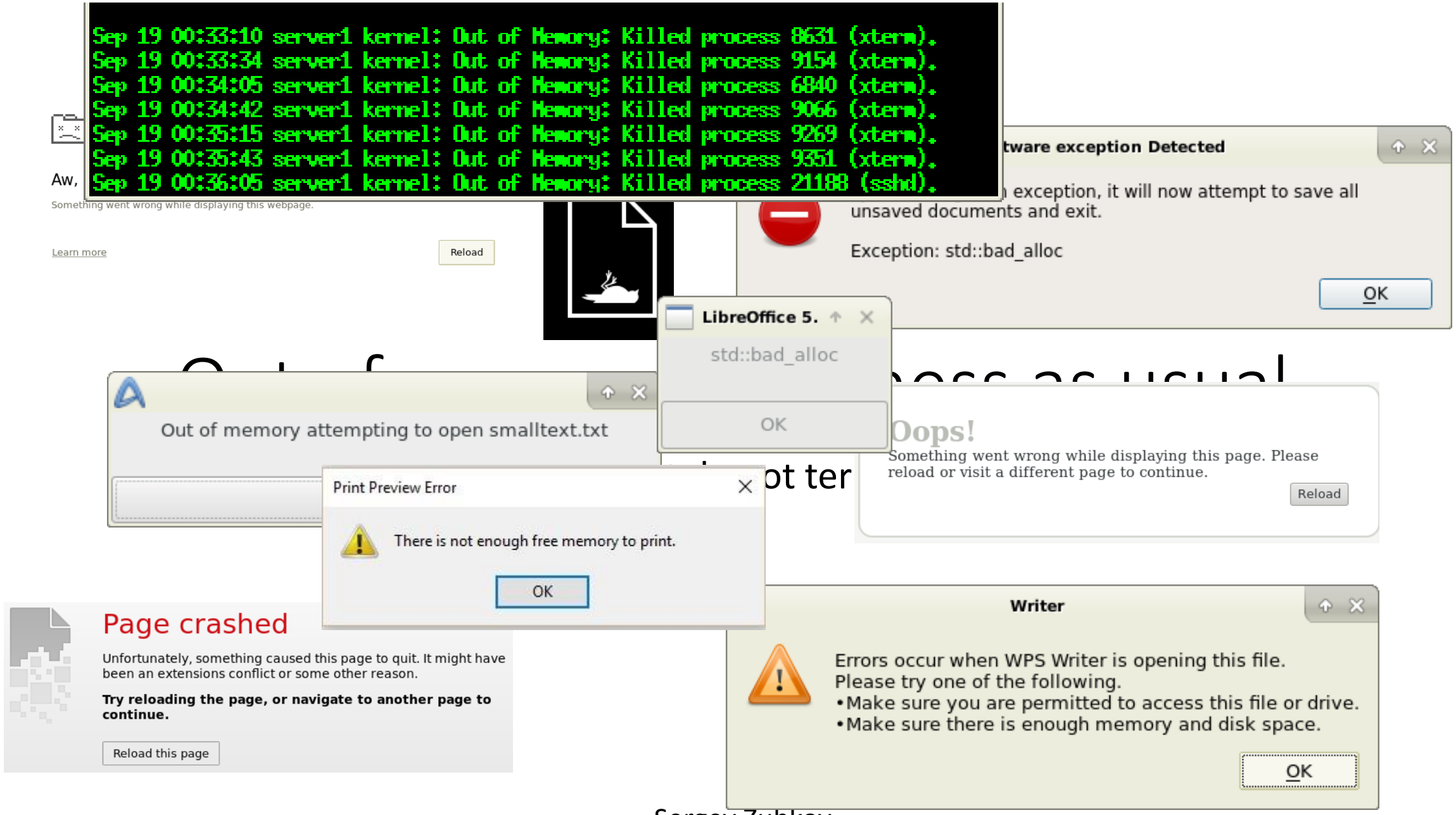
Out of memory? Business as usual.

