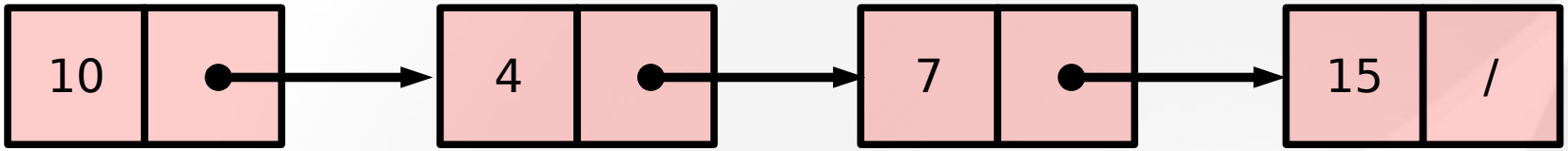
liste

Liste singolarmente linkate

Una lista è una struttura dati costituita da un insieme di nodi collegati da puntatori

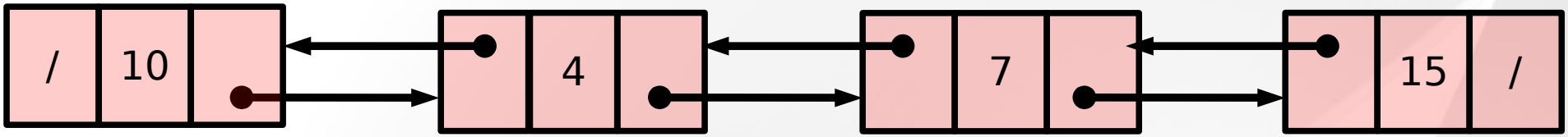


In una lista singolarmente linkata ogni nodo è composto da :

- Il valore di una chiave
- Un puntatore al nodo successivo nella lista o al puntatore nullo (NULL) se si tratta dell'ultimo nodo della lista

Liste doppiamente linkate

Una lista è una struttura dati costituita da un insieme di nodi collegati da puntatori



In una lista doppiamente linkata ogni nodo è composto da :

- Il valore di una chiave
- Un puntatore al nodo successivo nella lista o al puntatore nullo (NULL) se si tratta dell'ultimo nodo della lista
- Un puntatore al nodo precedente nella lista o al puntaore nullo (NULL) se si tratta del primo nodo della lista

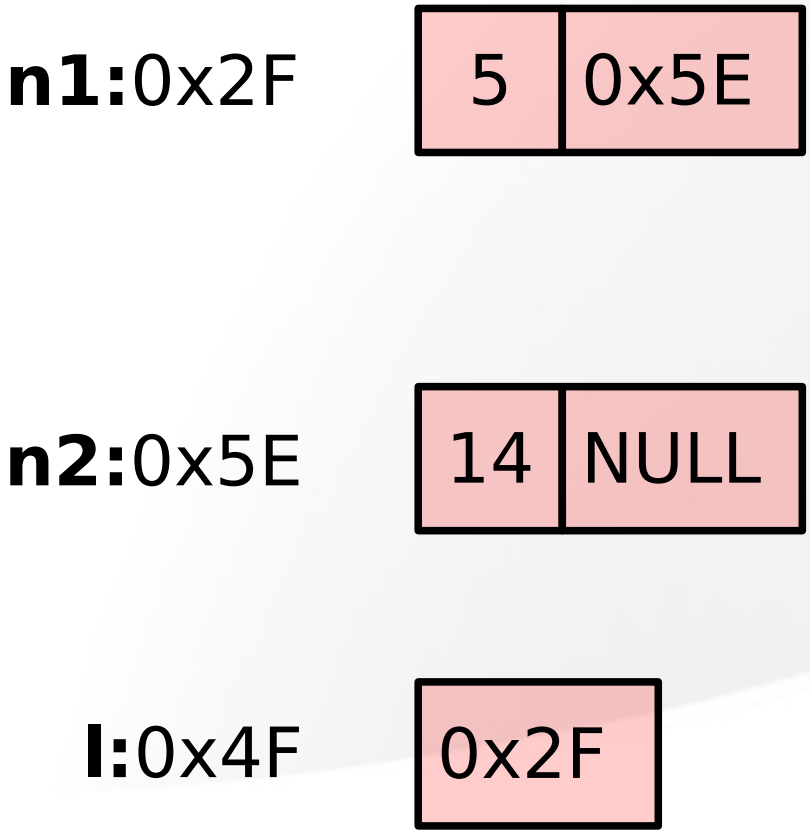
Implementazione C (liste singole)1/2

```
/*
Definizione del tipo Nodo (ricorsivamente!)
*/
typedef struct _nodo{
    int key;                  //chiave intera
    struct _nodo* next; //puntatore(!!!!) al prossimo nodo
} Nodo;


/*
Rappresentiamo una lista come un puntatore che punta al primo
nodo della lista oppure a NULL se la lista è vuota
*/
typedef Nodo* Lista;
```

