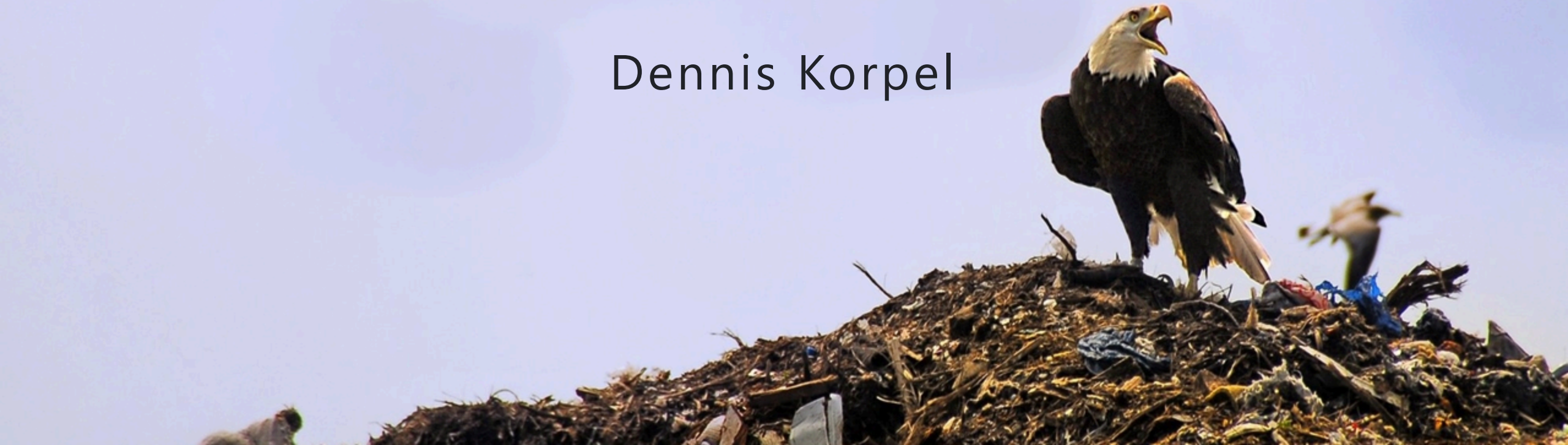
Avoid the Garbage Collector in 80 lines

Dennis Korpel



Garbage Collection

Memory is automatically managed by occasionally pausing all threads and scanning for memory still in use, and freeing the rest.*

*we'll get back to this

Phobos API

```
import std.stdio, std.process;

void printPath()
{
    string s = environment.get("PATH");
    writeln(s);
}
```

Windows API

```
import std.stdio, core.sys.windows.windows;

void printPath()
{
    const lengthZ = GetEnvironmentVariableW("PATH", null, 0);
    wchar[] buf = new wchar[lengthZ];
    const len = GetEnvironmentVariableW("PATH", buf.ptr, buf.length);
    writeln(buf[0 .. len]);
}
```

https://github.com/dkorpel/dconf

Conclusion

Good thing D has a Garbage Collector 👍

Except...

- Some people don't like the GC
- I tried `@nogc` approaches so you don't have to!
- Often awkward
  - until epiphany:
- `@safe @nogc` allocator in just 80 lines
- Built on top of `malloc` and `free`

https://github.com/dkorpel/dconf

https://github.com/dkorpel/dconf

BLUF (bottom line up front)

```d
import core.memory, core.bitop;

Allocator gc() => Allocator(null);

struct Allocator
{
    AllocatorBase* x;

    T[] array(T)(size_t length) return scope @trusted if (__traits(isPOD, T))
    {
        if (x == null || __ctfe)
            return new T[length];
        return cast(T[]) x.allocate(T.sizeof * length, T.alignof, x);
    }
}

alias AllocateFunction = ubyte[] function(size_t size, size_t alignment, scope void*

struct AllocatorBase
{
    in
}

struct Arena
{
    @system private AllocatorBase base = AllocatorBase(&arenaAllocate);
    @system private ArenaPage* page = null;
    @system private ubyte[] buffer; // slice of free space

    private static ubyte[] arenaAllocate(size_t size, size_t alignment, scope void*
        (cast(Arena*) ctx).allocate(size, alignment);

    @disable this(this);
    @disable void opAssign();

    this(return scope ubyte[] initialBuffer) scope @trusted
    {
        this.buffer = initialBuffer;
    }
```

```d
    ubyte[] allocate(size_t size, size_t alignment) scope @trusted
            newPage(size);
        else
            this.buffer = this.buffer[shift .. $];
        auto result = this.buffer[0 .. size];
        this.buffer = this.buffer[size .. $];
        return result;
    }

    private void newPage(size_t size) @trusted
    {
        const newSize = size_t(1) << (1 + bsr(ArenaPage.sizeof + size));
        assert(newSize >= size);
        auto p = pureMalloc(newSize);
        GC.addRange(p, newSize, null);
        assert(p);
        auto oldPage = this.page;
        this.page = cast(ArenaPage*) p;
        this.page.prev = oldPag
        this.buffer = cast(ubyte[]) p[ArenaPage.sizeof .. newSize];
    }

    Allocator alloc() scope return @trusted => __ctfe ? gc() : Allocator

    ~this() scope @trusted
    {
        while (this.page)
        {
            void* toFree = this.page;
            this.page = this.page.prev;
            pureFree(toFree);
        }
    }
}

private struct ArenaPage
{
    ArenaPage* prev;
}
```

```d
string environmentGet(string key, return scope Allocator alloc = gc)
{
    // return new char[length];
    return alloc.array!char(length);
}

void main()
{
    Arena a;

    string s = environmentGet("PATH", a.alloc);
    writeln(s);
    writeln(environmentGet("TMP", a.alloc));

    // a.~this();
}
```

