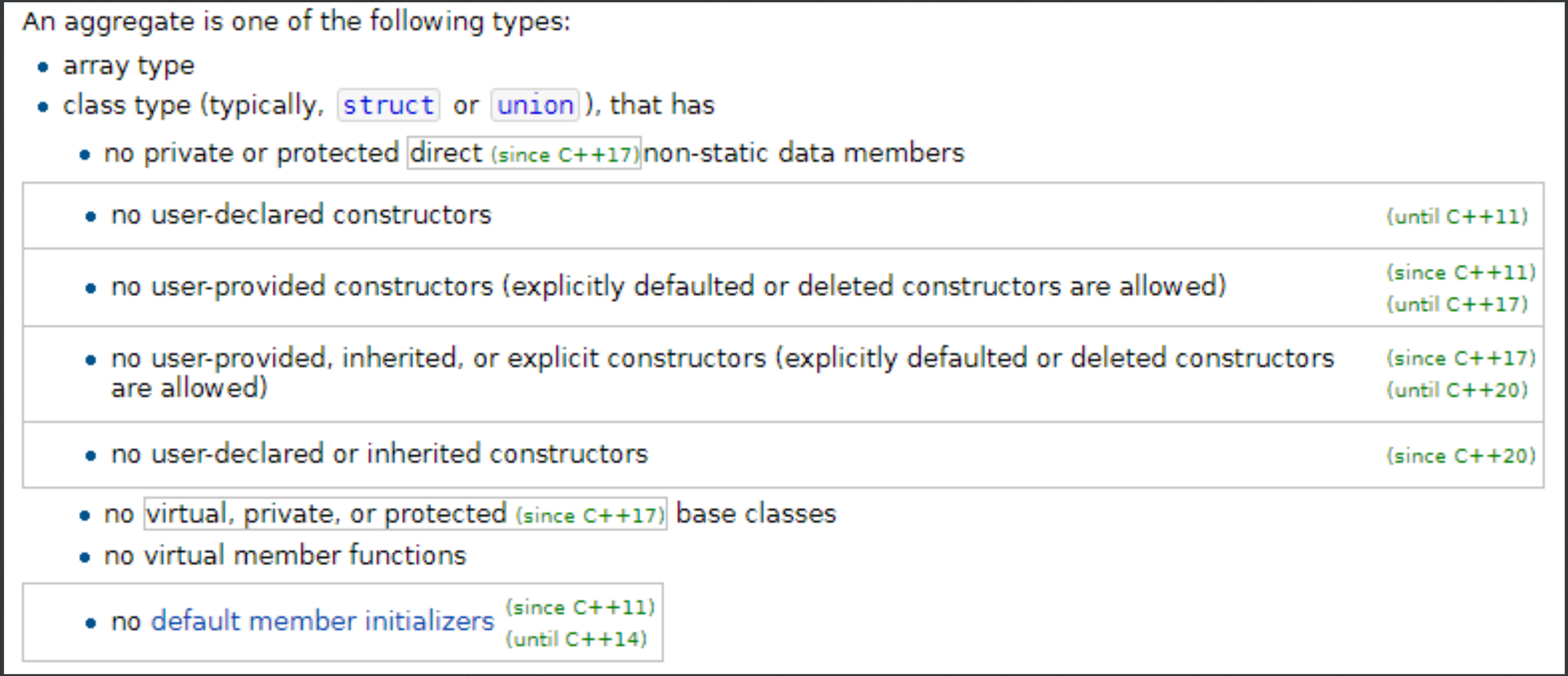
C++20 - the small things

Part 1 of 4: Initialization & CTAD

Overview

- □ Designated initializers (for aggregates)
- □ Direct initialization of aggregates
- □ Constant initialization
- □ Recap from C++17: CTAD

