++it Italian C++ Community

www.italiancpp.org

Perché nel 2015 parliamo ancora di C++?

Marco Arena – Meetup Pordenone, 7 Febbraio 2015

marco@italiancpp.org

Un ringraziamento speciale

- A Marco Parenzan e 

- Agli speakers di oggi
- A chi ci offre hosting gratuito 

Grazie!

++it ad oggi

- ~290 utenti iscritti
- ~20.000 visite al mese
- ~200 discussioni
- ~600 risposte

Herb Sutter

Scott Meyers

That's ++it!

Bartosz Milewski

- Social e gruppi di discussione
- Uno spazio dedicato ai neofiti
- Risorse consigliate e materiale divulgativo
- News e newsletter mensile
- Offerte di lavoro
- Meetup, Eventi e Partecipazioni
- Piattaforma per pubblicare articoli

Articoli con snippet compilabili!

In C++14 è possibile creare delle **lambda generiche** (dette anche *polimorfe*), tramite **auto**:

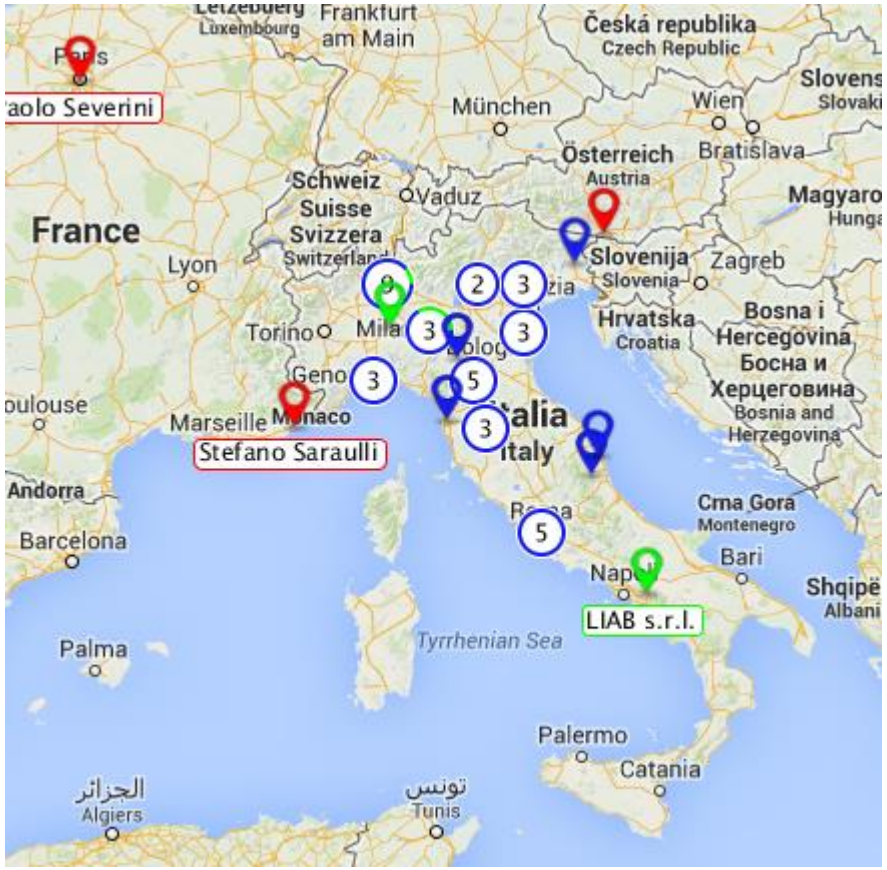
```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7     auto printer = [](auto value) {
8        cout << "PRINTING: " << value << endl;
9     };
10
11    printer(10);
12    printer("hello");
13 }
```

Esegui

```
PRINTING: 10
PRINTING: hello
```

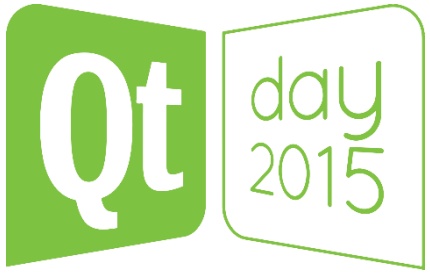
http://www.italiancpp.org/2014/02/03/una-sbirciatina-al-cpp14/

Dove sono i C++-isti italiani?



http://www.italiancpp.org/map

Collaborazioni Partecipazioni e Media Partnerships



Importante: dateci feedback!

https://joind.in/event/italiancpp-pordenone-2015

Perché nel 2015 parliamo ancora di C++?

When I hear they’re adding features to C++

http://this-plt-life.tumblr.com/post/36425239482/when-i-hear-theyre-adding-features-to-c

Nel 2015 non punterei sul C++ perché…

- "…è complicato…e poi con tutti quei templates…" – Luca, studente di Ingegneria Informatica
- "…non ha un vero equivalente dei .net framework" – Matteo, Software Engineer
- "…dov'è il mio garbage collector??" – Laura, Programmatrice Java
- "…voglio applicare gli stessi idiomi che uso in C# o Java" – Claudio, sostenitore della Campagna Anti-IF
- "…se devo andar forte faccio tutto in C" – Carlo, Programmatore Senior
- "…non è produttivo" – Gerri, Senior Architect
- …

Quindi perché nel 2015 parliamo ancora di C++?

Fonte: http://www.stroustrup.com/applications.html

TIOBE Programming Community Index

Source: www.tiobe.com

Programming Language	2014	2009	2004	1999
C	1	2	1	1
Java	2	1	2	3
Objective-C	3	26	37	-
C++	4	3	3	2
C#	5	5	8	12
PHP	6	4	5	30
Python	7	6	6	23
JavaScript	8	8	9	9
Visual Basic .NET	9	-	-	-
Perl	10	7	4	4
Pascal	15	13	76	7

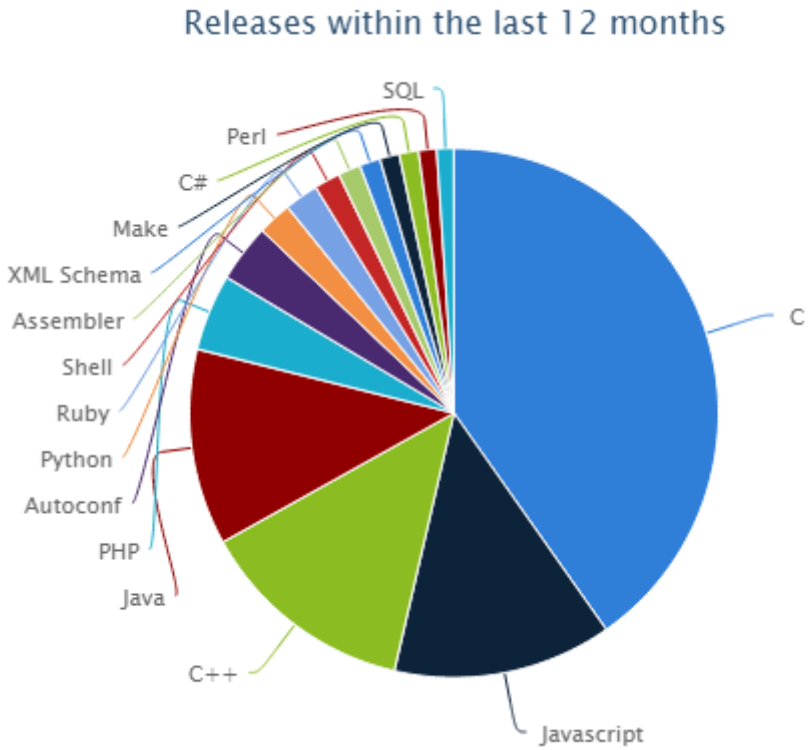
Fonte: www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Segreto #0: È popolare

