Объектно-ориентированное программирование

Object-oriented programming

XIV. Полиморфные типы данных

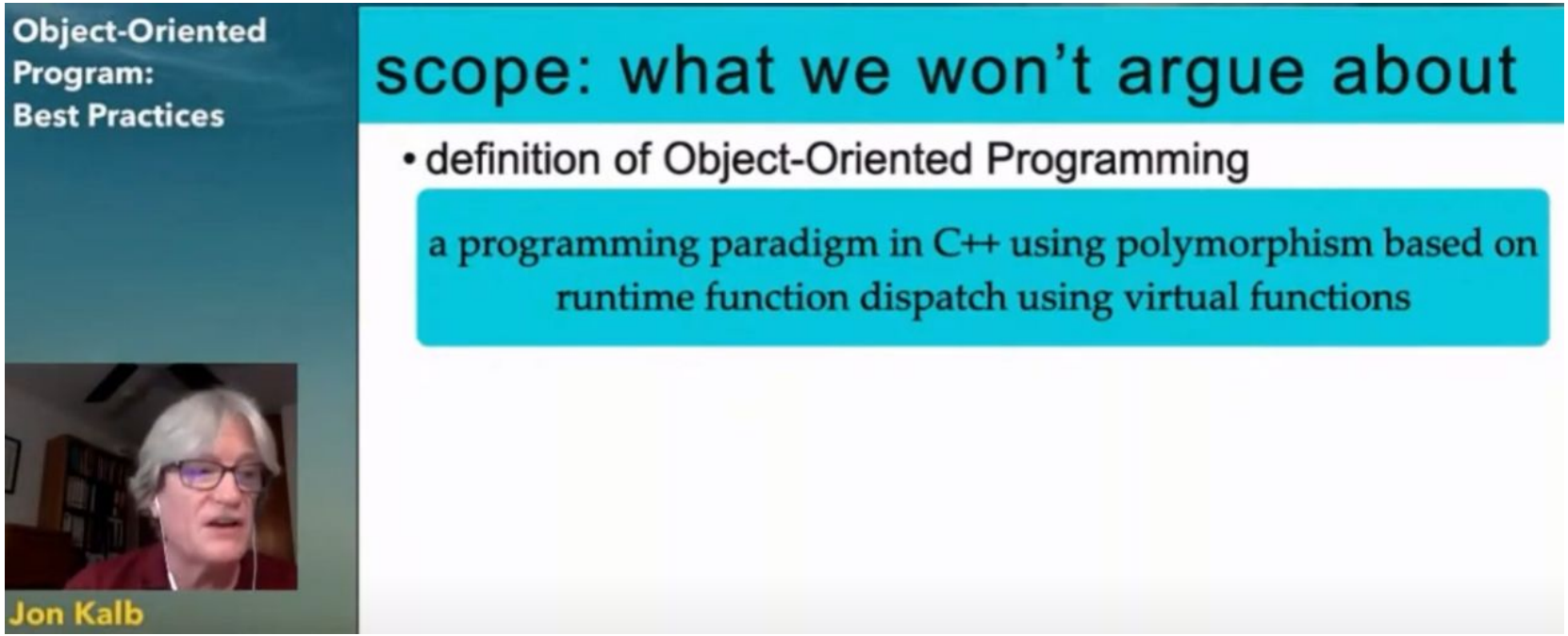
Discriminated unions

Что такое OOP?

- инкапсуляция (абстракция состояния и избавление от манипуляции состоянием в коде, **public** и **private**)
- ~~наследование~~
- ~~полиморфизм~~
- делегация? (**forwarding**, **move** semantics)
- ~~динамическое связывание~~
- абстракция (скрытая реализация, открытый **интерфейс**)
- ~~подмена типов~~
- обобщенное поведение (**generics**)

Что такое OOP? (2020)

https://youtu.be/c0lutJECNUA



"I strongly felt then, as I still do, that there is no one right way of writing every program, and a language designer has no business trying to *force* programmers to use a particular **style**. The language designer does, on the other hand, have an obligation to encourage and **support a variety of styles and practices** that have proven effective and to provide language features and tools to help programmers avoid the well-known traps and pitfalls."

B. Stroustrup

"A History of C++: 1979-1991"

A variety of styles

```
//  “structured” (basic loop)
    int c = 0;
    for (std::size_t i = 0; i < 3; ++i) c += a[i] * b[i];
```

```
//  “object-oriented” (iterator pattern)
    int c = std::inner_product(a.begin(), a.end(), b.begin(), 0);
```

```
//  “functional” (pipes-and-filters pattern)
    int c = std::ranges::fold_left_first(
        std::views::zip(a, b) | std::views::transform(
            [](auto t) { return std::get<0>(t) * std::get<1>(t); }),
        std::plus<>());
```

https://godbolt.org/z/GEh9sTMGn

Является ли C++ OO-языком?

