C++ now

Newer Isn't Always Better

Investigating Legacy Design Trends and Their Modern Replacements

Katherine Rocha

2024

Newer Isn’t Always Better

Investigating Legacy Design Trends and Their Modern Replacements

Katherine Rocha

About Me

- Software Engineer at Atomos Space
- Working in a C++20 Codebase with approximately 100,000 lines of C++
- Previously Worked in a 20+ Year Old Codebase
- “Software Historian/Genealogist”

Initial Discovery

- Understanding the past
- Investigating the new patterns with the same scrutiny as the old
- Tend to make our initial evaluation and stick with it
- Is it a Fad or is it good?

Investigative Process

Timeline

- When was the original trend introduced?
- When did the trend transition?

Original Trend

- What is the original trend?
- Why is it used?

New Trend

- What is the new trend?
- Why did it replace the original?

Original Code

- What is the original solution?
- How elegant is it?
- What are the problems with it?

New Code

- What is the new solution?
- How elegant is it?
- What are the problems with it?

Analysis

- Pros and Cons of the original trend
- Pros and Cons of the new trend
- Comparison of the trends

Global Interfaces/Global State

Use Cases

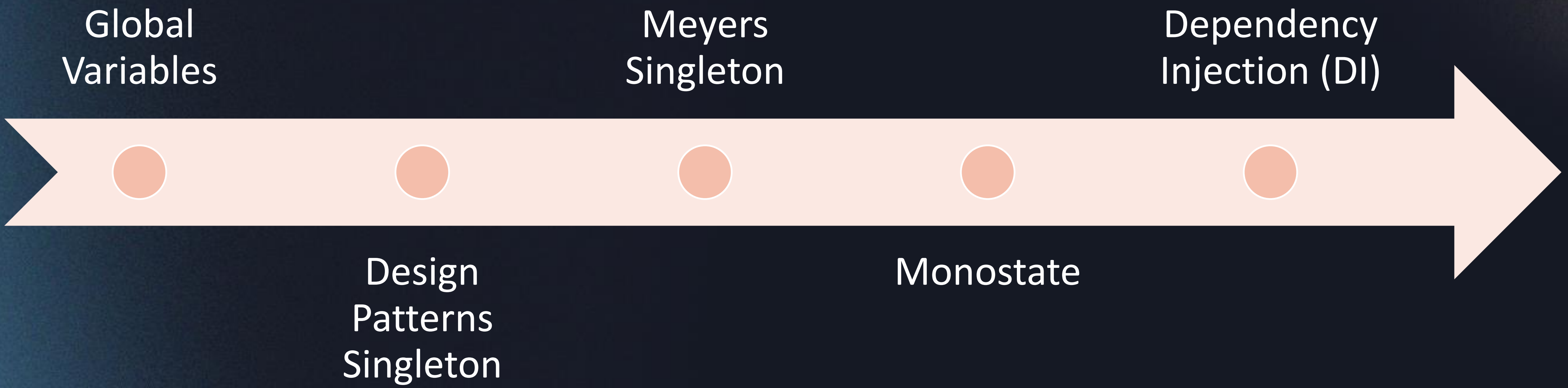
Global Interface

- Logging
- External I/O
- Resource Management
- Plotting

Global Data

- Initial Parameters
- State Parameters

Timeline


Global Variables
Design Patterns Singleton
Meyers Singleton
Monostate
Dependency Injection (DI)

Original Trend - Singleton

- Hold one copy of global data/interface and allow others access
- Usually accessed through a `getInstance()` or `Instance()` function
- Easily accessed
- Identifiable
- Hard to test
- Quintessentially Overused

Original Code – Design Patterns Singleton vs Meyers' Singleton

```cpp
class PlottingSingleton
{
    public:
        static PlottingSingleton* getInstance()
        {
            if (!instance) // race condition
                instance = new PlottingSingleton;
            return instance;
        }

        void plot(double x, double y)
        {
            // ...
        }

    protected:
        PlottingSingleton();

    private:
        inline static PlottingSingleton* instance {NULL};
};
```

```cpp
class PlottingSingleton
{
    public:
        static PlottingSingleton& getInstance()
        {
            static PlottingSingleton instance {};
            return instance;
        }

        void plot(double x, double y)
        {
            // ...
        }

    private:
        PlottingSingleton();
};
```

Original Code – Singleton Wrapper

```cpp
template <typename T>
class Singleton
{
    public:
        static T& getInstance()
        {
            static T instance;
            return instance;
        }

    private:
        Singleton();
};

class Plotting
{
    public:
        void plot(double x, double y)
        {
            // ...
        }
};

using PlottingSingleton = Singleton<Plotting>;
```

New Trend – Monostate

- Make every object in the class static
- Multiple objects all with the same value
- Easy to transition to multiple objects
- May not work well to replace interface singletons

New Code – Monostate

```cpp
class Plotting
{
    public:
        void plot(double x, double y)
        {
            // ...
        }
    private:
        static std::queue plottingQueue;
};
```