Whoami

- Msc. Computer Science TU Delft
- Part time Issue Manager for D Language Foundation
- Part time D programmer at SARC

Coming up

- On GC avoidance
- On simplicity
- 6 suboptimal `@nogc` approaches
- The 80 line solution

On GC avoidance

The GC is controversial

- I find myself on neither side of the debate
- "GC makes D bad for real-time apps"
- "But there's `@nogc` "
- "But then you lose most of Phobos"
- Perhaps use Reference Counting in Phobos?

https://github.com/dkorpel/dconf

(Automatic) Reference counting

```
struct RefCountedString {
    string* payload;
    int* count;

    this(string s) {
        payload = malloc(s.length);
        count = new int(1);
    }

    this(this) { ++*count; }

    ~this() {
        if (--*count == 0) free();
    }
}
```

https://github.com/dkorpel/dconf

Example: 🎹 Audio programming

```
float phase = 0;

void audioCallback(float[] buffer)
{
    foreach (i; 0 .. buffer.length)
    {
        buffer[i] = sin(phase);
        phase += 0.0576;
    }
}
```

48 Khz sample rate, 10 ms latency ⇒ 480 samples

Garbage collector comes!

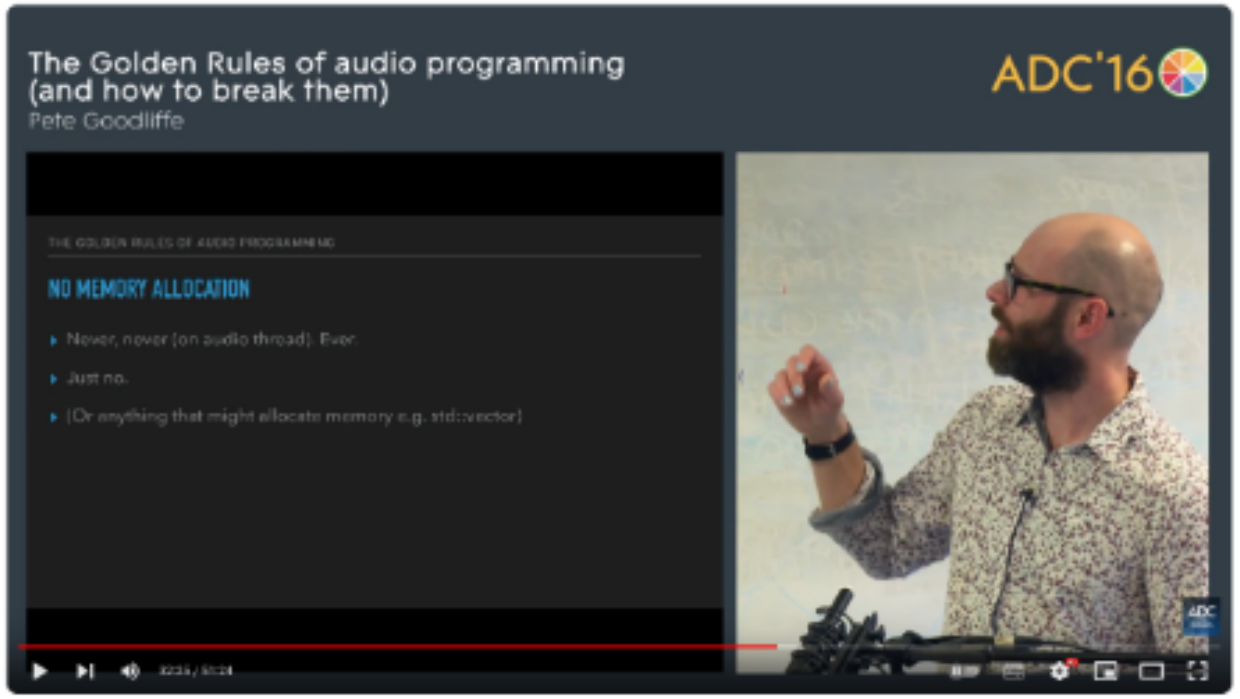
Deadline missed?

