

# Container (abstract data type)

---

From Wikipedia, the free encyclopedia

(Redirected from [Collection class](#))

In [computer science](#), a **container** is a [class](#), a [data structure](#),<sup>[1][2]</sup> or an [abstract data type](#) (ADT) whose instances are collections of other objects. In other words; they are used for storing objects in an organized way following specific access rules. The size of the container depends on the number of the objects (elements) it contains. The underlying implementation of various types of containers may vary in space and time complexity allowing for flexibility in choosing the right implementation for a given scenario.

Container classes are expected to implement methods to do the following:

- create an empty container (constructor);
- insert objects into the container;
- delete objects from the container;
- delete all the objects in the container (clear);
- access the objects in the container;
- access the number of objects in the container (count).

Containers are sometimes implemented in conjunction with [iterators](#).



Classi contenitore della libreria STL:  
string, list, vector, map, set,...



## DETTAGLIO CONSUMI



### Sintesi Traffico Voce - Numero di Telefono

Tipologia di chiamata	Numero Chiamate	Durata	Costo IVA € (IVA inclusa)
Locale	4	0:10:37	0,00000
Nazionale	23	4:22:29	0,00000
Cellulari Nazionali	2	0:01:04	0,79198
Internazionale 1	16	0:23:23	8,61132
Numeri Verdi	4	0:11:09	0,00000
<b>Totale Traffico Voce</b>			<b>9,40330</b>

Data	Ora	Numero Chiamato	Destinazione	Durata	Fascia Oraria	Costo IVA € (IVA inclusa)
<b>Locale</b>						
13-09-2015	18:48	0236634***	Milano	0:00:07		0,00000
20-09-2015	17:09	0236635***	Milano	0:09:11		0,00000
27-09-2015	16:56	0236635***	Milano	0:00:01		0,00000
05-10-2015	15:31	0236635***	Milano	0:01:18		0,00000

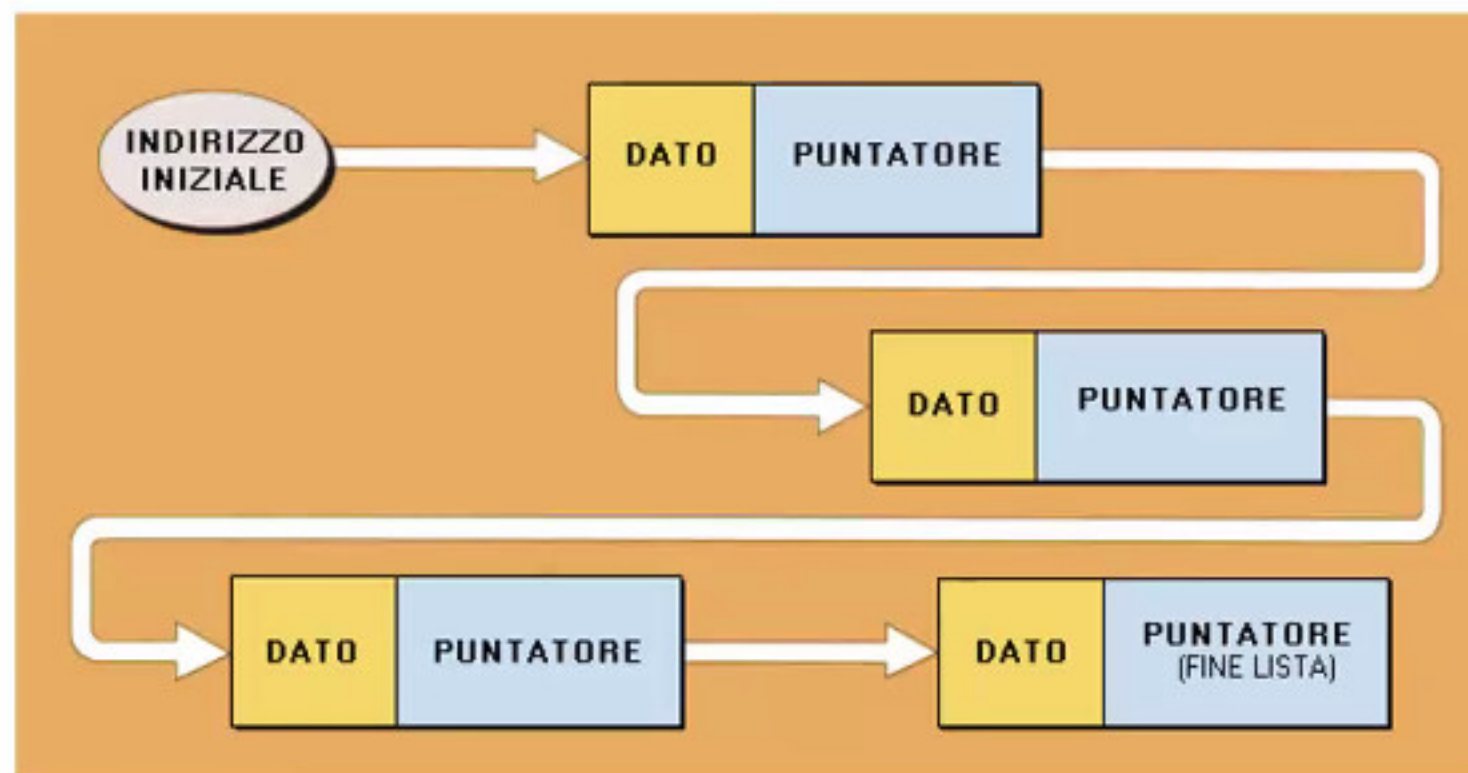


```
#ifndef BOLLETTA_H // file bolletta.h
#define BOLLETTA_H
#include "telefonata.h"
class bolletta {
public:
    bolletta();
    bool Vuota() const;
    void Aggiungi_Telefonata(const telefonata&);
    void Togli_Telefonata(const telefonata&);
    telefonata Estrai_Una();

    .....
    .....
    .....

};
#endif
```