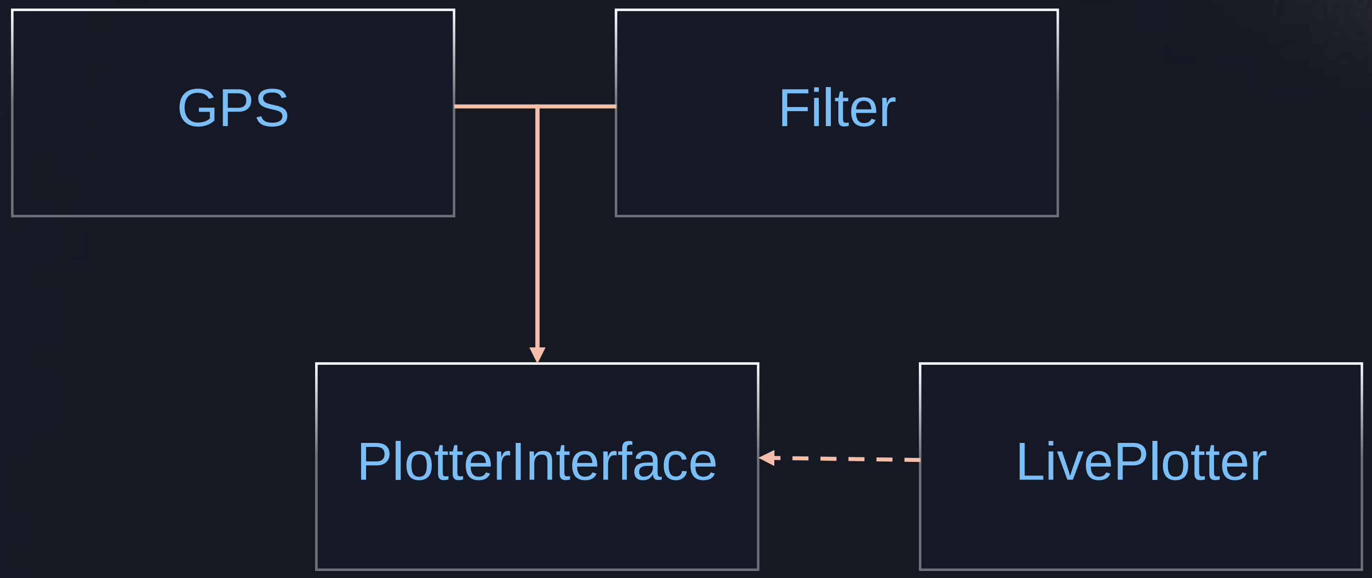
New Trend – Dependency Injection

- Not a global object
- Injects the dependency into each of the using objects

Aside: Dependency Injection (DI) Vs Dependency Inversion Principle (DIP)

Dependency Inversion Principle (DIP)

- Eliminates the dependency by inverting and adding an interface class
- Reduces volatility due to implementation
- Allows for testing and mocking



Dependency Injection (DI)

- Inject the dependency into the object
  - Injected 3 ways
    - Interface/Template Parameter Injection (Type 1)
    - Setter (Type 2)
    - Constructor (Type 3)
  - One Object being shared

New Code – Dependency Injection

```
class Plotting
{
    public:
        void plot(double x, double y)
        {
            // ...
        }
};

class Gps
{
    public:
        Gps(Plotting plotter&);
        void setPlotter(Plotting plotter&);
        void getPositionVelocityAcceleration(Plotting plotter&);
    private:
        Plotting& plotter;
};
```

Comparison

Singleton	Monostate	Dependency Injection (DI)
• Easy to Recognize • Easy Access	• Non-Intuitive Shared Access • Easy Transition to Individual Objects • Less Powerful than the Singleton?	• Explicit Access

SFINAE & Concepts

Use Case

- Function Requirements
- Breaking SOONER in compile time

Usage Example

Runge-Kutta 4 – approximate solution to nonlinear equations

```
inline constexpr double runge_kutta4(std::function<double(double, double)> fun,
                                     double time,
                                     double y0,
                                     double timestep)
{
    auto k1 = fun(time, y0);
    auto k2 = fun(time + timestep * 0.5, y0 + k1 * timestep * 0.5);
    auto k3 = fun(time + timestep * 0.5, y0 + k2 * timestep * 0.5);
    auto k4 = fun(time + timestep, y0 + k3 * timestep);

    return (y0 + (k1 + 2 * k2 + 2 * k3 + k4) * timestep / 6);
}
```

```
double stateOut = common::math::runge_kutta4<double, double>(derivFun, currTime, stateIn, dt);
```

Usage Example Continued

```cpp
inline constexpr Eigen::Matrix<double, 1, 6> runge_kutta4(std::function<Eigen::Matrix<double, 1, 6>(double, Eigen::Matrix<double, 1, 6>)> fun,
                                                          double time,
                                                          Eigen::Matrix<double, 1, 6> y0,
                                                          double timestep)
{
    auto k1 = fun(time, y0);
    auto k2 = fun(time + timestep * 0.5, y0 + k1 * timestep * 0.5);
    auto k3 = fun(time + timestep * 0.5, y0 + k2 * timestep * 0.5);
    auto k4 = fun(time + timestep, y0 + k3 * timestep);

    return (y0 + (k1 + 2 * k2 + 2 * k3 + k4) * timestep / 6);
}

Eigen::Matrix<double, 1, 6> stateOut = common::math::runge_kutta4<double, Eigen::Matrix<double, 1, 6>>(derivFun, currTime, stateIn, dt);
```

