Algorytmy i struktury danych

Listy
Część I

Aleksander Lamża
ZKSB · Instytut Informatyki
Uniwersytet Śląski w Katowicach

aleksander.lamza@us.edu.pl

Zawartość

- Budowa i zasada działania listy
- Rodzaje list
- Operacje na listach jednokierunkowych
- Implementacja listy jednokierunkowej

Tablice

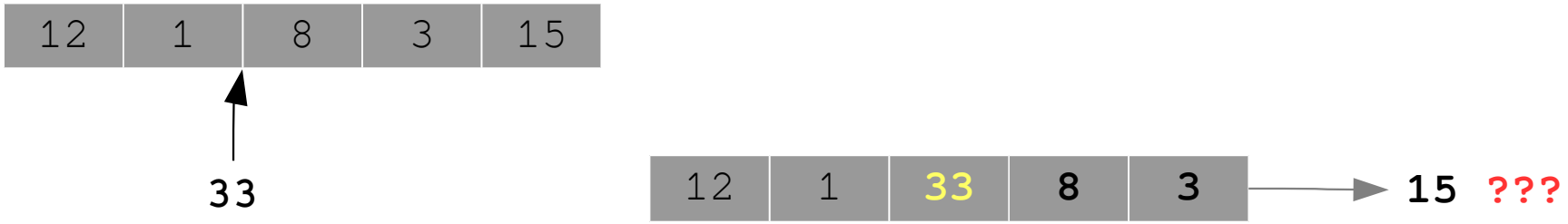
Tablice pozwalają na przechowywanie wielu danych określonego typu.
Mają one jednak dwie podstawowe wady:

– Rozmiar tablicy musi być określony przed kompilacją.

```
int tablica[10];
```

~~~~
int rozmiar = 10;
int tablica[rozmiar];
~~~~

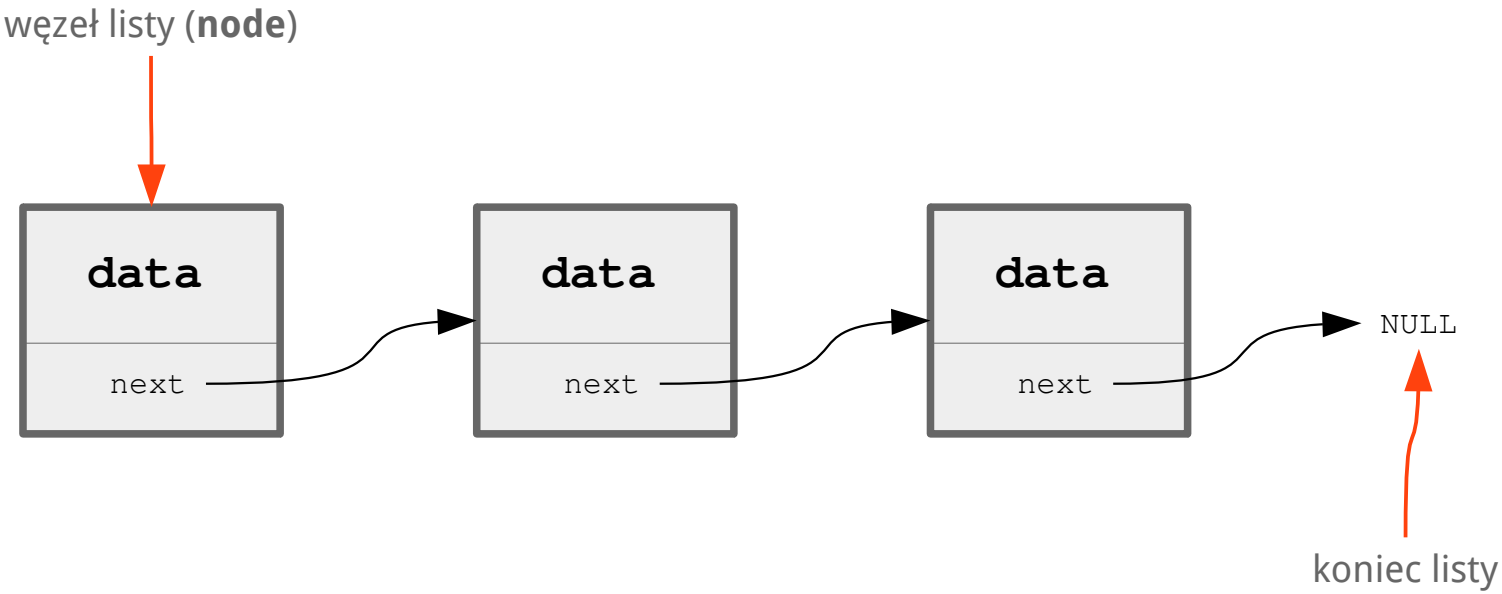
– Wstawienie elementu w środku tablicy wymaga przesunięcia pozostałych danych (co może stwarzać problemy i być czasochłonne).



Budowa listy

W wielu sytuacjach lepszym rozwiązaniem jest zastosowanie **listy**.

Lista jest tzw. **strukturą połączoną** złożoną z powiązanych ze sobą **węzłów**:

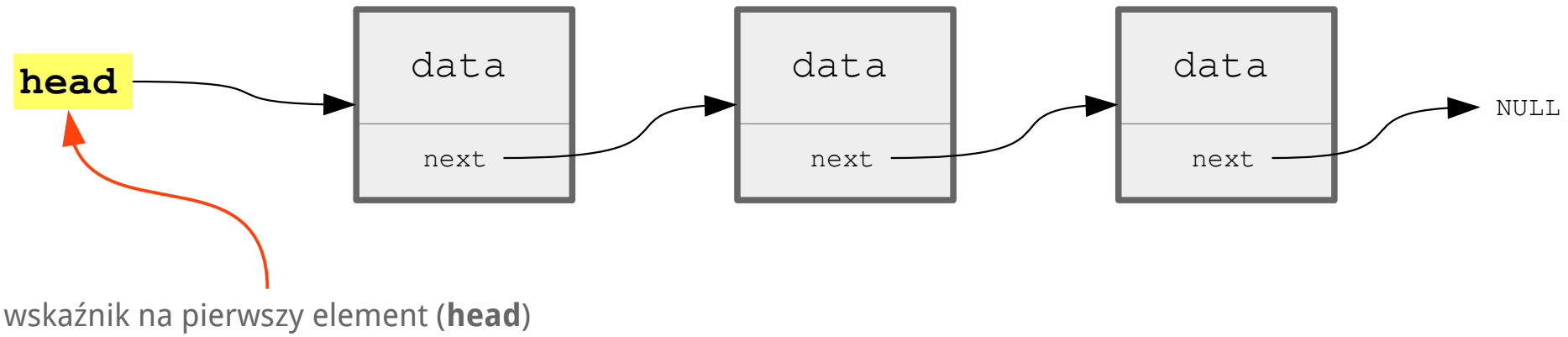


Każdy węzeł przechowuje **dane** (`data`) oraz **wskaźnik na kolejny węzeł** (`next`).

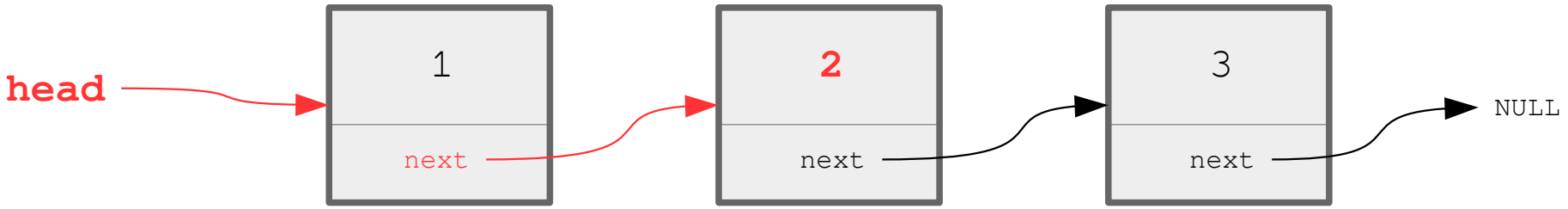
Wskaźnik w ostatnim elemencie listy ma wartość `NULL`.

Budowa listy

Aby można było się dostać do poszczególnych elementów, musimy dysponować **wskaźnikiem na pierwszy element** listy (tzw. głowę, czyli ang. **head**).

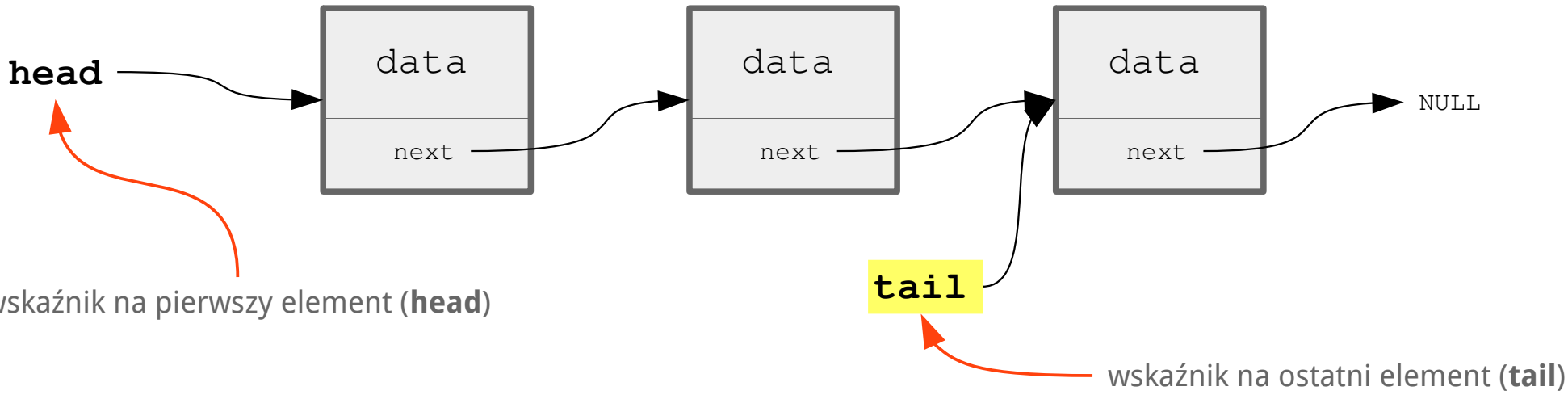


Odczytanie wybranego elementu listy polega na przejściu drogi od początku (head) poprzez kolejne węzły (za pośrednictwem wskaźników zapisanych w węzłach):



Budowa listy

W przypadku długich list dojście do ostatniego elementu (np. w celu wstawienia nowego węzła na koniec listy, o czym dalej) jest czasochłonne, więc warto dysponować jeszcze jednym wskaźnikiem – na **ostatni element** (tzw. ogon, czyli **tail**).



Rodzaje list

Tego typu listy mają pewne ograniczenia, które w niektórych zastosowaniach są nie do przyjęcia (choćby ze względów wydajnościowych).

Nie można, na przykład, poruszać się po takiej liście w kierunku od końca do początku. Właśnie z tego względu nazywa się je **listami jednokierunkowymi**.

W praktyce są stosowane również inne rodzaje list, m.in.:

- **listy dwukierunkowe**
- **listy cykliczne**
- listy z przeskokami
- listy samoorganizujące się

Dwie pierwsze listy omówię dokładniej na wykładzie, a Wy je zaimplementujecie.

Operacje na listach

Na listach można przeprowadzić kilka operacji:

- Wyszukiwanie elementu o podanej wartości
- Wyszukiwanie n-tego elementu
- Wstawianie elementu na początek listy
- Wstawianie elementu na koniec listy
- Wstawianie elementu w środku listy
- Usuwanie elementu

W tej prezentacji zostaną opisane tylko operacje **wyszukiwania** i **wstawiania** elementów na listę jednokierunkową. Usuwanie opiszę w kolejnej prezentacji.

