TurtleSec

# The Floor is Lava

Patricia Aas

Meeting C++ 2022

@pati_gallardo

# Trying to teach C++

## with Corentin Jabot

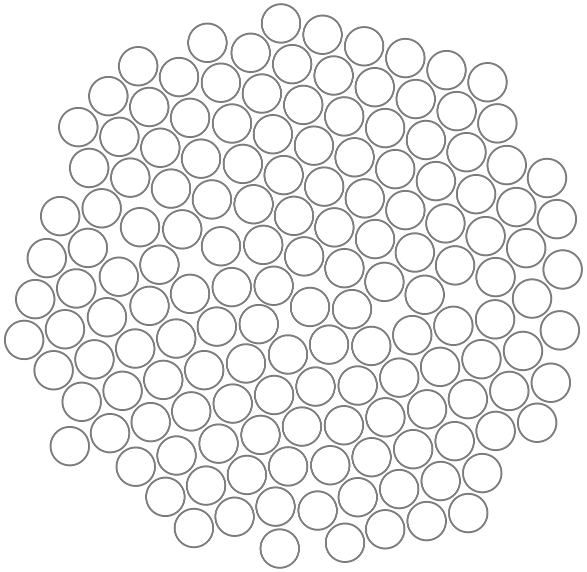
Mod(C++) Fundamentals

Mod(C++) Intermediate

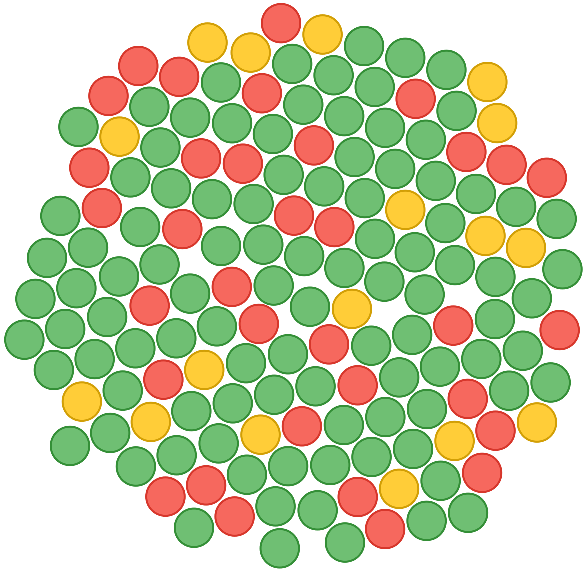
How do you teach a programming language

# when the Floor is Lava?

# Do we even know what is Lava?



# We could try to classify them



Have we really gotten much further?

Maybe the Lava is just a pool of anti-patterns?

## Stuff You Shouldn't Do

And we can just skip it?

Maybe it's only for

*advanced specialized*

use-cases?

Who knows what ***std::launder*** does anyway?

Or what ***Subsumption*** is?

# Who needs to know?

Maybe there is a

"modern" subset of C++

that is "safe"?

# C++ is vast

Teaching an introduction to C++, requires us to

select a subset of the language

which will be the basis of their future learning

They will be someone's colleague in the future

# They don't know any C++

They need to learn enough to be productive

We can try to draw some lines in the sand, and define

# a Minimal Viable Subset of C++

# How though?

To select such a subset we have to have

some criteria

# Criteria

Should be ***mostly harmless***,

sharp edges should be avoided if possible

Has to be ***useful***,

should give them the skills they need

Be relevant to ***their current level***

Like all programming languages C++ has a culture

But there is

# no definite list

of what C++'s sharp edges are

We could try to suss out

a sense of what the community feels…

And of course we do this in the traditional way

using Twitter Polls

So the data in this talk is massaged data

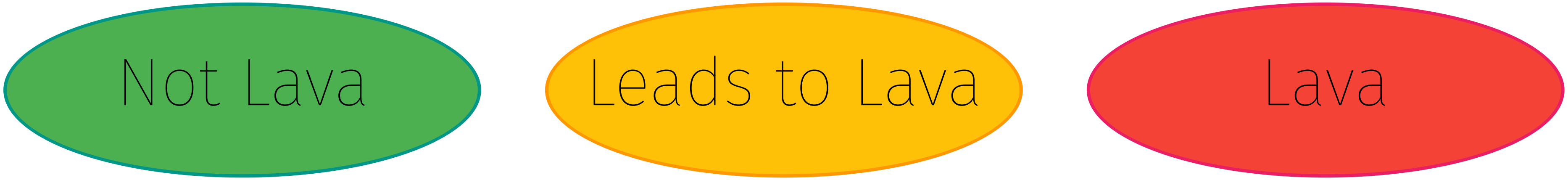
from Twitter Polls

for science

Because we have data, it must be visualized.

