

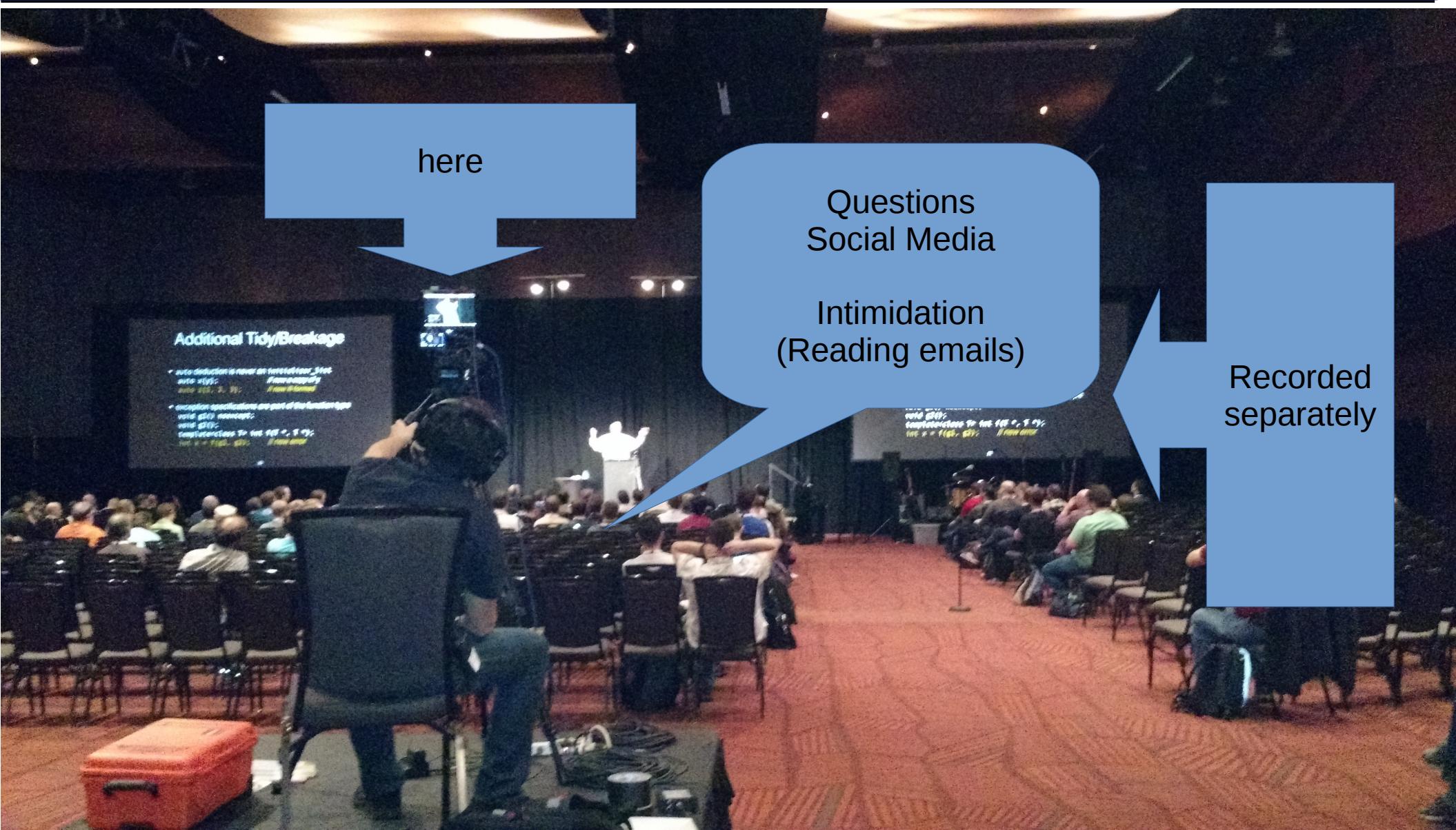
Presenting Code
Jens Weller
Meeting C++

CppCon 2016
@meetingcpp

Motivation

- Inspired by Scott Meyers Keynote
 - Meeting C++ 2014
 - Last keynote from Scott Meyers
- Seeing many talks
 - Different ways
 - Its actually an important part of your talk

Where is your Audience?



Forms of Code...

