



Functional Reactive Programming in C++

Meeting C++ 2016

Ivan Čukić

ivan.cukic@kde.org
<http://cukic.co>

Disclaimer

Make your code readable. Pretend the next person who looks at your code is a psychopath and they know where you live.

Philip Wadler

Disclaimer

The code snippets are optimized for presentation, it is not production-ready code.

std namespace is omitted, value arguments used instead of const-refs or forwarding refs, etc.



FUNCTIONAL DESIGN

Higher-order functions

Purity and referential transparency

Elements of functional design

- Higher-order functions
- Purity
- Immutable state
- ...

Higher-order functions

We have had higher-order functions since C++98.

No lambdas, no `std::function` ... needed.

```
find_if(begin(cs), end(cs), is_error);  
  
auto gt42 = bind(greater<>(), _1, 42);  
           // or old bind1st, ...
```


Word frequency

1986: Donald Knuth was asked to implement a program for the "Programming pearls" column in the Communications of ACM journal.

The task: Read a file of text, determine the *n* most frequently used words, and print out a sorted list of those words along with their frequencies.

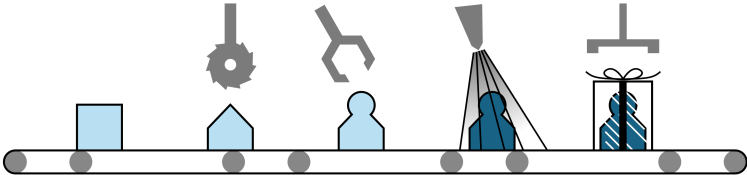
His solution written in Pascal was **10** pages long.

Word frequency

Response by Doug McIlroy was a 6-line shell script that did the same:

```
tr -cs A-Za-z '\n' |  
tr A-Z a-z |  
sort |  
uniq -c |  
sort -rn |  
sed ${1}q
```

Functional thinking – data transformation



Word frequency in C++

```
It was a bright cold day in April,\nand the clocks were striking thirteen.\nWinston Smith, his chin nuzzled
```

```
data | transform(  
    [] (char c) {  
        return isalnum(c) ? c : '\\n';  
    })
```

```
It\nwas\na\nbright\ncold\nday\nin\nApril\n\nand\nthe\nclocks\nwere\nstriking\nthirteen\n\nWinston\nSmith\n\nhis\nchin\nnuzzled
```

Word frequency in C++

```
It \nwas \na \nbright \ncold \nday \nin \nApril \n\n
and \nthe \nclocks \nwere \nstriking \nthirteen \n\n
Winston \nSmith \n\nhis \nchin \nnuzzled
```

```
... | transform(tolower)
```

```
it \nwas \na \nbright \ncold \nday \nin \napril \n\n
and \nthe \nclocks \nwere \nstriking \nthirteen \n\n
winston \nsmith \n\nhis \nchin \nnuzzled
```

Word frequency in C++

```
it\nwas\na\nbright\ncold\nday\nin\napril\n\nand\nthe\nclocks\nwere\nstriking\nthirteen\n\nwinston\nsmith\n\nhis\nchin\nnuzzled
```

```
... | split('\\n')
```

it **was** **a** **bright** **cold** **day** **in** **april**
and **the** **clocks** **were** **striking** **thirteen**
winston **smith** **his** **chin** **nuzzled**

Word frequency in C++

it was a bright cold day in april
 and the clocks were striking thirteen
 winston smith his chin nuzzled

```

... | sort
    | group_by(equal_to<>())
  
```

```

{ a a a a a ... },
{ as as as as as ... },
{ at at at at at ... }
  
```


Word frequency in C++

```
{ a a a a a ... },
{ and and and and and ... }
{ as as as as as ... },
```

```
... | transform([] (const auto &grp) {
      return make_pair(
          count(grp), *grp.cbegin());
    })
```

```
( 181, a ),
( 163, and ),
( 39, as ),
```