## Hence this web based talk

# Lava (Harmlessness)

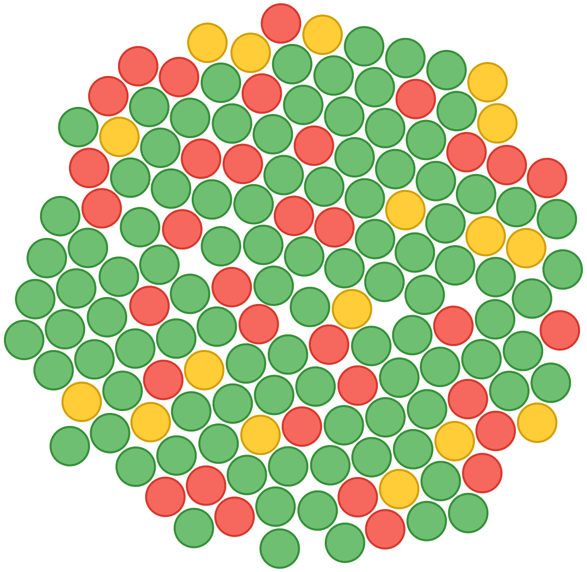


Not Lava

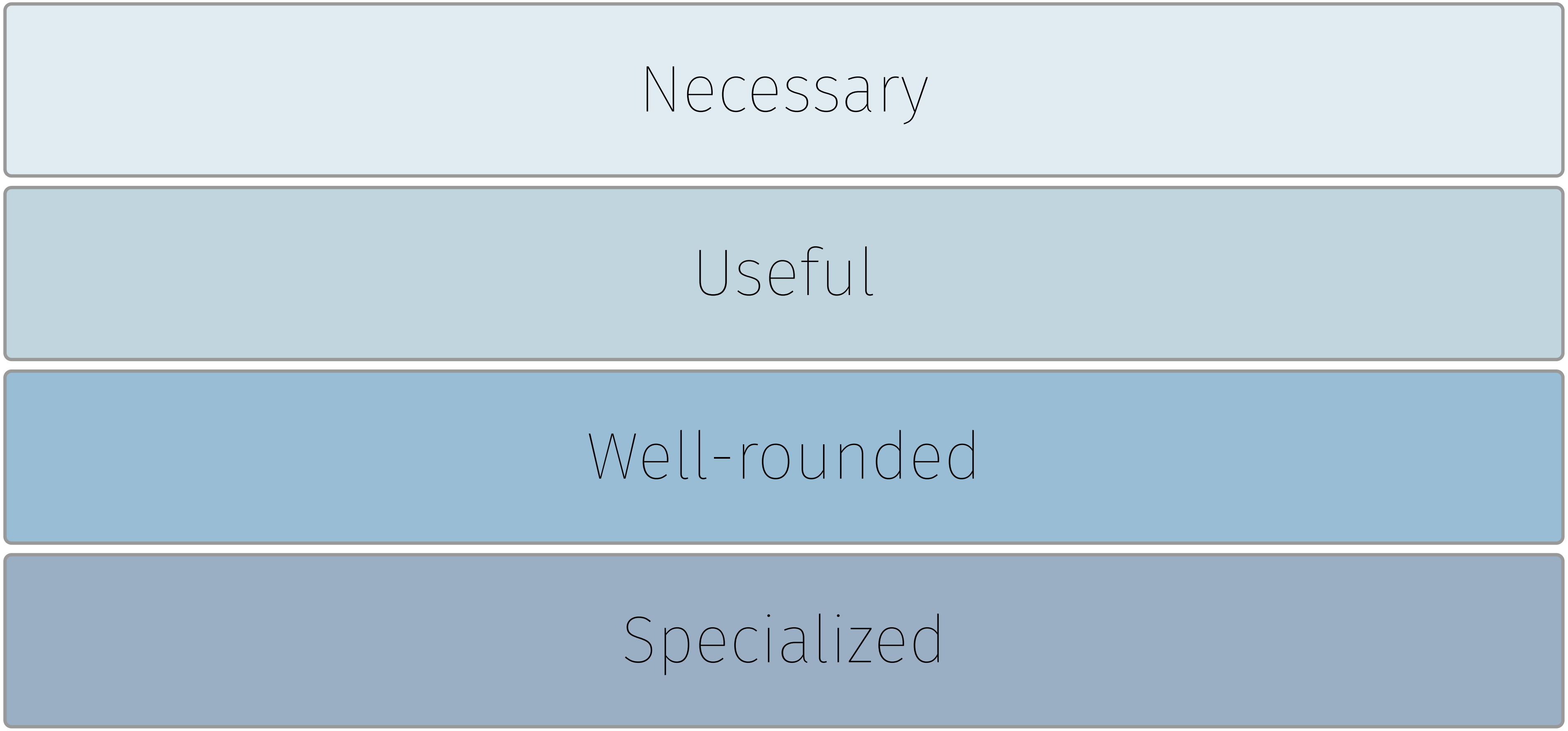
Leads to Lava

Lava

# Yes, let's figure out what is Lava



# but also how useful something is



Necessary

Useful

Well-rounded

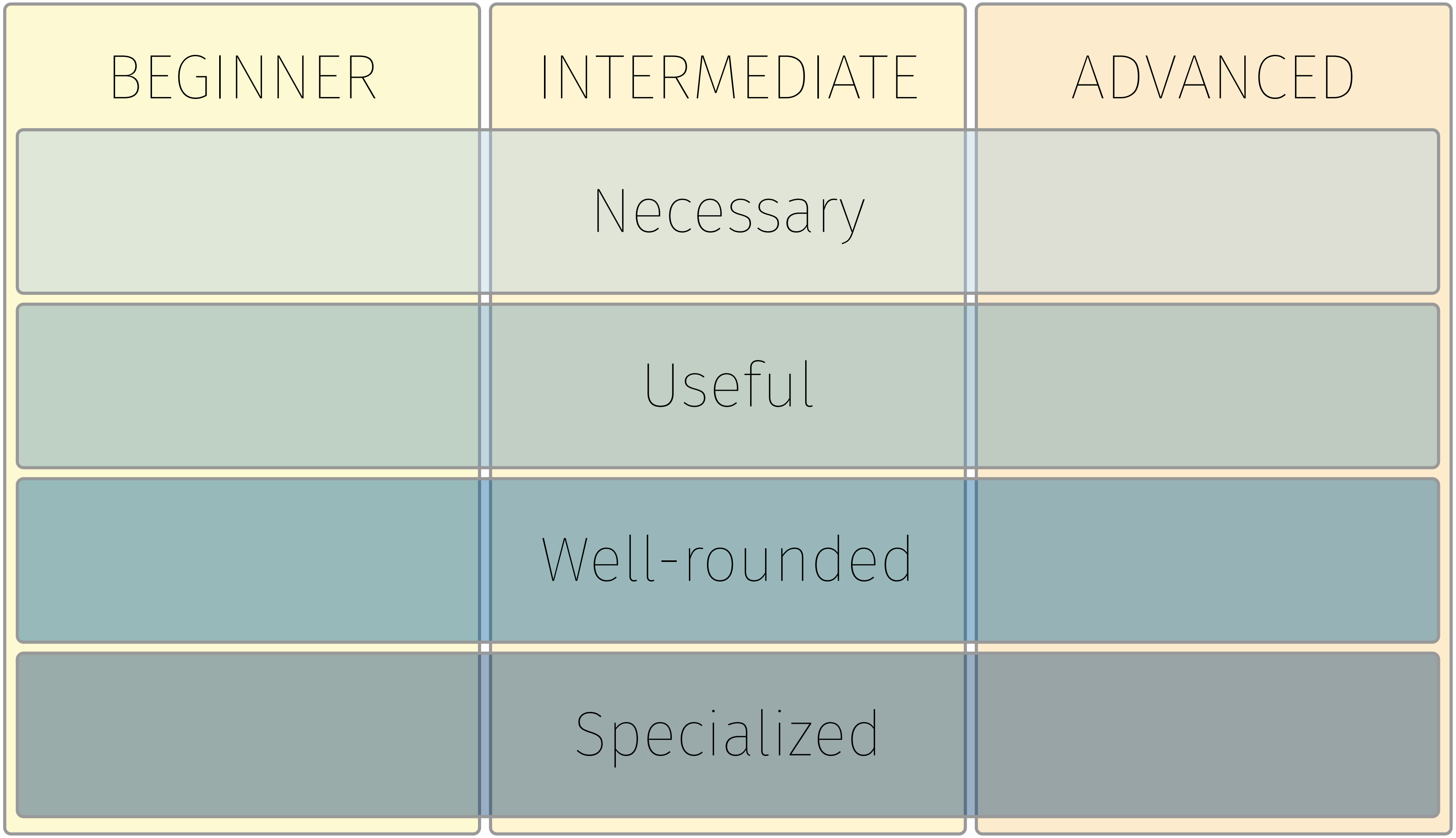
Specialized

# and at what level this is expected

| BEGINNER | INTERMEDIATE | ADVANCED |
|---|---|---|

# This gives us 3 axis on which to classify

TurtleSec



BEGINNER

INTERMEDIATE

ADVANCED

Necessary

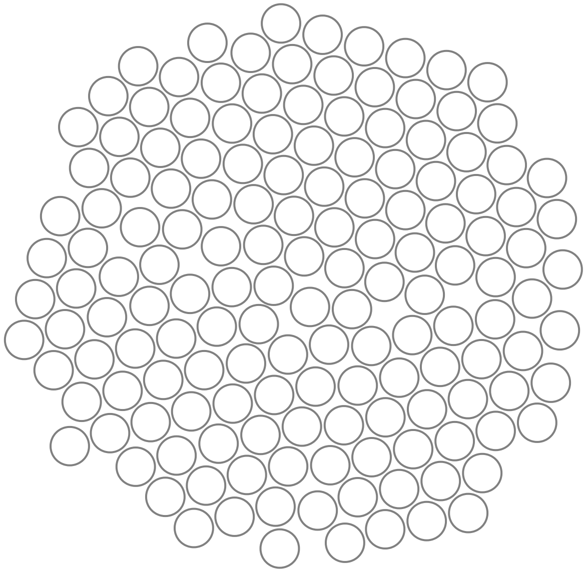
Useful