Uso dei linguaggi in progetti open source attivi

Language	%
C	38.87
Javascript	12.93
C++	12.81
Java	11.57
PHP	4.53
Autoconf	3.40
Python	2.02
Ruby	1.99
Shell	1.55
Assembler	1.32
XML Schema	1.22
Make	1.15
C#	1.14

Fonte: https://www.blackducksoftware.com/resources/data/this-years-language-use



Segreto #0: È popolare

Realizzato tramite http://jibjab.com

Segreto #1: È compatibile con il C

Segreto #1: È compatibile con il C

- Per i programmatori:
  - Fondamentale nel 1983
  - Molto importante ancora oggi

Programming Language	2014	2009	2004	1999
C	1	2	1	1
Java	2	1	2	3
Objective-C	3	26	37	-
C++	4	3	3	2
C#	5	5	8	12

- Alcune differenze minori, per il resto puoi usare buona parte del C in C++

- Spesso C e C++ sono integrati nello stesso prodotto (e.g. IDE/Editor/Debuggers/)

Segreto #1: È compatibile con il C

```
int greater(const void* p, const void* q)
{
   double x = *(double*)p;
   double y = *(double*)q;
   if (x>y) return 1;
   if (x<y) return -1;
   return 0;
}

void do_my_sort(double* p, unsigned int n)
{
   qsort(p, n, sizeof(*p), greater);
}

int main()
{
   double a[500000];
   // ... fill a ...
   do_my_sort(a, sizeof(a)/sizeof(*a));
}
```

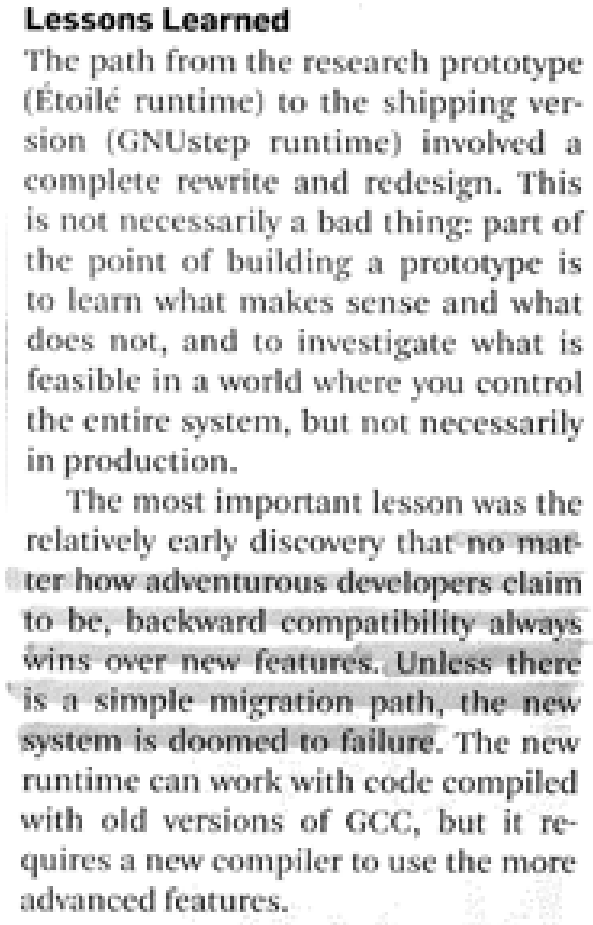
```
void do_my_sort(vector<double>& v)
{
   sort(begin(v), end(v), [](double x, double y) {
         return x>y;
   });
}

int main()
{
   vector<double> vd;
   // ... fill vd ...
   do_my_sort(vd);
}
```

Questa versione è potenzialmente più efficiente

Fonte: http://isocpp.org/blog/2014/12/myths-3

Segreto #2: Si preoccupa del passato

Segreto #2: Si preoccupa del passato



Lessons Learned

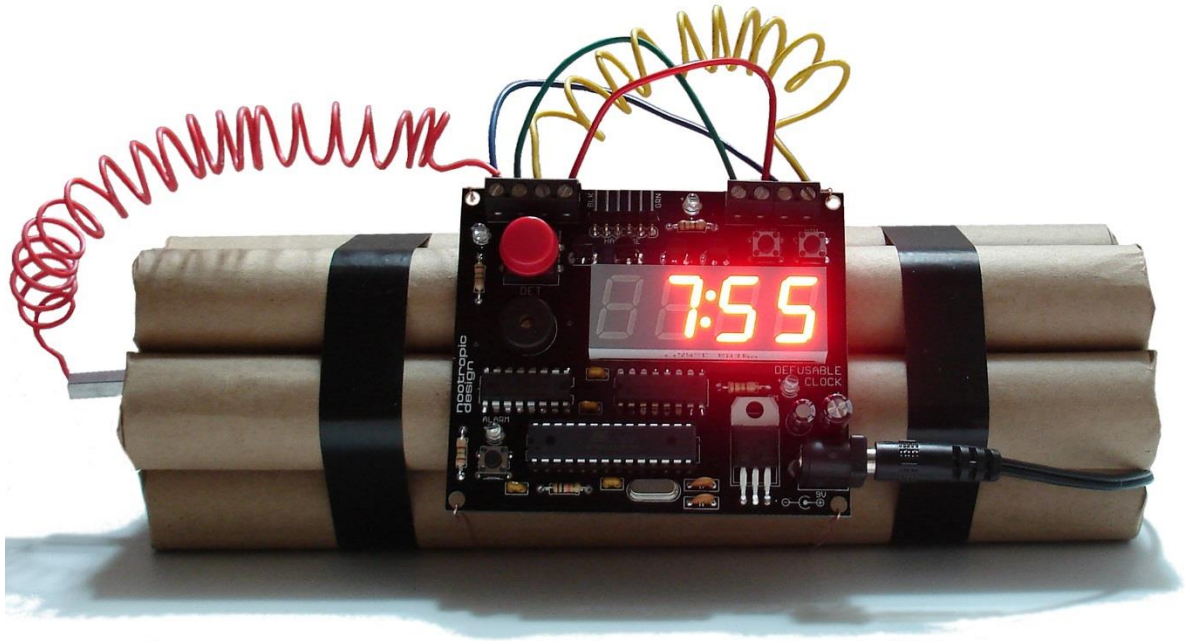
The path from the research prototype (Étoilé runtime) to the shipping version (GNUstep runtime) involved a complete rewrite and redesign. This is not necessarily a bad thing: part of the point of building a prototype is to learn what makes sense and what does not, and to investigate what is feasible in a world where you control the entire system, but not necessarily in production.