- No, audio thread is 'detached' from GC
- What if we want to load a sample in audioCallback?
- `std.stdio` uses GC 😯
- But Reference Counting wouldn't have helped

https://github.com/dkorpel/dconf

Audio guidelines

- No locks
- No malloc
- No file I/O



The Golden Rules of Audio Programming - Pete Goodliffe - ADC16

https://youtu.be/SJXGSJ6Zoro?si=F-QLhLi-AME7HDeo

https://github.com/dkorpel/dconf

@nogc should have a reason

Are we @nogc yet? #56

Open **atilaneves** opened this issue on Dec 17, 2019 · 4 comments

atilaneves commented on Dec 17, 2019 • edited by RazvanN7 Member

Description

It's common to see potential D users comment that large parts of the standard library aren't usable with the GC. It's difficult to counter that assertion without any data - I don't think anyone knows how much of Phobos can be used from `@nogc` code, nor could we point people to a resource for more information.

Similarly to Python's former wall of shame that got renamed to the wall of superpowers, it would be great if D had a webpage titled "Are we **@nogc** yet", preferably with a code-coverage-style visualisation of Phobos. At the very least, a list of functions/structs/classes that are `@nogc` compatible.

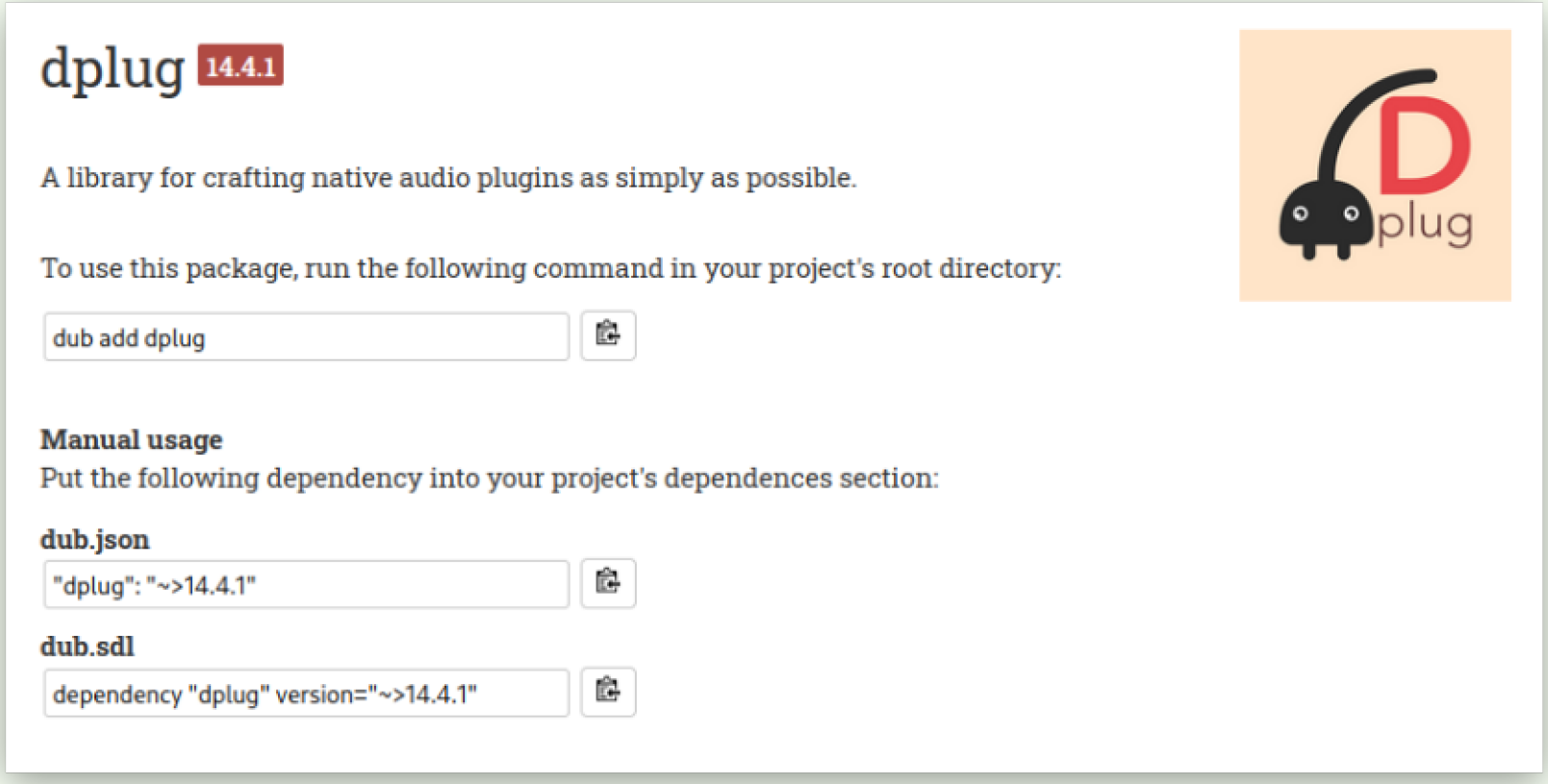
What are rough milestones of this project?

