10/7/2015

C++11 Containers `set` & `map`

Sandy Engelhardt
Medical and Biological Informatics (MBI)

dkfz.
GERMAN
CANCER RESEARCH CENTER
IN THE HELMHOLTZ ASSOCIATION

50 Years – Research for
A Life Without Cancer

Standard Containers

- `set<Key, Compare>`
- `map<Key, Value, Compare>`

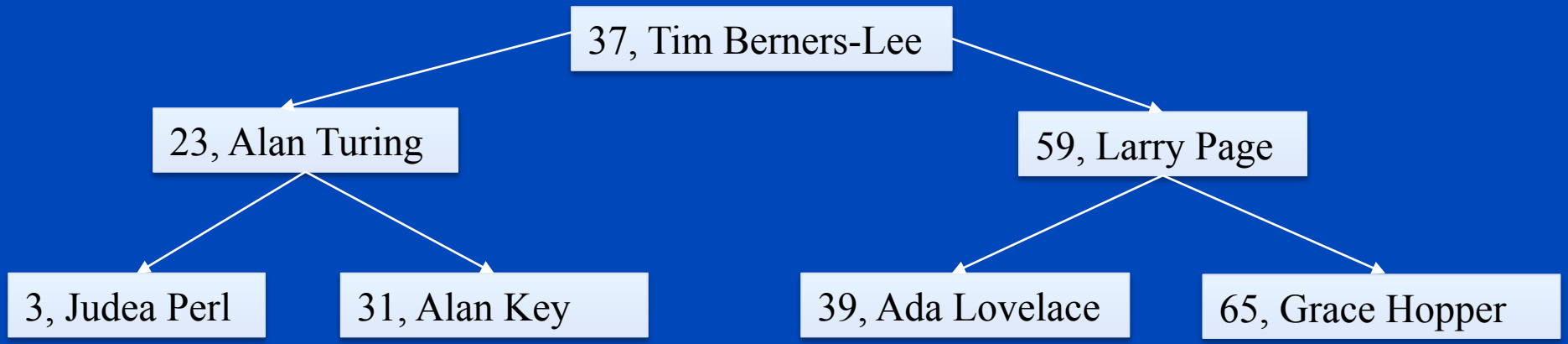
- Maintains its elements internally in a way that it …
  - allows for fast traversal in non-decending key order
  - allows for fast retrieval by `key` according to the provided `Compare` type e.g. „`operator<()`“
- whenever a **new key** is inserted into a container, the container finds a proper place for insertion so that it maintains the proper ordering of the internal data structure

Key Insertion

Whenever a **new key** is inserted into a container, the container finds a proper place for insertion so that it maintains the proper ordering of the internal data structure.

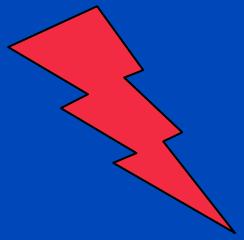
Internally usually stored as binary tree in all current standard library implementations

```
map<int, string> m;
```



Changing the Key

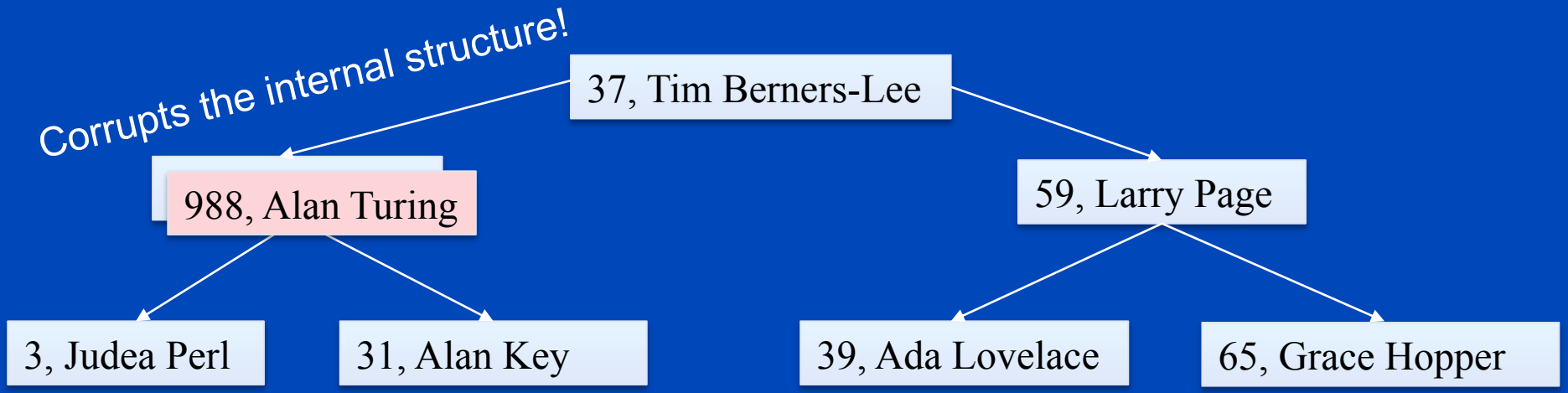
Once a key has been inserted, it should **not be changed** in a way that would change its **relative position** in the container.



