Embrace No-Paradigm Programming

Klaus Iglberger, London C++, April, 29th, 2020

klaus.iglberger@gmx.de

C++ Trainer since 2016

Author of the blaze C++ math library

(Co-)Organizer of the Munich C++ user group

Regular presenter at C++ conferences

Email: klaus.iglberger@gmx.de

Klaus Iglberger

C++ has changed tremendously in the past decade. …

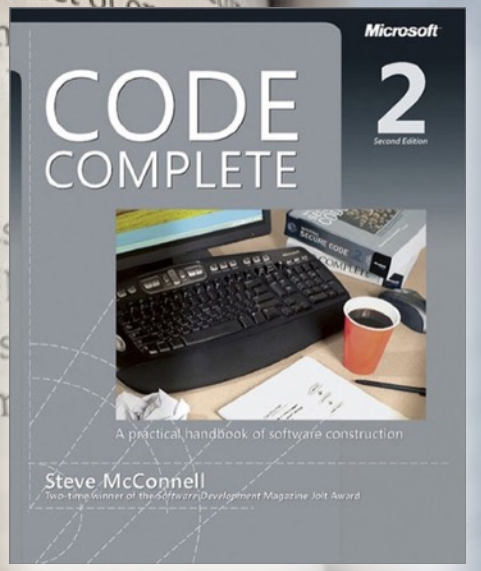
… Everyone is talking about cool new features, …

… but too few people are talking about software design. …

… There is still a strong tendency to design software based on the teachings from 20 years ago.

UNIX operating
data, structured control flow, machine
has also been called a "portable assembly language" because it
pointers and addresses, has some low-level constructs such as
is weakly typed.

C was developed in the 1970s at Bell Labs. It was originally de
the DEC PDP-11—whose operating system, C compiler, and U
grams were all written in C. In 1988, an ANSI standard was is
was revised in 1999. C was the de facto standard for microcom
programming in the 1980s and 1990s.



C++

C++, an object-oriented language founded on C, was developed at Bell Laboratories in the 1980s. In addition to being compatible with C, C++ provides classes, polymorphism, exception handling, templates, and it provides more robust type checking than C does. It also provides an extensive and powerful standard library.

C#

C# is a general-purpose, object-oriented language and programming environment developed by Microsoft with syntax similar to C, C++, and Java, and it provides extensive tools that aid development on Microsoft platforms.

Cobol

LearnCpp.com

Tutorials to help you master C++ and object-oriented programming

LearnCpp.com is a free website devoted to teaching you how to program in C++. Whether you've had any prior programming experience or not, the tutorials on this site will walk you through all the steps to write, compile, and debug your C++ programs, all with plenty of examples.

Becoming an expert won't happen overnight, but with a little patience, you'll get there. And LearnCpp.com will show you the way.

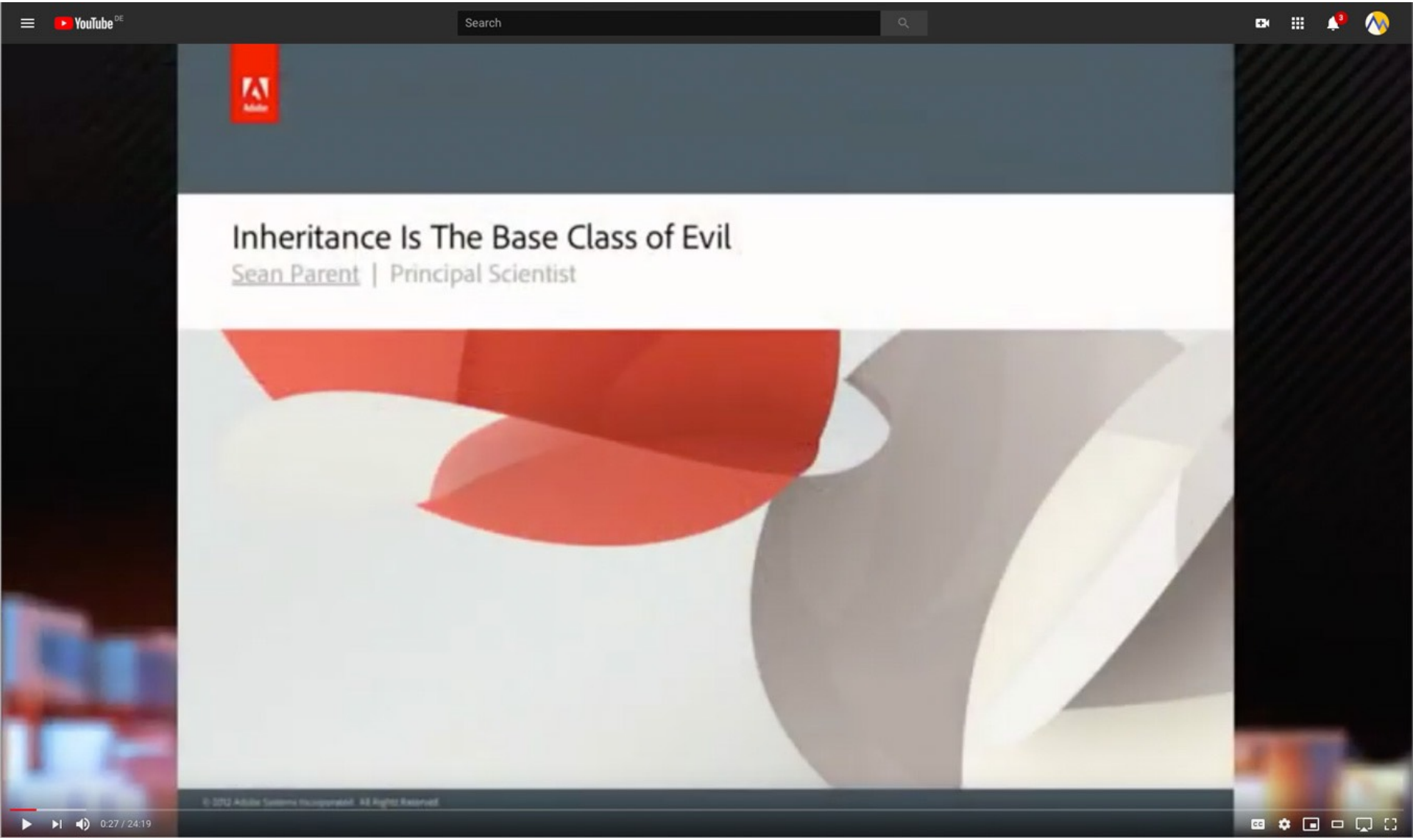
Having trouble remembering where you saw something? Not sure where to find something? Use our site index to find what you're looking for!

“… Overall you can get up and running pretty quickly with an object-oriented programming language like C++. …”

Brad Larson, Episode 234 of CppCast, February, 13th, 2020

More than 70% of my students would agree…

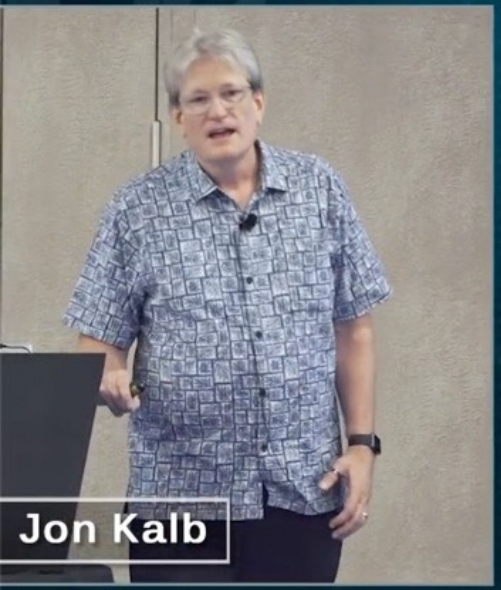
Has C++ truly changed?


YouTube DE
Search
Inheritance Is The Base Class of Evil
Sean Parent | Principal Scientist
0:27 / 24:19

YouTube DE

Search

Cppcon | 2019
The C++ Conference
cppcon.org



Jon Kalb

Back to Basics:
Object-Oriented
Programming

Video Sponsorship Provided By:
ansatz

THENEWSTACK Ebooks Podcasts Events Newsletter

Architecture Development Operations

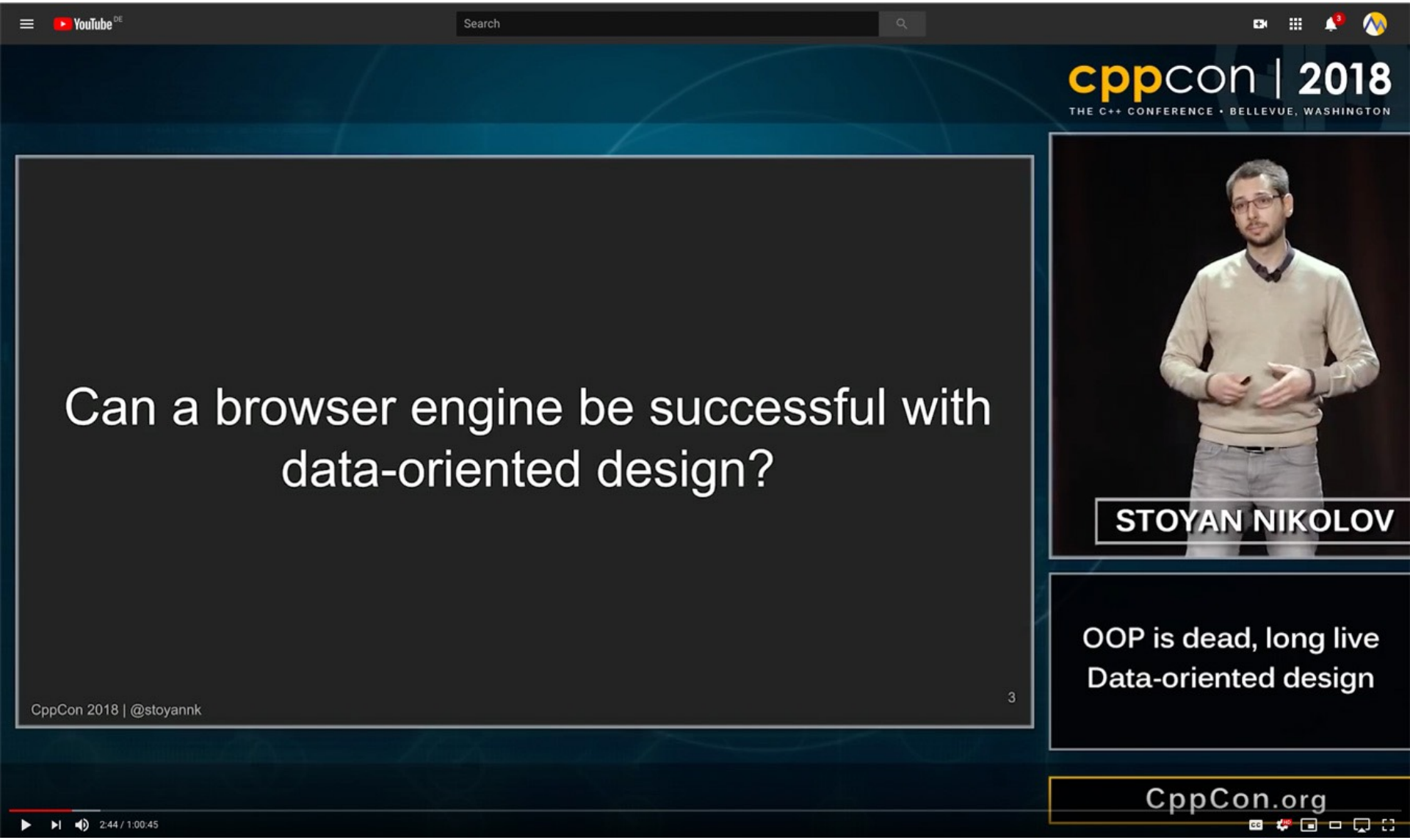
CULTURE / DEVELOPMENT

Why Are So Many Developers Hating on Object-Oriented Programming?

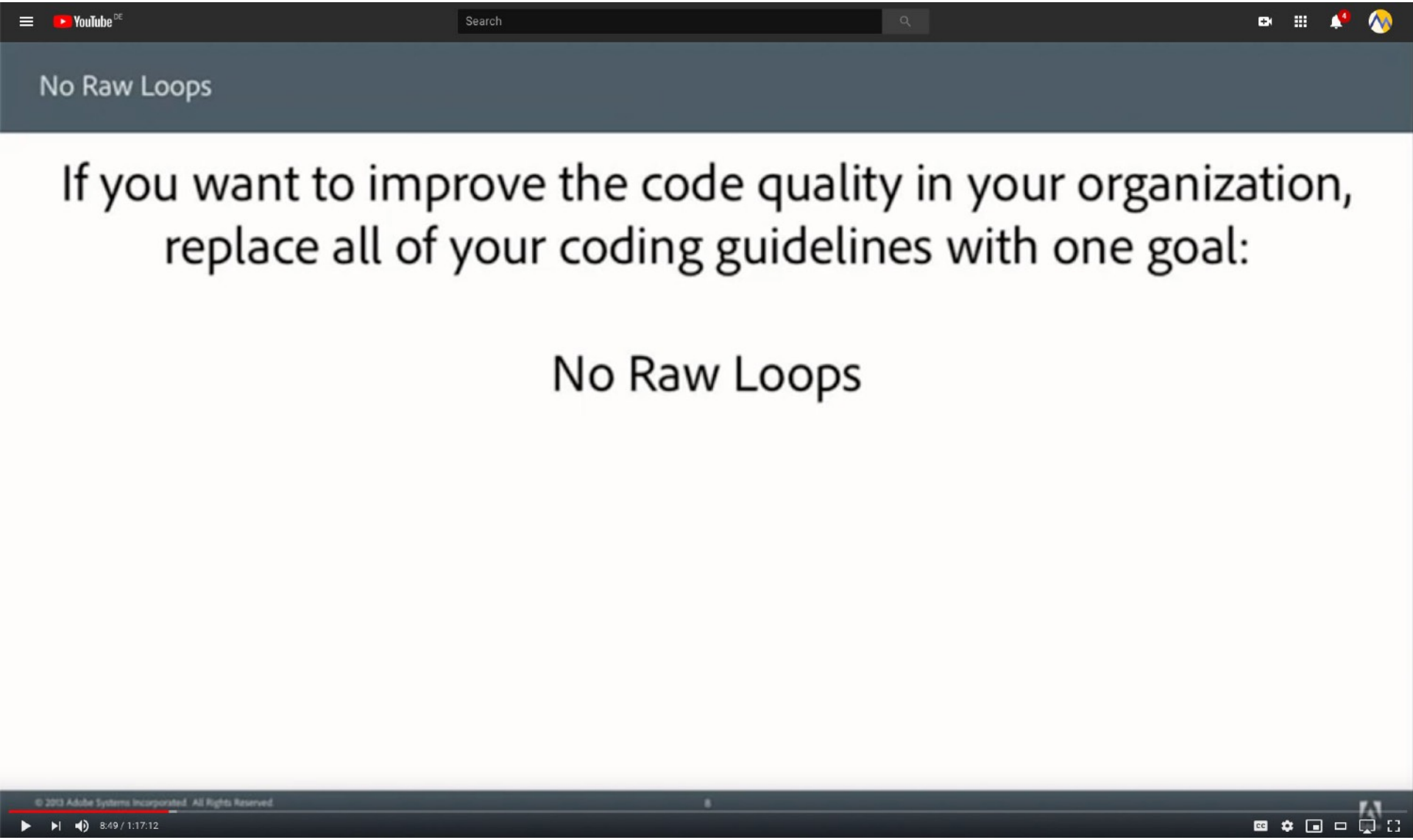
21 Aug 2019 12:00pm, by David Cassel

2

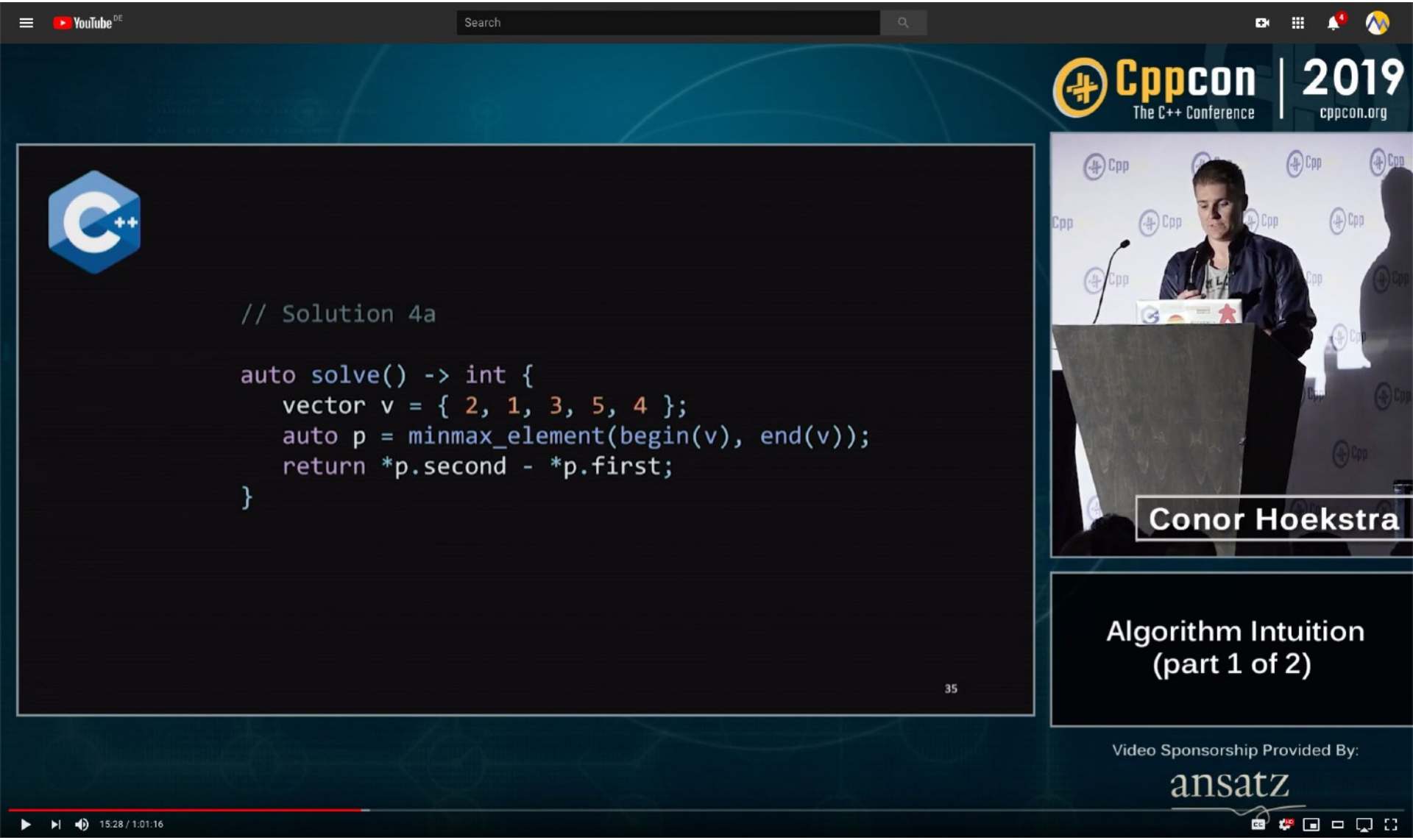
0:30 / 59:58


YouTube DE
Search
cppcon | 2018
THE C++ CONFERENCE • BELLEVUE, WASHINGTON
Can a browser engine be successful with data-oriented design?
CppCon 2018 | @stoyannk
3
STOYAN NIKOLOV
OOP is dead, long live Data-oriented design
CppCon.org
2:44 / 1:00:45


YouTube DE
c++ seasoning
8:36 / 1:17:12


YouTube DE
Search
No Raw Loops
If you want to improve the code quality in your organization, replace all of your coding guidelines with one goal:
No Raw Loops
© 2013 Adobe Systems Incorporated. All Rights Reserved.
8:49 / 1:17:12


YouTube DE
Search
cppcon | 2018
THE C++ CONFERENCE • BELLEVUE, WASHINGTON
Fluent C++
Value Queries
Property Queries
2-ranges Properties
Lonely Islands
Land of Queries
Algos on Sets
Raw Memory
Movers
Research
Structure Changers
Value Modifiers
Permutations
Sorted Ranges
The World of C++ STL Algorithms
JONATHAN BOCCARA
105 STL Algorithms in Less Than an Hour
CppCon.org
9:05 / 57:45


YouTube DE
Search
Cppcon
The C++ Conference
2019
cppcon.org
// Solution 4a
auto solve() -> int {
vector v = { 2, 1, 3, 5, 4 };
auto p = minmax_element(begin(v), end(v));
return *p.second - *p.first;
}
35
Conor Hoekstra
Algorithm Intuition
(part 1 of 2)
Video Sponsorship Provided By:
ansatz
15:28 / 1:01:16


YouTube DE
Search
Functional C++
@KevlinHenney
Play (k)
0:33 / 1:01:48


YouTube DE
Search
++
Profit
Functional C++
for Fun and Profit
Phil Nash
0:01 / 47:44

C++

From Wikipedia, the free encyclopedia

"CXX" redirects here. For the Roman numerals, see 120 (number).

