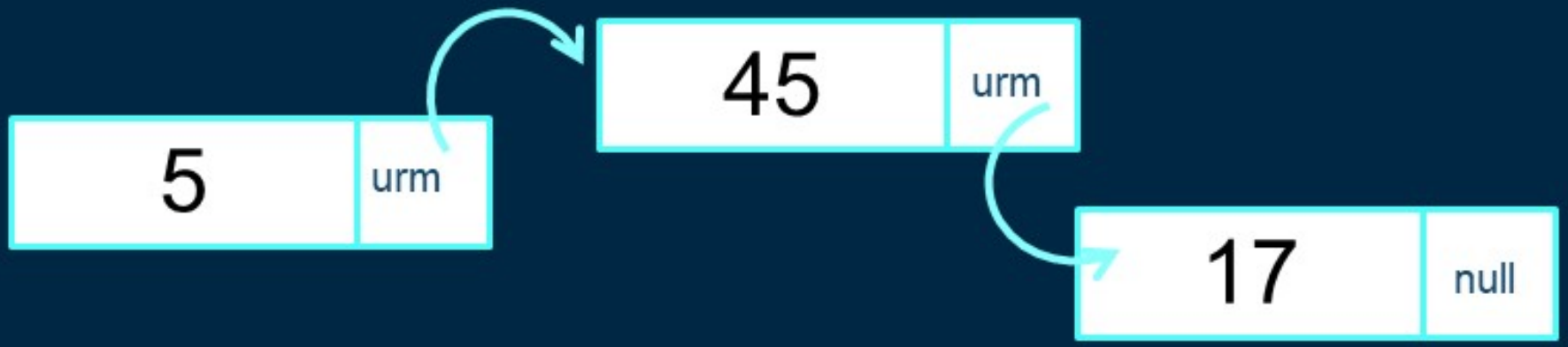
# Operatii

- Afisare

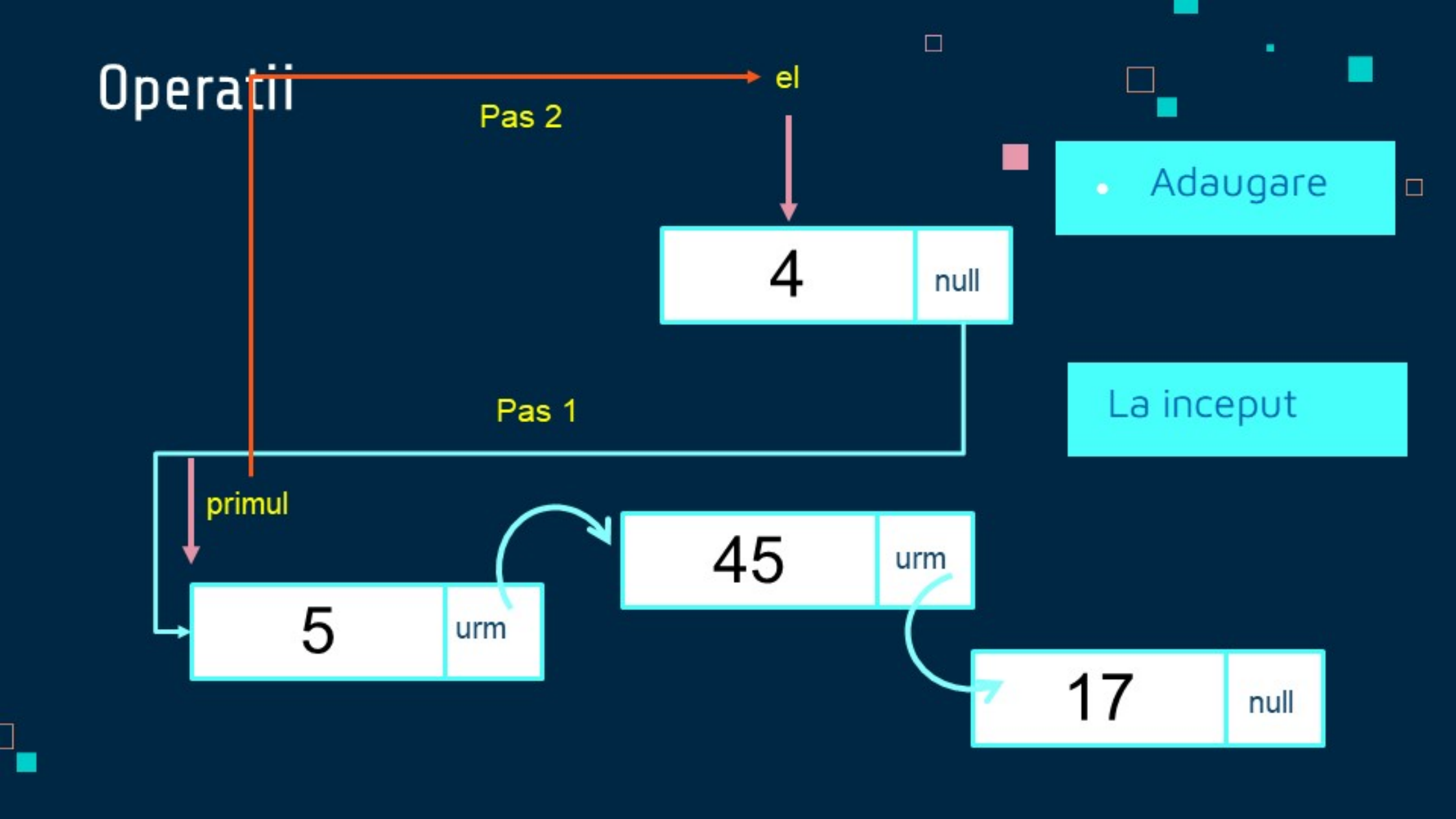
```
// Structura unui element  dintr-o lista simplu inlantuit
struct Nod {
  // datele efective memorate
  char nume[100];
  int an;
  // legatura catre nodul urmator
  Nod * urm;
};
```