- Annotation of Phobos unittests with `@nogc` if possible
- A tool to extract the information of what Phobos code is called/covered by such unittests
- A web page to visualise the data

https://github.com/dlang/project-ideas/issues/56

https://github.com/dkorpel/dconf

@nogc should have a reason



✅

https://code.dlang.org/packages/dplug

On simplicity

1960s: Linear Congruential Generator

$$X_{n+1} = (aX_n + c) \bmod m$$

```
int seed = 1;
int RANDU()
{
    seed = seed * 65539 + 0;
    return seed;
}
```

"Truly horrible" - Donald Knuth

https://en.wikipedia.org/wiki/RANDU


MERSENNE TWISTER

Visualization by Cmglee, CC-BY-SA-3.0: https://commons.wikimedia.org/wiki/File:Mersenne_Twister_visualisation.svg

https://github.com/dkorpel/dconf

1997: MERSENNE TWISTER

- Rectifies flaws of older PRNGs
- Used by Excel, Matlab, GNU octave
- And Phobos (`std.random: MersenneTwisterEngine`)
- Fails TestU01 Big Crush test (2007)

https://en.wikipedia.org/wiki/Mersenne_Twister

2014: PCG Random

- Passes TestU01 suite
- More complex than the twister?
- Nope, just LCG with good constants and a tweak

https://www.pcg-random.org/

Just permute the LCG

```
10111010000000101001001000101000100100110100001000101111110110011
10111                  |
  | [01000000010100100100010100010010]
  |                    |
  +----------(rotate_bits)
                       |
    10100100100010100010010][010000000 ---> output
```

Full implementation:

```
uint randomPcg32(ref ulong seed)
{
    const ulong x = seed;
    seed = x * 0x5851F42D4C957F2D + 0x14057B7EF767814F;
    uint xorshifted = cast(uint)(((x >> 18UL) ^ x) >> 27UL);
    uint rot = cast(uint)(x >> 59UL);
    return (xorshifted >> rot) | (xorshifted << ((-rot) & 0b11111));
}
```

> Anybody can come up with a a complex solution. A simple one takes genius. You know it's genius when others say: "phui, anyone could have done that!" Except that nobody did.
>
> -Walter Bright

https://forum.dlang.org/post/t2i7mg$22am$1@digitalmars.com

Reference Counting is complex

- Truly horrible - ~~Donald Knuth~~
- Spawns lots of language features
  - `__mutable` / `__metadata` storage class (DIP1xxx)
  - Argument Ownership and Function Calls (DIP1021)
- Complicates types
  - `string` vs `RefCountedString`
  - `str` vs `String`

0:00 / 0:10

GC is... difficult

- Simple for user
- Complex to implement *in systems language*
  - Requires program-wide knowledge
  - False pointers
  - Non-portable (No WASM implementation yet)

6 suboptimal @nogc solutions

0. Manually free

```
void main()
{
    string s = environmentGet("PATH");
    writeln(s);
    free(s.ptr);
}
```

0. Manually free

```
void main()
{
    string s = environmentGet("PATH");
    scope(exit)
        free(s.ptr);
    writeln(s);
}
```

0. Manually free

- `malloc` ⟺ `free`
- COM programming with `ITypeInfo` and `IMoniker`:
- `GetFuncDesc` ⟺ `ReleaseFuncDesc`
- `GetVarDesc` ⟺ `ReleaseVarDesc`
- `GetNames` ⟺ `SysFreeString`
- `GetDisplayName` ⟺ `CoTaskMemFree`

https://github.com/dkorpel/dconf

0. Manually free

Documentation suggests `IMalloc::Free`

```
void getString(IMoniker moniker, IBindCtx ctx)
{
    BSTR displayName;
    moniker.GetDisplayName(ctx, null, &displayName);

    writeln(displayName.fromStringz);

    IMalloc allocator;
    CoGetMalloc(1, &allocator);
    allocator.Free(displayName);
    allocator.release();
}
```

https://learn.microsoft.com/en-us/windows/win32/api/objidl/nf-objidl-imoniker-getdisplayname

0. Manually free

- Simple (if you don't go nuts)
- Risky (memory leaks, double free)
- `@live` functions offer some protection
  - But doesn't distinguish GC/malloc pointers

https://github.com/dkorpel/dconf

The borrow checker makes it safe, right?

```
void main() @live
{
    int* x = cast(int*) malloc(4);
    free(x);
}
```

https://github.com/dkorpel/dconf

Right?

