accu
2023

NOBODY CAN PROGRAM CORRECTLY

LESSONS FROM 20 YEARS OF DEBUGGING C++ CODE

SEBASTIAN THEOPHIL

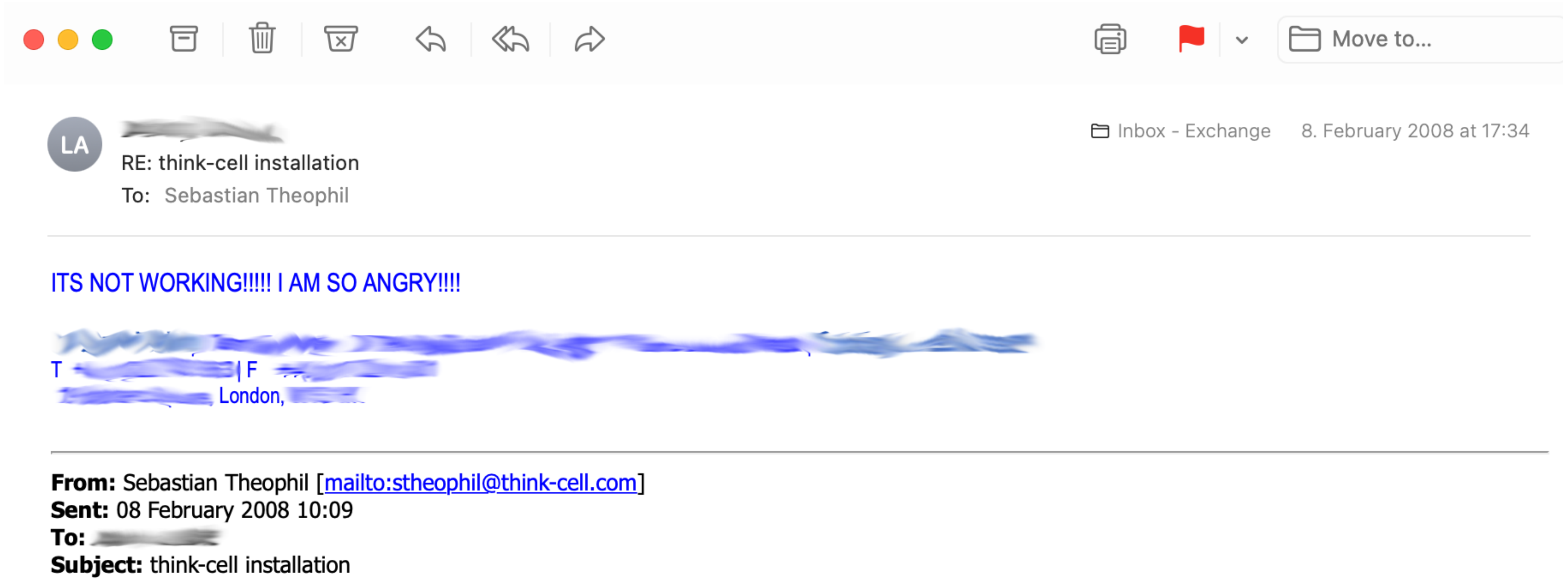
What is a bug?

Your program has a **specification**: *Implicit* (usually) or *explicit* (rarely)

Your program has a **bug** if its behavior does not conform to the specification.

A **bug report** describes a *symptom* of a bug. Not the bug itself.

First, you have to notice the bug



Move to...

[redacted]

RE: think-cell installation

To: Sebastian Theophil

Inbox - Exchange 8. February 2008 at 17:34

ITS NOT WORKING!!!!! I AM SO ANGRY!!!!

[redacted]

T [redacted] | F [redacted]

[redacted] London, [redacted]

From: Sebastian Theophil [mailto:stheophil@think-cell.com]
Sent: 08 February 2008 10:09
To: [redacted]
Subject: think-cell installation

First, you have to notice the bug

How do you notice that there is a bug in your program?

- a crash
- a core dump
- an error message
- a line written to a log file
- an observable misbehavior

First, you have to notice the bug

How do you notice that there is a bug in your program?

- a crash
- a core dump
- an error message
- a line written to a log file
- an observable misbehavior

How many do you notice?

How many when they occur on your client's computer?

First, you have to notice the bug

At Compile Time

- Type checking
- `static_assert`
- `constexpr` evaluation

First, you have to notice the bug

At Compile Time

- Type checking
- `static_assert`
- `constexpr` evaluation

CppCon 2020: “Constexpr Everything” - The Standard Library, Microkernel, Apps, and Unit Tests - Rian Quinn

First, you have to notice the bug

think-cell

At Compile Time

- Type checking
- `static_assert`
- `constexpr` evaluation

CppCon 2020: “Constexpr Everything” - The Standard Library, Microkernel, Apps, and Unit Tests - Rian Quinn