The most important lesson was the relatively early discovery that no matter how adventurous developers claim to be, backward compatibility always wins over new features. Unless there is a simple migration path, the new system is doomed to failure. The new runtime can work with code compiled with old versions of GCC, but it requires a new compiler to use the more advanced features.

Fonte: http://herbsutter.com/2012/12/04/compatibility/

"So just as C++ 'won' in the 90s because of its own strengths plus its C compatibility, C++11 is being adopted because of its own strengths plus its C++98 compatibility."

Herb Sutter



Segreto #3: Non è garbage collected

Segreto #3: Non è garbage collected

- Di default il lifetime di oggetti/variabili è **scoped** ((de)allocazione automatica)
- `~distruttore()` come meccanismo di **UNDO** general-purpose (e.g. acquisizione/rilascio di risorse – **RAII**)

```
class widget {
    gadget g;
};


void do_work() {
    auto x = …;

    auto y = …;
}
```

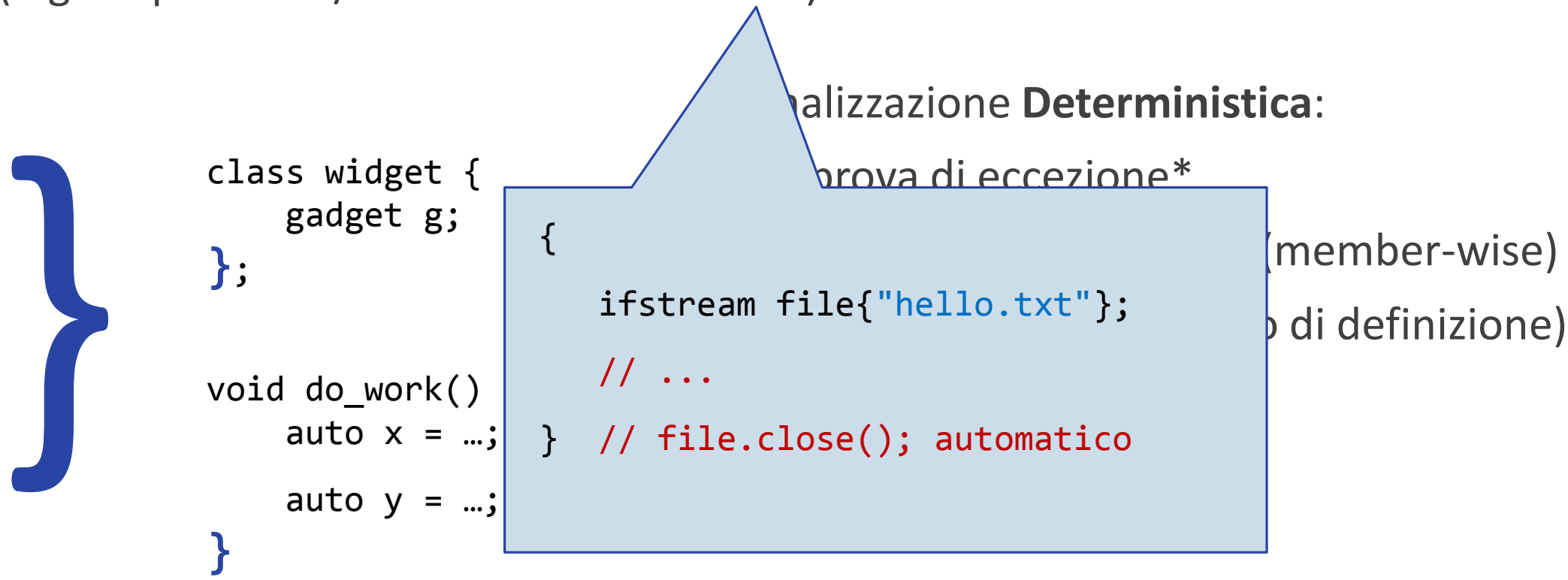
Finalizzazione **Deterministica**:

- A prova di eccezione*
- Generazione automatica (member-wise)
- In ordine (inverso a quello di definizione)

* a meno di `noexcept`

Segreto #3: Non è garbage collected

- Di default il lifetime di oggetti/variabili è **scoped** ((de)allocazione automatica)
- `~distruttore()` come meccanismo di **UNDO** general-purpose (e.g. acquisizione/rilascio di risorse – **RAII**)



* a meno di noexcept

Segreto #3: Non è garbage collected

- Di default il lifetime di oggetti/variabili è **scoped** ((de)allocazione automatica)
- `~distruttore()` come meccanismo di **UNDO** general-purpose (e.g. acquisizione/rilascio di risorse – **RAII**)

```
class widget {
    gadget g;
};

void do_work() {
    auto x = …;
    auto y = …;
}
```

Finalizzazione **Deterministica**:

- A prova di eccezione*
- Generazione automatica (member-wise)
- …izione)

```
~widget() {
    // eventualmente il tuo codice
    g.~gadget(); // automatico
}
```

* a meno di noexcept

Segreto #3: Non è garbage collected

- Di default il lifetime di oggetti/variabili è **scoped** ((de)allocazione automatica)
- `~distruttore()` come meccanismo di **UNDO** general-purpose (e.g. acquisizione/rilascio di risorse – **RAII**)

```
class widget {
    gadget g;
};

void do_work() {
    auto x = …;
    auto y = …;
}
```

```
~y();
~x();
```

Finalizzazione **Deterministica**:

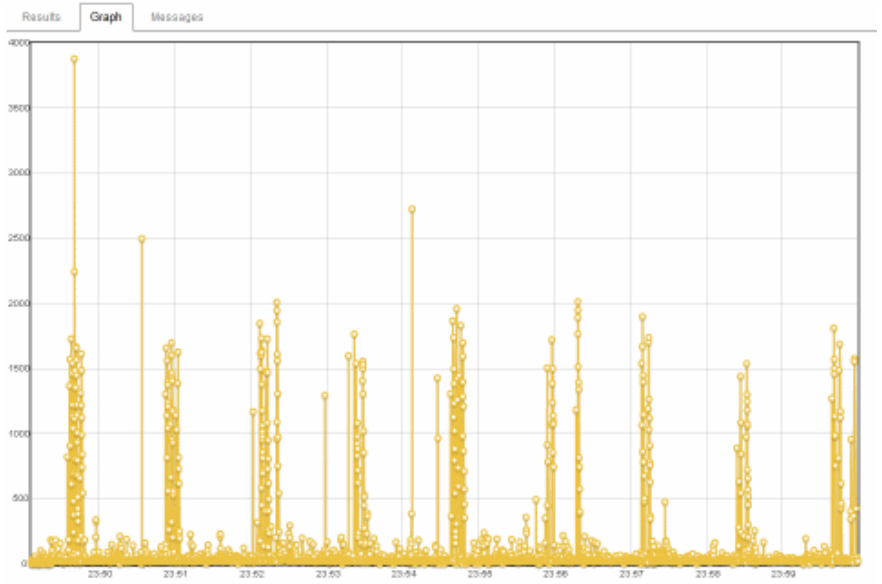
- A prova di eccezione*
- Generazione automatica (member-wise)
- In ordine (inverso a quello di definizione)