C++ (/ˌsiːplʌsˈplʌs/) is a general-purpose programming language created by Bjarne Stroustrup as an extension of the C programming language, or "C with Classes". The language has expanded significantly over time, and modern C++ has object-oriented, generic, and functional features in addition to facilities for low-level memory manipulation. It is almost always implemented as a compiled language, and many vendors provide C++ compilers, including the Free Software Foundation, LLVM, Microsoft, Intel, Oracle, and IBM, so it is available on many platforms.[6]

C++ was designed with a bias toward system programming and embedded, resource-constrained software and large systems, with performance, efficiency, and flexibility of use as its design highlights.[7] C++ has also been found useful in many other contexts, with key strengths being software infrastructure and resource-constrained applications,[7] including desktop applications, servers (e.g. e-commerce, Web search, or SQL servers), and performance-critical applications (e.g. telephone switches or space probes).[8]

C++ is standardized by the International Organization for Standardization (ISO), with the latest standard version ratified and published by ISO in December 2017 as *ISO/IEC 14882:2017* (informally known as C++17).[9] The C++ programming language was initially standardized in 1998 as *ISO/IEC 14882:1998*, which was then amended by the C++03, C++11 and C++14 standards. The current C++17 standard supersedes these with new features and an enlarged standard library. Before the initial standardization in 1998, C++ was developed by Danish computer scientist Bjarne Stroustrup at Bell Labs since 1979 as an extension of the C language; he wanted an efficient and flexible language similar to C that also provided high-level features for program organization.[10] C++20 is the next planned standard, keeping with the current trend of a new version every three years.[11]

Contents [hide]

C++

Paradigms	Multi-paradigm: procedural, functional, object-oriented, generic
Designed by	Bjarne Stroustrup
Developer	ISO/IEC JTC1 (Joint Technical Committee 1) / SC22 (Subcommittee 22) / WG21 (Working Group 21)
First appeared	1985; 35 years ago
Stable release	C++17 (ISO/IEC 14882:2017) / 1 December 2017; 2 years ago
Preview release	C++20
Typing discipline	Static, nominative, partially inferred
Filename extensions	.C, .cc, .cpp, .cxx, .c++, .h, .hh, .hpp, .hxx, .h++
Website	isocpp.org

Major implementations

Wikipedia The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

In other projects
Wikimedia Commons
Wikibooks
Wikiquote
Wikiversity

Print/export
Download as PDF
Printable version

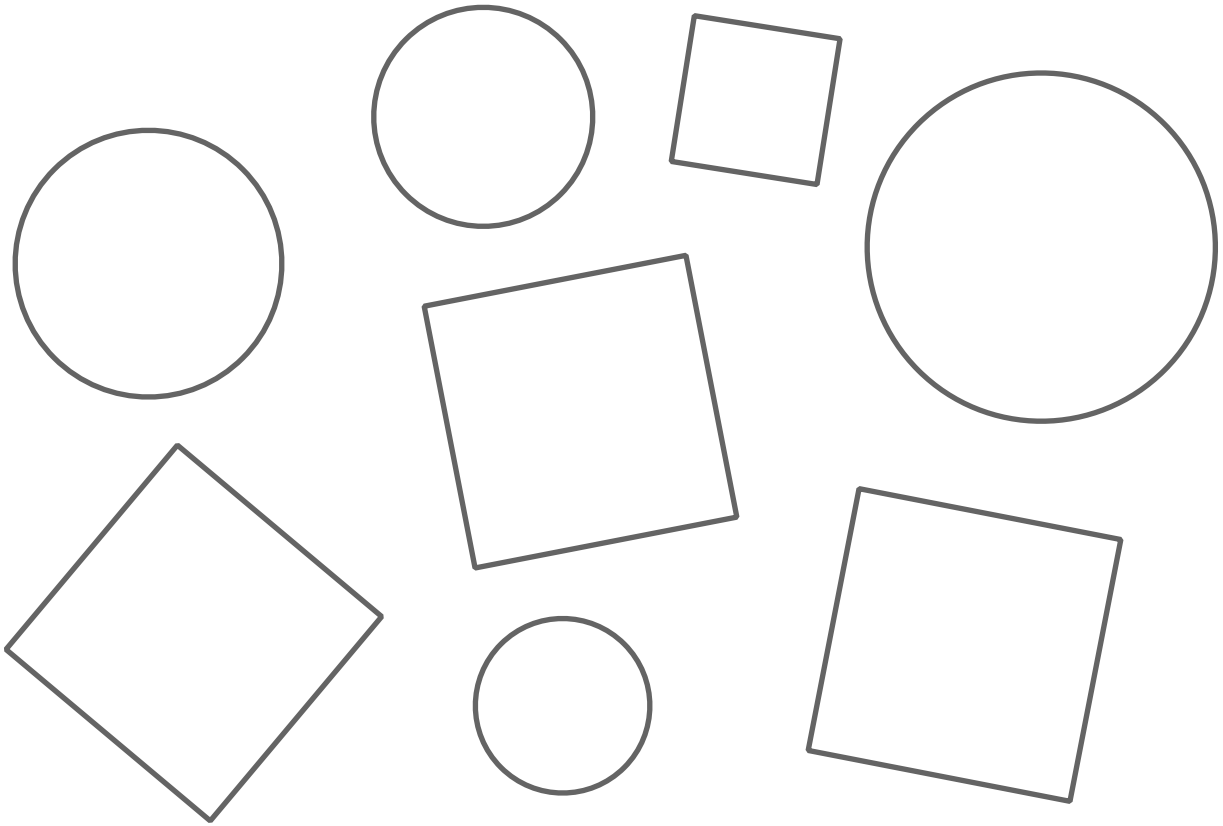
Languages
Dansk

Let’s talk about “Modern C++” design …

Disclaimer

- C++ is too complex to give a complete answer
- I consider only solutions with dynamic polymorphism
- I consider only some solutions
- I consider only the state of the art of C++
- There will be a lot of code …
- … but it's not about the details, it's about the design
- I deliberately choose a simple problem …

Our Toy Problem: Drawing Shapes



A Procedural Solution

```
enum ShapeType
{
   circle,
   square
};

class Shape
{
 public:
   explicit Shape( ShapeType t )
      : type{ t }
   {}

   virtual ~Shape() = default;
   ShapeType getType() const noexcept;

 private:
   ShapeType type;
};

class Circle : public Shape
{
 public:
   explicit Circle( double rad )
      : Shape{ circle }
      , radius{ rad }
      , // ... Remaining data members
   {}
```

A Procedural Solution

```
enum ShapeType
{
   circle,
   square
};

class Shape
{
 public:
   explicit Shape( ShapeType t )
      : type{ t }
   {}

   virtual ~Shape() = default;
   ShapeType getType() const noexcept;

 private:
   ShapeType type;
};

class Circle : public Shape
{
 public:
   explicit Circle( double rad )
      : Shape{ circle }
      , radius{ rad }
      , // ... Remaining data members
   {}
```

A Procedural Solution

```
enum ShapeType
{
   circle,
   square
};

class Shape
{
 public:
   explicit Shape( ShapeType t )
      : type{ t }
   {}

   virtual ~Shape() = default;
   ShapeType getType() const noexcept;

 private:
   ShapeType type;
};

class Circle : public Shape
{
 public:
   explicit Circle( double rad )
      : Shape{ circle }
      , radius{ rad }
      , // ... Remaining data members
   {}
```

A Procedural Solution

```
 private:
   ShapeType type;
};

class Circle : public Shape
{
 public:
   explicit Circle( double rad )
      : Shape{ circle }
      , radius{ rad }
      , // ... Remaining data members
   {}

   virtual ~Circle() = default;
   double getRadius() const noexcept;
   // ... getCenter(), getRotation(), ...

 private:
   double radius;
   // ... Remaining data members
};

void translate( Circle&, Vector3D const& );
void rotate( Circle&, Quaternion const& );
void draw( Circle const& );

class Square : public Shape
{
 public:
   explicit Square( double s )
      : Shape{ square }
      side{ s }
```

A Procedural Solution

```
void translate( Circle&, Vector3D const& );
void rotate( Circle&, Quaternion const& );
void draw( Circle const& );

class Square : public Shape
{
 public:
   explicit Square( double s )
      : Shape{ square }
      , side{ s }
      , // ... Remaining data members
   {}

   virtual ~Square() = default;
   double getSide() const noexcept;
   // ... getCenter(), getRotation(), ...

 private:
   double side;
   // ... Remaining data members
};

void translate( Square&, Vector3D const& );
void rotate( Square&, Quaternion const& );
void draw( Square const& );

void draw( std::vector<std::unique_ptr<Shape>> const& shapes )
{
   for( auto const& s : shapes )
   {
      switch ( s->type )
      {
```

A Procedural Solution

```
   double side;
   // ... Remaining data members
};

void translate( Square&, Vector3D const& );
void rotate( Square&, Quaternion const& );
void draw( Square const& );

void draw( std::vector<std::unique_ptr<Shape>> const& shapes )
{
   for( auto const& s : shapes )
   {
      switch ( s->type )
      {
         case circle:
            draw( *static_cast<Circle const*>( s.get() ) );
            break;
         case square:
            draw( *static_cast<Square const*>( s.get() ) );
            break;
      }
   }
}

int main()
{
   using Shapes = std::vector<std::unique_ptr<Shape>>;

   // Creating some shapes
   Shapes shapes;
   shapes.push_back( std::make_unique<Circle>( 2.0 ) );
   shapes.push_back( std::make_unique<Square>( 1.5 ) );
```

A Procedural Solution

```
                draw( *static_cast<Circle const*>( s.get() ) );
                break;
            case square:
                draw( *static_cast<Square const*>( s.get() ) );
                break;
        }
    }
}

int main()
{
    using Shapes = std::vector<std::unique_ptr<Shape>>;

    // Creating some shapes
    Shapes shapes;
    shapes.push_back( std::make_unique<Circle>( 2.0 ) );
    shapes.push_back( std::make_unique<Square>( 1.5 ) );
    shapes.push_back( std::make_unique<Circle>( 4.2 ) );

    // Drawing all shapes
    draw( shapes );
}
```

Design Evaluation

Enum					

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)				
Enum					

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)			
Enum					

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)		
Enum					

1 = very bad, …, 9 = very good

The SOLID Principles

Single Responsibility Principle

Open-Closed Principle

Liskov Substitution Principle

Interface Segregation Principle

Dependency Inversion Principle

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)		
Enum					

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	
Enum					

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum					

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1				

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7			

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5		

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	

1 = very bad, …, 9 = very good

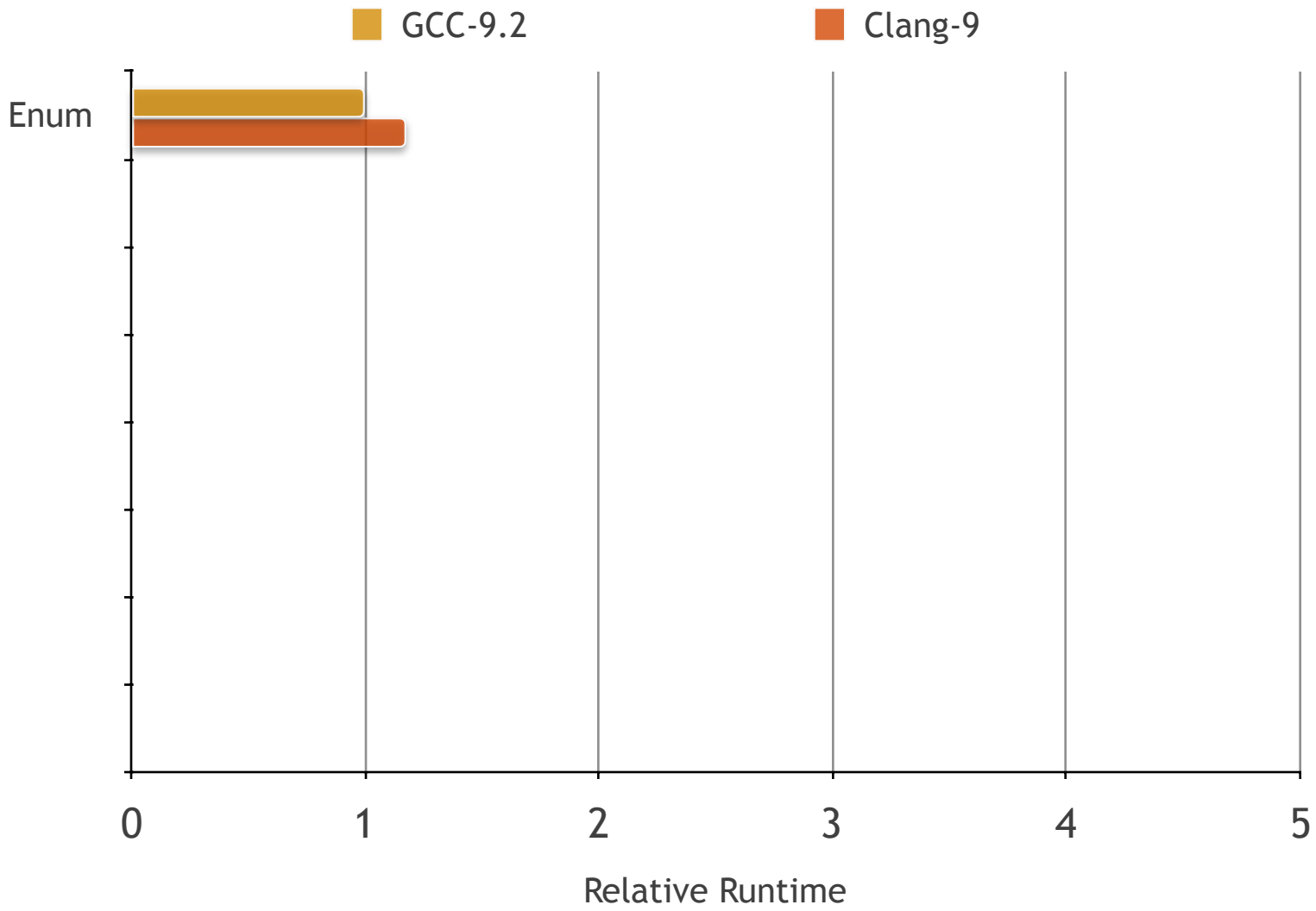
Performance Results

Benchmarks!