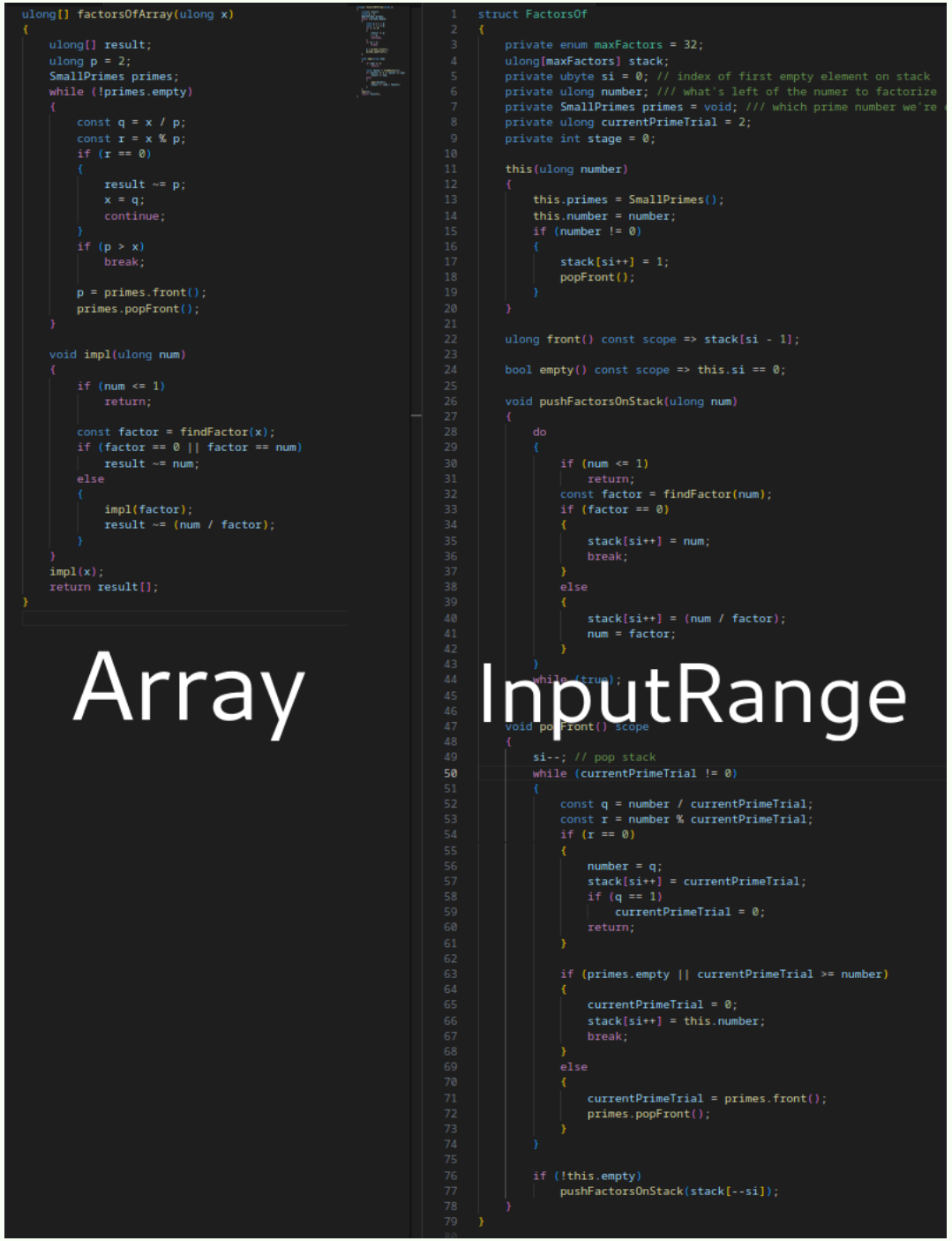
...

```
void main() @live
{
    int* x = new int;
    free(x); // No error, by design
}
```

1. Don't allocate

```
void main()
{
    string paths = "C:/dmd;C:/ldc2";
    foreach (string path; paths.splitter(';'))
    {
        writeln(path);
    }
}
```

- Return lazy ranges instead of arrays
- Annoying to write for recursive algorithms


Array
InputRange

1. Don't allocate

Voldemort Types can be annoying

```
import std.stdio, std.path;

void main()
{
    File f = File(withExtension("basilisk", ".txt"));
    // Error: none of the overloads of `this` are
    // callable using argument types `(Result)`

    import std.array;
    File g = File(withExtension("basilisk", ".txt").array);
}
```

https://wiki.dlang.org/Voldemort_types

2. Stack memory

- Automatically cleaned up
- Can't return it

```
char[] environmentGet(string var)
{
    char[32768] buf = void;
    // GetEnvironmentVariable(var, buf[]);
    return buf[]; // Error
}
```

2. Stack memory

- Annoying to call
- Small, fixed sizes only

```
void main()
{
    char[32768] buf;
    const str = environmentGet("PATH", buf[]);
}
```

https://github.com/dkorpel/dconf

3. OutputRanges / Appenders

```
void environmentGet(O)(string name, ref O sink)
{
    import std.range: put;
    put(sink, "...");
}

void main()
{
    import std.array : Appender;

    Appender!string appender;
    environmentGet("PATH", appender);
    string result = appender.data;
}
```

