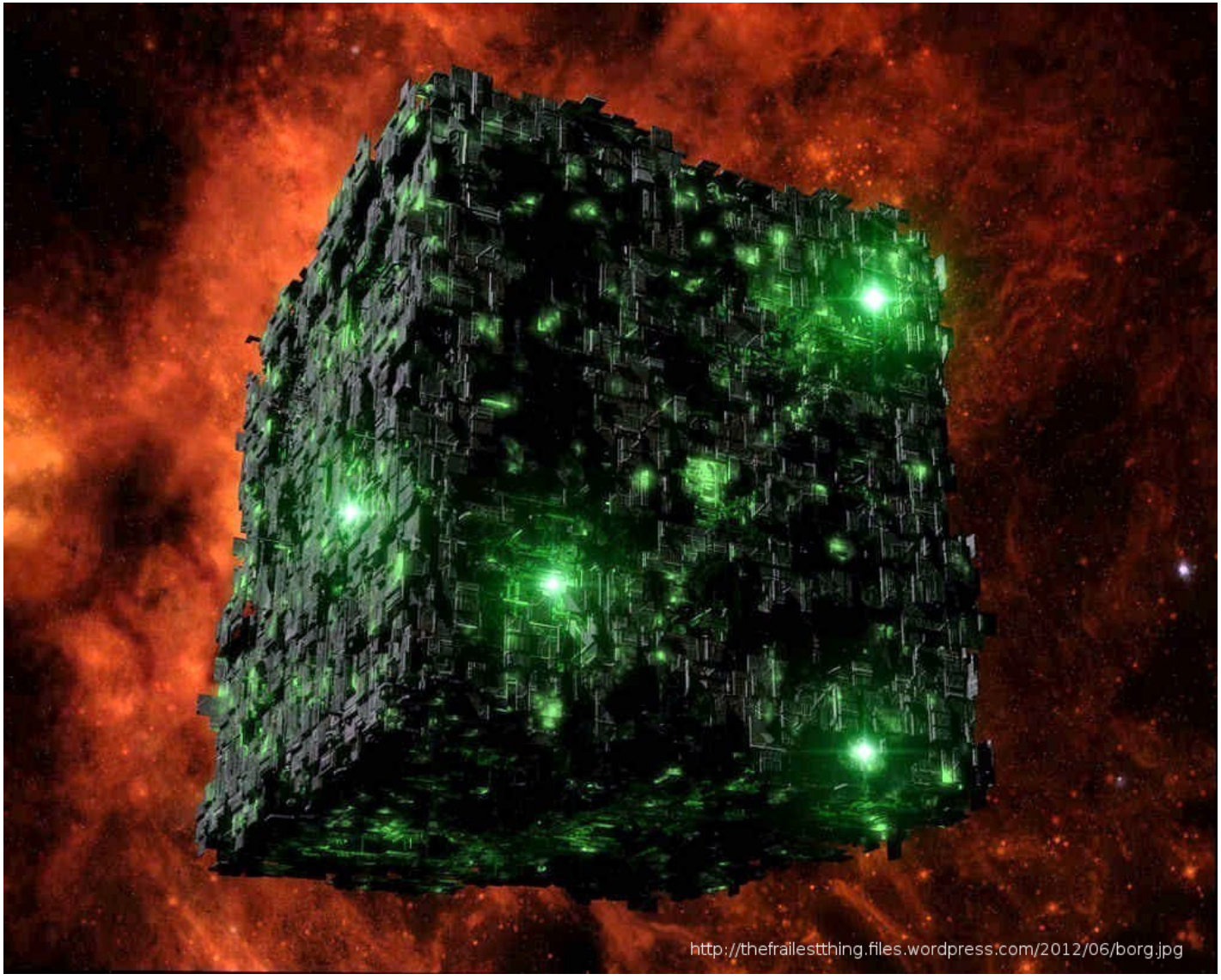
Dyspozytor w C++14

(czyt. dispatcher)

