Embedded Domain Specific Langauges for Embedded Bare Metal Projects

ciere consulting

Michael Caisse

C++Now 2019 | mcaisse@ciere.com | follow @MichaelCaisse

Part I

We can’t have nice things

No nice things

- Too slow
- Too bloated
- Too strange

Too Slow

You can write slow performing code in any language.

Too Slow

You can write slow performing code in any language.

Too Slow

Optimization is usually turned off:

- Easier to map generated assembly
- Easier to step through code
- Fixes strange problems

Too Slow

Optimization is usually turned off:

- Easier to map generated assembly
- Easier to step through code
- Fixes strange problems

```
-O0
Reduce compilation time and make debugging produce the
expected results.  This is the default.
```

ciere.com

Too Slow

Optimization is usually turned off:

- Easier to map generated assembly
- Easier to step through code
- Fixes strange problems

Too Slow - Fixes strange problems

Typically this is an indicator that all the proper constraints are not set properly.

- `volatile`
- IRQ handling
- Memory barriers

Too Slow

Modern Techniques depend on the optimizer.

Too Bloated

Bad code is often bloated

Too Bloated

- RTTI
- Exceptions
- Compile / Link optimizations off

Too Bloated

- RTTI
- Exceptions
- Compile / Link optimizations off

Too Bloated

- RTTI
- Exceptions
- Compile / Link optimizations off

Use compiler switches to turn off RTTI and Exceptions.

Too Bloated

- RTTI
- Exceptions
- Compile / Link optimizations off

Too Bloated

- RTTI
- Exceptions
- Compile / Link optimizations off

Use -Os and LTO switches.

Too Bloated

- Understanding templates
- Object Oriented bloat

Too Bloated

- Understanding templates
- Object Oriented bloat

Not Bloated

Raw loop:

```
int no_algo()
{
    int count = 0;
    for(int i=0; i<a.size(); ++i)
    {
        if(a[i] == 8){ ++count; }
    }

    return count;
}
```

Using standard algorithm:

```
int algo()
{
    return
    std::count_if( a.cbegin(), a.cend()
                 , [](int v)
                   {
                       return v==8;
                   }
    );
}
```

Not Bloated

Raw loop:

```
int no_algo()
{
    int count = 0;
    for(int i=0; i<a.size(); ++i)
    {
        if(a[i] == 8){ ++count; }
    }

    return count;
}
```

Using standard algorithm:

```
int algo()
{
    return
    std::count_if( a.cbegin(), a.cend()
                 , [](int v)
                   {
                       return v==8;
                   }
    );
}
```

Not Bloated

Raw loop:

```
int no_algo()
{
    int count = 0;
    for(int i=0; i<a.size(); ++i)
    {
        if(a[i] == 8){ ++count; }
    }

    return count;
}
```

Using standard algorithm:

```
int algo()
{
    return
    std::count_if( a.cbegin(), a.cend()
                 , [](int v)
                   {
                      return v==8;
                   }
    );
}
```

Not Bloated

Raw loop:

```
1  no_algo():
2          adrp    x2, a
3          add     x2, x2, :lo12:a
4          mov     x1, 0
5          mov     w0, 0
6  .L3:
7          ldr     w3, [x2, x1, lsl 2]
8          add     x1, x1, 1
9          cmp     w3, 8
10         cinc    w0, w0, eq
11         cmp     x1, 20
12         bne     .L3
13         ret
```

Using standard algorithm:

```
1  algo():
2          adrp    x2, a
3          add     x2, x2, :lo12:a
4          mov     x1, 0
5          mov     x0, 0
6  .L3:
7          ldr     w3, [x1, x2]
8          cmp     w3, 8
9          bne     .L2
10         add     x0, x0, 1
11 .L2:
12         add     x1, x1, 4
13         cmp     x1, 80
14         bne     .L3
15         ret
```

Bloated

Old FIR:

```
float do_old_fir(data_array_t const & data)
{
    constexpr std::array<float, 4>
       coeff{1.0, 0.5, 0.2, 0.1};

    static_assert( data.size() == coeff.size()
                 , "fir data length mismatch");

    float result = data[0] * coeff[0]
                 + data[1] * coeff[1]
                 + data[2] * coeff[2]
                 + data[3] * coeff[3];

    return result;
}
```

Algorithm FIR:

```
float do_fir(data_array_t const & d)
{
    constexpr std::array<float, 4>
       coeff{1.0, 0.5, 0.2, 0.1};

    static_assert( data.size() == coeff.size()
                 , "fir data length mismatch");

    return
       std::inner_product( d.cbegin(), d.cend()
                         , coeff.cbegin()
                         , 0.0 );
}
```

Bloated

Old FIR:

```
float do_old_fir(data_array_t const & data)
{
    constexpr std::array<float, 4>
       coeff{1.0, 0.5, 0.2, 0.1};

    static_assert( data.size() == coeff.size()
                 , "fir data length mismatch");

    float result = data[0] * coeff[0]
                 + data[1] * coeff[1]
                 + data[2] * coeff[2]
                 + data[3] * coeff[3];

    return result;
}
```

Algorithm FIR:

```
float do_fir(data_array_t const & d)
{
    constexpr std::array<float, 4>
       coeff{1.0, 0.5, 0.2, 0.1};

    static_assert( data.size() == coeff.size()
                 , "fir data length mismatch");

    return
       std::inner_product( d.cbegin(), d.cend()
                         , coeff.cbegin()
                         , 0.0 );
}
```

Bloated

Old FIR:

```
float do_old_fir(data_array_t const & data)
{
    constexpr std::array<float, 4>
       coeff{1.0, 0.5, 0.2, 0.1};

    static_assert( data.size() == coeff.size()
                 , "fir data length mismatch");

    float result = data[0] * coeff[0]
                 + data[1] * coeff[1]
                 + data[2] * coeff[2]
                 + data[3] * coeff[3];

    return result;
}
```

Algorithm FIR:

```
float do_fir(data_array_t const & d)
{
    constexpr std::array<float, 4>
       coeff{1.0, 0.5, 0.2, 0.1};

    static_assert( data.size() == coeff.size()
                 , "fir data length mismatch");

    return
       std::inner_product( d.cbegin(), d.cend()
                         , coeff.cbegin()
                         , 0.0 );
}
```