Word frequency in C++

```
( 181, a ),
( 163, and ),
( 39, as ),
```

```
... | sort
     | reverse
     | take(2)
```

```
( 439, the ),
( 256, of ),
```

Word frequency in C++

```

data | transform([] (char c) {
      |     return isalnum(c) ? c : '\n';
      | })
      | transform(tolower)
      | split('\n')
      | sort
      | group_by(equal_to<>())
      | transform([] (const auto &grp) {
          |     return make_pair(
              |         count(grp), *grp.cbegin());
          | })
      | sort
      | reverse
      | take(n)

```

* Inspired by the solution written by N. Milev in Haskell

Purity and referential transparency

```
int answer()  
{  
    std::cout << "Calculating the result\n";  
    return 42;  
}
```

```
...  
auto result = answer();  
...
```

```
$ ./a.out  
Calculating the result
```

Purity and referential transparency

```
int answer()  
{  
    std::cout << "Calculating the result\n";  
    return 42;  
}
```

...

```
auto result = 42;
```

...

```
$ ./a.out
```

Handling the program state



```
void on_clicked(const click_event &event)
{
    employees_view->visible = true;
    employees_view->load_team();
}
```



Handling the program state



```
void on_clicked(const click_event &event)
{
    employees_view->visible = true;
    employees_view->load_team();
}
```



Handling the program state



```
void on_clicked(const click_event &event)
{
    employees_view->visible = true;
    employees_view->load_team();
}
```



Handling the program state



```
void on_clicked(const click_event &event)
{
    employees_view->visible = true;
    employees_view->load_team();
}
```



Handling the program state



```

void on_clicked(const click_event &event)
{
    employees_view->visible = true;
    employees_view->load_team();
}
  
```



Object-oriented design

Don't ask for the information you need to do the work;
ask the object that has the information to do the work
for you.

Allen Holub

Object-oriented design

Step one in the transformation of a successful procedural developer into a successful object developer is a lobotomy.

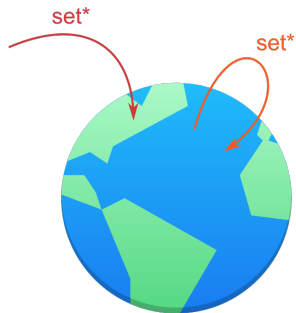
David West, Object Thinking

Object-oriented design

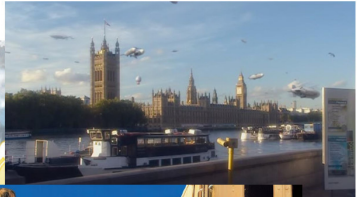
[...] Besides that, object thinking will lead to object immutability [...]

Yegor Bugayenko

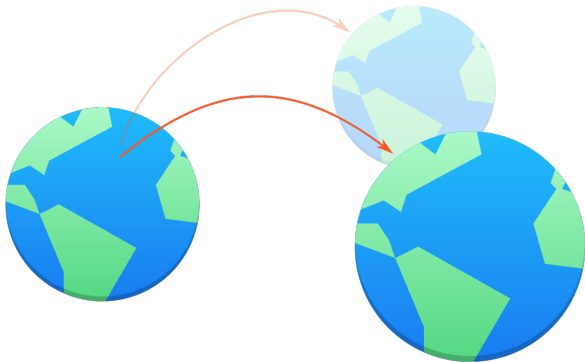
Object-oriented design



State in pure functional programs




State in pure functional programs



Isn't creating new worlds expensive?

Option 1: We don't really want copies:

```
class world {
    world with_population
        (int new_population) &&
    {
        world result(std::move(*this));
        ...
        return result;
    }
};
```



Option 2: Use immutable data structures (see Okasaki)

Isn't creating new worlds expensive?

```
class world {
    world with_population(int) &&;
};
```

The compiler enforces us not to write inefficient code.

If we provide anything else, we explicitly tell through our API that we support having parallel worlds.

