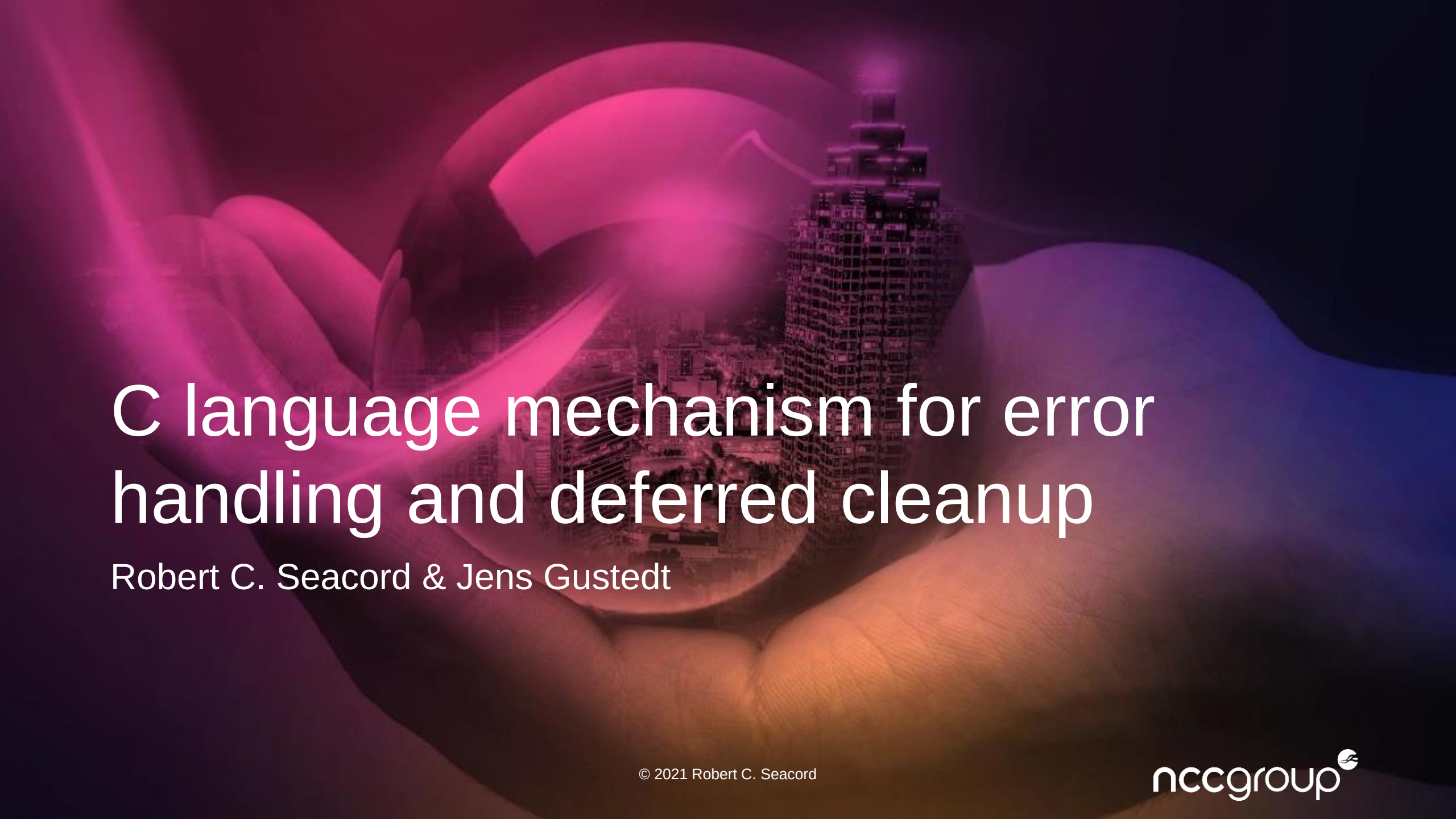


C language mechanism for error handling and deferred cleanup

Robert C. Seacord & Jens Gustedt

nccgroup

Presenter Bio: Robert C. Seacord

Technical Director

- Develop and deliver secure coding training classes in Java, C, C++ and C#
- Code security review
- Research WG Lead: Reducing Vuls at Scale & OSS

Robert is an ISO/IEC JTC1/SC22/WG14 international standardization working group for the C programming language expert.

nccgroup

Defer Mechanism

The defer mechanism can restore a previously known property or invariant that is altered during the processing of a code block.

- useful for paired operations, where one operation is performed at the start of a code block and the paired operation is performed before exiting the block.
- pattern is common in
  - resource management
  - synchronization
  - outputting balanced strings (e.g., parenthesis or HTML).

Resource Management

Examples of C standard library functions that acquire resources include:

- allocated storage: **`malloc`**, **`calloc`**, **`realloc`**, **`aligned_alloc`**, **`strdup`**, **`strndup`**
- streams: **`fopen`**, **`freopen`**
- temporary file: **`tmpfile`**
- threads: **`thrd_create`**
- thread specific storage: **`tss_create`**
- condition variable: **`cnd_init`**
- condition variable: **`cnd_wait`**
- mutexes: **`mtx_init`**, **`mtx_lock`**, **`mtx_timedlock`**, **`mtx_trylock`**

Security Concerns

A *denial-of-service* (DoS) attack occurs when users are unable to access information systems, devices, or other network resources as the result of an attack.