# Operatii

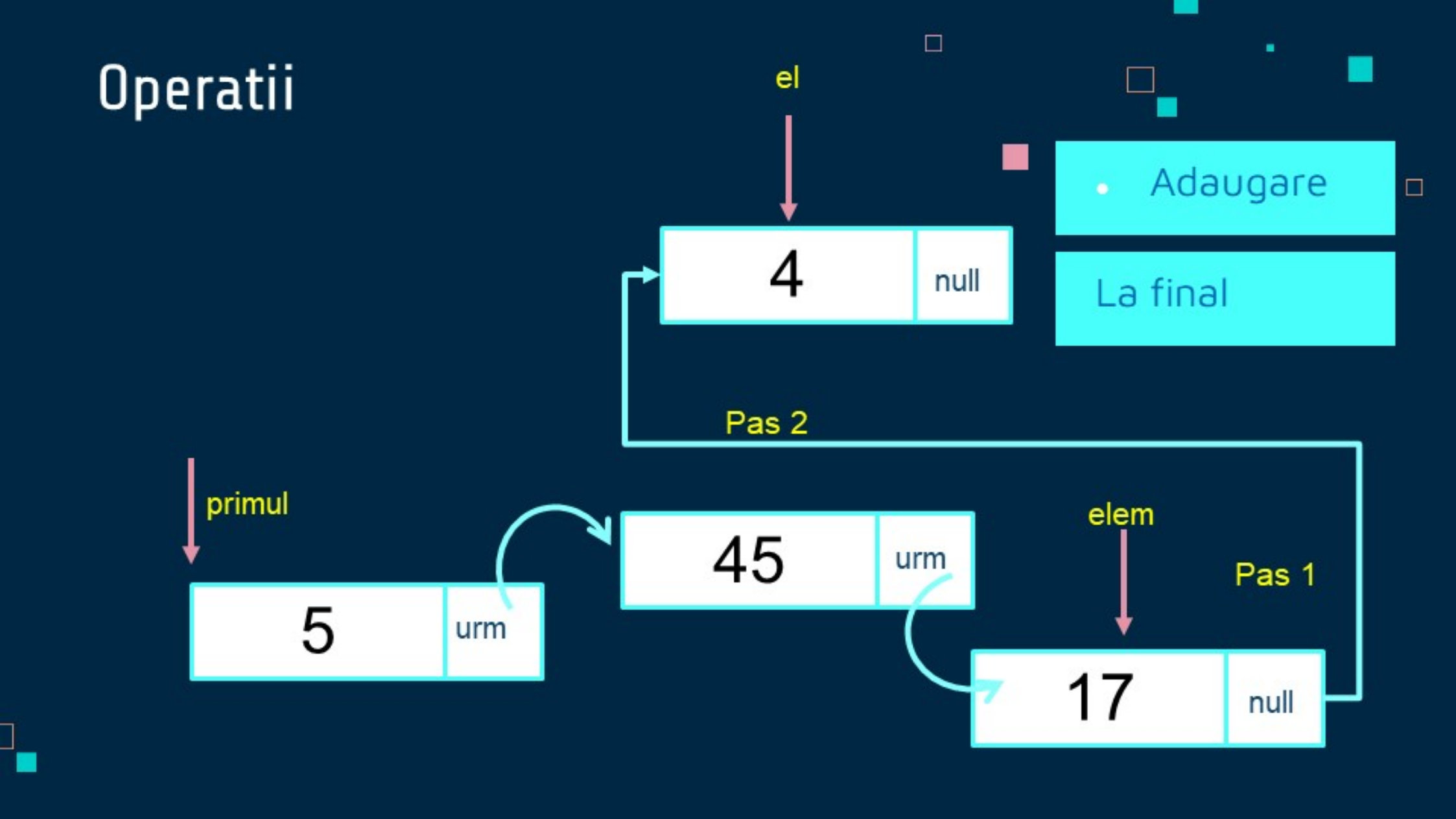
- Adaugare
- Inserare



# Operatii

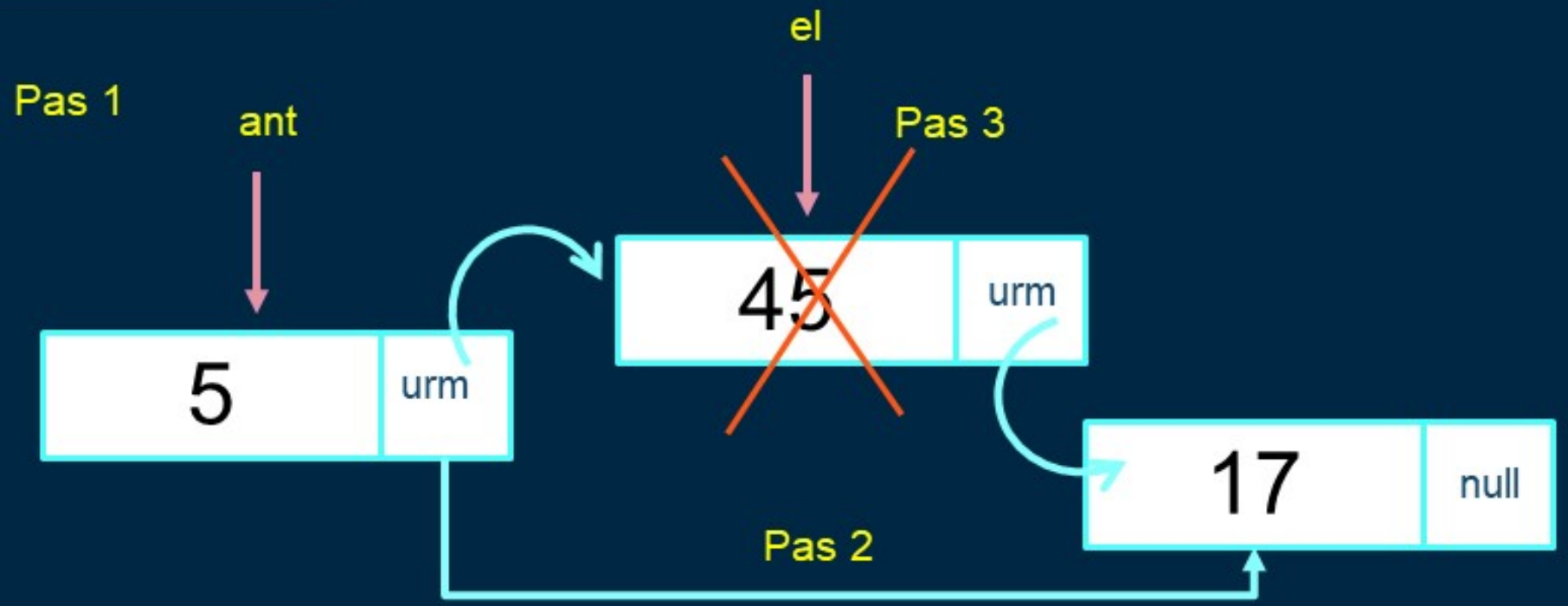

el
Pas 2
Adaugare
4
null
La inceput
Pas 1
primul
45
urm
5
urm
17
null

# Operatii


el
4
null
Adaugare
La final
Pas 2
primul
5
urm
45
urm
elem
Pas 1
17
null

# Operatii

- Stergere

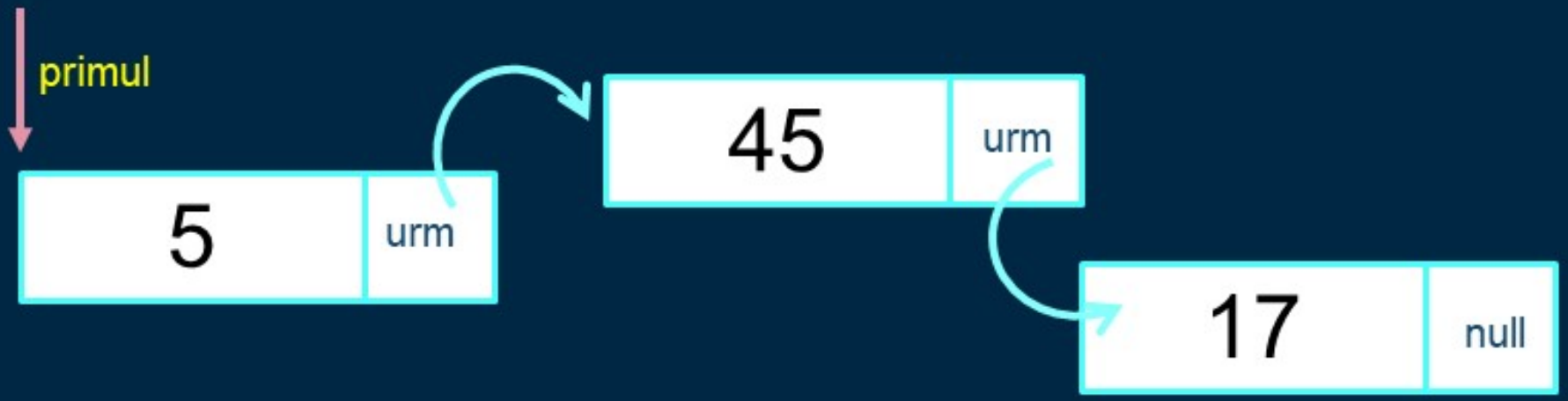


# Operatii

- Cautare

Parcurgere lista, de la primul nod pana la ultimul si comparam valoarea din nod cu valoarea de cautat.

**Cautare 45**
**Cautare 77**



# Aplicatii lista simplu inlantuit

- implementare stive și cozi, fundamentale în domeniul informaticii
- prevenirea coliziunilor dintre datele din harta hash
- efectuarea de funcții de anulare, refacere sau ștergere
- redare fotografii într-o prezentare de diapositive
- rezolvare cereri (liste, cozi cu sau fara prioritate)
- Implelemntare grafuri (matrice de adiacenta)
- alocare dinamica de memorie ( lista pentru blocurile de memorie libere)
- mentinere lista de nume ( DNS)
- operatii artimetice cu numere mari
- lucru cu polinoame
- lucru cu matrici rare

# Lista dublu inlantuit

02

# Lista dublu inlantuit

Tip abstract de date = set de date asupra căruia se pot efectua operații specifice