Bloated

Old FIR:

```
1  do_old_fir(std::array<float, 4ul> const&):
2          ldp     s1, s0, [x0]
3          fmov    s2, 5.0e-1
4          mov     w1, 52429
5          movk    w1, 0x3e4c, lsl 16
6          fmadd   s0, s0, s2, s1
7          ldr     s1, [x0, 8]
8          fmov    s2, w1
9          fmadd   s1, s1, s2, s0
10         ldr     s2, [x0, 12]
11         mov     w0, 52429
12         movk    w0, 0x3dcc, lsl 16
13         fmov    s0, w0
14         fmadd   s0, s2, s0, s1
15         ret
```

Algorithm FIR:

```
1  do_fir(std::array<float, 4ul> const&):
2          sub     sp, sp, #16
3          mov     x1, 1065353216
4          movk    x1, 0x3f00, lsl 48
5          movi    d0, #0
6          str     x1, [sp]
7          mov     x1, 52429
8          movk    x1, 0x3e4c, lsl 16
9          movk    x1, 0xcccd, lsl 32
10         movk    x1, 0x3dcc, lsl 48
11         str     x1, [sp, 8]
12         mov     x1, 0
13 .L2:
14         ldr     s1, [x0, x1]
15         ldr     s2, [sp, x1]
16         add     x1, x1, 4
17         cmp     x1, 16
18         fmul    s1, s1, s2
19         fcvt    d1, s1
20         fadd    d0, d0, d1
21         bne     .L2
22         fcvt    s0, d0
23         add     sp, sp, 16
24         ret
```

Too Strange

- Too Strange
- Too Complex
- Too Foreign
- Nobody Understands

ciere.com

Too Strange

- Too Strange
- Too Complex
- Too Foreign
- Nobody Understands

Too Strange

- Too Strange
- Too Complex
- Too Foreign
- Nobody Understands

Too Strange

“Now that the project has less complexity, should we move to C?”

Too Strange - Move to C

No

```
/me :eyeroll:
```

Too Strange - Move to C

No language below

Too Strange - Move to C

No language below ... except assembly and C.

Too Strange - Move to C

C++ now 2018 MAY 7 - 11 cppnow.org

NL.26: Use conventional `const` notation

Reason

Conventional notation is more familiar to more programmers. Consistency in large code bases.

Example

```
const int x = 7;    // OK
int const y = 9;    // bad

const int *const p = nullptr;   // OK, constant pointer to constant int
int const *const p = nullptr;   // bad, constant pointer to constant int
```

Note

We are well aware that you could claim the "bad" examples more logical than the ones marked "OK", but they also confuse more people, especially novices relying on teaching material using the far more common, conventional OK style.

As ever, remember that the aim of these naming and layout rules is consistency and that aesthetics vary immensely.

Enforcement

Flag `const` used as a suffix for a type.

Jon Kalb

This is Why We Can't Have Nice Things

Why C++ in Embedded

- Higher abstraction without overhead
- Type system adds safety
- Allows declarative style
- Supports composition

ciere.com

Part II

Declaritive Thinking

What is declarative?

What to do.

Not how to do it.

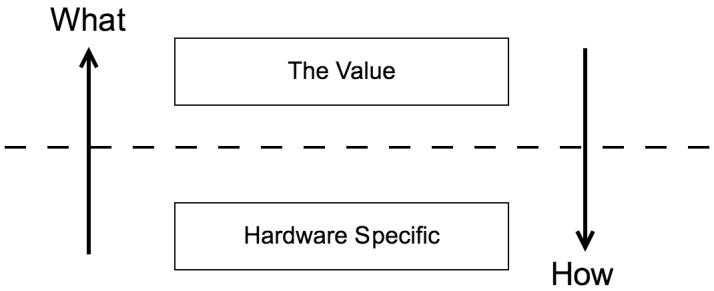
ciere.com

What is declarative?

What to do.

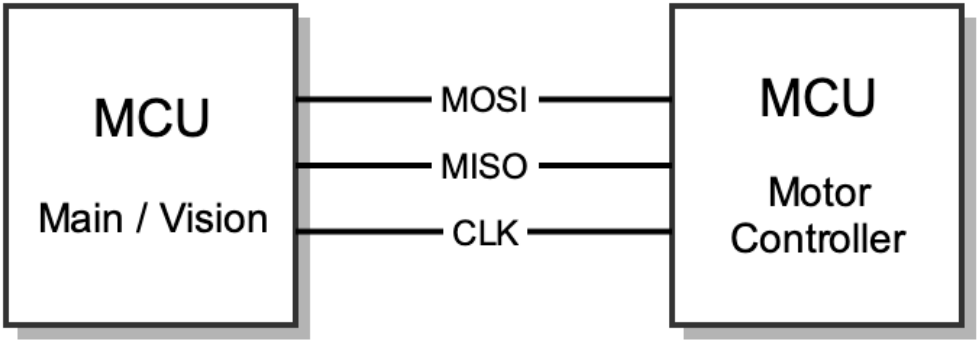
Not how to do it.

Declarative Style



Declarative Style

SPI Serialization



Messaging in C

```
int setThing(struct mcu_s *mcu, int thing, uint16_t value, callback cb, void *ctx)
{
   unsigned length = 4;
   uint8_t data[length];

   data[0] = SET_THING_CMD;
   data[1] = thing;
   memcpy(&data[2], &value, 2);

   return request(mcu->spi_link, data, length, cb, ctx);
}


int setThingMode(struct mcu_s *mcu, int thing, int mode, callback cb, void *ctx)
{
   unsigned length = 3;
   uint8_t data[length];

   data[0] = SET_THING_MODE_CMD;
   data[1] = thing;
   data[2] = mode;

   return request(mcu->spi_link, data, length, cb, ctx);
}
```

Messaging in C++

```cpp
// protocol.hpp

namespace protocol
{
   enum class thing_id : uint8_t
   {
      left, right, other
   };

   struct set_thing
   {
      thing_id thing;
      uint16_t value;
   };

   struct set_thing_mode
   {
      thing_id thing;
      bool mode;
   };
}
```

```cpp
// protocol_adapted.hpp

CIERELABS_ADAPT_STRUCT(
   protocol::set_thing,
   (thing)(value) )

CIERELABS_ADAPT_STRUCT(
   protocol::set_thing_mode,
   (thing)(mode) )

namespace protocol
{
   using message_list = meta_list<
      protocol::set_thing,
      protocol::set_thing_mode
      >;
}
```