Implementazione C (liste singole)2/2

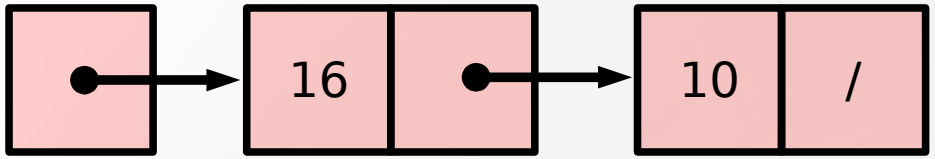
```
/*
Creazione di una lista di due nodi
*/
int main(){
     Nodo* n1=malloc(sizeof(Nodo));
     Nodo* n2=malloc(sizeof(Nodo));
     Lista l;          //N.B. Equivale a scrivere Nodo* l;

   n1->key=5;
   n1->next=n2;

     n2->key=14
   n2->next=NULL;

   l=n1;
}
```

Rappresentazione in memoria



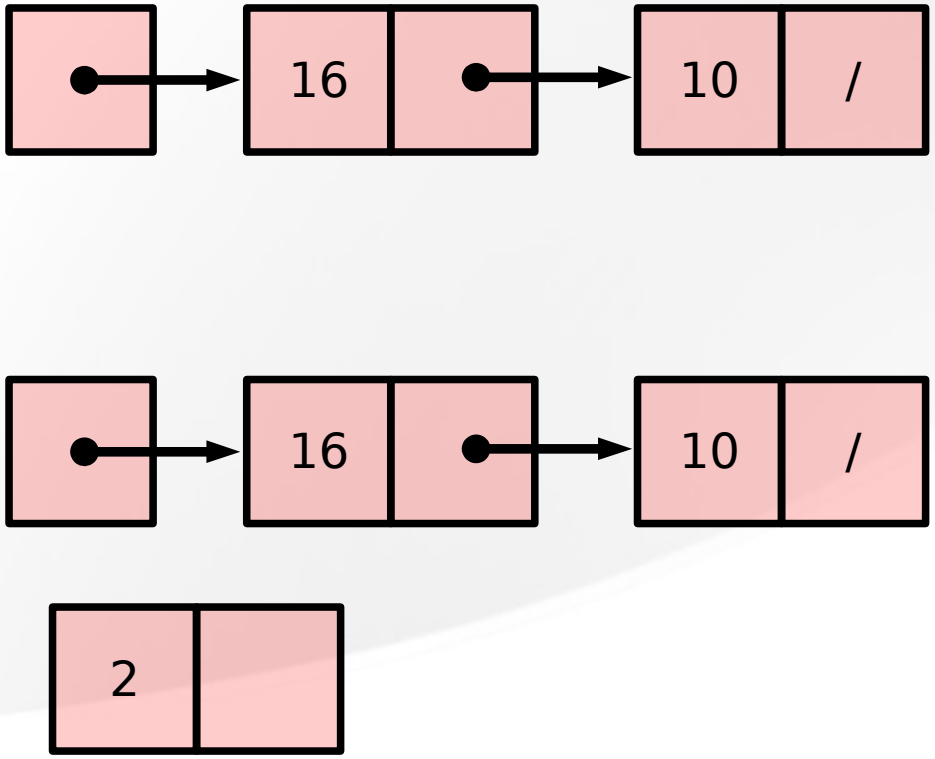