Usage Example Continued

```
template <typename Time, typename OutputType>
inline constexpr OutputType runge_kutta4(std::function<OutputType(Time, OutputType)> fun,
                                         Time time,
                                         OutputType y0,
                                         Time timestep)
{
    auto k1 = fun(time, y0);
    auto k2 = fun(time + timestep * 0.5, y0 + k1 * timestep * 0.5);
    auto k3 = fun(time + timestep * 0.5, y0 + k2 * timestep * 0.5);
    auto k4 = fun(time + timestep, y0 + k3 * timestep);

    return (y0 + (k1 + 2 * k2 + 2 * k3 + k4) * timestep / 6);
}

Eigen::Matrix<double, 1, 6> stateOut = common::math::runge_kutta4<double, Eigen::Matrix<double, 1, 6>>(derivFun, currTime,
stateInOut, dt);
```

Compiler Error Output

```
double stateOut = common::math::runge_kutta4<double, std::string>(fun, currTime, std::string(), dt);
```

```
[build] runge_kutta4.hpp:16:50: error: invalid operands to binary expression ('std::basic_string<char>' and 'double')
[build]    16 |     auto k2 = fun(time + timestep * 0.5, y0 + k1 * timestep * 0.5);
[build]       |                                          ~~ ^ ~~~~~~~~~
[build] example.cpp:145:23: note: in instantiation of function template specialization 'common::math::runge_kutta4<double,
std::basic_string<char>>' requested here
[build]   145 |         common::math::runge_kutta4<double, std::string>(derivFun, currFiltTime, std::string(), dt);
[build]       |                       ^
[build] /usr/bin/../lib/gcc/x86_64-linux-gnu/12/../../../../include/c++/12/complex:392:5: note: candidate template ignored: could not
match 'complex' against 'basic_string'
[build]   392 |     operator*(const complex<_Tp>& __x, const complex<_Tp>& __y)
[build]       |     ^
[build] /usr/bin/../lib/gcc/x86_64-linux-gnu/12/../../../../include/c++/12/complex:401:5: note: candidate template ignored: could not
match 'complex' against 'basic_string'
[build]   401 |     operator*(const complex<_Tp>& __x, const _Tp& __y)
[build]       |     ^
[build] /usr/bin/../lib/gcc/x86_64-linux-gnu/12/../../../../include/c++/12/complex:410:5: note: candidate template ignored: could not
match 'complex<_Tp>' against 'double'
[build]   410 |     operator*(const _Tp& __x, const complex<_Tp>& __y)
[build]       |     ^
[build] example.cpp:144:12: error: no viable conversion from 'std::basic_string<char>' to 'double'
[build]   144 |     double stateOut =
[build]       |            ^
[build]   145 |         common::math::runge_kutta4<double, std::string>(derivFun, currFiltTime, std::string(), dt);
[build]       |         ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[build] /usr/bin/../lib/gcc/x86_64-linux-gnu/12/../../../../include/c++/12/bits/basic_string.h:944:7: note: candidate function
[build]   944 |       operator __sv_type() const noexcept
[build]       |       ^
[build] 2 errors generated.
```

Timeline

SFINAE

- ~2002

Type Traits Introduced

- C++11

Concepts

- C++20

Original Trend (SFINAE)

- Substitution Failure Is Not An Error
- Constraints on templates
- Known for difficult to read errors
- Difficult to constrain

Original Code - SFINAE

```
template <typename Time, typename OutputType, typename = std::enable_if_t<std::is_arithmetic_v<Time>>>
inline constexpr OutputType runge_kutta4(std::function<OutputType(Time, OutputType)> fun,
                                         Time time,
                                         OutputType y0,
                                         Time timestep)
{
    auto k1 = fun(time, y0);
    auto k2 = fun(time + timestep * 0.5, y0 + k1 * timestep * 0.5);
    auto k3 = fun(time + timestep * 0.5, y0 + k2 * timestep * 0.5);
    auto k4 = fun(time + timestep, y0 + k3 * timestep);

    return (y0 + (k1 + 2 * k2 + 2 * k3 + k4) * timestep / 6);
}
```

We also want to constrain OutputType…

Original Code – SFINAE Continued

```cpp
#include <boost/type_traits/has_operator.hpp>

template <typename Time,
          typename OutputType,
          typename = std::enable_if_t<std::is_arithmetic_v<Time>>,
          typename = std::enable_if_t<std::is_arithmetic_v<OutputType> ||
                                      (boost::has_multiplies<OutputType, Time>::value &&
                                       boost::has_plus<OutputType>::value)>>
inline constexpr OutputType runge_kutta4(std::function<OutputType(Time, OutputType)> fun,
                                         Time time,
                                         OutputType y0,
                                         Time timestep)
{
    auto k1 = fun(time, y0);
    auto k2 = fun(time + timestep * 0.5, y0 + k1 * timestep * 0.5);
    auto k3 = fun(time + timestep * 0.5, y0 + k2 * timestep * 0.5);
    auto k4 = fun(time + timestep, y0 + k3 * timestep);

    return (y0 + (k1 + 2 * k2 + 2 * k3 + k4) * timestep / 6);
}
```

