Adapting the Pony language onto the seL4 microkernel

In search of a capability-focused microkernel programming interface

Stewart Webb
Email: sjwebb@student.unimelb.edu.au

A research project completed as part of a
Master of Science (Computer Science)

Supervised by Toby Murray

School of Computing and Information Systems
The University of Melbourne

November 2023

Full thesis available online at `https://swebb.id.au/research`

About me / Project context

- Computing and Software Systems undergrad at Unimelb (2012 $\rightarrow$ 2014)
- Several years in industry + a startup
- Computer Science Masters part time (mid-2018 $\rightarrow$ end of 2022)
    - Exchange semester at ETH Zurich (2019),
      'Advanced Operating Systems' subject, under Timothy Roscoe's Systems Group + using their Barrelfish microkernel OS
    - seL4 capability project from UNSW Trustworthy Systems via AOS staff link for Masters thesis
- ...???

Background to cover

seL4
=
Capability-based Operating System

Pony
=
Object-Capability Language

seL4 - a high-assurance, formally-verified microkernel[1][2]

Security. Performance. Proof.

Includes proofs of *functional correctness*, *integrity* + *confidentiality* in access control, and *information-flow noninterference*

Capabilities centric to programming model

[1] Gerwin Klein, Kevin Elphinstone, et al., “seL4: formal verification of an OS kernel” 2009.

[2] Gerwin Klein, June Andronick, et al., “Comprehensive Formal Verification of an OS Microkernel” 2014.

What are Capabilities?

- **Unforgeable** references that represent both access rights and access level/type
- seL4 capabilties rough equivalent: Linux file descriptors

```
#include <fcntl.h>
int fd;
char buf[50];

fd = open("test.txt", O_RDONLY);
int bytes_read = read(fd , buf, 10);
```

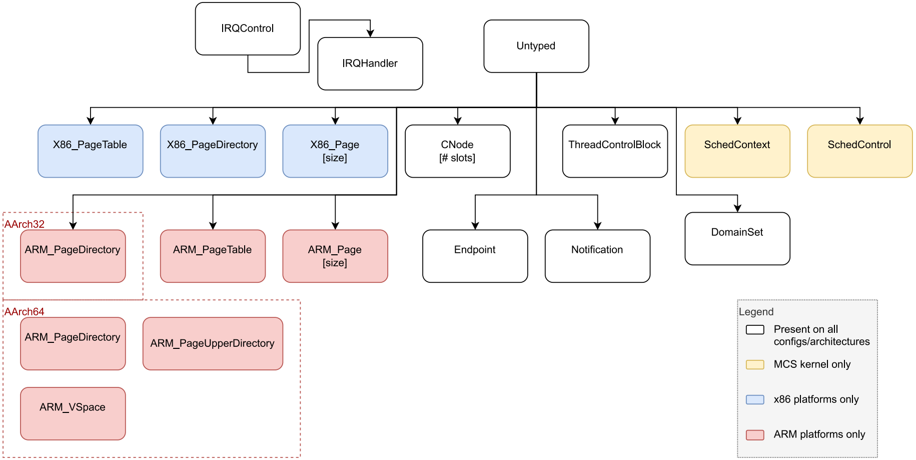
Core related problem: **Ambient Authority**
+ Principle of Least Authority

Best motivated by the 'Confused Deputy' example[3]

[3] Norm Hardy, "The Confused Deputy: (Or Why Capabilities Might Have Been Invented)" 1988.

seL4 Capabilities

∼12 capability types



(rough diagram, not comprehensive)

seL4 Mechanisms

Eight (only 8!) system calls:

seL4 syscall API

1. `seL4_Yield()`
2. `seL4_Send(capability, message)`
3. `seL4_Recv(endpoint_or_notification_cap, out sender_info)`
4. `seL4_NBSend(capability, message)`
5. `seL4_NBRecv(endpoint_or_notification_cap, out sender_info))`
6. `seL4_Call(capability, message)`
7. `seL4_Reply(message)`
8. `seL4_ReplyRecv(recv_endpoint_or_notification_cap, reply_message, out recv_sender_info)`

seL4 Capabilities ✓

Object Capability Languages ?