- Text
 - Highlighted?
- Screenshot
- Live demo?
- Often on slides
 - Not so well supported

Example

- Just Code

```
template<class AbstractClass, class IdType = size_t,  
         class MakeType = boost::function<AbstractClass*()>>  
struct Factory  
{  
    boost::container::flat_map<IdType, MakeType> factory_map;  
public:  
    void register_factory(IdType type_id, const MakeType& make)  
    ...  
    template<class ...args>  
    abstract_type* create(IdType id, args&&... a) const  
    ...
```

- Highlight what you are going to talk about

```
template<class AbstractClass,class IdType = size_t,  
        class MakeType = boost::function<AbstractClass*()>>  
struct Factory  
{  
    boost::container::flat_map<IdType,MakeType> factory_map;  
public:  
    void register_factory(IdType type_id,const MakeType& make)  
    ...  
    template<class ...args>  
    abstract_type* create(IdType id, args&&... a) const  
    ...  
factory.registerType(dir_typeid,boost::bind(boost::factory<DirPanel*>(),_1,_2));
```

Another good Example

Generic Server

```
template <typename ConnectionHandler>
class asio_generic_server
{
    using shared_handler_t = std::shared_ptr<ConnectionHandler>;
public:
    asio_generic_server(int thread_count)
        , thread_count_(thread_count)
        , acceptor_(io_service_);
    void start_server(uint16_t port)
    {
    }
private:
    void handle_new_connection( shared_handler_t handler
                               , system::error_code const & error )
    {
    }
    int thread_count_;
    std::vector<std::thread> thread_pool_;
    asio::io_service io_service_;
    asio::ip::tcp::acceptor acceptor_;
};
```

Another good Example

Generic Server

```
template <typename ConnectionHandler>
class asio_generic_server
{
    using shared_handler_t = std::shared_ptr<ConnectionHandler>;
public:
    asio_generic_server(int thread_count=1)
        : thread_count_(thread_count)
        , acceptor_(io_service_)
    {}
    void start_server(uint16_t port)
    {
    }
private:
    void handle_new_connection( shared_handler_t handler
                                , system::error_code const : error )
    {
    }
    int thread_count_;
    std::vector<std::thread> thread_pool_;
    asio::io_service io_service_;
    asio::ip::tcp::acceptor acceptor_;
};
```

Also

Writing a TCP server

```
#include <os>
#include <net/inet4>

void Service::start(const std::string& {
    auto& server = net::Inet4::stack().tcp().bind(80);
    server.on_connect([](auto conn) {
        conn->on_read([conn](auto buf, size_t n) {
            conn->write("My first unikernel!\n"s);
        });
    });
}
```

Handle any delegate
TCP events

Laserpointers!

```
p00.cpp      x p01.cpp      *
31     return unique_ptr<T>(new T{forward<Args>(args)...});
32 }
33 */
34 // The proposed `constexpr if` has to follow these rules:
35 /*
36     * Restricted to block scopes.
37
38     * Always going to establish a new scope.
39
40     * Required that there exists values of the condition so
41     * that either condition branch is well-formed.
42 */
43
44 // The above rules deal with the controversial ideas of N3613, making
45 // `constexpr if` an intuitive and unsurprising compile-time version
46 // of the regular `if` statement.
47
48 // The syntax is not yet definitive: some slight changes were proposed
49 // in P0292R0:
50 // open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0292r0.html
51 // ("constexpr if: A slightly different syntax")
52
53 // The paper proposes the following syntax:
54 /*
55
56 */
57 // If you want to know more about
```

Laserpointers!

Abstractions: Patterns, Policies, and Spaces



- Parallel Pattern of user's computations
 - parallel_for, parallel_reduce, parallel_scan, task-graph, ... (*extensible*)
- Execution Policy tells *how* user computation will be executed
 - Static scheduling, dynamic scheduling, thread-teams, ... (*extensible*)
- Execution Space tells *where* user computations will execute
 - Which cores, numa region, GPU, ... (*extensible*)
- Memory Space tells *where* user data resides
 - Host memory, GPU memory, high bandwidth memory, ... (*extensible*)
- Layout (policy) tells *how* user array data is laid out in memory
 - Row-major, column-major, array-of-struct, struct-of-array ... (*extensible*)
- Differentiating: Layout and Memory Space
 - Versus other programming models (OpenMP, OpenACC, ...)
 - Critical for performance portability ...

Laserpointers!

Did you see it?

Where is your god now?



Conclusions

- Laser pointers are not recorded
 - Are already difficult to see for the audience
- Highlight what you are talking about
- Less code is better
 - Slide++
- Livedemo
- Watch the keynote from Scott Meyers