* a meno di noexcept

Segreto #3: Non è garbage collected – Una riflessione

In alcuni casi il Garbage Collector può complicarti la vita, esempi:

- Sistemi latency-critical (e.g. non vuoi che il GC salti fuori senza controllo)
- Il rilascio di risorse dev'essere deterministico (e.g. files & threads)



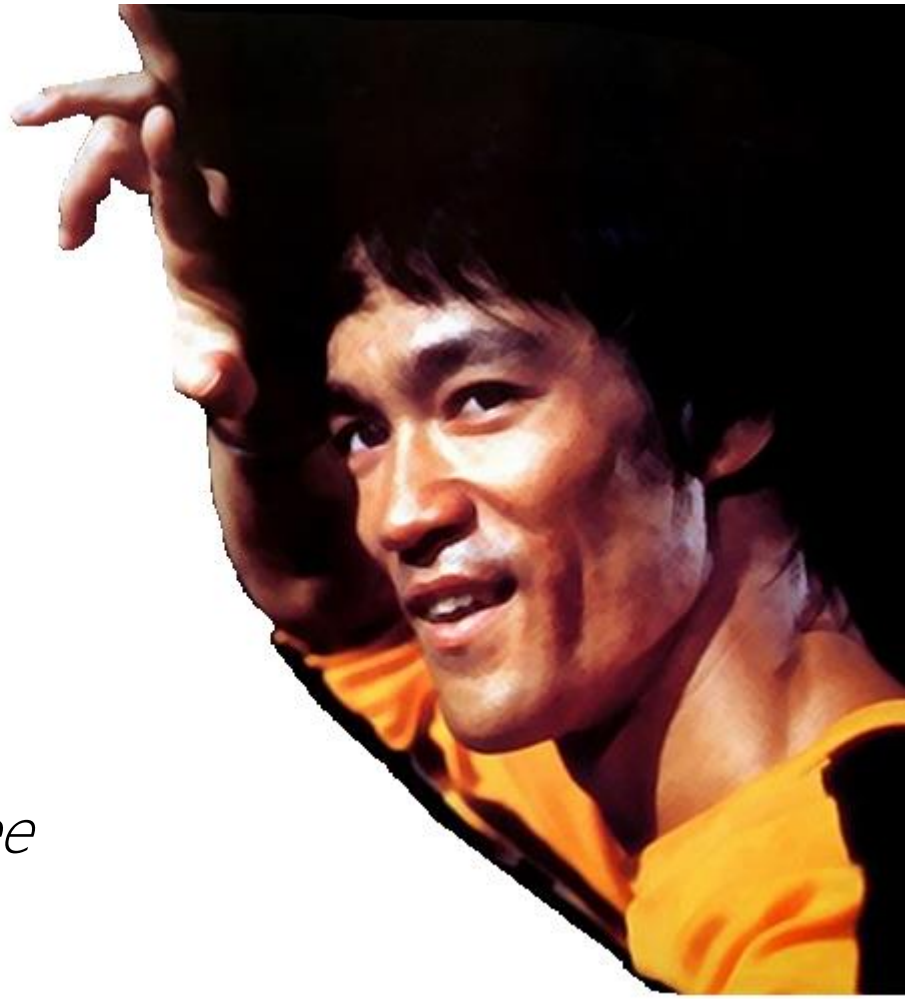
Fonte: http://tinyurl.com/42l566j

Una storia vera:
Stack Overflow vs .NET Garbage Collector

http://samsaffron.com/archive/2011/10/28/in-managed-code-we-trust-our-recent-battles-with-the-net-garbage-collector

Il miglior combattente non è un pugile, un karateka o un judoka. Il miglior combattente è qualcuno che si può adattare a qualsiasi stile di combattimento.

Bruce Lee



Segreto #4: È indipendente dal paradigma

Segreto #4: È indipendente dal paradigma

Generic

```
void rotate_and_draw(vector<Shape*>& vs, int r)
{
    for_each(vs.begin(), vs.end(), [](Shape* p) {
        p->rotate(r);
    });

    for (Shape* p : vs)
        p->draw();
}
```

Functional
(una specie ☺)

Object-Oriented

Fonte: http://isocpp.org/blog/2014/12/myths-1

Segreto #4: È indipendente dal paradigma

```
int sum = accumulate(
                    view::ints(1)
                  | view::transform([](int i){ return i*i; })
                  | view::take(10)
          , 0);
```

Fonte (range V3 di Eric Niebler): https://ericniebler.github.io/range-v3/

Segreto #4: È indipendente dal paradigma – *Anti-IF in C#*

```
public void CatchExceptions_IF()
{
   var promocode = new PromocodeStatusIF();
   try
   {
      promocode.Apply("g128g7d2g");
   }
   catch (AlreadyUsedPromocodeException)
   {
      Assert.Pass("Already used");
   }
   catch (ExpiredPromocodeException)
   {
      Assert.Pass("Expired");
   }
   catch (NotValidPromocodeException)
   {
      Assert.Pass("Not valid");
   }
   Assert.Fail("no exception");
}
```

```
public void RemoveCatchExceptionsAndUseMessages_NOIF()
{
   var promocode = new PromocodeStatus();
   promocode
      .AlreadyUsed(() = > Assert.Pass("Already used"))
      .Expired(() = > Assert.Pass("Expired"))
      .NotValid(() = > Assert.Pass("Not valid"))
      .Apply("g128g7d2g");

   Assert.Fail("cannot handle this promocode");
}

int main()
{
    PromocodeStatus{}
      .AlreadyUsed([] { Assert::Pass("Already used"); })
      .Expired([] { Assert::Pass("Expired"); })
      .NotValid([] {Assert::Pass("NotValid"); })
      .Apply("g128g7d2g");

    Assert::Fail("cannot handle this promocode");
}
```

Fonte: http://www.agileday.it/front/sessioni-2014/#iop_vs_oop

Segreto #4: È indipendente dal paradigma – *Anti-IF in C#*

```
public class PromocodeStatus
{
   private Action alreadyUsed = () = > { };
   private Action expired = () = > { };
   private Action notValid = () = > { };

   public PromocodeStatus AlreadyUsed(Action action) {
      alreadyUsed = action;
      return this;
   }

   public PromocodeStatus Expired(Action action) {
      expired = action;
      return this;
   }

   public PromocodeStatus NotValid(Action action) {
      notValid = action;
      return this;
   }

   public void Apply(string promocode) {
      expired();
   }
}
```

Fonte: http://www.agileday.it/front/sessioni-2014/#iop_vs_oop

```
using Action = function<void()>;
class PromocodeStatus
{
   Action alreadyUsed = []{};
   Action expired = []{};
   Action notValid = []{};

public:
   PromocodeStatus& AlreadyUsed(Action action) {
      alreadyUsed = action;
      return *this;
   }

   PromocodeStatus& Expired(Action action) {
      expired = action;
      return *this;
   }

   PromocodeStatus& NotValid(Action action) {
      notValid = action;
      return *this;
}

   void Apply(const string& promocode) {
      expired();
   }
};
```