At Build Time & QA

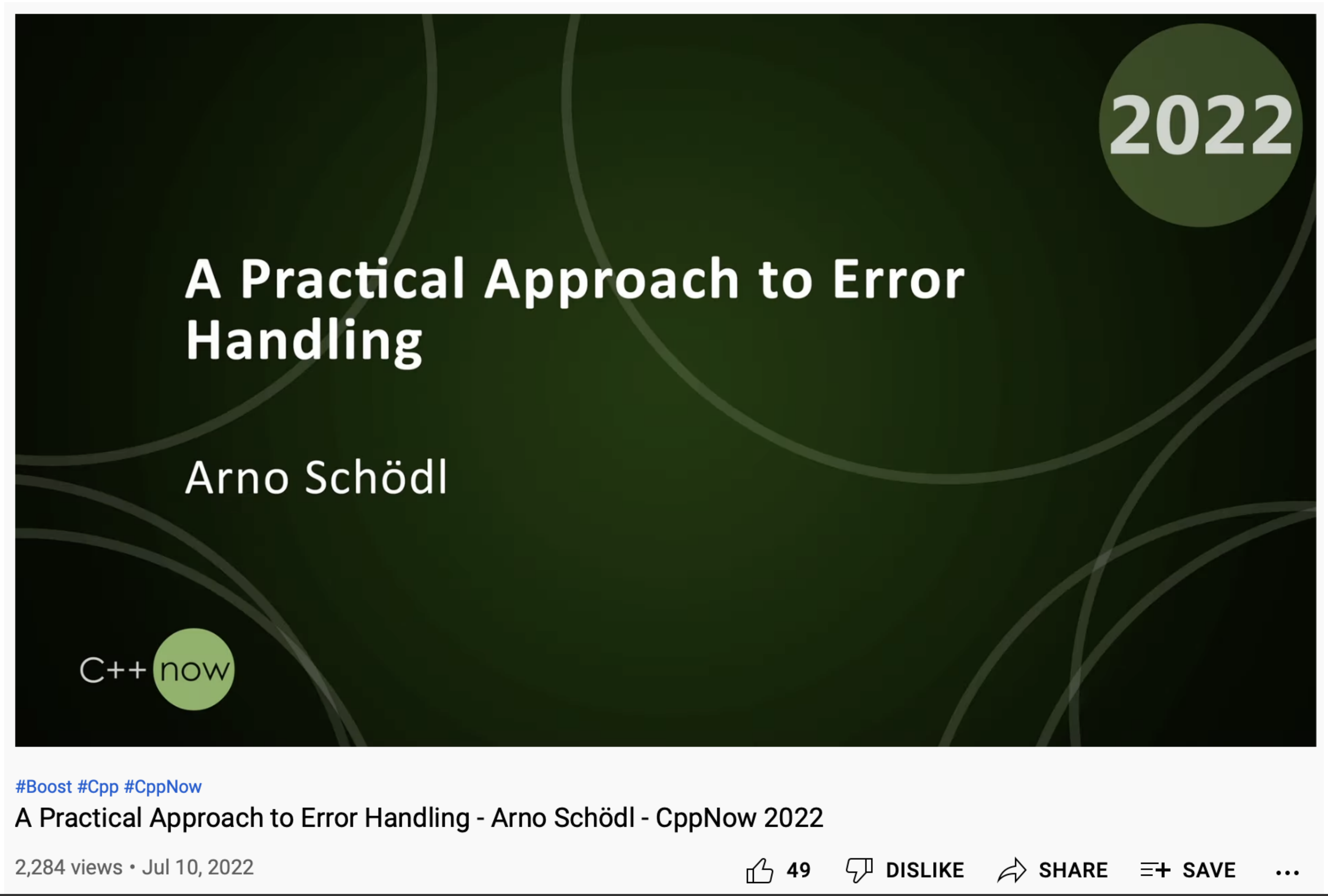
- Unit testing
- Automated testing

First, you have to notice the bug

At Runtime

- Strict error checking
  - Check all API return values and report unexpected values
  - Assert pre-conditions and post-conditions
  - Report if they fail
- Enforce invariants, notice unexpected behavior sooner

First, you have to notice the bug

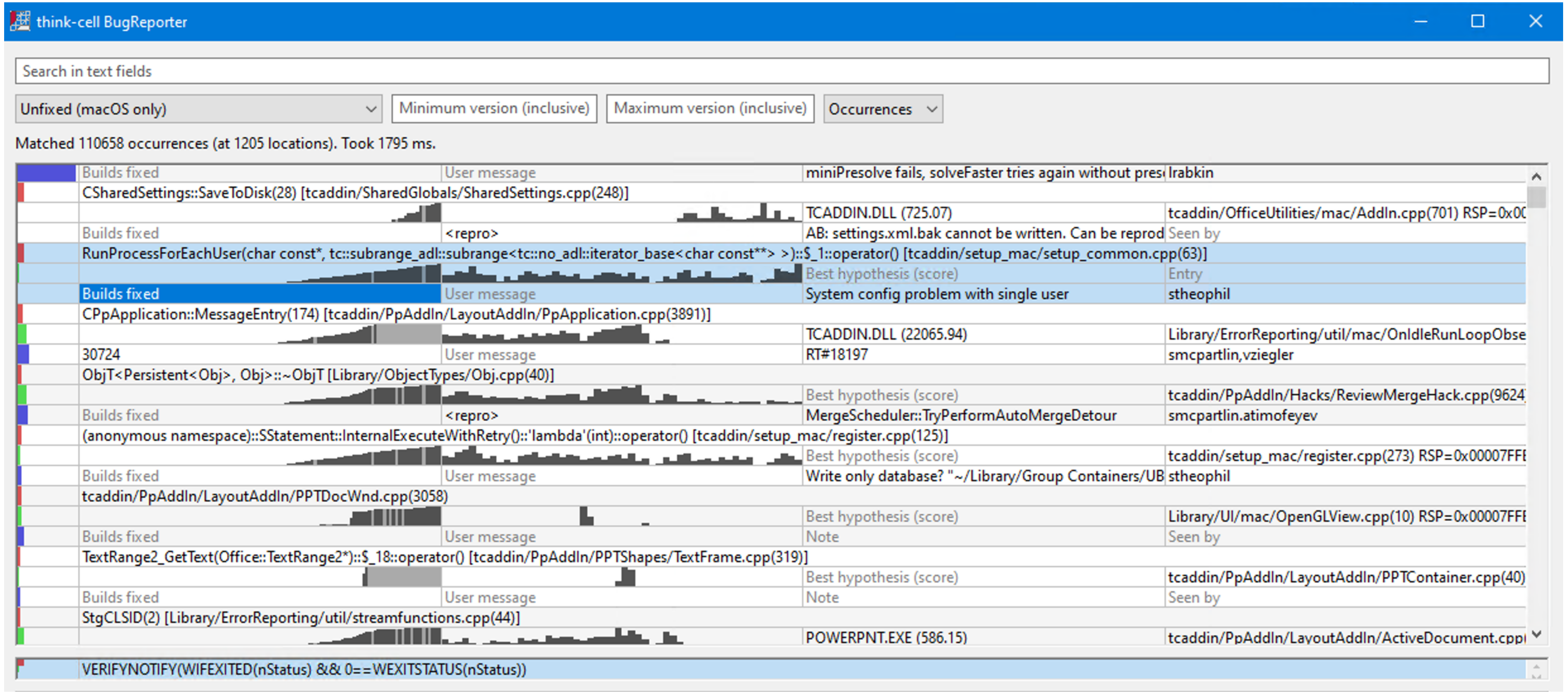


#Boost #Cpp #CppNow

A Practical Approach to Error Handling - Arno Schödl - CppNow 2022

2,284 views • Jul 10, 2022

First, you have to notice the bug



think-cell BugReporter

Search in text fields

Unfixed (macOS only) | Minimum version (inclusive) | Maximum version (inclusive) | Occurrences

Matched 110658 occurrences (at 1205 locations). Took 1795 ms.