Examining applications that do not terminate on std::bad_alloc

Sergey Zubkov



Sergey Zubkov

What is this about?

1. What is “out of memory” and what is “bad allocation”
2. What do those who catch std::bad_alloc do with it
3. How common (or rare) are these applications

All resources are not created equal

- “Resource” == “limited availability” (Wikipedia, cppreference)
- Assumed resources
  - CPU time/cores/caches, Network bandwidth, RNG entropy, Electricity, **Stack memory**
- Checked resources
  - Disk space, other hardware, file/socket descriptors, threads, locks, other software, **Heap memory**

Resource acquisition can fail: constructors can throw. What do they throw in stdlib?

- std::system_error:

  thread, unique_lock, shared_lock, maybe shared_ptr

  basic_socket (Network TS), display_surface (Graphics TS)
- std::bad_alloc:

  any, shared_ptr, function, boyer_moore_(horspool_)searcher, basic_string, (forward_)list, vector, deque, (unordered_)(multi)(map|set), stack, (priority_)queue, valarray, basic_(i|o)stringbuf, stringstream, , (un)synchronized_pool_resource, path, directory_entry, (runtime_|logic_|etc)error, basic_regex, match_results, promise, packaged_task, …

Common wisdom

- Effective C++ (2nd ed. Item 7, 2003, no longer in 3rd ed.)

  “Regardless of whether you use "normal" (i.e., exception-throwing) new or "nothrow" new, it's important that you be prepared to handle memory allocation failures.”

- Sutter’s Mill “To new, perchance to throw part 2”, 2001

  “except for special cases, even when you detect new failure there's not always much you can do if there really is no memory left.“

- DIP-33 rationale by Walter Bright, 2013

  “I've almost never seen a program that could successfully recover from out of memory errors, even ones that purport to.”

- CppCoreGuidelines rule F.6, 2016

  “after memory runs out it is hard to do anything clever […] the majority of programs and execution environments cannot meaningfully handle a failure to allocate”

What does “Out of memory” mean?

- “Memory” means page-based virtual memory.

  Process sees a homogeneous address space, but each page may be private or shared, read-only or (COW-)writeable, clean or dirty, resident or paged-out

- Unused memory is wasted memory

  Operating systems swap or reclaim inactive pages for buffers/caches.

- *Commit charge*: all writeable pages that are not file-backed (stack, data, heap, private mmaps, shared library .GOT’s)
- If commit charge exceeds free RAM + free swap, it’s OOM... or is it?

The fork/exec problem

- fork() duplicates the entire process memory, exec() throws it away

```
$ ./test 10
allocating 10737418240 bytes (10GB)...Allocated
parent: fork ok
child: fork ok
$ ./test 20
allocating 21474836480 bytes (20GB)...Allocated
fork: Not enough space
```

- Not an issue on Windows
- Alternatives (vfork and spawn) exist, but have their own issues

Overcommit

Operating systems with strict commit accounting:

- Windows, Solaris, HP/UX, and more;

  Note: private COW mappings are fully accounted; fork() can OOM

Operating systems with overcommit and OOM killer:

- AIX: per-process opt-out by installing SIGDANGER handler
- FreeBSD: can be turned off systemwide, process can opt-out with protect(1)
- Linux: three systemwide settings: always, never, and heuristic (+oom_adj, +cgroups).

Some Linux users swear by it, some swear it off

Note: in “always overcommit” mode, rlimits and address space limits will still throw

Note: in “never overcommit”, kernel reserves RAM to fork a shell/top/kill

Note: neither mode is the default: the default is heuristic

Common ~~wisdom~~myths

Sutter's Mill "To new, perchance to throw part 2" (and many others)

> "checking for new failure isn't as important as one might think … On some operating systems, including specifically Linux, memory allocation always succeeds."