Fonte: http://www.italiancpp.org/2014/11/23/anti-if-idioms-in-cpp/

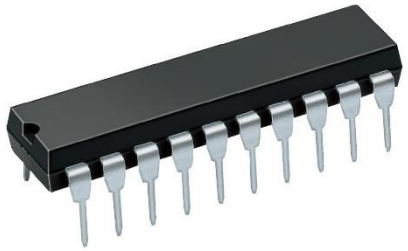
Segreto #4: È indipendente dal paradigma – *Go's defer*

```
// go
func SomeFuntion()
{
    defer func() {
        Log("SomeFunction executed")
    }
    ... Anything ...
}
```

Segreto #4: È indipendente dal paradigma – *Go's defer*

```
// C++
void SomeFuntion()
{
    defer d{ [](){
        Log("SomeFunction executed");
    }};
    ... Anything ...
}
```

Implementazione: http://www.italiancpp.org/2013/07/16/defer-con-raii-lambda/



Segreto #5: È adatto alla Systems Programming

Segreto #5: È adatto alla Systems Programming

Di default:

- Buone/ottime prestazioni ("*You don’t pay (at runtime) for what you don’t use*" - **0-overhead principle**)
- Value Types (e.g. allocazione su stack) & move semantics
- Finalizzazione deterministica
- Contiguità
- Cross-platform

Segreto #5: È adatto alla Systems Programming

Di default:

- Buone/ottime prestazioni ("*You don't pay (at runtime) for what you don't use*" - **0-overhead principle**)
- Value Types (e.g. allocazione su stack) & move semantics
- Finalizzazione deterministica
- Contiguità
- Cross-platform

```
vector<int> make_vector() {
   vector<int> v1 = {1,2,3};
   auto v2 = v1; // (deep) copy
   return v1+v2; // move
}
```

Segreto #5: È adatto alla Systems Programming

Di default:

- Buone/ottime prestazioni ("*You don't pay (at runtime) for what you don't use*" - **0-overhead principle**)
- Value Types (e.g. allocazione su stack) & move semantics
- Finalizzazione deterministica
- Contiguità
- Cross-platform

Se serve, controllo efficiente e completo di:

- Lifetime dei tuoi oggetti (copy, move, …)
- Hardware e risorse del SO (non sempre portabile)
- Data Layout (PODs, …)
- Program size
- "Fine tuning"

Segreto #5: È adatto alla Systems Programming

```
sort(array<unique_ptr<T>>)
```