Inserimento di un elemento 1/3

Inserimento in testa alla lista



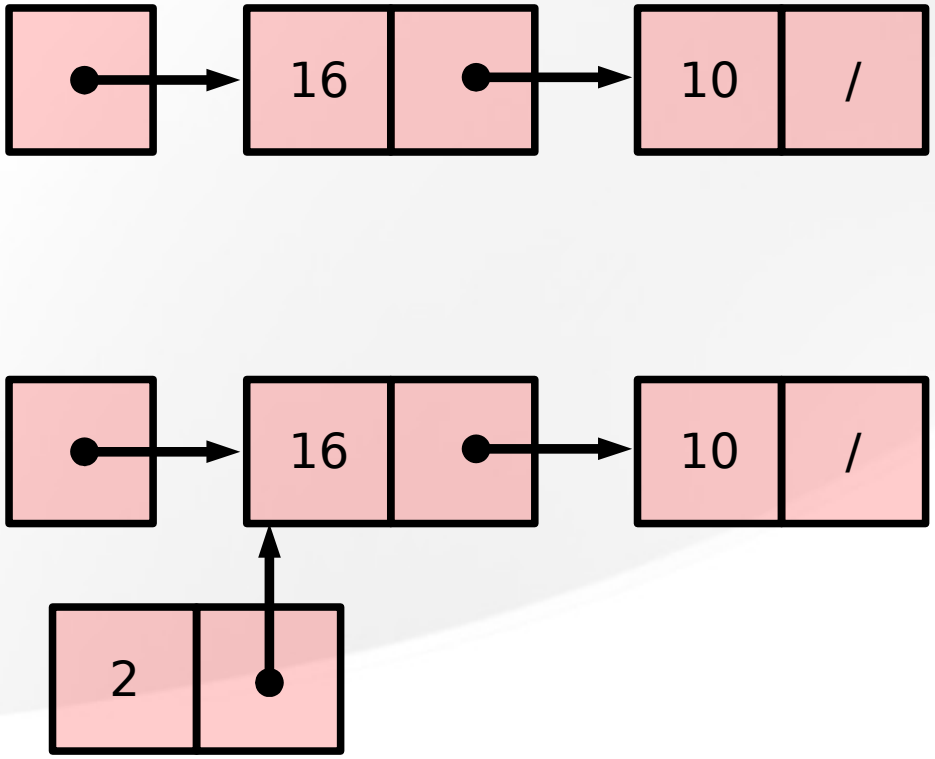
Inserimento di un elemento 1/3

Inserimento in testa alla lista



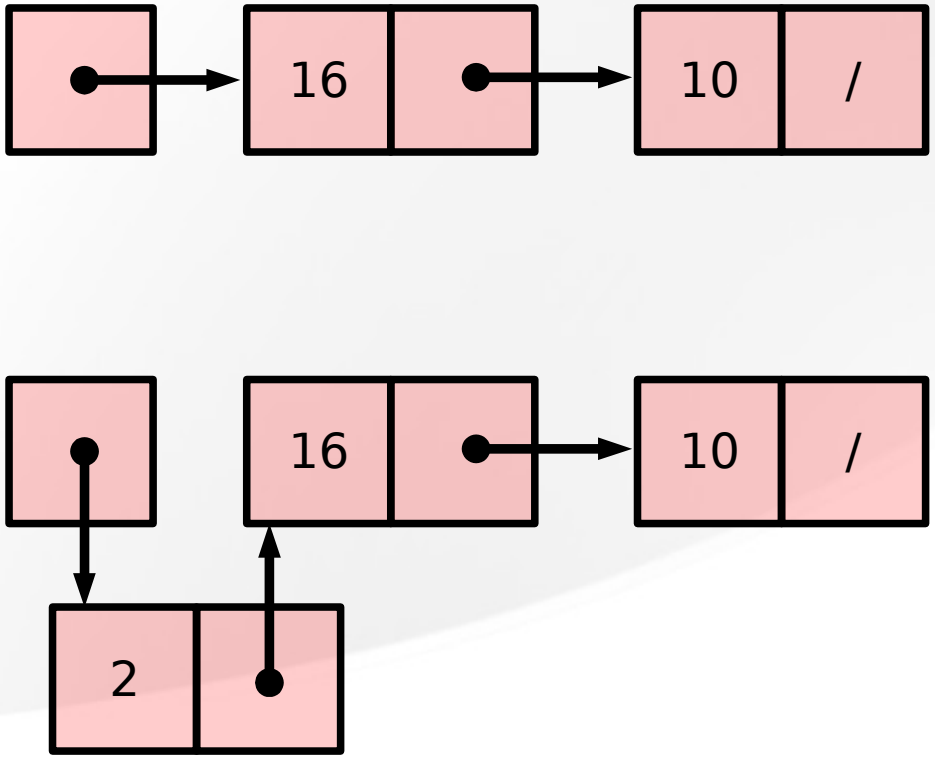
Inserimento di un elemento 1/3

Inserimento in testa alla lista



Inserimento di un elemento 1/3

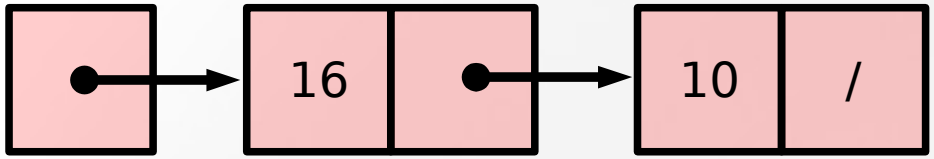
Inserimento in testa alla lista

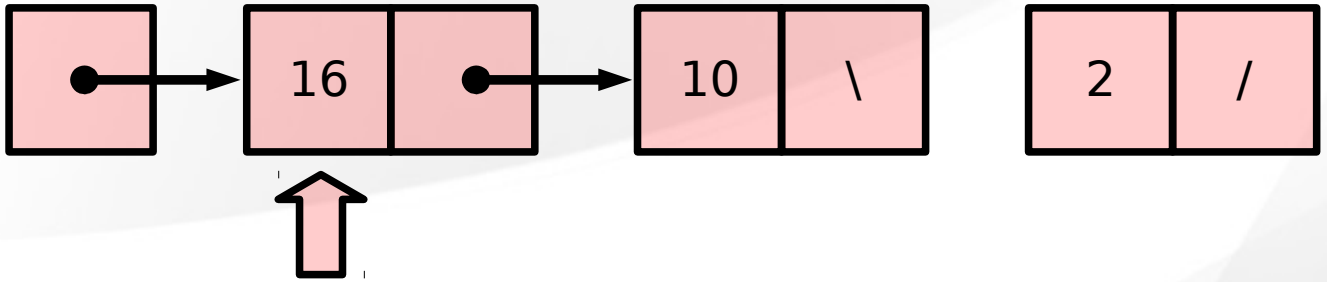


Complessita = O(1)