Messaging in C++

```cpp
// protocol.hpp

namespace protocol
{
   enum class thing_id : uint8_t
   {
      left, right, other
   };

   struct set_thing
   {
      thing_id thing;
      uint16_t value;
   };

   struct set_thing_mode
   {
      thing_id thing;
      bool mode;
   };
}
```

```cpp
// protocol_adapted.hpp

CIERELABS_ADAPT_STRUCT(
   protocol::set_thing,
   (thing)(value) )

CIERELABS_ADAPT_STRUCT(
   protocol::set_thing_mode,
   (thing)(mode) )

namespace protocol
{
   using message_list = meta_list<
      protocol::set_thing,
      protocol::set_thing_mode
      >;
}
```

Messaging in C++

```
protocol::set_thing_mode set_thing_mode{ protocol::thing_id::left
                                       , false };

// ...

send_message(set_thing_mode);
```

Messaging in C++

```
send_message( protocol::set_thing_mode{ protocol::thing_id::left
                                      , false }  );
```

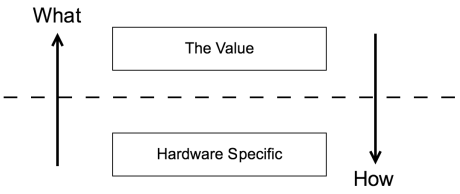
ciere.com

Abstraction - SPI Serialization

C Version LOC : 3292

C++ Version LOC : 1194

Messaging - How it is done




```
enum class thing_id : uint8_t {
   left, right, other
};

struct set_thing_mode {
   thing_id thing;
   bool mode;
};

CIERELABS_ADAPT_STRUCT( set_thing_mode,
   (thing)(mode) )

send_message(set_thing_mode);
```

Magic

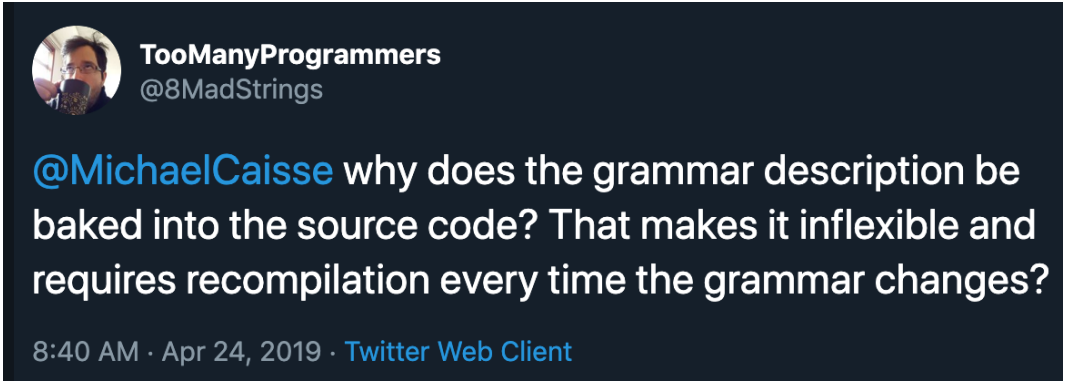
Part III

EDSL ... why?

Why use an EDSL?

Why would you want an
Embedded Domain Specifc Langauge

Why use an EDSL?



Why use an EDSL?

- One stage in build to deal with
- “easily” integrates with code you are writing
- It is still C++
- Recompile if change ... LEX/YACC or generators same
- Run-time loaded descriptions slower
- Run-time loaded still needs a mechanism to handle DSL
- Exposing more to the compiler produces better results

Why use an EDSL?

Declarative & Composition

Why use an EDSL?

Switch statement state machines?

From MQTT client library, formally knows as MaQiaTTo

```
struct transition_table : TransitionTable<
   //     Start              Event                        Next              Action              Guard
   //   +------------------+----------------------------+---------------+-------------------+--------
   Row <  NotConnected     , event::connect             , ConnectBroker , none              , none  >,
   Row <  ConnectBroker    , none                       , Connected     , none              , none  >,
   Row <  Connected        , event::publish_deferred    , none          , send_packet       , none  >,
   Row <  Connected        , event::publish_out         , none          , send_packet       , none  >,
   Row <  Connected        , event::subscribe           , none          , send_packet       , none  >,
   Row <  Connected        , event::unsubscribe         , none          , send_packet       , none  >,
   Row <  Connected        , event::connect             , none          , none              , none  >,
   Row <  Connected        , event::keepalive_timeout   , none          , keepalive_timeout , none  >,
   Row <  Connected        , event::disconnect          , ShuttingDown  , none              , none  >,
   Row <  ShuttingDown     , event::subscribe           , none          , none              , none  >,
   Row <  ShuttingDown     , event::unsubscribe         , none          , none              , none  >,
   Row <  ShuttingDown     , event::disconnect          , none          , none              , none  >,
   Row <  ShuttingDown     , event::publish_out         , none          , none              , none  >,
   Row <  ShuttingDown     , event::keepalive_timeout   , none          , none              , none  >,
   Row <  ShuttingDown     , event::shutdown_timeout    , NotConnected  , none              , none  >
   //   +------------------+----------------------------+---------------+-------------------+--------
   >{};
```

Why use an EDSL?

EDSLs allow us to think in domain specific concepts.

Higher level thinking without a substantial price.

Part IV

State machines

State machines

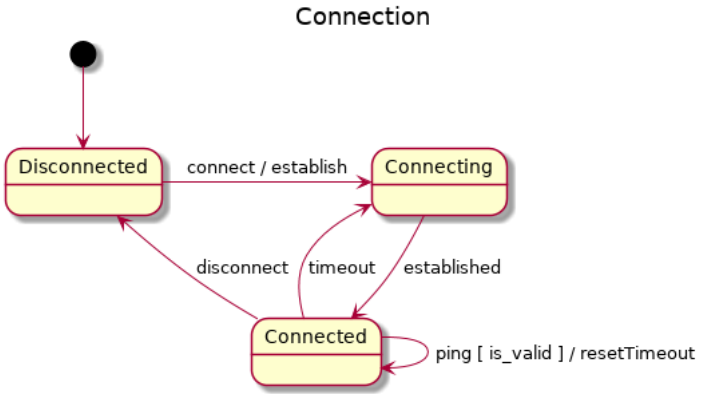
- Boost.MSM
- Boost.SML – Not a Boost library *yet*.

