10.3 – Standard Template Library (STL)

`std::vector`, `std::list`, `std::string`

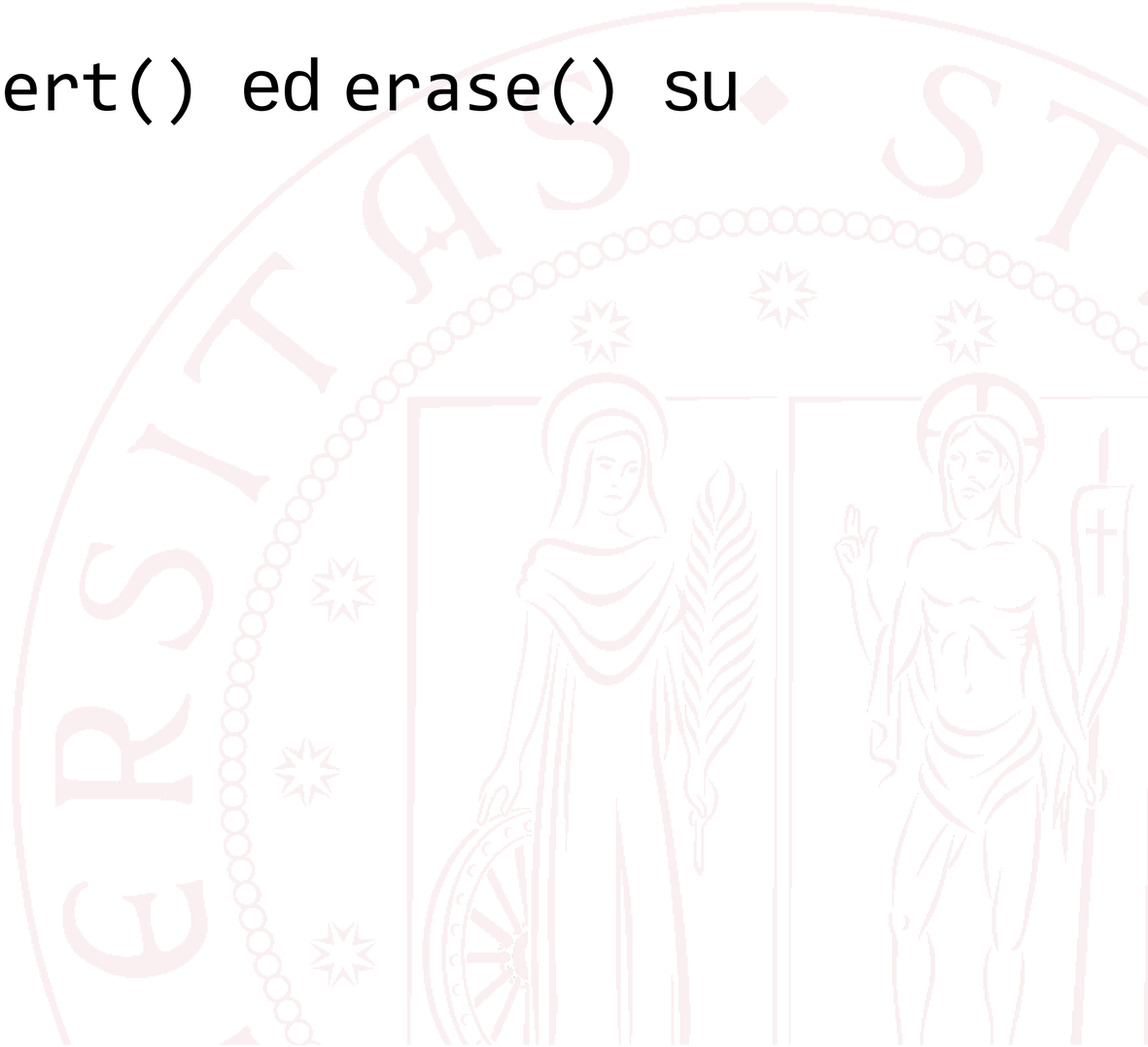
Libro di testo:

- Capitoli 20.7, 20.10

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Agenda

- Tre container a confronto
- Funzioni che invalidano gli iteratori: `insert()` ed `erase()` su `std::vector`
- Overview dei contenitori STL