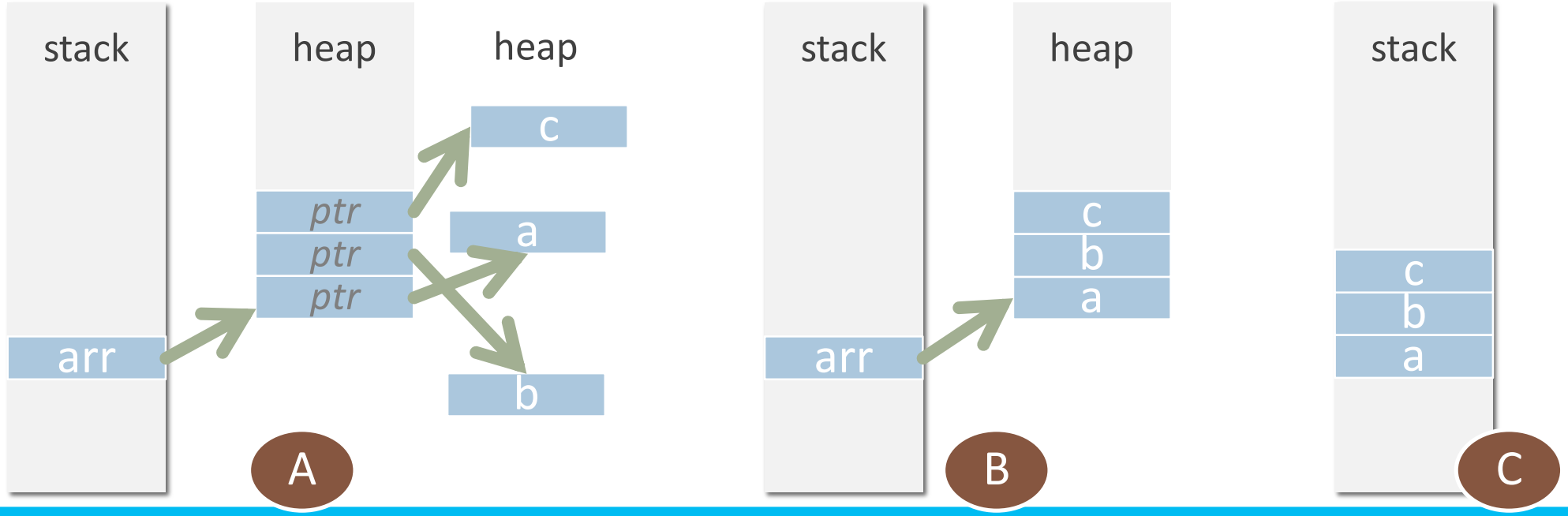
vs

```
sort(array<T>))
```

Il primo è circa **3** volte più **lento** del secondo *

* Per T molto grandi potrebbe non essere così evidente

Segreto #5: È adatto alla Systems Programming – Contiguità

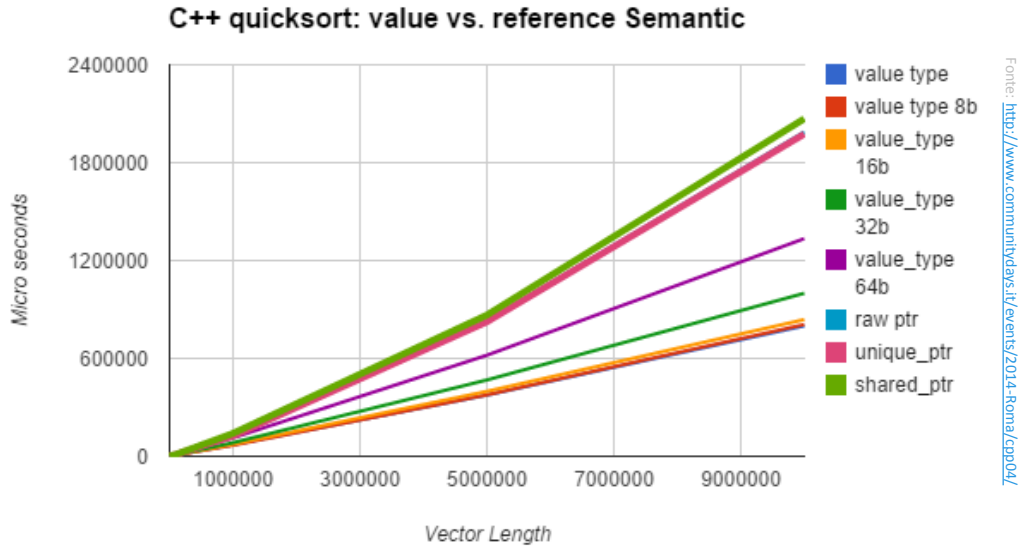


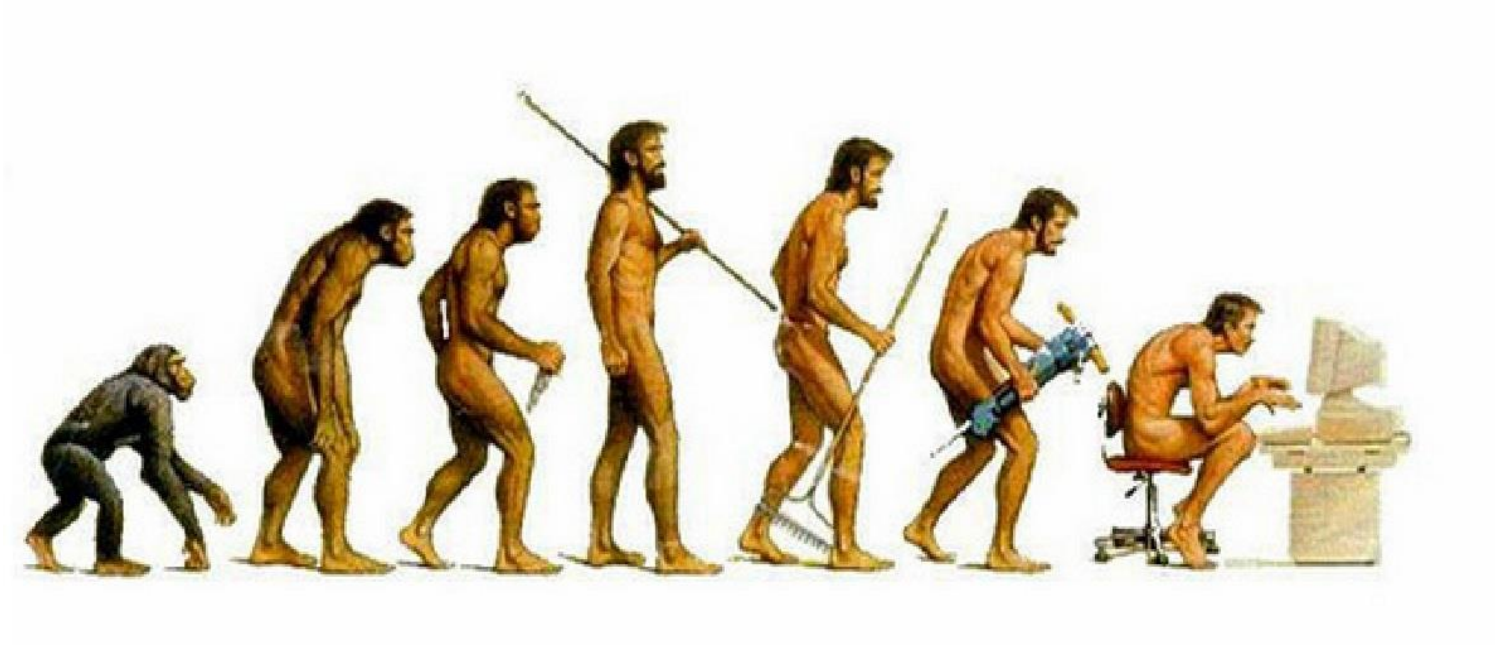
	Contents are not contiguous	Contents contiguous	Contents contiguous
Java/C#	class widget { … } **var arr = new ArrayList();** arr.Add(new widget()); *// etc.*	*// not possible for class types*	*// not possible for class types*
C++	class widget { … }; **vector<unique_ptr<widget>> arr;** arr.push_back(make_unique<widget>()); *// etc.*	class widget { … }; **vector<widget> arr{ a, b, c };**	class widget { … }; **array<widget,N> arr{ a, b, c };**

Fonte: http://channel9.msdn.com/Events/Build/2014/2-661

Segreto #5: È adatto alla Systems Programming – Contiguità

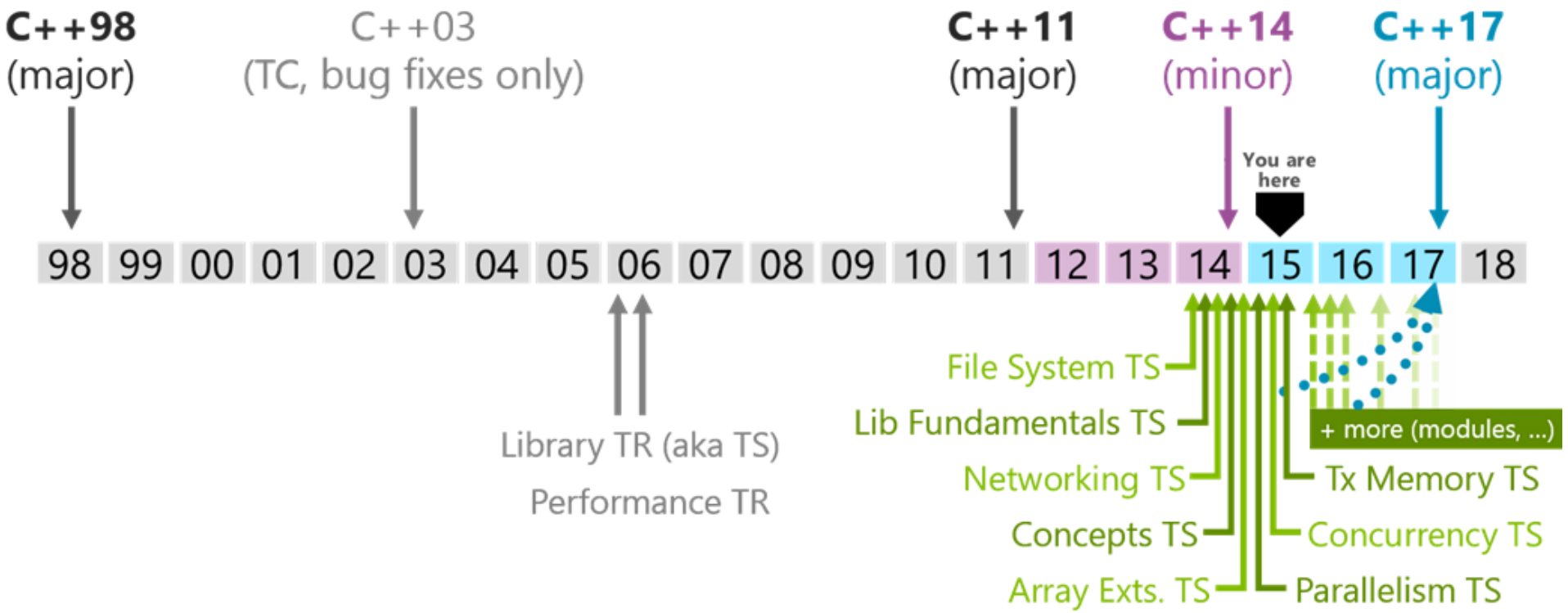
- *Quicksort* in C++:
  - `time(sort(array<unique_ptr<T>>)) ~= 3 * time(sort(array<T>))`
- Lo stesso e' vero in C# per il sort di un array di class T rispetto ad un array di structs.
  - I tempi di `Array.Sort<T>(vec)` cambiano di un fattore ~= **3.5**
- La differenza è dovuta sia all'**indirezione** che all'**aumento di cache misses**.
  - Cache prefetching non funziona bene con un pattern di accesso random alla memoria.