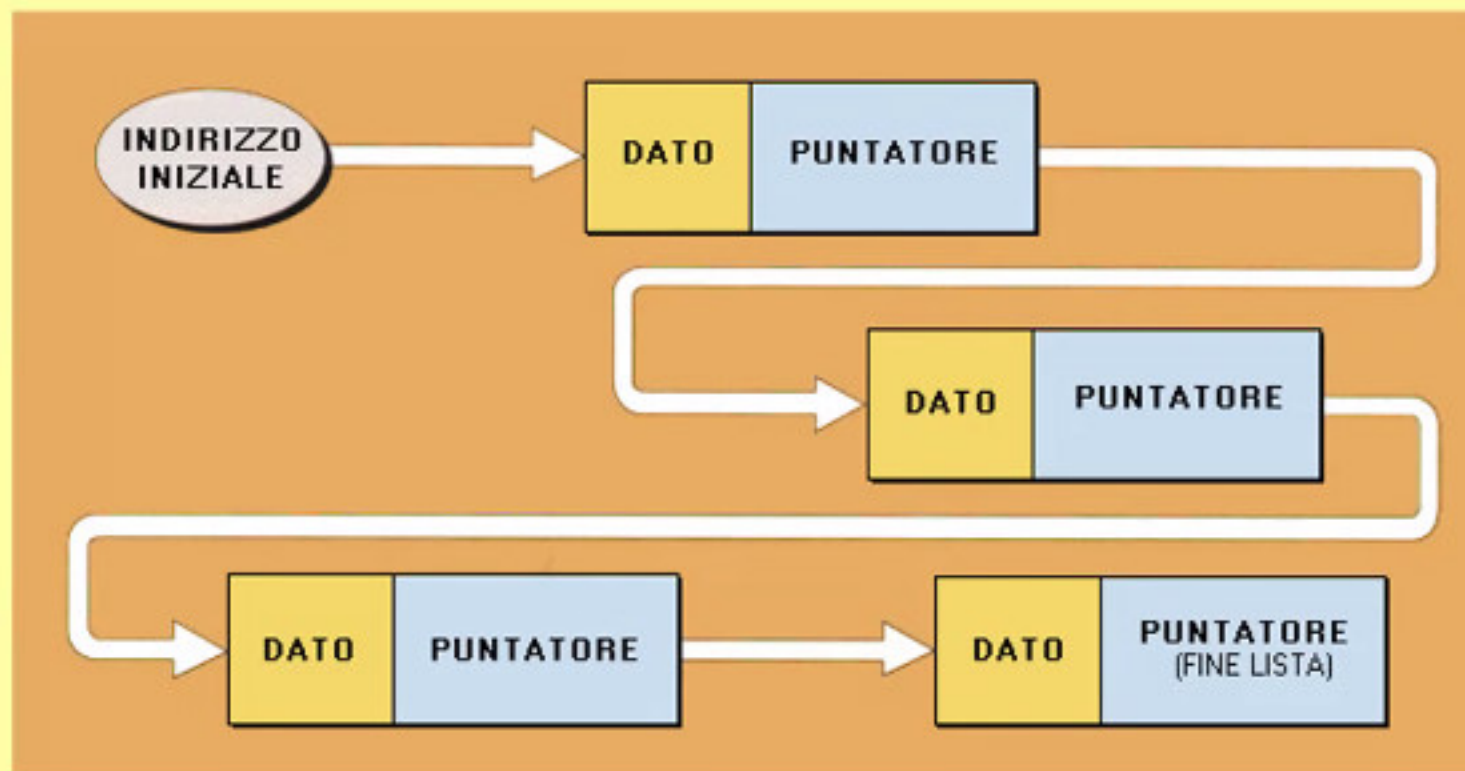
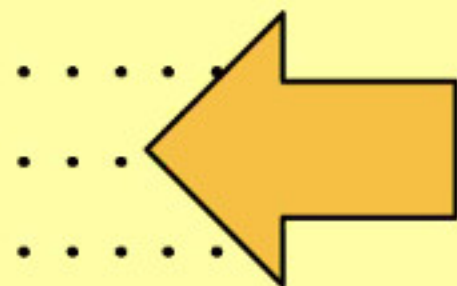
# Linked list



```

#ifndef BOLLETTA_H // file bolletta.h
#define BOLLETTA_H
#include "telefonata.h"
class bolletta {
public:
    bolletta();
    bool Vuota() const;
    void Aggiungi_Telefonata(const telefonata&);
    void Togli_Telefonata(const telefonata&);
    telefonata Estrai_Una();
private:

```




```

};
#endif

```



```
#ifndef BOLLETTA_H // file bolletta.h
#define BOLLETTA_H
#include "telefonata.h"
class bolletta {
public:
    bolletta();
    bool Vuota() const;
    void Aggiungi_Telefonata(const telefonata&);
    void Togli_Telefonata(const telefonata&);
    telefonata Estrai_Una();
private:
    class nodo {  // nodo: classe interna privata
    public:
        nodo();
        nodo(const telefonata&, nodo*);
        telefonata info;
        nodo* next;
    };
    nodo* first; // puntatore al primo nodo della lista
};
#endif
```

```
// file bolletta.cpp
#include "bolletta.h"

bolletta::nodo::nodo() : next(0) {}
    // costruttore di default per il campo dati info

bolletta::nodo::nodo(const telefonata& t, nodo* s)
    : info(t), next(s) {}


bolletta::bolletta(): first(0) {}

bool bolletta::Vuota() const {
    return first == 0;
}

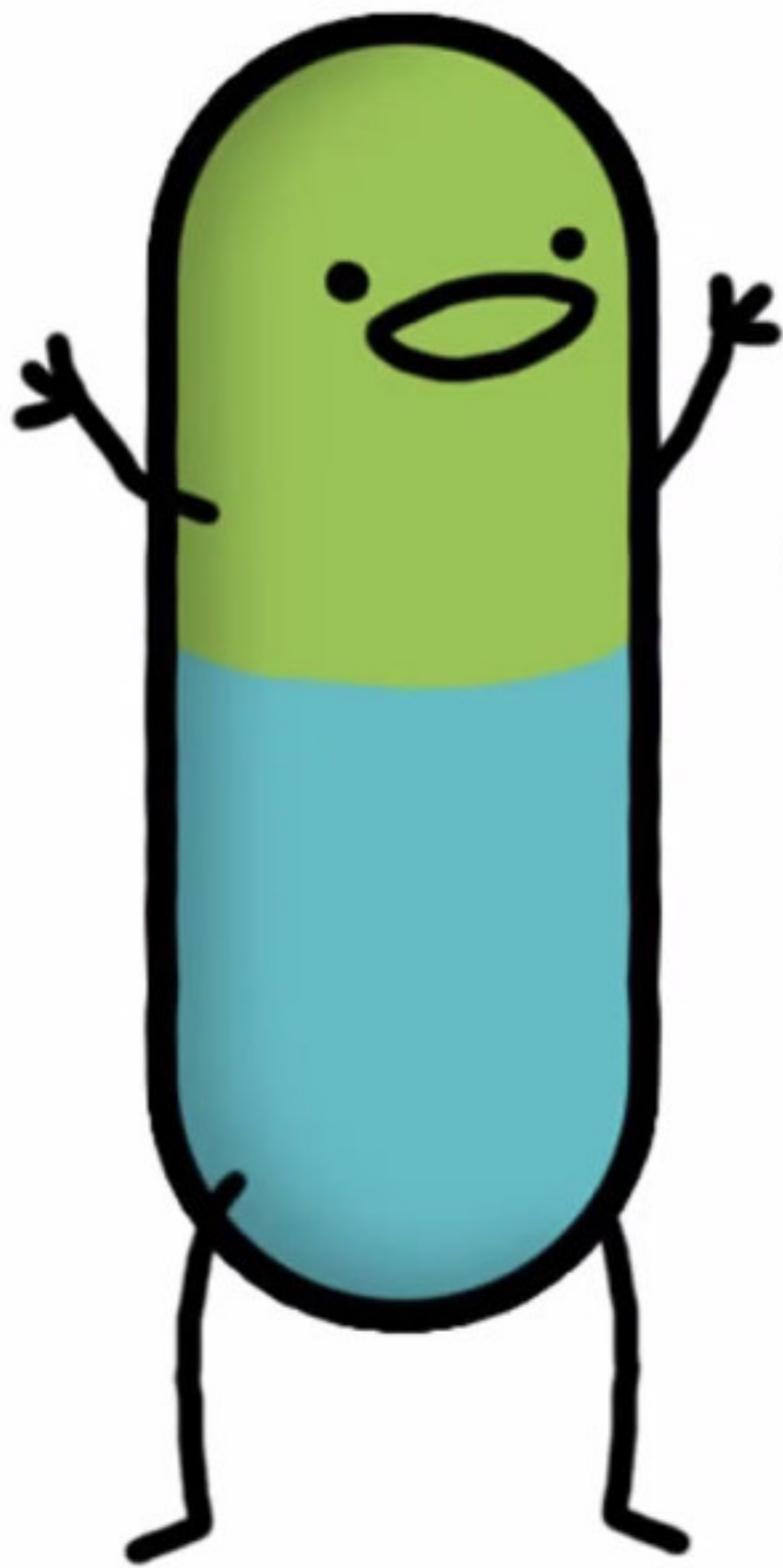
void bolletta::Aggiungi_Telefonata(const telefonata& t) {
    first = new nodo(t, first);
    // aggiunge in testa alla lista
}
```



```
void bolletta::Togli_Telefonata(const telefonata& t) {
    nodo* p = first, *prec = nullptr;
    while (p && !(p->info == t)) {
        prec = p;
        p = p->next;
    } // p==0 (not found) o p punta al nodo da rimuovere
    if (p) { // ho trovato t
        if (!prec) // p punta al primo nodo
            first = p->next;
        else // p punta ad un nodo successivo al primo
            prec->next = p->next;
        delete p; // attenzione: deallocare!
    }
}
```



```
telefonata bolletta::Estrai_Una() {  
    //Precondizione: bolletta non vuota (!(first==0))  
    nodo* p = first;  
    first = first->next;  
    telefonata aux = p->info; // costruttore di copia  
    delete p;                // attenzione: deallocare!  
    return aux;  
}
```




Side effects may  
include death!