Nivel: utilizare Exemplu: lista

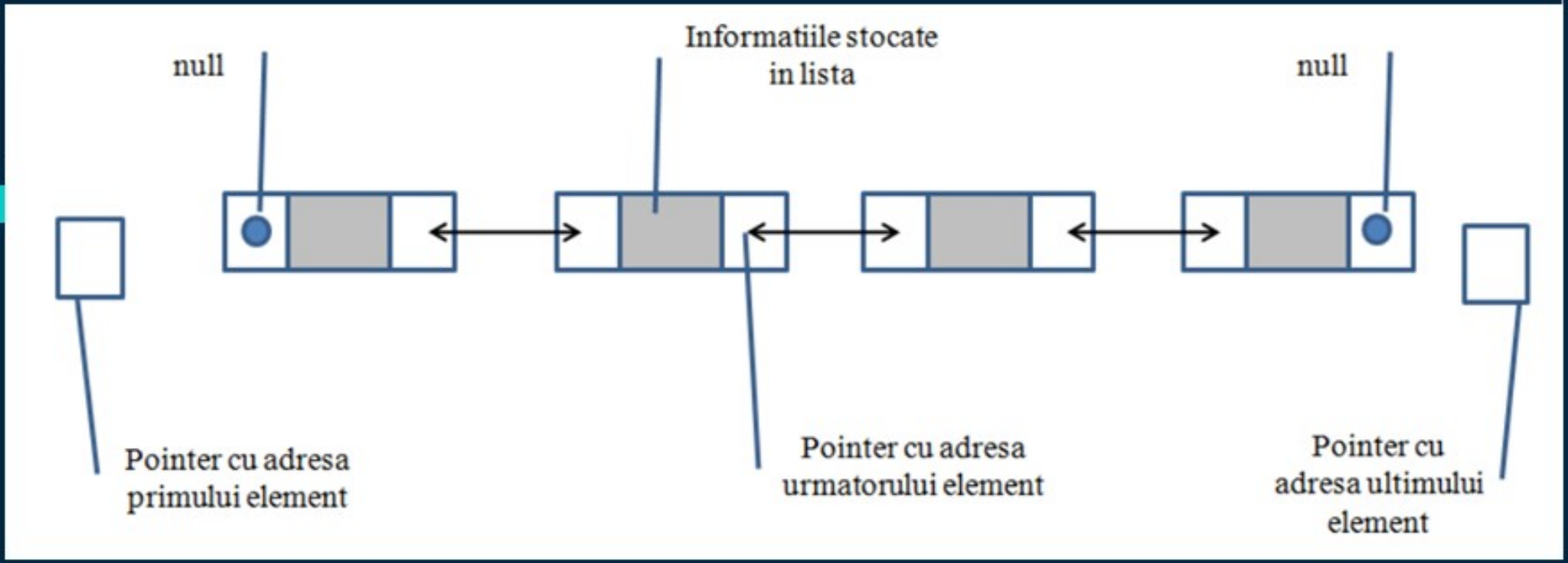
Operații specifice listelor: **interogare = căutarea unui element**

- După poziție
  - Elementul aflat pe o anumită poziție
  - Elementul următor/ anterior unui element specificat (element curent)
- După valoare
  - Elementul care conține o valoare specificată
  - Elementul care conține cea mai mică/ mare valoare
  - Elementul care conține o valoare specificată prin poziția relativă în raport cu alte valori (ex: al treilea element în ordine crescătoare, elementul median etc)

Caz particular: parcurgerea tuturor elementelor într-o anumită ordine

# Lista dublu inlantuit este o structura de date dinamice.

lista nu este alocata ca bloc omogen de memorie, ci ca elemente separate de memorie