Compiler Error Output

```
double stateOut = common::math::runge_kutta4<double, std::string>(fun, currTime, std::string(), dt);

[build] example.cpp:144:9: error: no matching function for call to 'runge_kutta4'
[build]   144 |         common::math::runge_kutta4<double, std::string>(fun, currTime, std::string(), dt);
[build]       |         ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[build] runge_kutta4.hpp:16:22: note: candidate template ignored: requirement 'std::is_arithmetic_v<std::basic_string<char,
std::char_traits<char>, std::allocator<char>>> || (boost::has_multiplies<std::basic_string<char, std::char_traits<char>,
std::allocator<char>>, double, boost::binary_op_detail::dont_care>::value && boost::has_plus<std::basic_string<char,
std::char_traits<char>, std::allocator<char>>, std::basic_string<char, std::char_traits<char>, std::allocator<char>>,
boost::binary_op_detail::dont_care>::value)' was not satisfied [with Time = double, OutputType = std::string, $2 =
std::enable_if_t<std::is_arithmetic_v<double>>]
[build]    16 | constexpr OutputType runge_kutta4(std::function<OutputType(Time, OutputType)> fun,
[build]       |                      ^
```

New Trend (Concepts)

- Compile Time constraints
- Named set of requirements
- Improved compiler errors
- Easier to create custom constraints for

New Code - Concepts

```cpp
template<typename T>
concept arithmetic = std::integral<T> || std::floating_point<T>;

template <arithmetic Time, typename OutputType>
inline constexpr OutputType runge_kutta4(std::function<OutputType(Time, OutputType)> fun,
                                         Time time,
                                         OutputType y0,
                                         Time timestep)
{
    auto k1 = fun(time, y0);
    auto k2 = fun(time + timestep * 0.5, y0 + k1 * timestep * 0.5);
    auto k3 = fun(time + timestep * 0.5, y0 + k2 * timestep * 0.5);
    auto k4 = fun(time + timestep, y0 + k3 * timestep);

    return (y0 + (k1 + 2 * k2 + 2 * k3 + k4) * timestep / 6);
}
```

New Code – Concepts Continued

```
template<typename T>
concept arithmetic = std::integral<T> || std::floating_point<T>;

template<class T, typename Num>
concept add_multiply = requires(T t, Num num)
{
    t * num;
    t + t;
};

template <arithmetic Time, typename OutputType>
requires (add_multiply<OutputType, Time>)
inline constexpr OutputType runge_kutta4(std::function<OutputType(Time, OutputType)> fun,
                                         Time time,
                                         OutputType y0,
                                         Time timestep)

{
    auto k1 = fun(time, y0);
    auto k2 = fun(time + timestep * 0.5, y0 + k1 * timestep * 0.5);
    auto k3 = fun(time + timestep * 0.5, y0 + k2 * timestep * 0.5);
    auto k4 = fun(time + timestep, y0 + k3 * timestep);

    return (y0 + (k1 + 2 * k2 + 2 * k3 + k4) * timestep / 6);
}
```

Compiler Error Output

```cpp
double stateOut = common::math::runge_kutta4<double, std::string>(derivFun, currTime, std::string(), dt);
```

```
[build] example.cpp:144:9: error: no matching function for call to 'runge_kutta4'
[build]   144 |         common::math::runge_kutta4<double, std::string>(derivFun, currTime, std::string(), dt);
[build]       |         ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[build] runge_kutta4.hpp:22:29: note: candidate template ignored: constraints not satisfied [with Time = double, OutputType =
std::string]
[build]    22 | inline constexpr OutputType runge_kutta4(std::function<OutputType(Time, OutputType)> fun,
[build]       |                             ^
[build] runge_kutta4.hpp:21:11: note: because 'add_multiply<std::basic_string<char>, double>' evaluated to false
[build]    21 | requires (add_multiply<OutputType, Time>)
[build]       |           ^
[build] runge_kutta4.hpp:16:7: note: because 't * num' would be invalid: invalid operands to binary expression
('std::basic_string<char>' and 'double')
[build]    16 |     t * num;
[build]       |       ^
[build] 1 error generated.
```