MacBook Pro 13-inch, MacOS 10.14.6 (Mojave), Intel Core i7, 2,7Ghz, 16 GByte, Compilation flags: -std=c++17 -O3 -DNDEBUG

100 random shapes, 2.5M updates each, normalized to the enum solution

Performance Results



MacBook Pro 13-inch, MacOS 10.14.6 (Mojave), Intel Core i7, 2,7Ghz, 16 GByte, Compilation flags: -std=c++17 -O3 -DNDEBUG

100 random shapes, 2.5M updates each, normalized to the enum solution

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9

1 = very bad, …, 9 = very good

An Object-Oriented Solution

```
class Shape
{
 public:
   Shape() = default;
   virtual ~Shape() = default;

   virtual void translate( Vector3D const& ) = 0;
   virtual void rotate( Quaternion const& ) = 0;
   virtual void draw() const = 0;
};

class Circle : public Shape
{
 public:
   explicit Circle( double rad )
      : radius{ rad }
      , // ... Remaining data members
   {}

   virtual ~Circle() = default;

   double getRadius() const noexcept;
   // ... getCenter(), getRotation(), ...

   void translate( Vector3D const& ) override;
   void rotate( Quaternion const& ) override;
   void draw() const override;

 private:
   double radius;
```

An Object-Oriented Solution

```
class Shape
{
 public:
   Shape() = default;
   virtual ~Shape() = default;

   virtual void translate( Vector3D const& ) = 0;
   virtual void rotate( Quaternion const& ) = 0;
   virtual void draw() const = 0;
};

class Circle : public Shape
{
 public:
   explicit Circle( double rad )
      : radius{ rad }
      , // ... Remaining data members
   {}

   virtual ~Circle() = default;

   double getRadius() const noexcept;
   // ... getCenter(), getRotation(), ...

   void translate( Vector3D const& ) override;
   void rotate( Quaternion const& ) override;
   void draw() const override;

 private:
   double radius;
```

An Object-Oriented Solution

```
   virtual void translate( Vector3D const& ) = 0;
   virtual void rotate( Quaternion const& ) = 0;
   virtual void draw() const = 0;
};

class Circle : public Shape
{
 public:
   explicit Circle( double rad )
      : radius{ rad }
      , // ... Remaining data members
   {}

   virtual ~Circle() = default;

   double getRadius() const noexcept;
   // ... getCenter(), getRotation(), ...

   void translate( Vector3D const& ) override;
   void rotate( Quaternion const& ) override;
   void draw() const override;

 private:
   double radius;
   // ... Remaining data members
};

class Square : public Shape
{
 public:
   explicit Square( double s )
      : side{ s }
```

An Object-Oriented Solution

```
 private:
   double radius;
   // ... Remaining data members
};

class Square : public Shape
{
 public:
   explicit Square( double s )
      : side{ s }
      , // ... Remaining data members
   {}

   virtual ~Square() = default;

   double getSide() const noexcept;
   // ... getCenter(), getRotation(), ...

   void translate( Vector3D const& ) override;
   void rotate( Quaternion const& ) override;
   void draw() const override;

 private:
   double side;
   // ... Remaining data members
};

void draw( std::vector<std::unique_ptr<Shape>> const& shapes )
{
   for( auto const& s : shapes )
   {
      s->draw()
```

An Object-Oriented Solution

```
   // ... getCenter(), getRotation(), ...

   void translate( Vector3D const& ) override;
   void rotate( Quaternion const& ) override;
   void draw() const override;

 private:
   double side;
   // ... Remaining data members
};

void draw( std::vector<std::unique_ptr<Shape>> const& shapes )
{
   for( auto const& s : shapes )
   {
      s->draw()
   }
}

int main()
{
   using Shapes = std::vector<std::unique_ptr<Shape>>;

   // Creating some shapes
   Shapes shapes;
   shapes.push_back( std::make_unique<Circle>( 2.0 ) );
   shapes.push_back( std::make_unique<Square>( 1.5 ) );
   shapes.push_back( std::make_unique<Circle>( 4.2 ) );

   // Drawing all shapes
   draw( shapes );
}
```

An Object-Oriented Solution

```
void draw( std::vector<std::unique_ptr<Shape>> const& shapes )
{
   for( auto const& s : shapes )
   {
      s->draw()
   }
}

int main()
{
   using Shapes = std::vector<std::unique_ptr<Shape>>;

   // Creating some shapes
   Shapes shapes;
   shapes.push_back( std::make_unique<Circle>( 2.0 ) );
   shapes.push_back( std::make_unique<Square>( 1.5 ) );
   shapes.push_back( std::make_unique<Circle>( 4.2 ) );

   // Drawing all shapes
   draw( shapes );
}
```

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO					

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8				

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2			

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2		

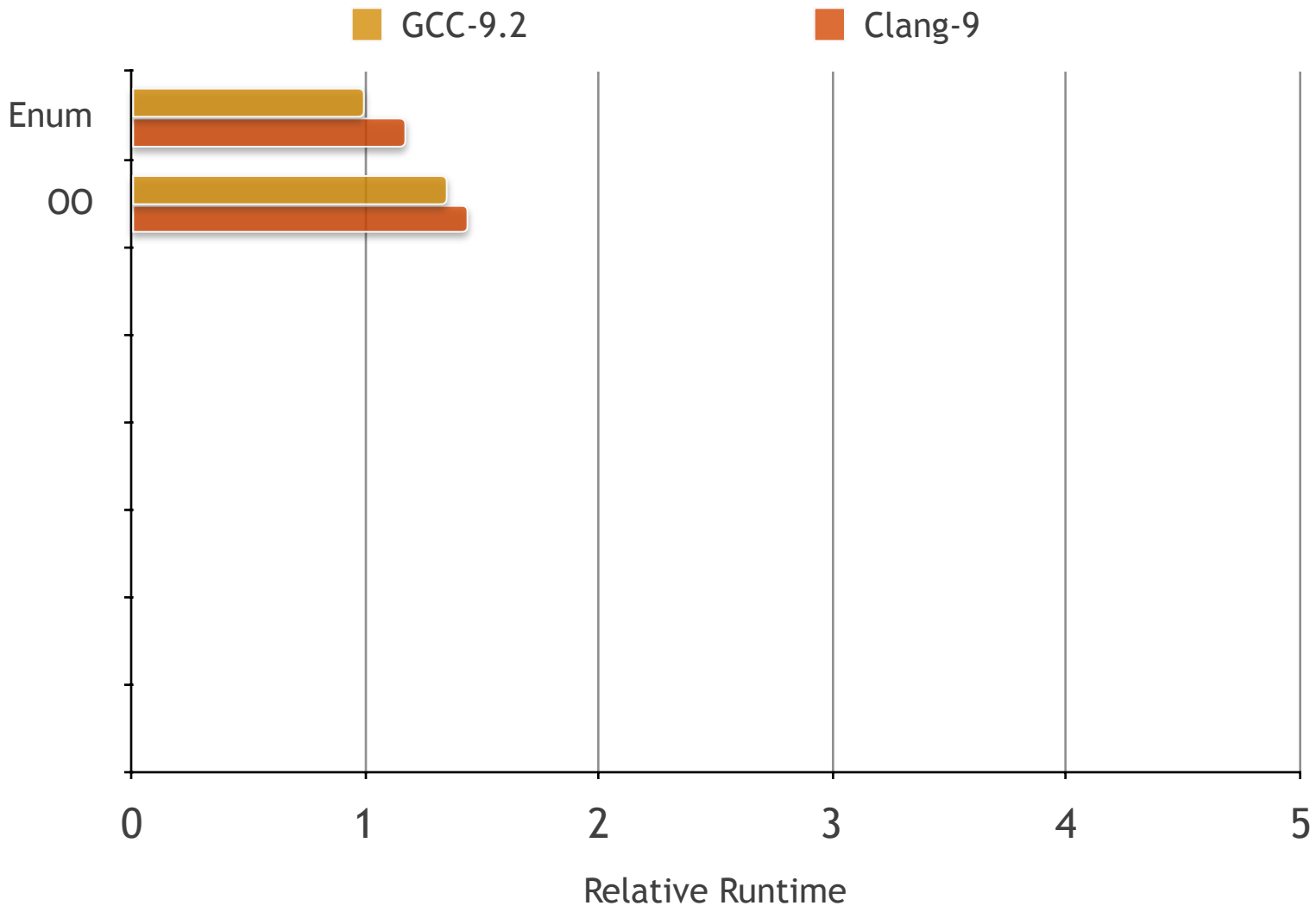
1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	

1 = very bad, …, 9 = very good

Performance Results



MacBook Pro 13-inch, MacOS 10.14.6 (Mojave), Intel Core i7, 2,7Ghz, 16 GByte, Compilation flags: -std=c++17 -O3 -DNDEBUG

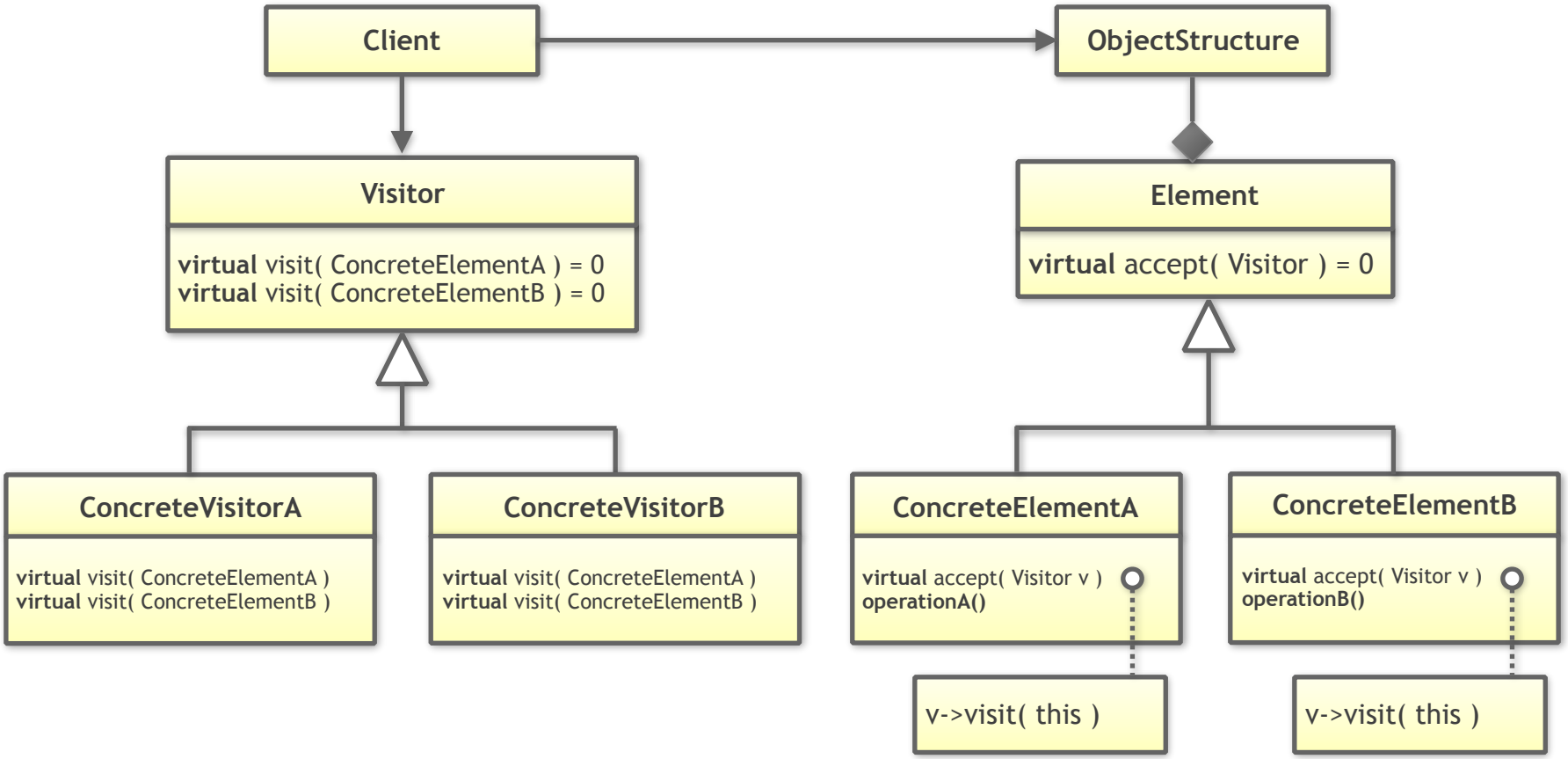
100 random shapes, 2.5M updates each, normalized to the enum solution

Design Evaluation

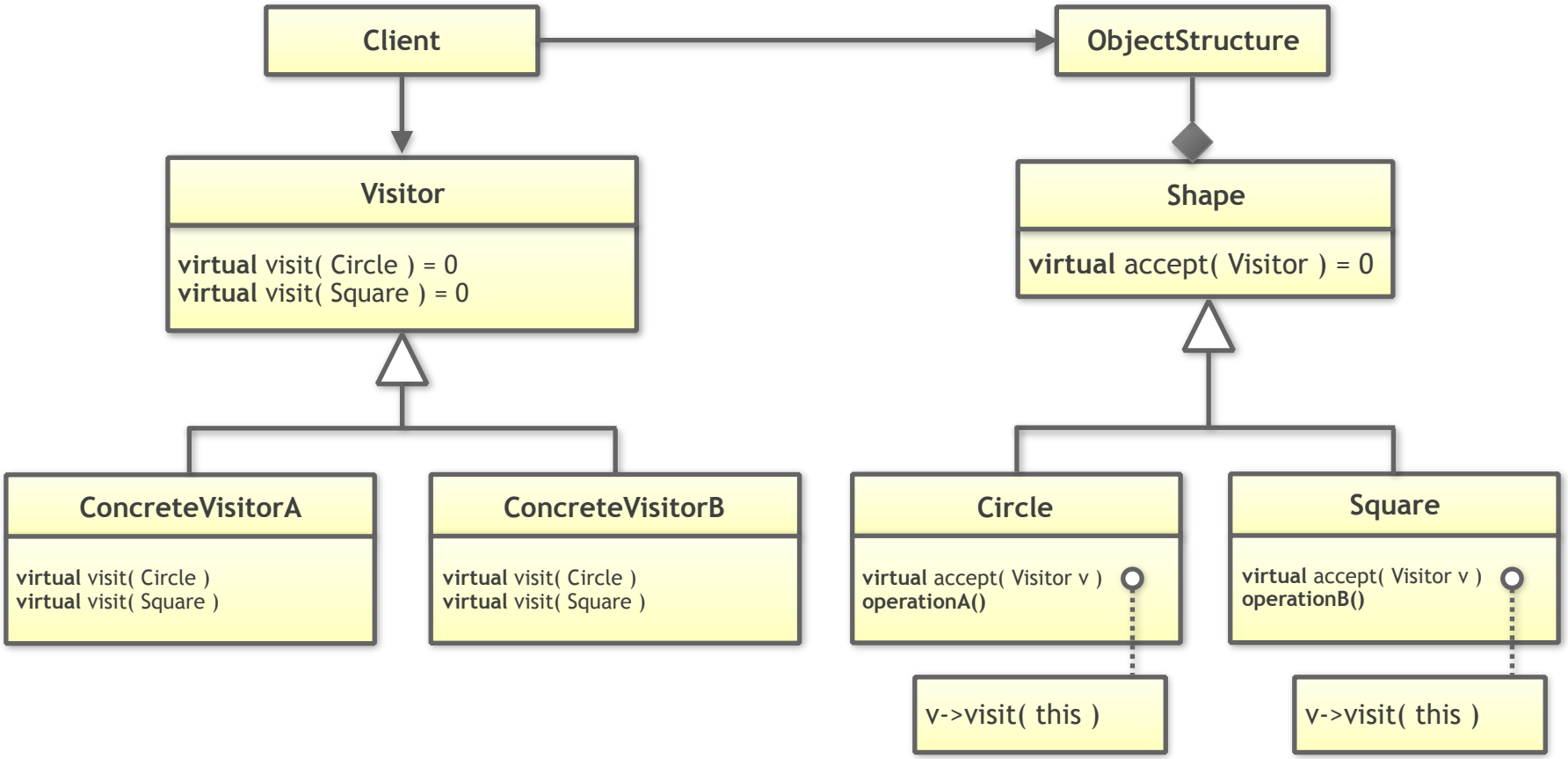
	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6

1 = very bad, …, 9 = very good

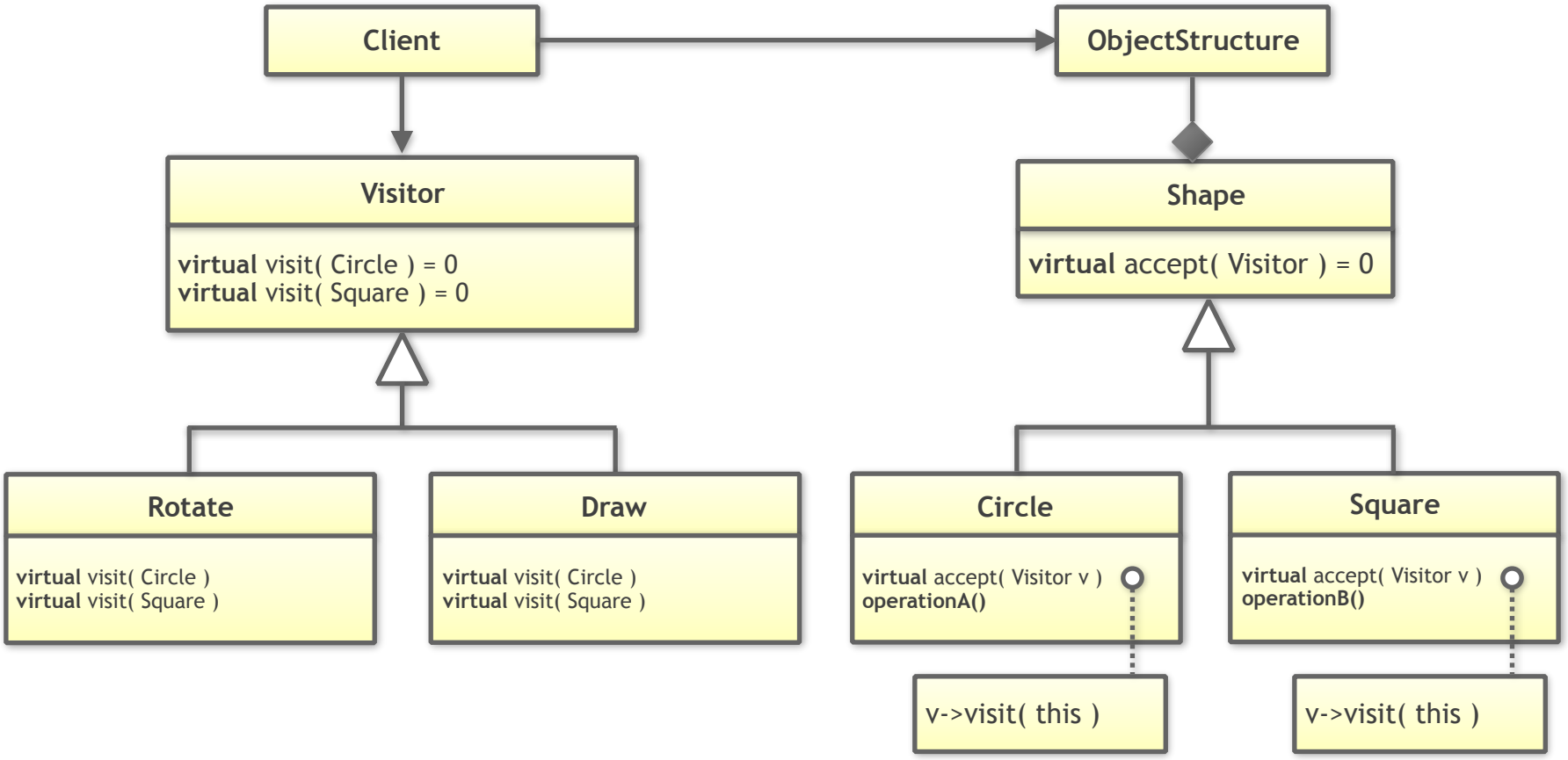
The Visitor Design Pattern



The Visitor Design Pattern



The Visitor Design Pattern



A Visitor-Based Solution

```
class Circle;
class Square;

class Visitor
{
 public:
   virtual ~Visitor() = default;

   virtual void visit( Circle const& ) const = 0;
   virtual void visit( Square const& ) const = 0;
};

class Shape
{
 public:
   Shape() = default;
   virtual ~Shape() = default;

   virtual void accept( Visitor const& ) = 0;
};


class Circle : public Shape
{
 public:
   explicit Circle( double rad )
      : radius{ rad }
      , // ... Remaining data members
   {}
```

A Visitor-Based Solution

```
class Circle;
class Square;

class Visitor
{
 public:
   virtual ~Visitor() = default;

   virtual void visit( Circle const& ) const = 0;
   virtual void visit( Square const& ) const = 0;
};

class Shape
{
 public:
   Shape() = default;
   virtual ~Shape() = default;

   virtual void accept( Visitor const& ) = 0;
};


class Circle : public Shape
{
 public:
   explicit Circle( double rad )
      : radius{ rad }
      , // ... Remaining data members
   {}
```

A Visitor-Based Solution

```
class Circle;
class Square;

class Visitor
{
 public:
   virtual ~Visitor() = default;

   virtual void visit( Circle const& ) const = 0;
   virtual void visit( Square const& ) const = 0;
};

class Shape
{
 public:
   Shape() = default;
   virtual ~Shape() = default;

   virtual void accept( Visitor const& ) = 0;
};


class Circle : public Shape
{
 public:
   explicit Circle( double rad )
      : radius{ rad }
      , // ... Remaining data members
   {}
```