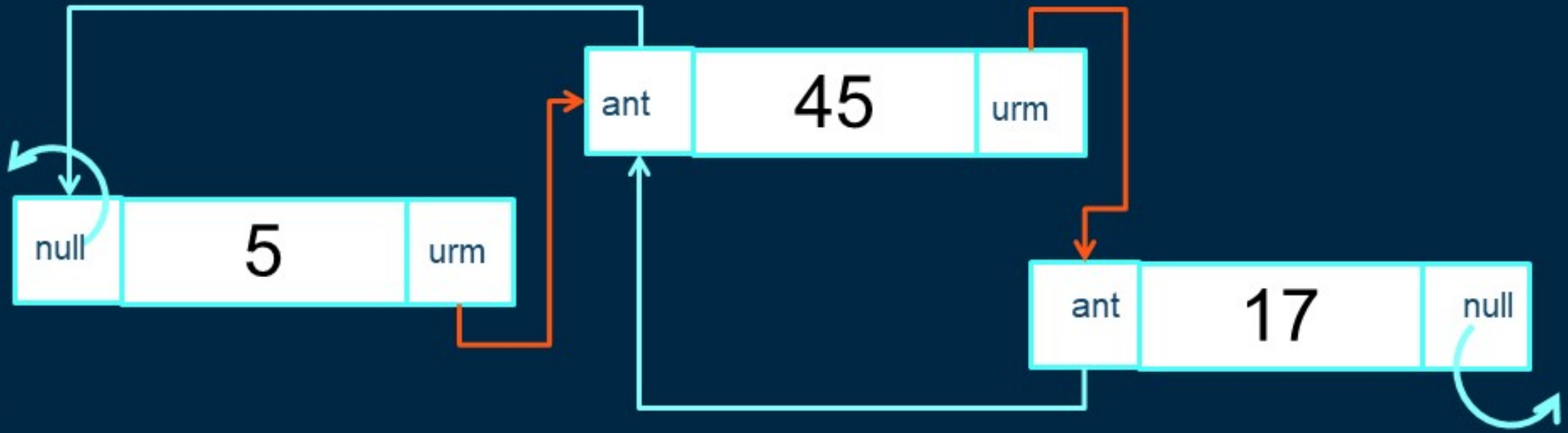
# Operatii

- Afisare

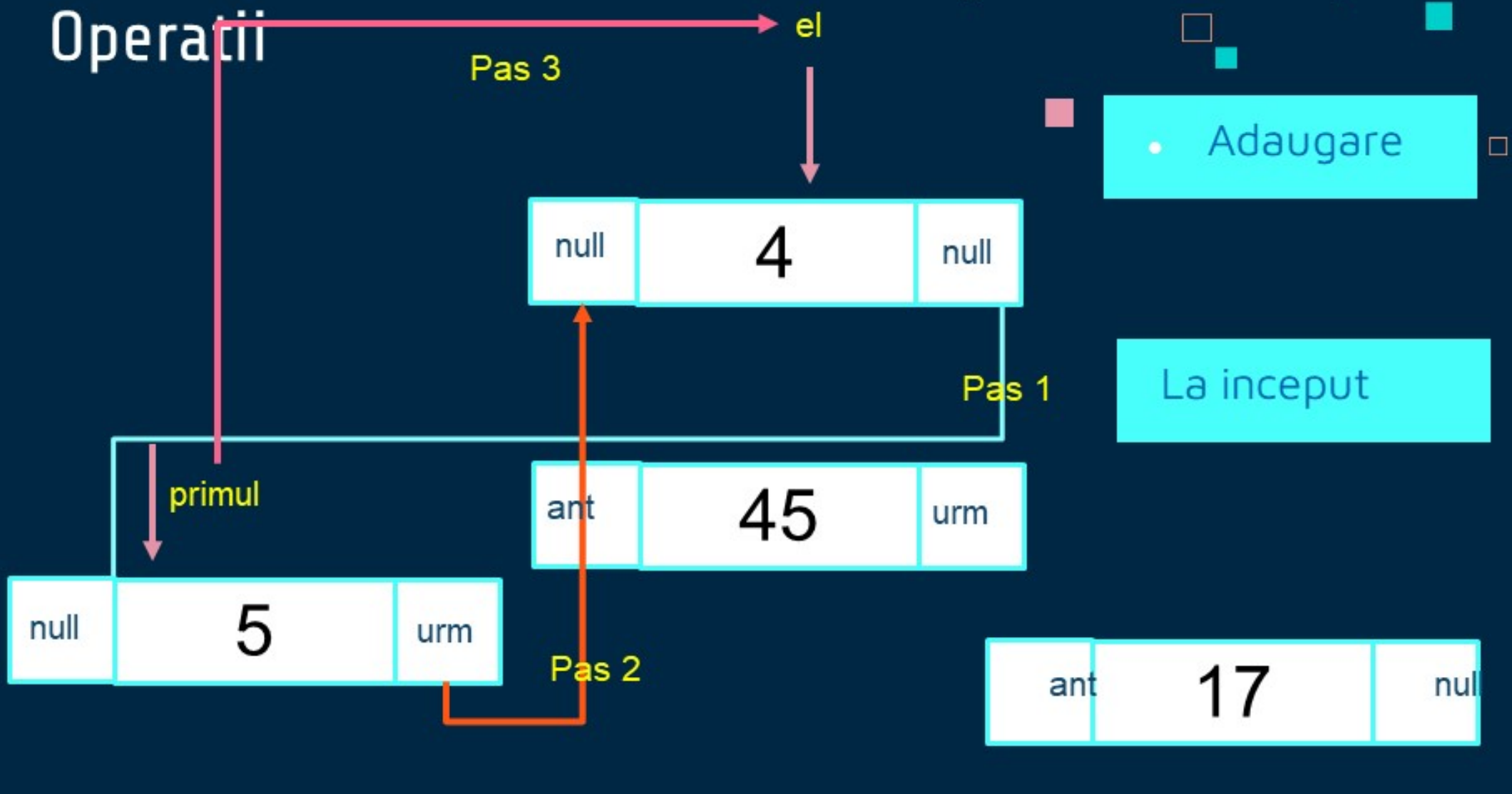
```
// Structura unui element  dintr-o lista dublu inlantuit
struct Nod {
  // datele efective memorate
   char nume[100];
   int an;
   // legatura catre nodrile anterior si urmator
   Nod * ant=NULL, * urm=NULL;
};
```