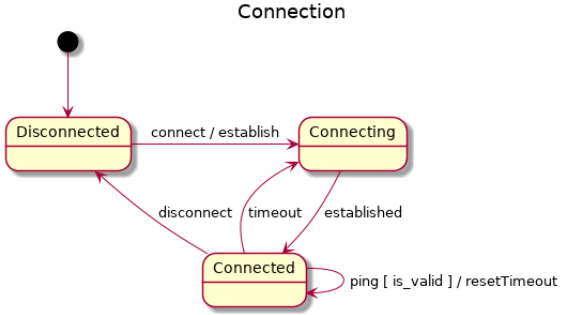
Boost.SML

Name: Boost.SML State Machine Langauge/Lite/Library
What: EDSL for hierarchical state machines
Who: Kris Jusiac

Boost.SML

Name: Boost.SML[3] State Machine Langauge/Lite/Library
What: EDSL for hierarchical state machines
Who: Kris Jusiac


Connection
Disconnected
connect / establish
Connecting
disconnect
timeout
established
Connected
ping [is_valid] / resetTimeout




```
sml::sm connection_machine = [] {
   return transition_table{
      * "Disconnected"_s  + connect / establish                = "Connecting"_s    ,
        "Connecting"_s    + established                        = "Connected"_s     ,
        "Connected"_s     + ping [is_valid]  / resetTimeout                        ,
        "Connected"_s     + timeout / establish                = "Connecting"_s    ,
        "Connected"_s     + disconnect / close                 = "Disconnected"_s
   };
};
```

Boost.SML

```
sml::sm connection_machine = [] {
   return transition_table{
      * "Disconnected"_s  + connect / establish              = "Connecting"_s   ,
        "Connecting"_s    + established                      = "Connected"_s    ,
        "Connected"_s     + ping [is_valid]  / resetTimeout                     ,
        "Connected"_s     + timeout / establish              = "Connecting"_s   ,
        "Connected"_s     + disconnect / close               = "Disconnected"_s
   };
};


const auto establish  = []{ /* something */ };
const auto disconnect = []{ /* something */ };


const auto is_valid   = [](auto event){ return true; };
```

Boost.SML

```
sml::sm connection_machine = [] {
   return transition_table{
      * "Disconnected"_s  + connect / establish              = "Connecting"_s   ,
        "Connecting"_s    + established                      = "Connected"_s    ,
        "Connected"_s     + ping [is_valid]  / resetTimeout                     ,
        "Connected"_s     + timeout / establish              = "Connecting"_s   ,
        "Connected"_s     + disconnect / close               = "Disconnected"_s
   };
};


const auto establish  = []{ /* something */ };
const auto disconnect = []{ /* something */ };


const auto is_valid   = [](auto event){ return true; };
```

Boost.SML

```
sml::sm connection_machine = [] {
   return transition_table{
      * "Disconnected"_s  + connect / establish              = "Connecting"_s   ,
        "Connecting"_s    + established                      = "Connected"_s    ,
        "Connected"_s     + ping [is_valid]  / resetTimeout                     ,
        "Connected"_s     + timeout / establish              = "Connecting"_s   ,
        "Connected"_s     + disconnect / close               = "Disconnected"_s
   };
};


const auto establish  = []{ /* something */ };
const auto disconnect = []{ /* something */ };


const auto is_valid   = [](auto event){ return true; };
```

Usage:

```
connection_machine.process_event(connect{});
```

Boost.SML

Technique	SM Size (bytes)
Naive	3
Enum/Switch	1
SML	1

Kris Jusiak
https://github.com/boost-experimental/sml

ciere.com

Technique	SM Size (bytes)
Naive	3
Enum/Switch	1
SML	1

Kris Jusiak

`https://github.com/boost-experimental/sml`

Part V

Protocols

Protocols

- Spirit
- CTRE

Boost.Spirit

Name: Boost.Spirit
What: EDSL for parsing and generating
Who: Joel de Guzman, Hartmut Kaiser, and Dan Nuffer

ciere.com

Ad-hoc Parsing

```
auto iter = argument.cbegin();
auto iter_end = argument.cend();
while( iter != iter_end )
{
   if( *iter == '+' )
   {
      if( building_key ){ key += ' ';     }
      else              { value += ' ';   }
   }
   else if( *iter == '=' )
   {
      building_key = false;
   }
   else if( *iter == '&' )
   {
      argument_map[ key ] = value;
      key = "";
      value = "";
      building_key = true;
   }
   else if( *iter == '?' )
```

PEG grammar Email (not really)

Parsing Expression Grammar

```
name      <-  [a-z]+ ("." [a-z]+)*
host      <-  [a-z]+ "." ("com" / "org" / "net")
email     <-  name "@" host
```

```
auto name  = +char_("a-z") >> *('.' >> +char_("a-z"));
auto host  = +char_("a-z") >> '.' >> (lit("com") | "org" | "net");
auto email = name >> '@' >> host;
```

CTRE

Name: CTRE Compile Time Regular Expressions
What: EDSL regular expressions
Who: Hana Dusikova

CTRE

```
std::optional<date> extract_date(std::string_view s) noexcept
{
   using namespace ctre::literals;
   if (auto [whole, year, month, day]

           = ctre::match<"^(\\d{4})/(\\d{1,2}+)/(\\d{1,2}+)$">(s);

        whole)
   {
      return date{year, month, day};
   }
   else
   {
      return std::nullopt;
   }
}
```


Hana Dusíková @hankadusikova · Apr 3
I'm going to sleep. You can discuss it in the meantime :)
ctre::search<"\0{5}">(blob);
compiler-explorer.com/z/C-nmBG