"Any sufficiently complicated **C** or **Fortran** program contains an ad hoc, informally-specified, bug-ridden, slow implementation of half of Common **Lisp**."

Philip Greenspun (1993)

"Building new semantics on top of a language that lacks them is often called **greenspunning** after this rule[...]. Let's take **JavaScript** as an example. It looks kind-of **OO if you tilt your head sideways**. But, are you really doing "**messaging**" if you don't have something like a method_missing handler that lets an **object decide extremely late** whether and how it responds to a message?"

Reginald Braithwaite (2013)

https://braythwayt.com/2013/12/22/wrong.html

XIV.a. Гетерогенные типы данных в C++

std::variant

std::variant

Defined in header <variant>

```
template< class... Types >
class variant;
```
(since C++17)

The class template std::variant represents a type-safe union.

An instance of variant at any given time either holds a value of one of its alternative types, or in the case of error - no value (this state is hard to achieve, see valueless_by_exception).

As with unions, if a variant holds a value of some object type T, the T object is nested within the variant object.

A variant is not permitted to hold references, arrays, or the type void.

A variant is permitted to hold the same type more than once, and to hold differently cv-qualified versions of the same type.

Consistent with the behavior of unions during aggregate initialization, a default-constructed variant holds a value of its first alternative, unless that alternative is not default-constructible (in which case the variant is not default-constructible either). The helper class std::monostate can be used to make such variants default-constructible.

https://en.cppreference.com/w/cpp/utility/variant.html

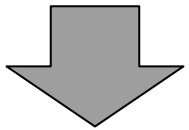
std::variant

```
#include <variant>

int main() {
    std::variant<int, long, std::string> var;
    var.index();                    // 0
    std::get<0>(var);               // 0
    std::get<int>(var);             // 0
    var = “hello”;
    var.index();                    // 2
    std::get<2>(var);               // “hello”
    std::get<std::string>(var);     // “hello”
    var = 42;                       // var.index() == 0
    var = 77L;                      // var.index() == 1
}
```

Полиморфизм в ML

```
fun factorial n =
    if n <= 1 then 1
    else factorial (n-1) * n
```



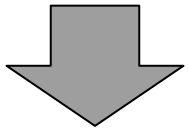
```
val factorial = fn : int -> int
```

https://en.wikipedia.org/wiki/Hindley%E2%80%93Milner_type_system

Полиморфизм в ML

```
fun filter pred []          = []
   | filter pred (a::rest) =
   if pred a
   then a::(filter pred rest)
   else (filter pred rest)

filter (fn x => x <> "text") ["text", "notext"]
```

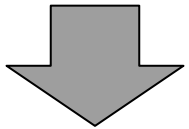


```
val filter = fn : ('a -> bool) -> 'a list -> 'a list
```

Полиморфизм в ML

```
fun map (f, xs) =
    case xs of
        []       => []
      | x::xs' => (f x)::(map(f, xs'))

map (fn x => x + 1) [4, 8, 12, 16]
```



```
val map = fn : ('a -> 'b) * 'a list -> 'b list
```

Discriminated unions

"The programmer will usually at some stage wish to determine to which of the possible **subclasses** the **record** currently referenced by that variable actually belongs. This can be achieved by a construction known as a **record class discriminator.**"

C. A. R. "Tony" Hoare

"Record Handling" (1966)

Возникновение понятия “класс”?