Comparison

SFINAE

- Hard to Read Error Messages
- Difficult to Make Complicated Checks

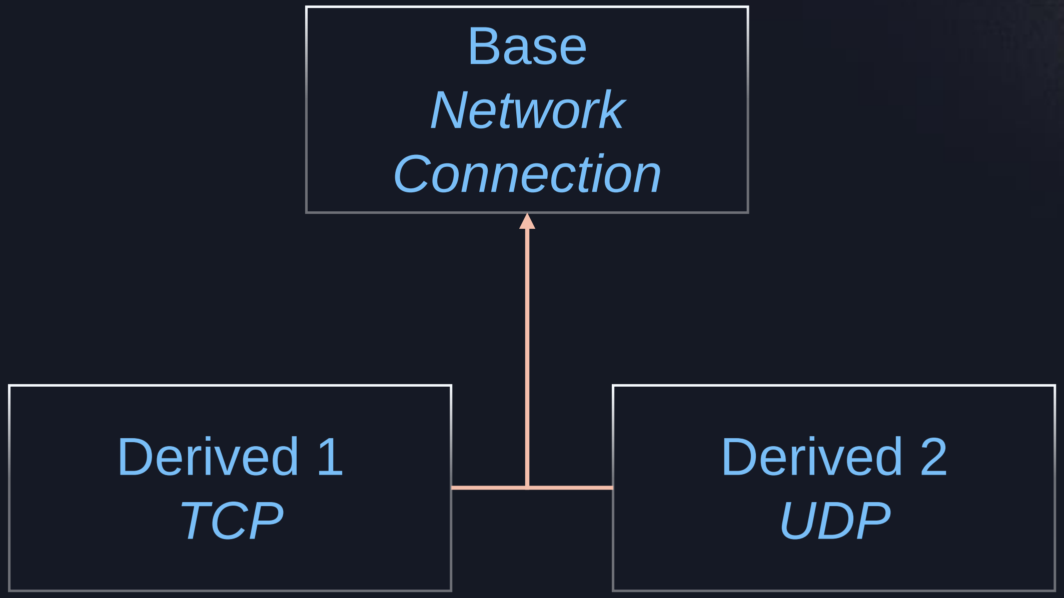
Concepts

- Replaced SFINAE
- Easy to Read Error Messages
- Easy to Make Custom Checks
- Easy to Read Checks

Polymorphism

Use Case

- One interface with multiple implementations
- Key Object-Oriented Design method
- Implementation for Don’t Repeat Yourself (DRY)



Timeline

Virtual Functions

- ~1986

Curiously Recurring Template Pattern (CRTP)

- ~1995

Deducing This

- C++23

Original Trend – Virtual Functions

- Run-Time Polymorphism
- Quintessential Object Oriented Method
- Overused

Original Code – Virtual Functions

```cpp
struct NetworkConnection
{
    virtual void initializeConfig() = 0; // Pure Virtual

    void init()
    {
        initializeConfig();
        // ...
    };
};
```

```cpp
struct Tcp : public NetworkConnection
{
    void initializeConfig() override
    {
        // ...
    }
};
```

```cpp
struct Udp : public NetworkConnection
{
    void initializeConfig() override
    {
        // ...
    }
};
```

New Trend – Curiously Recurring Template Pattern (CRTP)

- Compile Time Polymorphism
- Force a Downcast from the Parent to Access Child Elements
- Explicit Cast

New Code – CRTP

```
template <class derived>
class NetworkConnection
{
    public:
        void init()
        {
            (static_cast<derived*>(this))->initializeConfig();
            // ...
        };
};
```

```
class Tcp : public NetworkConnection<Tcp>
{
    public:
        void initializeConfig()
        {
            // ...
        }
};
```

```
class Udp : public NetworkConnection<Udp>
{
    public:
        void initializeConfig()
        {
            // ...
        }
};
```

New Trend – Deducing This

- C++23 Feature
- Simplifies Compile Time Polymorphism

New Code – Deducing This

```
struct NetworkConnection
{
    public:
        void init(this auto&& self)
        {
            self.initializeConfig();
            // ...
        };
};

class Tcp : public NetworkConnection
{
    public:
        void initializeConfig()
        {
            // ...
        }
};
```

```
class Udp : public NetworkConnection
{
    public:
        void initializeConfig()
        {
            // ...
        }
};
```

Multi-Level Inheritance – Virtual Attempt

```cpp
// https://godbolt.org/z/T51xE5qbK
struct NetworkConnection
{
    virtual void initializeConfig() = 0; // Pure Virtual

    void init()
    {
        initializeConfig();
        // ...
    };
};


struct Tcp : public NetworkConnection
{
    void initializeConfig() override
    {
        std::cout << "tcp\n";
        // ...
    }
};
```

```cpp
struct Session : public Tcp
{
    void initializeConfig() override
    {
        std::cout << "session\n";
        // ...
    }
};


int main()
{
    Tcp a;
    a.init();

    Session b;
    b.init();
}
```

Output of x86-64 clang (trunk) (Compiler #1)

A Wrap lines Select all

```
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 0
 tcp
 session
```

Multi-Level Inheritance – CRTP Attempt

```cpp
#include <type_traits>

// https://godbolt.org/z/s3ed4Yorv
template <class derived>
struct NetworkConnection
{
    void init()
    {
        (static_cast<derived*>(this))->initializeConfig();
        // ...
    };
};

template <class T = void>
struct Tcp : public NetworkConnection<Tcp<T>>
{
    void initializeConfig()
    {
        std::cout << "tcp\n";
    }
};
```

```cpp
struct Session : public Tcp<Session>
{
    void initializeConfig()
    {
        std::cout << "session\n";
    }
};

int main()
{
    Tcp a;
    a.init();

    Session b;
    b.init();
}
```