Builds fixed	User message	miniPresolve fails, solveFaster tries again without pres	lrabkin
CSharedSettings::SaveToDisk(28) [tcaddin/SharedGlobals/SharedSettings.cpp(248)]			
		TCADDIN.DLL (725.07)	tcaddin/OfficeUtilities/mac/AddIn.cpp(701) RSP=0x00
Builds fixed	<repro>	AB: settings.xml.bak cannot be written. Can be reprod	Seen by
RunProcessForEachUser(char const*, tc::subrange_adl::subrange<tc::no_adl::iterator_base<char const**> >)::$_1::operator() [tcaddin/setup_mac/setup_common.cpp(63)]			
		Best hypothesis (score)	Entry
Builds fixed	User message	System config problem with single user	stheophil
CPpApplication::MessageEntry(174) [tcaddin/PpAddIn/LayoutAddIn/PpApplication.cpp(3891)]			
		TCADDIN.DLL (22065.94)	Library/ErrorReporting/util/mac/OnIdleRunLoopObse
30724	User message	RT#18197	smcpartlin,vziegler
ObjT<Persistent<Obj>, Obj>::~ObjT [Library/ObjectTypes/Obj.cpp(40)]			
		Best hypothesis (score)	tcaddin/PpAddIn/Hacks/ReviewMergeHack.cpp(9624
Builds fixed	<repro>	MergeScheduler::TryPerformAutoMergeDetour	smcpartlin.atimofeyev
(anonymous namespace)::SStatement::InternalExecuteWithRetry()::'lambda'(int)::operator() [tcaddin/setup_mac/register.cpp(125)]			
		Best hypothesis (score)	tcaddin/setup_mac/register.cpp(273) RSP=0x00007FFE
Builds fixed	User message	Write only database? "~/Library/Group Containers/UB	stheophil
tcaddin/PpAddIn/LayoutAddIn/PPTDocWnd.cpp(3058)			
		Best hypothesis (score)	Library/UI/mac/OpenGLView.cpp(10) RSP=0x00007FFE
Builds fixed	User message	Note	Seen by
TextRange2_GetText(Office::TextRange2*)::$_18::operator() [tcaddin/PpAddIn/PPTShapes/TextFrame.cpp(319)]			
		Best hypothesis (score)	tcaddin/PpAddIn/LayoutAddIn/PPTContainer.cpp(40)
Builds fixed	User message	Note	Seen by
StgCLSID(2) [Library/ErrorReporting/util/streamfunctions.cpp(44)]			
		POWERPNT.EXE (586.15)	tcaddin/PpAddIn/LayoutAddIn/ActiveDocument.cpp

VERIFYNOTIFY(WIFEXITED(nStatus) && 0==WEXITSTATUS(nStatus))

Learning from a single occurrence?

think-cell

"One in a million is always next Tuesday."

Gordon Letwin, architect for MS-DOS 4

https://docs.microsoft.com/en-us/archive/blogs/larryosterman/one-in-a-million-is-next-tuesday

May be a rare chance to analyze a problem.

Hard to reproduce in the lab, yet with an obvious fix.

Will occur 1000s of times once product is rolled out!

Learning from a single occurrence?

"One in a million is always next Tuesday."

Gordon Letwin, architect for MS-DOS 4

https://docs.microsoft.com/en-us/archive/blogs/larryosterman/one-in-a-million-is-next-tuesday

May be a rare chance to analyze a problem.

Hard to reproduce in the lab, yet with an obvious fix.

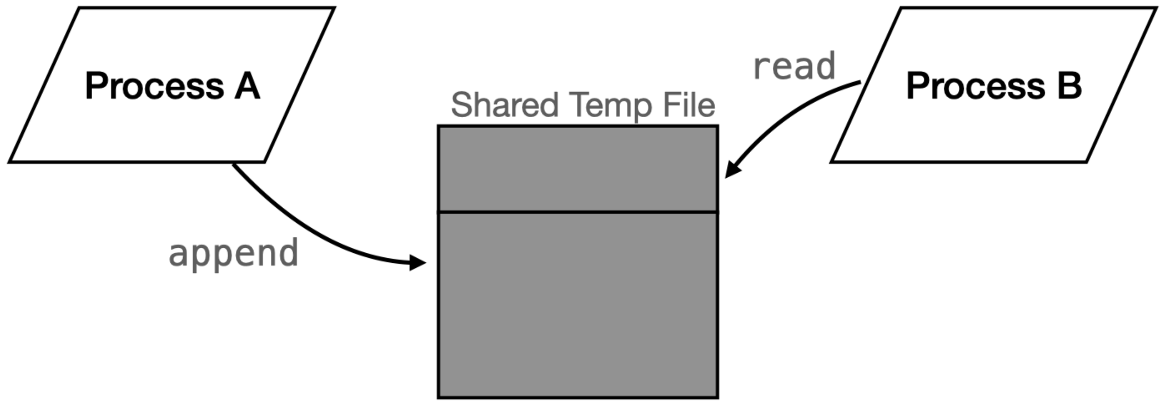
Will occur 1000s of times once product is rolled out!

What can you do?

- Form a hypothesis on the cause of the symptom
- Better error reporting, stricter invariants?