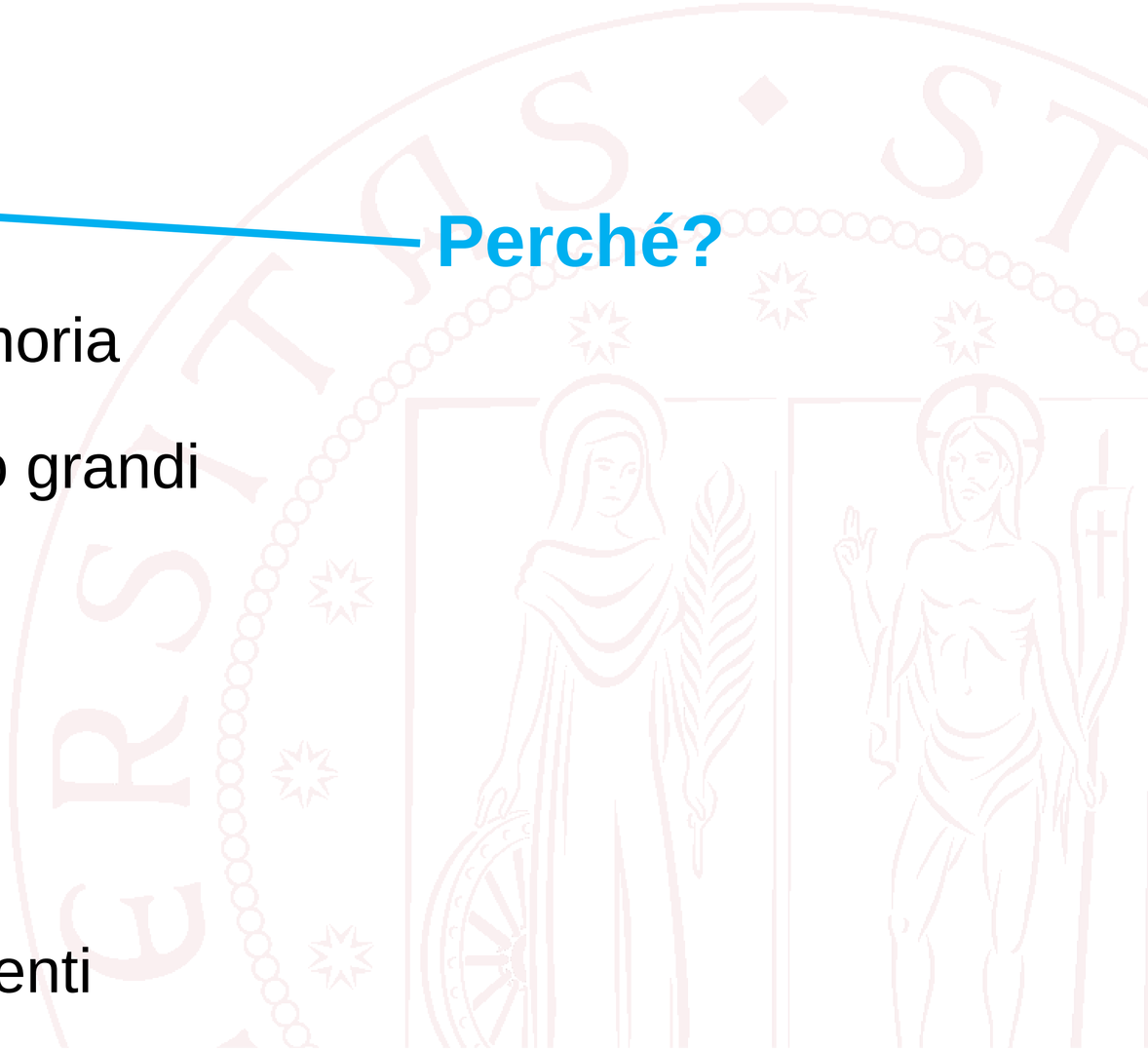
Confronto tra contenitori

- Abbiamo già usato tre contenitori STL:
  - `std::vector`
  - `std::string`
    - Confronto con `char[]`
  - `std::list`
- Analizziamo le caratteristiche principali