# Operatii

- Adaugare
- Inserare



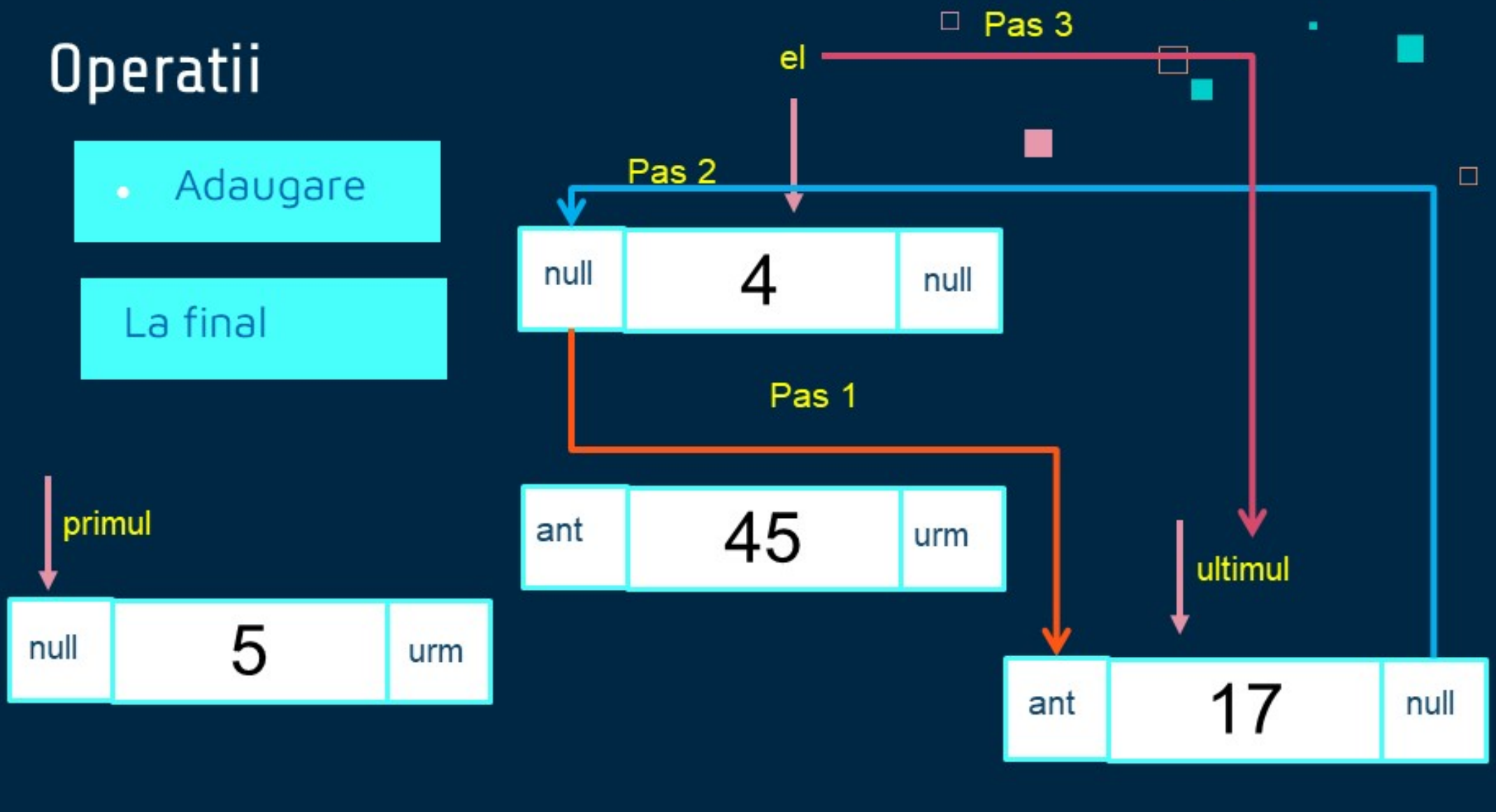
# Operatii


el
Pas 3
null
4
null
Adaugare
Pas 1
La inceput
primul
ant
45
urm
null
5
urm
Pas 2
ant
17
null

# Operatii

- Adaugare

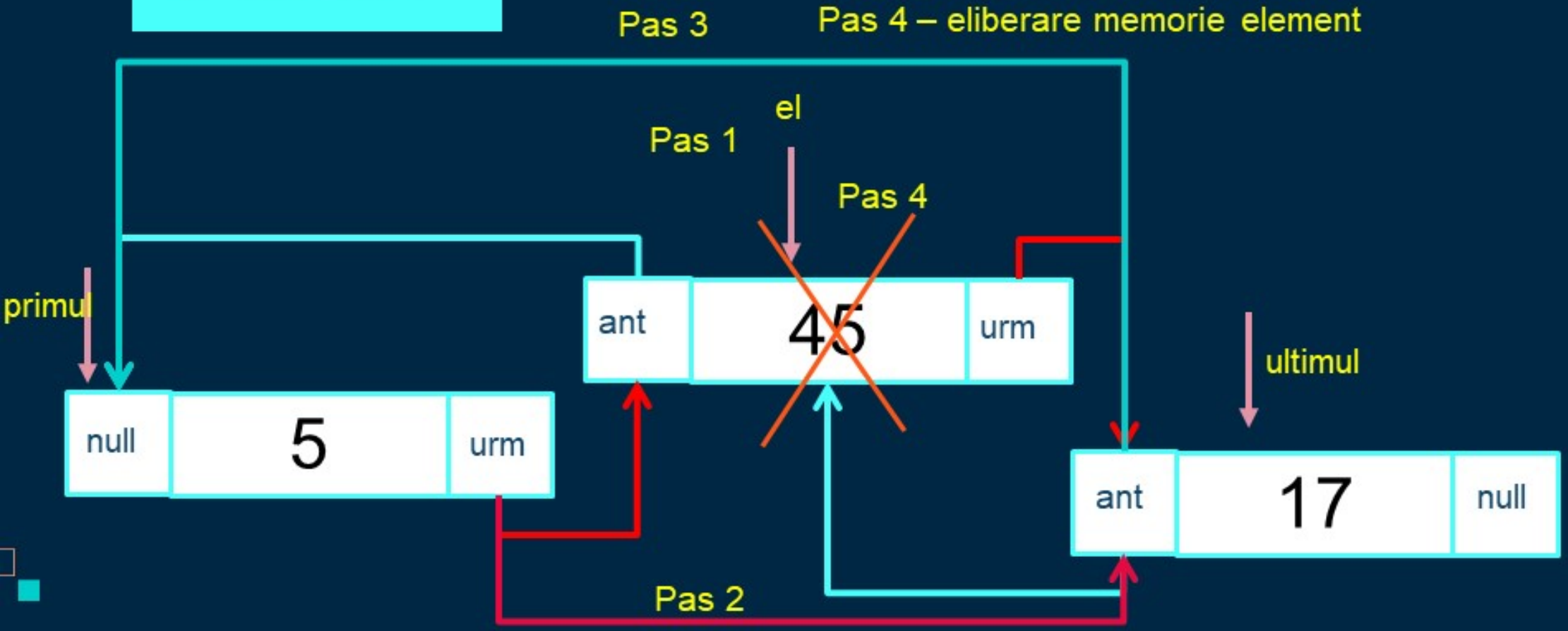
La final



# Operatii

- Stergere

Pas 1 – cautare element de sters
Pas 2 – legatura element anterior catre element urmator
Pas 3 – legatura element urmator catre element anterior
Pas 4 – eliberare memorie element