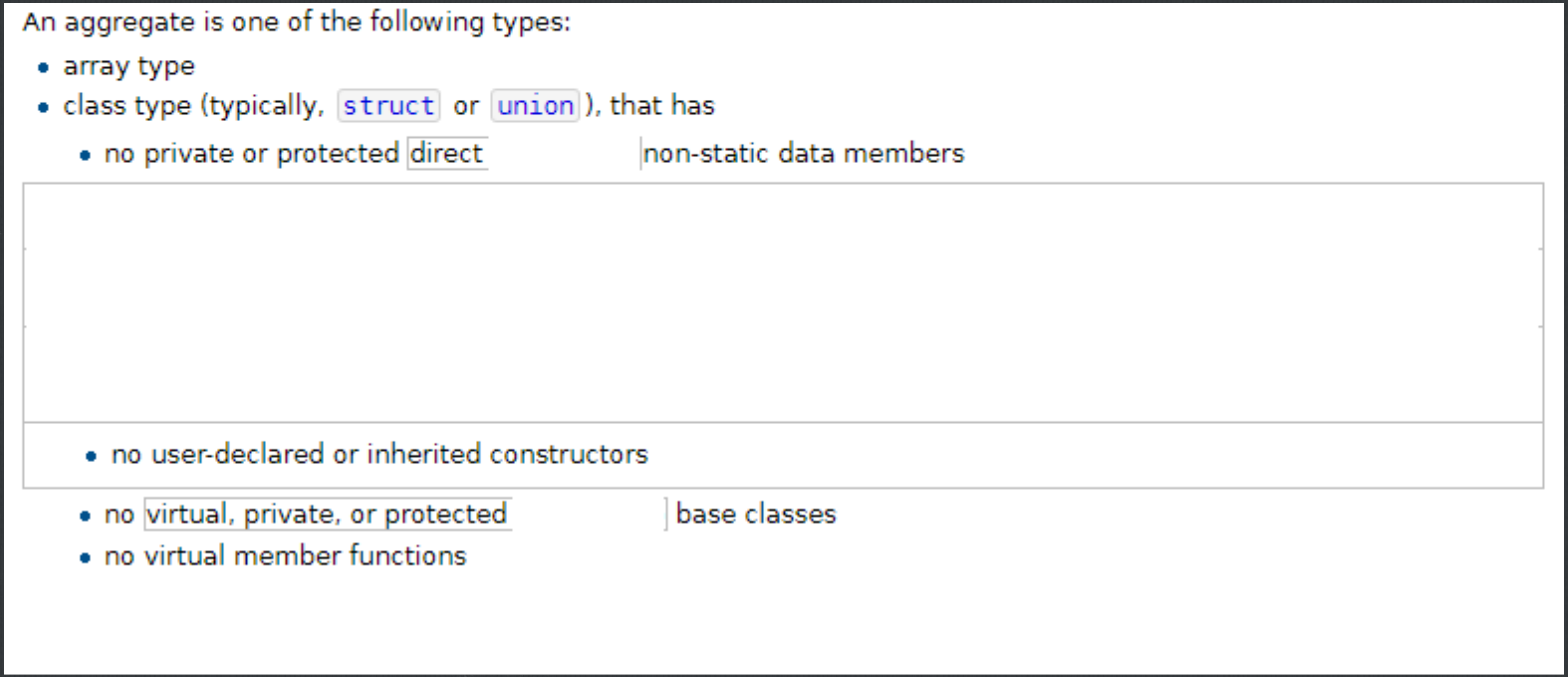
What are aggregates?



An aggregate is one of the following types:

- array type
- class type (typically, `struct` or `union`), that has
  - no private or protected direct (since C++17) non-static data members
  - no user-declared constructors (until C++11)
  - no user-provided constructors (explicitly defaulted or deleted constructors are allowed) (since C++11) (until C++17)
  - no user-provided, inherited, or explicit constructors (explicitly defaulted or deleted constructors are allowed) (since C++17) (until C++20)
  - no user-declared or inherited constructors (since C++20)
  - no virtual, private, or protected (since C++17) base classes
  - no virtual member functions
  - no default member initializers (since C++11) (until C++14)

What are aggregates? (C++20)



An aggregate is one of the following types:

- array type
- class type (typically, `struct` or `union`), that has
  - no private or protected direct non-static data members
  - no user-declared or inherited constructors
  - no virtual, private, or protected base classes
  - no virtual member functions

Designated initializers (for aggregates)

```cpp
struct Rectangle
{
   int x;
   int y;
   int w;
   int h;
};

// before C++20:
Rectangle r{1, 1, 17, 24};  // correct order?

// or
Rectangle r;
r.x = 1;
r.y = 1;
r.w = 17;
r.h = 24;       // very lengthy...
```

Designated initializers (for aggregates)

```
struct Rectangle
{
    int x;
    int y;
    int w;
    int h;
};

// using designated initializers:
Rectangle r{.x=1, .y=1, .w=17, .h=24};

// or
Rectangle r = {.x=1, .y=1, .w=17, .h=24};

// or
Rectangle r{.x{1}, .y{1}, .w{17}, .h{24}};
```