Well-rounded

Specialized

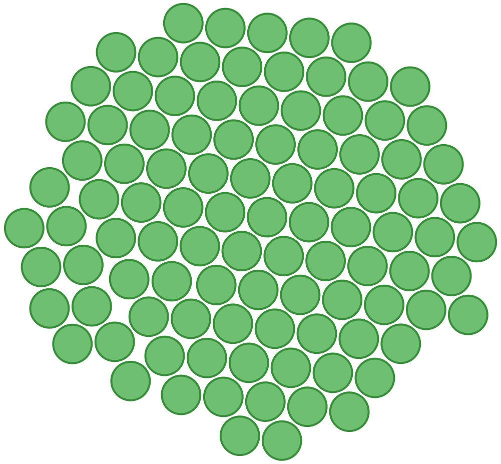
TurtleSec

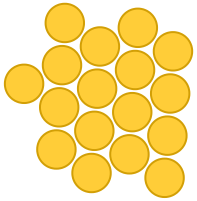
# So we can take some C++ features

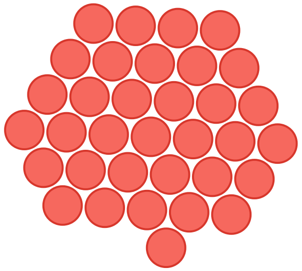


@pati_gallardo

# Use color for Lava

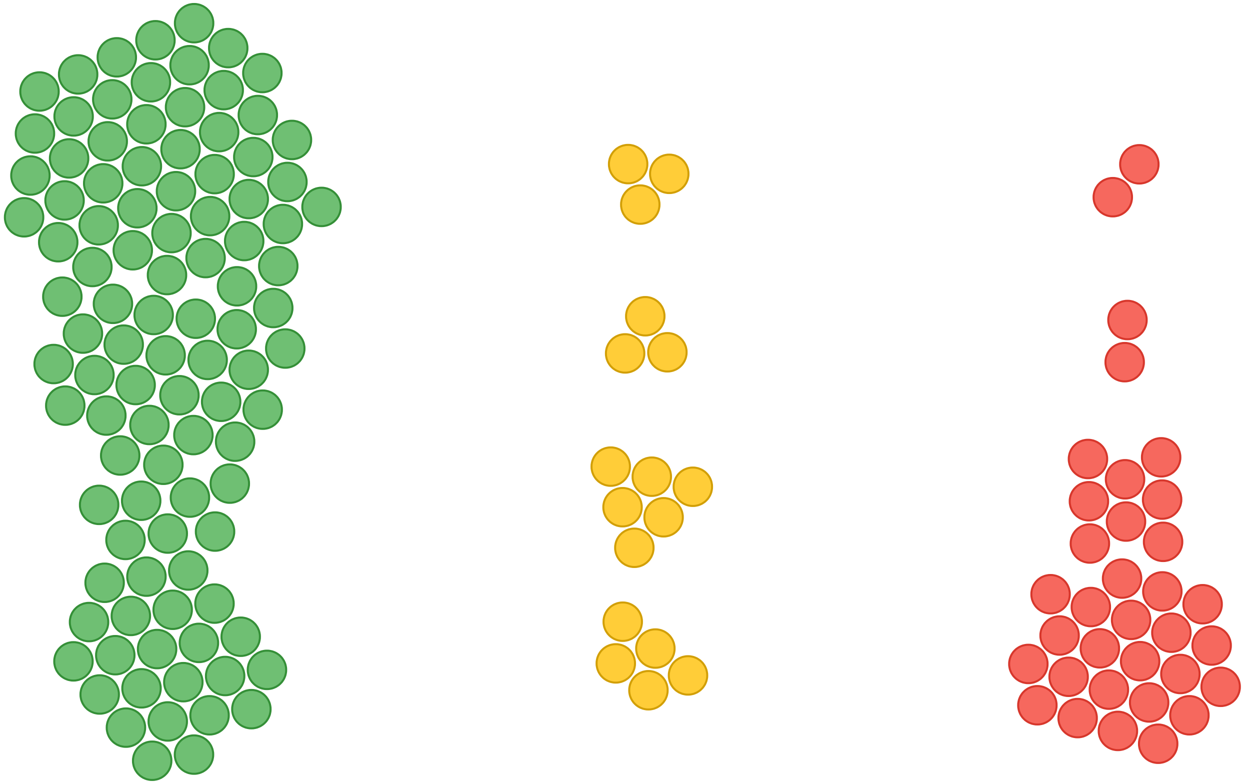






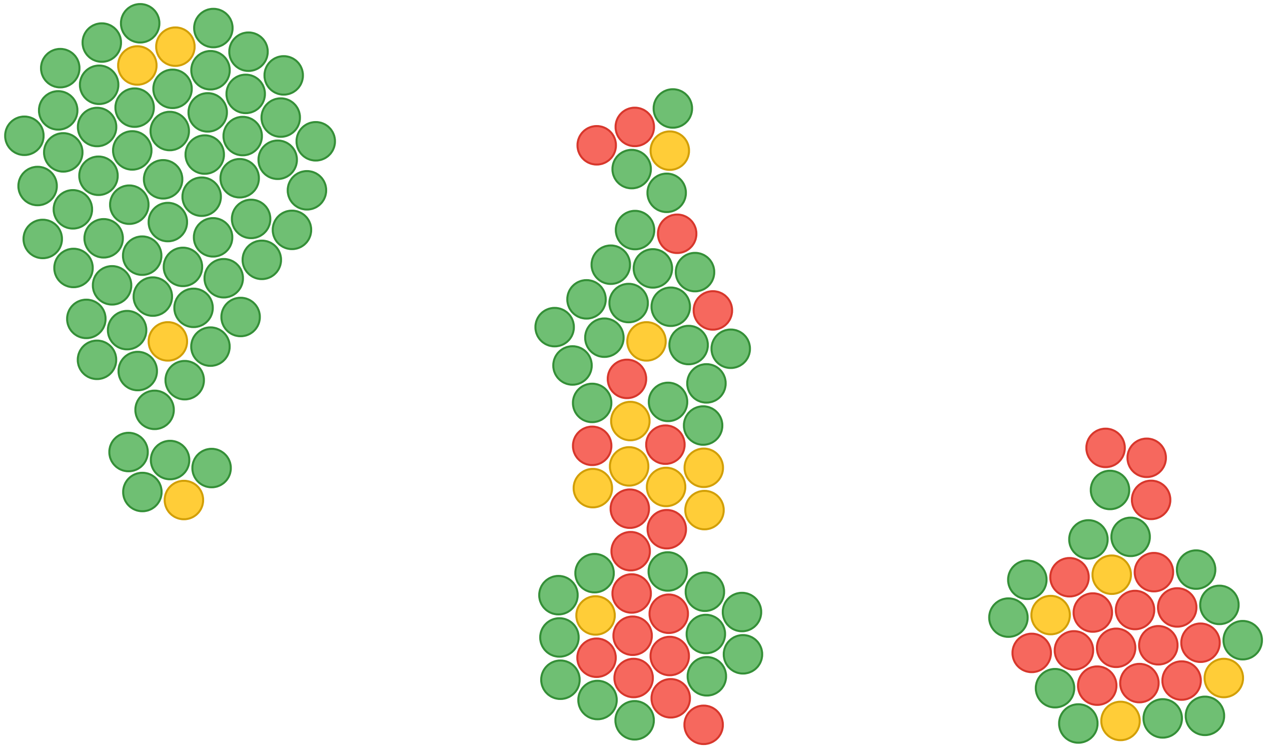
TurtleSec

# Rank by utility



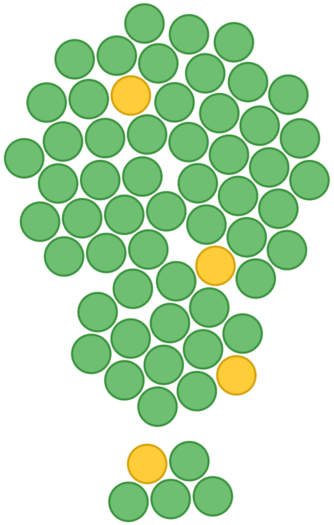
@pati_gallardo

# And group by level



TurtleSec

# What is beginner?



Interestingly, beginner features in C++
seem to be mostly ***Not Lava***,
and mostly ***Necessary*** and ***Useful***