Inserimento di un elemento 2/3

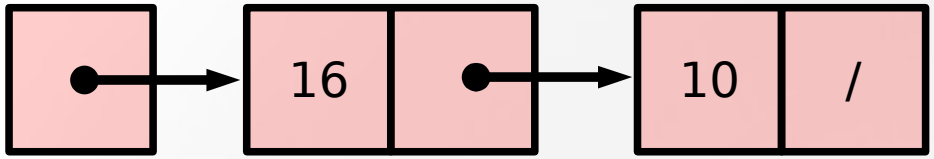
Inserimento in coda alla lista

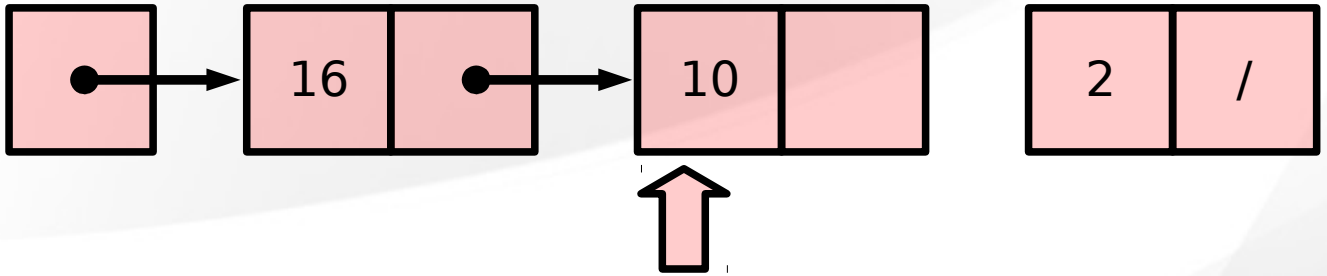




Inserimento di un elemento 2/3

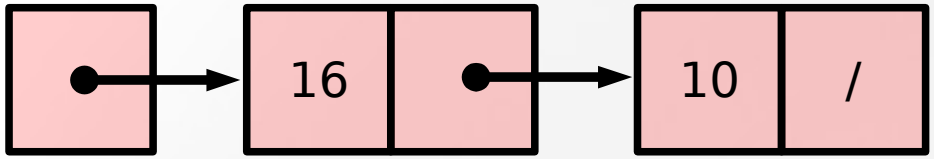
Inserimento in coda alla lista

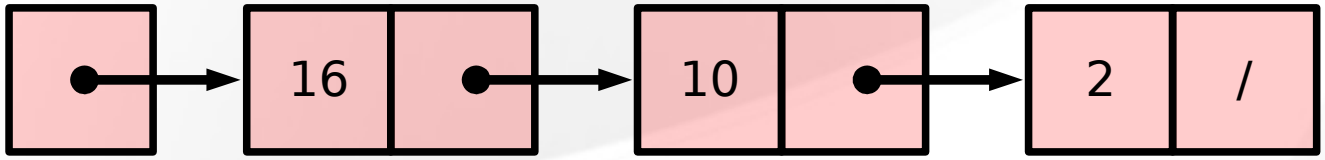




Inserimento di un elemento 2/3

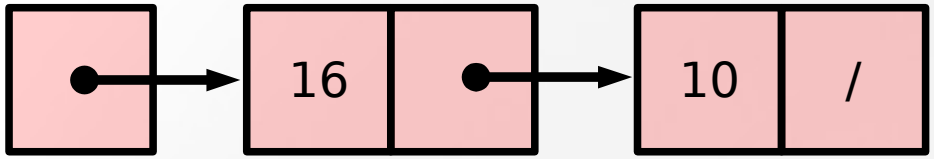