LevelDB issue #335 "Question about exception safety"

> "on Linux, you will only get a std::bad_alloc thrown if the virtual address space has been exhausted"

Note: the Linux default is not always-overcommit

Bad allocation is not always OOM

This is a bad allocation:

```
std::vector<int>(-1); // always throws (including Linux with overcommit_always)
```

That's clearly programmer's fault. The first unit test will catch it. Why care?

```
std::vector<int>(content_length); // do you trust Content-Length:?
```

How hard can it be to check your inputs?

- CVE-2016-2109 OpenSSL OOM DoS due to short invalid encoding
- CVE-2016-2463 Android OOM DoS via crafted media file
- CVE-2016-6170 ISC BIND OOM DoS due to large UPDATE message
- CVE-2015-7540 samba AD-DC OOM DoS via crafted packets
- CVE-2015-1819 libxml OOM DoS via crafted XML file
- CVE-2014-3506 OpenSSL OOM DoS via crafted DTLS handshake
- CVE-2013-7447 cairo OOM DoS via crafted image file

Aren’t all those CVEs against C libs?

`malloc` returns NULL, but how do you inform the caller, possibly many stack frames up?

- Some meticulously return error codes
- Some longjmp
- Some gave up:

```
mem = malloc (n_bytes);
TRACE (GLIB_MEM_ALLOC((void*) mem, (unsigned int) n_bytes, 0, 0));
if (mem)
  return mem;

g_error ("%s: failed to allocate %"G_GSIZE_FORMAT" bytes",
         G_STRLOC, n_bytes);
```

https://github.com/GNOME/glib/blob/master/glib/gmem.c#L87-L106

What would C++ do?

```
if (reinterpret_cast<const uint8_t*>(&_isize) == buffer) {
    // Read length of data to follow
    try {
        _isize = ntohl(_isize);
        if (0 == _isize || _isize > MAX_XRL_INPUT_SIZE)
            throw bad_alloc();
        _input_buffer.resize(_isize);
    } catch (bad_alloc) {
        XLOG_ERROR("Bad input buffer size (%d bytes) from wire, "
                   "dropping connection", XORP_INT_CAST(_isize));
        error_event();
        return;
    }
```

https://github.com/greearb/xorp.ct/blob/master/xorp/libxipc/finder_tcp.cc#L161-L173

“One Bug To Rule Them All” (GSEC-TZO-44-2009 aka CVE-2009-1692)

IE5, IE6, IE7, IE8, Netscape, Firefox, Safari, Opera, Konqueror, Seamonkey, Wii, PS3, iPhone, iPod, Nokia, Siemens.... and more

- Konqueror (Ubuntu): allocates 2GB of memory then either crashes the browser or (most often) the OS reboots.
- Chrome :allocates 2GB of memory then crashes tab
- Firefox : allocates 2GB of memory then the browser crashes
- IE5,6,7,8 : allocates 2GB of memory then the browser crashes
- Opera : will not crash but other applications will become unstable
- Nintento WII (Opera) : Console hangs, needs hard reset
- Sony PS3 - Console hangs, needs hard reset
- iPhone - iPhone hangs and needs hard reset