Bartek ’BaSz’ Szurgot

bartek.szurgot@baszerr.eu

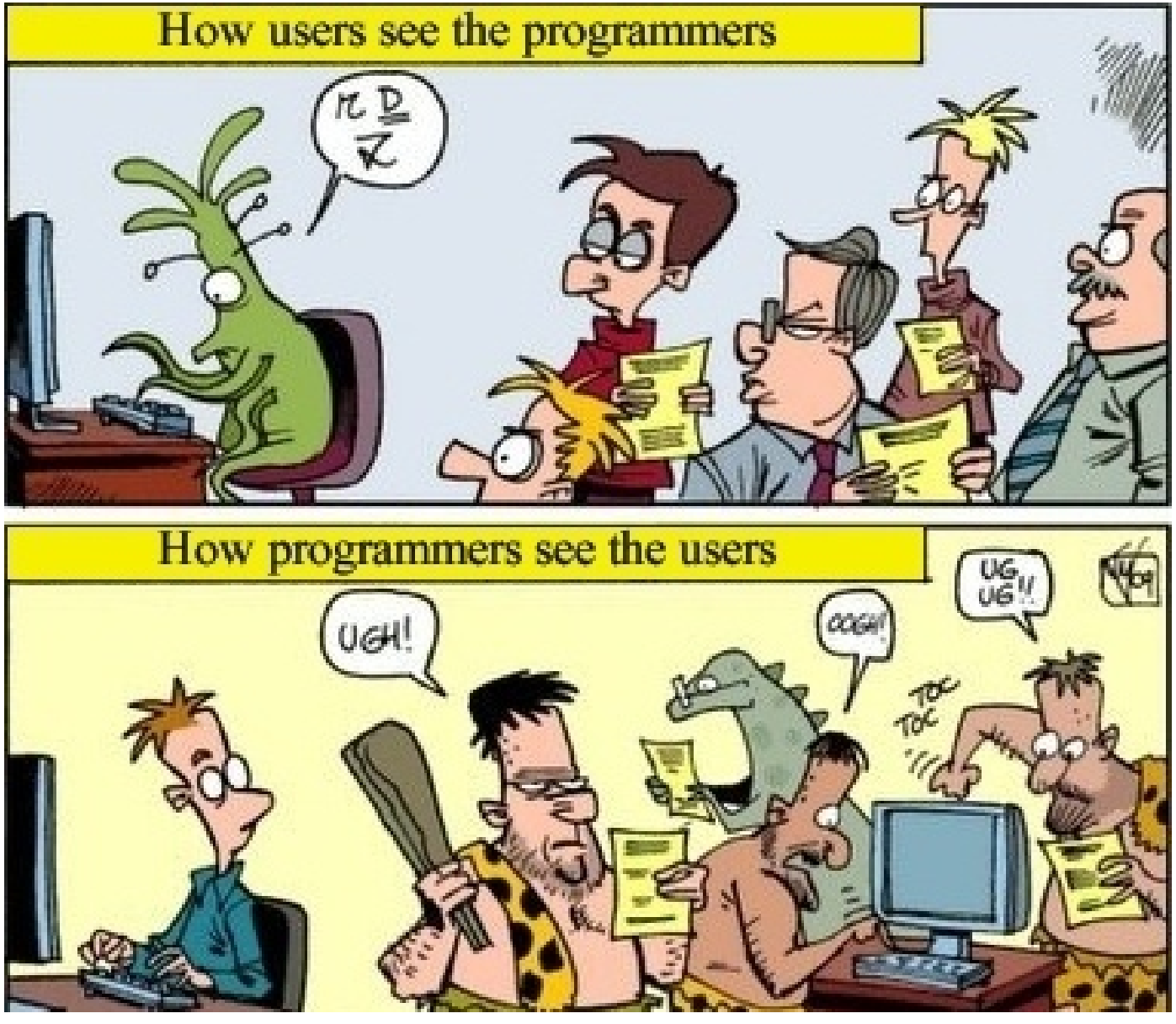
2014-11-26


http://thefrailestthing.files.wordpress.com/2012/06/borg.jpg

Plan wycieczki

Homo calculatus

Homo calculatus



Dawno temu, w pewnej korporacji...

Praktyka

Plan wycieczki

Zadanie

- "Message delivering framework"
- Brak wspólnej klasy bazowej dla wiadomości

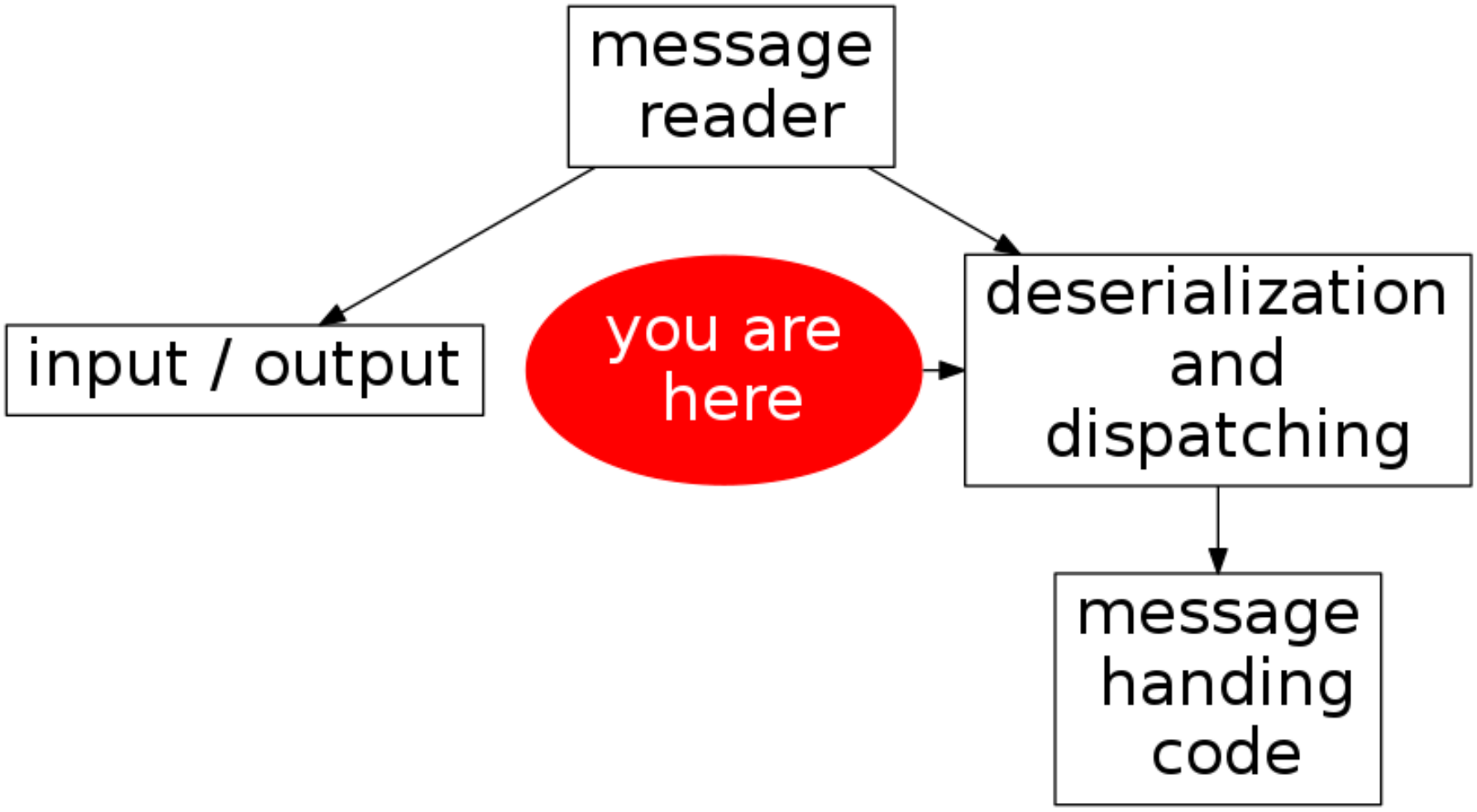
Zadanie

- "Message delivering framework"
- Brak wspólnej klasy bazowej dla wiadomości
- Framework powinien:
  - Obsługiwać przychodzące wiadomości
  - robić-całą-magię©
  - Przekazać końcowy typ do użyszkodnika :-)

Zadanie

- "Message delivering framework"
- Brak wspólnej klasy bazowej dla wiadomości
- Framework powinien:
  - Obsługiwać przychodzące wiadomości
  - robić-całą-magię©
  - Przekazać końcowy typ do użyszkodnika :-)
- Poza zakresem:
  - Implementacja serializacji
  - Implementacja deserializacji
  - Przesył danych binarnych

Mapa systemu



Format wiadomości

```
1  struct Hello
2  {
3    static constexpr int type() { return 142; }
4    // some data...
5  };
6  struct Say
7  {
8    static constexpr int type() { return 675; }
9    // other data...
10 };
11 struct Bye
12 {
13   static constexpr int type() { return 234; }
14   // more data...
15 };
```

Postać zserializowana

```
1  #include <string>
2
3  struct BinaryMsg
4  {
5    int         type_;
6    std::string data_;
7  };
```

(De)serializacja

```
1  #include "BinaryMsg.hpp"
2
3  template<typename M>
4  BinaryMsg serialize(M const& m);
5
6  template<typename M>
7  M deserialize(BinaryMsg const& b);
```