std::vector

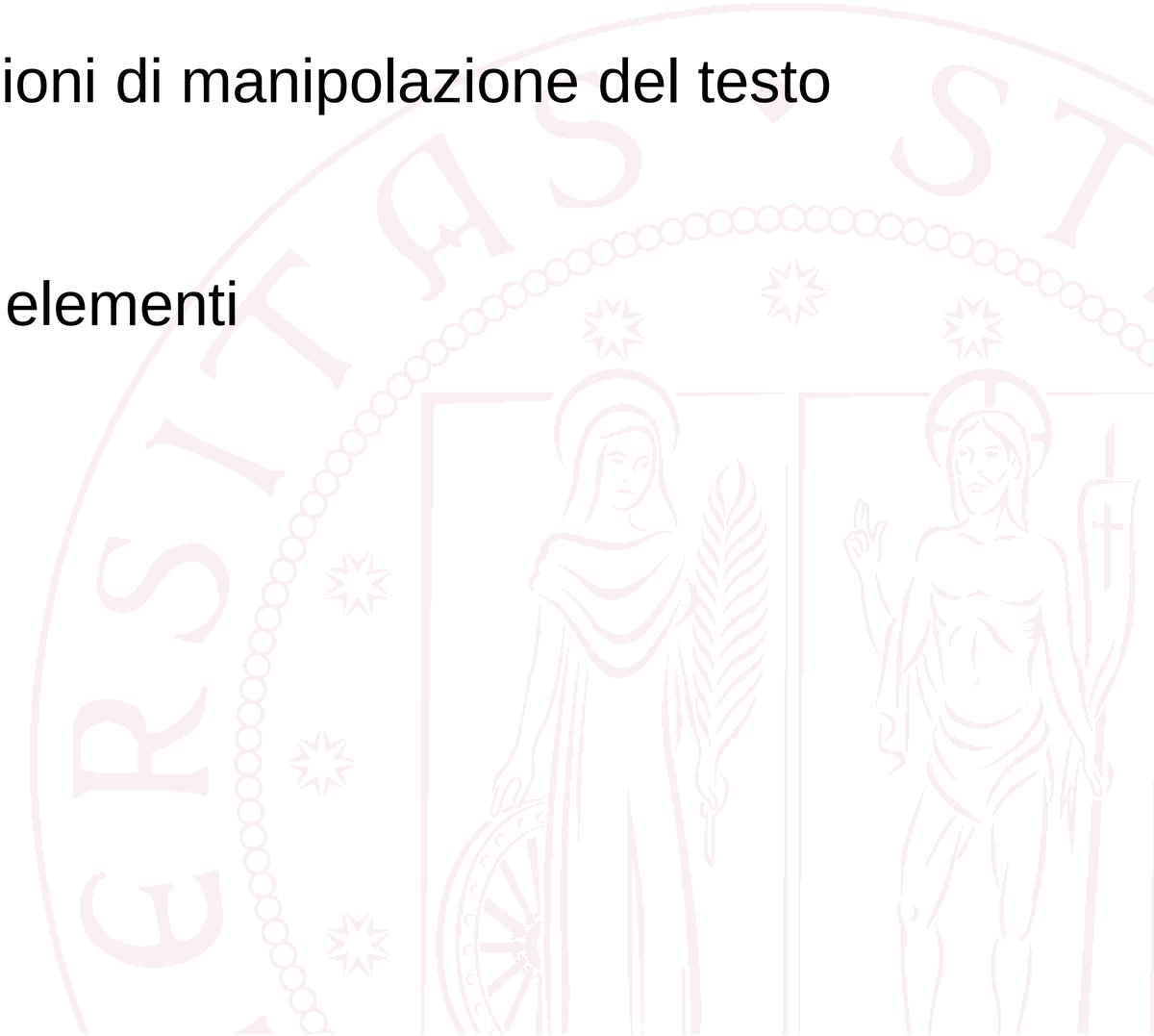
- std::vector
  - Supporta tutte le operazioni, inclusi insert() e erase()
  - Fornisce []
  - insert() ed erase() sono inefficienti
    - devono muovere molti elementi in memoria
    - un problema per collezioni di dati molto grandi
  - Fornisce range check (su richiesta!)
  - Espandibile
  - Elementi contigui in memoria
  - Funzioni di confronto confrontano gli elementi

Perché?



`std::string`

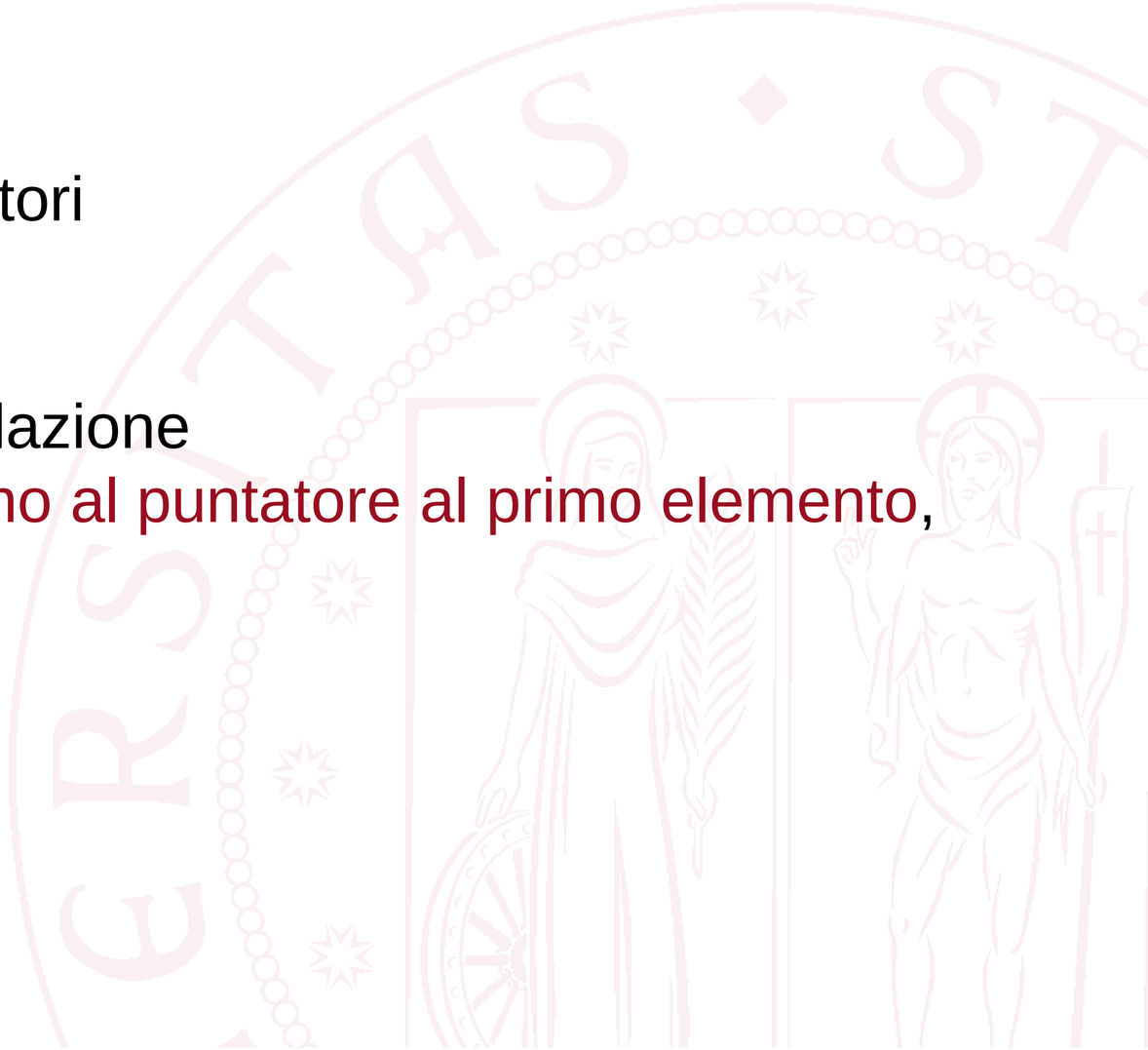
- `std::string`
  - Come i vettori, ma aggiungono le operazioni di manipolazione del testo
  - Concatenazione: `+, +=`
  - Gli operatori di confronto confrontano gli elementi