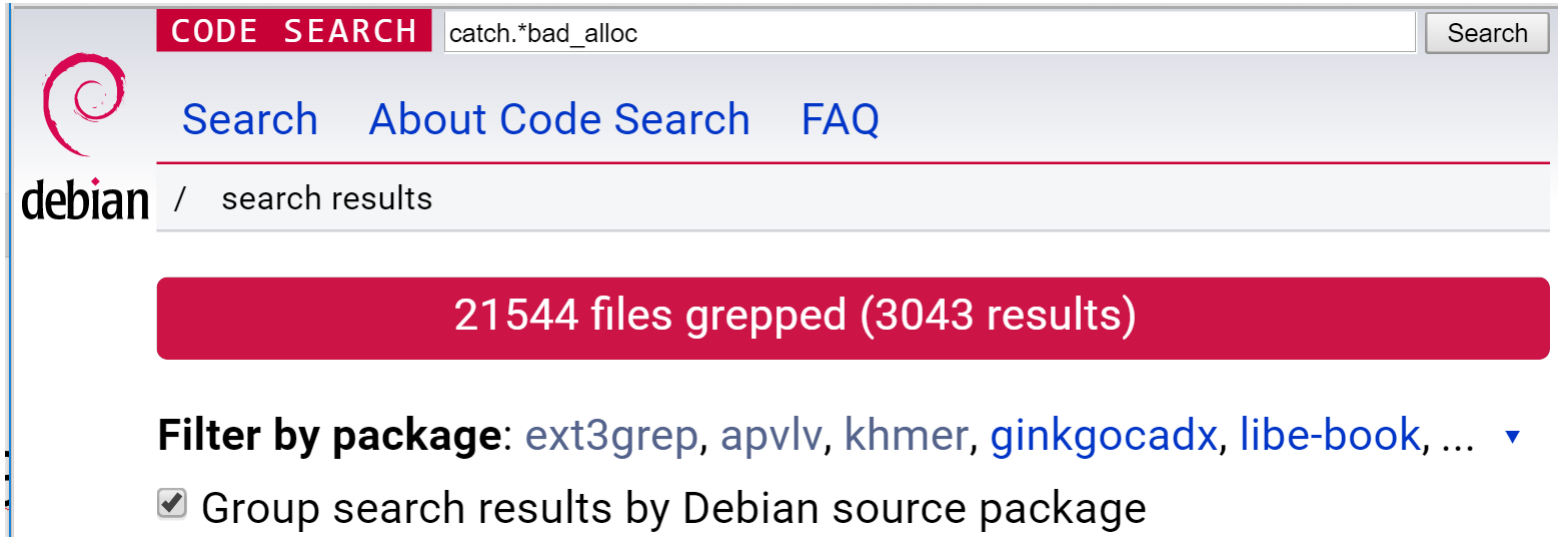
What was this awful bug?

```
e = document.createElement("select");
e.length=2147483647;
```

Just one of infinite ways by which JavaScript can use memory

Does anyone really handle std::bad_alloc?

Debian Code Search: `codesearch.debian.net/search?q=catch.*bad_alloc`



Not `std::exception` or `catch(...)`, only explicit handling for `std::bad_alloc`:

3043 results in 341 packages.

Disclaimer: I can't promise I fully understood each of those 3000+ pieces of opensource code

178 (46%) Somebody else's problem

92 (23%) Convert to error code (either returned or stored as a flag)

gdal qscintilla libsdl2 rtaudio armadillo libsdc++ breakpad ACE vtk coinutils …

50 (13%) Convert to custom exception

dlib ipopt poco povray gdal libreoffice capnproto galera-3 libosl gnudatalanguage …

20 (5%) Convert to OOM error in a different language (`PyErr_NoMemory`)

boost cython pyclingo matplotlib openjdk libreoffice openvrml healpy pytaglib …

16 (4%) Rethrow as-is

mp3diags 3dldf timbl deqp cxxtools orthanc-postgresql duma synergy mrpt madness …

Somebody else's problem
92 (23%) Convert to error code

Many libraries return error codes

```
try {
        int position = Length();
        InsertString(position, data, length);…
} catch (std::bad_alloc &) {
        return SC_STATUS_BADALLOC;
```

https://github.com/mirror/scintilla/blob/master/src/Document.cxx#L1056-L1060

the caller can handle this generically:

```
if (_pscratchTilla->execute(SCI_GETSTATUS) != SC_STATUS_OK)
        throw;
```

https://github.com/notepad-plus-plus/notepad-plus-plus/blob/master/PowerEditor/src/ScitillaComponent/Buffer.cpp#L1423-L1424