A Visitor-Based Solution

```
   virtual ~Shape() = default;

   virtual void accept( Visitor const& ) = 0;
};


class Circle : public Shape
{
 public:
   explicit Circle( double rad )
      : radius{ rad }
      , // ... Remaining data members
   {}

   virtual ~Circle() = default;

   double getRadius() const noexcept;
   // ... getCenter(), getRotation(), ...

   void accept( Visitor const& ) override;

 private:
   double radius;
   // ... Remaining data members
};


class Square : public Shape
{
 public:
   explicit Square( double s )
      : side{ s }
      , // ... Remaining data members
   {}
```

A Visitor-Based Solution

```
 private:
   double radius;
   // ... Remaining data members
};


class Square : public Shape
{
 public:
   explicit Square( double s )
      : side{ s }
      , // ... Remaining data members
   {}

   virtual ~Square() = default;

   double getSide() const noexcept;
   // ... getCenter(), getRotation(), ...

   void accept( Visitor const& ) override;

 private:
   double side;
   // ... Remaining data members
};


class Draw : public Visitor
{
 public:
   void visit( Circle const& ) const override;
   void visit( Square const& ) const override;
};
```

A Visitor-Based Solution

```
 private:
   double side;
   // ... Remaining data members
};


class Draw : public Visitor
{
 public:
   void visit( Circle const& ) const override;
   void visit( Square const& ) const override;
};


void draw( std::vector<std::unique_ptr<Shape>> const& shapes )
{
   for( auto const& s : shapes )
   {
      s->accept( Draw{} )
   }
}


int main()
{
   using Shapes = std::vector<std::unique_ptr<Shape>>;

   // Creating some shapes
   Shapes shapes;
   shapes.push_back( std::make_unique<Circle>( 2.0 ) );
   shapes.push_back( std::make_unique<Square>( 1.5 ) );
```

A Visitor-Based Solution

```
void draw( std::vector<std::unique_ptr<Shape>> const& shapes )
{
   for( auto const& s : shapes )
   {
      s->accept( Draw{} )
   }
}


int main()
{
   using Shapes = std::vector<std::unique_ptr<Shape>>;

   // Creating some shapes
   Shapes shapes;
   shapes.push_back( std::make_unique<Circle>( 2.0 ) );
   shapes.push_back( std::make_unique<Square>( 1.5 ) );
   shapes.push_back( std::make_unique<Circle>( 4.2 ) );

   // Drawing all shapes
   draw( shapes );
}
```

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor					

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2				

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8			

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8		

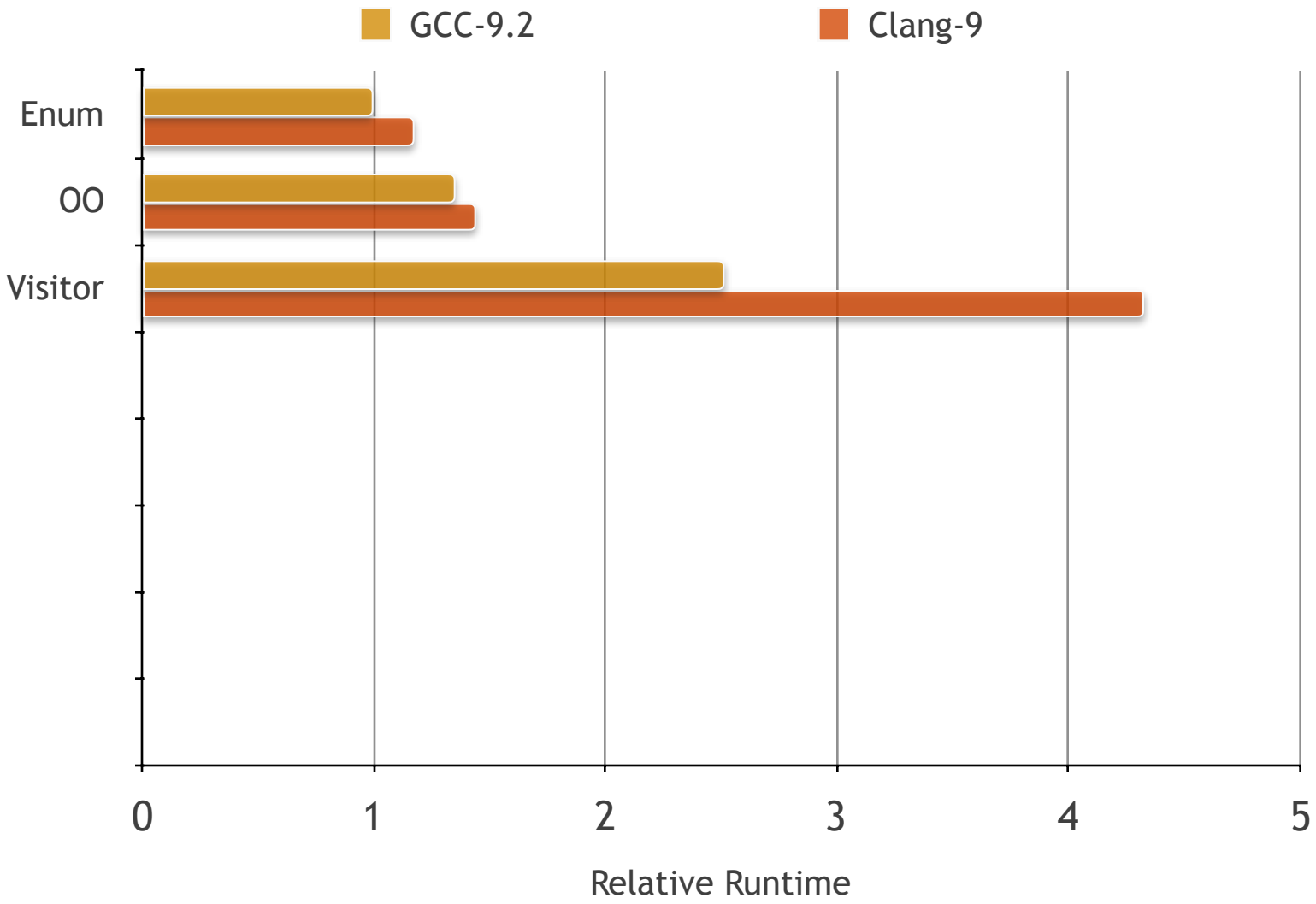
1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	

1 = very bad, …, 9 = very good

Performance Results



MacBook Pro 13-inch, MacOS 10.14.6 (Mojave), Intel Core i7, 2,7Ghz, 16 GByte, Compilation flags: -std=c++17 -O3 -DNDEBUG

100 random shapes, 2.5M updates each, normalized to the enum solution

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	1

1 = very bad, …, 9 = very good

A “Modern C++” Solution

```
class Circle
{
 public:
   explicit Circle( double rad )
      : radius{ rad }
      , // ... Remaining data members
   {}

   double getRadius() const noexcept;
   // ... getCenter(), getRotation(), ...

 private:
   double radius;
   // ... Remaining data members
};


class Square
{
 public:
   explicit Square( double s )
      : side{ s }
      , // ... Remaining data members
   {}

   double getSide() const noexcept;
   // ... getCenter(), getRotation(), ...

 private:
   double side;
```

A “Modern C++” Solution

```
class Circle
{
 public:
   explicit Circle( double rad )
      : radius{ rad }
      , // ... Remaining data members
   {}

   double getRadius() const noexcept;
   // ... getCenter(), getRotation(), ...

 private:
   double radius;
   // ... Remaining data members
};


class Square
{
 public:
   explicit Square( double s )
      : side{ s }
      , // ... Remaining data members
   {}

   double getSide() const noexcept;
   // ... getCenter(), getRotation(), ...

 private:
   double side;
```

A “Modern C++” Solution

```
    // ... getCenter(), getRotation(), ...

 private:
    double radius;
    // ... Remaining data members
};


class Square
{
 public:
    explicit Square( double s )
       : side{ s }
       , // ... Remaining data members
    {}

    double getSide() const noexcept;
    // ... getCenter(), getRotation(), ...

 private:
    double side;
    // ... Remaining data members
};

class Draw
{
 public:
    void operator()( Circle const& ) const override;
    void operator()( Square const& ) const override;
};


using Shape = std::variant<Circle,Square>;
```

A “Modern C++” Solution

```
   // ... getCenter(), getRotation(), ...

 private:
   double side;
   // ... Remaining data members
};

class Draw
{
 public:
   void operator()( Circle const& ) const override;
   void operator()( Square const& ) const override;
};


using Shape = std::variant<Circle,Square>;

void draw( std::vector<Shape> const& shapes )
{
   for( auto const& s : shapes )
   {
      std::visit( Draw{}, s );
   }
}


int main()
{
   using Shapes = std::vector<Shape>;

   // Creating some shapes
   Shapes shapes;
   shapes.push_back( Circle{ 2.0 } );
```

A “Modern C++” Solution

```
void draw( std::vector<Shape> const& shapes )
{
   for( auto const& s : shapes )
   {
      std::visit( Draw{}, s );
   }
}


int main()
{
   using Shapes = std::vector<Shape>;

   // Creating some shapes
   Shapes shapes;
   shapes.push_back( Circle{ 2.0 } );
   shapes.push_back( Square{ 1.5 } );
   shapes.push_back( Circle{ 4.2 } );

   // Drawing all shapes
   draw( shapes );
}
```

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	1
std::variant					

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	1
std::variant	3				

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	1
std::variant	3	9			

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	1
std::variant	3	9	9		

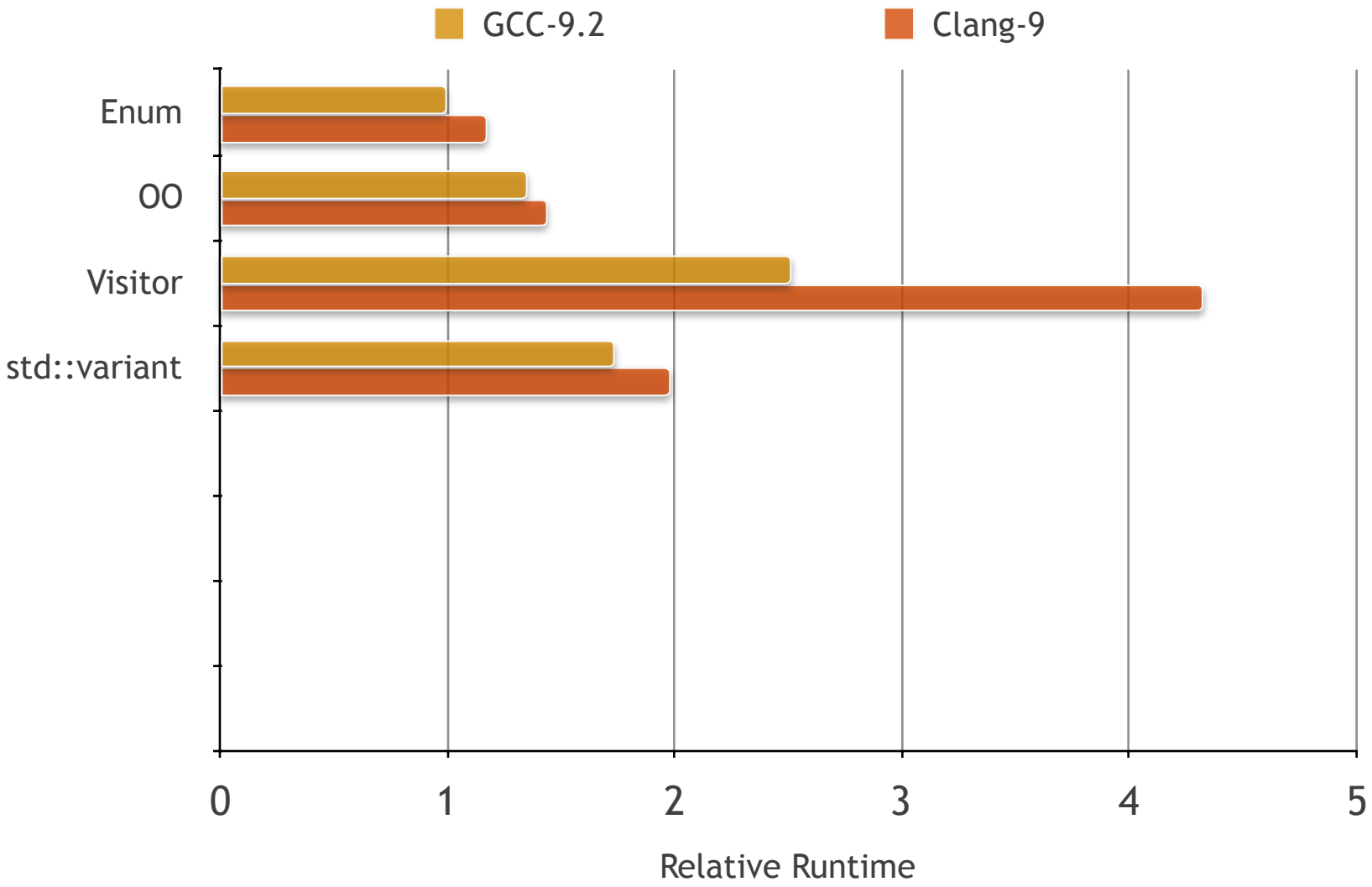
1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	1
std::variant	3	9	9	9	

1 = very bad, …, 9 = very good

Performance Results



MacBook Pro 13-inch, MacOS 10.14.6 (Mojave), Intel Core i7, 2,7Ghz, 16 GByte, Compilation flags: -std=c++17 -O3 -DNDEBUG

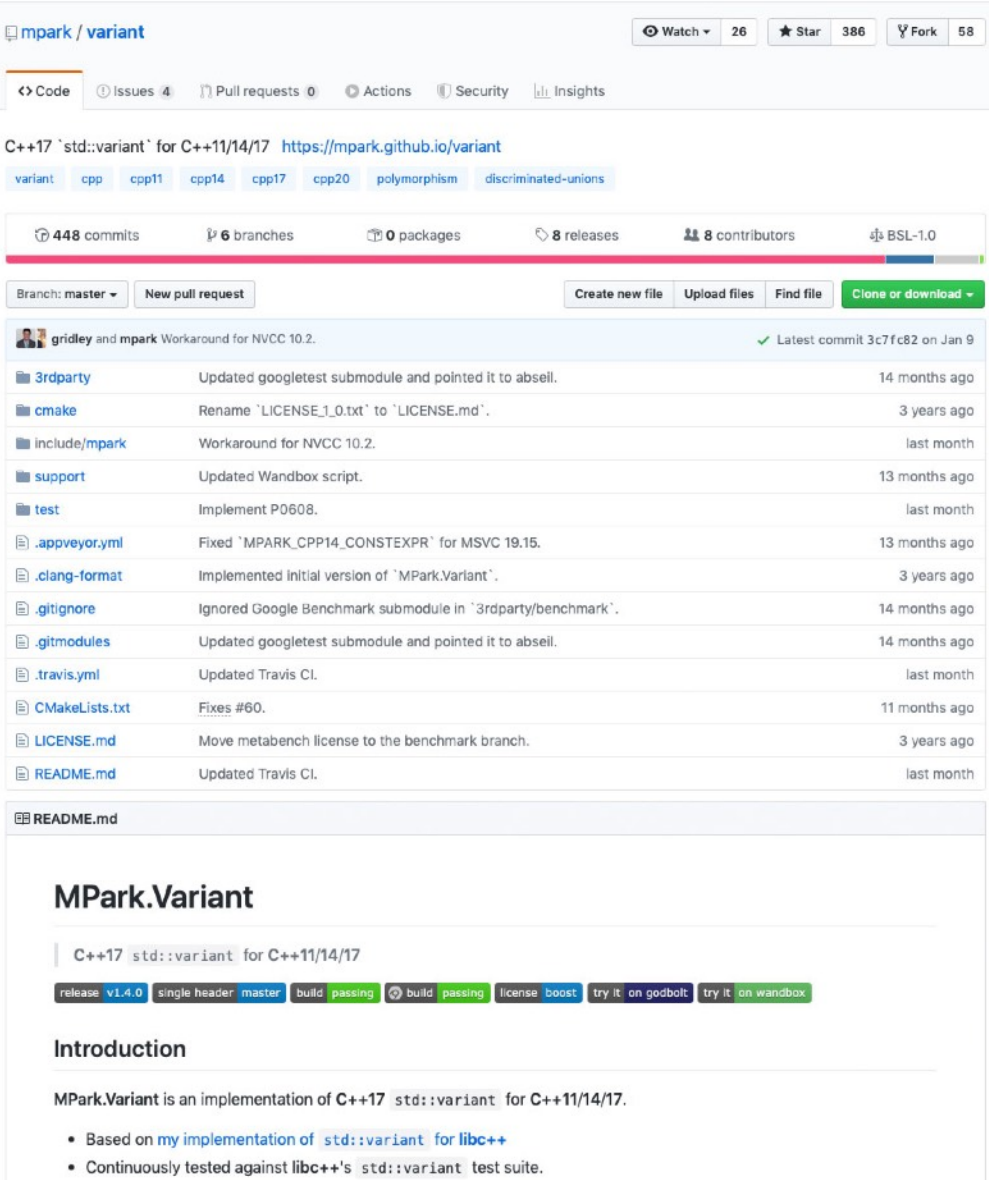
100 random shapes, 2.5M updates each, normalized to the enum solution

Performance Analysis

```
template< typename T, typename V >
constexpr auto make_func() {
   return + []( const char* b, V v ) {
      const auto& x = *reinterpret_cast<const T*>(b);
      v(x);
   };
}

template< typename… Ts, typename V >
void foo( std::size_t i, const char* b, V v ) {
   static constexpr std::array<void(*)(const char*, V v ),
      sizeof…(Ts)> table = { make_func<Ts,V>()... };
   table[i](b,v);
}
```

mpark::variant



mpark / variant

Watch 26 | Star 386 | Fork 58

Code | Issues 4 | Pull requests 0 | Actions | Security | Insights

C++17 `std::variant` for C++11/14/17 https://mpark.github.io/variant

variant cpp cpp11 cpp14 cpp17 cpp20 polymorphism discriminated-unions

448 commits | 6 branches | 0 packages | 8 releases | 8 contributors | BSL-1.0

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

gridley and mpark Workaround for NVCC 10.2. Latest commit 3c7fc82 on Jan 9

3rdparty	Updated googletest submodule and pointed it to abseil.	14 months ago
cmake	Rename `LICENSE_1_0.txt` to `LICENSE.md`.	3 years ago
include/mpark	Workaround for NVCC 10.2.	last month
support	Updated Wandbox script.	13 months ago
test	Implement P0608.	last month
.appveyor.yml	Fixed `MPARK_CPP14_CONSTEXPR` for MSVC 19.15.	13 months ago
.clang-format	Implemented initial version of `MPark.Variant`.	3 years ago
.gitignore	Ignored Google Benchmark submodule in `3rdparty/benchmark`.	14 months ago
.gitmodules	Updated googletest submodule and pointed it to abseil.	14 months ago
.travis.yml	Updated Travis CI.	last month
CMakeLists.txt	Fixes #60.	11 months ago
LICENSE.md	Move metabench license to the benchmark branch.	3 years ago
README.md	Updated Travis CI.	last month