Inserimento in coda alla lista

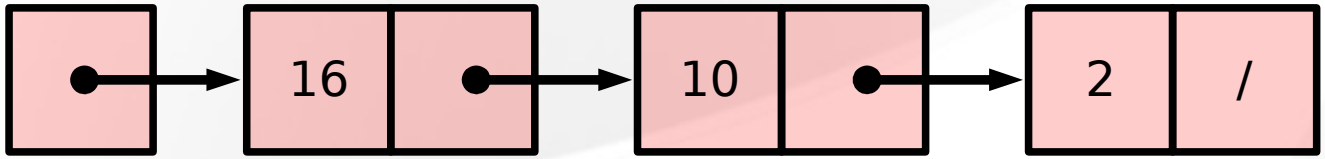




Inserimento di un elemento 2/3

Inserimento in coda alla lista

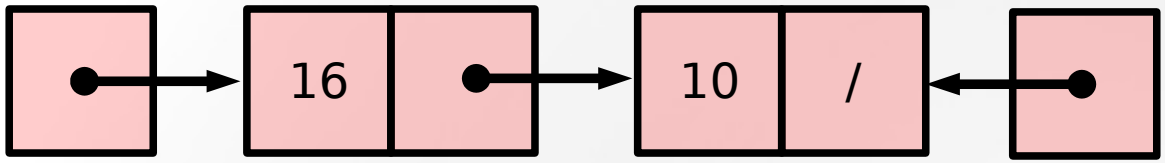




Complessita = O(n)

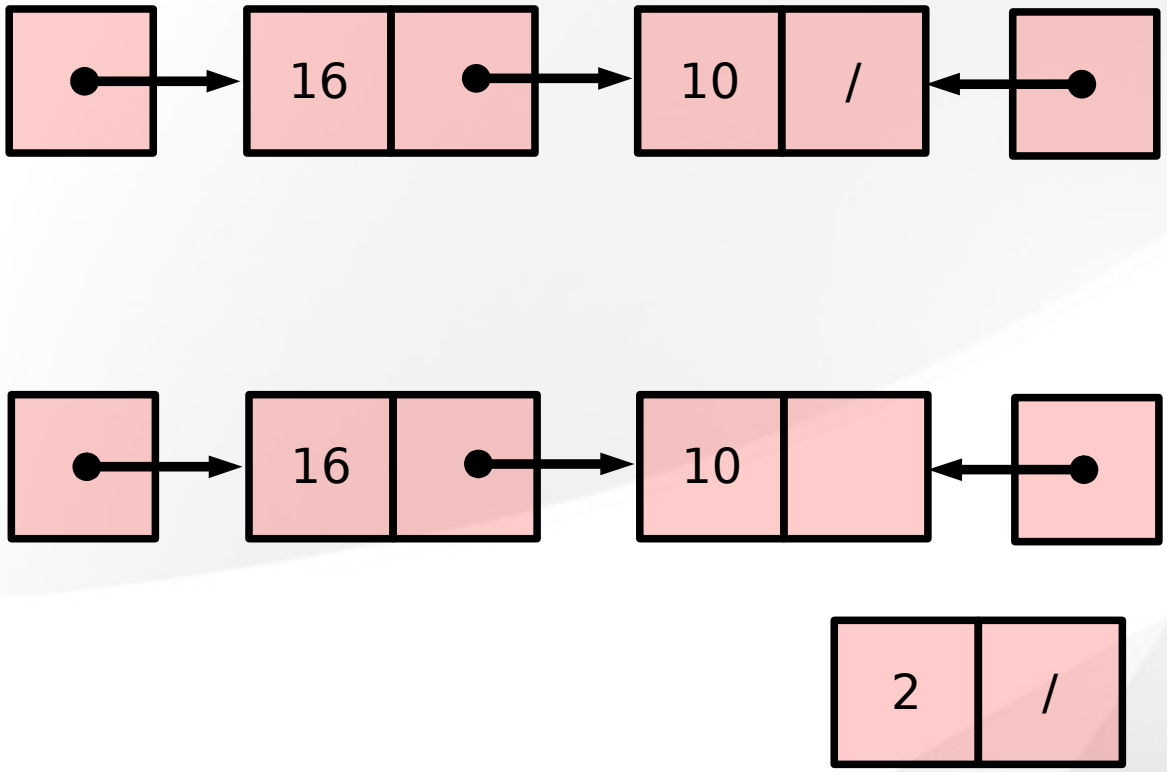
Inserimento di un elemento 3/3

Inserimento in coda alla lista (con puntatore all'elemento finale)