Klasa bazowa dyspozytora

```
1  #include "BinaryMsg.hpp"
2
3  struct Dispatcher
4  {
5    virtual ~Dispatcher(void) = default;
6    virtual void dispatch(BinaryMsg const& bin) = 0;
7  };
```

Plan wycieczki

Implementacja if-then...

```
1  #include <stdexcept>
2  #include "Dispatcher.hpp"
3  #include "messages.hpp"
4
5  struct ManualDispatcher: public Dispatcher
6  {
7    void handle(Hello const& msg);
8    void handle(Say const& msg);
9    void handle(Bye const& msg);
10
11   virtual void dispatch(BinaryMsg const& bin) final override
12   {
13     if( bin.type_ == Hello::type() )
14     {
15       handle( deserialize<Hello>(bin) );
16       return;
17     }
18     if( bin.type_ == Say::type() )
19     {
20       handle( deserialize<Say>(bin) );
21       return;
22     }
23     if( bin.type_ == Bye::type() )
24     {
25       handle( deserialize<Bye>(bin) );
26       return;
27     }
28     throw std::runtime_error{"unknnown message"};
29   }
30 };
```

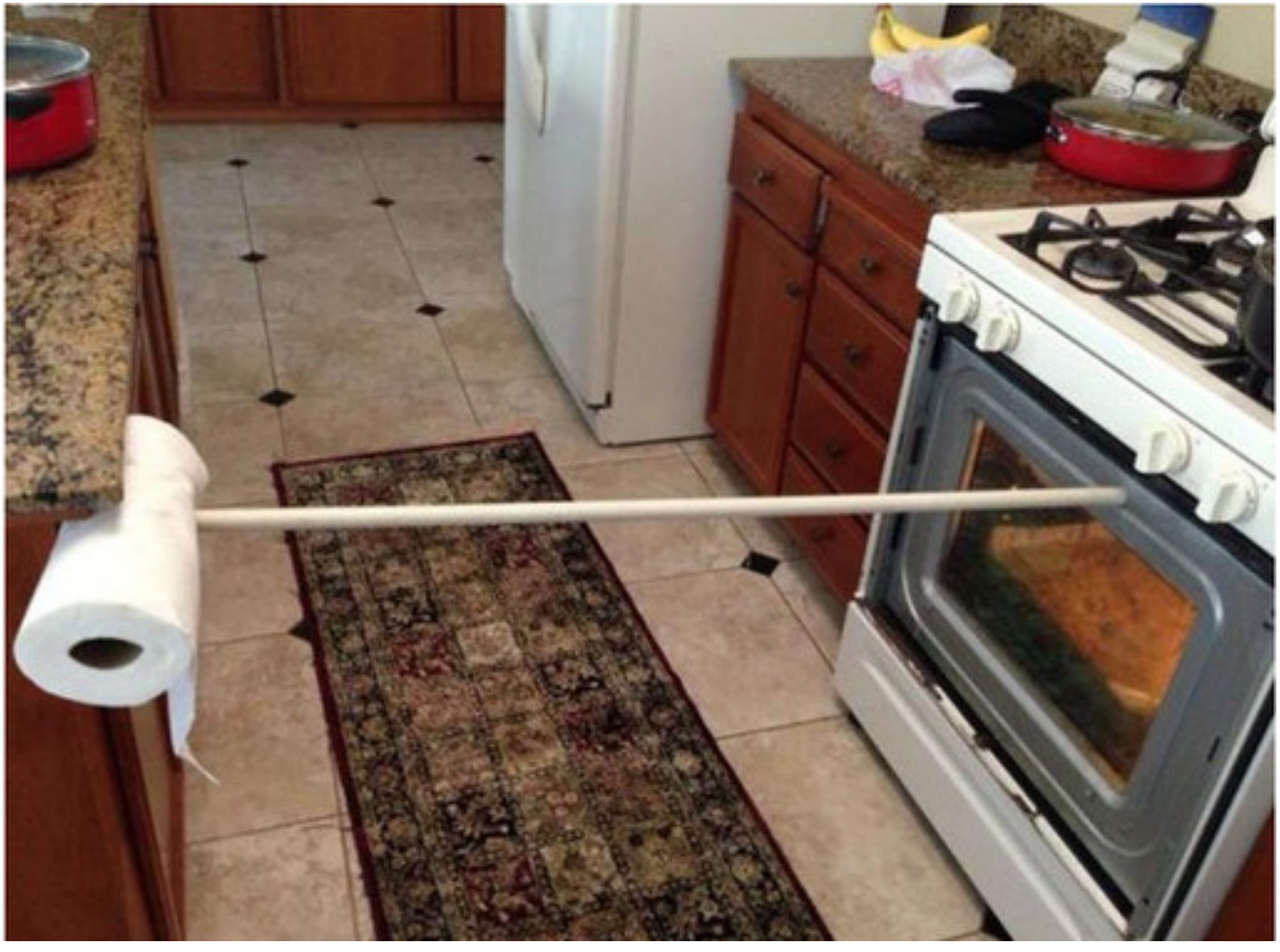
Old school

https://upload.wikimedia.org/wikipedia/commons/7/7b/California_Western_Railroad_Locomotive_45.jpg

Implementacja switch-case...

```
1  #include <stdexcept>
2  #include "Dispatcher.hpp"
3  #include "messages.hpp"
4
5  struct ManualDispatcher: public Dispatcher
6  {
7    void handle(Hello const& msg);
8    void handle(Say const& msg);
9    void handle(Bye const& msg);
10
11   virtual void dispatch(BinaryMsg const& bin) final override
12   {
13     switch(bin.type_)
14     {
15       case Hello::type():
16            handle( deserialize<Hello>(bin) );
17            return;
18       case Say::type():
19            handle( deserialize<Say>(bin) );
20            return;
21       case Bye::type():
22            handle( deserialize<Bye>(bin) );
23            return;
24     }
25     throw std::runtime_error{"unknnown message"};
26   }
27 };
```

Prowizorka – zawsze najtrwalsza



Plan wycieczki

Spróbujmy inaczej!