Object-Capability programming languages

A family of research programming languages

- E (2006)[4] , Joe-E (2010)[5]
- Secure EcmaScript (2013)[6] / Jessie (2018)[7]
- Pony (2015) [8] [9]
- Dala (2021) [10]

[4] Mark S. Miller, “Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control” 2006

[5] Adrian Mettler, David Wagner, and Tyler Close, “Joe-E: A Security-Oriented Subset of Java.” 2010

[6] Mark S. Miller, Tom Van Cutsem, and Bill Tulloh, “Distributed Electronic Rights in JavaScript” 2013

[7] `https://github.com/endojs/Jessie`

[8] Sylvan Clebsch, “'Pony': co-designing a type system and a runtime.” 2017

[9] `https://www.ponylang.io/`

[10] Kiko Fernandez-Reyes et al., “Dala: a simple capability-based dynamic language design for data race-freedom” 2021

Object Capabilities

What are *object* capabilities?

- Capability principle applied to objects in object-oriented programming
- Assumes memory-safety - references can't be invented (e.g. pointer dereference)
- Allows clear understanding of what authority code has access to - only what references it got passed!
  e.g. importing a 3rd-party library - "what will this code be able to do?"

Motivation - dynamic systems

From the original abstract for this project:

> *Whilst many tools have been developed for building systems with seL4 such as capDL and CAmKES, they* **rely on static distribution of capabilities upfront at project build time**, *and in general do not provide for dynamic capability transfer once the system has started.*
> *In practice, for runtime/dynamic access control, many systems revert to standard POSIX layers where ambient authority for resource access returns, throwing out the fine-grained access control possibilities that come for free with a capability model.*
> *In theory, an object capability language would provide a better means for building dynamic capability systems, especially if such a language can have seL4 capabilities for kernel objects mapped into it.*

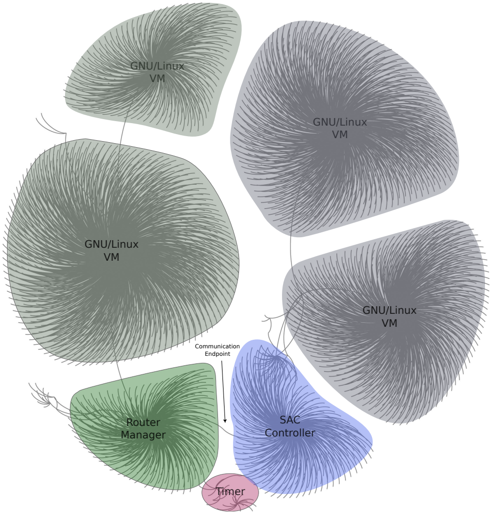
capDL example

```
arch ia32

objects {
  my_tcb = tcb
  my_cnode = cnode (3 bits)
  my_frame = frame (4k, paddr: 0x12345000) // paddr
 ↪  is optional
  my_page_table = pt
  my_page_directory = pd
}

caps {
  // Specify cap addresses (ie. CPtrs) in cnodes.
  my_cnode {
    1: my_tcb
    2: my_frame
    3: my_page_table
    4: my_page_directory
  }
  //...
```

```
  //...
  // Specify address space layout.
  // With 4gb page directories, 4mb page tables, and
 ↪  4kb frames,
  // the frame at paddr 0x12345000 will be mapped at
 ↪  vaddr 0xABCDE000.
  my_pd {
    0x2AF: my_pt
  }
  my_pt {
    0xDE: my_frame
  }

  // Specify root cnode and root paging structure of
 ↪  thread.
  my_tcb {
    vspace: my_pd
    cspace: my_cnode
  }
}
```

Sourced from the capDL documentation at
`https://docs.sel4.systems/projects/capdl/`

capDL system example - “Secure Access Controller”



Taken from Gerwin Klein, June Andronick, et al., “Comprehensive Formal Verification of an OS Microkernel” 2014, pg 2:46

Research goal

seL4 Capabilities

Object Capability Languages

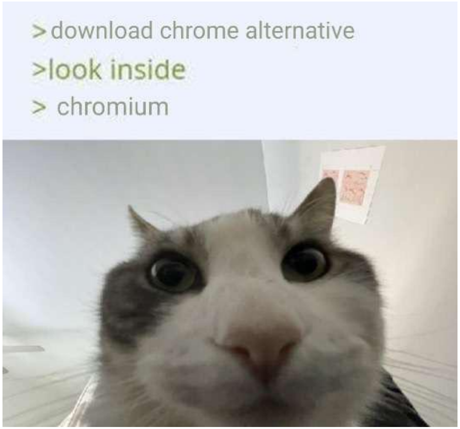
Can the two be made to work together?