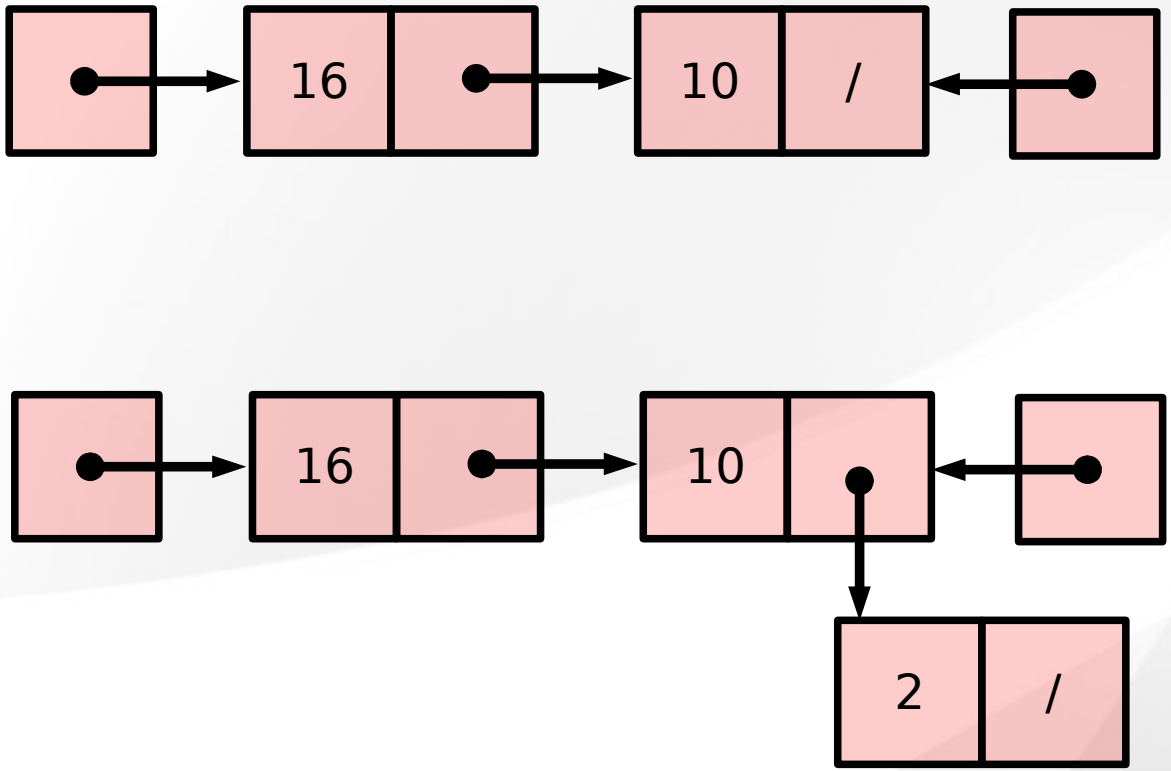
Inserimento di un elemento 3/3

Inserimento in coda alla lista (con puntatore all'elemento finale)