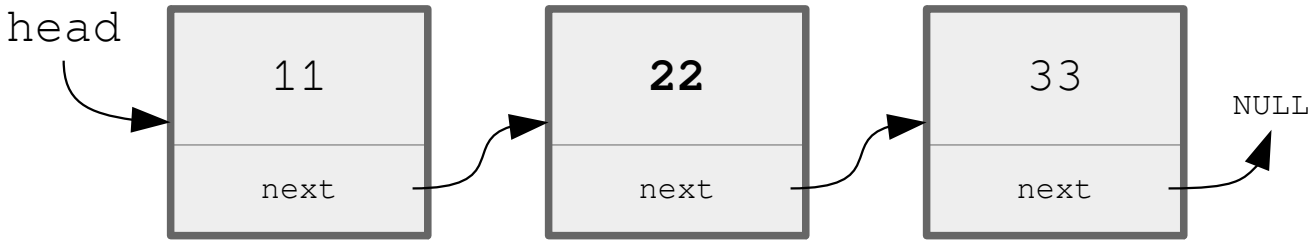
Operacje na listach

Zaczynamy od **wyszukiwania**

Wyszukiwanie elementu o podanej wartości

Aby w liście jednokierunkowej wyszukać węzeł zawierający określone dane, trzeba ją przejrzeć element po elemencie, rozpoczynając od początku.

Przypuśćmy, że w poniższej liście szukamy węzła z wartością 22.



Oto pseudokod przedstawiający algorytm wyszukiwania:

```
x   //szukana wartość

node = head

while (node != NULL && node.data != x)
    node = node.next
```

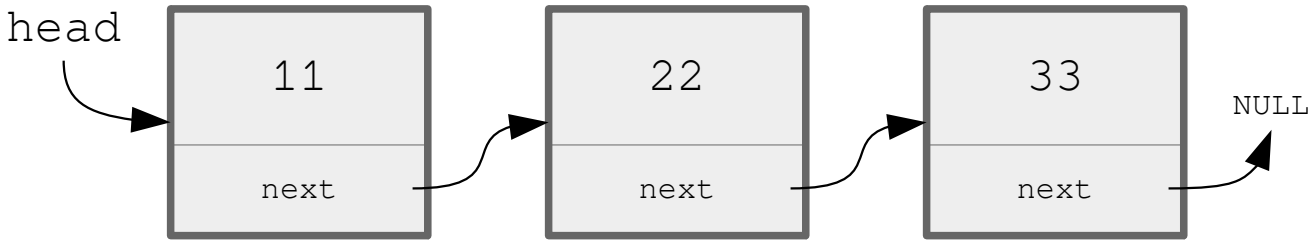
W zmiennej `node` znajdzie się wskaźnik na węzeł zawierający szukaną wartość lub NULL (jeżeli węzeł nie został znaleziony).

Wyszukiwanie n-tego elementu

Szukanie n-tego elementu listy jednokierunkowej przebiega podobnie, ale zamiast porównywać wartości zapisane w węzłach, musimy odliczać przejrzane węzły.

Przypuśćmy, że w poniższej liście szukamy **trzeciego** węzła (o numerze **2**).

pierwszy element ma numer 0 (tak samo jak w tablicach)



Pseudokod przedstawiający algorytm wyszukiwania:

```
n   //numer szukanego elementu listy

node = head

while (node != NULL && n-- > 0)
    node = node.next
```

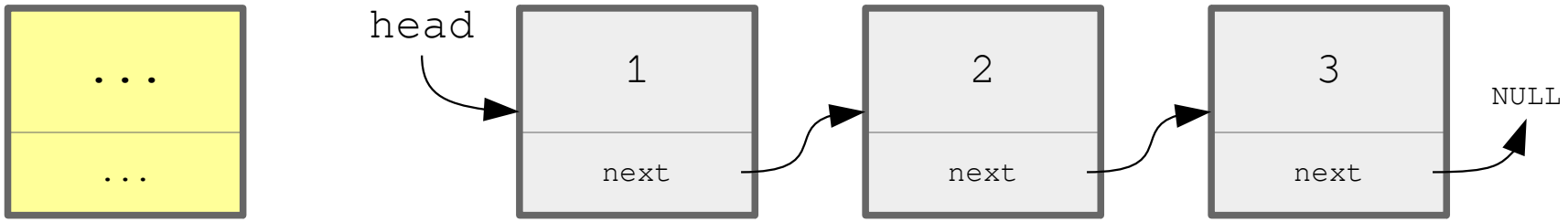
Podobnie jak poprzednio, w zmiennej `node` znajdzie się wskaźnik na węzeł zawierający szukaną wartość lub NULL (jeżeli węzeł nie został znaleziony).

Teraz **wstawianie elementów**

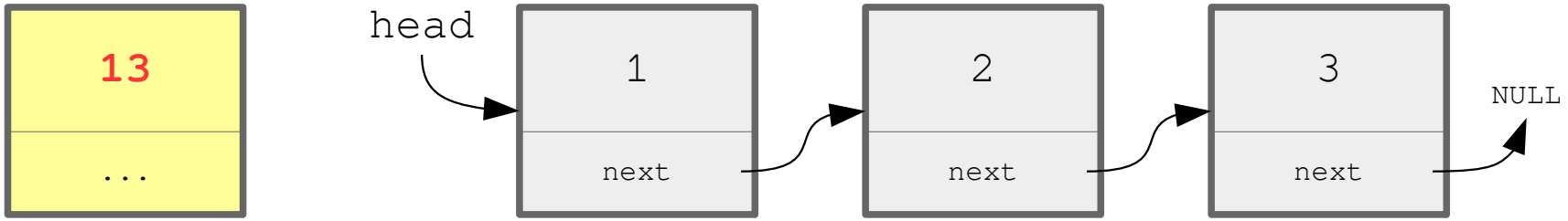
Wstawianie elementu na początek listy

Operację **wstawiania elementu na początek listy** można podzielić na cztery etapy:

❶ Tworzymy pusty węzeł:

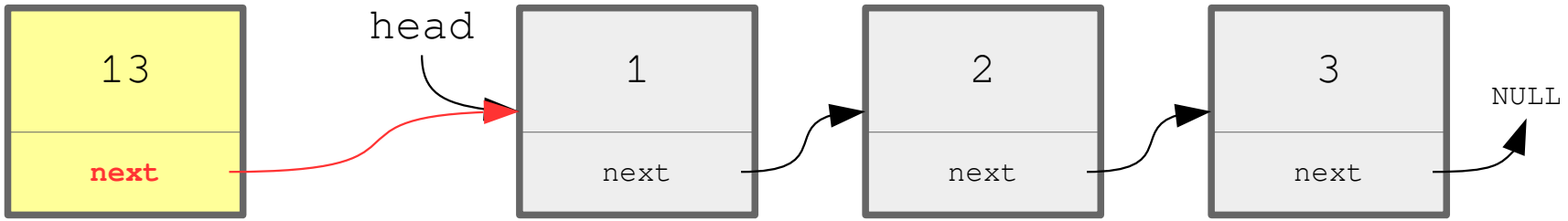


❷ Ustawiamy pole danych:

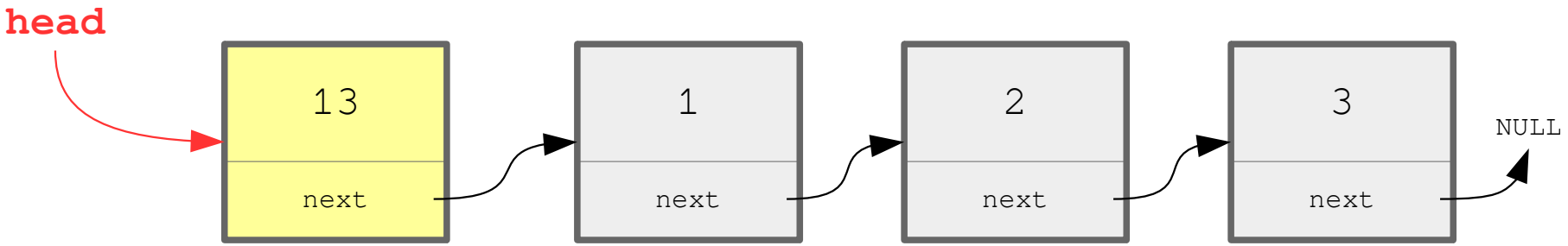


Wstawianie elementu na początek listy

❸ Pole `next` nowego węzła ustawiamy na wartość `head`:



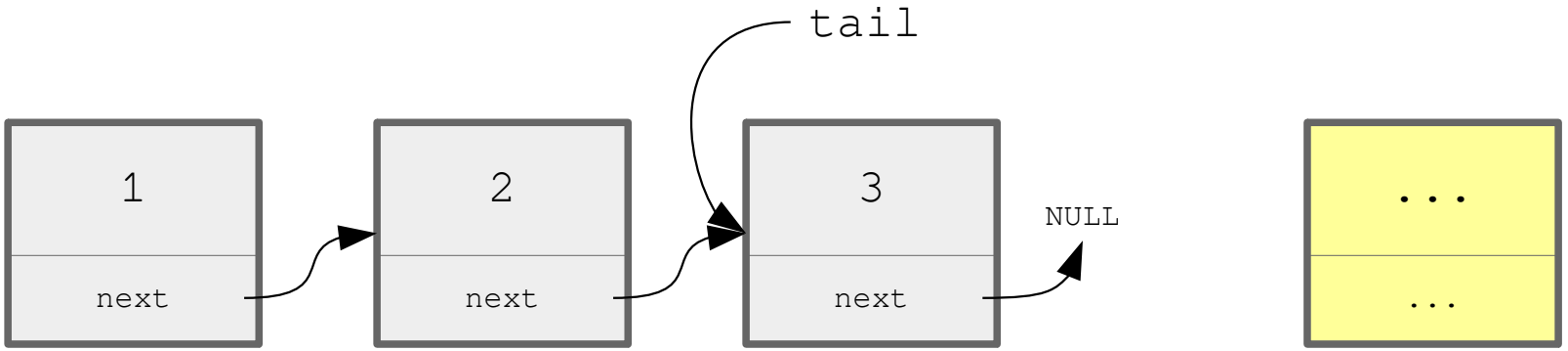
❹ Zmiennej `head` przypisujemy wskaźnik na nowy węzeł:



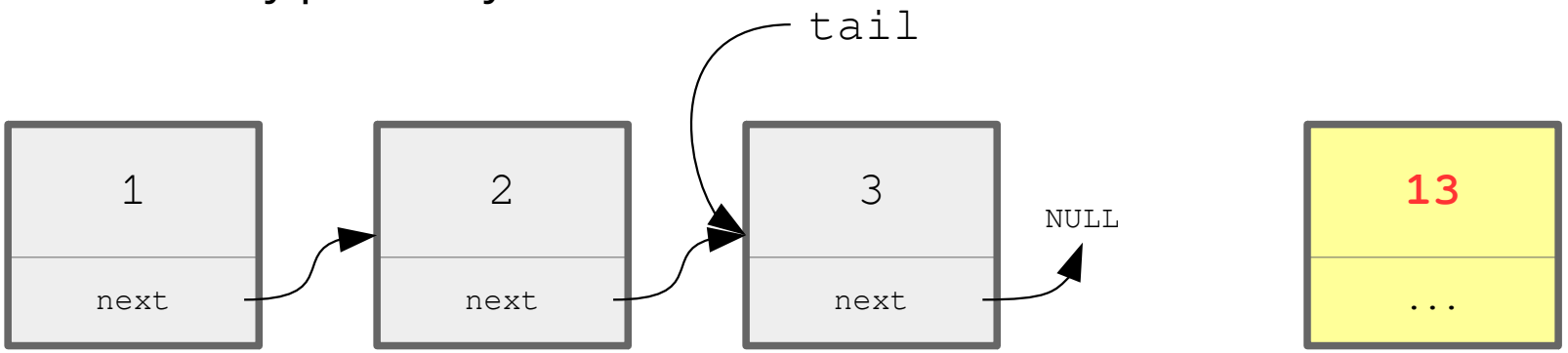
Wstawianie elementu na koniec listy

Operację **wstawiania elementu na koniec listy** można podzielić na pięć etapów:

❶ Tworzymy pusty węzeł:

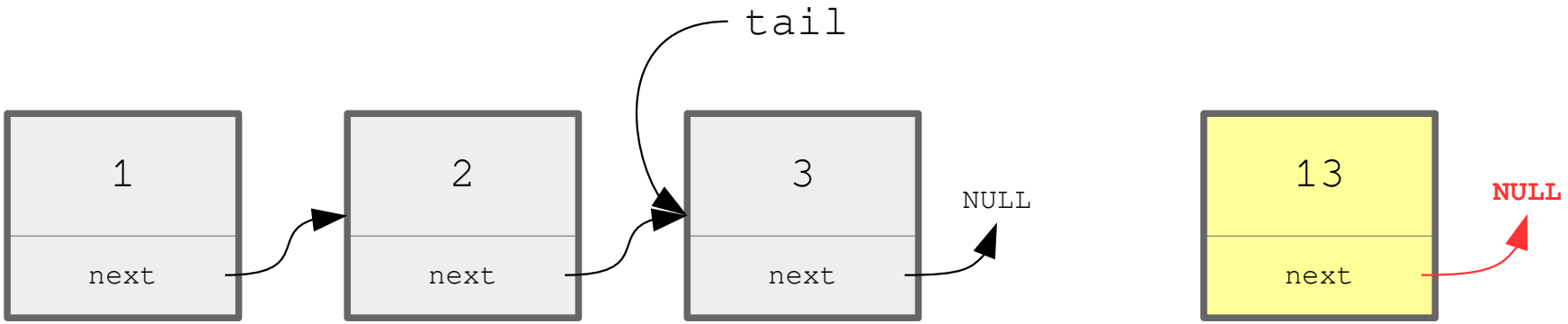


❷ Ustawiamy pole danych:

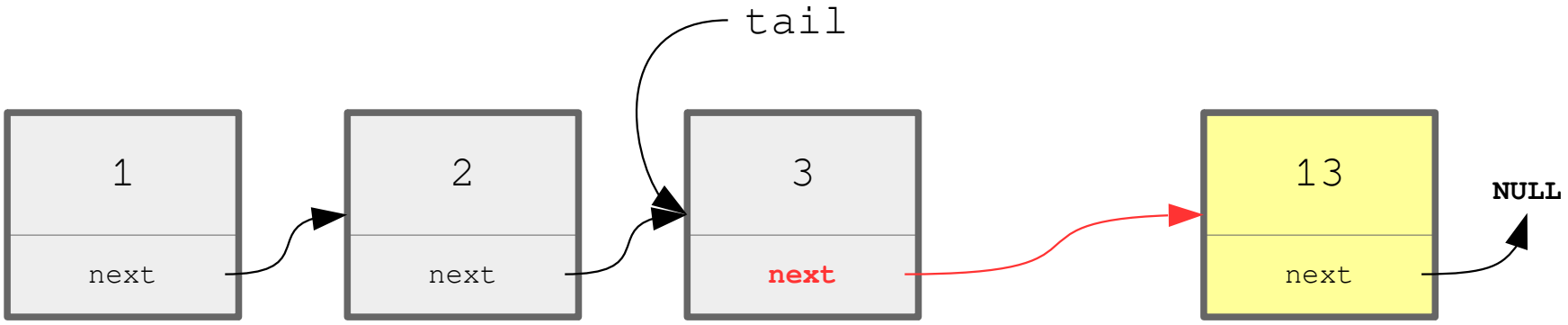


Wstawianie elementu na koniec listy

❸ Pole `next` nowego węzła ustawiamy na `NULL`:

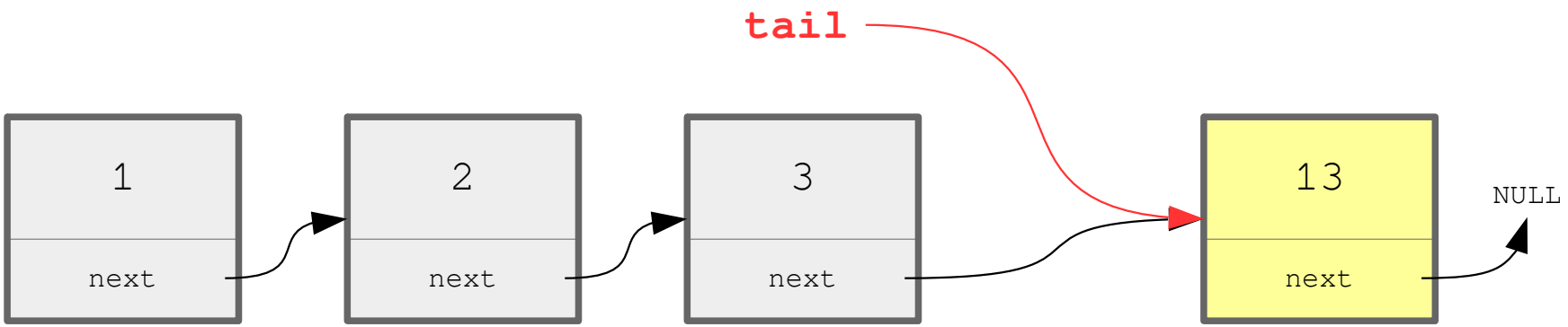


❹ Polu `next` ostatniego węzła (wskazywanego przez `tail`) przypisujemy wskaźnik na nowy węzeł:



Wstawianie elementu na koniec listy

❺ Zmiennej `tail` przypisujemy wskaźnik na nowy węzeł:



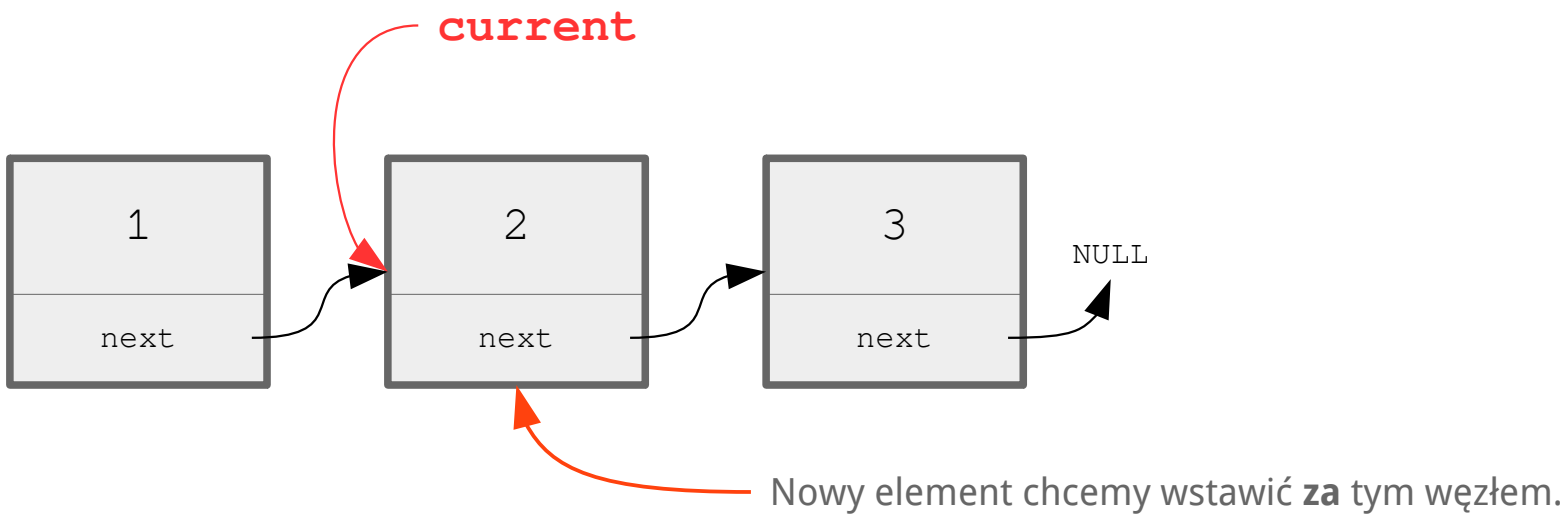
Wstawianie elementu w środku listy

Operacja **wstawiania elementu w środku listy** jest trochę bardziej skomplikowana, ponieważ wymaga znalezienia elementu, po którym należy wstawić nowy.

Odszukiwanie elementu wymaga wprowadzenia zmiennej przechowującej wskaźnik na bieżący element (`current`).

Na potrzeby tego przykładu pomijam szczegóły związane z wyszukiwaniem węzła.

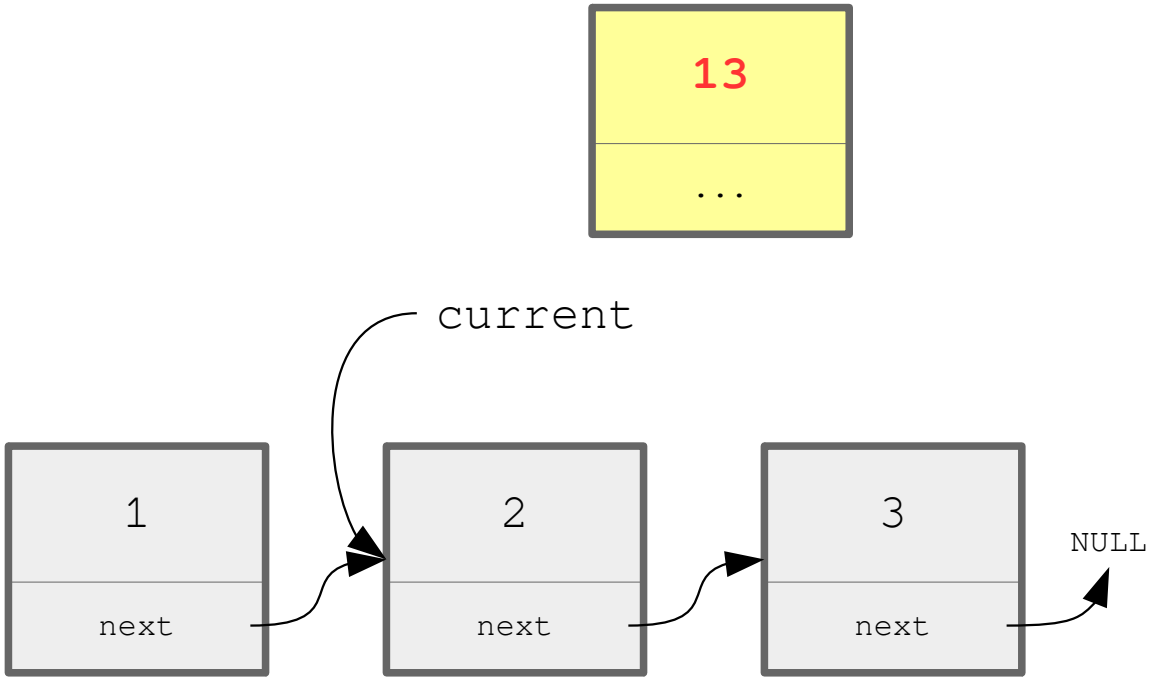
❶ Odszukujemy węzeł, za którym ma zostać wstawiony nowy element. Wskaźnik na ten węzeł znajduje się w zmiennej `current`:



Wstawianie elementu w środku listy

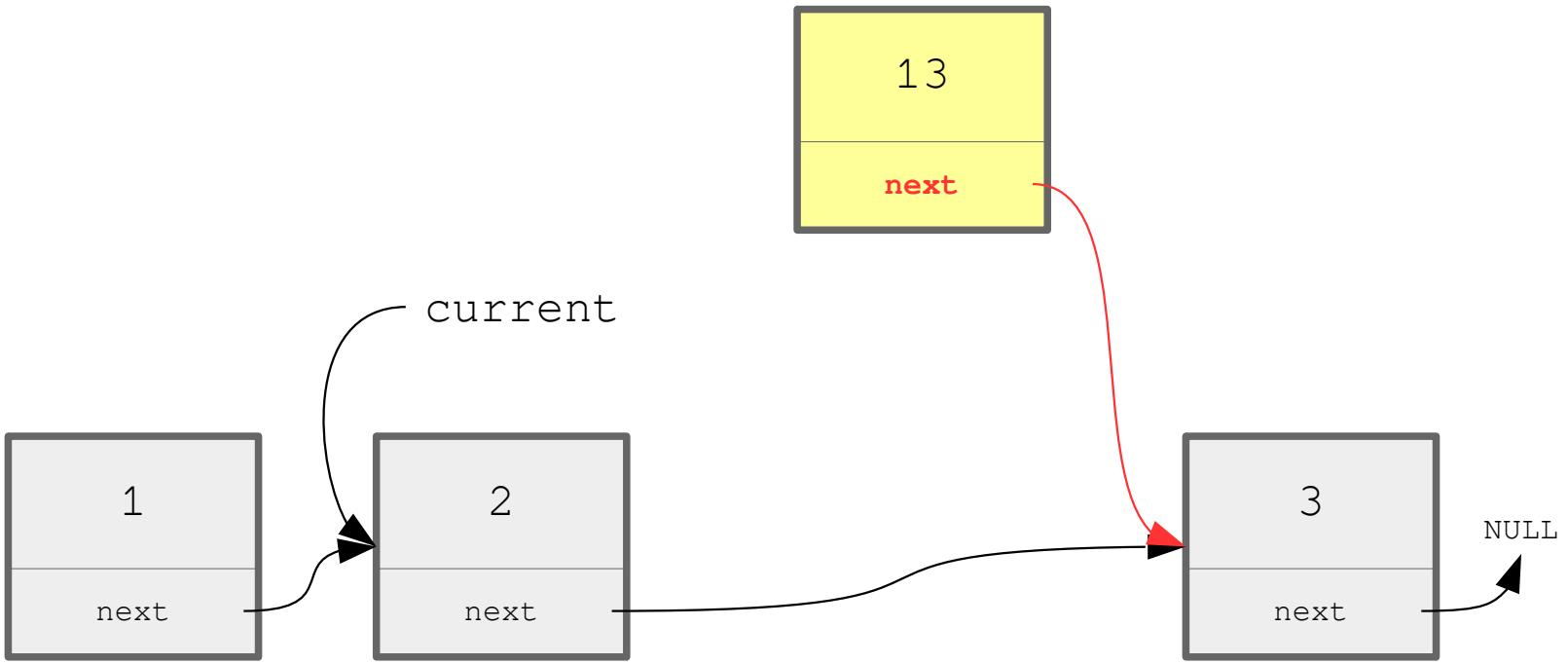
❷ Tworzymy pusty węzeł...

❸ ...i ustawiamy pole danych:



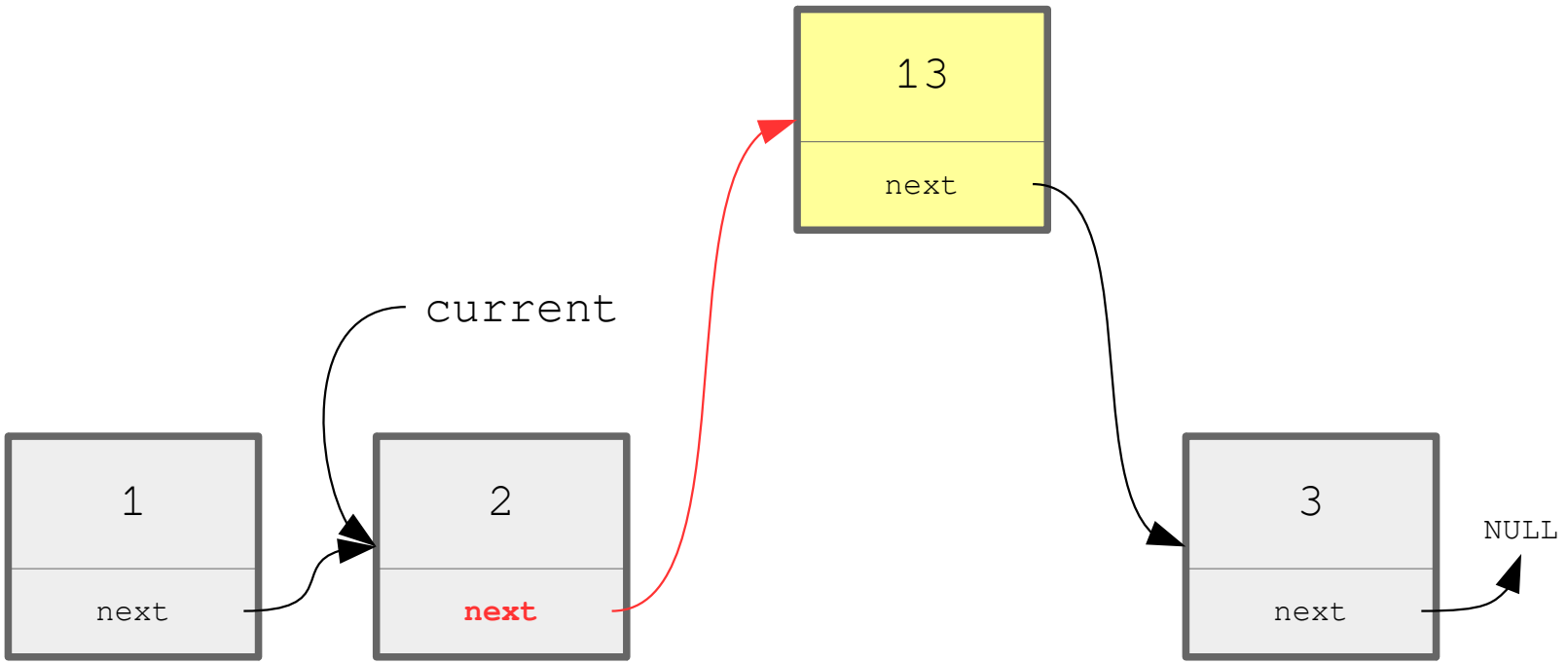
Wstawianie elementu w środku listy

❹ Polu `next` nowego węzła przypisujemy wskaźnik znajdujący się w polu `next` bieżącego węzła:



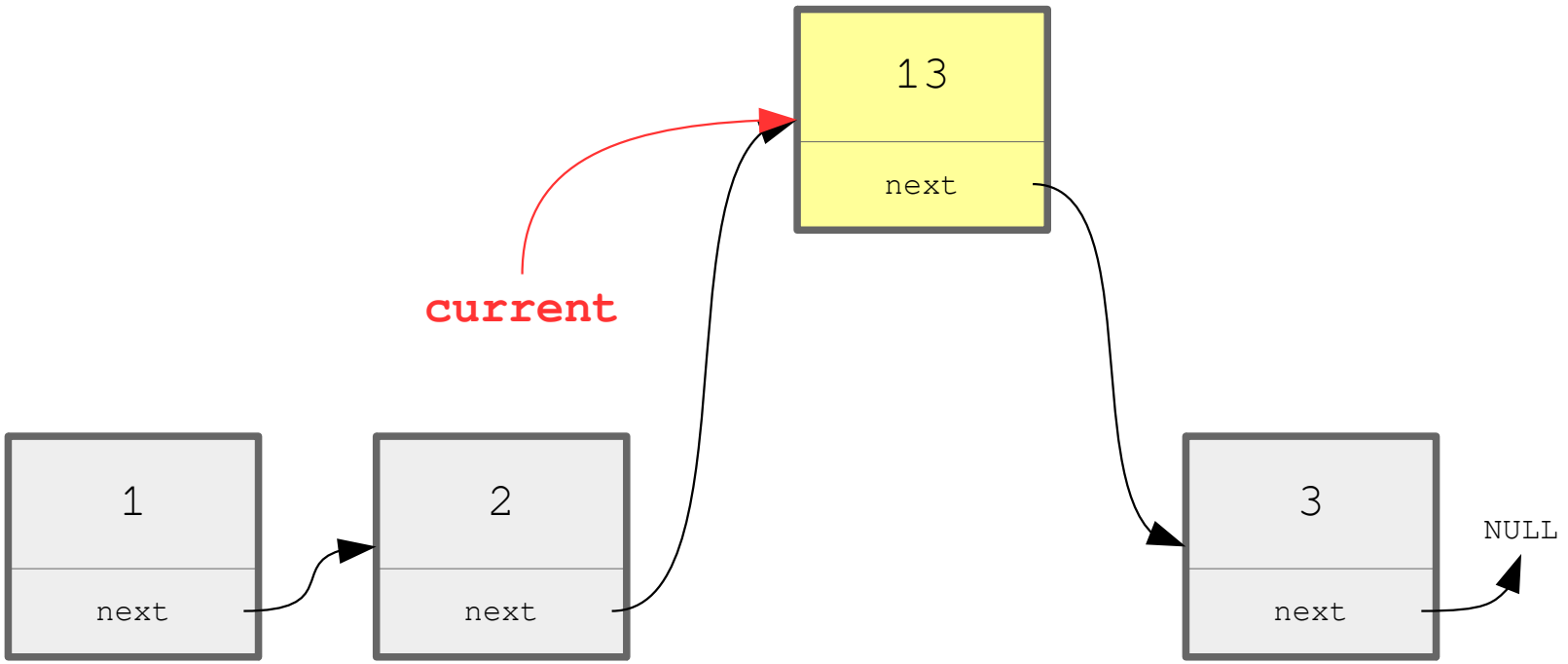
Wstawianie elementu w środku listy

❺ Polu `next` bieżącego węzła (`current`) przypisujemy wskaźnik na wstawiany węzeł:



Wstawianie elementu w środku listy

❻ Zmiennej `current` przypisujemy wskaźnik na wstawiony węzeł (opcjonalnie).



Implementacja listy jednokierunkowej

Zaimplementowanie listy będzie wymagało przygotowania struktury reprezentującej węzeł oraz funkcji operujących na węzłach.

Założenia:

- węzeł ma przechowywać pojedynczą **liczbę całkowitą** (`int`)
- początkowo węzeł ma być **strukturą** (później zamienimy ją na klasę)
- jako pierwsze mają zostać zaimplementowane funkcje do **przeglądania listy** i **wyszukiwania elementów**
- następnie należy dopisać funkcje do **wstawiania elementów** (trzy przypadki)

Implementacja listy jednokierunkowej

Mamy taką strukturę:

Odpowiadający jej kod wygląda następująco:

```
typedef struct Node {
    int data;
    Node* next;
} Node;
```

Zwróćcie uwagę, że nazwa `Node` występuje dwukrotnie!

Dlaczego?

Implementacja listy jednokierunkowej

Najpierw trzeba zaimplementować funkcje:

`int next()` – „przeskakuje” bieżący węzeł i zwraca jego wartość

`void gotoHead()` – przechodzi na pierwszy węzeł

`void gotoTail()` – przechodzi na ostatni węzeł

`bool hasNext()` – zwraca `true` jeśli istnieje następny węzeł, a jeśli nie istnieje, zwraca `false`

`Node* findByData(int)` – zwraca wskaźnik na węzeł zawierający wartość przekazaną jako parametr lub `NULL`, jeśli nie znajdzie

`Node* findByIndex(int)` – zwraca wskaźnik na n-ty węzeł lub `NULL` jeśli nie znajdzie

Implementacja listy jednokierunkowej

Później zostaną tylko funkcje wstawiające:

- `void addToHead(Node*)` – dodaje węzeł na początku listy
- `void addToTail(Node*)` – dodaje węzeł na końcu listy
- `void add(Node*)` – dodaje węzeł za bieżącym elementem