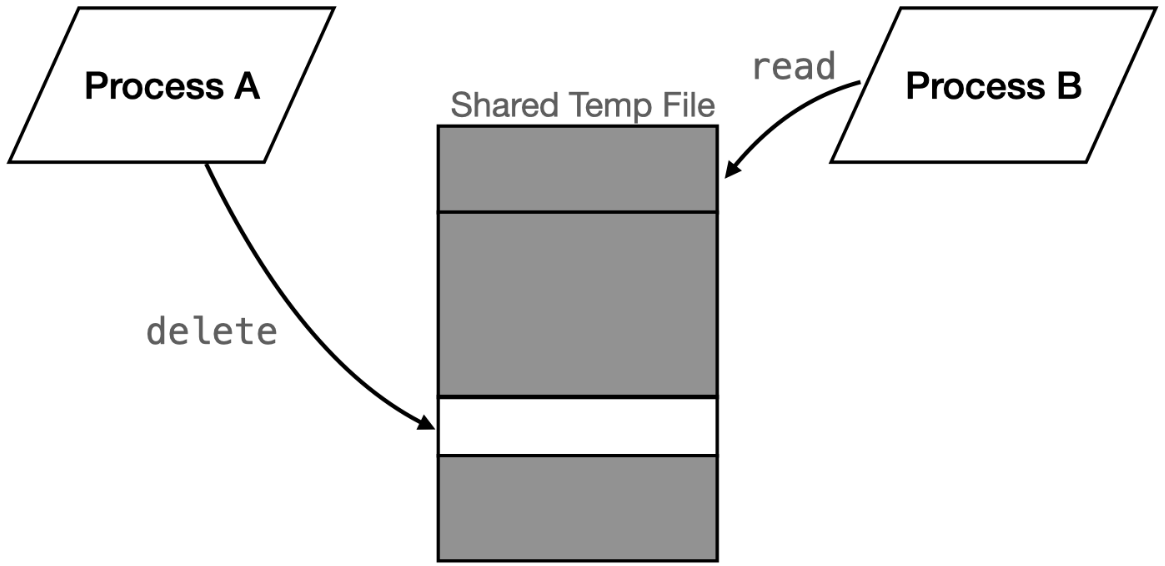
Quiz

Shared temporary data between processes in a single file



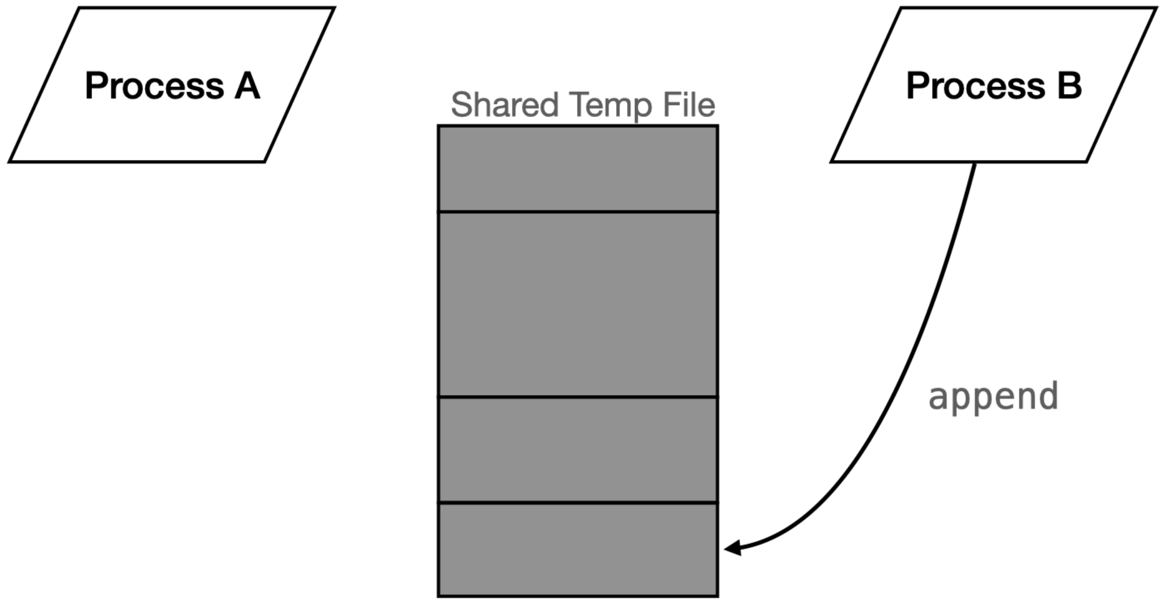
Quiz

Shared temporary data between processes in a single file



Quiz

Shared temporary data between processes in a single file



Quiz

Shared temporary data between processes in a single file

```
int append(std::vector<std::byte> vecbyte) {
  std::scoped_lock lock(lock_on_temp_file());

  CompactSharedTempFileIfNecessary();

  int handleNew = AppendDataToTempFile(vecbyte);
  return handleNew;
}
```

Quiz

Shared temporary data between processes in a single file

```
int append(std::vector<std::byte> vecbyte) {
  std::scoped_lock lock(lock_on_temp_file());

  CompactSharedTempFileIfNecessary();

  int handleNew = AppendDataToTempFile(vecbyte);
  return handleNew;
}
```

```
void read(int handle, void* pv, std::size_t offset, std::size_t count) {
  std::shared_lock lock(lock_on_temp_file());
  CopyDataAtHandleOffset(handle, offset, count, pv);
}
```

Quiz

Shared temporary data between processes in a single file

```
int append(std::vector<std::byte> vecbyte) {
  std::scoped_lock lock(lock_on_temp_file());

  CompactSharedTempFileIfNecessary();

  int handleNew = AppendDataToTempFile(vecbyte);
  return handleNew;
}
```

```
void read(int handle, void* pv, std::size_t offset, std::size_t count) {
  std::shared_lock lock(lock_on_temp_file());
  CopyDataAtHandleOffset(handle, offset, count, pv);
}
```

```
void delete(int handle) noexcept {
  std::shared_lock lock(lock_on_temp_file());
  LookupHandle(handle).m_bDeleted = true;
}
```

Quiz

Shared temporary data between processes in a single file

```
int append(std::vector<std::byte> vecbyte) {
  std::scoped_lock lock(lock_on_temp_file());

  CompactSharedTempFileIfNecessary();

  int handleNew = AppendDataToTempFile(vecbyte);
  return handleNew;
}
```

```
void read(int handle, void* pv, std::size_t offset, std::size_t count) {
  std::shared_lock lock(lock_on_temp_file());
  CopyDataAtHandleOffset(handle, offset, count, pv);
}
```

```
void delete(int handle) noexcept {
  std::shared_lock lock(lock_on_temp_file());
  LookupHandle(handle).m_bDeleted = true;
}
```

Debugging Process

Debugging Process

Debugging is an iterative process

Don't do it fast, do it right.