Some update global or member variables instead

```
__try
  { __words = new (std::nothrow) _Words[__newsize]; }
__catch(const std::bad_alloc&)
  { __words = nullptr; }
if (!__words)
  {
    _M_streambuf_state |= badbit;
```

https://github.com/gcc-mirror/gcc/blob/master/libstdc++-v3/src/c++11/ios.cc#L127-L133

Somebody else's problem
50 (13%) Convert to custom exception

```
try
{
        if (aPosition >= _bindVector.size())
        {
                _bindVector.resize(aPosition + 1);
        }

        InputParameter inputParameter(aFieldType, aBufferPtr, aLength);

        _bindVector[aPosition] = inputParameter;
}
catch (std::bad_alloc&)
{
        PostgreSQLException("Memory allocation error while binding");
}
```

https://github.com/pocoproject/poco/blob/develop/Data/PostgreSQL/src/Binder.cpp#L333-L347

Somebody else's problem
16 (4%) Rethrow as-is

Clean up non-RAII resources

```
        m_allocations.push_back(ptr);
}
catch (std::bad_alloc& )
{
        delete [] ptr;
        throw;
```

https://github.com/zenoalbisser/chromium/blob/master/third_party/deqp/src/framework/referencerenderer/rrVertexPacket.cpp#L64-L72

Handle other exceptions differently

```
catch( const std::bad_alloc& e )
{
    throw e;
}
catch(...)
{
    _allocator.deallocate(_instance, 1);
    _instance = 0;
    throw;
}
```

https://github.com/deniskin82/cxxtools/blob/master/include/cxxtools/singleton.h#L85-L94

82 (21%) Cleanup and terminate

- 45 (12%) Not from main

  rethinkdb ipopt fluxbox lzip shogun thrift polyml krita pingus bowtie …

- 32 (8%) From main

  tripwire smartmontools tango timbl taskd ppl ossim mame dwarfutils …

- 3 (1%) Configurable handler defaulting to abort

  igraph r-cran-igraph gdal

- 1 (0%) “Parachute“ (as featured in Code Complete)

  scantailor

82 (21%) Cleanup and terminate

"Cleanup" often means removing temp or lock files

```
} catch (const std::bad_alloc &) {
    unlink_ofile(oname);
    printErr(iname,"out of memory");
    e_exit(EXIT_ERROR);
```

https://github.com/ferseiti/upx-ucl/blob/master/src/work.cpp#L310-L313

Termination from deep in a library may have major impact:

```
void* fastMalloc(size_t n)
{
    void* result = malloc(n);
    if (!result)
        CRASH();
```

https://github.com/WebKit/webkit/blob/master/Source/WTF/wtf/FastMalloc.cpp#L131-L135

35 (9%) Moving forward

- 10 (3%) Try allocating less (smaller audio buffer, on-the-fly calcs)

  audacity eigen3 frobby tuvok lammps libreoffice mira openttd spring vxl

- 10 (3%) Swallow bad_alloc (in a destructor, adding to cache)

  frobby jade libreoffice love poco ncbi-blast+ opencv openvrml ossim otb

- 7 (2%) Alternative algorithm (in-place instead of out-of-place a-la STL)

  vtk aseprite bowtie bowtie2 krita mlpack octave

- 6 (2%) Free up some memory (drop caches, cannibalize freelists)

  libstdc++ mrpt sonic-visualizer scylladb diagnostics libosl

- 2 (1%) Just try again (???)

  gnuradio infinidb

35 (9%) Moving forward
7 (2%) In-place algorithm

```
bool inline inplace_transpose(arma::Mat<eT>& X)
{
  try
  {
    X = arma::trans(X);
    return false;
  }
  catch (std::bad_alloc&)
  {
#if (ARMA_VERSION_MAJOR >= 4) || \
    ((ARMA_VERSION_MAJOR == 3) && (ARMA_VERSION_MINOR >= 930))
    arma::inplace_trans(X, "lowmem");
    return true;
```

https://github.com/mlpack/mlpack/blob/master/src/mlpack/core/data/load_impl.hpp#L65-L77