Idealne API...

```
1  #include "Dispatcher.hpp"
2  #include "messages.hpp"
3
4  struct IdealDispatcher: public Dispatcher
5  {
6    void handle(Hello const& msg);
7    void handle(Say const& msg);
8    void handle(Bye const& msg);
9  };
```

API z pośrednikiem

```
1  #include "Dispatcher.hpp"
2  #include "messages.hpp"
3
4  struct DispatcherImpl: public Dispatcher
5  {
6    virtual void dispatch(BinaryMsg const& bin) final override
7    { /* >>> here the miracle occurs << */ }
8  };
9
10 struct AlmostIdealDispatcher: public DispatcherImpl
11 {
12   void handle(Hello const& msg);
13   void handle(Say const& msg);
14   void handle(Bye const& msg);
15 };
```

Możliwa implementacja

```
1  #include "Dispatcher.hpp"
2  #include "messages.hpp"
3
4  template<typename FinalType, typename ...Msgs>
5  struct AutoDispatcher: public Dispatcher
6  {
7    AutoDispatcher()
8    { /* registers type to ID->handler map */ }
9
10   virtual void dispatch(BinaryMsg const& bin) final override
11   { /* uses map to find proper handler */ }
12 };
13
14 struct DoableDispatcher final:
15   public AutoDispatcher<DoableDispatcher,
16                         // messages lists:
17                         Hello, Say, Bye>
18 {
19   void handle(Hello const& msg);
20   void handle(Say const& msg);
21   void handle(Bye const& msg);
22 };
```

Implementacja – nagłówek

```
1  #include <unordered_map>
2  #include "Dispatcher.hpp"
3
4  template<typename FinalType, typename ...M>
5  struct AutoDispatcher: public Dispatcher
6  {
7    AutoDispatcher(void);
8    virtual void dispatch(BinaryMsg const& bin) final override;
9
10 private:
11   using HandlerFunc = void(*)(FinalType&, BinaryMsg const&);
12   std::unordered_map<int, HandlerFunc> handlers_;
13 };
```

Implementacja - dispatch()

```
1  #include "AutoDispatcher.hpp"
2
3  template<typename FinalType, typename ...M>
4  void AutoDispatcher<FinalType, M...>::dispatch(BinaryMsg const& bin)
5  {
6    auto it = handlers_.find(bin.type_);
7    if(it==std::end(handlers_))
8      throw std::runtime_error{"unknown message"};
9    auto h = it->second;
10   assert(h);
11   (*h)( *boost::polymorphic_downcast<FinalType*>(this), bin );
12 }
```

Implementacja - konstruktor

```
1  #include <type_traits>
2  #include "AutoDispatcher.hpp"
3
4  template<typename FinalType, typename ...M>
5  AutoDispatcher<FinalType, M...>::AutoDispatcher()
6  {
7    static_assert( std::is_final<FinalType>::value,
8        "FinalType is not the last type, in the derive chain" );
9
10   Reg<FinalType, M...>::call(handlers_);
11
12   assert( handlers_.size() == sizeof...(M) && "non-unique message ids" );
13 }
```

Pomocniczy szablon Reg

```
1  #include "BinaryMsg.hpp"
2  #include "serialization.hpp"
3
4  template<typename FinalType, typename ...M>
5  struct Reg
6  {
7    template<typename C>
8    static void call(C& c) { }
9  };
10
11 template<typename FinalType, typename M, typename ...Tail>
12 struct Reg<FinalType, M, Tail...>
13 {
14   template<typename C>
15   static void call(C& c)
16   {
17     auto dispatch = [](auto& ft, auto const& bin)
18                     { ft.handle( deserialize<M>(bin) ); };
19     c[M::type()] = dispatch;
20     Reg<FinalType, Tail...>::call(c);
21   }
22 };
```

Teraz wystarczy...

```
struct DispatcherImpl: public Dispatcher { virtual void dispatch(BinaryMsg const& bin) final }; struct AlmostIdealDispatcher: public DispatcherImpl { void handle(Hello const& msg); void handle(Say const& msg); void handle(Bye const& msg); }; template< typename FinalType, typename ...Msgs> struct AutoDispatcher: public Dispatcher { AutoDispatcher() virtual void dispatch (BinaryMsg const& bin) final }; struct DoableDispatcher final: public AutoDispatcher<DoableDispatcher, Hello, Say, Bye> { void handle(Hello const& msg); void handle(Say const& msg); void handle (Bye const& msg); }; template<typename FinalType, typename ...M> struct AutoDispatcher: public Dispatcher { AutoDispatcher(void); virtual void dispatch(BinaryMsg const& bin) final override; private: using HandlerFunc = void(*)(FinalType&, BinaryMsg const&); std::unordered_map<int, HandlerFunc> handlers_; }; template<typename FinalType, typename ...M> AutoDispatcher<FinalType, M...>::AutoDispatcher() { Reg<FinalType, M...>::call(handlers_); assert( handlers_.size() == sizeof...(M) && "non-unique message ids" ); } template <typename FinalType, typename ...M> void AutoDispatcher<FinalType, M...>::
"FinalType is not the last type, in the derive chain" ); auto it = handlers_.find(bin.type_); if(it== std::end(handlers_)) throw std:: runtime_error{"unknown message"}; auto h = it->second; assert(h); (*h)( *boost:: polymorphic_downcast<FinalType*>(this), bin ); } template<> Hello deserialize( BinaryMsg const&) { return {}; } template <> Bye deserialize(BinaryMsg const&) { return {}; } struct MyDispatcher final: public AutoDispatcher<MyDispatcher, Hello, Bye> { void handle(Hello const&) { cout << __PRETTY_FUNCTION__ << endl; } void handle(Bye const&) { cout << __PRETTY_FUNCTION__ << endl; } }; int main() { try { MyDispatcher md; md.dispatch( {42, "foo"} ); } catch( std::exception const& ex) { cerr << "EXCEPTION: " << typeid(ex).name() << ": " << ex.what() << endl; return 1; } } struct BinaryMsg { int type_; std::string data_; }; struct Dispatcher { virtual ~Dispatcher(void) = default; virtual void dispatch(BinaryMsg const& bin) = 0; }; using MyMock = AUTOMOCK(MyDispatcher); TEST(MyCode, UseMock) { MyMock mock; .Times(1); Dispatcher& base = mock; base.dispatch( {Hello::type(), "foo"} ); } struct IdealDispatcher: public
```

Koniec pracy!