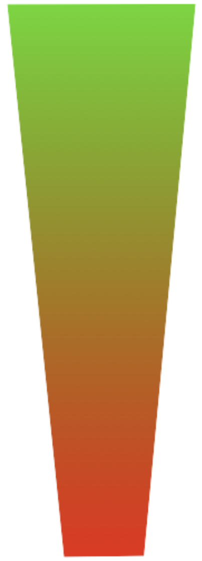
Debugging Process

1. **Bug Report**

2. ***Reproduction***

Reproduction

1. Always reproducible in debug builds, with interactive debugger, on any machine
2. Always reproducible in debug builds, on any machine
3. Sometimes reproducible, in debug, on any machine
4. Always reproducible, in debug, on specific machines
5. Sometimes reproducible, in debug, on specific machines
6. ...
7. ...
8. Sometimes reproducible, only in release builds, only on specific machines



We want to move up!

Reproduction

"Only sometimes reproducible, ..."

Use tools that detect hard-to-reproduce issues:

AddressSanitizer, ThreadSanitizer, UndefinedBehaviorSanitizer

- Is it a timing issue?
- *Careful:* Interactive debuggers make some issues disappear because the code is running too slow.
- Can you write a stress test to force the issue?
- Can you write to a log file and still reproduce the bug?
- Write code to diagnose system when the problem occurs.

Reproduction

"Only reproducible on some machines ..."

- Gather info about environment:
  - OS version, CPU, version of your software, etc
- Anything that could interfere with your program? *Desktop environments are the worst!*
  - Virus scanner blocking files
  - System tools hooking file access
  - System management tools disabling parts of your software
  - Non-standard user rights management
  - DRM software that hooks into your software
- Can you reproduce this environment in a VM? On a client machine? Can the client ship an identical machine?
- Could also be a timing issue. Very slow machine? Very fast? Very busy?

Reproduction

If all else fails:

- Automatically report errors to catch more similar issues (Google CrashPad)
- Write and ship analysis code that tries to nail down the issue
- Try out fixes if you have great reporting
- Can you look at your program state after the problem?
- A file written by your program showing the wrong output?

Quiz

- Customer report: Starting our software fails. According to log file with `std::bad_alloc`.

Quiz

- Customer report: Starting our software fails. According to log file with `std::bad_alloc`.
- Telemetry tells us, loading settings file fails.
- No reproduction but we get the settings file and can look at it.
- Contains bogus data in number formatted string.
- 2^18 times format string `"%a"` instead of once.

Quiz

- Customer report: Starting our software fails. According to log file with `std::bad_alloc`.
- Telemetry tells us, loading settings file fails.
- No reproduction but we get the settings file and can look at it.
- Contains bogus data in number formatted string.
- 2^18 times format string `"%a"` instead of once.
- Format string doubled 18 times?

Quiz

- Customer report: Starting our software fails. According to log file with `std::bad_alloc`.
- Telemetry tells us, loading settings file fails.
- No reproduction but we get the settings file and can look at it.
- Contains bogus data in number formatted string.
- 2^18 times format string `"%a"` instead of once.
- Format string doubled 18 times?
- `std::vector<std::pair<std::size_t, format>>`

Quiz

- Customer report: Starting our software fails. According to log file with `std::bad_alloc`.
- Telemetry tells us, loading settings file fails.
- No reproduction but we get the settings file and can look at it.
- Contains bogus data in number formatted string.
- 2^18 times format string `"%a"` instead of once.
- Format string doubled 18 times?
- `std::vector<std::pair<std::size_t, format>>`
- **Stored in shared memory, shared between different processes**

Quiz

- Customer report: Starting our software fails. According to log file with `std::bad_alloc`.
- Telemetry tells us, loading settings file fails.
- No reproduction but we get the settings file and can look at it.
- Contains bogus data in number formatted string.
- 2^18 times format string `"%a"` instead of once.
- Format string doubled 18 times?
- `std::vector<std::pair<std::size_t, format>>`
- **Stored in shared memory, shared between different processes**
- Vector size `(end - begin) / sizeof(std::pair<std::size_t, format>)`

Quiz

- Customer report: Starting our software fails. According to log file with `std::bad_alloc`.
- Telemetry tells us, loading settings file fails.
- No reproduction but we get the settings file and can look at it.
- Contains bogus data in number formatted string.
- 2^18 times format string `"%a"` instead of once.
- Format string doubled 18 times?
- `std::vector<std::pair<std::size_t, format>>`
- **Stored in shared memory, shared between different processes**
- Vector size `(end - begin) / sizeof(std::pair<std::size_t, format>)`
- How big is `sizeof(std::size_t)`?

Quiz

- Customer report: Starting our software fails. According to log file with `std::bad_alloc`.
- Telemetry tells us, loading settings file fails.
- No reproduction but we get the settings file and can look at it.
- Contains bogus data in number formatted string.
- 2^18 times format string `"%a"` instead of once.
- Format string doubled 18 times?
- `std::vector<std::pair<std::size_t, format>>`
- **Stored in shared memory, shared between different processes**
- Vector size `(end - begin) / sizeof(std::pair<std::size_t, format>)`
- How big is `sizeof(std::size_t)`?
- **Not the same size in 32 bit and 64 bit processes**

Debugging Process

1. **Bug Report**

2. **Reproduction**

3. ***Identify the Problem***

Finding the problem

From Reproduction to Problem Description

- Sometimes symptom does not lead directly to cause
- Problem may have happened earlier & somewhere else in your code

How to find the real problem?