Part VI

Making an EDSL

Unicorns

cppcon | 2017
THE C++ CONFERENCE • BELLEVUE, WASHINGTON

@odinthenerd

Tabea
Marie
Paul

Auto-Intern GmbH designed by freepik.com 17

Agent based
class design

CppCon.org

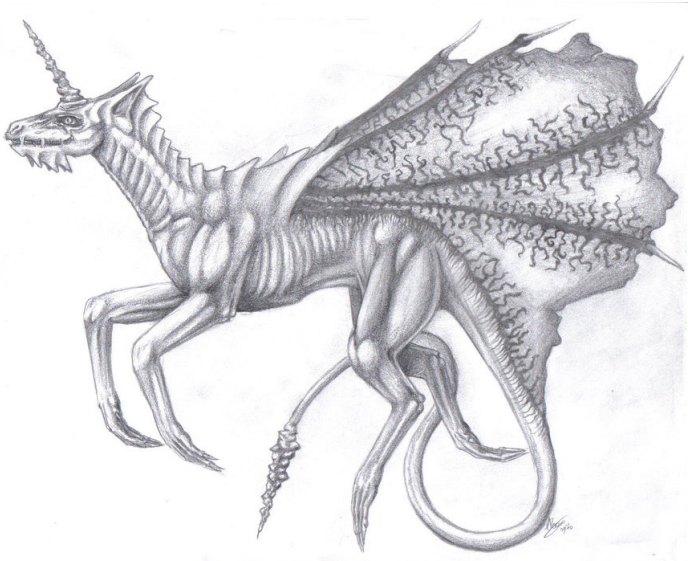
There be Dragons

Abraham Ortelius, Tehatrum Orbis Terrarum, 1570

Unicorn Dragons

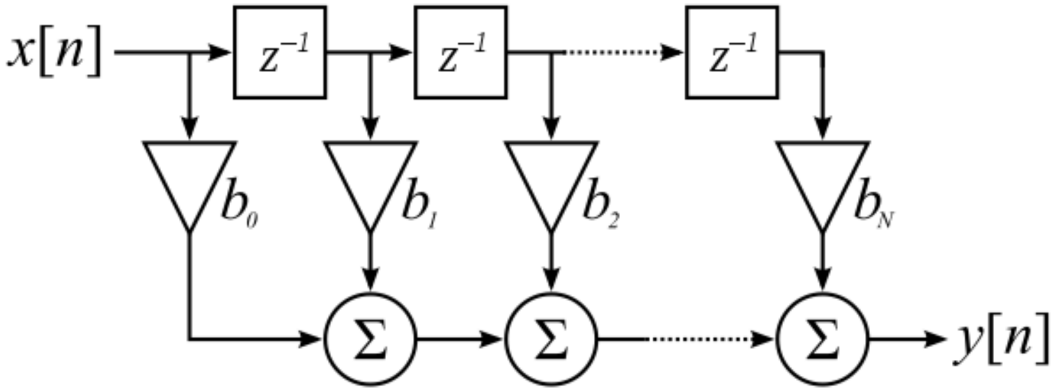
Image: Dragon Mania Legends

Unicorn Dragons

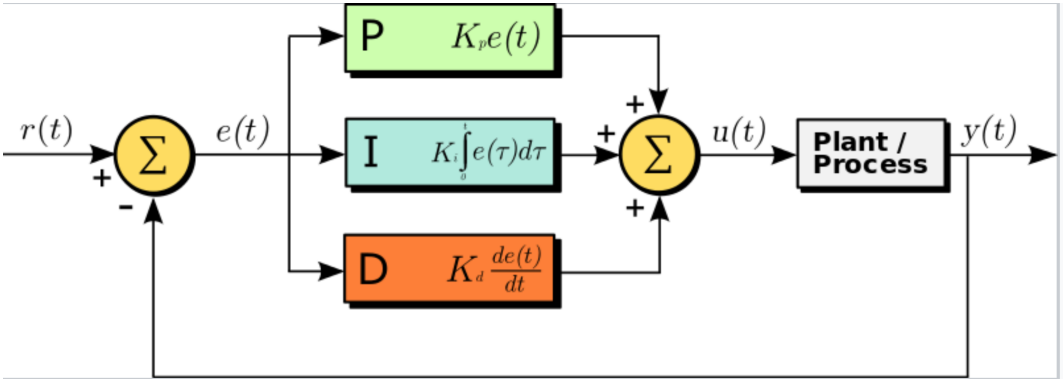


Drawing by: AyoWolf

FIR Filter



PID Control


r(t)
+
-
e(t)
P
$K_p e(t)$
I
$K_i \int_0^t e(\tau)d\tau$
D
$K_d \frac{de(t)}{dt}$
+
+
+
u(t)
Plant / Process
y(t)

Chaining Functions Stinks

```
int foo(int v){ return v*4; }

int bar(int v){ return v/2; }

int gorp(int v){ return v+7; }

int test()
{
    // foo -> bar -> gorp
    return gorp(bar(foo(g)));
}
```

Fix It

```
auto v = gorp(bar(foo(g)));
```

```
auto v = g | foo | bar | gorp;
```

Fix It

```
auto v = gorp(bar(foo(g)));
```

```
auto v = g | foo | bar | gorp;
```

Fix It

```
auto v = gorp(bar(foo(g)));
```

```
auto v = g | foo | bar | gorp;
```

Fix It

```
auto v = gorp(bar(foo(g)));
```

```
auto v = g | foo | bar | gorp;
```

Fix It

```
auto v = gorp(bar(foo(g)));
```

```
auto v = g | foo | bar | gorp;
```

Fix It

```
auto v = gorp(bar(foo(g)));
```

```
auto v = g | foo | bar | gorp;
```

Fix It

```
auto v = gorp(bar(foo(g)));
```

```
auto f = in | foo | bar | gorp;
auto v = f(g);
```

What do we need?

```
auto f = in | foo | bar | gorp;
auto v = f(g);
```

ciere.com

Identity

```
class identity
{
public:
    identity() = default;

    template <typename T>
    auto operator()(T v) const
    {
        return v;
    }
};
```

ciere.com

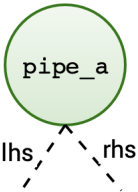
Identity

```
class identity
{
public:
    identity() = default;

    template <typename T>
    auto operator()(T v) const
    {
        return v;
    }
};


 auto in = identity{};
 auto vi = in(8);
 auto vf = in(7.5);
```

Building Pipe

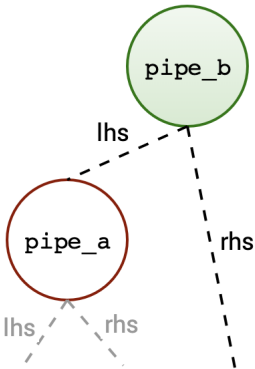
```
auto f = in | foo | bar | gorp;
auto v = f(g);
```