@pati_gallardo

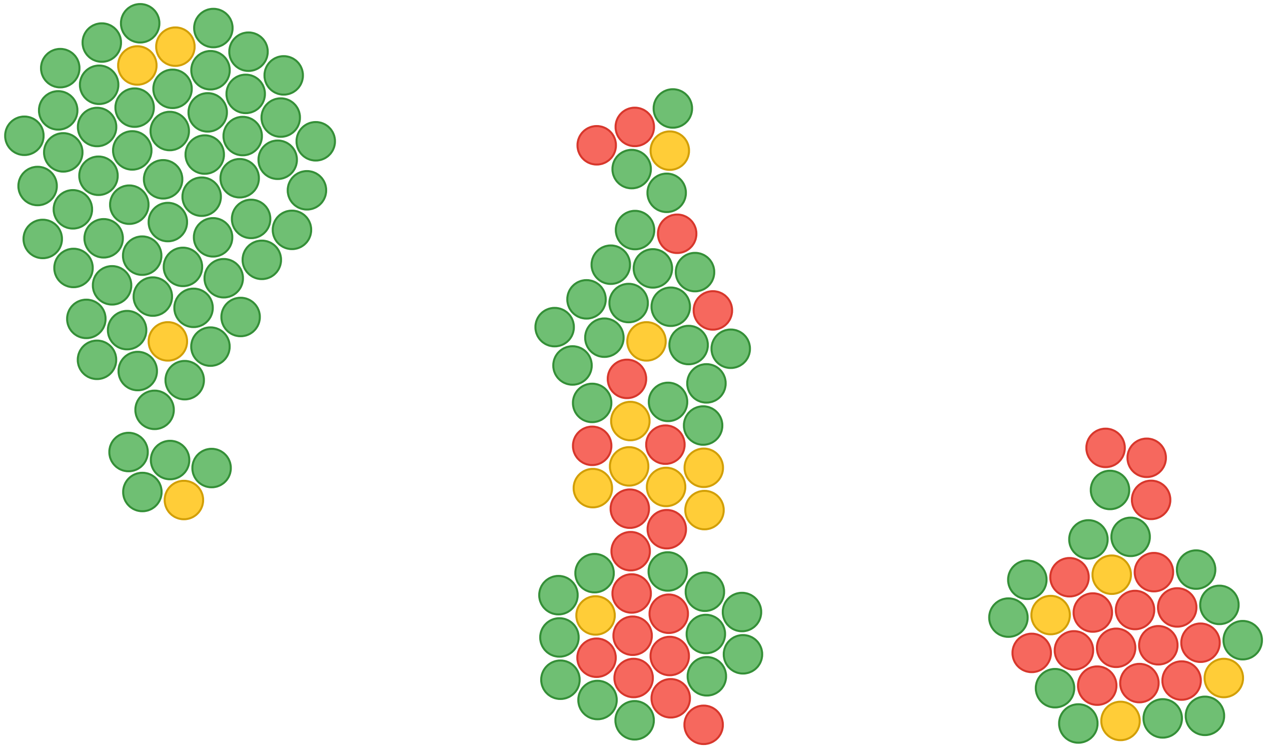
# What is beginner?

- iostream
- Pass parameter by reference
- #pragma once
- ASan (Address Sanitizer)
- auto
- C-style for loops
- clang-format
- clang-tidy
- class
- const
- Debuggers
- enum class
- equals default
- Heap allocation
- Includes
- inline
- Iterators
- Lambda capture
- Lambdas
- MSan (Memory Sanitizer)
- namespaces
- Object Oriented Programming (OOP)
- Package Manager (Conan/Vcpkg)
- Pass parameter by const reference
- Pass parameter by value
- References
- Return by value
- Short-Circuit Boolean Expressions (|| &&)
- Signed integer
- Signed Integer addition/multiplication
- Stack allocation
- std::array
- std::optional
- std::size
- std::string
- std::unique_ptr
- std::vector
- STL algorithms
- String literals (no suffix)
- struct
- UBSan (Undefined Behavior Sanitizer)
- Values
- Warnings

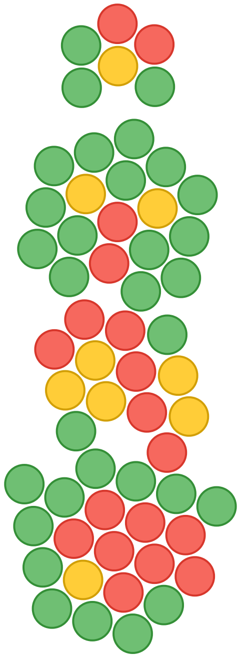
- Exceptions
- [[maybe_unused]]
- {fmt}
- constexpr
- Digits separator
- import
- nullptr_t
- Overloading
- static_cast
- std::from_chars
- std::string_view
- std::tuple
- std::variant

- std::initializer_list
- bitwise operations
- Dynamic polymorphism
- Forward declarations
- String literals with std::string suffix

# Group by level



# What is intermediate? TurtleSec



Intermediate features in C++ are a mixed bag,
they seem to be filling out in the areas of ***Leads to Lava*** and ***Lava***,
and you already see a lean towards ***Specialized***

# What is intermediate?

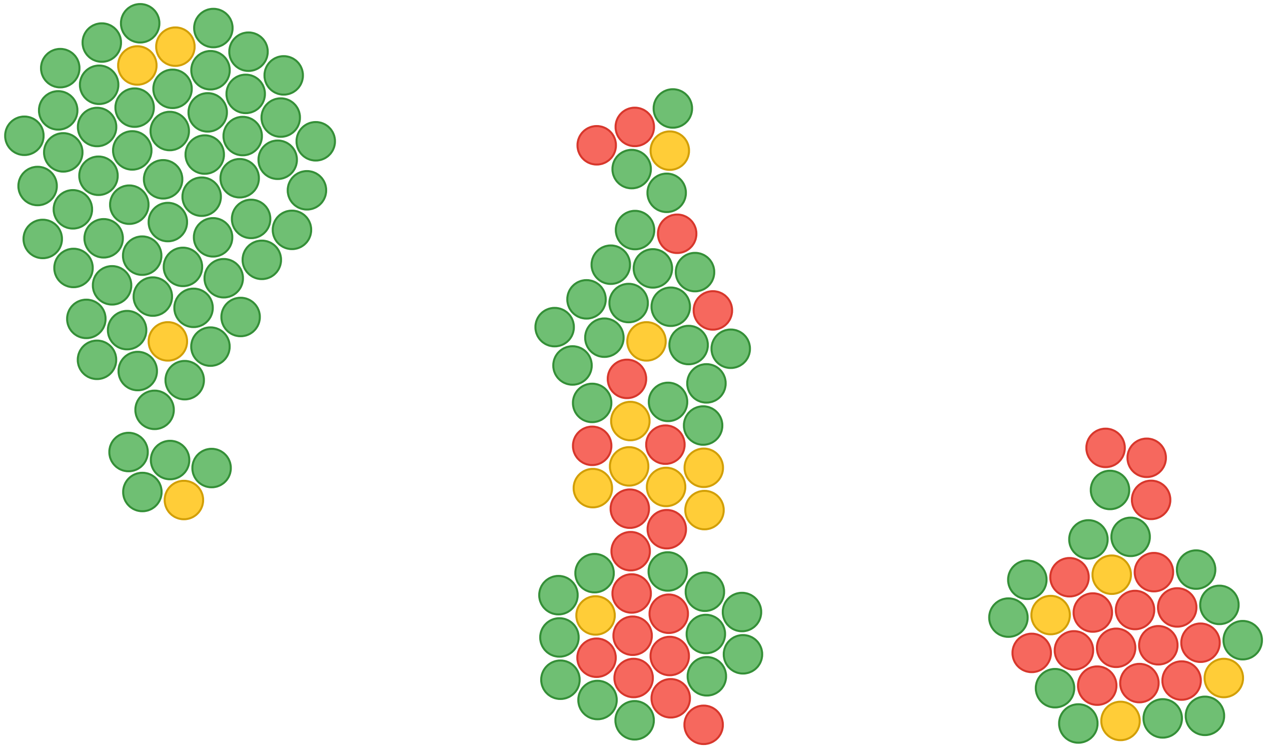
- Globals
- Undefined Behavior
- new/delete
- CMake
- One Definition Rule (ODR)
- static