- Again, use all sanitizers!
- And use an interactive debugger!

Finding the problem

Gain understanding of larger system by tracing

- Get an understanding of code being called/order of calls
- Good old `printf` debugging
  - Downside: Requires recompilation
- Better: Use gdb/lldb/Visual Studio tracing breakpoints
  - No recompilation necessary
  - Tracing breakpoints can be added to OS functions, binary code
  - gdb and lldb are powerful
    - Can print stack traces
    - Can be scripted to execute commands automatically, add a large number of breakpoints automatically
    - gdb/lldb even have Python API

Careful: Tracing may make timing-dependent bugs disappear

Finding the problem

Gain understanding of OS interaction by tracing

- Know your OS specific logging tools, e.g.,
  - Windows: ProcessMonitor (https://docs.microsoft.com/en-us/sysinternals/downloads/procmon)
  - macOS: dtrace (http://dtrace.org/blogs/about/)
  - Linux: strace etc (https://linux-audit.com/monitor-file-access-by-linux-processes/)
- Know your Operating System!
  - Semantics of OS primitives
  - File locks, shared memory, virtual memory, file system, I/O, User Interface, Rendering

Finding the problem

Isolate the wrong part of your code

- Do you have a state that still worked?
- Can you find breaking change by binary searching your commits?
    - `git bisect`
    - Understand change to prevent error cycles
- Step through working/broken versions in parallel, where does behavior differ?
- Do the reverse:
    - disable code until bug disappears
    - Again, use "binary search" to track down the problem in least number of steps
- Both require knowledge of code base. Can be very time consuming.

Finding the problem

Improve code to find the problem

- Document invariants: Use asserts a lot
- For complex checks, use temporary asserts to narrow down problems
- **Legacy code?**
  - Introduce safe programming techniques e.g. smart pointers, RAII etc.
  - May fix bugs you haven't even reproduced yet

Finding the problem

think-cell

- Reverse debugging tools
  - let you step backwards through program
  - WinDbg https://www.youtube.com/watch?v=l1YJTg_A914
  - Undo (Linux) https://undo.io
  - rr (Linux) https://rr-project.org
- Know your debugger itself
  - Do you use data breakpoints/watchpoints?
  - Do you put frequently used functionality into debugger scripts?
  - Write debug visualizers for your data types!
- Get at least passive assembly skills

Quiz

```
std::array<int, 4> an = {1, 5, 7, 8};
auto rng = GetItemFromIndices(&an[0], 4);
assert(rng.size()==4); // Fails with rng.size()==2
```

Quiz

```
std::array<int, 4> an = {1, 5, 7, 8};
auto rng = GetItemFromIndices(&an[0], 4);
assert(rng.size()==4); // Fails with rng.size()==2
```

```
lldb> watch set expression -s 4 -- &an[0]
```

Quiz

```
std::array<int, 4> an = {1, 5, 7, 8};
auto rng = GetItemFromIndices(&an[0], 4);
assert(rng.size()==4); // Fails with rng.size()==2
```

```
lldb> watch set expression -s 4 -- &an[0]
```

```
std::byte* p = &an[0];
for(int v=0; v<4; ++v) {
  // do something with p + v * sizeof(int)
}
```

Quiz

```
std::array<int, 4> an = {1, 5, 7, 8};
auto rng = GetItemFromIndices(&an[0], 4);
assert(rng.size()==4); // Fails with rng.size()==2
```

```
lldb> watch set expression -s 4 -- &an[0]
```

```
std::byte* p = &an[0];
for(int v=0; v<4; ++v) {
  // do something with p + v * sizeof(int)
}
```

```
    ldr    x8, [x9, #0x100]
    ldrsw  x10, [x9, #0xf8]
->  add    x8, x8, x10, lsl #3
```

Quiz

```
std::array<int, 4> an = {1, 5, 7, 8};
auto rng = GetItemFromIndices(&an[0], 4);
assert(rng.size()==4); // Fails with rng.size()==2
```

```
lldb> watch set expression -s 4 -- &an[0]
```

```
std::byte* p = &an[0];
for(int v=0; v<4; ++v) {
  // do something with p + v * sizeof(int)
}
```

```
    ldr    x8, [x9, #0x100]     ; get pointer p to an
    ldrsw  x10, [x9, #0xf8]     ; get loop variable v
->  add    x8, x8, x10, lsl #3  ; access p[v<<3] or p[v*8]
```

Quiz

think-cell

```
std::array<int, 4> an = {1, 5, 7, 8};
auto rng = GetItemFromIndices(&an[0], 4);
assert(rng.size()==4); // Fails with rng.size()==2
```

```
lldb> watch set expression -s 4 -- &an[0]
```

```
std::byte* p = &an[0];
for(int v=0; v<4; ++v) {
  // do something with p + v * sizeof(int)
}
```

```
    ldr    x8, [x9, #0x100]     ; get pointer p to an
    ldrsw  x10, [x9, #0xf8]     ; get loop variable v
->  add    x8, x8, x10, lsl #3  ; access p[v<<3] or p[v*8]
```

The buggy code was ported from Windows to macOS.

Quiz

```
std::array<int, 4> an = {1, 5, 7, 8};
auto rng = GetItemFromIndices(&an[0], 4);
assert(rng.size()==4); // Fails with rng.size()==2
```

```
lldb> watch set expression -s 4 -- &an[0]
```

```
std::byte* p = &an[0];
for(int v=0; v<4; ++v) {
  // do something with p + v * sizeof(int)
}
```

```
    ldr    x8, [x9, #0x100]     ; get pointer p to an
    ldrsw  x10, [x9, #0xf8]     ; get loop variable v
->  add    x8, x8, x10, lsl #3  ; access p[v<<3] or p[v*8]
```

The buggy code was ported from Windows to macOS.

`int*` is cast to `long*` somewhere. On Windows, `sizeof(long)==4`. On macOS, `sizeof(long)==8`.