Aggiungi\_Telefonata() e  
Togli\_Telefonata() possono  
provocare **side effects** sulla  
bolletta di invocazione



```
int main() {  
    bolletta b1; // costruttore senza argomenti  
  
    telefonata t1(orario(9,23,12),orario(10,4,53),2121212);  
    telefonata t2(orario(11,15,4),orario(11,22,1),3131313);  
  
    b1.Aggiungi_Telefonata(t1);  
    b1.Aggiungi_Telefonata(t2);  
    cout << b1; // supponiamo di avere l'output di bolletta  
  
    bolletta b2;  
    b2 = b1;   
  
    b2.Togli_Telefonata(t1);  
    cout << b1 << b2;  
}
```

**OUTPUT**

TELEFONATE IN BOLLETTA: (NB: b1)

1) INIZIO 11:15:4 FINE 11:22:1 NUMERO 3131313

2) INIZIO 9:23:12 FINE 10:4:53 NUMERO 2121212

// dopo b2.Togli\_Telefonata(t1);

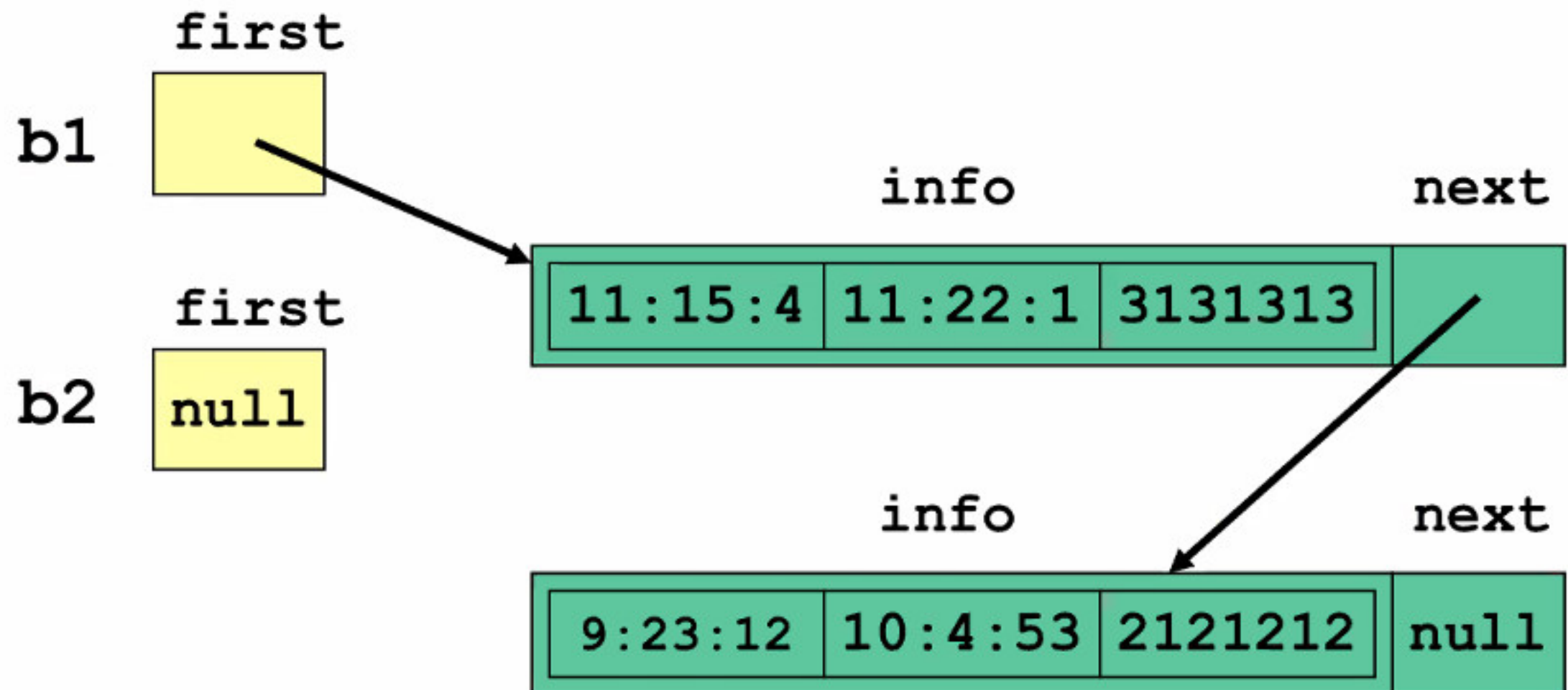
TELEFONATE IN BOLLETTA: (NB: b1)

1) INIZIO 11:15:4 FINE 11:22:1 NUMERO 3131313

TELEFONATE IN BOLLETTA: (NB: b2)