Building Pipe



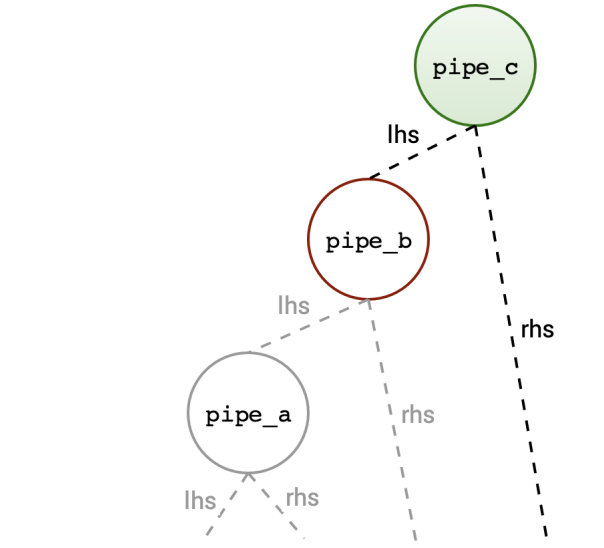

```
auto f = in | foo | bar | gorp;
```

Building Pipe




```
auto f = in | foo | bar | gorp;
```

Building Pipe




```
auto f = in | foo | bar | gorp;
```

Pipe

```
template <typename L, typename R>
class pipe
{
public:
    pipe(L lhs, R rhs)
        : lhs_(lhs), rhs_(rhs)
    {}

    template <typename ... T>
    auto operator()(T... v) const
    {
        return call_rhs(lhs_(v...));
    }

private:
    template <typename ... T>
    auto call_rhs(T && ... v) const
    {
        return rhs_(std::forward<T>(v)...);
    }

    L lhs_;
    R rhs_;
};
```

Pipe

```
template <typename L, typename R>
class pipe
{
public:
    pipe(L lhs, R rhs)
        : lhs_(lhs), rhs_(rhs)
    {}

    template <typename ... T>
    auto operator()(T... v) const
    {
        return call_rhs(lhs_(v...));
    }

private:
    template <typename ... T>
    auto call_rhs(T && ... v) const
    {
        return rhs_(std::forward<T>(v)...);
    }

    L lhs_;
    R rhs_;
};
```

Pipe

```
template <typename L, typename R>
class pipe
{
public:
    pipe(L lhs, R rhs)
        : lhs_(lhs), rhs_(rhs)
    {}

    template <typename ... T>
    auto operator()(T... v) const
    {
        return call_rhs(lhs_(v...));
    }

private:
    template <typename ... T>
    auto call_rhs(T && ... v) const
    {
        return rhs_(std::forward<T>(v)...);
    }

    L lhs_;
    R rhs_;
};
```

Pipe

```
template <typename L, typename R>
class pipe
{
public:
    pipe(L lhs, R rhs)
        : lhs_(lhs), rhs_(rhs)
    {}

    template <typename ... T>
    auto operator()(T... v) const
    {
        return call_rhs(lhs_(v...));
    }

private:
    template <typename ... T>
    auto call_rhs(T && ... v) const
    {
        return rhs_(std::forward<T>(v)...);
    }

    L lhs_;
    R rhs_;
};
```

Pipe

```
template <typename L, typename R>
class pipe
{
public:
   pipe(L lhs, R rhs)
      : lhs_(lhs), rhs_(rhs)
   {}

   template <typename ... T>
   auto operator()(T... v) const
   {
      return call_rhs(lhs_(v...));
   }

private:
   template <typename ... T>
   auto call_rhs(T && ... v) const
   {
      return rhs_(std::forward<T>(v)...);
   }

   L lhs_;
   R rhs_;
};
```

Pipe Operator

```
template <typename L, typename R>
auto operator|(L lhs, R rhs)
{
    return pipe<L,R>(lhs, rhs);
}
```

Tainting the line-up

The wonders of ADL - Argument Dependent Lookup

```
namespace filter
{
    class identity
    {
        // ...
    };

    template <typename L, typename R>
    class pipe
    {
        // ...
    };

    template <typename L, typename R>
    auto operator|(L lhs, R rhs);
}
```

The Result

```
int foo(int v){ return v*4; }

int bar(int v){ return v/2; }

int gorp(int v){ return v+7; }

int test()
{
   auto in = filter::identity{};
   auto f =  in | foo | bar | gorp;
   return f(g);
}
```

Comparison

```
int foo(int v){ return v*4; }

int bar(int v){ return v/2; }

int gorp(int v){ return v+7; }

int test()
{
    // foo -> bar -> gorp
    return gorp(bar(foo(g)));
}
```

```
int foo(int v){ return v*4; }

int bar(int v){ return v/2; }

int gorp(int v){ return v+7; }

int test()
{
   auto in = filter::identity{};
   auto f =  in | foo | bar | gorp;
   return f(g);
}
```

Generated Assembly

Nested Functions : gcc 8.2 ARM64 -O3

```
1  foo(int):
2          lsl     w0, w0, 2
3          ret
4  bar(int):
5          add     w0, w0, w0, lsr 31
6          asr     w0, w0, 1
7          ret
8  gorp(int):
9          add     w0, w0, 7
10         ret
11 test():
12         adrp    x0, g
13         ldr     w0, [x0, #:lo12:g]
14         lsl     w0, w0, 2
15         asr     w0, w0, 1
16         add     w0, w0, 7
17         ret
```

Pipe Operators : gcc 8.2 ARM64 -O3

```
1  foo(int):
2          lsl     w0, w0, 2
3          ret
4  bar(int):
5          add     w0, w0, w0, lsr 31
6          asr     w0, w0, 1
7          ret
8  gorp(int):
9          add     w0, w0, 7
10         ret
11 test():
12         adrp    x0, g
13         ldr     w0, [x0, #:lo12:g]
14         lsl     w0, w0, 2
15         asr     w0, w0, 1
16         add     w0, w0, 7
17         ret
```