Finding the problem

From reproduction to cause of the bug

Iterative Process: Analysis - Hypothesis - Test - Repeat

Use all tools at your disposal and learn to use them

- debuggers, sanitizers
- reverse debuggers
- OS facilities to capture process traces

Get to know the operating system

Finding the problem

From reproduction to cause of the bug

Iterative Process: Analysis - Hypothesis - Test - Repeat

Use all tools at your disposal and learn to use them

- debuggers, sanitizers
- reverse debuggers
- OS facilities to capture process traces

Get to know the operating system

But most importantly

Finding the problem

From reproduction to cause of the bug

Iterative Process: Analysis - Hypothesis - Test - Repeat

Use all tools at your disposal and learn to use them

- debuggers, sanitizers
- reverse debuggers
- OS facilities to capture process traces

Get to know the operating system

But most importantly

Question your assumptions.

Debugging Process

1. **Bug Report**

2. **Reproduction**

3. **Identify the Problem**

*4. **Classify the Bug***

Classify bug

You didn't write what you meant

- Uninitialised data (e.g. indices?)
- Memory management problem
  - use after free,
  - or rather reference counting bug?
  - use of out-of-scope temporary
- Stack corruption
- Data corruption through missing locks

Often, fix is a local change or use of better programming practices

Classify bug

You didn't write what you meant

- Uninitialised data (e.g. indices?)
- Memory management problem
  - use after free,
  - or rather reference counting bug?
  - use of out-of-scope temporary
- Stack corruption
- Data corruption through missing locks

Often, fix is a local change or use of better programming practices

But: Always check the rest of code base!

Classify bug

You wrote what you meant, but meant wrong

- i.e. your mental model was wrong
- You need a new one. A local fix will not be enough.
- You didn't understand the spec of somebody else's code correctly
  - Wrong use of internal and external API
  - Use of OS facilities that don't work like you thought they did
- You didn't understand your own requirements correctly
  - Is your algorithm correct at all?
  - Is the algorithm the best choice?

Classify bug

How to tell those two cases apart

That is not easy.

Classify bug

How to tell those two cases apart

That is not easy.

```
void do_something_interesting(std::vector<std::unique_ptr<T>> const& vecp) noexcept {
    std::ranges::for_each(vecp, [](auto const& p) noexcept {
        foo(*p);
    });
}
```

Classify bug

How to tell those two cases apart

That is not easy.

```
void do_something_interesting(std::vector<std::unique_ptr<T>> const& vecp) noexcept {
    std::ranges::for_each(vecp, [](auto const& p) noexcept {
        foo(*p); // crashes here with dereferencing null pointer
    });
}
```

Classify bug

How to tell those two cases apart

That is not easy.

```
void do_something_interesting(std::vector<std::unique_ptr<T>> const& vecp) noexcept {
    std::ranges::for_each(vecp, [](auto const& p) noexcept {
        foo(*p); // crashes here with dereferencing null pointer
    });
}
```

```
void do_something_interesting(std::vector<std::unique_ptr<T>> const& vecp) noexcept {
    std::ranges::for_each(vecp, [](auto const& p) noexcept {
        if(p) { // BOOM! Fixed.
          foo(*p);
        }
    });
}
```

Classify bug

How to tell those two cases apart

That is not easy.

```
void do_something_interesting(std::vector<std::unique_ptr<T>> const& vecp) noexcept {
    std::ranges::for_each(vecp, [](auto const& p) noexcept {
        foo(*p); // crashes here with dereferencing null pointer
    });
}
```

```
void do_something_interesting(std::vector<std::unique_ptr<T>> const& vecp) noexcept {
    std::ranges::for_each(vecp, [](auto const& p) noexcept {
        if(p) { // BOOM! Fixed.
          foo(*p);
        }
    });
}
```

Wrong! Or at least, possibly wrong.

Classify bug

How to tell those two cases apart

1. Look at the bigger picture
2. What are you trying to achieve?
3. How are you trying to achieve that?
4. Is that the correct approach?
5. Do you understand all the invariants?
6. Do you assert them in the code?
7. Is the bug violating one of those invariants?

Rethink your assumptions, your mental model!

Debugging Process

1. **Bug Report**

2. **Reproduction**

3. **Identify the Problem**

4. **Classify the Bug**

5. ***Fix the Bug***

Fixing bugs

Smallest possible fix

- Solves the problem
- Does not introduce new bugs

Fixing bugs

Smallest possible fix

- Solves the problem
- Does not introduce new bugs

But

- May not fix the root of the problem
- Or worse, it only hides one instance of the problem.
- May, over time, reduce code quality and make code harder to understand

Fixing bugs

What should the ideal solution look like in an ideal world?

Given everything you know now, how would you solve the problem?

You need to move towards this solution!

Given my constraints, what should I change now?

- Do you need to ship a fix quickly?
- Do you work in an especially secure environment?
- In a very regulated environment?

1. Deliver small fix in the stable build, ship it fast.
2. Attempt thorough fix in development branch.

Fixing bugs

Why did the bug happen in the first place?

- Was it too hard to program correctly, easy to program incorrectly?
- What was missing to program correctly?
  - A library feature or helper function?
  - An algorithm?
  - A standard programming practice?

How can we prevent it from happening again?

- Can you make your fix elsewhere in the code?
  - e.g. introducing smart pointers
  - replacing self-written for-loops with the correct standard algorithm
- Look through your codebase for that pattern!
- Can you introduce the missing abstraction?

Fixing bugs

think-cell

Missing abstractions

```
std::vector<int> vecn;
std::ranges::sort(vecn, std::ranges::less());
auto rng2 = std::ranges::unique(rng1, std::ranges::equal());
```

Fixing bugs

Missing abstractions

```
std::vector<int> vecn;
std::ranges::sort(vecn, std::ranges::less());
auto rng2 = std::ranges::unique(rng1, std::ranges::equal());
```

Note the two different predicates!

They must be compatible! This was done 74 times in our code base, in different ways, some were wrong!