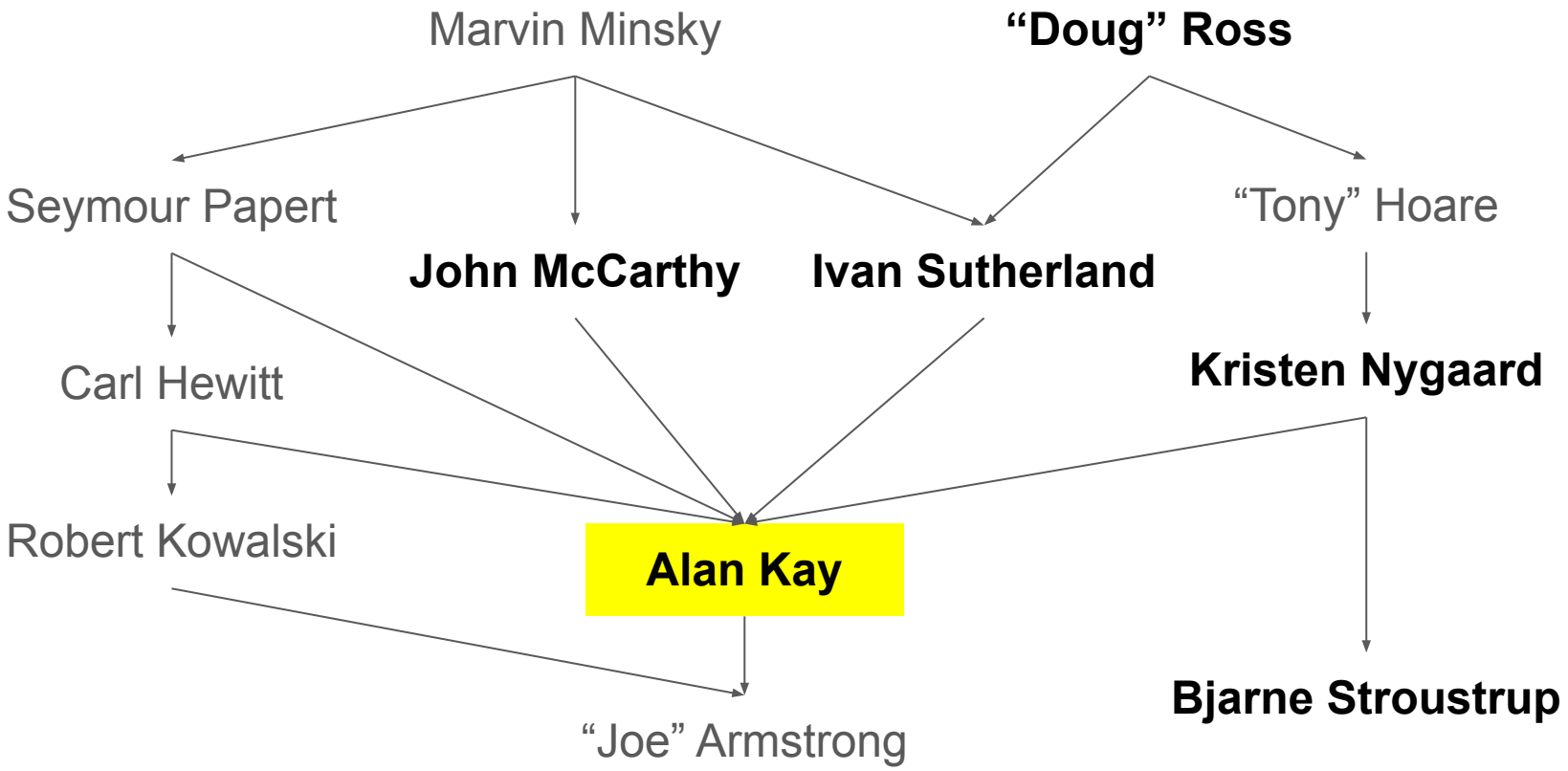
```
record class expression (
    subclasses
        constant (real value);
        variable (string printname);
        pair ...
);

consider e when constant then ...
           when variable then ...
           when pair     then ...
```

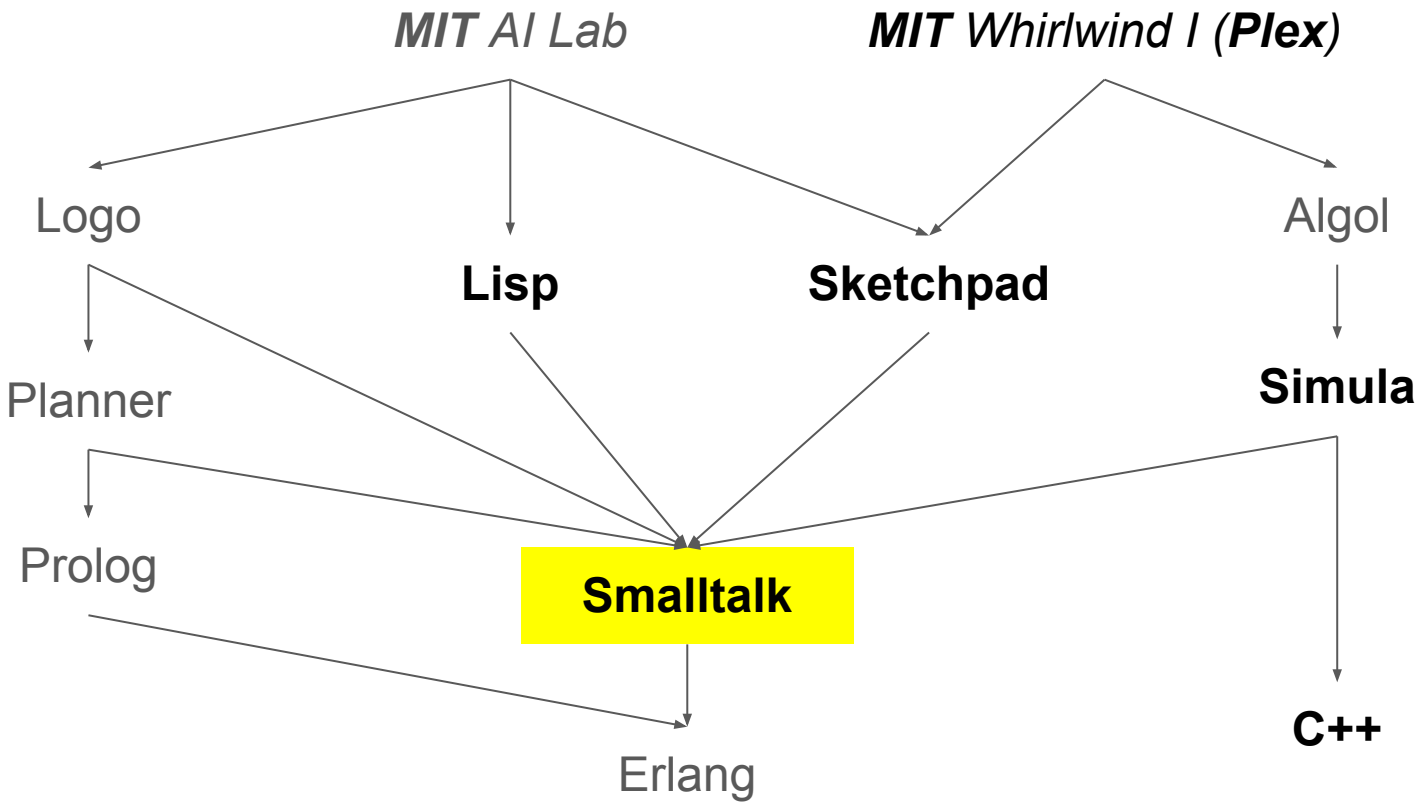
XIV.b. Что такое ООП?