Designated initializers (for aggregates)

```
struct Position
{
   int x;
   int y;
};

struct Rectangle
{
   Position pos;
   int w;
   int h;
};

// nested:
Rectangle r{.pos={.x=1, .y=1}, .w=17, .h=24};
```

Designated initializers (for aggregates)

```
Rectangle r{.x=1, .y=1, .w=17, .h=24};
```

□ Actually a C99 feature, but stricter in C++

□ Cannot be out of order

```
Rectangle r{.y=1, .x=1, .h=24, .w=17}; // error
```

□ No “flat” nesting

```
Rectangle r{.pos.x=1, ...}; // error
```

□ Cannot be mixed with regular initializers

```
Rectangle r{.x=1, .y=1, 17, 24}; // error
```

□ Cannot be used with arrays

```
int arr[3]{.[1]=7}; // error
```

Designated initializers: named argument “emulation”

```
// original function
void connect(std::string host, unsigned short port,
             Duration connectTimeout, Duration responseTimeout)
// caller
connect("localhost", 80, 5s, 10s);

// with "named arguments"
struct ConnectArgs
{
   std::string host;
   unsigned short port;
   Duration connectTimeout;
   Duration responseTimeout;
};
void connect(ConnectArgs);

// caller
connect({.host="localhost", .port=80, .connectTimeout=5s, .responseTimeout=10s});
```

Direct initialization of aggregates

```
struct Position
{
    int x;
    int y;
};
Position pos{1, 2};
```

- □ **Problems with braced direct initialization in C++17:**
  - □ **Does not work with macros:**

    ```
    assert(Position(2, 3)); // ok
    assert(Position{2, 3}); // preprocessor error :-(
    ```

  - □ **Can’t do perfect forwarding in templates**
    - □ **No** `emplace()`, `make_unique()`, **… for aggregates ☹**

Direct initialization of aggregates

```
struct Position
{
    int x;
    int y;
};
Position pos{1, 2};
Position pos(1, 2); // works in C++20
```

- In C++20, `(args)` and `{args}` will do the same thing, except:
  - `()` does not call `std::initializer_list` constructors
  - `{}` does not allow narrowing conversions

Constant initialization

- □ Does anybody remember the initialization order fiasco for static/global objects?!
- □ One solution: use constexpr:

```
constexpr float getPi(bool fake) { return fake ? 47.11 : 3.14; }
constexpr float pi = getPi(false);
```

  - □ Pro: compile-time initialization, no runtime order problems
  - □ Con: implies const, i.e. cannot be changed after initialization

Constant initialization: constinit

```cpp
// constinit forces static or thread storage duration
// and initialization at compile-time
static constinit std::mutex s_mutex;
static constinit LogThread* s_thread = nullptr;

// static constinit LogThread* s_thread = new LogThread(); // ill-formed, error
void initLogging()
{
   std::scoped_lock lock(s_mutex);
   if (!s_thread) // ...
}
void shutdownLogging()
{
   std::scoped_lock lock(s_mutex);
   if (s_thread) // ...
}
// constexpr = constinit + const
```

CTAD – Class Template Argument Deduction

```
// before C++17:

std::pair<int, const char*> pair(13, "Hello");
auto pair = std::make_pair(13, "Hello");

std::tuple<int, float, bool, std::string> t(1, 3.14, true, "text");

std::array<int, 4> values = {1, 2, 3}; // ooops, forgot one
std::vector<int> values{1, 2, 3};

std::recursive_mutex mutex;
std::lock_guard<std::recursive_mutex> lock(mutex);

std::unique_ptr<T> ptr(new T());

// quite a lot to type ...
```

CTAD – Class Template Argument Deduction

```
// with CTAD

std::pair pair(13, "Hello");     // std::pair<int, const char*>


auto t = std::tuple(1, 3.14, "text"s);   // std::tuple<int, float, std::string>

constexpr auto values = std::array{1, 2, 3}; // std::array<int, 3>
std::vector values = {1, 2, 3};              // std::vector<int>

std::recursive_mutex mutex;
std::scoped_lock lock(mutex);      // also: use scoped_lock instead of lock_guard

std::unique_ptr ptr(new T());

// note: can be controlled/customized with "deduction guides"
```

CTAD: changes in C++20

```cpp
// C++20 adds CTAD for aggregates and template aliases

template<typename T, typename U>
struct aggr_pair
{
    T first;
    U second;
};

aggr_pair p = {1, true};    // requires deduction guide in C++17,
                            // works out-of-the-box in C++20
```