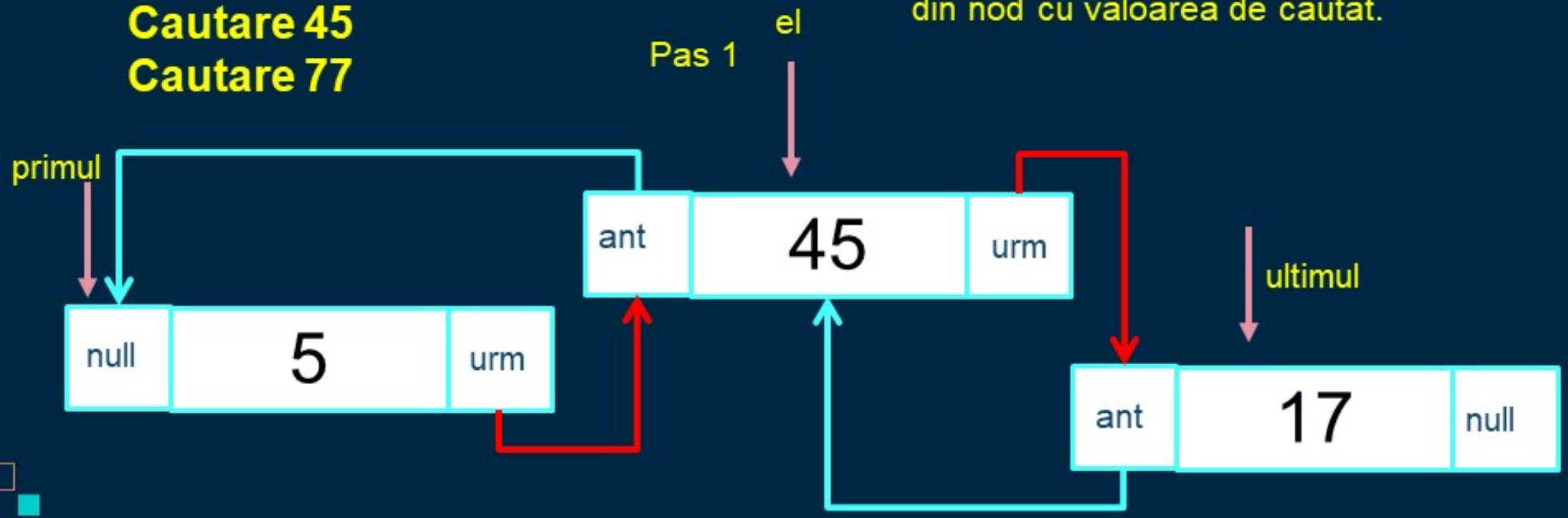


# Operatii

- Cautare

Parcurgere lista, de la primul nod pana la ultimul si comparam valoarea din nod cu valoarea de cautat.
Parcurgere lista, de la ultimul nod pana la primul si comparam valoarea din nod cu valoarea de cautat.

**Cautare 45**
**Cautare 77**



# Aplicatii lista dublu inlantuit

- în sistemele de navigație, deoarece necesită navigare față și spate (redo si undo)
- in browser, când vrem să folosim funcția înapoi sau următorul pentru a schimba fila
- implementarea altor structuri de date, cum ar fi un arbore binar, tabele hash, stivă etc.
- redare a muzicii
- In sisteme de operare, planificatorul de fire de executie - multitasking
- pachet de cărți de joc

# Lista circulara

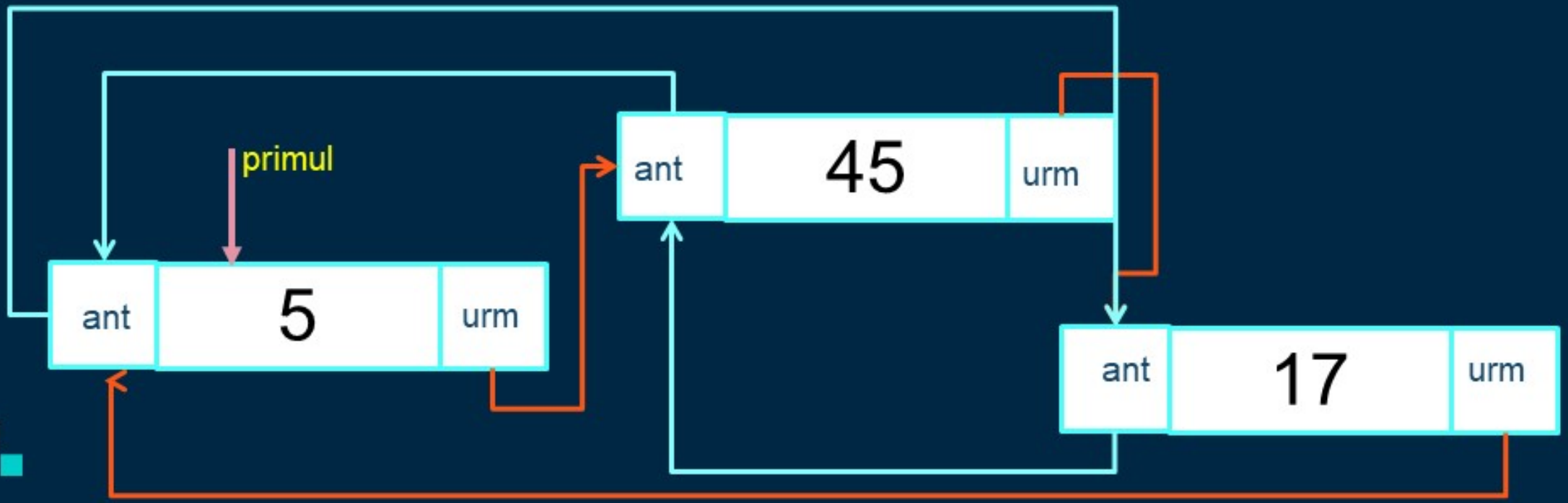
03

# Lista circulara

- Simplu inlantuit

# Lista circulara

- Dublu inlantuit



# Aplicatii lista circulara

- orice actiuni repetitive, dupa un numar de pasi ( repetare circulara)
- cozi prin menținerea unui pointer către ultimul nod inserat
- implementarea structurilor avansate de date precum Fibonacci Heap.
- In sistemul de operare pentru a partaja timpul pentru diferiți utilizatori, partajare a timpului Round-Robin ( în principal pentru multitasking).
- jocurile multiplayer folosesc o listă circulară pentru a schimba jucătorii într-o buclă
- in Photoshop, Word sau orice programe de desenare/editare în funcția de anulare

# Implementare – simplu inlantuit

```
struct Nod {
        int Data;
        Struct Nod* next;
};
```

**Adaugare nod**

```
void insert(struct Nod* prim, int data){
struct Nod* newNod= (struct Nod) malloc(sizeof(struct Nod);

   newNod>data = data;
   newNod>next = *prim;

   *prim = newNod;
}
```

# Implementare

**Stergere nod**

```
void deletePrim(struct Nod** prim){
    struct Nod* temp = *prim;

    if(*prim == NULL){
        printf("Lista nu are noduri!");
        return;
    }
    *prim = (*prim)->next;
    free(temp);
}
```

**Afisare continut lista**

```
void afisare(struct Nod* nod)
{
    printf("Lista: ");

    while(nod!=NULL){
        printf("%d ",nod->data);
        nod = nod->next;
    }

    printf("\n");
}
```