Benefits of having parallel worlds

- Time travel (and reverse debugging, like gdb does)
- Testing different possible scenarios at the same time
- Maybe even having them interact with each other

REACTIVE PROGRAMMING

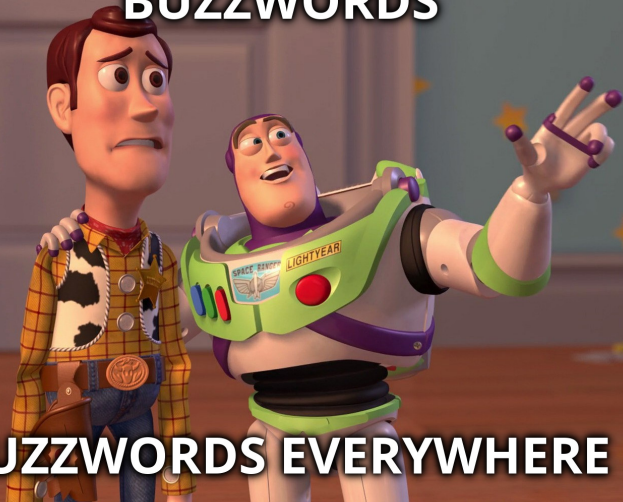
What is reactive?

What is reactive?

We believe that a coherent approach to systems architecture is needed, and we believe that all necessary aspects are already recognised individually: we want systems that are Responsive, Resilient, Elastic and Message Driven. We call these Reactive Systems. Systems built as Reactive Systems are more flexible, loosely-coupled and scalable. This makes them easier to develop and amenable to change. They are significantly more tolerant of failure and when failure does occur they meet it with elegance rather than disaster. Reactive Systems are highly responsive, giving users effective interactive feedback.

Reactive Manifesto 2.0

BUZZWORDS



BUZZWORDS EVERYWHERE

Reactive systems

One view of being reactive:

- responds quickly
- resilient to failure
- responsive under workload
- based on message-passing

What is reactive?

Showing a response to a stimulus

Oxford Dictionary

Ways to be reactive?

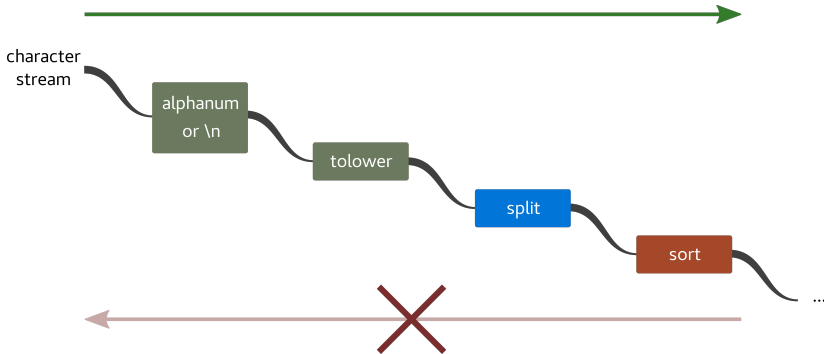
- C: event call-backs
- Java: event listeners
- C++/Qt: signals and slots
- Threads for all!

Lets try this again...

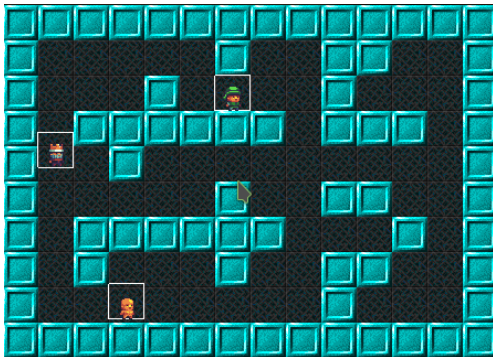
Design components:

- to react to requests, not to respond
- to request, not to need the response

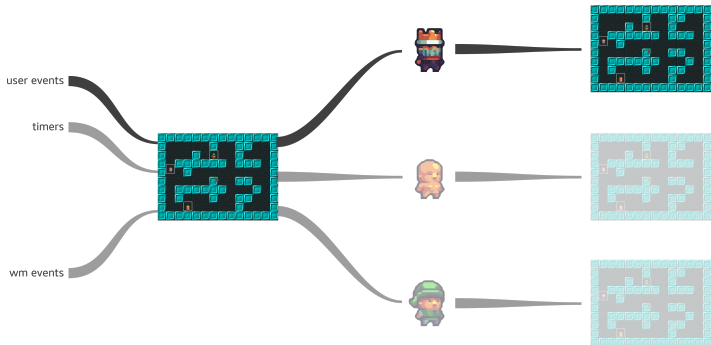
Lets try this again...



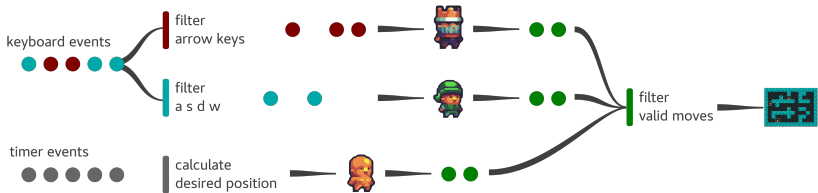
Lets try this again...



Lets try this again...



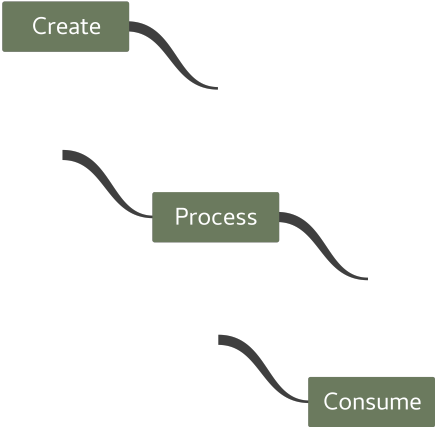
Lets try this again...



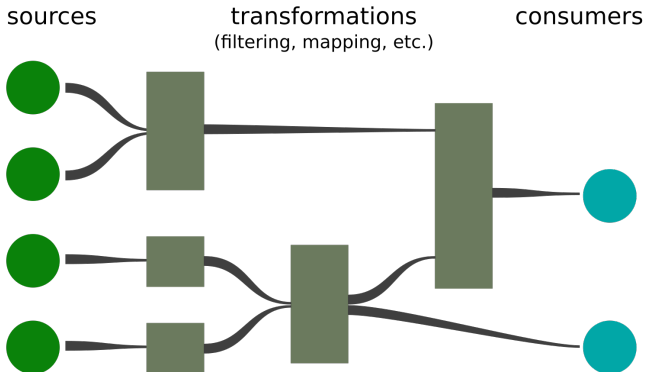
Modify and pass on

- No shared mutable state
- Separate isolated components
- Communication only through message passing
- No upstream response messages

Modify and pass on



Modify and pass on

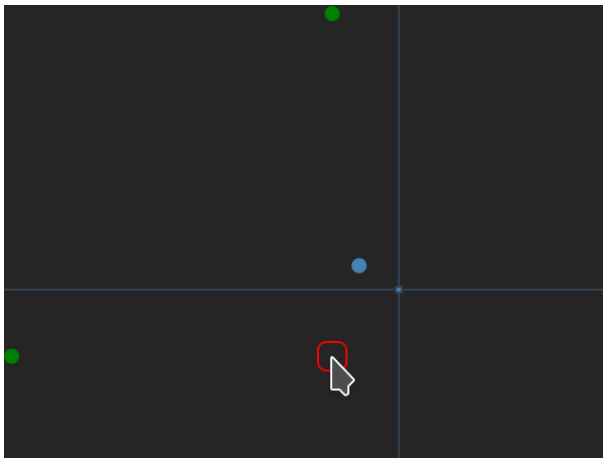


Streams are ranges

Streams can only be transformed with algorithms that accept input ranges, since we don't have all the items. We don't even know when (if) they will end.

transform, filter, take, drop, etc.