- Macros
- Manual memory management
- Out parameters
- Template meta programming
- anonymous namespace
- Concepts
- consteval
- Header guards
- modules
- Move
- noexcept
- operator literal
- Parameter packs
- Return Value Optimization (RVO)
- spaceship operator
- static_assert
- std::expected
- Templates

- Implicit conversions
- Preprocessor
- printf
- Slicing
- Static Initialization Order
- #define
- Concurrency
- dynamic_cast
- friend
- Pass parameter by pointer *
- Function pointers
- String view literals (with suffix)

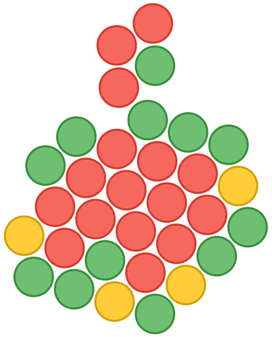
- array pointer decay
- C
- C arrays
- C casts
- C enums
- C pointers
- C strings (char *)
- comma operator
- stoi stol stoll
- Pointer arithmetic
- atomics
- coroutines
- Curiously Recurring Template Pattern (CRTP)
- decltype
- Pimpl
- std::bind
- std::bind_front/back
- std::future / std::promise
- std::invoke
- std::jthread
- std::shared_ptr
- Variadic templates

# Group by level



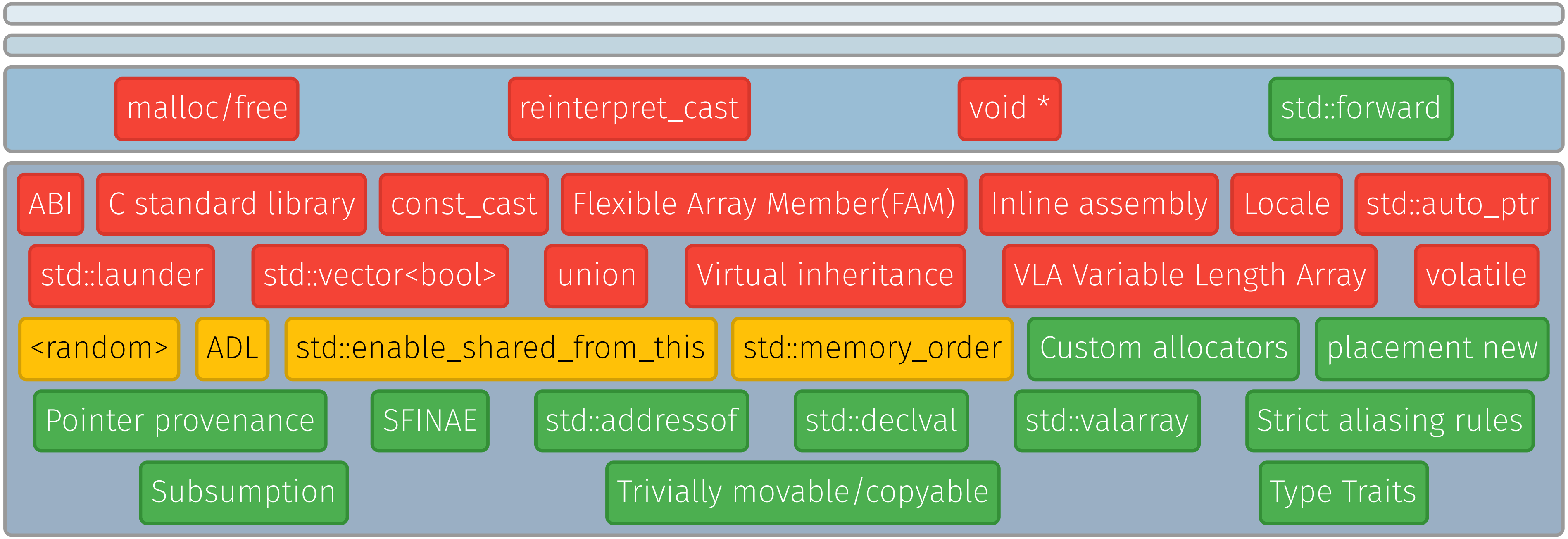
# What is advanced?

TurtleSec



Advanced features in C++ are clearly in the ***Specialized*** category, here I suspect only a few percent of C++ developers are proficient.

# What is advanced?



malloc/free reinterpret_cast void * std::forward

ABI C standard library const_cast Flexible Array Member(FAM) Inline assembly Locale std::auto_ptr

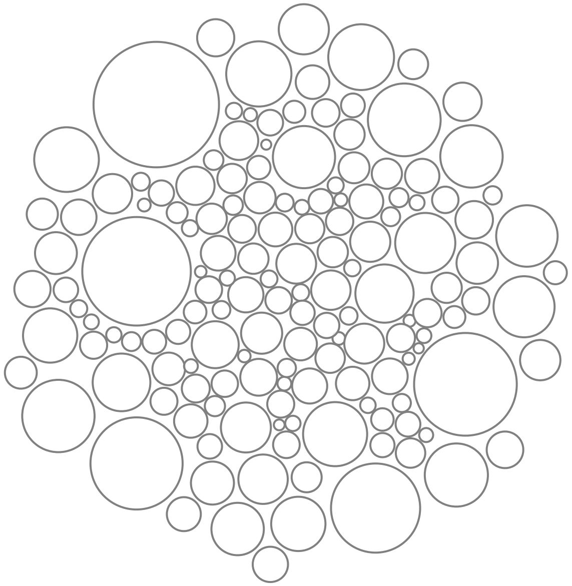
std::launder std::vector<bool> union Virtual inheritance VLA Variable Length Array volatile

<random> ADL std::enable_shared_from_this std::memory_order Custom allocators placement new

Pointer provenance SFINAE std::addressof std::declval std::valarray Strict aliasing rules

Subsumption Trivially movable/copyable Type Traits

# Do we even know what is Wat?

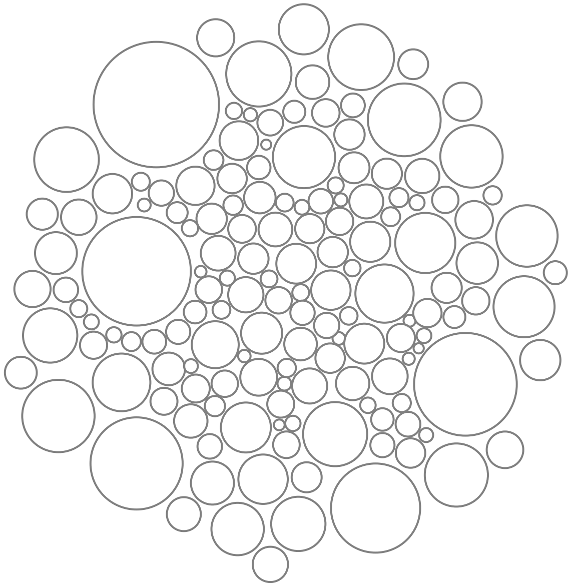


# Do we even know what is Wat?

Turns out that the biggest Wat's in the poll was

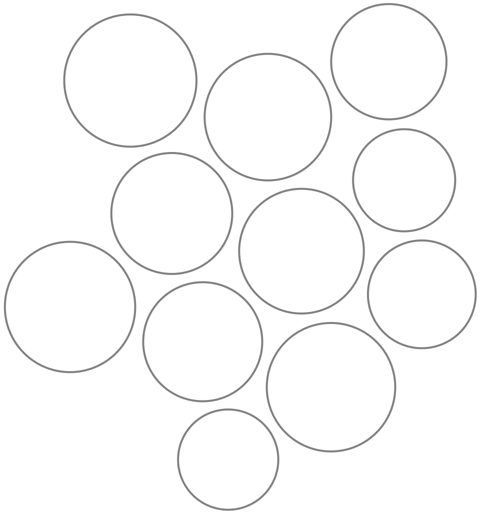
***Subsumption, std::launder and Flexible Array Member(FAM)***