README.md

MPark.Variant

C++17 `std::variant` for C++11/14/17

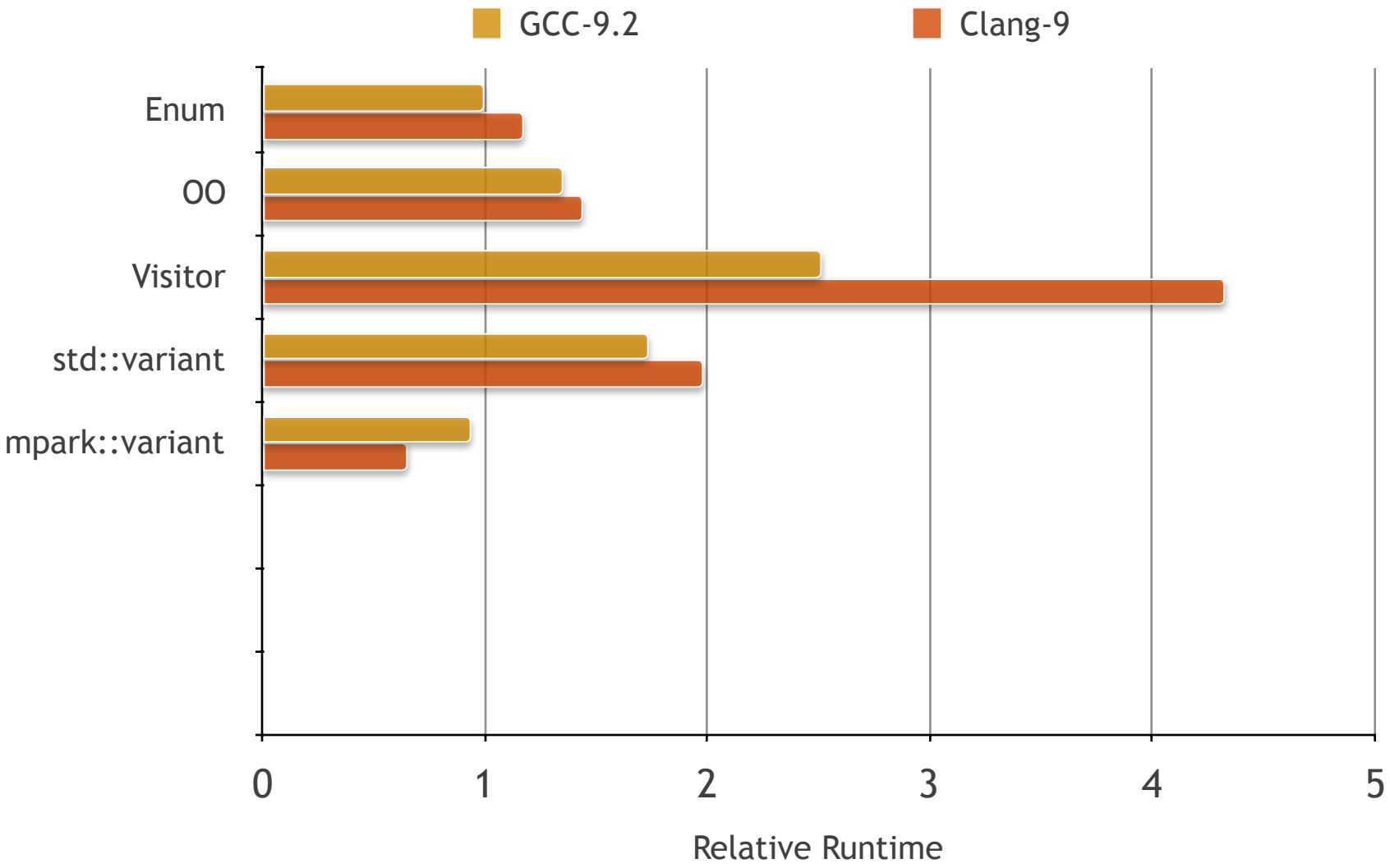
release v1.4.0 | single header master | build passing | build passing | license boost | try it on godbolt | try it on wandbox

Introduction

MPark.Variant is an implementation of **C++17** `std::variant` for **C++11/14/17**.

- Based on my implementation of `std::variant` for **libc++**
- Continuously tested against **libc++**'s `std::variant` test suite.

Performance Results



MacBook Pro 13-inch, MacOS 10.14.6 (Mojave), Intel Core i7, 2,7Ghz, 16 GByte, Compilation flags: -std=c++17 -O3 -DNDEBUG

100 random shapes, 2.5M updates each, normalized to the enum solution

Performance Analysis

```
// ...

switch (v.index()) {
  case B + 0: return MPARK_DISPATCH(B + 0);
  case B + 1: return MPARK_DISPATCH(B + 1);
  case B + 2: return MPARK_DISPATCH(B + 2);
  case B + 3: return MPARK_DISPATCH(B + 3);
  case B + 4: return MPARK_DISPATCH(B + 4);
  case B + 5: return MPARK_DISPATCH(B + 5);
  case B + 6: return MPARK_DISPATCH(B + 6);
  case B + 7: return MPARK_DISPATCH(B + 7);
  case B + 8: return MPARK_DISPATCH(B + 8);
  case B + 9: return MPARK_DISPATCH(B + 9);
  case B + 10: return MPARK_DISPATCH(B + 10);
  case B + 11: return MPARK_DISPATCH(B + 11);
  case B + 12: return MPARK_DISPATCH(B + 12);
  case B + 13: return MPARK_DISPATCH(B + 13);
  case B + 14: return MPARK_DISPATCH(B + 14);
  case B + 15: return MPARK_DISPATCH(B + 15);
  case B + 16: return MPARK_DISPATCH(B + 16);
  case B + 17: return MPARK_DISPATCH(B + 17);
  case B + 18: return MPARK_DISPATCH(B + 18);
  case B + 19: return MPARK_DISPATCH(B + 19);
  case B + 20: return MPARK_DISPATCH(B + 20);
  case B + 21: return MPARK_DISPATCH(B + 21);
  case B + 22: return MPARK_DISPATCH(B + 22);
  case B + 23: return MPARK_DISPATCH(B + 23);
  case B + 24: return MPARK_DISPATCH(B + 24);
  case B + 25: return MPARK_DISPATCH(B + 25);
  case B + 26: return MPARK_DISPATCH(B + 26);
```

Performance Analysis

```
    case B + 8: return MPARK_DISPATCH(B + 8);
    case B + 9: return MPARK_DISPATCH(B + 9);
    case B + 10: return MPARK_DISPATCH(B + 10);
    case B + 11: return MPARK_DISPATCH(B + 11);
    case B + 12: return MPARK_DISPATCH(B + 12);
    case B + 13: return MPARK_DISPATCH(B + 13);
    case B + 14: return MPARK_DISPATCH(B + 14);
    case B + 15: return MPARK_DISPATCH(B + 15);
    case B + 16: return MPARK_DISPATCH(B + 16);
    case B + 17: return MPARK_DISPATCH(B + 17);
    case B + 18: return MPARK_DISPATCH(B + 18);
    case B + 19: return MPARK_DISPATCH(B + 19);
    case B + 20: return MPARK_DISPATCH(B + 20);
    case B + 21: return MPARK_DISPATCH(B + 21);
    case B + 22: return MPARK_DISPATCH(B + 22);
    case B + 23: return MPARK_DISPATCH(B + 23);
    case B + 24: return MPARK_DISPATCH(B + 24);
    case B + 25: return MPARK_DISPATCH(B + 25);
    case B + 26: return MPARK_DISPATCH(B + 26);
    case B + 27: return MPARK_DISPATCH(B + 27);
    case B + 28: return MPARK_DISPATCH(B + 28);
    case B + 29: return MPARK_DISPATCH(B + 29);
    case B + 30: return MPARK_DISPATCH(B + 30);
    case B + 31: return MPARK_DISPATCH(B + 31);
    default: return MPARK_DEFAULT(B + 32);
  }

  // ...
```

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	1
mpark::variant	3	9	9	9	9

1 = very bad, …, 9 = very good

What kind of solution is that?

std::variant<Types...> VS INHERITANCE

INHERITANCE

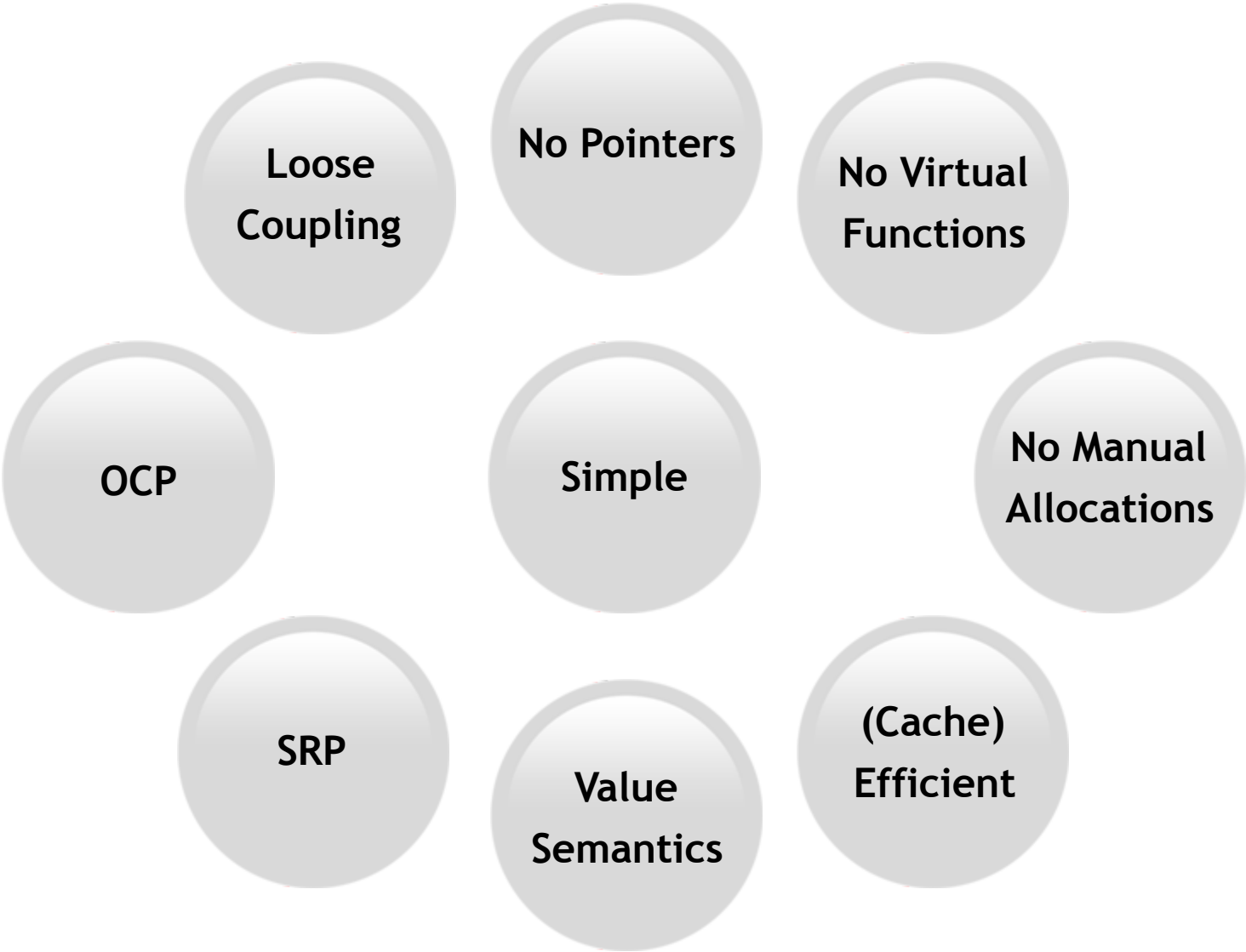
- Open/Closed to new alternatives
- Closed to new operations
- Multi-level
- OO
- Pointer semantics
- Design forced by the implementation details
- Forces dynamic memory allocations

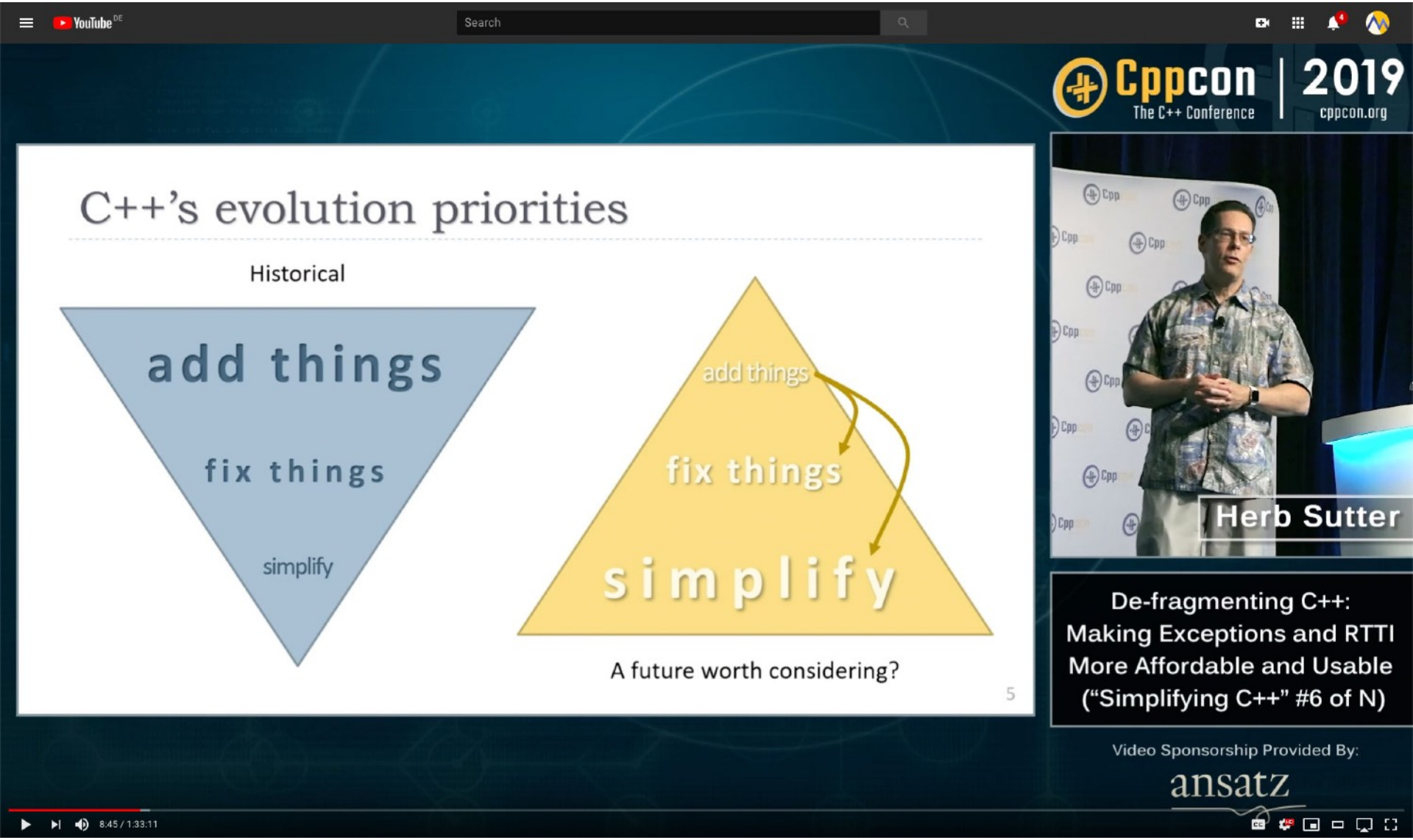
VARIANT

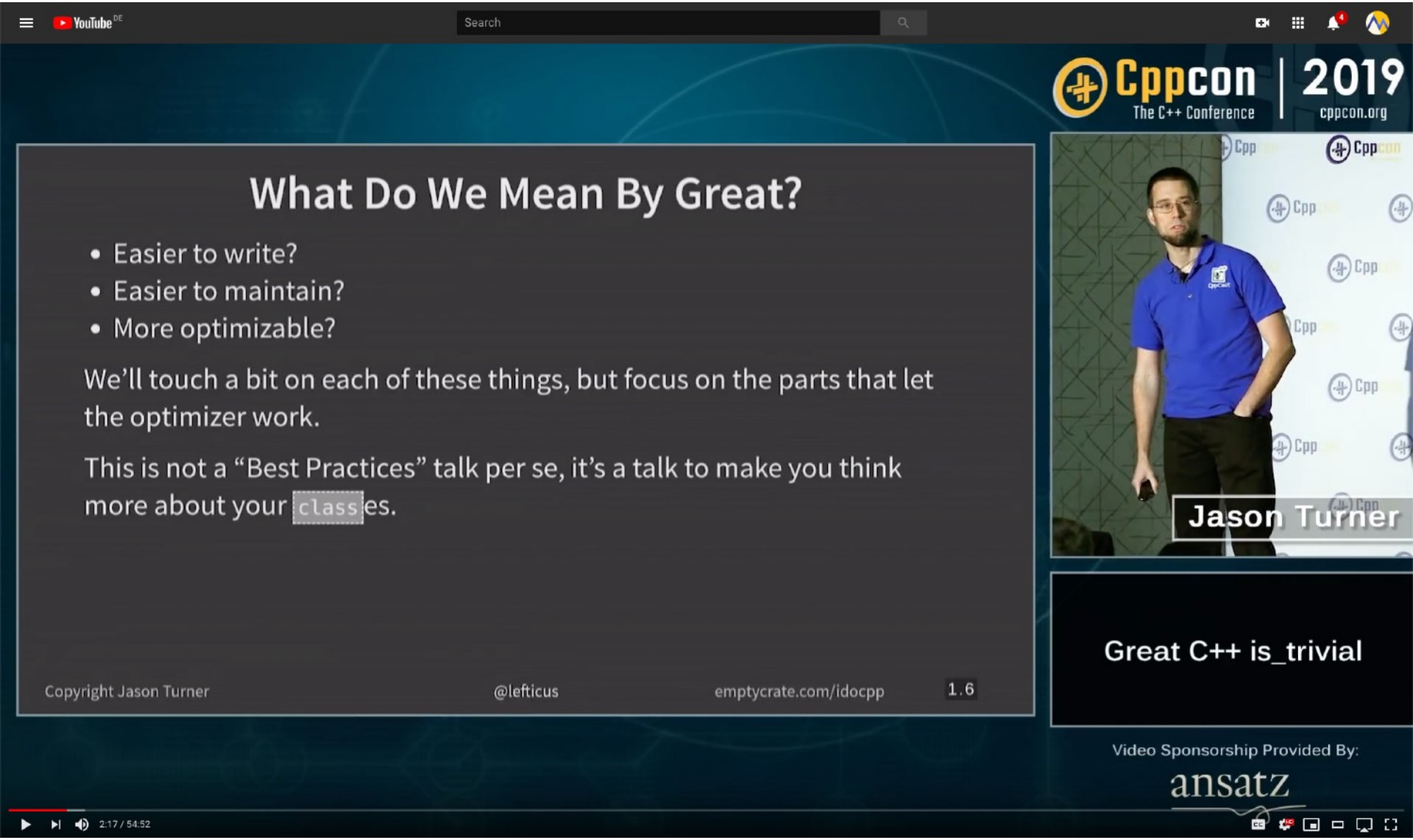
- Closed to new alternatives
- Open to new operations
- Single level
- Functional
- Value semantics
- Many design choices possible
- No dynamic memory allocations