Can an ocap language provide a more *useful* way to program on seL4?

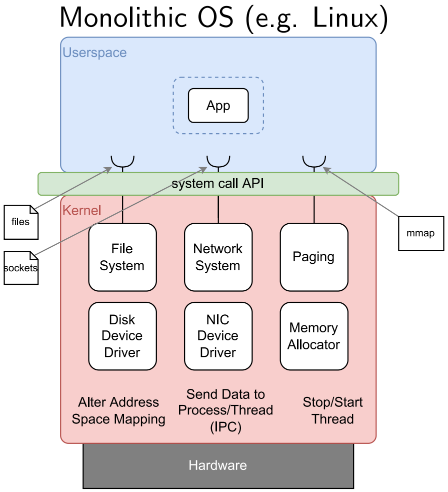
Choosing an OCap Language

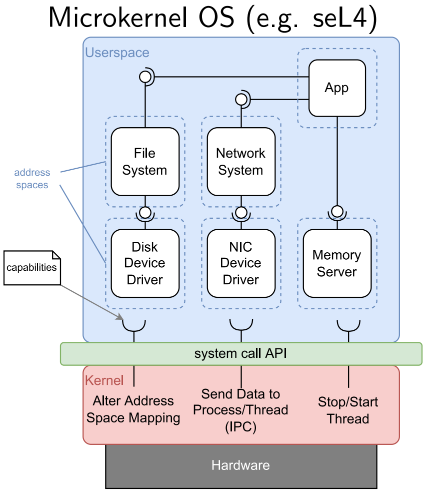
Which language to choose?

Key problem: **Dependencies**



Dependencies matter a lot on microkernels!





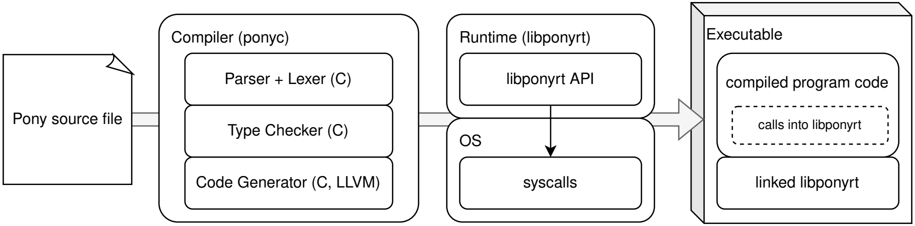
Choosing an OCap Language - Language Survey

E	✗ Java, JVM
Secure EcmaScript / Jessie	✗ JavaScript, JS engine
SHILL (+ PLASH)	✗ Racket extensions + Scheme
Dala	✗ Moth VM / Grace, 'SOMns' / Newspeak, TruffleSOM, Java GraalVM
Other options	
• Rust - `cap-std` + `ferros` crates	Unclear ocap environment
• WebAssembly (surprisingly!)	Still in development
• Microsoft Sing# [11]	Windows build system, partially-closed source
Pony	✓ LLVM, C library

[11] C# extensions from 'Singularity' project

Choosing an OCap Language - Why Pony?

- Minimal language dependencies (C and OS syscalls), no bytecode or interpreter
- Performance focus, compiled via LLVM



"Public" libponyrt API

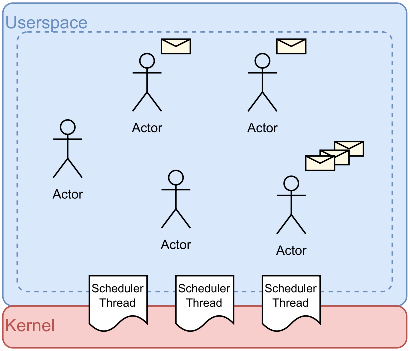
(C) Functions called by LLVM-compiled code:

```
// runtime control
pony_init()
pony_start()
pony_stop()
// get global context
pony_ctx()
// threads
pony_register_thread()
pony_unregister_thread()
// allocate actor
pony_create(ctx, type)
// switch current actor
pony_become(ctx, actor)
// message-passing
pony_alloc_msg(size_index, id)
pony_send(ctx, to_actor, msg)
// allocate on current actor's heap
pony_alloc(ctx, size)
```