Output of x86-64 clang (trunk) (Compiler #1)

A Wrap lines Select all

```
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 0
 tcp
 tcp
```

Multi-Level Inheritance – Deducing This Attempt

```cpp
// https://godbolt.org/z/ccsoaf3ec
struct NetworkConnection
{
    void init(this auto&& self)
    {
        self.initializeConfig();
        // ...
    };
};

struct Tcp : public NetworkConnection
{
    void initializeConfig()
    {
        std::cout << "tcp\n";
        // ...
    }
};
```

```cpp
struct Session : public Tcp
{
    void initializeConfig()
    {
        std::cout << "session\n";
        // ...
    }
};

int main()
{
    Tcp a;
    a.init();

    Session b;
    b.init();
}
```

Output of x86-64 clang (trunk) (Compiler #1)

A Wrap lines Select all

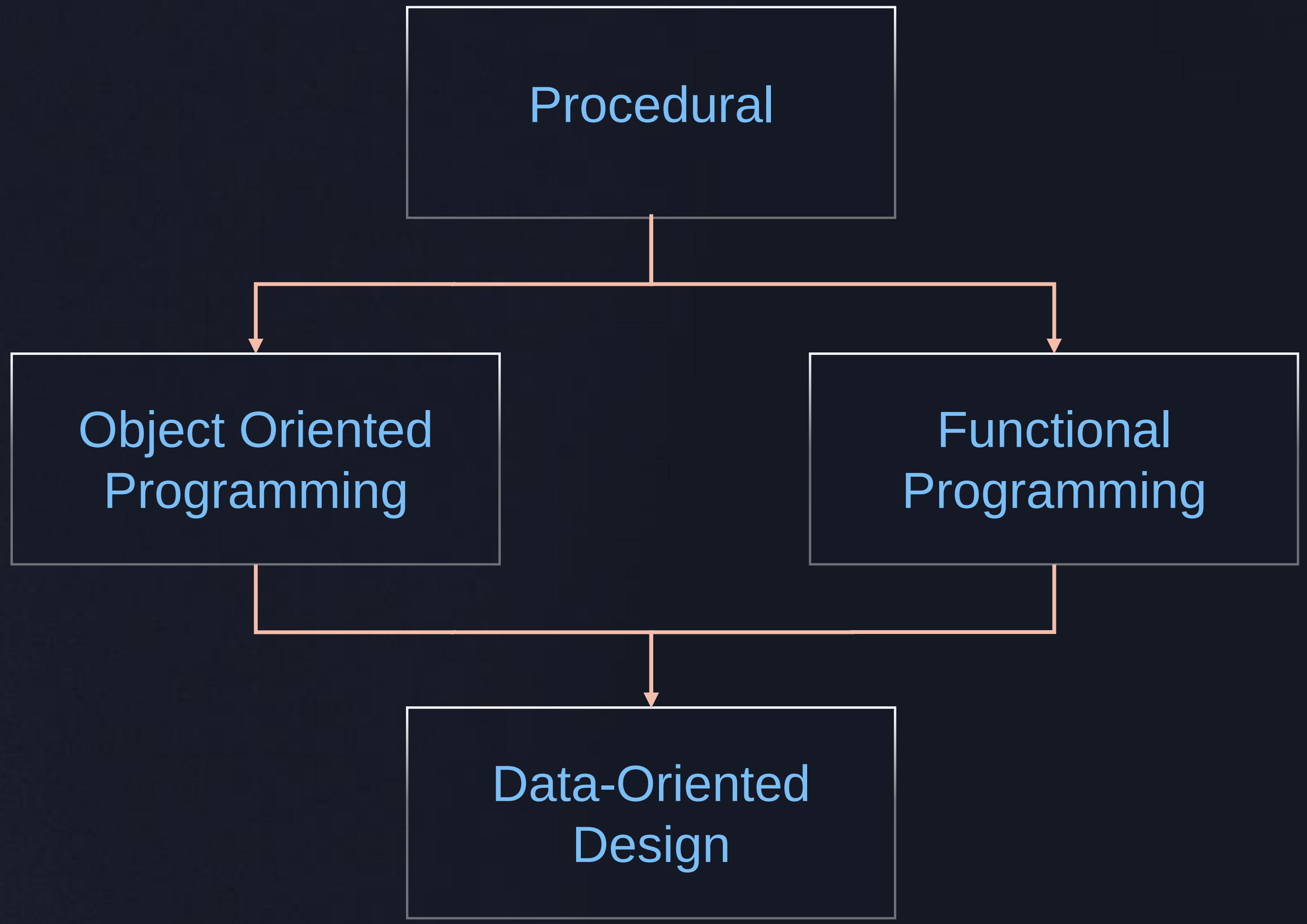
```
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 0
 tcp
 session
```