Effective replacement of dynamic polymorphism with std::variant

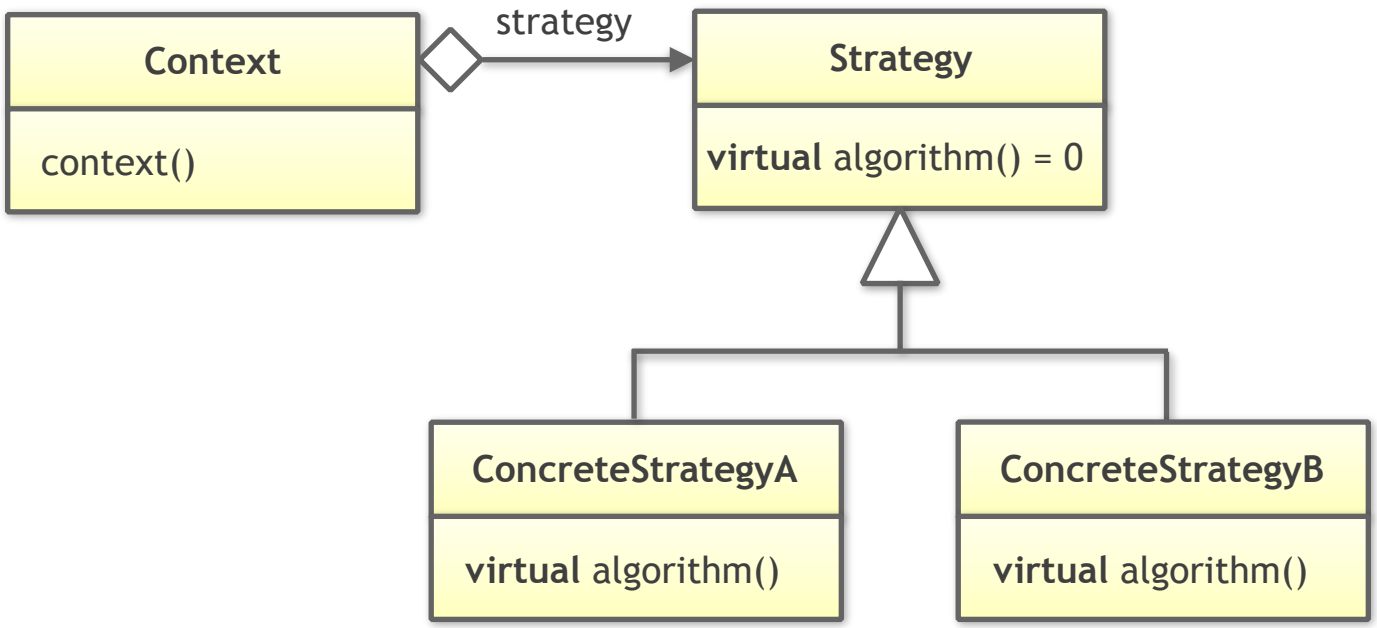
Solution Analysis



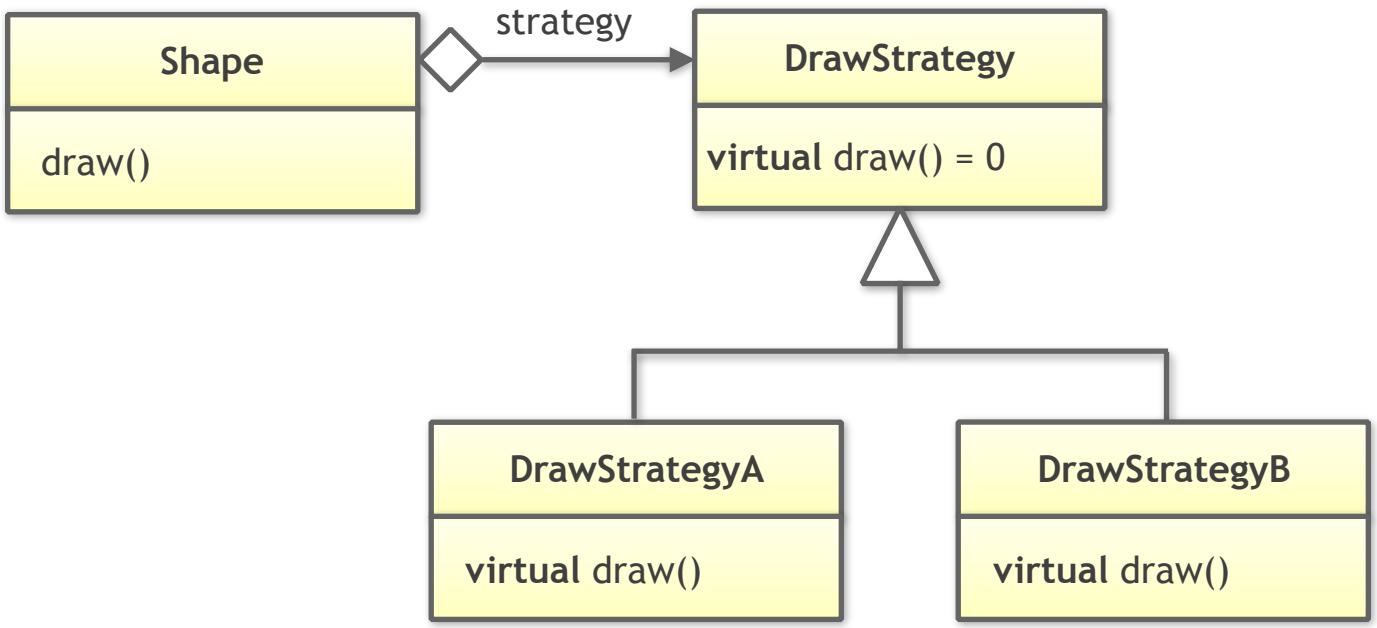

YouTube DE
Search
Cppcon
The C++ Conference
2019
cppcon.org
C++'s evolution priorities
Historical
add things
fix things
simplify
add things
fix things
simplify
A future worth considering?
5
Herb Sutter
De-fragmenting C++:
Making Exceptions and RTTI
More Affordable and Usable
("Simplifying C++" #6 of N)
Video Sponsorship Provided By:
ansatz
8:45 / 1:33:11


YouTube DE
Search
Cppcon
The C++ Conference
2019
cppcon.org
What Do We Mean By Great?
• Easier to write?
• Easier to maintain?
• More optimizable?
We'll touch a bit on each of these things, but focus on the parts that let the optimizer work.
This is not a "Best Practices" talk per se, it's a talk to make you think more about your classes.
Copyright Jason Turner
@lefticus
emptycrate.com/idocpp
1.6
Jason Turner
Great C++ is_trivial
Video Sponsorship Provided By:
ansatz
2:17 / 54:52

The Strategy Design Pattern



The Strategy Design Pattern



A Strategy-Based Solution

```
class Circle;
class Square;

class DrawStrategy
{
 public:
   virtual ~DrawStrategy() {}

   virtual void draw( const Circle& circle ) const = 0;
   virtual void draw( const Square& square ) const = 0;
};


class Shape
{
 public:
   Shape() = default;
   virtual ~Shape() = default;

   virtual void translate( Vector3D const& ) = 0;
   virtual void rotate( Quaternion const& ) = 0;
   virtual void draw() const = 0;
};


class Circle : public Shape
{
 public:
   explicit Circle( double rad, std::unique_ptr<DrawStrategy> ds )
      : radius{ rad }
```

A Strategy-Based Solution

```
class Circle;
class Square;

class DrawStrategy
{
 public:
   virtual ~DrawStrategy() {}

   virtual void draw( const Circle& circle ) const = 0;
   virtual void draw( const Square& square ) const = 0;
};


class Shape
{
 public:
   Shape() = default;
   virtual ~Shape() = default;

   virtual void translate( Vector3D const& ) = 0;
   virtual void rotate( Quaternion const& ) = 0;
   virtual void draw() const = 0;
};


class Circle : public Shape
{
 public:
   explicit Circle( double rad, std::unique_ptr<DrawStrategy> ds )
      : radius{ rad }
```

A Strategy-Based Solution

```
class DrawStrategy
{
 public:
   virtual ~DrawStrategy() {}

   virtual void draw( const Circle& circle ) const = 0;
   virtual void draw( const Square& square ) const = 0;
};


class Shape
{
 public:
   Shape() = default;
   virtual ~Shape() = default;

   virtual void translate( Vector3D const& ) = 0;
   virtual void rotate( Quaternion const& ) = 0;
   virtual void draw() const = 0;
};


class Circle : public Shape
{
 public:
   explicit Circle( double rad, std::unique_ptr<DrawStrategy> ds )
      : radius{ rad }
      , // ... Remaining data members
      , drawing{ std::move(ds) }
   {}

   virtual ~Circle() = default;
```

A Strategy-Based Solution

```
   virtual void draw() const = 0;
};


class Circle : public Shape
{
 public:
   explicit Circle( double rad, std::unique_ptr<DrawStrategy> ds )
      : radius{ rad }
      , // ... Remaining data members
      , drawing{ std::move(ds) }
   {}

   virtual ~Circle() = default;

   double getRadius() const noexcept;
   // ... getCenter(), getRotation(), ...

   void translate( Vector3D const& ) override;
   void rotate( Quaternion const& ) override;
   void draw() const override;

 private:
   double radius;
   // ... Remaining data members
   std:unique_ptr<DrawStrategy> drawing;
};



class Square : public Shape
{
 public:
```

A Strategy-Based Solution

```
};


class Square : public Shape
{
 public:
   explicit Square( double s, std::unique_ptr<DrawStrategy> ds )
      : side{ s }
      , // ... Remaining data members
      , drawing{ std::move(ds) }
   {}

   virtual ~Square() = default;

   double getSide() const noexcept;
   // ... getCenter(), getRotation(), ...

   void translate( Vector3D const& ) override;
   void rotate( Quaternion const& ) override;
   void draw() const override;

 private:
   double side;
   // ... Remaining data members
   std::unique_ptr<DrawStrategy> drawing;
};


void draw( std::vector<std::unique_ptr<Shape>> const& shapes )
{
   for( auto const& s : shapes )
   {
```

A Strategy-Based Solution

```
   void translate( Vector3D const& ) override;
   void rotate( Quaternion const& ) override;
   void draw() const override;

 private:
   double side;
   // ... Remaining data members
   std::unique_ptr<DrawStrategy> drawing;
};


void draw( std::vector<std::unique_ptr<Shape>> const& shapes )
{
   for( auto const& s : shapes )
   {
      s->draw()
   }
}


class OpenGLStrategy : public DrawStrategy
{
 public:
   virtual ~OpenGLStrategy() {}

   void draw( Circle const& circle ) const override;
   void draw( Square const& square ) const override;
};


int main()
{
   using Shapes = std::vector<std::unique_ptr<Shape>>;
```

A Strategy-Based Solution

```
};

void draw( std::vector<std::unique_ptr<Shape>> const& shapes )
{
   for( auto const& s : shapes )
   {
      s->draw()
   }
}

class OpenGLStrategy : public DrawStrategy
{
 public:
   virtual ~OpenGLStrategy() {}

   void draw( Circle const& circle ) const override;
   void draw( Square const& square ) const override;
};

int main()
{
   using Shapes = std::vector<std::unique_ptr<Shape>>;

   // Creating some shapes
   Shapes shapes;
   shapes.push_back( std::make_unique<Circle>( 2.0
                          , std::make_unique<OpenGLStrategy>() ) );
   shapes.push_back( std::make_unique<Square>( 1.5
                          , std::make_unique<OpenGLStrategy>() ) );
   shapes.push_back( std::make_unique<Circle>( 4.2
```

A Strategy-Based Solution

```
{
 public:
   virtual ~OpenGLStrategy() {}

   void draw( Circle const& circle ) const override;
   void draw( Square const& square ) const override;
};


int main()
{
   using Shapes = std::vector<std::unique_ptr<Shape>>;

   // Creating some shapes
   Shapes shapes;
   shapes.push_back( std::make_unique<Circle>( 2.0
                          , std::make_unique<OpenGLStrategy>() ) );
   shapes.push_back( std::make_unique<Square>( 1.5
                          , std::make_unique<OpenGLStrategy>() ) );
   shapes.push_back( std::make_unique<Circle>( 4.2
                          , std::make_unique<OpenGLStrategy>() ) );

   // Drawing all shapes
   draw( shapes );
}
```

A Strategy-Based Solution

```
class Circle;
class Square;

class DrawStrategy
{
 public:
   virtual ~DrawStrategy() {}

   virtual void draw( const Circle& circle ) const = 0;
   virtual void draw( const Square& square ) const = 0;
};


class Shape
{
 public:
   Shape() = default;
   virtual ~Shape() = default;

   virtual void translate( Vector3D const& ) = 0;
   virtual void rotate( Quaternion const& ) = 0;
   virtual void draw() const = 0;
};


class Circle : public Shape
{
 public:
   explicit Circle( double rad, std::unique_ptr<DrawStrategy> ds )
      : radius{ rad }
```

A Strategy-Based Solution

```
class Circle;
class Square;

class DrawCircleStrategy
{
 public:
   virtual ~DrawCircleStrategy() {}

   virtual void draw( const Circle& circle ) const = 0;
};

class DrawSquareStrategy
{
 public:
   virtual ~DrawSquareStrategy() {}

   virtual void draw( const Square& square ) const = 0;
};

class Shape
{
 public:
   Shape() = default;
   virtual ~Shape() = default;

   virtual void translate( Vector3D const& ) = 0;
   virtual void rotate( Quaternion const& ) = 0;
   virtual void draw() const = 0;
};
```

A Strategy-Based Solution

```
   virtual void rotate( Quaternion const& ) = 0;
   virtual void draw() const = 0;
};


class Circle : public Shape
{
 public:
   explicit Circle( double rad, std::unique_ptr<DrawCircleStrategy> ds )
      : radius{ rad }
      , // ... Remaining data members
      , drawing{ std::move(ds) }
   {}

   virtual ~Circle() = default;

   double getRadius() const noexcept;
   // ... getCenter(), getRotation(), ...

   void translate( Vector3D const& ) override;
   void rotate( Quaternion const& ) override;
   void draw() const override;

 private:
   double radius;
   // ... Remaining data members
   std:unique_ptr<DrawCircleStrategy> drawing;
};


class Square : public Shape
{
```

A Strategy-Based Solution

```
   std:unique_ptr<DrawCircleStrategy> drawing;
};


class Square : public Shape
{
 public:
   explicit Square( double s, std::unique_ptr<DrawSquareStrategy> ds )
      : side{ s }
      , // ... Remaining data members
      , drawing{ std::move(ds) }
   {}

   virtual ~Square() = default;

   double getSide() const noexcept;
   // ... getCenter(), getRotation(), ...

   void translate( Vector3D const& ) override;
   void rotate( Quaternion const& ) override;
   void draw() const override;

 private:
   double side;
   // ... Remaining data members
   std::unique_ptr<DrawSquareStrategy> drawing;
};


void draw( std::vector<std::unique_ptr<Shape>> const& shapes )
{
   for( auto const& s : shapes )
   {
```

A Strategy-Based Solution

```
{
   for( auto const& s : shapes )
   {
      s->draw()
   }
}


class OpenGLCircleStrategy : public DrawCircleStrategy
{
 public:
   virtual ~OpenGLStrategy() {}

   void draw( Circle const& circle ) const override;
};

class OpenGLSquareStrategy : public DrawSquareStrategy
{
 public:
   virtual ~OpenGLStrategy() {}

  void draw( Square const& square ) const override;
};


int main()
{
   using Shapes = std::vector<std::unique_ptr<Shape>>;

   // Creating some shapes
   Shapes shapes;
   shapes.push_back( std::make_unique<Circle>( 2.0
                        , std::make_unique<OpenGLCircleStrategy>() ) );
```

A Strategy-Based Solution

```
{
 public:
   virtual ~OpenGLStrategy() {}

  void draw( Square const& square ) const override;
};


int main()
{
   using Shapes = std::vector<std::unique_ptr<Shape>>;

   // Creating some shapes
   Shapes shapes;
   shapes.push_back( std::make_unique<Circle>( 2.0
                          , std::make_unique<OpenGLCircleStrategy>() ) );
   shapes.push_back( std::make_unique<Square>( 1.5
                          , std::make_unique<OpenGLSquareStrategy>() ) );
   shapes.push_back( std::make_unique<Circle>( 4.2
                          , std::make_unique<OpenGLCircleStrategy>() ) );

   // Drawing all shapes
   draw( shapes );
}
```

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	1
mpark::variant	3	9	9	9	9
Strategy					

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	1
mpark::variant	3	9	9	9	9
Strategy	7				

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	1
mpark::variant	3	9	9	9	9
Strategy	7	2			

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	1
mpark::variant	3	9	9	9	9
Strategy	7	2	7		

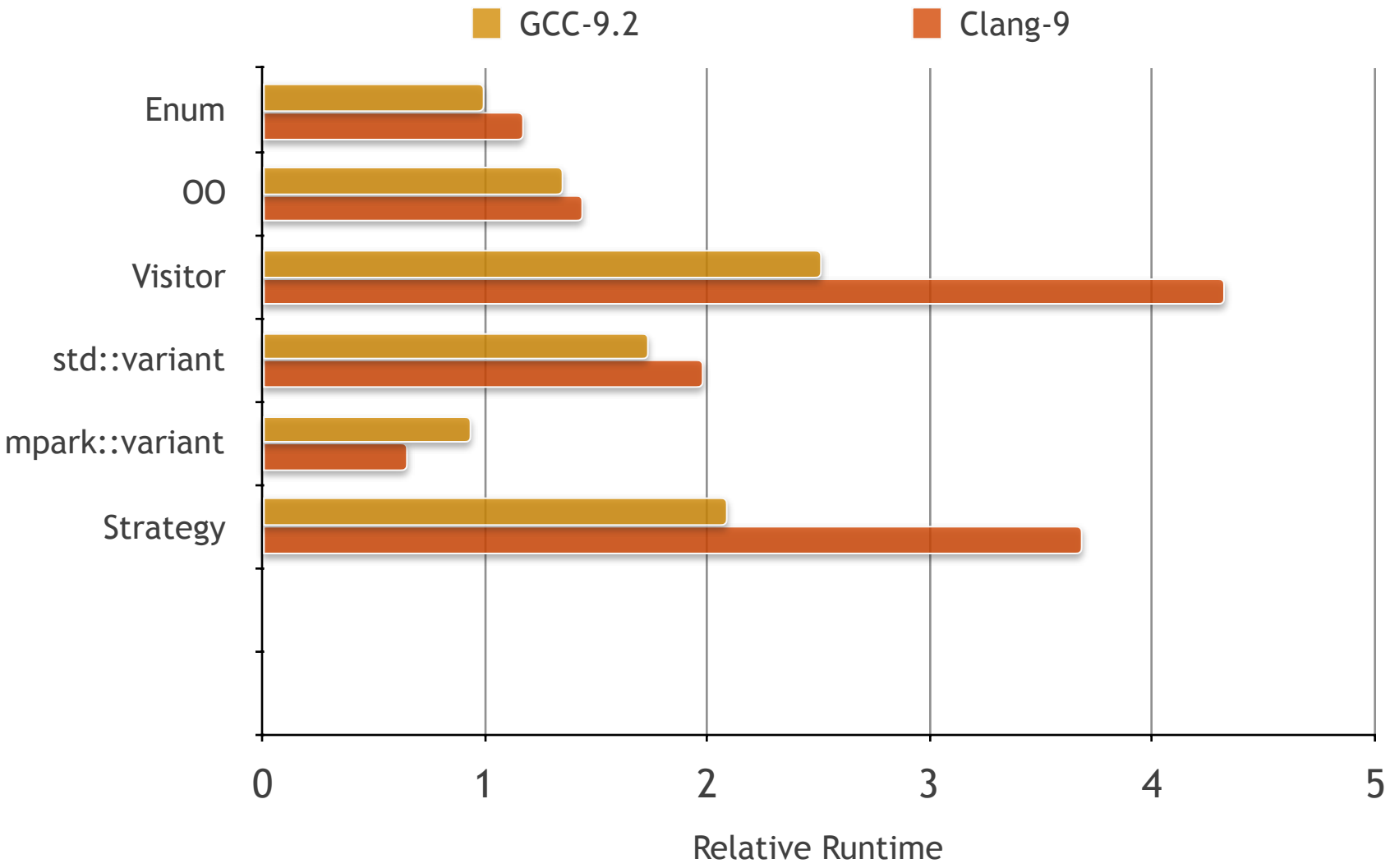
1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	1
mpark::variant	3	9	9	9	9
Strategy	7	2	7	4	