68 (18%) Roll back and do something else

- 47 (12%) Interactive apps refusing user actions (“Could not open file”)

  notepad-plus-plus libreoffice lfhex texstudio inkscape spring povray …
- 18 (5%) Servers dropping service requests
  - Network servers
    - apt-cacher-ng ntopng dc-qt dlib folly resiprocate xorp
  - Databases and other servers
    - clamav csound dindel glogg libclasp ring scylladb tarantool
  - Batch processors
    - clblas seqan ssdeep undertaker
- 3 (1%) Prepared fallback (Error texture, “NoSound” sound driver)

  0ad aiksaurus desmume

68 (18%) Roll back and do something else
47 (12%) UI and other interactive apps

"File too big to load" is the most common reason for this

```
                        docPtr = Poppler::Document::load(fileName);
        }
} catch (std::bad_alloc) {
        error = PopplerErrorBadAlloc;
        return QSharedPointer<Poppler::Document>();
```

TexStudio/blob/master/pdfrendermanager.cpp#L119-L123

And plenty other reasons

```
catch(vtkstd::bad_alloc &)
{
  throw IRISException("Out of memory during mesh computation");
```

```
catch(IRISException & IRISexc)
 {
 QMessageBox::warning(this, "Problem generating mesh", IRISexc.what());
```

https://github.com/pyushkevich/itksnap/blob/master/GUI/Model/Generic3DModel.cxx#L232-L234

68 (18%) Roll back and do something else
18 (5%) Servers dropping service requests

```
try {
  pass_verdict = processPacket(&h->ts, time, ethernet, vlan_id, iph,
                               ip6, h->caplen - ip_offset, h->len,
                               h, packet, shaped, ndpiProtocol);
} catch(std::bad_alloc& ba) {
```

https://github.com/ntop/ntopng/blob/dev/src/NetworkInterface.cpp#L1393-L1397

```
catch (std::bad_alloc &)
{
    std::cout << shapeStrings[i] << " threw bad_alloc exception, skipping this shape." << std::endl;
    continue;
}
```

https://github.com/seqan/seqan/blob/master/apps/razers/paramChooser.h#L675-L679

25 (6%) Unit tests

- Unit tests

  gdcm libstdc++ boost angle cmtk trilinos dune-common dune-istl eigen3 folly isc-kea libboost-geometry-utils-perl libc++ libloki libpqxx pugixml mia ossim ppl pugixml tbb vxl xapian-core z3 scylladb

```
ASSERT_THROWS_IN_TEST(
    {
        limit_foo_count_in_scope foo_limit(FooCount + planned_victim_size);
        victim = std::move(fixture.source); // fragmented assignment
    },
    std::bad_alloc, "", test_name
);
```

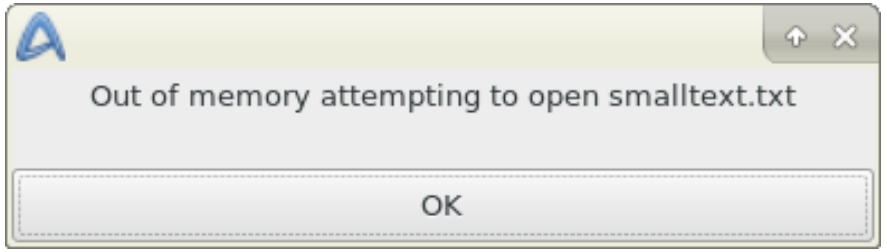
https://github.com/wjakob/tbb/blob/master/src/test/test_concurrent_vector.cpp#L1407-L1413