3. OutputRanges / Appenders

- Annoying to write
- Annoying to call (doesn't compose)
  - Can't do `environmentGet("PATH").splitter(';')`
- Still need a `@nogc` Appender
  - Hard to make `@safe`

4. Null garbage collection

> ~~Memory is automatically managed by occasionally pausing all threads and scanning for memory still in use, and freeing the rest.~~

https://github.com/dkorpel/dconf

4. Null garbage collection

- "Everybody thinks about garbage collection the wrong way" - Raymond Chen
- Simulating a computer with infinite memory
- Null garbage collector: never deallocate
- Works if enough RAM

https://devblogs.microsoft.com/oldnewthing/20100809-00/?p=13203

https://github.com/dkorpel/dconf

4. Null garbage collection

Amusing story from Kent Mitchell

https://devblogs.microsoft.com/oldnewthing/20180228-00/?p=98125

4. Null garbage collection

- DMD does this (unless `dmd -lowmem`)
- ctod does this for WebAssembly
- "Out Of Memory" risk

5. Scope Array

- Extension of stack memory
- Examples:
  - `std.internal.string: tempCString`
  - `dmd.common.string: SmallBuffer`

https://github.com/dlang/phobos/blob/master/std/internal/cstring.d

https://github.com/dkorpel/dconf

5. Scope Array

```
struct ScopeArray(T)
{
    T[32] stackMem;
    T[] big;

    this(size_t length)
    {
        if (length > stackMem.length)
            big = malloc(T.sizeof * length);
    }

    T[] opIndex() => big ? big[] : stackMem[];

    ~this() { if (big.ptr) free(big.ptr); }
}
```

5. Scope Array

Length must be given upfront

```
void main()
{
    auto a = ScopeArray!char(length: 1024);

    char[] path = environmentGet("PATH", a[]);

    writeln(path);

    // a.~this();
}
```

5. Scope Array

Unless... 🤔

```
void main()
{
    auto a = Arena();

    char[] path = environmentGet("PATH", &a);

    writeln(path);

    // a.~this();
}
```