DoS attacks attempts frequently take the form of a resource-exhaustion attack that makes a computer resource insufficiently available to the application.

Double Free vulnerabilities can be exploited to execute arbitrary code with the permissions of a vulnerable process.

A common source of this is the vulnerability is the release of resources during error handling and again during normal processing.

Defer Statement

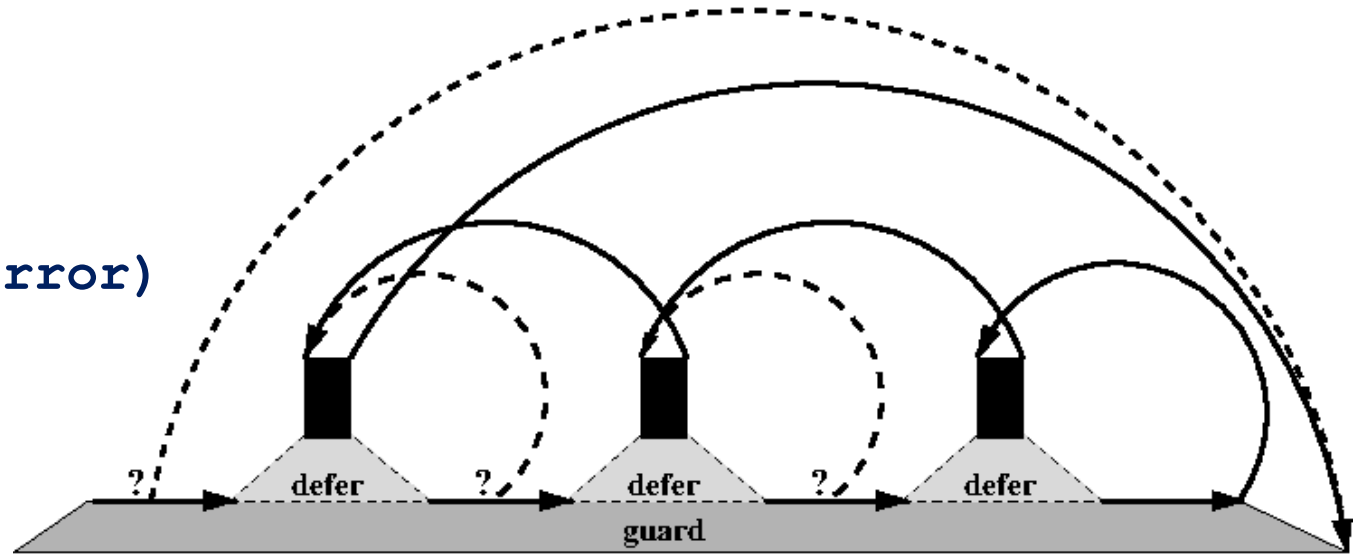
A **`defer`** statement defers the execution of a *deferred statement* until the containing guarded block terminates

Deferred statement is sequenced in last-in-first out (LIFO) order after all statements just before the guarded block terminates.

A block can contain multiple **`defer`** statements.

Defer Example

```
errno_t func() {
  void * const p = malloc(25);
  if (!p) return ENOMEM;
  defer free(p);
  void * const q = malloc(25);
  if (!q) return ENOMEM;
  defer free(q);
  if (mtx_lock(&mut) == thrd_error)
    return thrd_error;
  defer mtx_unlock(&mut);
  // all resources acquired
}
```



Stack Unwinding

Before the normal processing of the termination event, the C library functions **`exit`** and **`thrd_exit`** trigger an execution of all deferred statements for the current thread by unwinding the stack.

Panic and Recover

The primary purpose of

- `defer` is to manage the release of resources
- `panic/recover` is error handling

Panic/recover are similar to `throw/catch` in C++ while `defer` is similar to resource acquisition is initialization (RAII).

Panic Macro

A panic may potentially be the result of a trap, such as an invalid arithmetic operation or the result of invoking either of the panic macro.

The `panic` macro indicates an abnormal execution condition and will *unwind* the caller's stack and execute all deferred statements registered in that stack frame.