The map of object-oriented programming

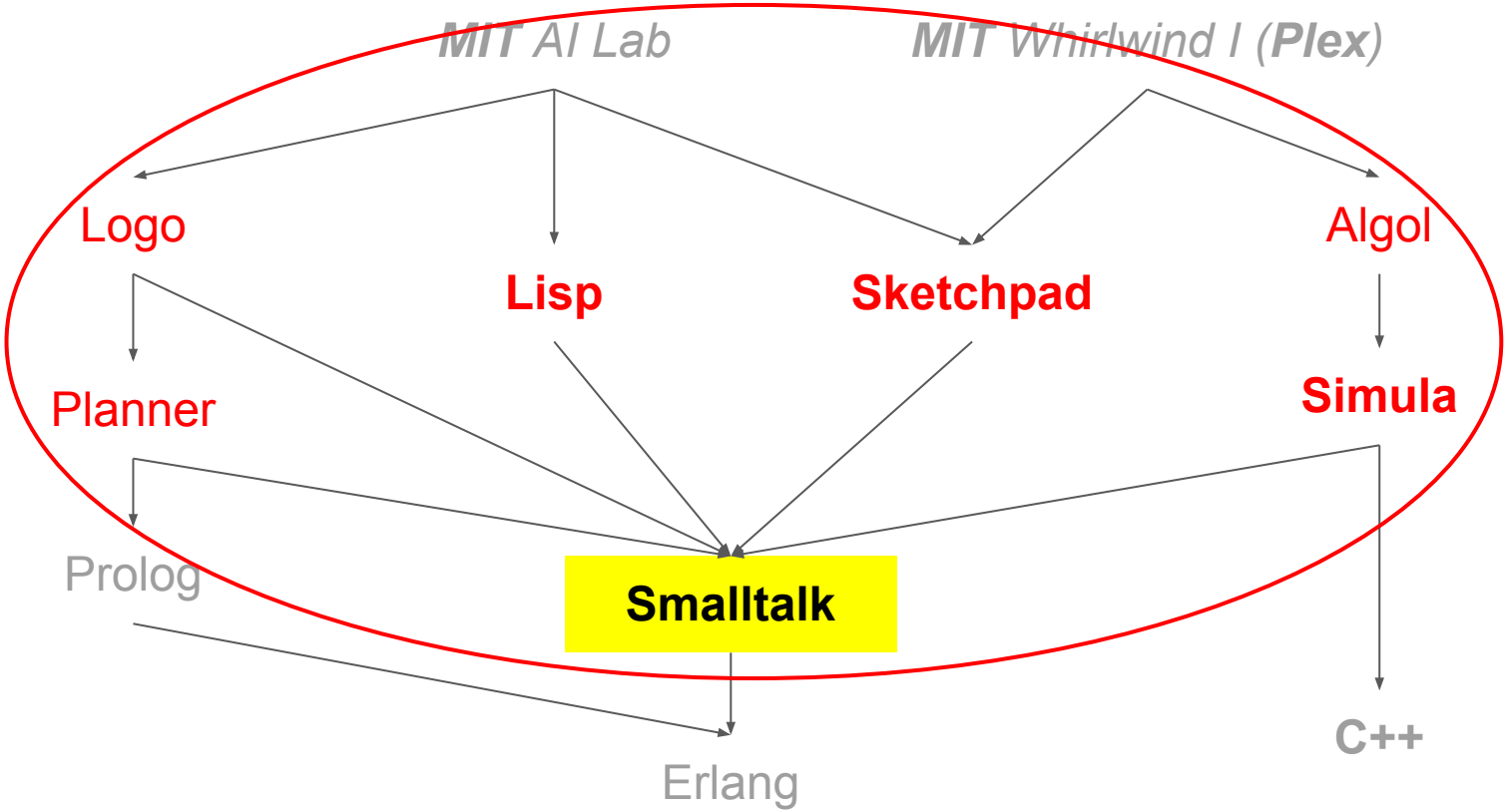
Карта объектно-ориентированного программирования