`std::string` vs. `char[]`

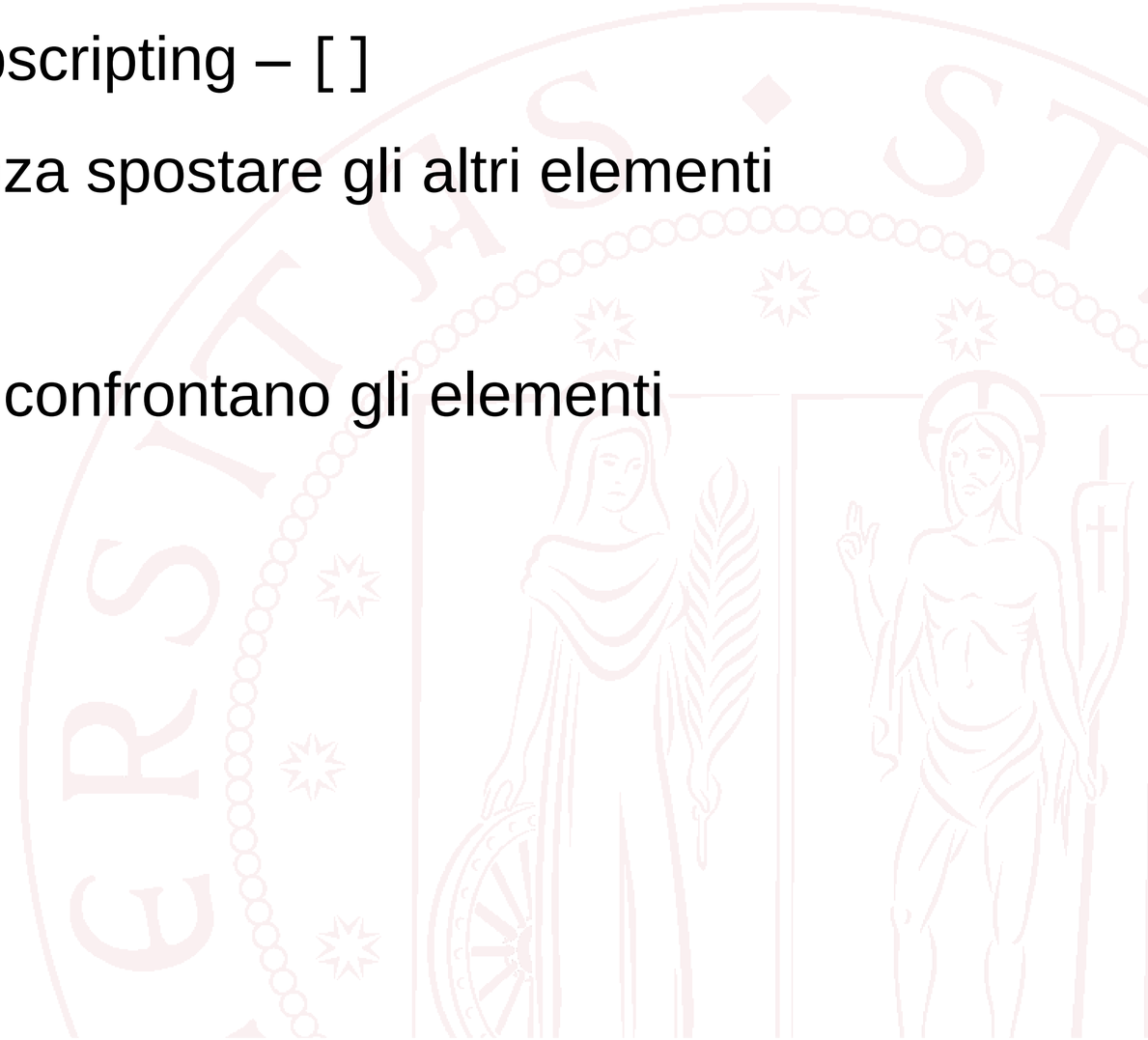
- Si noti la differenza tra `std::string` e `char[]`