Plan wycieczki

Metodologia

- Dwie wersje:
  - *Manual* – *if-else*
  - *Automatic* – prezentowany framework

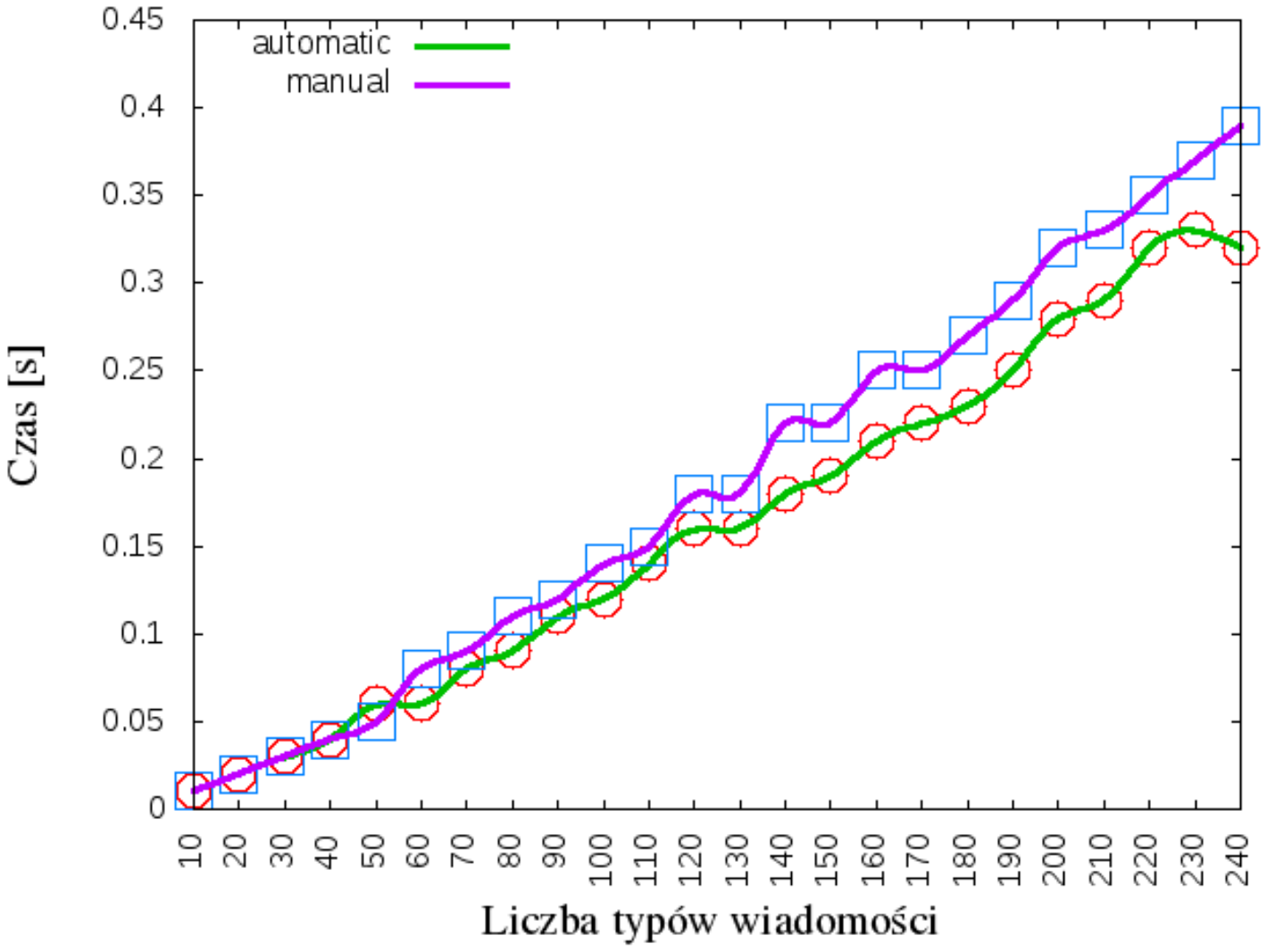
Metodologia

- Dwie wersje:
  - *Manual* – *if-else*
  - *Automatic* – prezentowany framework
- Kompilatory:
  - GCC-4.9.1
  - Clang-3.5.0

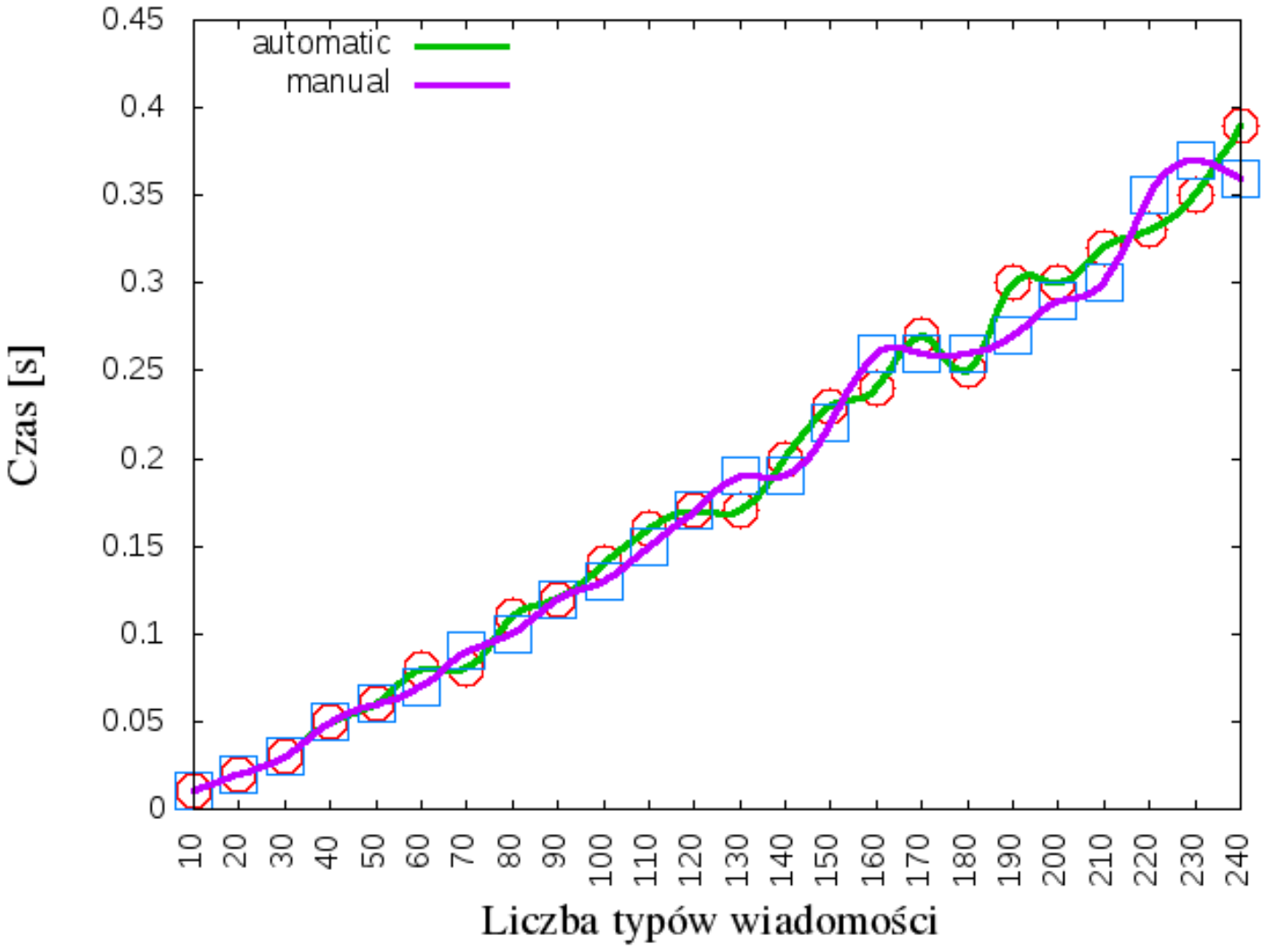
Metodologia

- Dwie wersje:
  - *Manual* – *if-else*
  - *Automatic* – prezentowany framework
- Kompilatory:
  - GCC-4.9.1
  - Clang-3.5.0
- 4-core Intel Core2 Quad 2.5GHz
- Sekwencyjne używanie kolejnych wiadomości
- Każda wiadomość innego rozmiaru
- Obsługa == wypisywanie (*/dev/null*)
- Lambda vs. funkcja