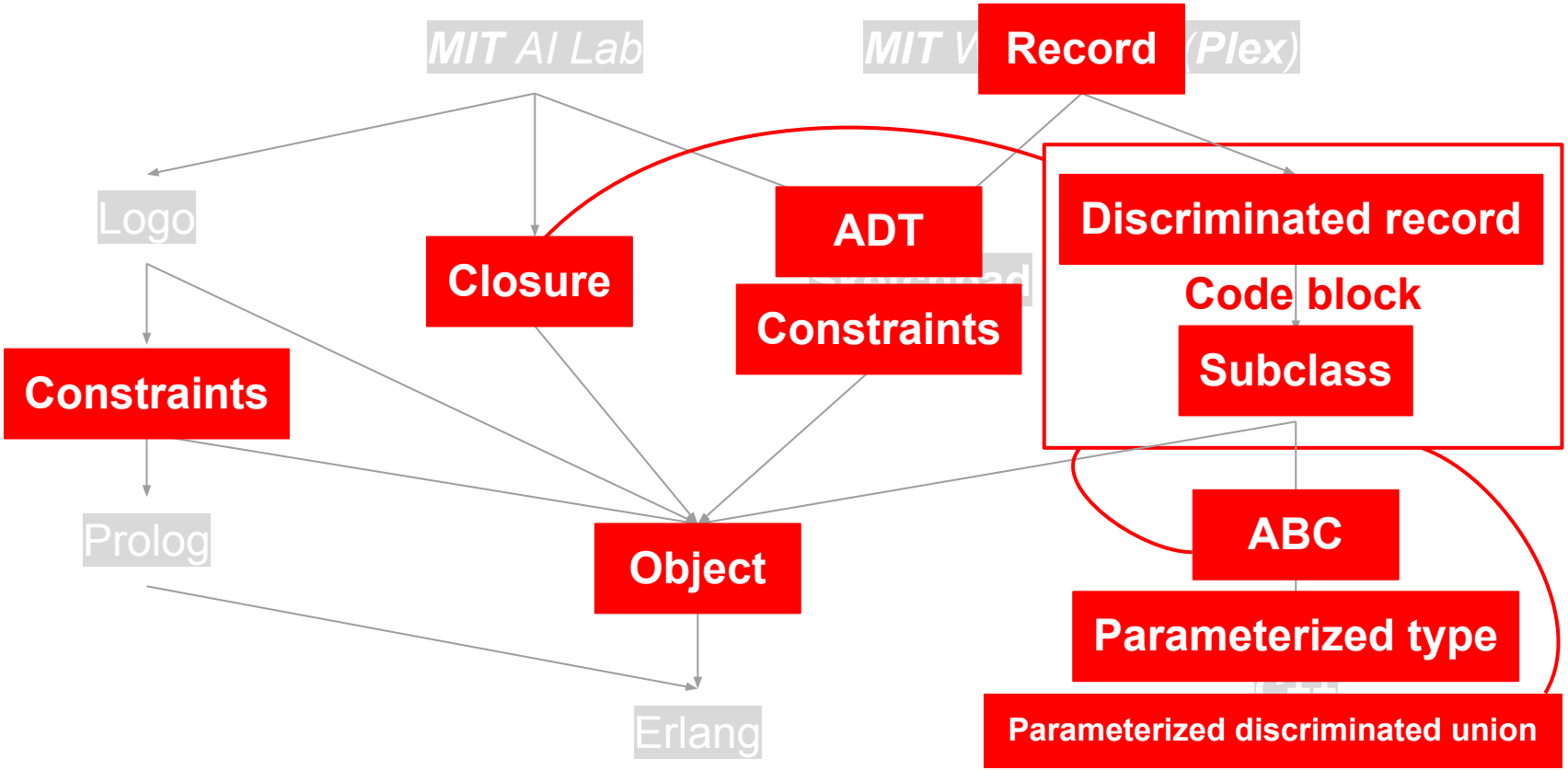
Карта объектно-ориентированного программирования