1 = very bad, …, 9 = very good

Performance Results



MacBook Pro 13-inch, MacOS 10.14.6 (Mojave), Intel Core i7, 2,7Ghz, 16 GByte, Compilation flags: -std=c++17 -O3 -DNDEBUG

100 random shapes, 2.5M updates each, normalized to the enum solution

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	1
mpark::variant	3	9	9	9	9
Strategy	7	2	7	4	2

1 = very bad, …, 9 = very good

A “Modern C++” Solution

```
class Circle;
class Square;

using DrawCircleStrategy = std::function<void(Circle const&)>;
using DrawSquareStrategy = std::function<void(Square const&)>;


class Shape
{
 public:
   Shape() = default;
   virtual ~Shape() = default;

   virtual void translate( Vector3D const& ) = 0;
   virtual void rotate( Quaternion const& ) = 0;
   virtual void draw() const = 0;
};


class Circle : public Shape
{
 public:
   explicit Circle( double rad, DrawCircleStrategy ds )
      : radius{ rad }
      , // ... Remaining data members
      , drawing{ std::move(ds) }
   {}

   virtual ~Circle() = default;
```

A “Modern C++” Solution

```
class Circle;
class Square;

using DrawCircleStrategy = std::function<void(Circle const&)>;
using DrawSquareStrategy = std::function<void(Square const&)>;


class Shape
{
 public:
   Shape() = default;
   virtual ~Shape() = default;

   virtual void translate( Vector3D const& ) = 0;
   virtual void rotate( Quaternion const& ) = 0;
   virtual void draw() const = 0;
};


class Circle : public Shape
{
 public:
   explicit Circle( double rad, DrawCircleStrategy ds )
      : radius{ rad }
      , // ... Remaining data members
      , drawing{ std::move(ds) }
   {}

   virtual ~Circle() = default;
```

A “Modern C++” Solution

```
class Circle;
class Square;

using DrawCircleStrategy = std::function<void(Circle const&)>;
using DrawSquareStrategy = std::function<void(Square const&)>;


class Shape
{
 public:
   Shape() = default;
   virtual ~Shape() = default;

   virtual void translate( Vector3D const& ) = 0;
   virtual void rotate( Quaternion const& ) = 0;
   virtual void draw() const = 0;
};


class Circle : public Shape
{
 public:
   explicit Circle( double rad, DrawCircleStrategy ds )
      : radius{ rad }
      , // ... Remaining data members
      , drawing{ std::move(ds) }
   {}

   virtual ~Circle() = default;
```

A “Modern C++” Solution

```
   virtual void rotate( Quaternion const& ) = 0;
   virtual void draw() const = 0;
};


class Circle : public Shape
{
 public:
   explicit Circle( double rad, DrawCircleStrategy ds )
      : radius{ rad }
      , // ... Remaining data members
      , drawing{ std::move(ds) }
   {}

   virtual ~Circle() = default;

   double getRadius() const noexcept;
   // ... getCenter(), getRotation(), ...

   void translate( Vector3D const& ) override;
   void rotate( Quaternion const& ) override;
   void draw() const override;

 private:
   double radius;
   // ... Remaining data members
   DrawCircleStrategy drawing;
};



class Square : public Shape
{
```

A “Modern C++” Solution

```
   DrawCircleStrategy drawing;
};


class Square : public Shape
{
 public:
   explicit Square( double s, DrawSquareStrategy ds )
      : side{ s }
      , // ... Remaining data members
      , drawing{ std::move(ds) }
   {}

   virtual ~Square() = default;

   double getSide() const noexcept;
   // ... getCenter(), getRotation(), ...

   void translate( Vector3D const& ) override;
   void rotate( Quaternion const& ) override;
   void draw() const override;

 private:
   double side;
   // ... Remaining data members
   DrawSquareStrategy drawing;
};


void draw( std::vector<std::unique_ptr<Shape>> const& shapes )
{
   for( auto const& s : shapes )
   {
```

A “Modern C++” Solution

```
void draw( std::vector<std::unique_ptr<Shape>> const& shapes )
{
   for( auto const& s : shapes )
   {
      s->draw()
   }
}


class OpenGLCircleStrategy
{
 public:
   void operator()( Circle const& circle ) const;
};

class OpenGLSquareStrategy
{
 public:
   void operator()( Square const& square ) const;
};


int main()
{
   using Shapes = std::vector<std::unique_ptr<Shape>>;

   // Creating some shapes
   Shapes shapes;
   shapes.push_back( std::make_unique<Circle>( 2.0
                          , OpenGLCircleStrategy{} ) );
   shapes.push_back( std::make_unique<Square>( 1.5
                          , OpenGLSquareStrategy{} ) );
```

A “Modern C++” Solution

```
};

class OpenGLSquareStrategy
{
 public:
   void operator()( Square const& square ) const;
};


int main()
{
   using Shapes = std::vector<std::unique_ptr<Shape>>;

   // Creating some shapes
   Shapes shapes;
   shapes.push_back( std::make_unique<Circle>( 2.0
                          , OpenGLCircleStrategy{} ) );
   shapes.push_back( std::make_unique<Square>( 1.5
                          , OpenGLSquareStrategy{} ) );
   shapes.push_back( std::make_unique<Circle>( 4.2
                          , OpenGLCircleStrategy{} ) );

   // Drawing all shapes
   draw( shapes );
}
```

A “Modern C++” Solution

```
void draw( std::vector<std::unique_ptr<Shape>> const& shapes )
{
   for( auto const& s : shapes )
   {
      s->draw()
   }
}


class OpenGLCircleStrategy
{
 public:
   void operator()( Circle const& circle ) const;
};

class OpenGLSquareStrategy
{
 public:
   void operator()( Square const& square ) const;
};


int main()
{
   using Shapes = std::vector<std::unique_ptr<Shape>>;

   // Creating some shapes
   Shapes shapes;
   shapes.push_back( std::make_unique<Circle>( 2.0
                           , OpenGLCircleStrategy{} ) );
   shapes.push_back( std::make_unique<Square>( 1.5
                           , OpenGLSquareStrategy{} ) );
```

A “Modern C++” Solution

```
void draw( std::vector<std::unique_ptr<Shape>> const& shapes )
{
   for( auto const& s : shapes )
   {
      s->draw()
   }
}
```

```
void draw( Circle const& circle ) const;
void draw( Square const& square ) const;
```

```
int main()
{
   using Shapes = std::vector<std::unique_ptr<Shape>>;

   // Creating some shapes
   Shapes shapes;
   shapes.push_back( std::make_unique<Circle>( 2.0, draw ) );
   shapes.push_back( std::make_unique<Square>( 1.5, draw ) );
   shapes.push_back( std::make_unique<Circle>( 4.2, draw ) );
```

A “Modern C++” Solution

```
void draw( std::vector<std::unique_ptr<Shape>> const& shapes )
{
   for( auto const& s : shapes )
   {
      s->draw()
   }
}
```

```
void draw( Circle const& circle ) const;
void draw( Square const& square ) const;

struct Draw
{
   template< typename T >
   void operator()( T const& drawable ) const {
      draw( drawable );
   }
};
```

```
int main()
{
   using Shapes = std::vector<std::unique_ptr<Shape>>;

   // Creating some shapes
   Shapes shapes;
   shapes.push_back( std::make_unique<Circle>( 2.0, Draw{} ) );
   shapes.push_back( std::make_unique<Square>( 1.5, Draw{} ) );
   shapes.push_back( std::make_unique<Circle>( 4.2, Draw{} ) );
```

A “Modern C++” Solution

```
struct Draw
{
   template< typename T >
   void operator()( T const& drawable ) const {
      draw( drawable );
   }
};


int main()
{
   using Shapes = std::vector<std::unique_ptr<Shape>>;

   // Creating some shapes
   Shapes shapes;
   shapes.push_back( std::make_unique<Circle>( 2.0, Draw{} ) );
   shapes.push_back( std::make_unique<Square>( 1.5, Draw{} ) );
   shapes.push_back( std::make_unique<Circle>( 4.2, Draw{} ) );

   // Drawing all shapes
   draw( shapes );
}
```

A “Modern C++” Solution

```
void draw( std::vector<std::unique_ptr<Shape>> const& shapes )
{
   for( auto const& s : shapes )
   {
      s->draw()
   }
}
```

```
void draw( Circle const& circle ) const;
void draw( Square const& square ) const;

struct Draw
{
   template< typename T >
   void operator()( T const& drawable ) const {
      draw( drawable );
   }
};
```

```
int main()
{
   using Shapes = std::vector<std::unique_ptr<Shape>>;

   // Creating some shapes
   Shapes shapes;
   shapes.push_back( std::make_unique<Circle>( 2.0, Draw{} ) );
   shapes.push_back( std::make_unique<Square>( 1.5, Draw{} ) );
   shapes.push_back( std::make_unique<Circle>( 4.2, Draw{} ) );
```

A “Modern C++” Solution

```
void draw( std::vector<std::unique_ptr<Shape>> const& shapes )
{
   for( auto const& s : shapes )
   {
      s->draw()
   }
}
```

```
void draw( Circle const& circle ) const;
void draw( Square const& square ) const;
```

```
int main()
{
   using Shapes = std::vector<std::unique_ptr<Shape>>;

   // Creating some shapes
   Shapes shapes;
   shapes.push_back( std::make_unique<Circle>( 2.0, Draw{} ) );
   shapes.push_back( std::make_unique<Square>( 1.5, Draw{} ) );
   shapes.push_back( std::make_unique<Circle>( 4.2, Draw{} ) );

   // Drawing all shapes
```

A “Modern C++” Solution

```
class Circle;
class Square;

using DrawCircleStrategy = std::function<void(Circle const&)>;
using DrawSquareStrategy = std::function<void(Square const&)>;

struct Draw
{
   template< typename T >
   void operator()( T const& drawable ) const {
      draw( drawable );
   }
};


class Shape
{
 public:
   Shape() = default;
   virtual ~Shape() = default;

   virtual void translate( Vector3D const& ) = 0;
   virtual void rotate( Quaternion const& ) = 0;
   virtual void draw() const = 0;
};


class Circle : public Shape
{
 public:
   explicit Circle( double rad, DrawCircleStrategy ds = Draw{} )
```

A “Modern C++” Solution

```
   virtual void rotate( Quaternion const& ) = 0;
   virtual void draw() const = 0;
};

class Circle : public Shape
{
 public:
   explicit Circle( double rad, DrawCircleStrategy ds = Draw{} )
      : radius{ rad }
      , // ... Remaining data members
      , drawing{ std::move(ds) }
   {}

   virtual ~Circle() = default;

   double getRadius() const noexcept;
   // ... getCenter(), getRotation(), ...

   void translate( Vector3D const& ) override;
   void rotate( Quaternion const& ) override;
   void draw() const override;

 private:
   double radius;
   // ... Remaining data members
   DrawCircleStrategy drawing;
};


class Square : public Shape
{
 public:
```

A “Modern C++” Solution

```
   DrawCircleStrategy drawing;
};


class Square : public Shape
{
 public:
   explicit Square( double s, DrawSquareStrategy ds = Draw{} )
      : side{ s }
      , // ... Remaining data members
      , drawing{ std::move(ds) }
   {}

   virtual ~Square() = default;

   double getSide() const noexcept;
   // ... getCenter(), getRotation(), ...

   void translate( Vector3D const& ) override;
   void rotate( Quaternion const& ) override;
   void draw() const override;

 private:
   double side;
   // ... Remaining data members
   DrawSquareStrategy drawing;
};


void draw( std::vector<std::unique_ptr<Shape>> const& shapes )
{
   for( auto const& s : shapes )
   {
```

A “Modern C++” Solution

```
    }
}


void draw( Circle const& circle ) const;
void draw( Square const& square ) const;


int main()
{
   using Shapes = std::vector<std::unique_ptr<Shape>>;

   // Creating some shapes
   Shapes shapes;
   shapes.push_back( std::make_unique<Circle>( 2.0 ) );
   shapes.push_back( std::make_unique<Square>( 1.5 ) );
   shapes.push_back( std::make_unique<Circle>( 4.2 ) );

   // Drawing all shapes
   draw( shapes );
}
```

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	1
mpark::variant	3	9	9	9	9
Strategy	7	2	7	4	2
std::function					

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	1
mpark::variant	3	9	9	9	9
Strategy	7	2	7	4	2
std::function	8				

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	1
mpark::variant	3	9	9	9	9
Strategy	7	2	7	4	2
std::function	8	3			

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	1
mpark::variant	3	9	9	9	9
Strategy	7	2	7	4	2
std::function	8	3	7		

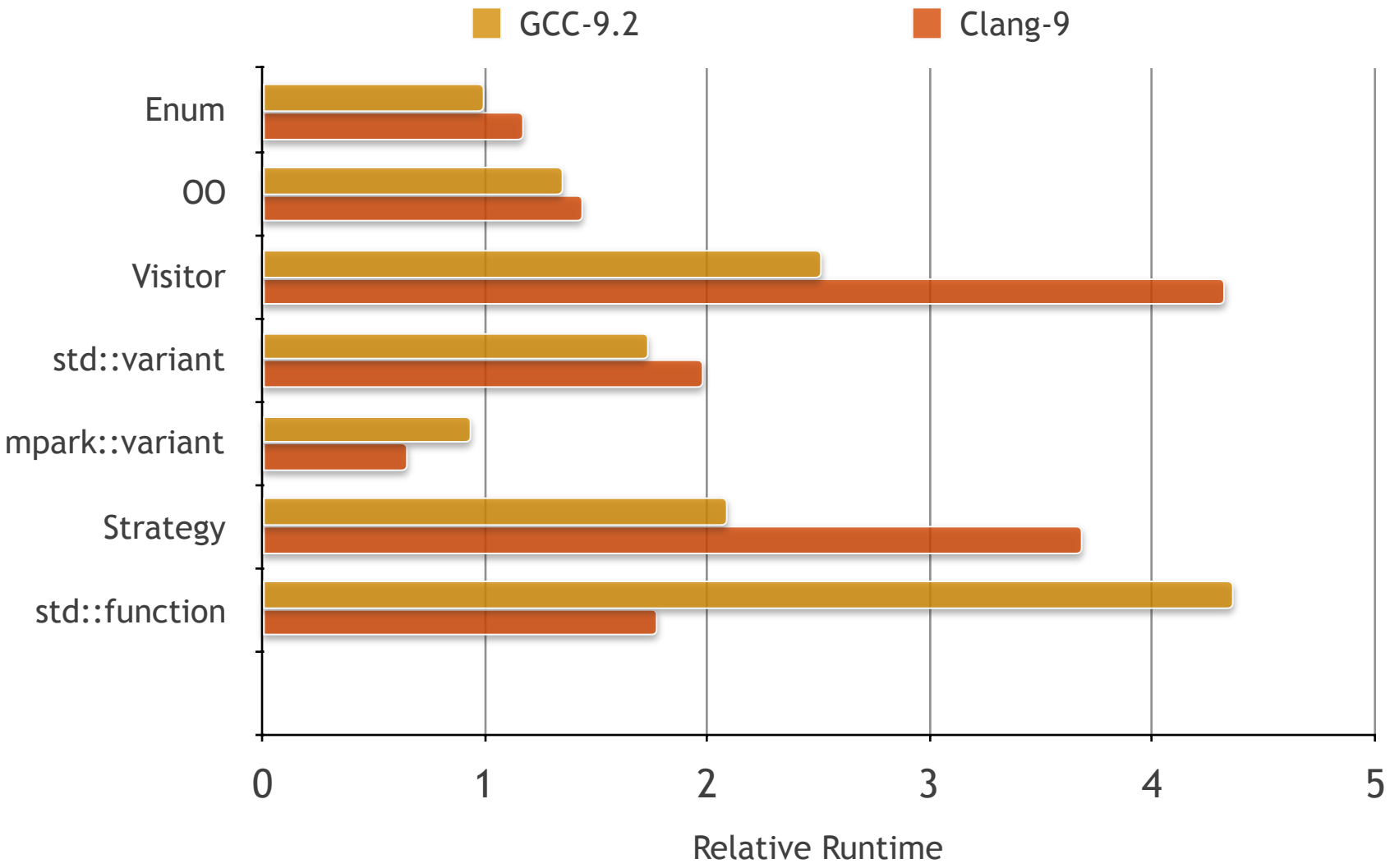
1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	1
mpark::variant	3	9	9	9	9
Strategy	7	2	7	4	2
std::function	8	3	7	7	

1 = very bad, …, 9 = very good

Performance Results



MacBook Pro 13-inch, MacOS 10.14.6 (Mojave), Intel Core i7, 2,7Ghz, 16 GByte, Compilation flags: -std=c++17 -O3 -DNDEBUG

100 random shapes, 2.5M updates each, normalized to the enum solution

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	1
mpark::variant	3	9	9	9	9
Strategy	7	2	7	4	2
std::function	8	3	7	7	5

1 = very bad, …, 9 = very good

A Type-Erased Solution

```
class Circle
{
 public:
   explicit Circle( double rad )
      : radius{ rad }
      , // ... Remaining data members
   {}

   double getRadius() const noexcept;
   // ... getCenter(), getRotation(), ...

 private:
   double radius;
   // ... Remaining data members
};

void translate( Circle const&, Vector3D const& );
void rotate( Circle const&, Quaternion const& );
void draw( Circle const& );


class Square
{
 public:
   explicit Square( double s )
      : side{ s }
      , // ... Remaining data members
   {}

   double getSide() const noexcept;
```