- `char[]` è un array stile C, quindi:
  - Non conosce la propria dimensione
  - Non possiede `begin()`, `end()` né altri iteratori
  - Non possiede range check
  - Elementi contigui in memoria
  - Dimensione fissa, decisa a tempo di compilazione
  - Confronto (`==`, `!=`) e output (`<<`) si riferiscono al puntatore al primo elemento, non al contenuto
  - `<<` riesce a gestire il puntatore



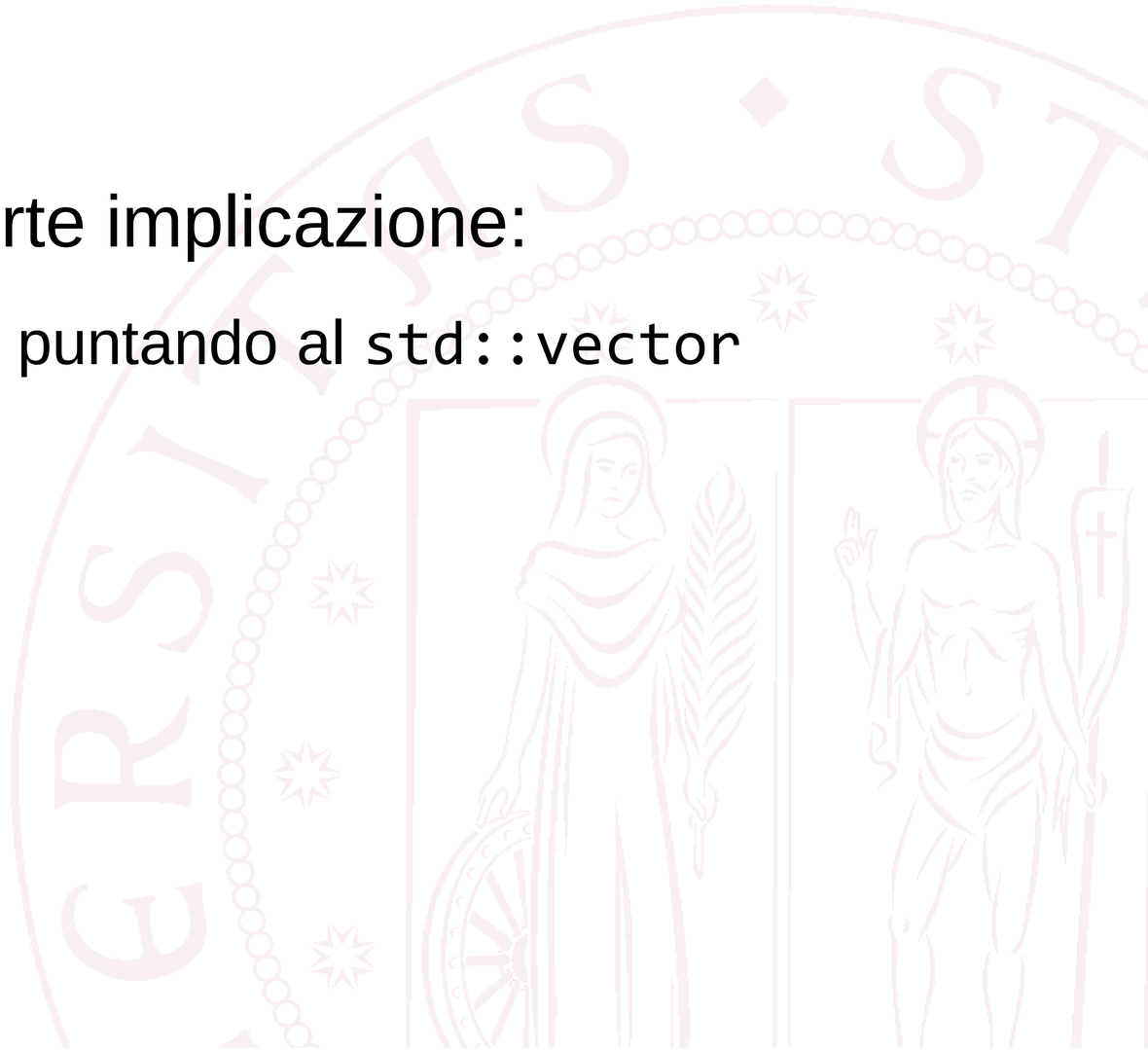
`std::list`

- `std::list`
  - Fornisce le operazioni abituali, ma non subscripting – `[]`
  - Possiamo usare `insert()` e `erase()` senza spostare gli altri elementi
  - Espandibile
  - Sono definiti gli operatori di confronto, che confrontano gli elementi