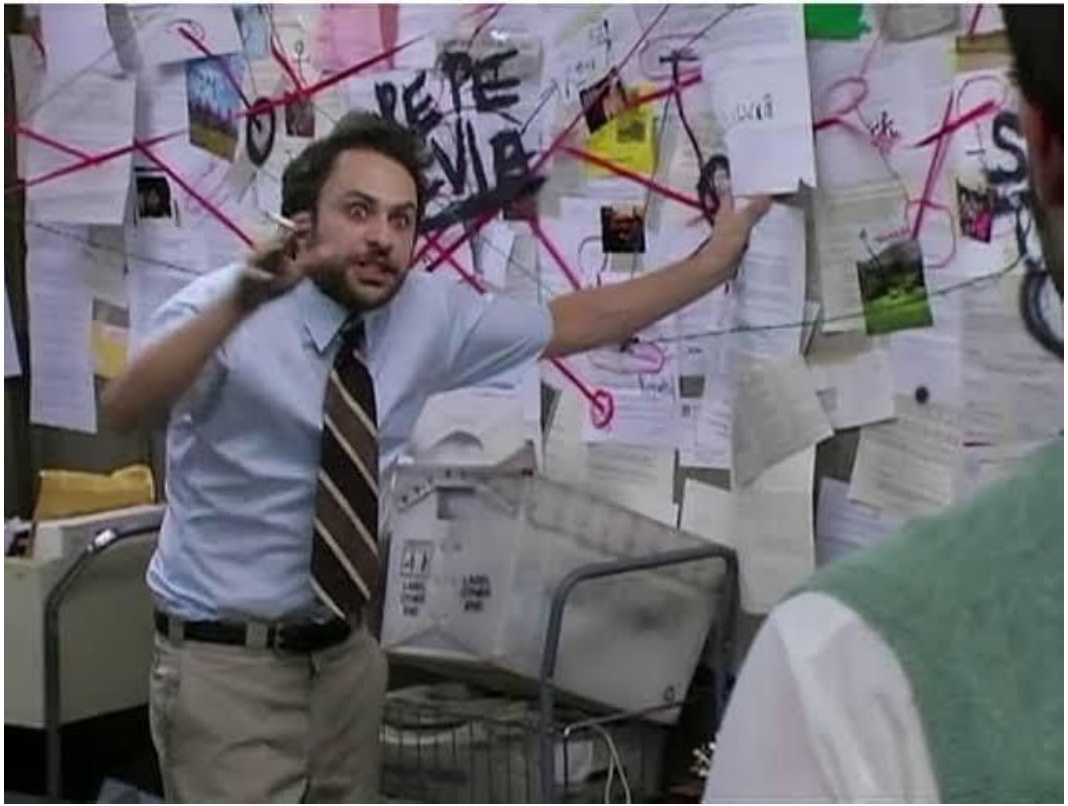
Круг замкнулся



Круг замкнулся



Карта объектно-ориентированного программирования



Tagged union

```
enum tag { Circle, Line, ... };

union types {
    Circle circle;
    Line line;
    ...
};

struct GeoObj {
    tag tag_;
    types type_;
};
```

```
void draw(const GeoObj& var) {
    switch(var.tag_) {
    case Circle:
        var.type_.circle.draw();
        break;
    case Line:
        var.type_.line.draw();
        break;
    case ...:
    ...
    }
}
```

ABC

```
struct GeoObj {
    virtual void draw() const = 0;
};

class Circle : public GeoObj {
public:
    virtual void draw() const override;
};

class Line : public GeoObj {
public:
    virtual void draw() const override;
};
```

```
void draw(const GeoObj& var) {
    var.draw();
}
```

https://github.com/AbsoluteVirtue/fcim_poo_21.6/blob/master/pr08.cpp

Template

```
template <class GeoObj>
void draw(const GeoObj& var) {
    var.draw();
}
```

std::variant

```
#include <variant>

using GeoObj =
    std::variant<Circle, Line, ...>;
```

```
void draw(const GeoObj& var) {
    switch(var.index()) {
    case 0:
        std::get<0>(var).draw();
        break;
    case 1:
        std::get<1>(var).draw();
        break;
    case ...:
    ...
    }
}
```