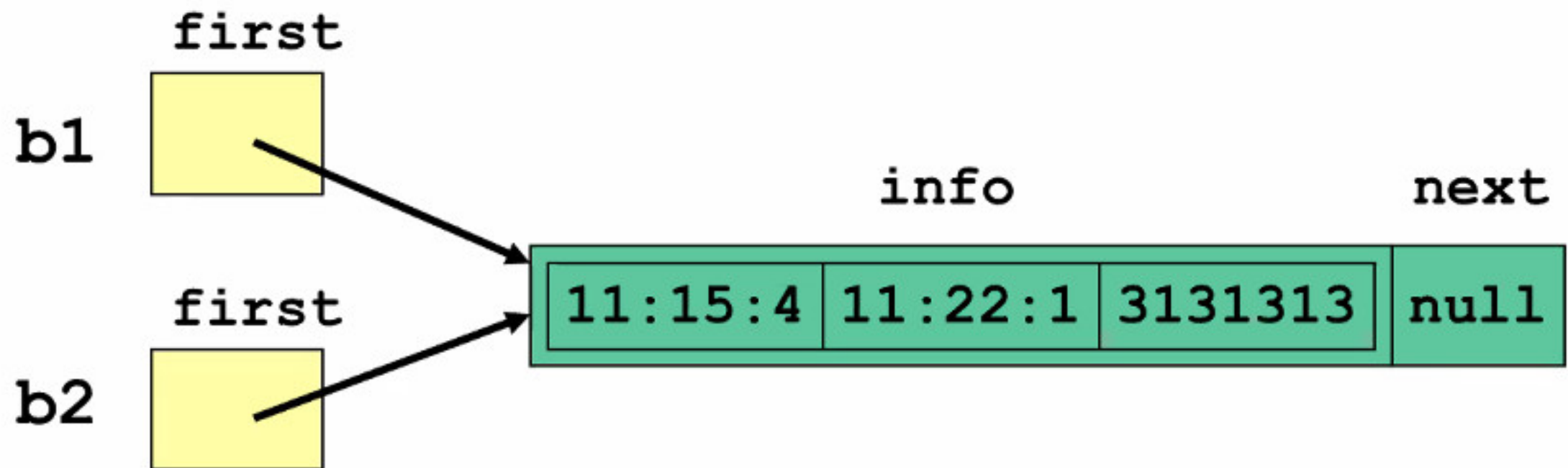
1) INIZIO 11:15:4 FINE 11:22:1 NUMERO 3131313

Situazione prima dell'assegnazione **b2=b1** ;