- assumptions about the ordering of the entries are violated
- searches for valid entries could fail
- iterators would no longer be guaranteed to traverse the contents in key order

Example with iterator


```
map<int, string> m;
```



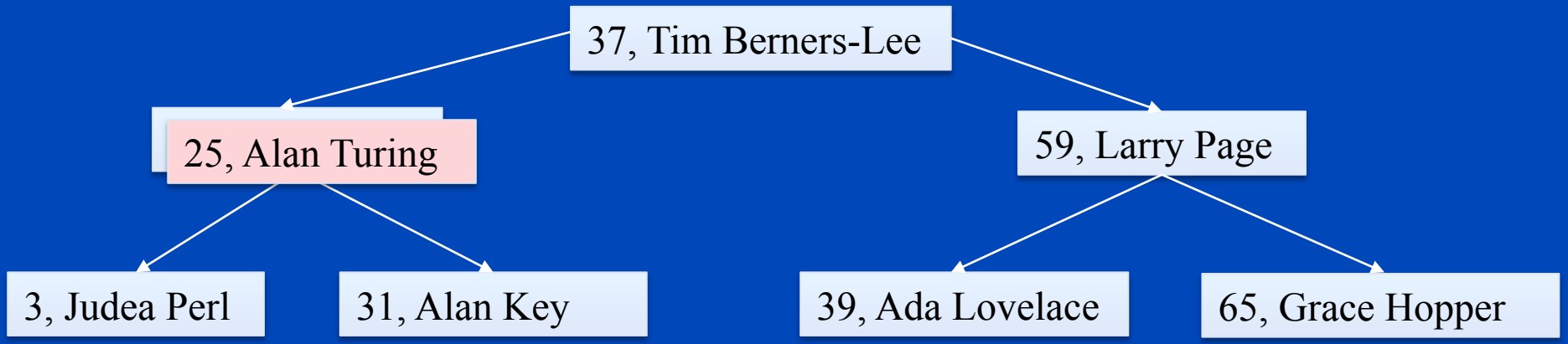

```
map<int,string>::iterator i = m.find(23);
if ( i != m.end() )
{
const_cast<int & > (i -> first) = 988;
}
```

points to a pair <const key, value>

--> a search for key 31 would fail

Example with iterator


```
map<int, string> m;
```




```
map<int,string>::iterator i = m.find(23);
if ( i != m.end() )
{
const_cast<int & > (i -> first) = 25;
}
```

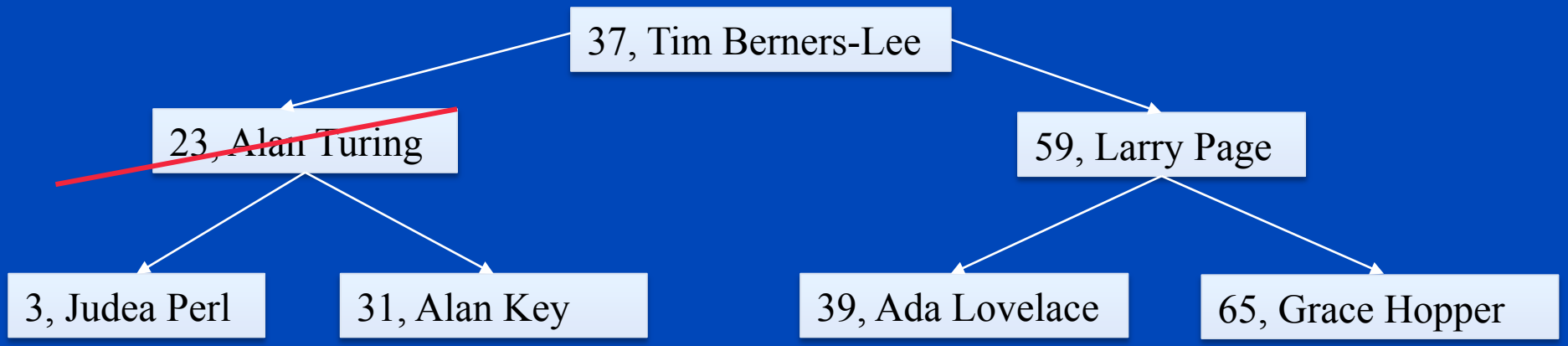
--> relative order remains unchanged

„Const Means const!“

- const_cast: C++ actively tries to prevent code that changes the relative ordering of keys
- `Keys` should not be modified at all, while `values` are of course allowed to be changed !!

Use standard methods


```
map<int,string>::iterator i = m.find(23);
if ( i != m.end() )
{
        string s = i->second;
        m.erase(i);
        m.insert( make_pair(988, s)); // better
}
```



std::set und std::map in C++ 11

- set<Key>
- map<Key, Value>

In the old standard, it was possible to change the key of a std::set , which was different in comparison to a std::map

```
map<int, int> mm{{1,1}, {6,6}, {4,4}}
auto it = ss.find(4);
it->second = 42; // ok, changes only the value
it->first = 42;  // never worked before: assignment of a
                    read-only location
```

```
set<int> ss {1,2,8,6,4};
auto it = ss.find(4);
*it = 7; // NEW in C++11 ERROR: assignment of a read-only
location
```

Thank you for your attention!

Further information on www.dkfz.de

dkfz.
GERMAN CANCER RESEARCH CENTER
IN THE HELMHOLTZ ASSOCIATION

50 Years – Research for
A Life Without Cancer

Further reading…

- Torsten T. Will, C++11 programmieren. Galileo Computing, 2012