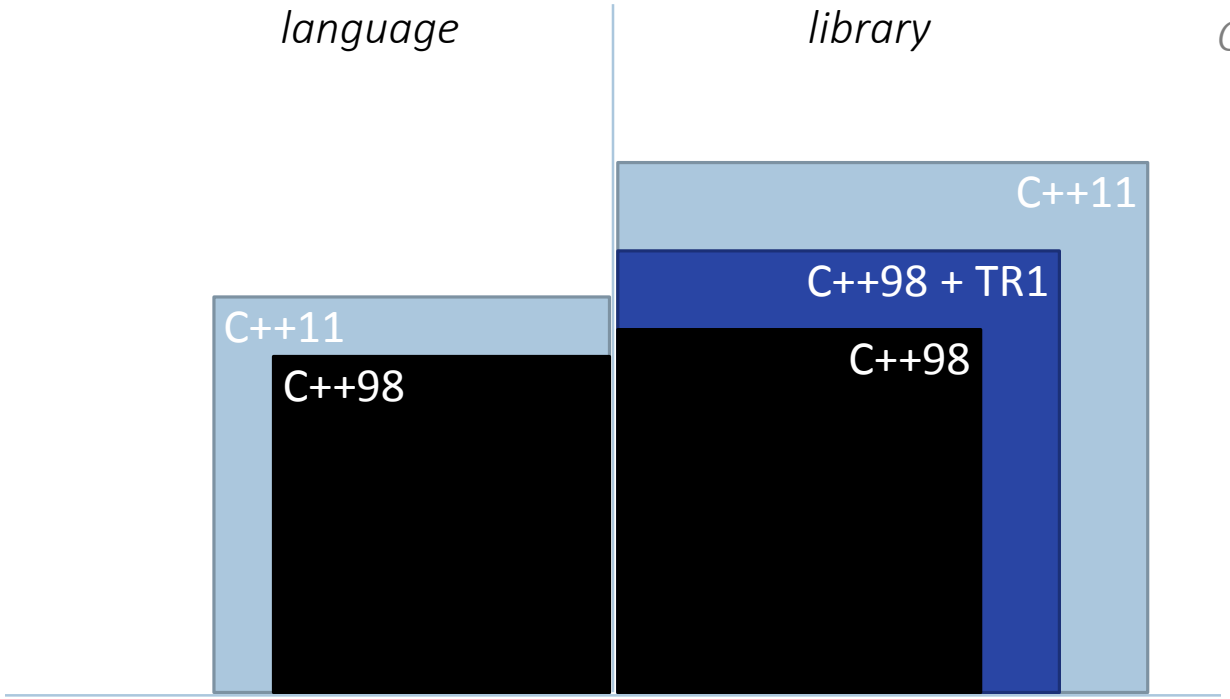


Segreto #6: Sta crescendo in fretta

Segreto #6: Sta crescendo in fretta



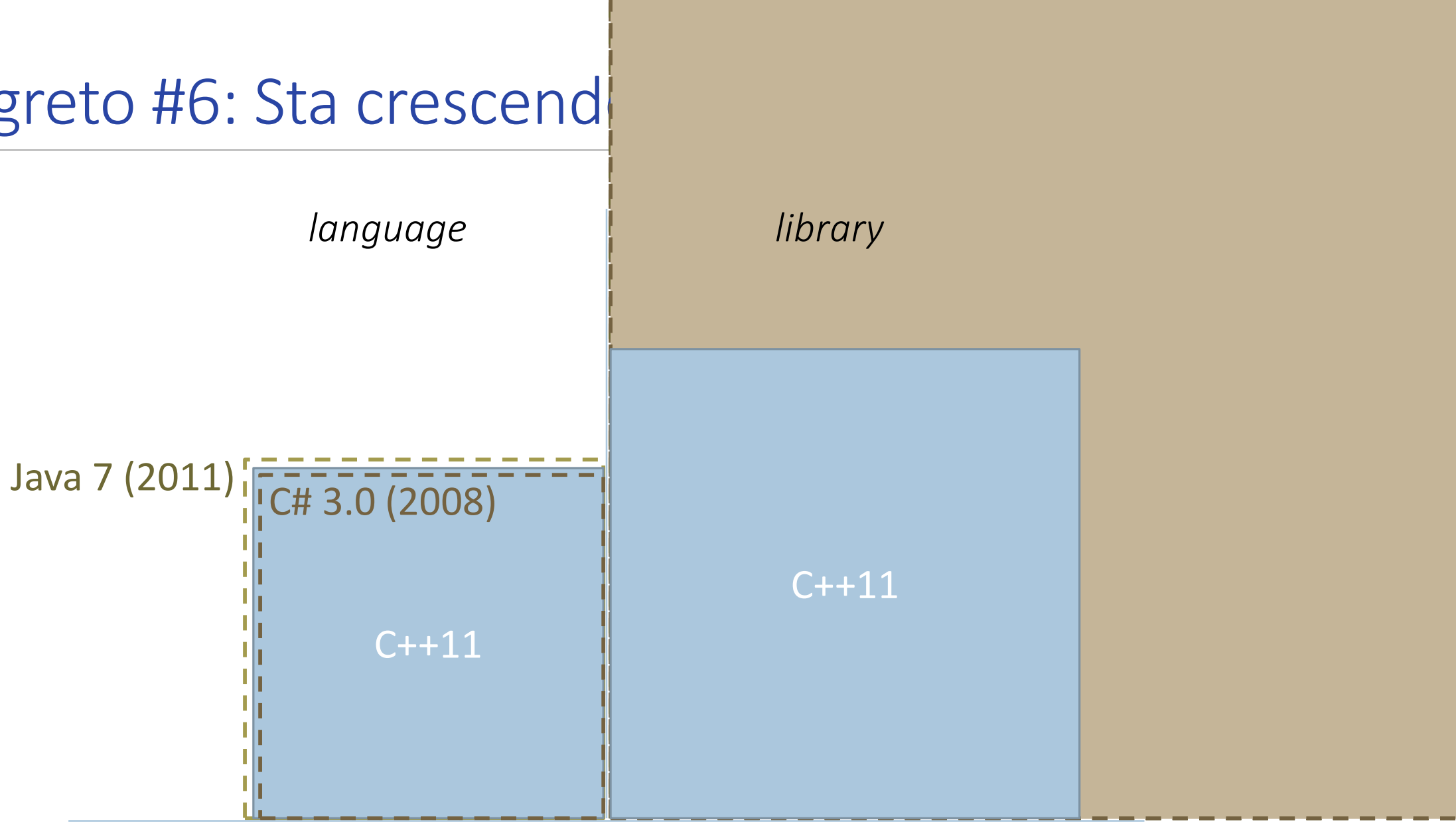
Segreto #6: Sta crescendo in fretta



Comparazione in #pagine delle specifiche

Fonte: http://channel9.msdn.com/Events/GoingNative/GoingNative-2012/C-11-VC-11-and-Beyond