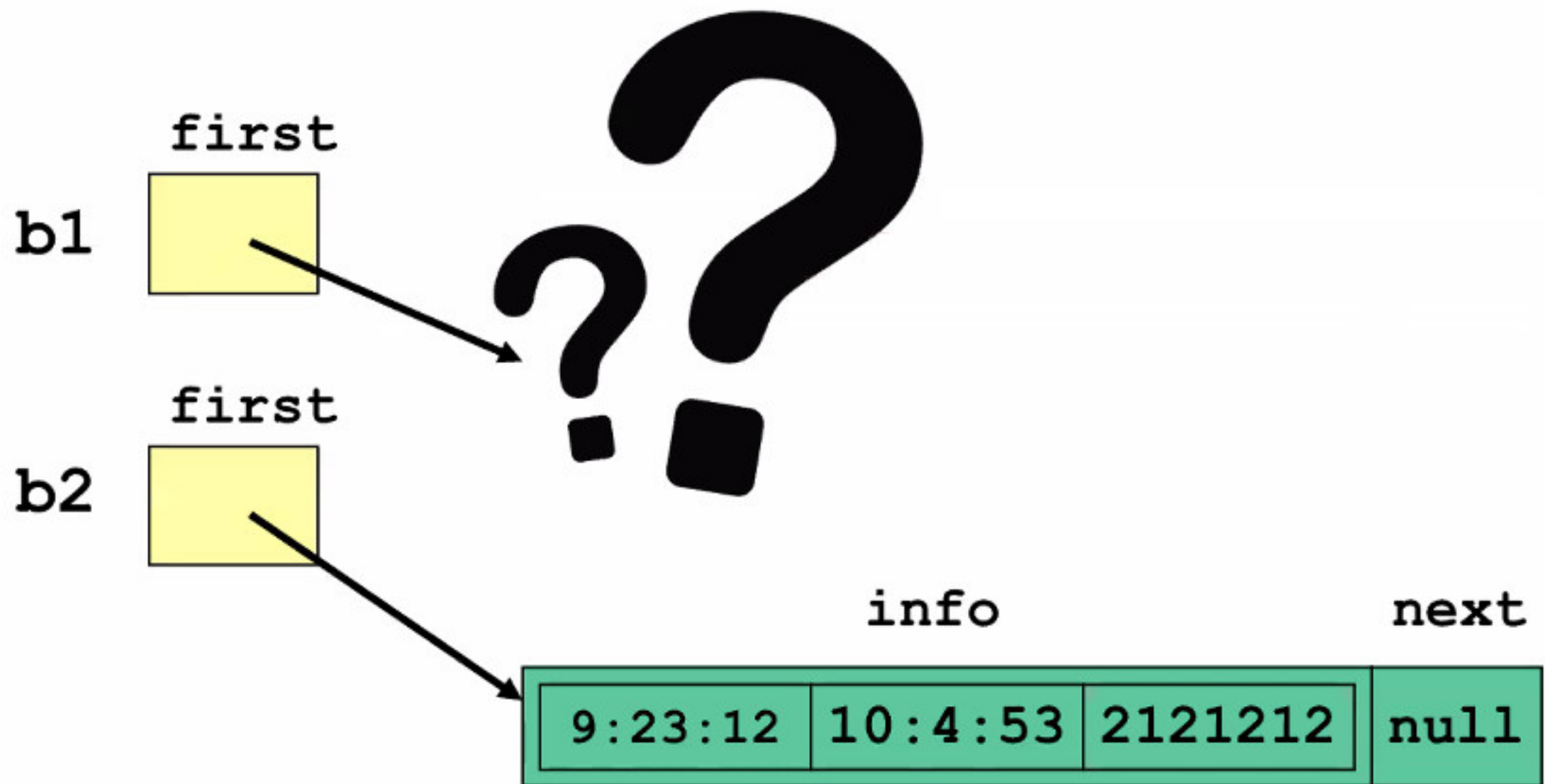




Situazione dopo `b2.Togli_Telefonata(t1) ;`



Se invece della telefonata  $t_1$  avessimo rimosso da  $b_2$  la telefonata  $t_2$  la situazione sarebbe stata anche peggiore:



Uhh, Houston we  
have a problem...  
again.

Go ahead  
student.





Fenomeno dell'**interferenza** o **aliasing**.

Due cause:

(1) Condivisione di memoria

(2) Funzioni con side effects

## Aliasing (computing)

---

From Wikipedia, the free encyclopedia

*For the method to replace command names in a computer shell, see [Alias \(command\)](#).*

In **computing**, **aliasing** describes a situation in which a data location in **memory** can be accessed through different symbolic names in the program. Thus, modifying the data through one name implicitly modifies the values associated with all aliased names, which may not be expected by the programmer. As a result, aliasing makes it particularly difficult to understand, analyze and optimize programs. **Aliasing analysers** intend to make and compute useful information for understanding aliasing in programs.



## facebook Engineering

[Open Source](#) [Platforms](#) [Infrastructure Systems](#) [Physical Infrastructure](#)

POSTED ON FEB 20, 2019 TO [ANDROID](#), [DEVELOPER TOOLS](#), [OPEN SOURCE](#)

### Open-sourcing SPARTA to make abstract interpretation easy



[Infer](#)[Docs](#)[Support](#)[Blog](#)[Twitter](#)[Facebook](#)[GitHub](#)

## A tool to detect bugs in Java and C/C++/Objective-C code before it ships

Infer is a static analysis tool - if you give Infer some Java or C/C++/Objective-C code it produces a list of potential bugs. Anyone can use Infer to intercept critical bugs before they have shipped to users, and help prevent crashes or poor performance.

[GET STARTED](#)[TRY INFER IN YOUR BROWSER](#)

★ Star

### Android and Java

Infer checks for null pointer exceptions, resource leaks, annotation reachability, missing lock guards, and concurrency race conditions in Android and Java code.

### C, C++, and iOS/Objective-C

Infer checks for null pointer dereferences, memory leaks, coding conventions and unavailable API's.

## Software Verification (LM)