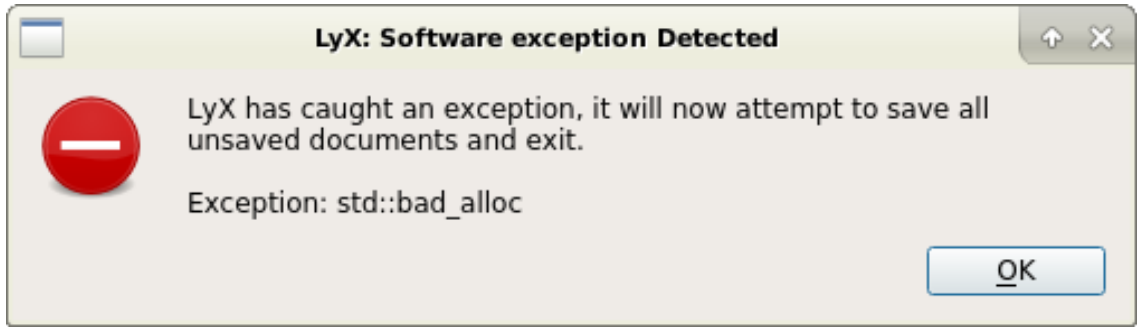
Case studies or “crash everything”

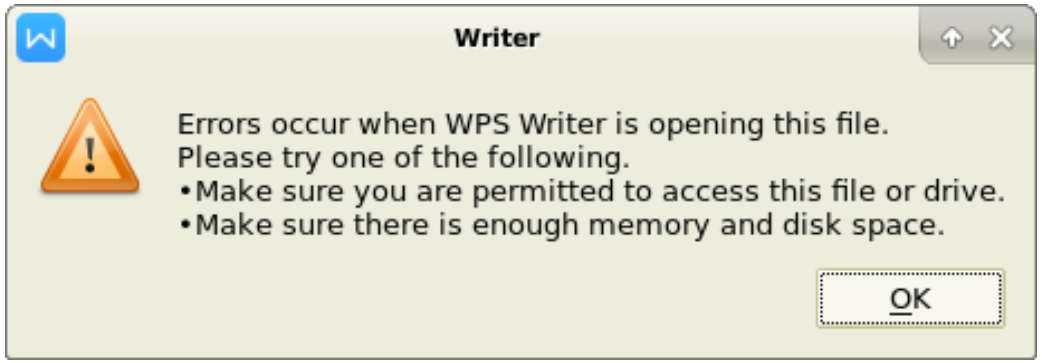
We’ve seen what people do with std::bad_alloc when they catch it, but how common is that, really?

C++ OOM Case study: Office applications

App	Large file	Large paste
Abiword	Survives	Uncaught bad_alloc
calligra	Survives	Survives, seems flaky
lyx	Save and quit	Save and quit
texmaker	Uncaught	Uncaught
WPS office	Survives	Survived, then crashed
libreoffice	Save and survive	abort
openoffice	Uncaught	Segfault
texstudio	Qt handler abort	Survives
texmacs	Qt handler abort	Survives
scribus		Uncaught
qchartdiary		Qt handler abort
kmymoney		Qt hanlder abort
kraft		Qt abort or size limit