Comparison

Virtual Polymorphism	CRTP	Deducing This
• Runtime Polymorphism • Easy to Read and Trace	• Compile Time Polymorphism • Harder to Read • Hard to Trace • Multi-Level Polymorphism is Difficult	• Compile Time Polymorphism • C++23 Feature • Hard to trace

Design Methodology

Flow of Design Methods



C++ Design Aims – From The Design and Evolution of C++

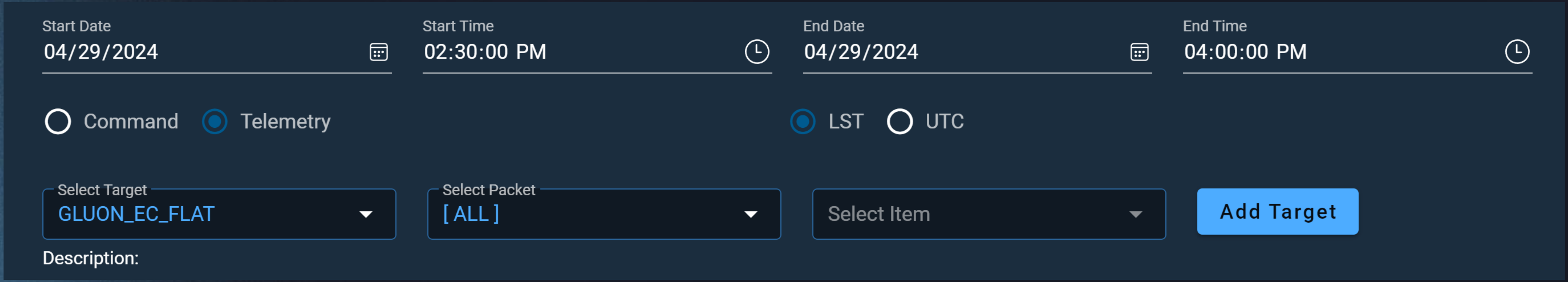
Aims:
C++ makes programming more enjoyable for serious programmers. C++ is a general-purpose programming language that – is a better C – supports data abstraction – supports object-oriented programming – supports generic programming

C++ Design Rules – From The Design and Evolution of C++

General rules:
C++'s evolution must be driven by real problems.
C++ is a language, not a complete system.
Don't get involved in a sterile quest for perfection.
C++ must be useful *now*.
Every feature must have a reasonably obvious implementation.
Always provide a transition path.
Provide comprehensive support for each supported style.
Don't try to force people.

Example Problem Space

- Extract all telemetry packets received during this talk
- Instantaneous and prolonged events



Procedural Programming

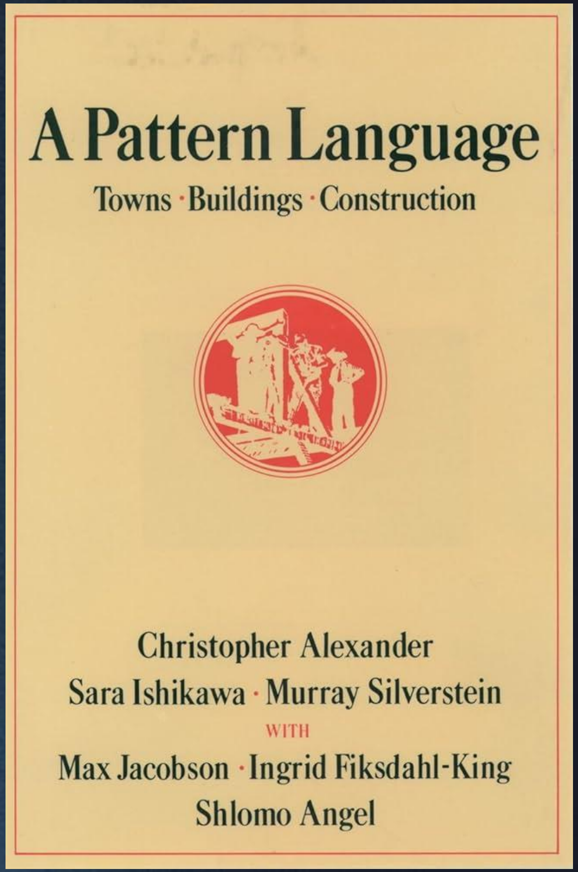
- Original Programming Style
- Think C not C++
- Do A then Do B then Do C