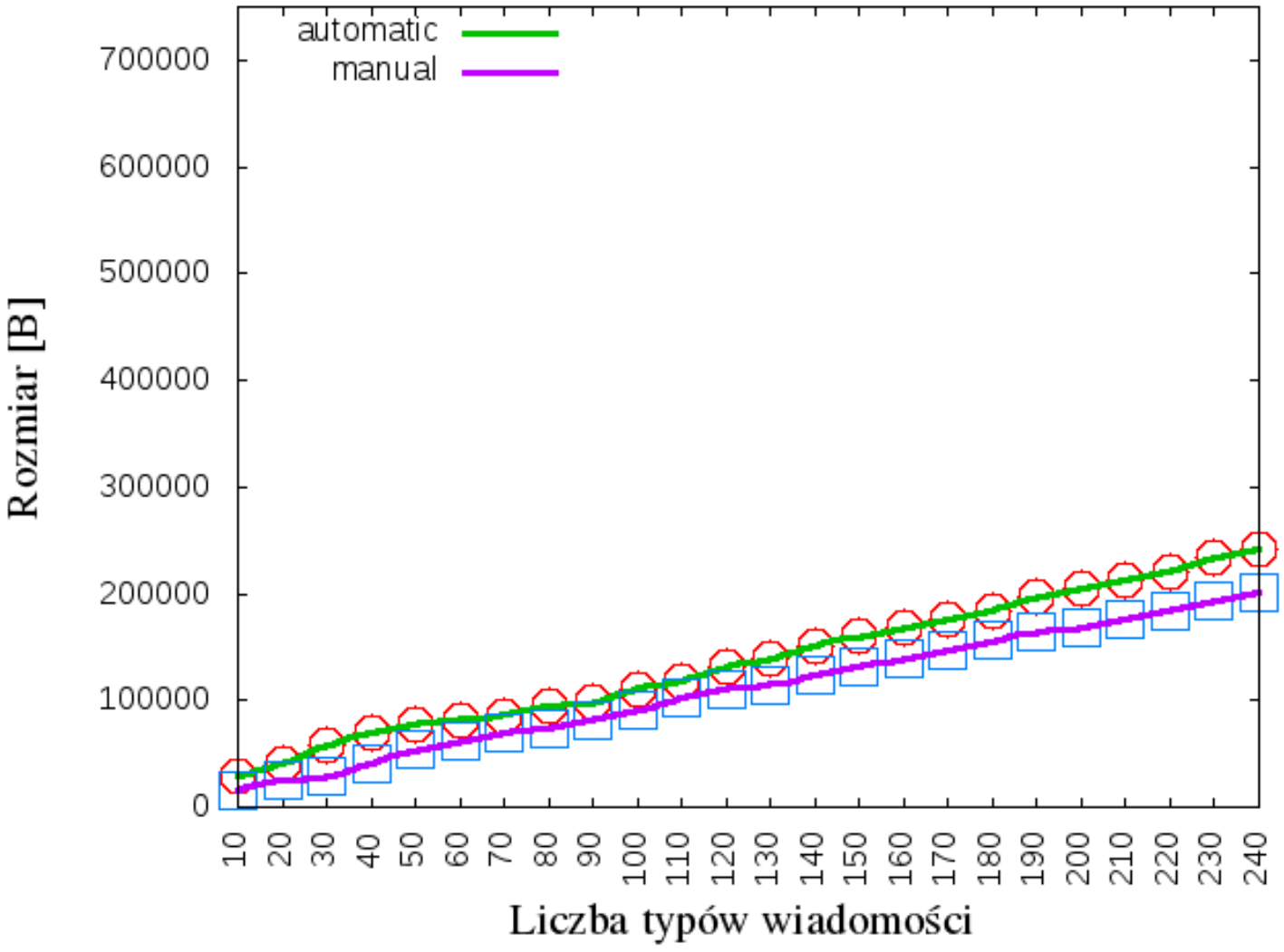
Czas wykonania – GCC



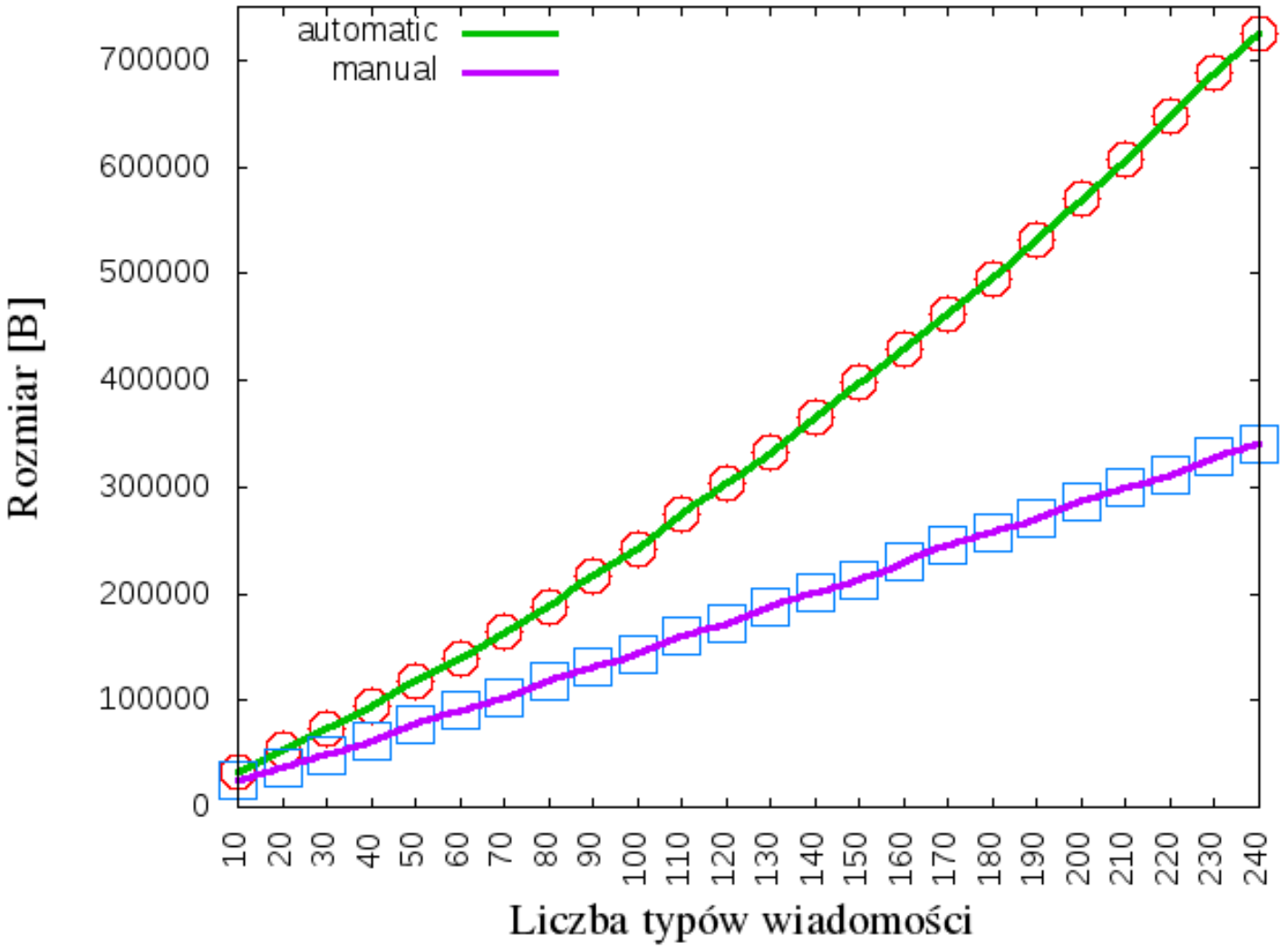
Czas wykonania – Clang



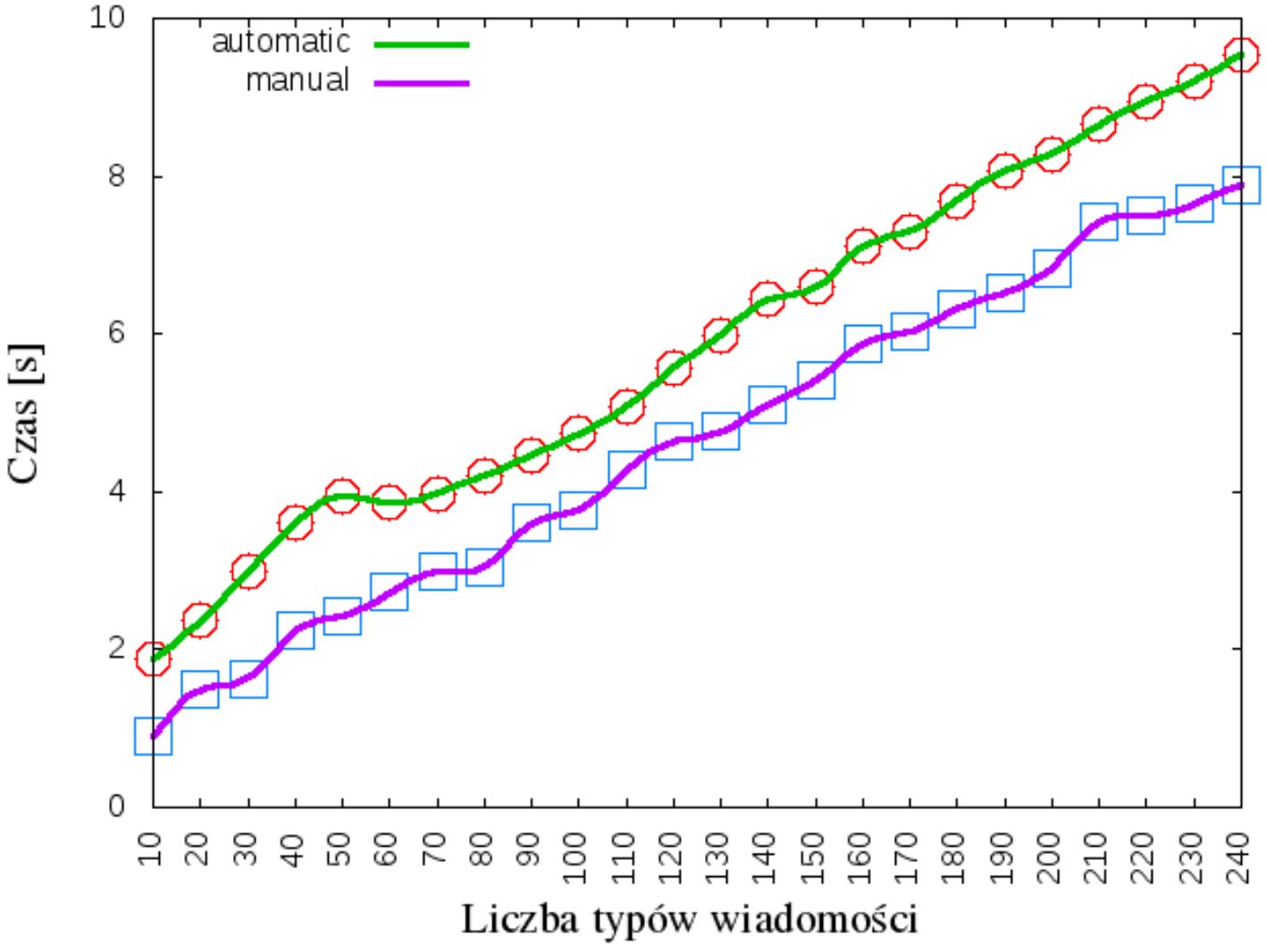
Rozmiar pliku wykonywalnego – GCC



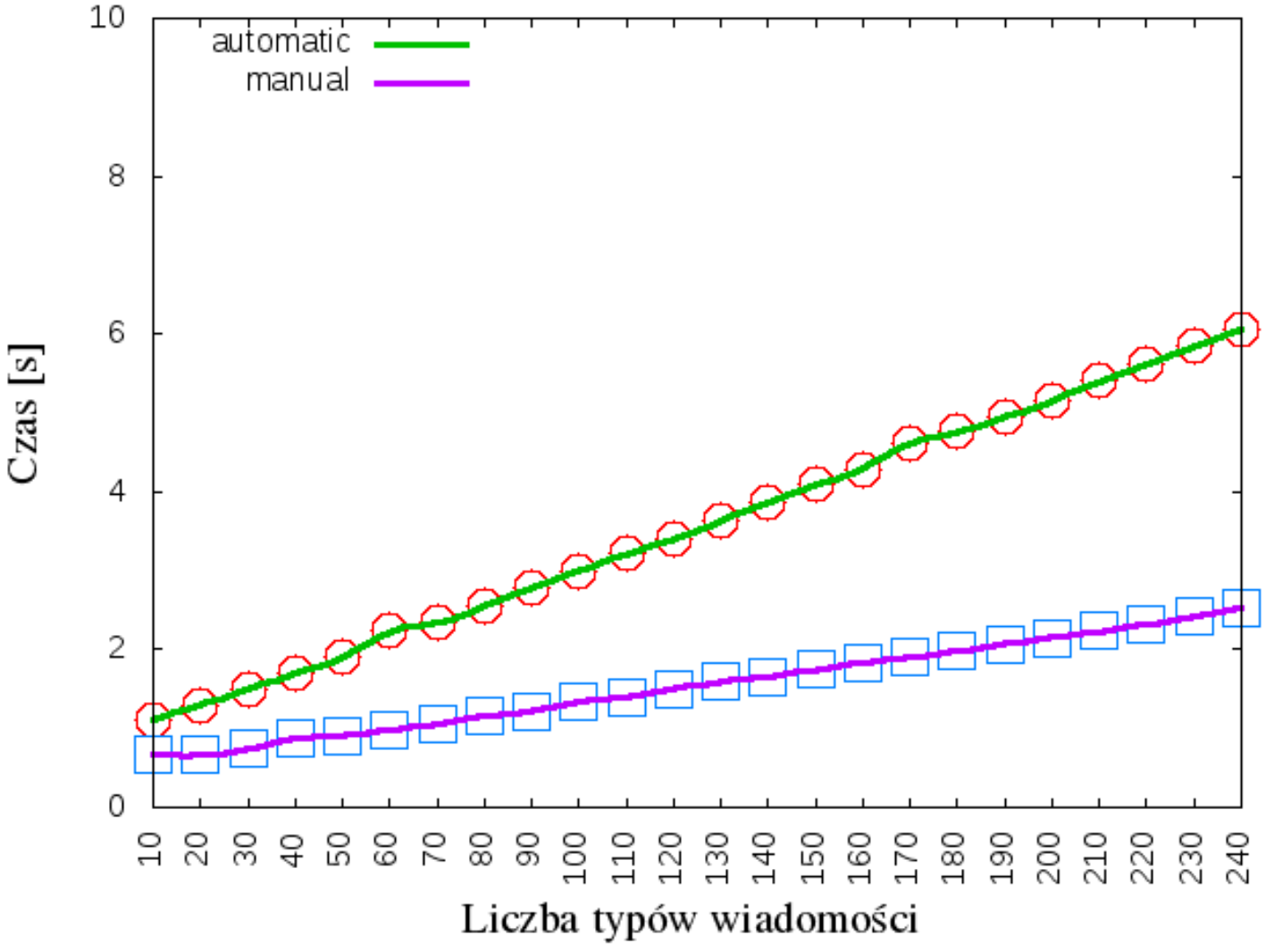
Rozmiar pliku wykonywalnego – Clang



Czas budowania – GCC



Czas budowania – Clang



Plan wycieczki

Testowanie

- Automatyczna generacja mocków
  - Są wszystkie potrzebne informacje
  - Metaprogramowanie do generowania metod
- Jeszcze mniej kodu do testów
- Łatwiejszy w utrzymaniu
- Szybsze testowanie

Modyfikacje

- W czasie kompilacji:
  - Weryfikacja unikalności IDków
  - Generowanie kodu wyszukiwania:
    - $O(N)$ – a'la *if-else*
    - $O(log(N))$ – binarne wyszukiwanie
    - $O(1)$ – (idealne) hashowanie

Modyfikacje

- W czasie kompilacji:
  - Weryfikacja unikalności IDków
  - Generowanie kodu wyszukiwania:
    - $O(N)$ – a'la *if-else*
    - $O(log(N))$ – binarne wyszukiwanie
    - $O(1)$ – (idealne) hashowanie
- W czasie wykonania:
  - IDki dostarczane w czasie wykonania
  - Dynamiczne (od)rejestrowanie obsługi wiadomości