`insert()` e `erase()` con `std::vector`

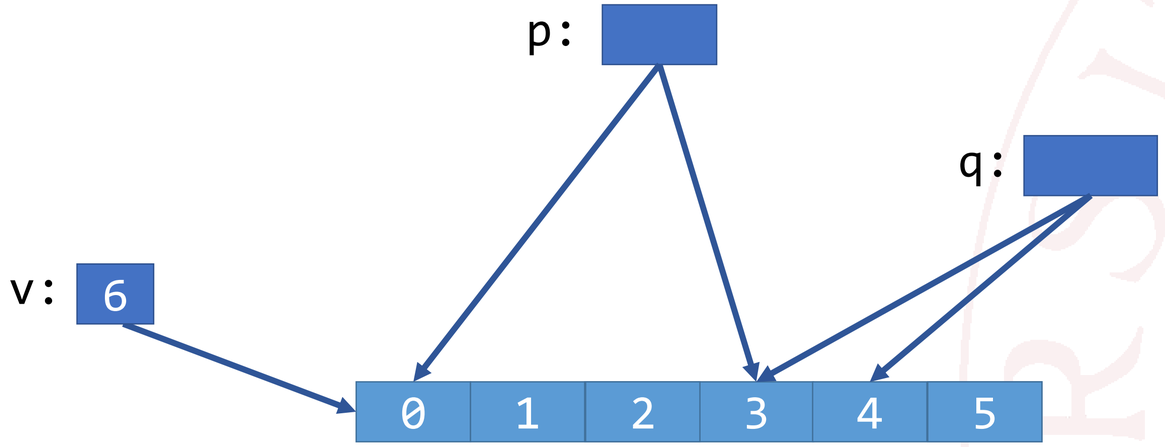
- Effettuare una `insert()` o una `erase()` in un `std::vector` può essere un'operazione inefficiente
- Inoltre, muovere gli elementi ha una forte implicazione:
  - **rende non validi gli iteratori** che stanno puntando al `std::vector`



insert() e erase() con std::vector

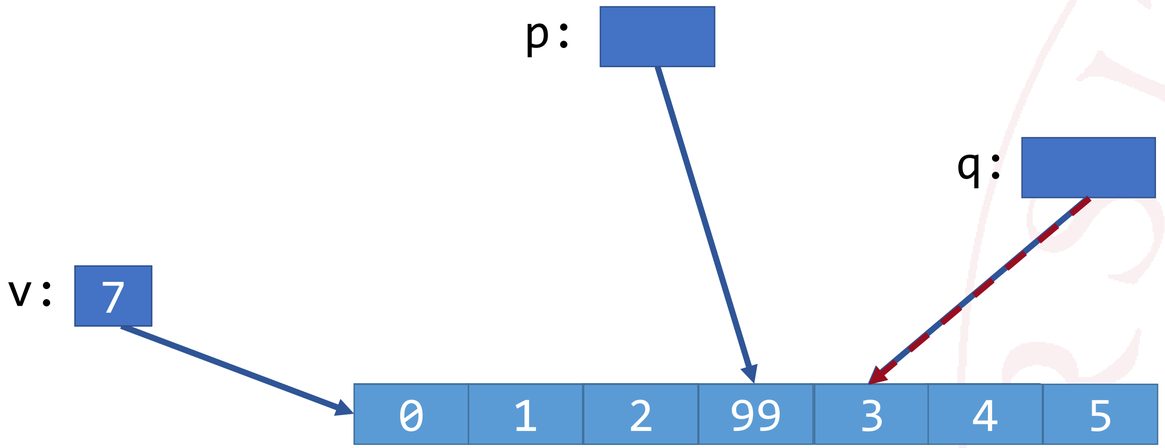
```
std::vector<int>::iterator p = v.begin();       // puntatore al primo
                                                // elemento del vettore
++p; ++p; ++p;                                  // puntatore al quarto
                                                // elemento

auto q = p;
++q;                                            // puntatore al quinto
                                                // elemento
```



insert() e erase() con std::vector

```
p = v.insert(p, 99);
```



q non è più valido

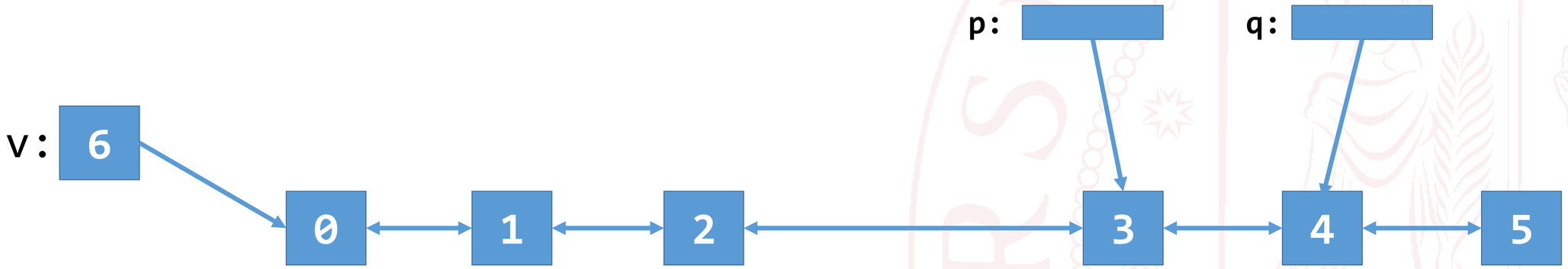
`insert()` e `erase()` con `std::list`

- Si ripropone lo stesso problema con le liste? Perché?