Small demo



Small demo

We have a stream of mouse coordinates: `mouse_stream`

And a set of receivers like `mouse_cursor`,
`top_ruler_marker`, etc.

Basic direct connection:

```
| mouse_stream | mouse_cursor->move_to; |
```


Transforming the input

We want to transform the mouse coordinate, to project it on the x axis:

```
mouse_stream |  
  transform(project_on_x) |  
  top_ruler_marker->move_to;
```

But the mouse cursor marker is no longer moving.

Forking the stream

```
mouse_stream |
  // Pass the events to the mouse cursor,
  // and pass them to the next
  // transformation in the chain
  tee(mouse_cursor->move_to) |

  // projecting the mouse coordinates on
  // the x axis
  transform(project_on_x) |
    top_ruler_marker->move_to;
```

Properly forking the stream

```
mouse_stream |
  tee(mouse_cursor->move_to) |
  // If we want to do something more
  // complex with both streams after
  // forking, tee is not readable
  fork(
    transform(project_on_x) |
      top_ruler_marker->move_to,
    transform(project_on_y) |
      left_ruler_marker->move_to
  );
```

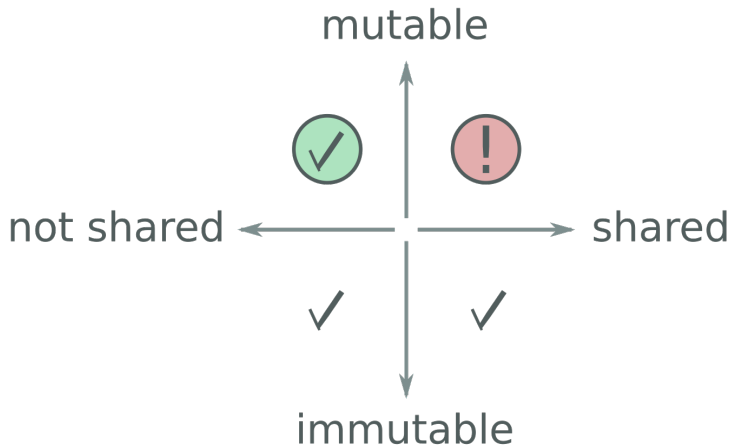
Transformations with state

```
class gravity_object {
public:
    ...

    point operator() (const point &new_point)
    {
        m_point.x = m_point.x * .99
                  + new_point.x * .01;
        m_point.y = ...;
        return m_point;
    }

private:
    point m_point;
    ...
};
```

Transformations with state



Transformations with state

```
mouse_stream |
  tee(mouse_cursor->move_to) |
  fork(
    map(project_on_x) |
      top_ruler_marker->move_to,
    map(project_on_y) |
      left_ruler_marker->move_to,
    map(gravity_object()) |
      gravity_marker->move_to
  );
```

Stream filtering

We want only points where `point.y % 100 == 0`

```

mouse_stream |
  tee(mouse_cursor->move_to) |
  fork(
    ...
    filter(point_filter) |
    filter_marker->move_to
  );

```

Generating new events

```
class continuous_points {
public:
    vector<point>
    operator() (const point &new_point) {
        // generate all the points between the
        // previous one and the new one
        vector<point> result = ...

        m_previous_point = new_point;
        return result;
    }

private:
    point m_previous_point;
};
```


Generating new events

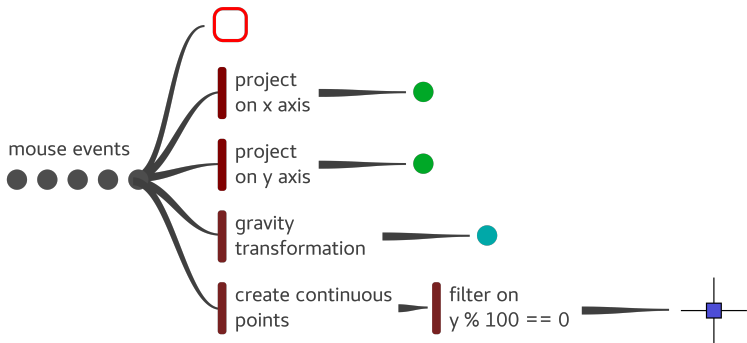
```
mouse_stream |
  tee(mouse_cursor->move_to) |
  fork(
    ...
    // or transform(...) | flatten
    flatmap(continuous_points()) |
    filter(point_filter) |
    filter_marker->move_to
  );
```

What are the benefits?