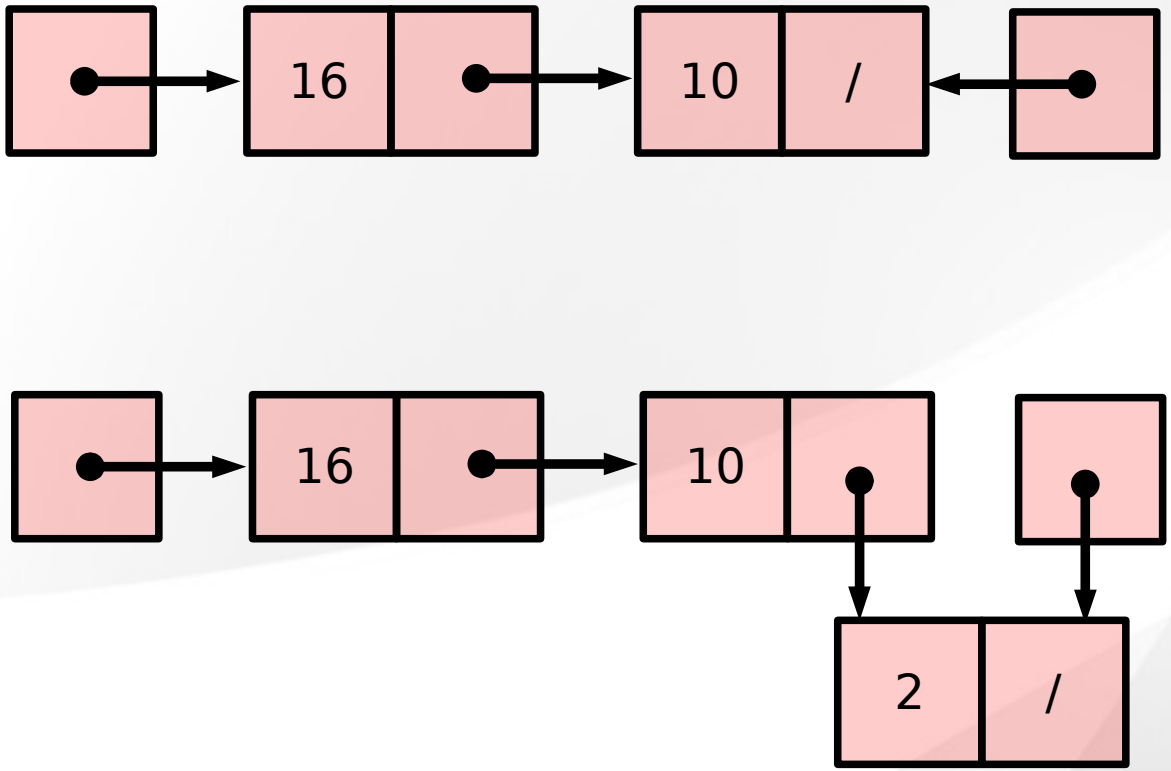
Inserimento di un elemento 3/3

Inserimento in coda alla lista (con puntatore all'elemento finale)



Inserimento di un elemento 3/3

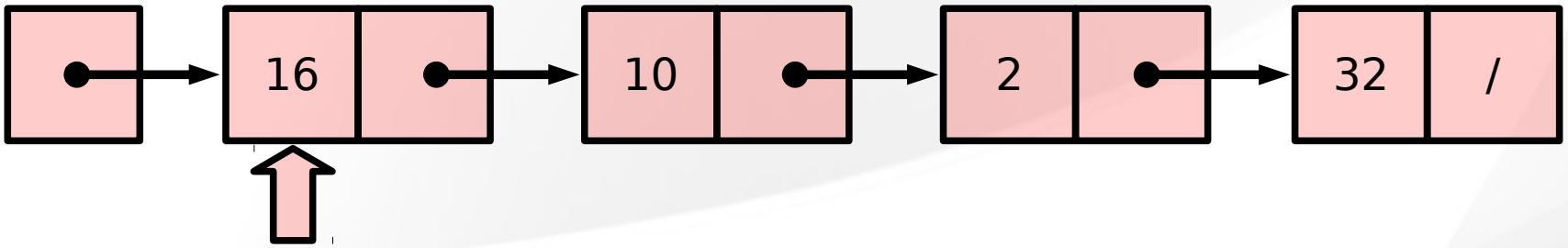
Inserimento in coda alla lista (con puntatore all'elemento finale)



Ricerca di un elemento

Scorriamo la lista partendo dal primo elemento e spostandoci di elemento in elemento fino a che non troviamo quello cercato o raggiungiamo la fine della lista.

Ricerca: 2

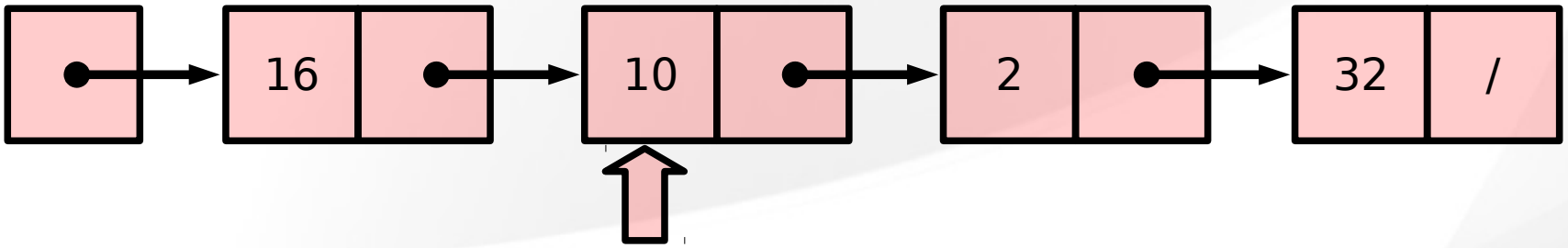


Complessita = O(n)

Ricerca di un elemento

Scorriamo la lista partendo dal primo elemento e spostandoci di elemento in elemento fino a che non troviamo quello cercato o raggiungiamo la fine della lista.