The `recover` function

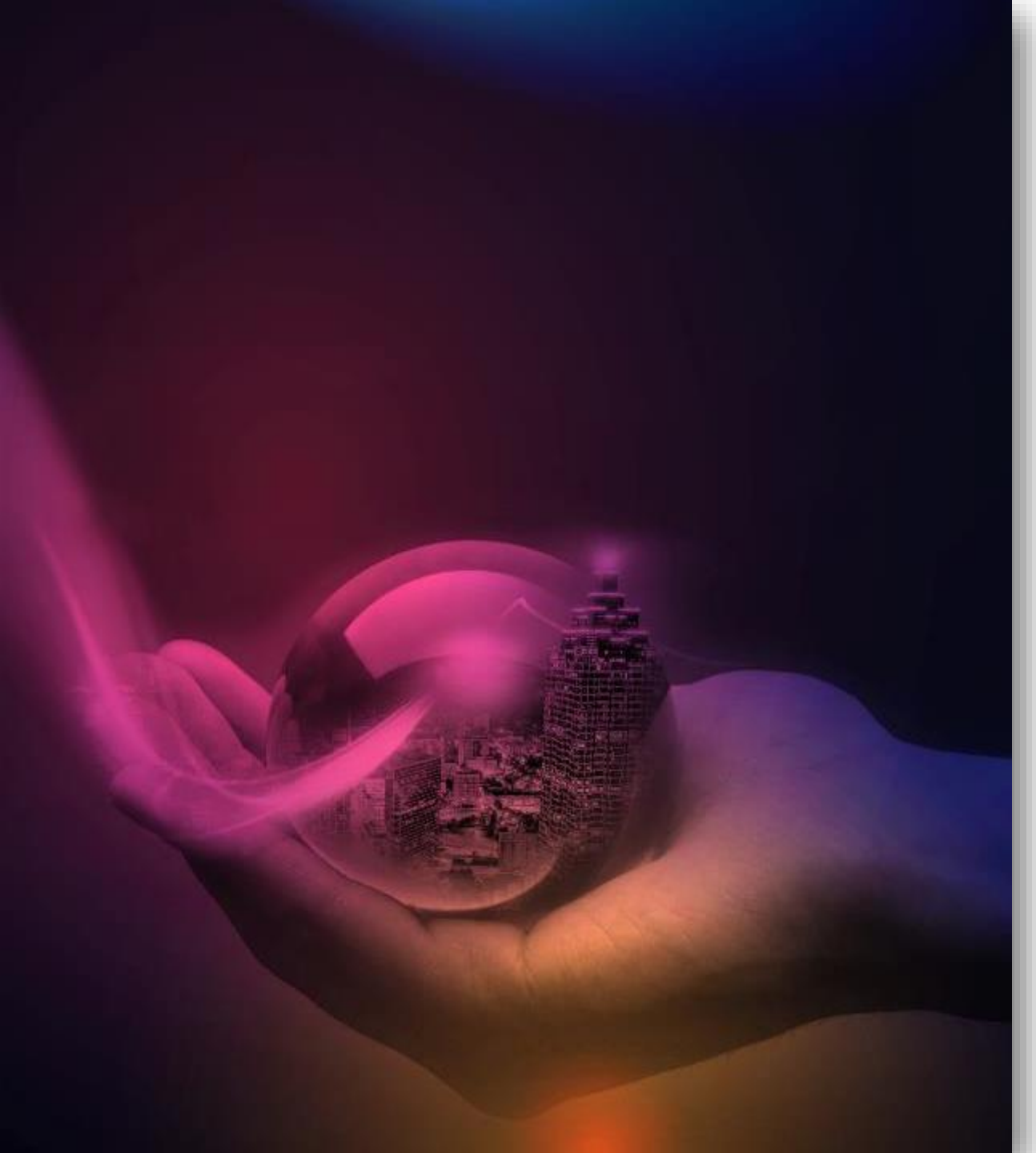
The `recover` function returns an integer value that indicates the reason the deferred statement is executing.

If the return value is equal to zero

- the execution of the deferred statement is the result of the regular termination of the guarded block.
- processing of deferred statements continues as if the `recover` function had not been called.

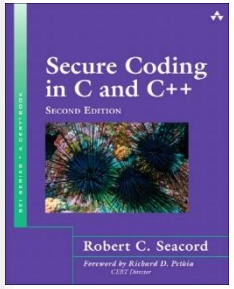
If the `recover` function returns a non-zero value, the thread or program is panicking.

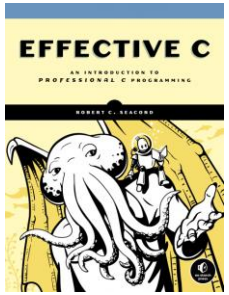
- processing of deferred statements stops with the termination of the current deferred statement.
- responsibility passes to the application.
- a new panic can be triggered by calling either the `panic` macro.

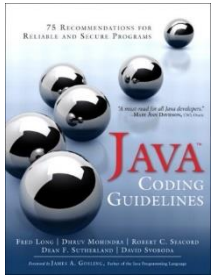


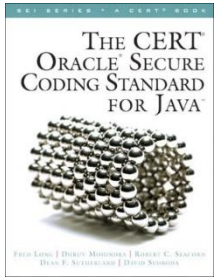
Point of Contact

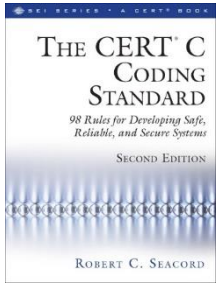
Robert C. Seacord
Technical Director
Twitter: @rcs
LinkedIn.com/in/robertseacord/
T: +1 (412) 580-2981
E: Robert.Seacord@nccgroup.com
W: www.nccgroup.com











nccgroup