insert() e erase() con std::list

- Si ripropone lo stesso problema con le liste? Perché?

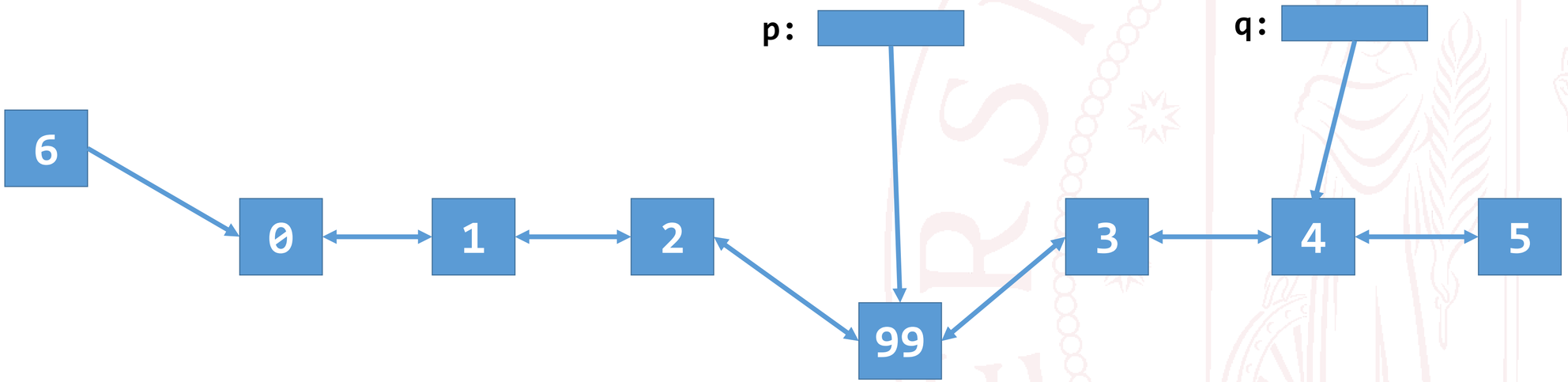
```
std::list<int>::iterator p = v.begin();
++p; ++p; ++p;
auto q = p;
++q;
```



insert() e erase() con std::list

- Si ripropone lo stesso problema con le liste? Perché?