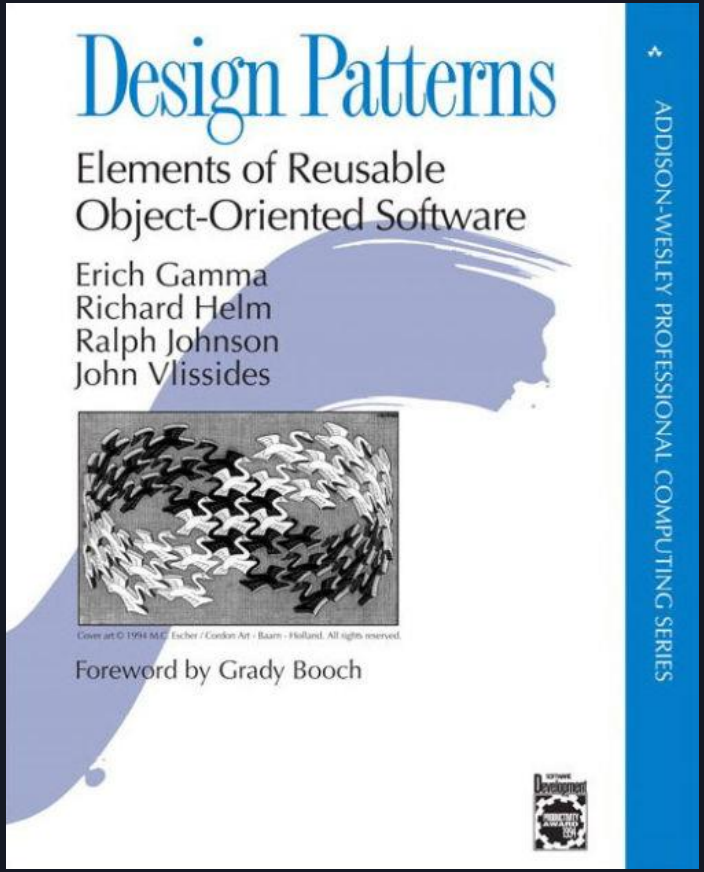
Procedural Programming Example

```cpp
struct Packet
{
    // ...
};

std::vector<Packet> telemetry = getTelemetry();

filterTelemetry(telemetry);
```

```cpp
std::vector<Packet> filterTelemetry(std::vector<Packet> telem)
{
    for (telemetry : telem)
    {
        if (telemetry.durationEvent)
        {
            // ...
        }
        else
        {
            // ...
        }
    }
}
```

Object Oriented Programing History





Note: SmallTalk (an OOP language) was invented before A Pattern Language was published

Common Object Oriented Patterns

- Creational Patterns
  - Factory
  - Builder
  - Prototype
  - Singleton
- Structural Patterns
  - Adapter
  - Bridge
  - Composite
  - Decorator
  - Façade
  - Flyweight
  - Proxy
- Behavioral Patterns
  - Chain of Responsibility
  - Command
  - InterpreterMediator
  - Memento
  - Observer
  - State
  - Strategy
  - Visitor

Object Oriented Programming

```
struct Packet
{
// ...
};

struct FilterTelemetryFacade
{
// ...
};

std::vector<Packet> telemetry = getTelemetry();

auto filter = FilterTelemetryFacade(telemetry);
std::vector<Packet> filter.getFilteredTelemetry();
```

Functional Programming History

- Lambda Calculus (1930s) by Alonzo Church
- Languages such as LISP and Haskell

Functional Patterns

- Functors
- Monads
- Applicatives

Functional Programming

```
struct Packet
{
    // ...
};

std::vector<Packet> telemetry = getTelemetry();

std::vector<Packet> filteredTelemetry;
```

```
auto filter = [](Packet telem)
{
    if (telemetry.durationEvent)
    {
        // ... Return true/false somewhere in here…
    }
    else
    {
        // ... Return true/false somewhere in here…
    }
}

std::copy_if(telemetry.begin(), telemetry.end(),
filteredTelemetry.begin(), filter);
```

Data-Oriented Design History

- Data-Oriented Design (Or Why You Might Be Shooting Yourself in The Foot With OOP) By Noel Llopis 2009
- Game Developer Perspective

Data-Oriented Design

```cpp
struct Packet
{
    // ...
};

std::vector<Packet> durationTelemetry = getDurationTelemetry();
std::vector<Packet> instantTelemetry = getInstantTelemetry();

filterDurationTelemetry(durationTelemetry);
filterInstantTelemetry(instantTelemetry);
```

```cpp
std::vector<Packet> filterDurationTelemetry(std::vector telem)
{
    for (telemetry : telem)
    {
        // ...
    }
}

std::vector<Packet> filterInstantTelemetry(std::vector telem)
{
    for (telemetry : telem)
    {
        // ...
    }
}
```

Pros and Cons

Procedural

- Simplistic
- Old
- Imperative
- Verbose

Object Oriented

- Colloquially Overused
- Heuristically Organize Data
- Pattern Based
- Prone to Anti-Patterns
- From Tony's talk this morning: "Objects are made of Velcro"

Functional

- Pure Functions
- Immutable Data
- Treat Functions as Data
- Describe the what not the how

Data-Oriented

- Hardware Oriented
- Performance Mindset
- Backwards to Traditional Thought

Other Potential Evaluations

- Union vs Variant
- Enum vs Enum Class
- Raw Pointers vs Reference vs Smart Pointers
- Raw Iterators vs Standard Algorithms
- C-Style Casts vs Fancy Casts (static, dynamic, reinterpret, const casts)
- Allocators vs PMR
- printf vs std::cout vs libfmt

Conclusion

- Newer Isn’t Always Better
- Consistently Reevaluate Alternatives
- Use Case Determines Usability


ATOMOS
GET TO YOUR PLACE IN SPACE
atomosspace.com