Annonymous Namespace

```
namespace {
int foo(int v){ return v*4; }

int bar(int v){ return v/2; }

int gorp(int v){ return v+7; }
}

int test()
{
   auto in = filter::identity{};
   auto f =  in | foo | bar | gorp;
   return f(g);
}
```

```
1 test():
2         adrp    x0, g
3         ldr     w0, [x0, #:lo12:g]
4         lsl     w0, w0, 2
5         asr     w0, w0, 1
6         add     w0, w0, 7
7         ret
```

constexpr all the things

```
class identity
{
public:
   constexpr identity() = default;

   template <typename T>
   constexpr auto operator()(T v) const
   {
      return v;
   }
};
```

```
template <typename L, typename R>
constexpr auto operator|(L lhs, R rhs)
{
   return pipe<L,R>(lhs, rhs);
}
```

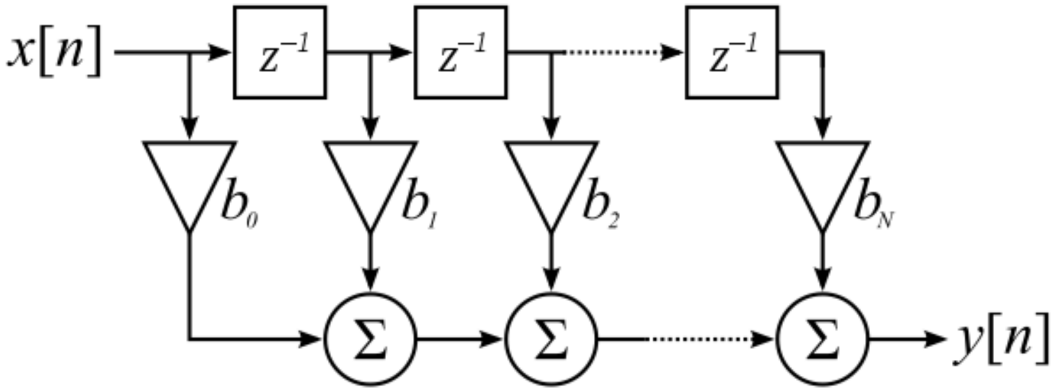
```
template <typename L, typename R>
class pipe
{
public:
   constexpr pipe(L lhs, R rhs)
      : lhs_(lhs), rhs_(rhs)
   {}

   template <typename ... T>
   constexpr auto operator()(T... v) const
   {
      return call_rhs(lhs_(v...));
   }

private:
   template <typename ... T>
   constexpr auto call_rhs(T && ... v) const
   {
      return rhs_(std::forward<T>(v)...);
   }

   L lhs_;
   R rhs_;
};
```

constexpr all the things

```
constexpr int foo(int v){ return v*4; }

constexpr int bar(int v){ return v/2; }

constexpr int gorp(int v){ return v+7; }

int main()
{
   auto in = filter::identity{};
   constexpr auto f =  in | foo | bar | gorp;
   std::array<float, f(4)> a;
   return a.size();
}
```

```
1 main:
2         mov     w0, 15
3         ret
```

FIR Filter



Constant

```
template <typename T=float>
class constant
{
public:

    constexpr constant(T v)
        : v_(v)
    {}


    template <typename ... Args>
    constexpr auto operator()(Args ...) const
    {
        return v_;
    }


private:
    T const v_;
};
```

Constant

```
template <typename T=float>
class constant
{
public:

   constexpr constant(T v)
      : v_(v)
   {}


   template <typename ... Args>
   constexpr auto operator()(Args ...) const
   {
      return v_;
   }


private:
   T const v_;
};
```

Constant

```
template <typename T=float>
class constant
{
public:

   constexpr constant(T v)
      : v_(v)
   {}


   template <typename ... Args>
   constexpr auto operator()(Args ...) const
   {
      return v_;
   }


private:
   T const v_;
};
```

Constant

```
float test1()
{
    auto c = filter::constant{42.3};

    auto f = c | [](int v){ return v*2; };

    return f();
}
```

```
1 test1():
2         mov     w0, 1118306304
3         fmov    s0, w0
4         ret
```

User Defined Literals - UDLs

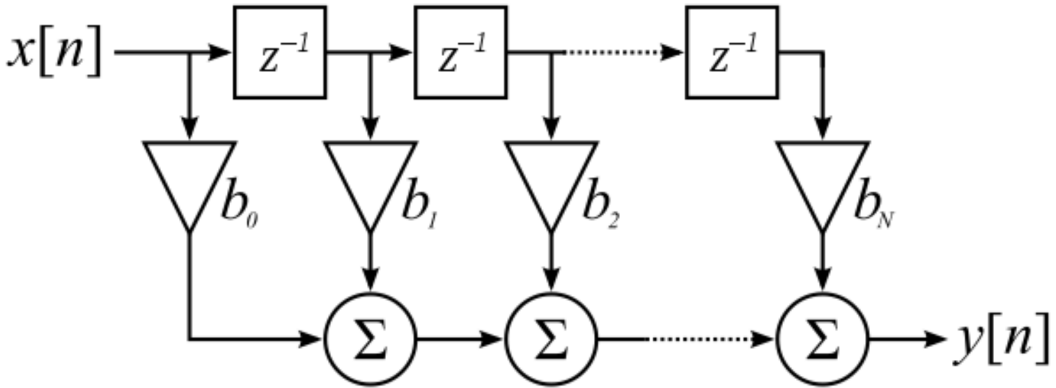
```
namespace literals
{
   constexpr constant<long double> operator ""_K(long double v)
   {
      return constant{v};
   }

   constexpr constant<unsigned long long> operator ""_K(unsigned long long v)
   {
      return constant{v};
   }
}
```

Constant - UDLs

```
float test2()
{
    using namespace filter::literals;

    auto f = 42.3_K | [](int v){ return v*2; };

    return f();
}
```

```
1  test2():
2          mov     w0, 1118306304
3          fmov    s0, w0
4          ret
```

FIR Filter



Delay Setup

```
auto in = filter::identity{};
auto f =
     in
   | [](auto v){return v*2;}
   | printer{};


for(auto v : {1.0, 2.0, 3.0, 4.0, 5.0})
{
   f(v);
}
```

```
[2 ]
[4 ]
[6 ]
[8 ]
[10 ]
```

Delay

```
template <typename T, int Z>
class delay
{
public:

   constexpr delay()     { z_buffer_.fill(T{}); }
   constexpr delay(T i) { z_buffer_.fill(i); }


   constexpr auto operator()(T const & v)
   {
      z_buffer_[head_] = v;
      if(++head_ > Z) head_ = 0;

      if(++tail_ > Z) tail_ = 0;
      return z_buffer_[tail_];
   }


private:
   using z_buffer_t = std::array<T,Z+1>;
   typename z_buffer_t::size_type head_ = 0;
   typename z_buffer_t::size_type tail_ = 0;
   z_buffer_t z_buffer_;
};
```