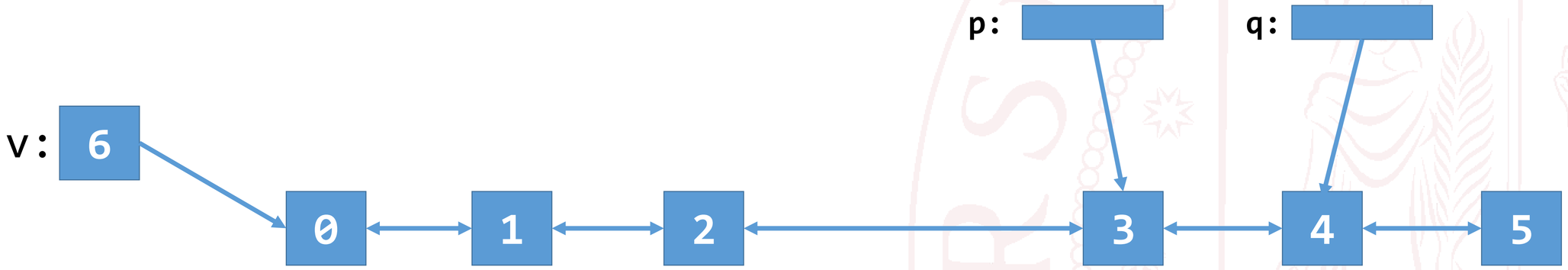
```
p = v.insert(p, 99);
```



insert() e erase() con std::list

- Si ripropone lo stesso problema con le liste? Perché?

```
p = v.erase(p);
```

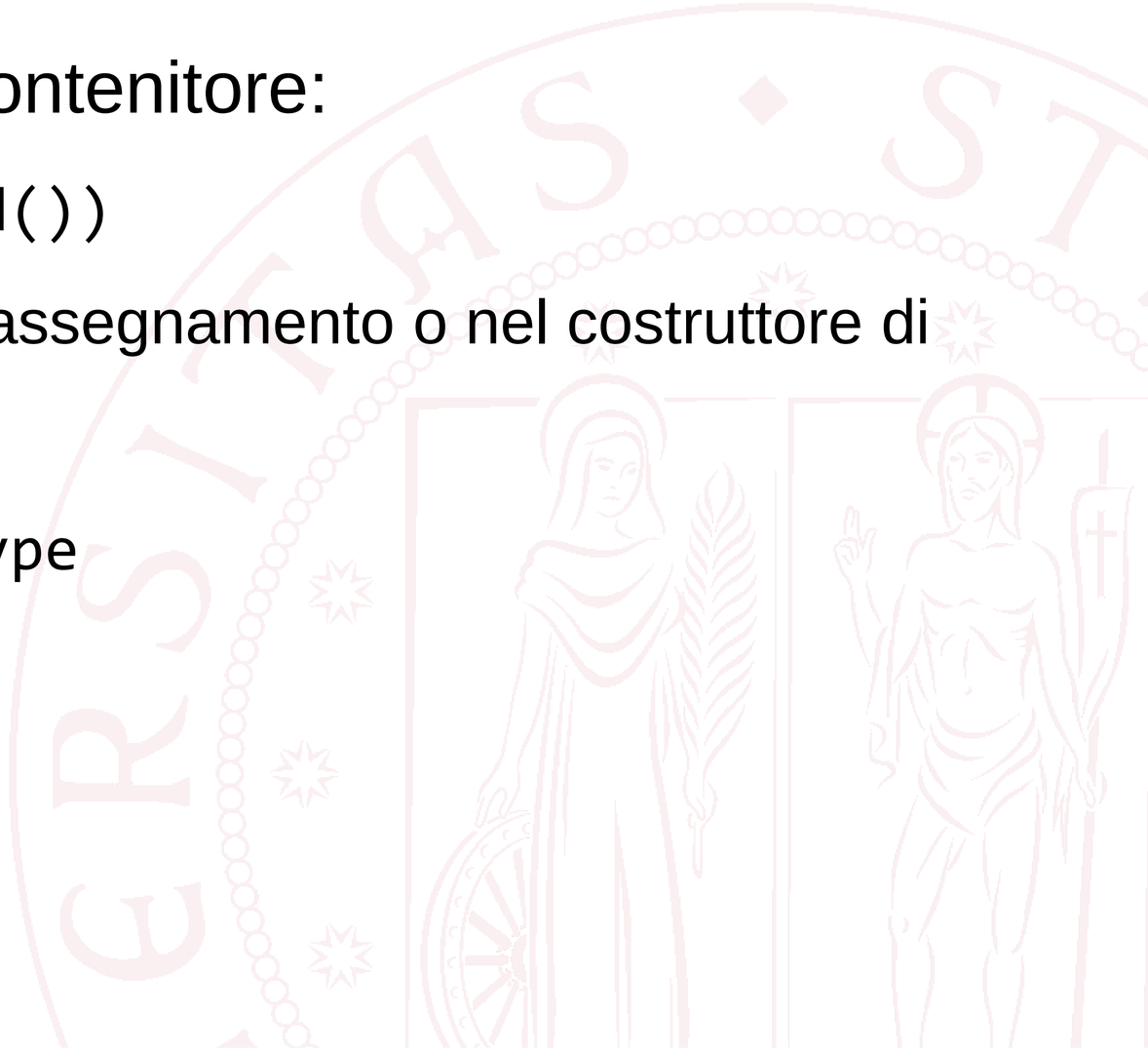


Overview dei contenitori STL

- STL fornisce molti contenitori
  - `std::vector`
  - `std::list`
  - `std::deque`
  - `std::map` (albero bilanciato e ordinato)
  - `std::multimap`
  - `std::unordered_map`
  - `std::unordered_multimap`
  - `std::set`
  - `std::multiset`
  - `std::unordered_set`
  - `std::unordered_multiset`
  - `std::array`

Contenitore STL

- Cos'è un contenitore STL?
- La definizione è complessa, però un contenitore:
  - È una sequenza di elementi `[begin():end())`
  - Supporta le operazioni di copia (usate nell'assegnamento o nel costruttore di copia)
  - Chiama il tipo dei propri elementi `value_type`



Contenitore STL

- Possiede tipi iteratore chiamati `iterator` e `const_iterator`
  - Gli iteratori forniscono `*`, `++` (pre e post), `==` e `!=`
  - Gli iteratori di `std::vector` forniscono anche `--`, `[]`, `+` e `-` (random access iterators)
  - Gli iteratori di `std::list` forniscono anche `--`
- Fornisce `insert()` ed `erase()`, `front()` e `back()`, `push_back()` e `pop_back()`, `size()`, ...
  - `std::vector` e `std::map` forniscono anche `[]`
- Forniscono operatori di confronto: `==`, `!=`, `<`, `<=`, `>`, `>=`

Recap

- Tre contenitori a confronto:
  - `std::vector`,
  - `std::string`,
  - `std::list`
- `insert()` ed `erase()` su `std::vector` – effetto sugli iteratori
- Overview dei contenitori STL
- Elementi comuni dei contenitori STL