A Type-Erased Solution

```
class Circle
{
 public:
   explicit Circle( double rad )
      : radius{ rad }
      , // ... Remaining data members
   {}

   double getRadius() const noexcept;
   // ... getCenter(), getRotation(), ...

 private:
   double radius;
   // ... Remaining data members
};

void translate( Circle const&, Vector3D const& );
void rotate( Circle const&, Quaternion const& );
void draw( Circle const& );


class Square
{
 public:
   explicit Square( double s )
      : side{ s }
      , // ... Remaining data members
   {}

   double getSide() const noexcept;
```

A Type-Erased Solution

```
void translate( Circle const&, Vector3D const& );
void rotate( Circle const&, Quaternion const& );
void draw( Circle const& );


class Square
{
 public:
   explicit Square( double s )
      : side{ s }
      , // ... Remaining data members
   {}

   double getSide() const noexcept;
   // ... getCenter(), getRotation(), ...

 private:
   double side;
   // ... Remaining data members
};

void translate( Square const&, Vector3D const& );
void rotate( Square const&, Quaternion const& );
void draw( Square const& );


class Shape
{
 private:
   struct Concept
   {
      virtual ~Concept() {}
```

A Type-Erased Solution

```
void translate( Square const&, Vector3D const& );
void rotate( Square const&, Quaternion const& );
void draw( Square const& );


class Shape
{
 private:
   struct Concept
   {
      virtual ~Concept() {}
      virtual void do_translate( Vector3D const& v ) const = 0;
      virtual void do_rotate( Quaternion const& q ) const = 0;
      virtual void do_draw() const = 0;
      // ...
   };

   template< typename T >
   struct Model : Concept
   {
      Model( T const& value )
         : object{ value }
      {}

      void do_translate( Vector3D const& v ) const override
      {
         translate( object, v );
      }

      void do_rotate( Quaternion const& q ) const override
      {
         rotate( object, q );
```

A Type-Erased Solution

```
};

template< typename T >
struct Model : Concept
{
   Model( T const& value )
      : object{ value }
   {}

   void do_translate( Vector3D const& v ) const override
   {
      translate( object, v );
   }

   void do_rotate( Quaternion const& q ) const override
   {
      rotate( object, q );
   }

   void do_draw() const override
   {
      draw( object );
   }

   // ...

   T object;
};

std::unique_ptr<Concept> pimpl;

friend void translate( Shape& shape, Vector3D const& v )
{
```

A Type-Erased Solution

```
void translate( Square const&, Vector3D const& );
void rotate( Square const&, Quaternion const& );
void draw( Square const& );


class Shape
{
 private:
   struct Concept
   {
      virtual ~Concept() {}
      virtual void do_translate( Vector3D const& v ) const = 0;
      virtual void do_rotate( Quaternion const& q ) const = 0;
      virtual void do_draw() const = 0;
      // ...
   };

   template< typename T >
   struct Model : Concept
   {
      Model( T const& value )
         : object{ value }
      {}

      void do_translate( Vector3D const& v ) const override
      {
         translate( object, v );
      }

      void do_rotate( Quaternion const& q ) const override
      {
         rotate( object, q );
```

A Type-Erased Solution

```
      // ...

      T object;
   };

   std::unique_ptr<Concept> pimpl;

   friend void translate( Shape& shape, Vector3D const& v )
   {
      shape.pimpl->do_translate( v );
   }

   friend void rotate( Shape& shape, Quaternion const& q )
   {
      shape.pimpl->do_rotate( q );
   }

   friend void draw( Shape const& shape )
   {
      shape.pimpl->do_draw();
   }

public:
   template< typename T >
   Shape( T const& x )
      : pimpl{ new Model<T>( x ) }
   {}

   // Special member functions
   Shape( Shape const& s );
   Shape( Shape&& s );
   Shape& operator=( Shape const& s );
```

A Type-Erased Solution

```
      // ...

      T object;
   };

   std::unique_ptr<Concept> pimpl;

   friend void translate( Shape& shape, Vector3D const& v )
   {
      shape.pimpl->do_translate( v );
   }

   friend void rotate( Shape& shape, Quaternion const& q )
   {
      shape.pimpl->do_rotate( q );
   }

   friend void draw( Shape const& shape )
   {
      shape.pimpl->do_draw();
   }

public:
   template< typename T >
   Shape( T const& x )
      : pimpl{ new Model<T>( x ) }
   {}

   // Special member functions
   Shape( Shape const& s );
   Shape( Shape&& s );
   Shape& operator=( Shape const& s );
```

A Type-Erased Solution

```
   friend void draw( Shape const& shape )
   {
      shape.pimpl->do_draw();
   }

 public:
   template< typename T >
   Shape( T const& x )
      : pimpl{ new Model<T>( x ) }
   {}

   // Special member functions
   Shape( Shape const& s );
   Shape( Shape&& s );
   Shape& operator=( Shape const& s );
   Shape& operator=( Shape&& s );

   // ...
};


void draw( std::vector<Shape> const& shapes )
{
   for( auto const& shape : shapes )
   {
      draw( shape );
   }
}


int main()
{
```

A Type-Erased Solution

```
   // Special member functions
   Shape( Shape const& s );
   Shape( Shape&& s );
   Shape& operator=( Shape const& s );
   Shape& operator=( Shape&& s );

   // ...
};


void draw( std::vector<Shape> const& shapes )
{
   for( auto const& shape : shapes )
   {
      draw( shape );
   }
}


int main()
{
   using Shapes = std::vector<Shape>;

   // Creating some shapes
   Shapes shapes;
   shapes.push_back( Circle{ 2.0 } );
   shapes.push_back( Square{ 1.5 } );
   shapes.push_back( Circle{ 4.2 } );

   // Drawing all shapes
   draw( shapes );
}
```

A Type-Erased Solution

```
void draw( std::vector<Shape> const& shapes )
{
   for( auto const& shape : shapes )
   {
      draw( shape );
   }
}


int main()
{
   using Shapes = std::vector<Shape>;

   // Creating some shapes
   Shapes shapes;
   shapes.push_back( Circle{ 2.0 } );
   shapes.push_back( Square{ 1.5 } );
   shapes.push_back( Circle{ 4.2 } );

   // Drawing all shapes
   draw( shapes );
}
```

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	1
mpark::variant	3	9	9	9	9
Strategy	7	2	7	4	2
std::function	8	3	7	7	5
Type Erasure					

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	1
mpark::variant	3	9	9	9	9
Strategy	7	2	7	4	2
std::function	8	3	7	7	5
Type Erasure	9				

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	1
mpark::variant	3	9	9	9	9
Strategy	7	2	7	4	2
std::function	8	3	7	7	5
Type Erasure	9	4			

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	1
mpark::variant	3	9	9	9	9
Strategy	7	2	7	4	2
std::function	8	3	7	7	5
Type Erasure	9	4	8		

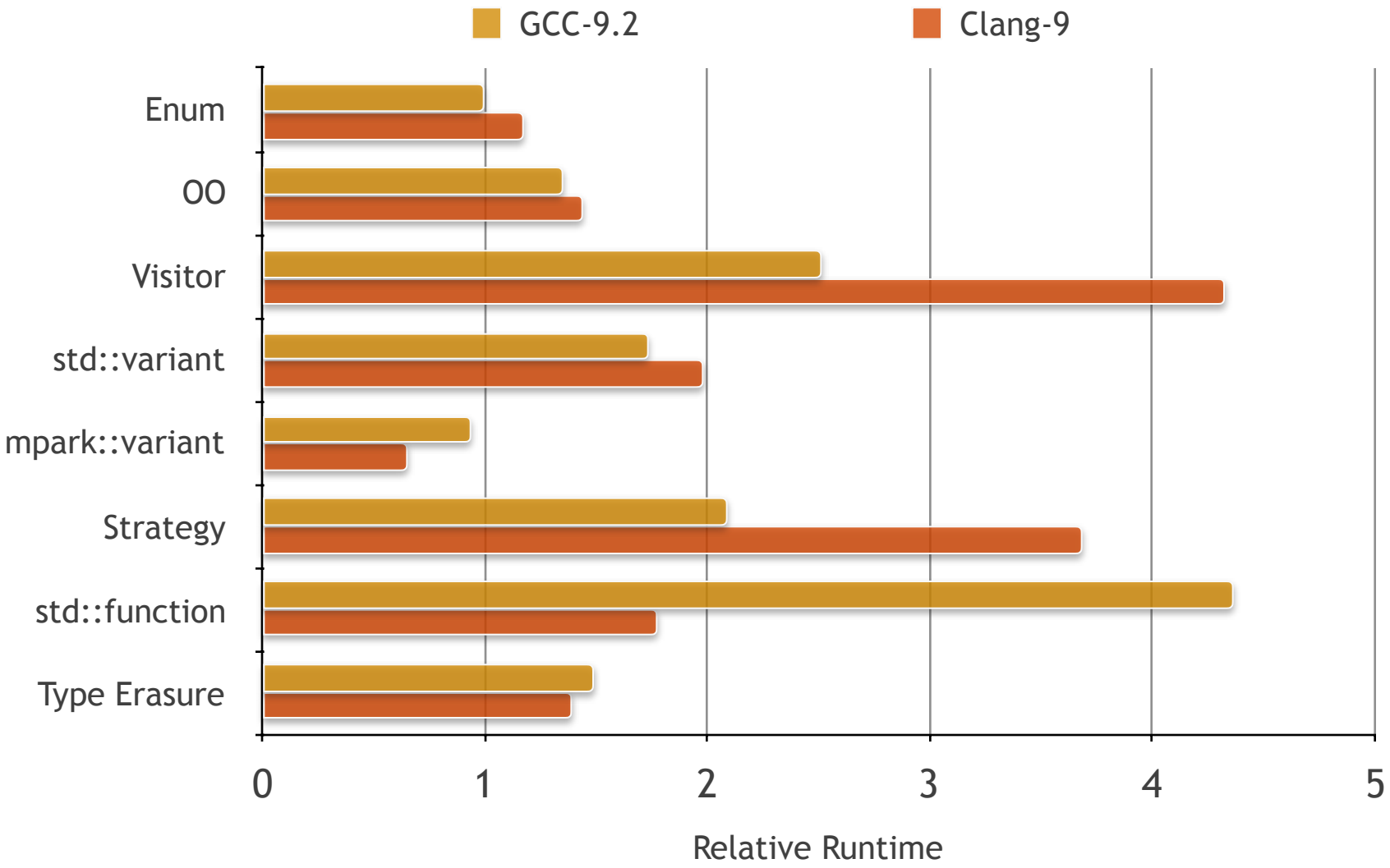
1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	1
mpark::variant	3	9	9	9	9
Strategy	7	2	7	4	2
std::function	8	3	7	7	5
Type Erasure	9	4	8	8	

1 = very bad, …, 9 = very good

Performance Results



MacBook Pro 13-inch, MacOS 10.14.6 (Mojave), Intel Core i7, 2,7Ghz, 16 GByte, Compilation flags: -std=c++17 -O3 -DNDEBUG

100 random shapes, 2.5M updates each, normalized to the enum solution

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	1
mpark::variant	3	9	9	9	9
Strategy	7	2	7	4	2
std::function	8	3	7	7	5
Type Erasure	9	4	8	8	6

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
Enum	1	7	5	6	9
OO	8	2	2	6	6
Visitor	2	8	8	3	1
mpark::variant	3	9	9	9	9
Strategy	7	2	7	4	2
std::function	8	3	7	7	5
Type Erasure	9	4	8	8	6

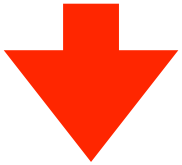
1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
mpark::variant	3	9	9	9	9
std::function	8	3	7	7	5
Type Erasure	9	4	8	8	6

1 = very bad, …, 9 = very good

Design Evaluation



	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
mpark::variant	3	9	9	9	9
std::function	8	3	7	7	5
Type Erasure	9	4	8	8	6

1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
mpark::variant	3	9	9	9	9
std::function	8	3	7	7	5
Type Erasure	9	4	8	8	6

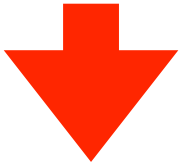
1 = very bad, …, 9 = very good

Design Evaluation

	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
mpark::variant	3	9	9	9	9
std::function	8	3	7	7	5
Type Erasure	9	4	8	8	6

1 = very bad, …, 9 = very good

Design Evaluation



	Addition of shapes (OCP)	Addition of operations (OCP)	Separation of Concerns (SRP)	Ease of Use	Performance
mpark::variant	3	9	9	9	9
std::function	8	3	7	7	5
Type Erasure	9	4	8	8	6

1 = very bad, …, 9 = very good

What is “Modern C++” design?

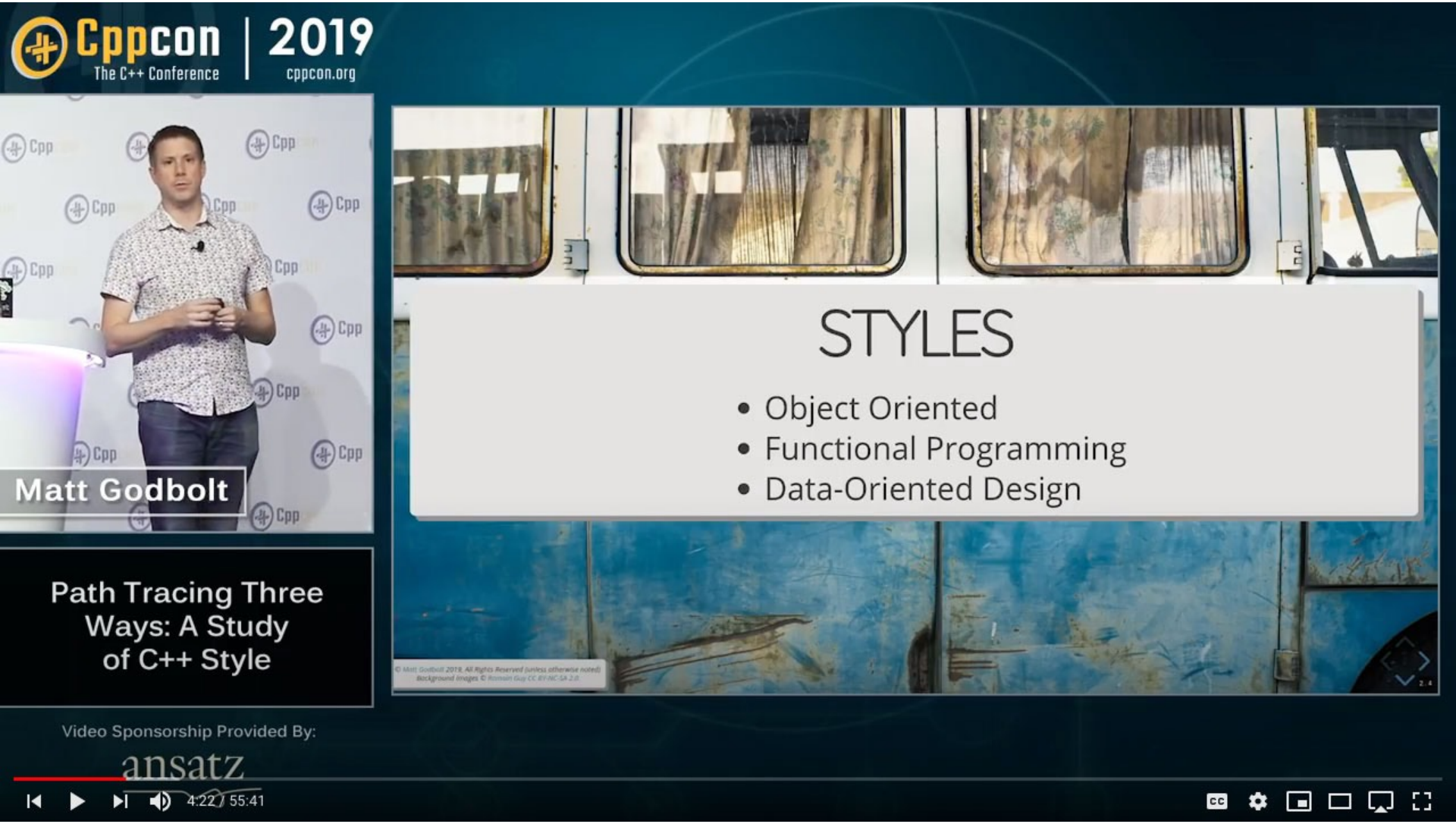
Procedural programming

Object-oriented programming

Functional programming

Generic programming

Multiparadigm programming


Cppcon
The C++ Conference
2019
cppcon.org
Matt Godbolt
Path Tracing Three Ways: A Study of C++ Style
STYLES
• Object Oriented
• Functional Programming
• Data-Oriented Design
Video Sponsorship Provided By:
ansatz
4:22 / 55:41

No-paradigm programming

Cppcon | 2019
The C++ Conference
cppcon.org

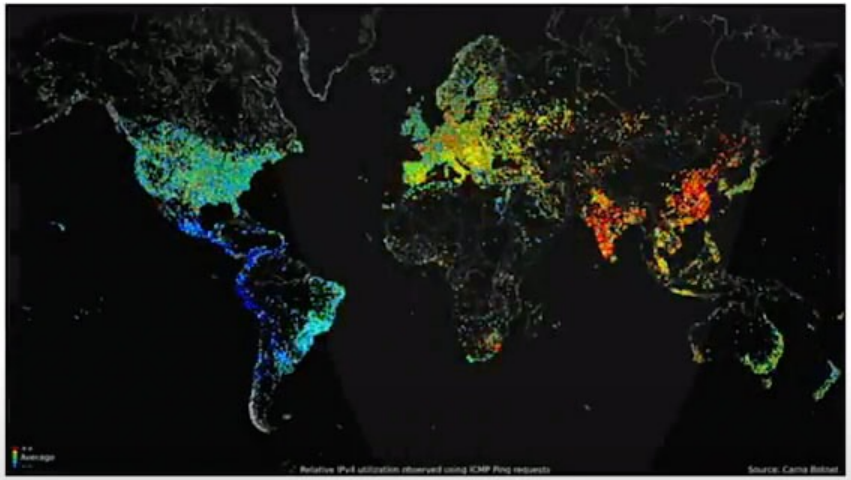
C++20:
C++ at 40

Video Sponsorship Provided By:
ansatz

1:56 / 1:31:25

C++20: C++ at 40

stability and evolution



Bjarne Stroustrup

Morgan Stanley, Columbia University

www.stroustrup.com

AURORA

CppCon 2019: Bjarne Stroustrup "C++20: C++ at 40"

126,954 views • Sep 17, 2019

2.2K 186 SHARE SAVE

Tell Them …

- Understand the virtues of “Modern C++”
  - Reduce the use of pointers and inheritance hierarchies
  - Prefer value semantics
  - Keep your code simple
- There is no “one-fits-all” solution
- Learn about the different programming paradigms
  - Learn about their advantages and weaknesses
  - Pick the good ideas
- Don’t shackle yourself with thinking about paradigms …

Embrace No-Paradigm Programming

Klaus Iglberger, London C++, April, 29th, 2020

klaus.iglberger@gmx.de