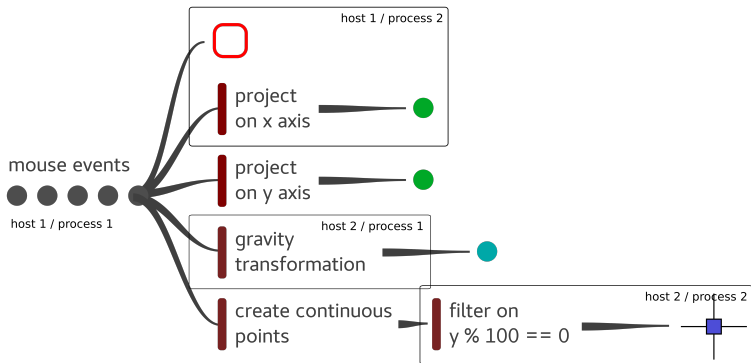
- Separate components
- Reusable, composable transformations
- Asynchronousness built into the software design

But that is not all...

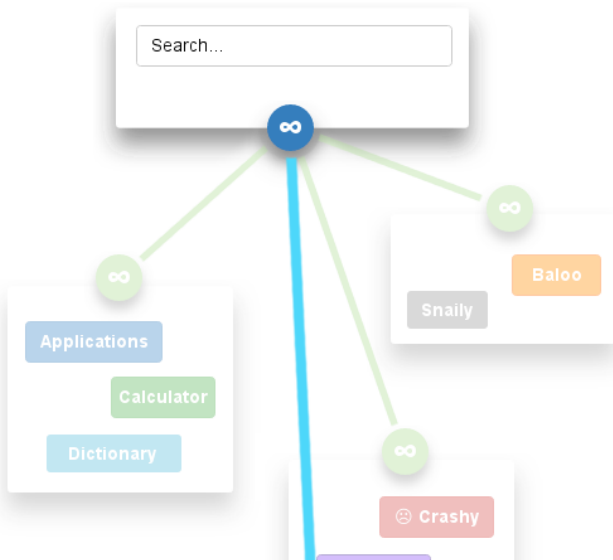
Microservices?



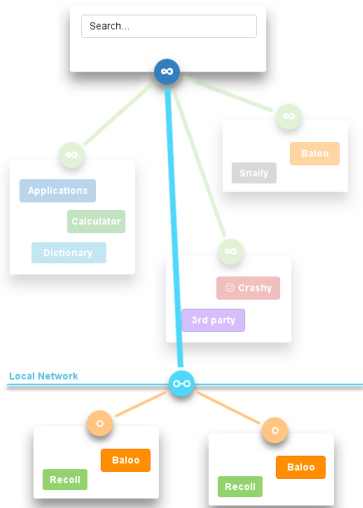
Microservices?



Microservices?



Microservices?



KDE Frameworks 5



Libraries

For the lower level parts:

- SObjectizer
- C++ Actor Framework
- ZeroMQ

For the higher level:

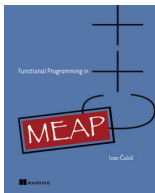
- RxCpp

Answers? Questions! Questions? Answers!

Kudos:

Friends at KDE and blueSystems

Dr Saša Malkov, Dr Zoltan Porkolab



MEAP – Manning Early Access Program
Functional Programming in C++
cukic.co/to/fp-in-cpp

Discount code:
meetingcpp