Fixing bugs

Missing abstractions

```
   std::vector<int> vecn;
   std::ranges::sort(vecn, std::ranges::less());
   auto rng2 = std::ranges::unique(rng1, std::ranges::equal());
```

Note the two different predicates!

They must be compatible! This was done 74 times in our code base, in different ways, some were wrong!

Replaced by

```
template<typename Rng, typename Less = tc::fn_less>
auto sort_inplace_unique_range(Rng&& rng, Less&& less)
```

Fixing bugs

Missing abstractions

Make correct error handling easy, convenient and mandatory!

```
if (auto const ohfile = ERRNORETRYIGNORE(
      open(file, ...)
      tc::err::returned_nonnegative_value(), // success
      tc::err::returned(invalid_filehandle, EINTR), // retry
      tc::err::returned(invalid_filehandle, {EPERM, ENOENT, EACCES}) // allowed errors
    ))
{}
```

Fixing bugs

Missing abstractions

Make correct error handling easy, convenient and mandatory!

```
if (auto const ohfile = ERRNORETRYIGNORE(
      open(file, ...)
      tc::err::returned_nonnegative_value(), // success
      tc::err::returned(invalid_filehandle, EINTR), // retry
      tc::err::returned(invalid_filehandle, {EPERM, ENOENT, EACCES}) // allowed errors
    ))
{}
```

```
RECT rect;
if(0==GetClientRect(wnd, &rect)) {
  Report(GetLastError(), std::source_location::current());
}
```

Fixing bugs

think-cell

Missing abstractions

Make correct error handling easy, convenient and mandatory!

```
if (auto const ohfile = ERRNORETRYIGNORE(
      open(file, ...)
      tc::err::returned_nonnegative_value(), // success
      tc::err::returned(invalid_filehandle, EINTR), // retry
      tc::err::returned(invalid_filehandle, {EPERM, ENOENT, EACCES}) // allowed errors
    ))
{}
```

```
RECT rect;
if(0==GetClientRect(wnd, &rect)) {
  Report(GetLastError(), std::source_location::current());
}
```

```
RECT rect;
APIERR(GetClientRect(wnd, &rect));
```

Debugging Process

1. Bug Report
2. Reproduction
3. Identify the Problem
4. Classify the Bug
5. Fix the Bug
6. *Deliver the Fix*

Delivering Fix

- Documentation in the code and outside the code
    - Write high-level documentation to explain concepts
    - Documentation in source files goes *less* out of date
    - Document what, when, who and why you did something
    - Can you reference an issue in your bug tracker?
    - The next developer may ask "Do we still need this?"
- Code Reviews
    - or other collaborative practices like pair programming
    - explain what you did to others
    - often, errors in your thinking become obvious, once you have to spell it out

Delivering Fix

- Good version control practices
  - Split changes into self-contained chunks
  - Separate refactors from changes to functionality
  - Do refactor during debugging!
- Tests
  - May also prevent a regression from ever happening again
  - Tests also need to be well-written
  - Run automatically ideally
  - Trivial test cases are useless
  - **Can you test random input?**

Debugging Process

1. Bug Report
2. Reproduction
3. Identify the Problem
4. Classify the Bug
5. Fix the Bug
6. Deliver the Fix

Thank you!

And yes, we are recruiting: `hr@think-cell.com`

Quiz

```
enum EState { WAITING, DATA, ERROR };

struct http_delegate {
  EState m_estate = WAITING;

  std::mutex m_mtx;
  std::condition_variable m_cv;

  void on_new_data() {
    {
      std::lock_guard lock(m_mtx);
      m_estate = is_error() ? ERROR : DATA;
      // copy data to buffer
    }
    m_cv.notify_all();
  }
};
```

Quiz

```
enum EState { WAITING, DATA, ERROR };

struct http_delegate {
  EState m_estate = WAITING;

  std::mutex m_mtx;
  std::condition_variable m_cv;

  void on_new_data() {
    {
      std::lock_guard lock(m_mtx);
      m_estate = is_error() ? ERROR : DATA;
      // copy data to buffer
    }
    m_cv.notify_all();
  }
};
```

```
struct sync_http_request {
  http_delegate m_delegate;

  sync_http_request(...) {
    // set up everything

    std::unique_lock lock(m_delegate.m_mtx);

    m_delegate.m_cv.wait(lock,
      [&]() {
        return m_delegate.m_estate!=WAITING;
      });

    if(m_delegate.m_estate==ERROR)
      throw http_exception();

    // do something with the data
  }
};
```

Quiz

think-cell

```
enum EState { WAITING, DATA, ERROR };

struct http_delegate {
  EState m_estate = WAITING;

  std::mutex m_mtx;
  std::condition_variable m_cv;

  void on_new_data() {
    {
      std::lock_guard lock(m_mtx);
      m_estate = is_error() ? ERROR : DATA;
      // copy data to buffer
    }
    m_cv.notify_all();
  }
};
```

```
struct sync_http_request {
  http_delegate m_delegate;

  sync_http_request(...) {
    // set up everything

    std::unique_lock lock(m_delegate.m_mtx);

    m_delegate.m_cv.wait(lock,
      [&]() {
        return m_delegate.m_estate!=WAITING;
      });

    if(m_delegate.m_estate==ERROR)
      throw http_exception();

    // do something with the data
  }
};
```

Quiz

```cpp
enum EState { WAITING, DATA, ERROR };

struct http_delegate {
  EState m_estate = WAITING;

  std::mutex m_mtx;
  std::condition_variable m_cv;

  void on_new_data() {
    {
      std::lock_guard lock(m_mtx);
      m_estate = is_error() ? ERROR : DATA;
      // copy data to buffer
      m_cv.notify_all();
    }
  }
};
```

```cpp
struct sync_http_request {
  http_delegate m_delegate;

  sync_http_request(...) {
    // set up everything

    std::unique_lock lock(m_delegate.m_mtx);

    m_delegate.m_cv.wait(lock,
      [&]() {
        return m_delegate.m_estate!=WAITING;
      });

    if(m_delegate.m_estate==ERROR)
      throw http_exception();

    // do something with the data
  }
};
```