# What was the least Wat?



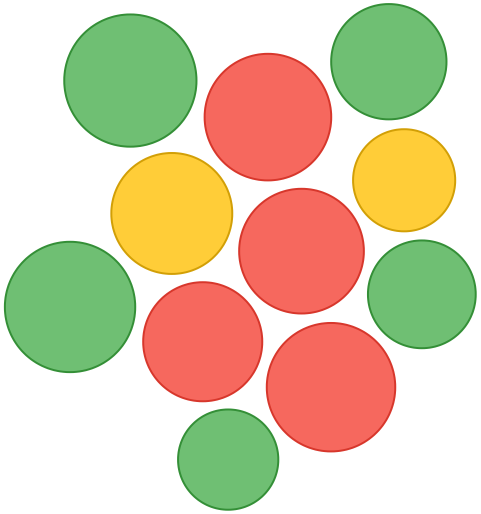
# What was the least Wat?

TurtleSec



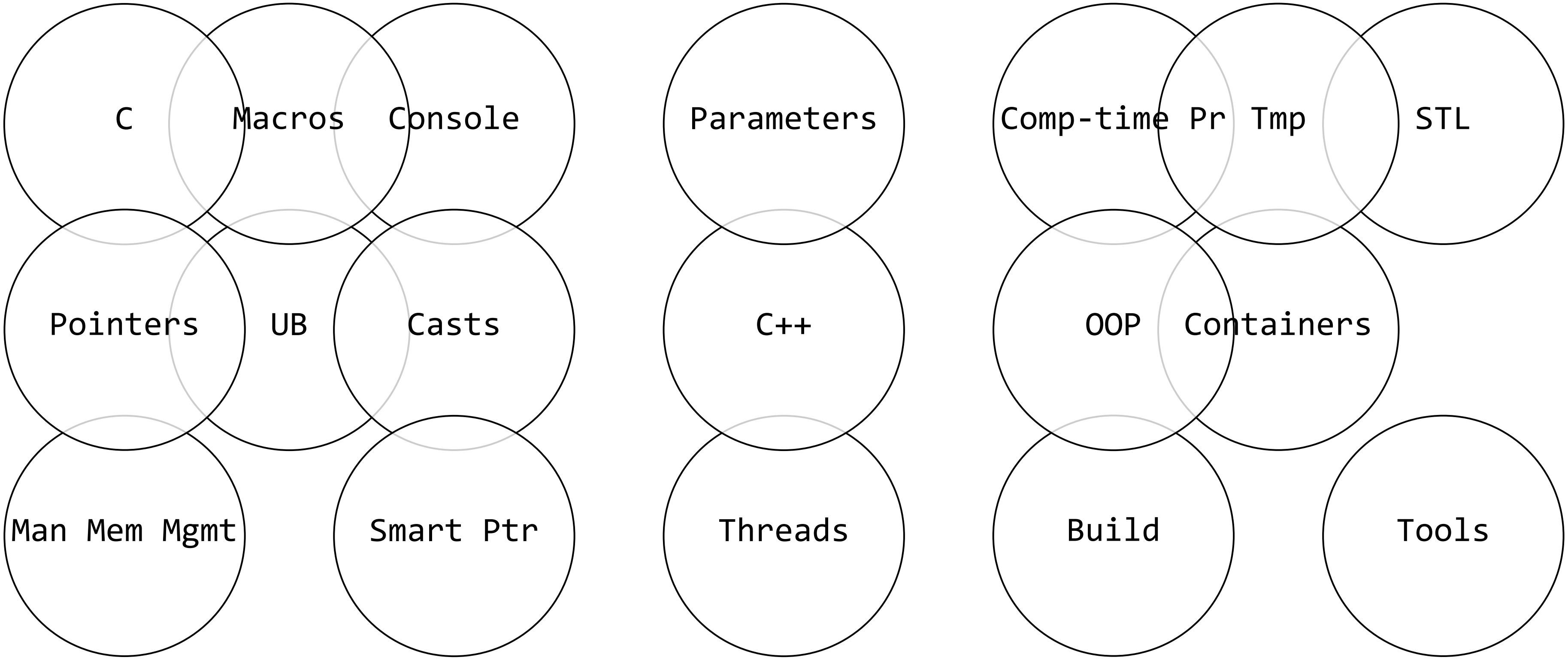
*Debuggers • Overloading • Undefined Behavior*
*Manual memory management • C • #define*
*Globals • struct • namespaces • Pointer arithmetic*
*bitwise operations*

# What was the least Wat?

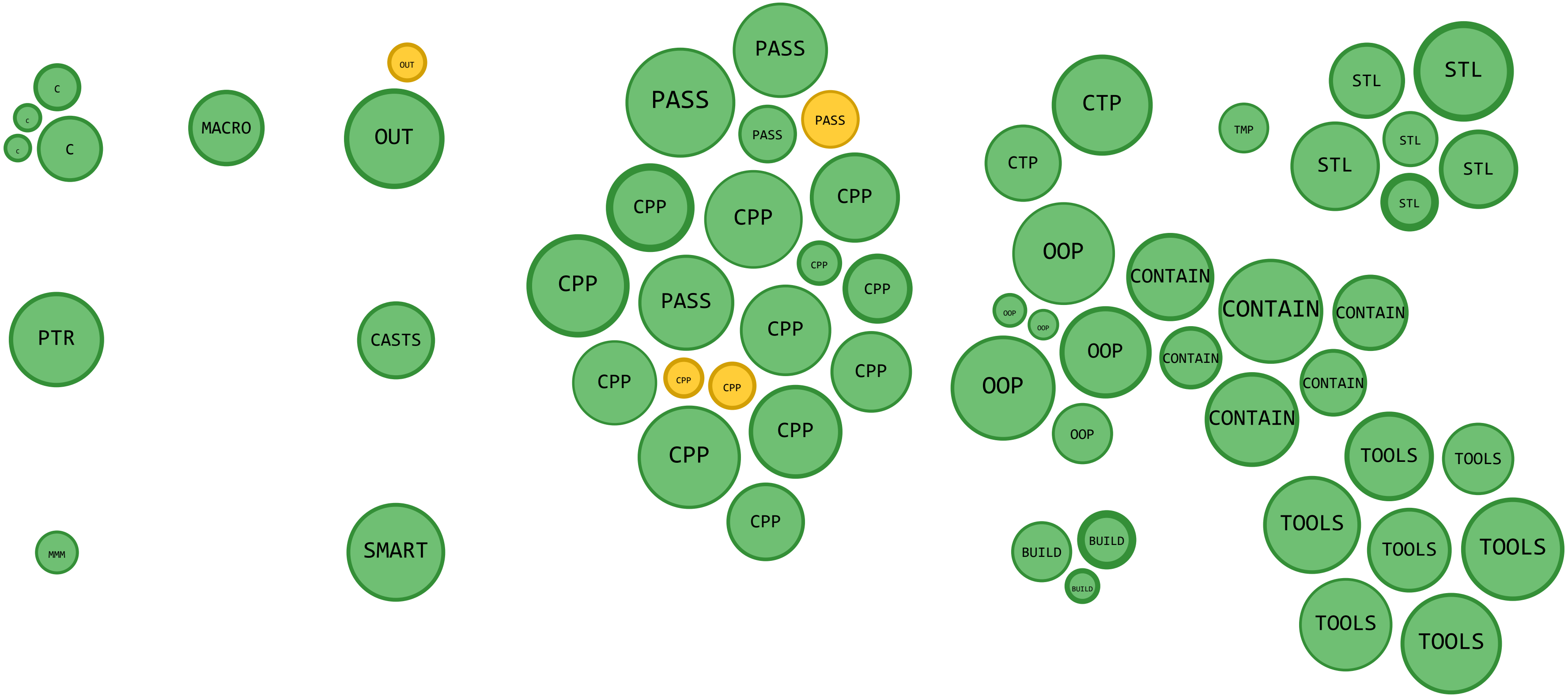


*Debuggers • Overloading • Undefined Behavior*
*Manual memory management • C • #define*
*Globals • struct • namespaces • Pointer arithmetic*
*bitwise operations*