Segreto #6: Sta crescend



Segreto #6: Sta crescendo in fretta

…per questo, il comitato vuole definire delle ***Portable C++ Libraries (PCL)***

Fonte: http://channel9.msdn.com/Events/GoingNative/GoingNative-2012/C-11-VC-11-and-Beyond

Segreto #6: Sta crescendo in fretta = Semplificazione

Simplification through added complexity

```
vector<Foo> v;

for(vector<Foo>::iterator it = v.begin(); it != v.end(); ++it) { // C++98
   // use *it
}

for (auto it = begin(v); it != end(v); ++it) { // C++11
   // use *it
}

for(const auto& i : v) { // Modern C++
   // use i
}
```

Segreto #6: Sta crescendo in fretta – A proposito di complessità

```
template <class _CharT, class _Traits>
basic_ostream<_CharT, _Traits>&
basic_ostream<_CharT, _Traits>::operator<<(bool __n)
{...}

...

template<class _CharT, class _Traits>
basic_ostream<_CharT, _Traits>&
operator<<(basic_ostream<_CharT, _Traits>& __os, _CharT __c)
{
    return _VSTD::__put_character_sequence(__os, &__c, 1);
}

...

template <class _Stream, class _Tp>
inline _LIBCPP_INLINE_VISIBILITY
typename enable_if
<
    !is_lvalue_reference<_Stream>::value &&
    is_base_of<ios_base, _Stream>::value,
    _Stream&&
>::type
operator<<(_Stream&& __os, const _Tp& __x)
{
    __os << __x;
    return _VSTD::move(__os);
}

...
```

Fonte: http://llvm.org/svn/llvm-project/libcxx/trunk/include/ostream

Segreto #6: Sta crescendo in fretta – A proposito di complessità

```
template <class _CharT, class _Traits>
basic_ostream<_CharT, _Traits>&
basic_ostream<_CharT, _Traits>::operator<<(bool __n)
{...}

...

template<class _CharT, class _Trait
basic_ostream<_CharT, _Traits>&
operator<<(basic_ostream<_CharT, _T
{
    return _VSTD::__put_character_s
}

...

template <class _Str                    Tr
inline _LIBCPP_INLINE_
typename enable_if
<
    !is_lvalue_reference<_Strea
    is_base_of<ios_base, _Stream>
    _Stream&&
>::type
operator<<(_Stream&& __os, const _T
{
    __os << __x;
    return _VSTD::move(__os);
}

...
```

Fonte: http://llvm.org/svn/llvm-project/libcxx/trunk/include/ostream

```
cout << "hello" << endl;
```

operator<<

Overload + Template + Specializzazioni (*in più scopes*)

Il C++ fornisce tutti gli strumenti per incapsulare/nascondere complessità

Segreto #6: Sta crescendo in fretta – A proposito di complessità

```
class CarSettings {
public:
   CarSettings() : someFlag (false) , coeffs ()
   {
   }

   CarSettings(const CarSettings& other)
      : someFlag (other.someFlag),
        coeffs(other.coeffs)
   {

   }
   // ... resto della classe
private:
   bool someFlag;
   double coeffs[SOME_MAGIC_DEFINE];
};
```

Segreto #6: Sta crescendo in fretta – A proposito di complessità

```
class CarSettings {
public:
   CarSettings() : someFlag (false) , coeffs ()
   {
   }

   CarSettings(const CarSettings& other)
      : someFlag (other.someFlag)
   {
      memcpy(coeffs, other.coeffs, sizeof(coeffs));
   }
   // ... resto della classe
private:
   bool someFlag;
   double coeffs[SOME_MAGIC_DEFINE];
};
```

Segreto #6: Sta crescendo in fretta – A proposito di complessità

```
class CarSettings {
public:
   CarSettings() : someFlag (false) , coeffs ()
   {
   }

   CarSettings(const CarSettings& other)
      : someFlag (other.someFlag),
        coeffs(other.coeffs) // OK!
   {

   }
   // ... resto della classe
private:
   bool someFlag;
   std::array<double, SOME_MAGIC_DEFINE> coeffs;
};
```

Segreto #6: Sta crescendo in fretta – A proposito di complessità

Non "aprite il cofano" quando non serve

Segreto #6: Sta crescendo in fretta – A proposito di complessità

Non "aprite il cofano" quando non serve

```
class CarSettings {
public:
// ... quello di cui ti devi davvero preoccupare
private:
   bool someFlag = false;
   double coeffs[SOME_MAGIC_DEFINE]; //oppure std::array<double, SOME_MAGIC_DEFINE>
};


// lo fa il compilatore, gratis (cioè, dipende ☺)
CarSettings() : someFlag (false) , coeffs ()  {}

CarSettings(const CarSettings& other)
...
```

Per approfondire: www.italiancpp.org/eventi/dettagli-meetup-bologna-2014/#zero

Segreto #6: Sta crescendo in fretta

L’evoluzione non si ferma al linguaggio:

- Compilatori (VC++, Clang, GCC, …)
- Tools (VAssistX, Clang Refactoring, …)
- Librerie (boost, POCO, Qt, …)
- Risorse (Stroustrup, Meyers, …)
- …

Dal meetup di Bologna (8/11/2014), foto di Gian Lorenzo Meocci

Segreto #7: Il suo ecosistema è straordinario

Segreto #7: Il suo ecosistema è straordinario

isocpp.org

cppcon.org

Communities C++ in Europa nel 2014

Fonte: http://tinyurl.com/m3n8s4e

Nel 2015 non punterei sul C++ perché…

- "...è complicato…e poi con tutti quei templates…" – Luca, studente di Ingegneria Informatica

Ha molte più cose da sapere rispetto ad altri linguaggi, ma è difficile che ti servano sempre tutte.

- "…non ha un vero equivalente dei .net framework" – Matteo, Software Engineer

Vedremo già qualcosa in più dal C++17. Per ora considera le tante librerie standard «de facto».

- "Dov'è il mio garbage collector??" – Laura, Programmatrice Java

La filosofia del C++ è differente. Il GC a volte può creare problemi (e.g. performance).

- "…voglio applicare gli stessi idiomi che uso in C# o Java" – Claudio, sostenitore della Campagna Anti-IF

Dal C++11 puoi farlo molto più facilmente.

- "...se devo andar forte faccio tutto in C" – Carlo, Programmatore Senior

Quando ne hai bisogno puoi semplificarti la vita perché C e C++ sono estremamente compatibili.

- "...non è produttivo" – Gerri, Senior Architect

Se produttività significa scrivere meno righe di codice, dal C++11 hai tante novità che aiutano.

Quindi perché nel 2015 parliamo ancora di C++?

Perché nel 2015 parliamo ancora di C++?

- È popolare
- È compatibile con il C
- Si preoccupa del passato
- Non è garbage collected
- È indipendente dal paradigma
- È adatto alla Systems Programming
- Sta crescendo in fretta
- Il suo ecosistema è straordinario

Domande?

Grazie!