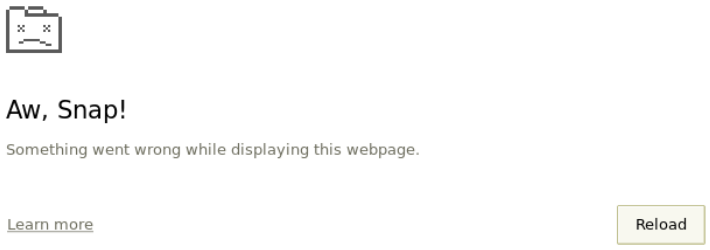


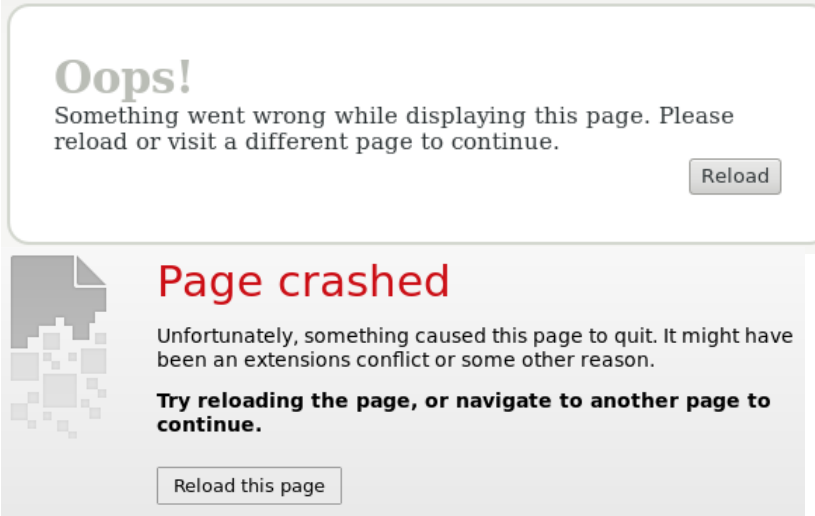
C++ OOM Case study: Code Editors/IDEs

App	Large file	Large paste
Sublime	Quietly exits	Saves and quietly exits
Nodepad++	Survives because of scintilla	
Codeblocks	Uncaught bad_alloc	Glib crash or segv, scintilla doesn't save this one
Geany	Glib crash	Survives because of scintilla, menus etc all work.
Leechcraft	Quietly refuses to load, survives	Survives because of scintilla (but crashed once)
U++ TheIDE	Warning popup and exit	Warning popup and exit
SciTE	Uncaught exception	Uncaught exception
Pikdev	Uncaught bad_alloc	Survives, without Scintilla's help; no bad_alloc
Sasm	Uncaught bad_alloc	Qt handler crash
Kdevelop, Monkeystudio	Qt handler crash	Qt handler crash
Kate	Uncaught bad_alloc	Uncaught bad_alloc
Qt Creator	segfault	Uncaught bad_alloc
CodeLite	Glib crash	Segfault
Rstudio	Imposes a file size limit	Webkit crash

C++ OOM Case study: Web browsers

App	Large page	JavaScript OOM
One tab per process:	tab crash	tab crash
Chromium, Epiphany, Opera, Vivaldi, Qupzilla		
Mozilla family:	Usually stops download, but sometimes crashes in glib or with “unhandlable oom while tenuring”	Stops the script, but doesn’t free (despite SpiderMonkey’s two-level OOM callbacks)
Firefox, Seamonkey, Conkeror		
Webkit family:	Webkit crash	Stops the script, but doesn’t free: further browsing quickly gets webkit crash or QThread::start failure
QtWeb, dwb, Otter		
leechcraft, rekonq, dillo, dooble	crash	crash





C++ OOM Case study: Databases

App		
scylladb	Survives…	except in future::then (open issue)
tarantool	Survives…	except when lua OOMs
rethinkdb, rocksdb, levledb, kyoto, mongodb, etc		crash (some easier than others), but users complain:

(Note: there are other databases that don't crash – this slide is incomplete)

https://github.com/google/leveldb/issues/335

"I found many palces in leveldb source code that bad_alloc will break the Ref/Unref balance."

https://github.com/rethinkdb/rethinkdb/issues/599

"A truly robust solution would deal with an out of memory condition with a response less drastic than killing the server, say by cancelling the offending query and freeing any memory associated with it"

Handling allocation failure

- Users want it (even Linux users)
  - When data loss, expensive recalculation, or service disruption may occur
- Consistent RAII makes it possible
  - just don't leak another bad_alloc from a destructor
- Don’t be perfect: heterogeneous memory handling strategies work
  - extreme case: preallocate everything
- Don't underestimate the importance of libraries
  - a library can break every user (glib, webkit)
  - or help every user (scintilla)