# Atlas orientation

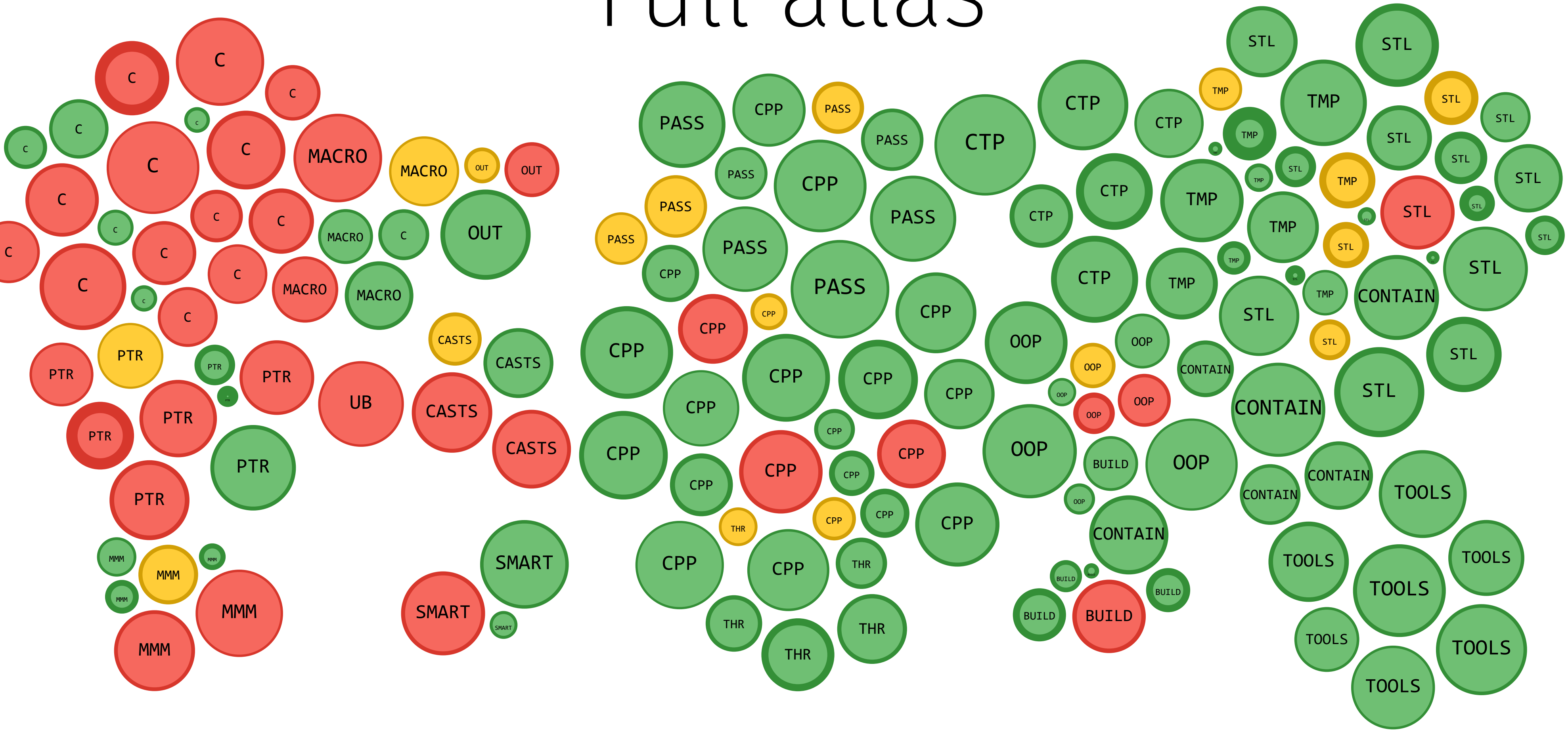


# Atlas for beginners



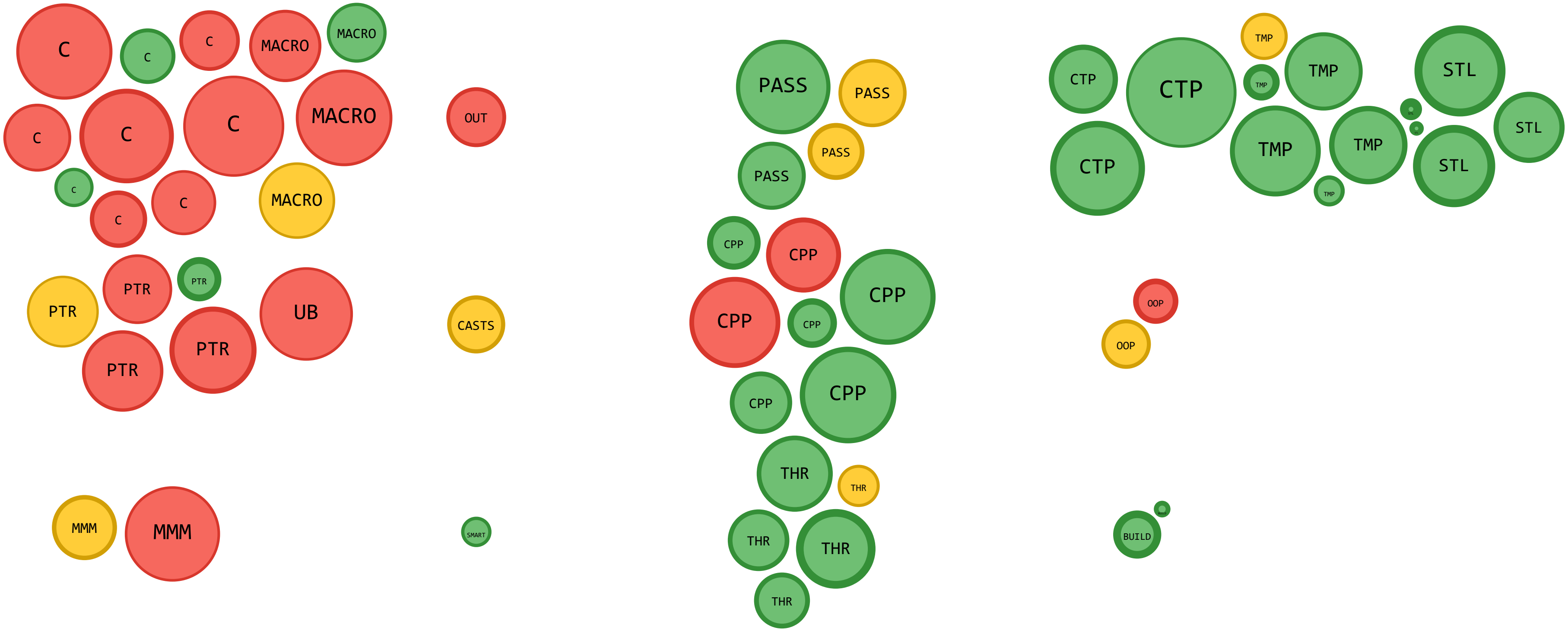
TurtleSec

# Full atlas

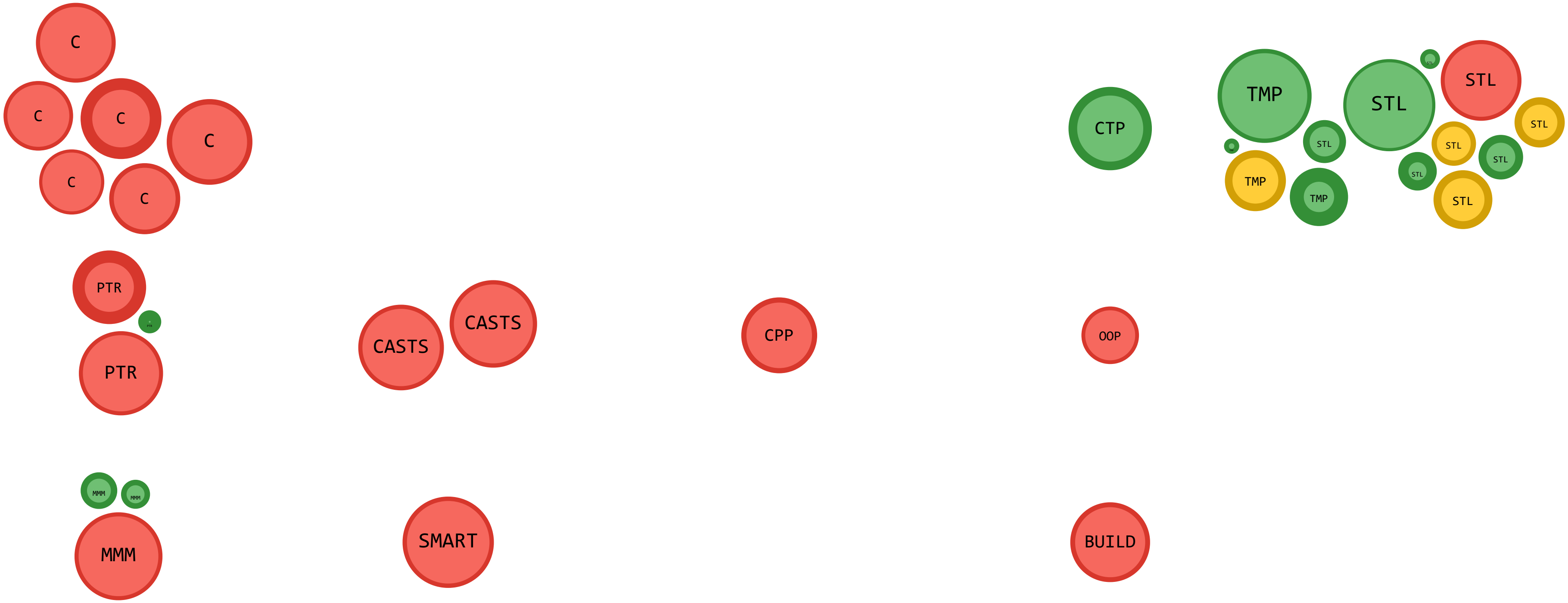


@pati_gallardo

# Atlas for intermediate



# Atlas for advanced



What does it mean?

# There seems to be an area

that we feel is appropriate for beginners

that is mostly ***Not Lava***

The question remains

is it enough?

Nothing is magic

You can't use it, if you don't teach it

# inline and ODR

*Intermediate*: One Definition Rule (ODR)

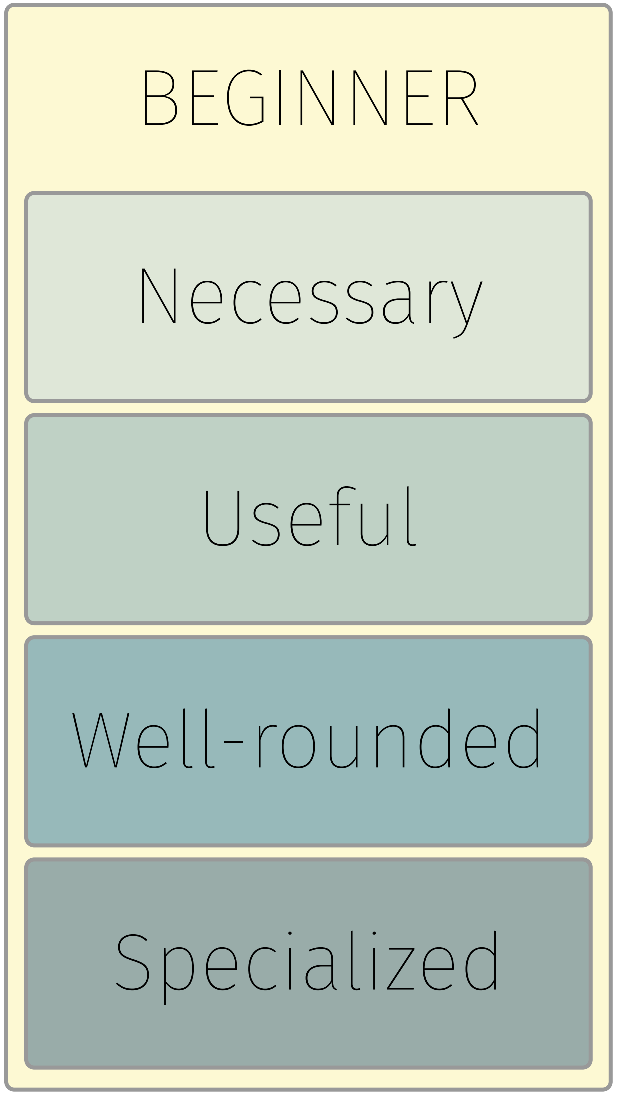
*Beginner*: inline

# Header Guards vs #pragma once

***Beginner***: #pragma once

***Intermediate***: Header Guards

***Intermediate***: Macros



BEGINNER

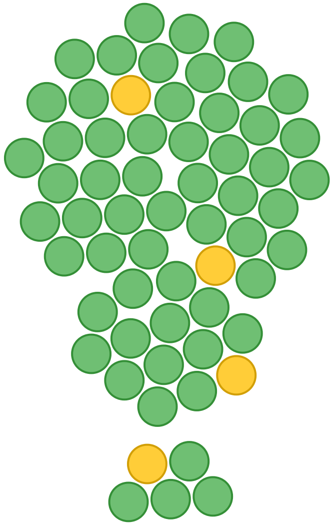
Necessary

Useful

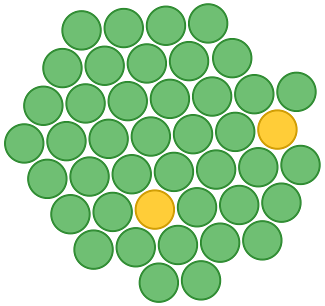
Well-rounded

Specialized

# What is beginner?



# What is beginner and necessary?



# Is this what we teach when the Floor is Lava?

- iostream
- Pass parameter by reference
- #pragma once
- ASan (Address Sanitizer)
- auto
- C-style for loops
- clang-format
- clang-tidy
- class
- const
- Debuggers
- enum class
- equals default
- Heap allocation
- Includes
- inline
- Iterators
- Lambda capture
- Lambdas
- MSan (Memory Sanitizer)
- namespaces
- Object Oriented Programming (OOP)
- Package Manager (Conan/Vcpkg)
- Pass parameter by const reference
- Pass parameter by value
- References
- Return by value
- Short-Circuit Boolean Expressions (|| &&)
- Signed integer
- Signed Integer addition/multiplication
- Stack allocation
- std::array
- std::optional
- std::size
- std::string
- std::unique_ptr
- std::vector
- STL algorithms
- String literals (no suffix)
- struct
- UBSan (Undefined Behavior Sanitizer)
- Values
- Warnings