Modyfikacje

- W czasie kompilacji:
  - Weryfikacja unikalności IDków
  - Generowanie kodu wyszukiwania:
    - $O(N)$ – a'la *if-else*
    - $O(log(N))$ – binarne wyszukiwanie
    - $O(1)$ – (idealne) hashowanie
- W czasie wykonania:
  - IDki dostarczane w czasie wykonania
  - Dynamiczne (od)rejestrowanie obsługi wiadomości
- Polityki oparte o szablony
- Optymalizacja pod rozmiar kodu
- Wybranie optymalizacji pod instancję
- Debug vs. release

Plan wycieczki

Pogląd na rozwiązanie

```
1  #include "Dispatcher.hpp"
2  #include "messages.hpp"
3
4  template<typename FinalType, // CRTP
5           typename ...M>      // messages list
6  struct AutoDispatcher: public Dispatcher
7  {
8    AutoDispatcher(void)
9    { /* registration of IDs from M... */ }
10   virtual void dispatch(BinaryMsg const& bin) final override
11   { /* dispatching deserialized messages to handle() methods */ }
12 };
13
14 struct MyStuff final:
15   public AutoDispatcher<MyStuff,
16                         // messages lists:
17                         Hello, Say, Bye>
18 {
19   void handle(Hello const& msg);