Ricerca: 2

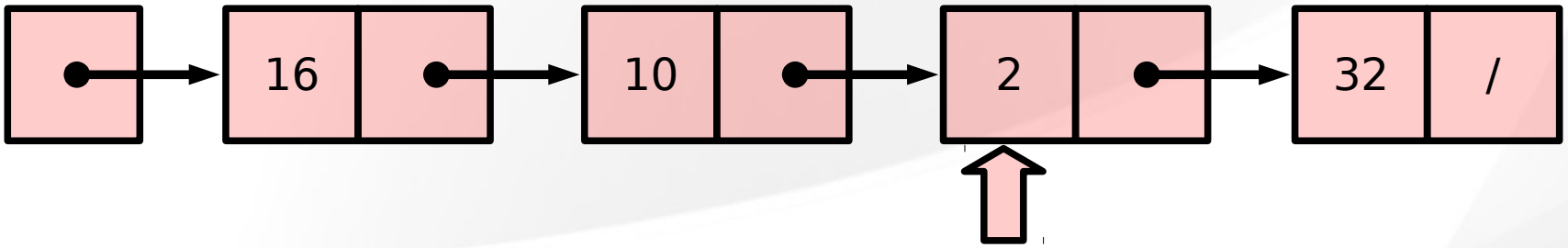


Complessita = O(n)

Ricerca di un elemento

Scorriamo la lista partendo dal primo elemento e spostandoci di elemento in elemento fino a che non troviamo quello cercato o raggiungiamo la fine della lista.

Ricerca: 2



Complessita = O(n)

Implementazione(Inserimento)

```
Nodo* InserisciTesta(Lista head,int key){
        Nodo* n_node=malloc(sizeof(Nodo));
        n_node->key=key;
        if(head==NULL) //Lista vuota!
                n_node->next=NULL;
        else
                n_node->next=head;
        return n_node;
}

int main(){{
   Lista l=NULL; // Creo lista vuota N.B.Lista=Node*(slide 5)


   l=InserisciTesta(l,2);     //Lista: 2
   l=InserisciTesta(l,6);     //Lista: 6->2
   l=InserisciTesta(l,12);    //Lista: 12->6->2
   l=InserisciTesta(l,2);     //Lista: 2->12->6->2
}
```

Implementazione (Ricerca)

```
int trovaElemento(Lista l,int key){
        int i=0;
        while(l!=NULL){
            if(l->key==key) return i;
            i++;
            l=l->next;
        }
        return -1;
}


int main(){
      Lista l;
      l=InserisciTesta(l,2);      //Lista: 2
      l=InserisciTesta(l,6);      //Lista: 6->2
      l=InserisciTesta(l,12);     //Lista: 12->6->2
      l=InserisciTesta(l,2);      //Lista: 2->12->6->2
      printf("%d\n",trovaElemento(l,12););
}
```

Implementazione (Ricerca)

```
int trovaElemento(Lista l,int key){
        int i=0;
        while(l!=NULL){
            if(l->key==key) return i;
            i++;
            l=l->next;
        }
        return -1;
}


int main(){
      Lista l;
      l=InserisciTesta(l,2);      //Lista: 2
      l=InserisciTesta(l,6);      //Lista: 6->2
      l=InserisciTesta(l,12);     //Lista: 12->6->2
      l=InserisciTesta(l,2);      //Lista: 2->12->6->2
      printf("%d\n",trovaElemento(l,12););
}
```

OUTPUT 1

Implementazione (Lunghezza)

```
int lunghezzaLista(Lista l){
    if(l==NULL)   return 0;
    else return 1+lunghezzaLista(l->next);
}


int main(){
     Lista l;
     l=InserisciTesta(l,2);    //Lista: 2
     l=InserisciTesta(l,6);    //Lista: 6->2
     l=InserisciTesta(l,12);   //Lista: 12->6->2
     l=InserisciTesta(l,2);    //Lista: 2->12->6->2
     int length=lunghezzaLista(l);
     printf("%d\n",length);
}
```

Implementazione (Lunghezza)

```
int lunghezzaLista(Lista l){
    if(l==NULL)   return 0;
    else return 1+lunghezzaLista(l->next);
}


int main(){
     Lista l;
     l=InserisciTesta(l,2);    //Lista: 2
     l=InserisciTesta(l,6);    //Lista: 6->2
     l=InserisciTesta(l,12);   //Lista: 12->6->2
     l=InserisciTesta(l,2);    //Lista: 2->12->6->2
     int length=lunghezzaLista(l);
     printf("%d\n",length);
}
```

OUTPUT 4