The 80 line solution

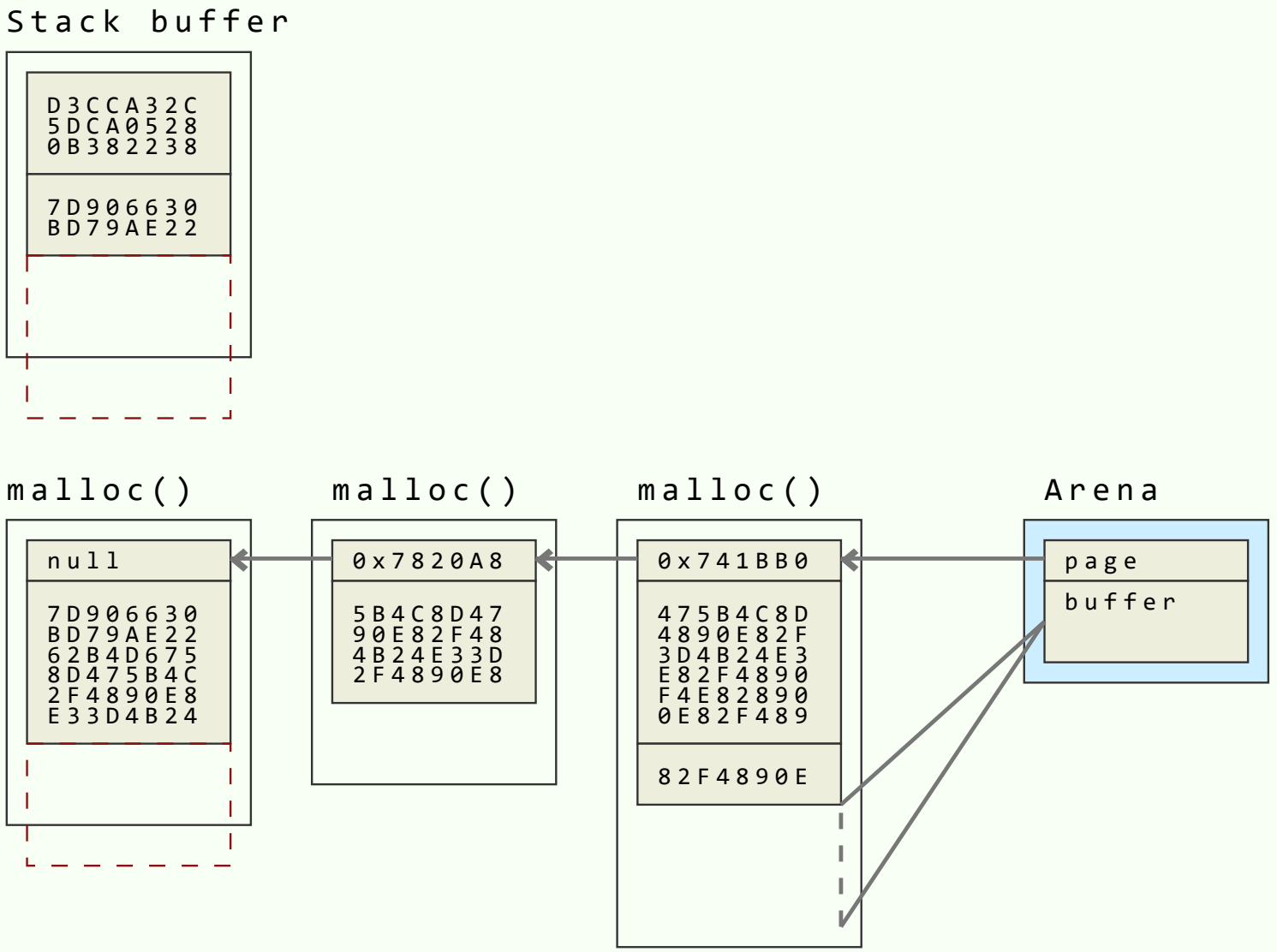
https://github.com/dkorpel/dconf

Arenas

```
struct Arena
{
    ubyte[] buffer;
    ArenaPage* page = null;
}

struct ArenaPage
{
    ArenaPage* previous;
    // variable number of bytes follow
}
```

https://github.com/dkorpel/dconf



Works with small buffer optimization

```
void heap()
{
    Arena a;
    ubyte[] res = a.allocate(100); // heap allocates
}

void stack()
{
    ubyte[512] buf = void;
    Arena a = Arena(buf[]);
    ubyte[] res = a.allocate(100); // uses stack buffer
}
```

Allocator interface

- Arena could be passed around by `ref` or pointer
- But we want something extensible

```
abstract class Allocator
{
    ubyte[] allocate(size_t size, size_t alignment);
}

class Arena : Allocator;
class GcAllocator : Allocator;
class FailAllocator : Allocator;
```

```
struct Allocator
{
    AllocatorBase* x;
}

struct AllocatorBase
{
    immutable AllocFunc allocate;
}

alias AllocFunc = ubyte[] function(size_t size, size_t alignment, void* this_);

struct Arena
{
    AllocatorBase base; // Old school struct inheritance
    ubyte[] buffer;
    ArenaPage* page;
}
```

Why are you re-inventing classes?

- No druntime dependency
- Reduce redundant pointers
- C compatibility
- Implementing allocators is low-level anyway

https://github.com/dkorpel/dconf

```
struct Arena
{
    Allocator alloc() return => Allocator(&this);
}

struct Allocator
{
    AllocatorBase* base;

    T[] array(T)(size_t length) =>
        cast(T[]) base.allocate(T.sizeof * length);
}

void main()
{
    Arena a;
    char[] str = a.alloc.array!char(128);
}
```

Allocator should have GC default

```
string environmentGet(string name, Allocator alloc = gc)
{
    return alloc.array!char(n);
}
```

"Hannah Montana functions"

```
void main()
{
    string s = environmentGet("PATH");

    Arena a;
    string s = environmentGet("PATH", a.alloc);
}
```

 Best of both worlds!

It can be @safe

```
// Requires `-preview=dip1000`
string environmentGet(string name, return scope Allocator alloc = gc);

string global;

void main() @safe
{
    global = environmentGet("PATH", gc); // Fine

    Arena a;
    global = environmentGet("PATH", a.alloc); // Error
}
```

But what about `@nogc`

- There's `return scope`, but no `@inout_nogc`
- DIPs for callback attributes still pending
- Cheat: pretend it is `@nogc`
- Hot take: `@nogc` should not be part of function type
- Linting tool instead

https://github.com/dlang/DIPs/pull/198

Allocator can be stored

```
struct Array(T)
{
    T[] slice;
    size_t capacity;
    Allocator alloc;
}
```

No more invalidation problem

```
void main() @safe
{
    import automem.vector;
    auto vec1 = vector(1, 2, 3);
    int[] slice1 = vec1[];
    vec1.reserve(4096);
    int[] slice2 = vec1[];
}
```