“Visitor” pattern

```
#include <variant>

using GeoObj =
    std::variant<Circle, Line, …>;

struct Visitor {
     void operator()(const Circle& c)
     const {
          c.draw();
     }
     void operator()(const Line& l)
     const {
          l.draw();
     }
};
```

```
void draw(const GeoObj& var) {
     std::visit(Visitor(), var);
}
```

std::variant w/ “visitor” pattern

```
#include <variant>

using GeoObj =
   std::variant<Circle, Line, …>;
```

```
void draw(const GeoObj& var) {
    std::visit(
        [](const auto& x){ x.draw(); };
        , var);
}
```

Tagged union - ABC - Template - std::variant

Virtual functions (dynamic polymorphism w/ inheritance)

- open heterogeneous collections
- tight coupling
- run-time performance-heavy
- all types have to provide the entire interface

Templates (static polymorphism w/ meta-programming/function overloading)

- homogeneous collections
- compile-time performance-heavy
- all types only have to provide what's being used

Tagged unions (dynamic polymorphism ~~w/o inheritance~~)

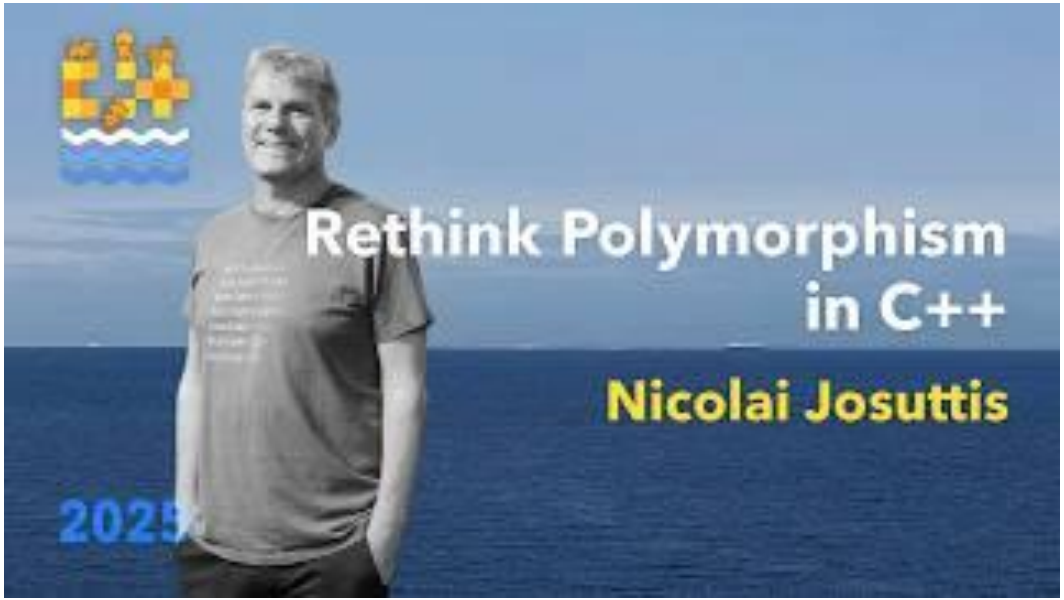
- closed heterogeneous collections
- memory-heavy
- copy-by-value performance-heavy

Объединения или наследование?

https://youtu.be/wo84LFzx5nI

Что лучше?

https://youtu.be/zI0DOKN6zr0



https://godbolt.org/z/qYdz8jbes

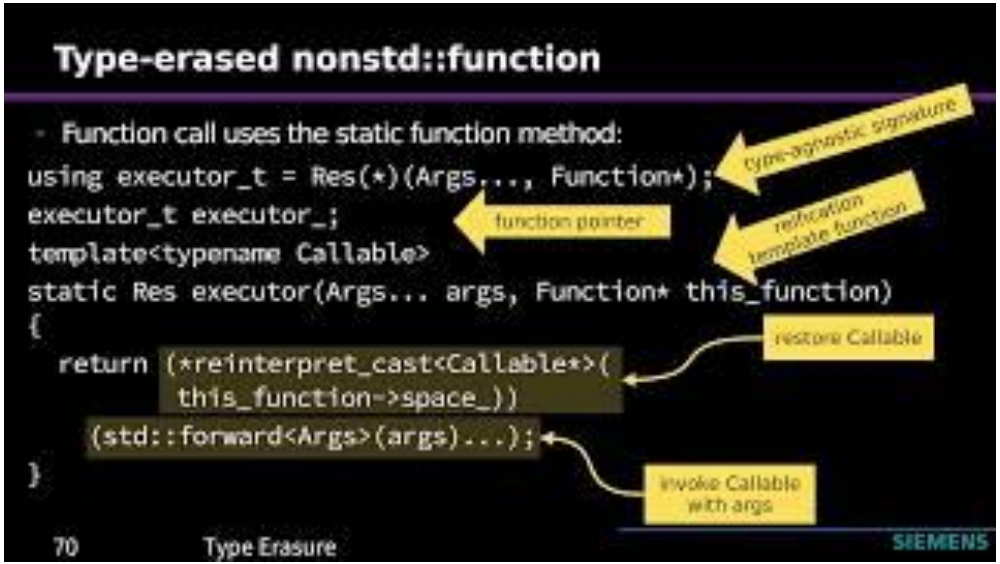
Что лучше?