# Implementare

Cod sursa complet

```c
#include<stdlib.h>
#include<stdio.h>
struct Nod{
    int data;
    struct Nod *next;
};
void deletePrim(struct Nod** prim){
void insert(struct Nod** prim, int data){
void afisare(struct Nod* Nod){
    printf("\nLista: ");
    while(Nod!=NULL){
        printf("%d ",Nod->data);
        Nod = Nod->next;
    }
    printf("\n");
}
int main()
```

```c
int main()
{
    struct Nod* prim = NULL;
    insert(&prim,100);
    insert(&prim,80);
    insert(&prim,60);
    insert(&prim,40);
    insert(&prim,20);
    afisare(prim);
    deletePrim(&prim);
    deletePrim(&prim);
    display(prim);

    return 0;
}
```

# Implementare – dublu inlantuit

```
struct Nod {
    int data;
    struct Nod *urm;
    struct Nod *ant;
};
```

**Adaugare nod**

```
void insertPrim(struct Nod** prim, int data){

    struct Nod* newNod = (struct Nod*) malloc(sizeof(struct Nod));

    newNod->data = data;
    newNod->urm = *prim;
    newNod->ant = NULL;
    if(*prim != NULL)
        (*prim)->ant = newNod;
    *prim = newNod;
}
```

# Implementare – dublu inlantuit

**Adaugare nod la final cu parcurgere**

```
void insertFinal(struct Nod** prim, int data){
    struct Nod* nodNou = (struct Nod*) malloc(sizeof(struct Nod));

    nodNou->data = data;
    nodNou->urm = NULL;
    nodNou->ant = NULL;

    if(*prim==NULL){
        *prim = nodNou;
        nodNou->urm = NULL;
        return;
    }
    struct Nod* temp = *prim;

    while(temp->next!=NULL)
        temp = temp->urm;
    temp->next = nodNou;
    nodNou->ant = temp;
}
```

# Implementare – dublu inlantuit

**Adaugare nod la final fara parcurgere**

```
void insertFinal(struct Nod** prim, int data){
    struct Nod* nodNou = (struct Nod*) malloc(sizeof(struct Nod));

    nodNou->data = data;
    nodNou->urm = NULL;
    nodNou->ant = NULL;

    if(*prim==NULL){
        *prim = nodNou;
        nodNou->urm = NULL;
        return;
    }

    ultim->next = nodNou;
    nodNou->ant = ultim;
    ultim = nodNou;
}
```

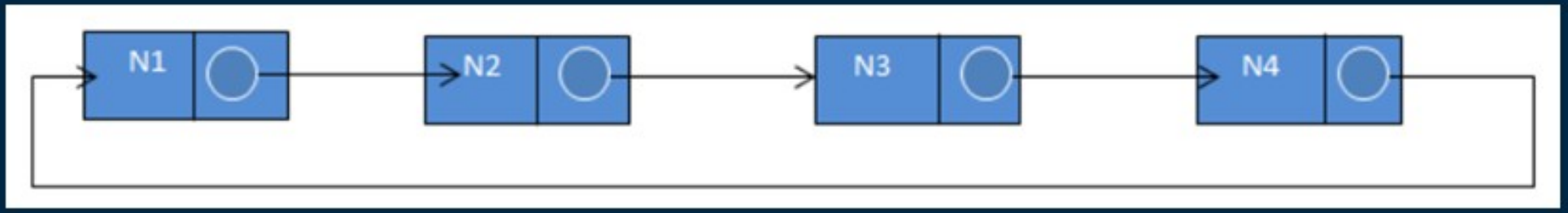
# Implementare – dublu inlantuit

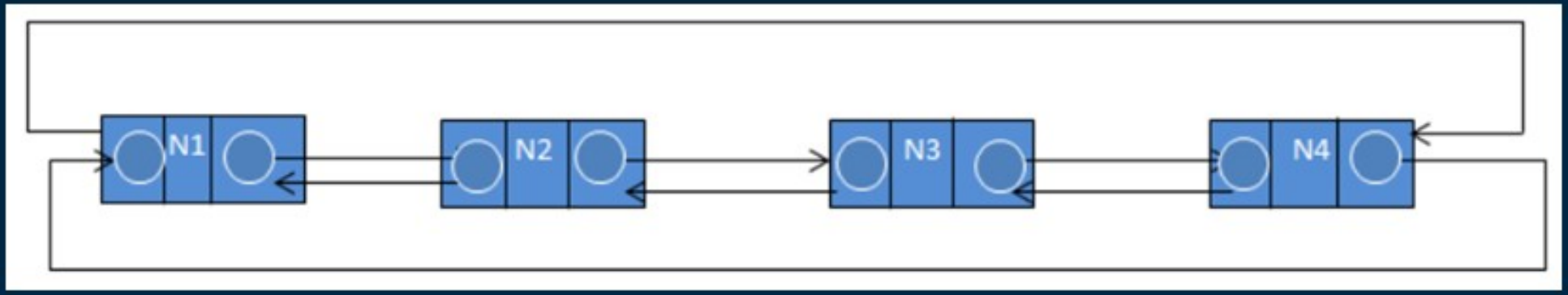
**Afisare continut lista**

```c
void afisare(struct Nod* nod)
{
    struct Nod* ultim;
    printf("\nParcurgere in sens direct\n");
    while (nod != NULL) {
        printf(" %d ", nod->data);
        ultim = nod;
        nod = nod->urm;
    }

    printf("\nParcurgere in sens invers \n");
    while (ultim != NULL) {
        printf(" %d ", final->data);
        final = final->ant;
    }
}
```

```c
int main()
{
    struct Nod* prim = NULL;

    insertPrim(&prim,1);
    insertPrim(&prim,2);
    insertPrim(&prim,3);

    afisare(prim);

    return 0;
}
```

# Implementare – lista circulara



Intrebari?

dorin.lordache@365.univ-ovidius.ro

# Mulțumesc