20   void handle(Say const& msg);
21   void handle(Bye const& msg);
22 };
```

Zalety i wady

+ Przenośny kod
+ Banalny w użyciu
+ Framework do ponownego użycia
+ Nie wymaga dziedziczenia przez schowek
+ Szybsze niż kod pisany ręcznie
+ Łatwe w testowaniu
+ Rozszerzalne (polityki)

Zalety i wady

+ Przenośny kod
+ Banalny w użyciu
+ Framework do ponownego użycia
+ Nie wymaga dziedziczenia przez schowek
+ Szybsze niż kod pisany ręcznie
+ Łatwe w testowaniu
+ Rozszerzalne (polityki)
- Dłuższy czas kompilacji
- Większe zużycie pamięci
- Większy plik wykonywalny

Zalety i wady

+ Przenośny kod
+ Banalny w użyciu
+ Framework do ponownego użycia
+ Nie wymaga dziedziczenia przez schowek
+ Szybsze niż kod pisany ręcznie
+ Łatwe w testowaniu
+ Rozszerzalne (polityki)
- Dłuższy czas kompilacji
- Większe zużycie pamięci
- Większy plik wykonywalny
- Wady vs. polityki

Więcej informacji

- Artykuł w *Programiście*
- Numer 8/2014 (27)

Otwarta implementacja

- Framework + kod do testów
- Dostępne na GitHub (revised BSD):
- `https://github.com/el-bart/but/tree/master/But/Pattern`

Pytania?