	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

https://www.youtube.com/watch?v=fwXaRH5ffJM

Type erasure - runtime polymorphism w/o inheritance

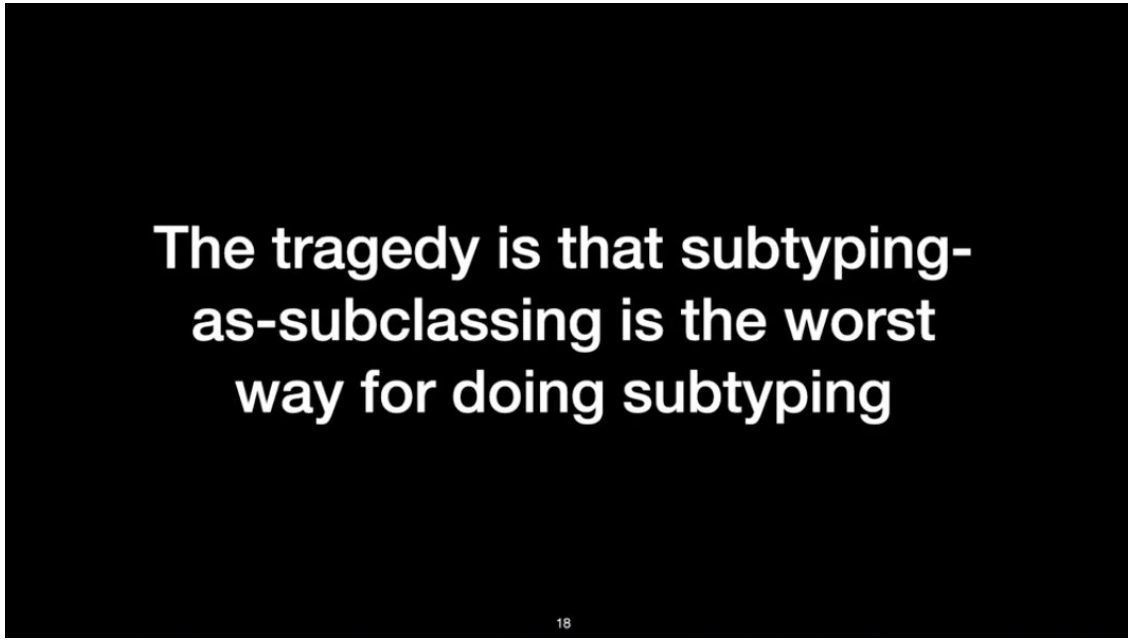
https://www.youtube.com/watch?v=p-qaf6OS_f4



https://quuxplusone.github.io/blog/2019/03/18/what-is-type-erasure/

Полиморфизм подтипов

https://www.youtube.com/watch?v=uM72qP5Wh18



Team Fat Struct

https://youtu.be/wo84LFzx5nI

https://hero.handmade.network/forums/code-discussion/t/7896-why_don%2527t_use_discriminated_union_rather_than_sparse_system_for_entity_system#24627

Да эти скуфы просто не хотят учиться новым вещам!

https://youtu.be/wo84LFzx5nI

Что такое OOP?

- **инкапсуляция** (абстракция состояния и избавление от манипуляции состоянием в коде, ~~**public и private**~~)
- **динамическое связывание** (**дискриминированные** объединения)
- **абстракция** (скрытая реализация, открытый **интерфейс**)
- **обобщенное** поведение (**generics**)

leon.brinzan@iis.utm.md

Что такое OOP?

“OOP to me means only **messaging**, **local retention** and **protection** and **hiding of state-process**, and extreme **late-binding** of all things.”

“The core of what I now have to call "**real OOP**" – namely *encapsulated modules all the way down with pure messaging* – still hangs in there strongly because **it is nothing more than an abstract view of complex systems**.”

A. C. Kay

Инкапсуляция на практике

https://youtu.be/wo84LFzx5nI

Инкапсуляция на практике

https://youtu.be/aBYqGD41E9g

https://youtu.be/wo84LFzx5nI

Что такое ООР?

ООП – это идея **Lisp**, знающими людьми доведенная до абсолюта, позже подобранная посторонними людьми и доведенная до абсурда.

Я *(2025)*

Что теперь делать?

https://www.youtube.com/watch?v=GKYCA3UsmrU