Delay

```
template <typename T, int Z>
class delay
{
public:

   constexpr delay()     { z_buffer_.fill(T{}); }
   constexpr delay(T i) { z_buffer_.fill(i); }


   constexpr auto operator()(T const & v)
   {
      z_buffer_[head_] = v;
      if(++head_ > Z) head_ = 0;

      if(++tail_ > Z) tail_ = 0;
      return z_buffer_[tail_];
   }


private:
   using z_buffer_t = std::array<T,Z+1>;
   typename z_buffer_t::size_type head_ = 0;
   typename z_buffer_t::size_type tail_ = 0;
   z_buffer_t z_buffer_;
};
```

Delay

```
template <typename T, int Z>
class delay
{
public:

   constexpr delay()     { z_buffer_.fill(T{}); }
   constexpr delay(T i) { z_buffer_.fill(i); }


   constexpr auto operator()(T const & v)
   {
      z_buffer_[head_] = v;
      if(++head_ > Z) head_ = 0;

      if(++tail_ > Z) tail_ = 0;
      return z_buffer_[tail_];
   }


private:
   using z_buffer_t = std::array<T,Z+1>;
   typename z_buffer_t::size_type head_ = 0;
   typename z_buffer_t::size_type tail_ = 0;
   z_buffer_t z_buffer_;
};
```

Delay

```
template <typename T, int Z>
class delay
{
public:

   constexpr delay()     { z_buffer_.fill(T{}); }
   constexpr delay(T i) { z_buffer_.fill(i); }


   constexpr auto operator()(T const & v)
   {
      z_buffer_[head_] = v;
      if(++head_ > Z) head_ = 0;

      if(++tail_ > Z) tail_ = 0;
      return z_buffer_[tail_];
   }


private:
   using z_buffer_t = std::array<T,Z+1>;
   typename z_buffer_t::size_type head_ = 0;
   typename z_buffer_t::size_type tail_ = 0;
   z_buffer_t z_buffer_;
};
```

Delay

```
auto in = filter::identity{};
auto f =
     in
   | filter::delay<float,1>{}
   | [](auto v){return v*2;}
   | printer{};


for(auto v : {1.0, 2.0, 3.0, 4.0, 5.0})
{
   f(v);
}
```

```
[0 ]
[2 ]
[4 ]
[6 ]
[8 ]
```

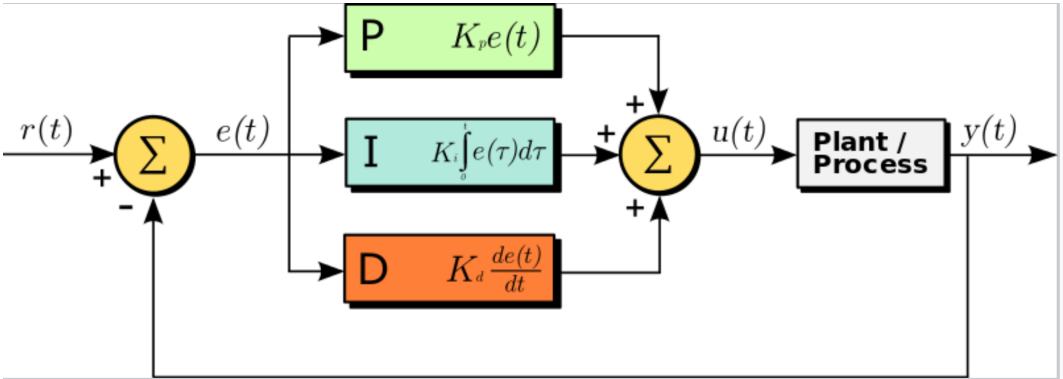
Delay - Z

```
template <int I, typename T=float>
constexpr auto Z(T v = T{})
{
   static_assert(I<0, "Z delay should be a negative number");
   return delay<T, I*-1>{v};
}
```

Delay - Z

```
auto in = filter::identity{};
auto f =
      in
    | Z<-1>()
    | [](auto v){return v*2;}
    | printer{};


for(auto v : {1.0, 2.0, 3.0, 4.0, 5.0})
{
    f(v);
}
```

PID Control


r(t)
+
-
e(t)
P
$K_p e(t)$
I
$K_i \int_0^t e(\tau)d\tau$
D
$K_d \frac{de(t)}{dt}$
+
+
+
u(t)
Plant / Process
y(t)

Fork

```
void test1()
{
    auto in = filter::identity{};

    auto f = in | fork(   [](auto v){ return v*1; }
                        ,  [](auto v){ return v*2; }
                        ,  [](auto v){ return v*3; }
                        ,  [](auto v){ return v*4; }
        );
}
```

Fork

```
template <typename ... T>
auto fork(T && ... v)
{
    return detail::fork_impl<T...>{std::forward<T>(v)...};
}
```

ciere.com

Fork

```
template <typename ... T>
class fork_impl
{
public:
    fork_impl(T && ... v)
        : nodes_(std::forward<T>(v)...)
    {}

    template <typename ... Tn>
    void operator()(Tn ... v) const
    {
        for_each_fork(nodes_, v...);
    }

private:
    std::tuple<T...> nodes_;
};
```

Fork

```
template < typename Tuple
         , typename ... Tn >
void for_each_fork(Tuple && tuple, Tn ... v)
{
   constexpr std::size_t N
      = std::tuple_size<std::remove_reference_t<Tuple>>::value;

   for_each_fork_impl( std::forward<Tuple>(tuple)
                     , std::make_index_sequence<N>{}
                     , v... );
}
```

ciere.com

Fork

```
template <typename Tuple, std::size_t ... Indices, typename ... Tn>
void for_each_fork_impl( Tuple && tuple
                       , std::index_sequence<Indices...>
                       , Tn ... v)
{
    (std::get<Indices>(std::forward<Tuple>(tuple))(v...), ...);
}
```

ciere.com

Thoughts

- Downsides : compile time
- Not C++ ... that is the point
- Learning another library

Thoughts

- Delaritive
- Correct by composition
- Speed and Size are not issues

Questions?