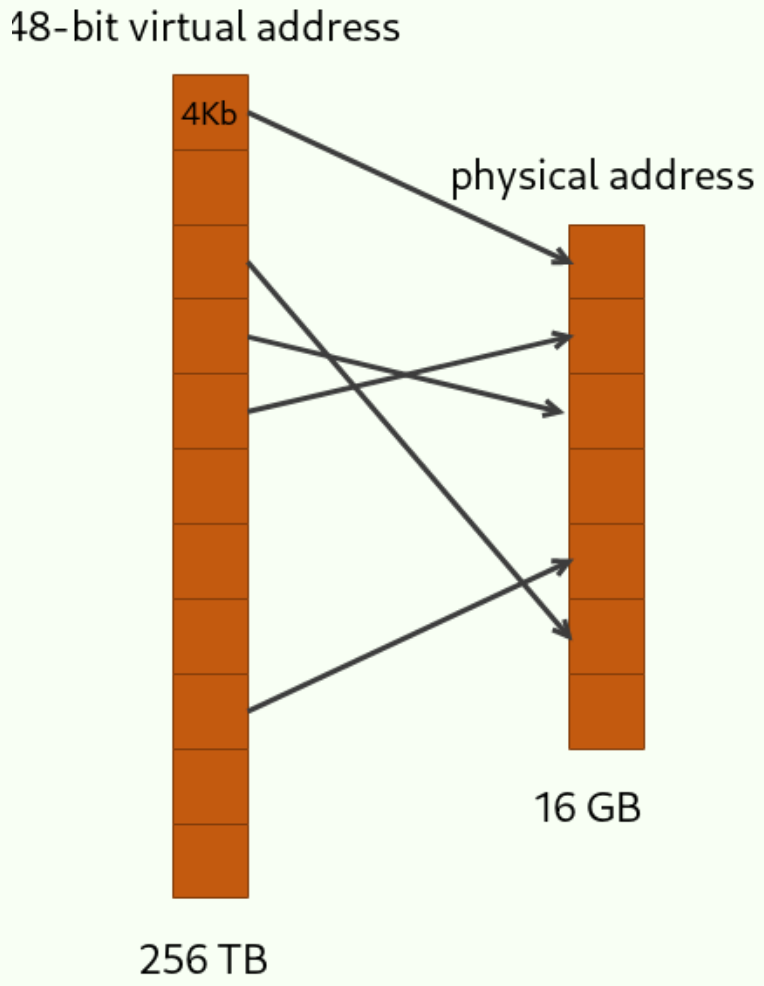
Just don't free when growing the array 🙈

https://github.com/atilaneves/automem/issues/25

https://github.com/dkorpel/dconf

Overhead can be reduced

- Memory mapping instead of linked list
- Non-portable



Resources

- Ryan Fleury:
  - Untangling Lifetimes: The Arena Allocator
  - Enter The Arena: Simplifying Memory Management
- Chris Wellons:
  - Arena allocator tips and tricks
  - Arenas and the almighty concatenation operator

It cleaned up my code

- Deleted tons of destructors and `free()` calls
- Less `@trusted` annotations
- Deleted `ScopeArray`, `Stack`, and `NogcAppender`
  - `Array` is all you need 🥰
- Found more uses for the pattern

https://github.com/dkorpel/dconf

GPU Memory mapping

```
void copying()
{
    float[] data;
    data ~= 3.0;
    data ~= 4.0;
    glBufferSubData(buffer, data);
}

void memoryMapping()
{
    float[] data = glMapBufferRange(buffer, 2 * float.sizeof);
    size_t i = 0;
    data[i++] = 3.0;
    data[i++] = 4.0;
    glUnmapBuffer(buffer);
}
```

Map ≔ alloc, unmap ≔ free

```
{
    auto mapper = Mapper(buffer, 2); // Arena
    scope float[] mappedBuffer = mapper[];

    auto data = Array!float(storage: mappedBuffer, failAllocator);
    data ~= 3.0;
    data ~= 4.0;

    // mapper.~this() unmaps
}
```

Ugly signatures

Clutter from long parameter declaration

```
string environmentGet(string name);

// vs

string environmentGet(string name, return scope Allocator alloc = gc);
```

Context struct could help

Language feature of Jai and Odin

```
main :: proc()
{
    context.user_index = 456
    {
        context.allocator = my_custom_allocator()
        context.user_index = 123
        supertramp() // `context` is implicitly passed
    }
    assert(context.user_index == 456)
}
```

https://odin-lang.org/docs/overview/#implicit-context-system

https://github.com/dkorpel/dconf

```
struct Context
{
    Allocator allocator;
    Allocator temp_allocator;
    Assertion_Failure_Proc assertion_failure_proc;
    Logger logger;
    Random_Generator random_generator;

    void* user_ptr;
    ptrdiff_t user_index;

    // Internal use only
    void* _internal;
}
```

https://github.com/odin-lang/Odin/blob/a25a9e6ebe58510cfac20e1187f41a01ec3ec2b2/base/runtime/core.odin#L434-L446

Wrapping up

Suggested GC strategy

- Write code as if you have infinite memory
  - (Optimization for a known future is okay)
- *If* you need to avoid the GC
  - Replace `new T[]` with `allocator.array!T`
  - Place `Arena` / `Allocator` where needed
  - Use `-preview=dip1000` for `@safe`
  - Otherwise it's `@system` / `@trusted`

Takeaways

- Look for simple solutions
- Calling `free()` is not `@safe`
- End-of-scope cleanup can be `@safe` with `scope`
- Give it a try!

https://github.com/dkorpel/dconf

Avoid the Garbage Collector in 80 slides

Dennis Korpel