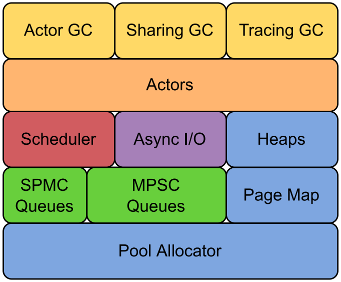
```
// scheduling
pony_schedule(ctx, actor)
pony_unschedule(ctx, actor)

// tracing for garbage collection
pony_trace(ctx, addr)
pony_traceknown(ctx, addr, type, mutabilit
pony_traceunknown(ctx, addr, mutability)
// message-based garbage collection
pony_gc_send(ctx)
pony_send_done(ctx)
pony_gc_recv(ctx)
pony_recv_done(ctx)
pony_gc_acquire(ctx)
pony_acquire_done(ctx)
pony_gc_release(ctx)
pony_release_done(ctx)
```

Pony Overview + Components

Actor-model programming language with message-passing queues



[12]



[12]Diagram adapted from Sylvan Clebsch, "'Pony': co-designing a type system and a runtime." 2017

Pony - Multicore Execution + Garbage Collection

Example from [13], showing how Pony GC occurs between actor behaviours - no 'stop the world' step.

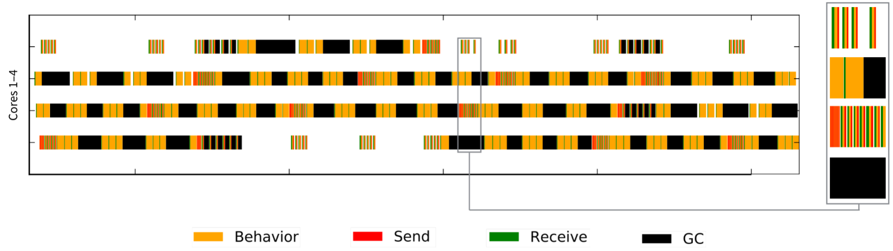


Fig. 1. CPU usage during part of a small Pony program. To the right is a blow-up of a small window. Y-axis: the different core IDs. X-axis the application's timeline (from 5002×10^5 to 5010×10^5 CPU cycles). The diagram demonstrates that while a core may be garbage collecting, other cores may also be garbage collecting, or executing behaviours, or tracing upon send/receipt. (Behaviour=mutator)

[13] Sylvan Clebsch et al., "Orca: GC and Type System Co-Design for Actor Languages" 2017

Capability Embedding Models (broader thesis)

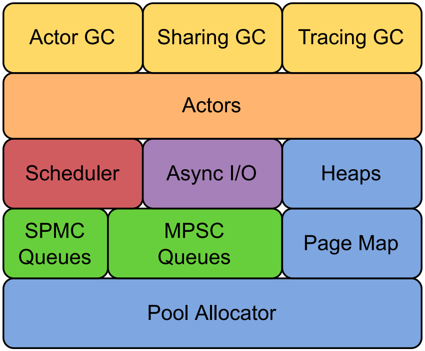
How might you go about linking Pony object capabilities and seL4 capabilities?

- Remote (Pony) actors with communication via 'message pumps' over seL4 endpoints?
- Handing off seL4 endpoints to represent capabilities to talk to objects/actors?
- Handing off physical memory between protection domains to represent transferring objects?
- Embedding seL4 capability types in Pony's type system and backing them with caprefs at runtime?

Various tradeoffs involved

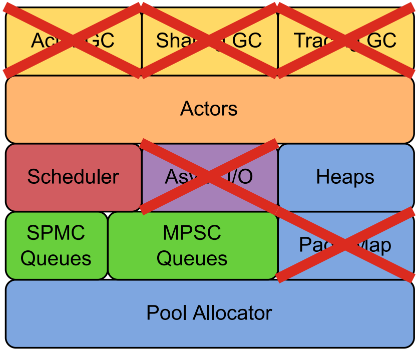
Regardless... seL4 types likely required in Pony types, and thus, runtime required

Pony on seL4



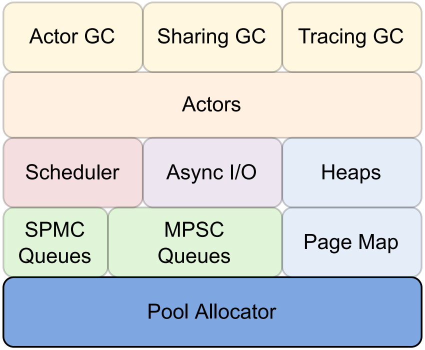
Let's port to seL4! "How hard could it be?"

Pony Components to Port

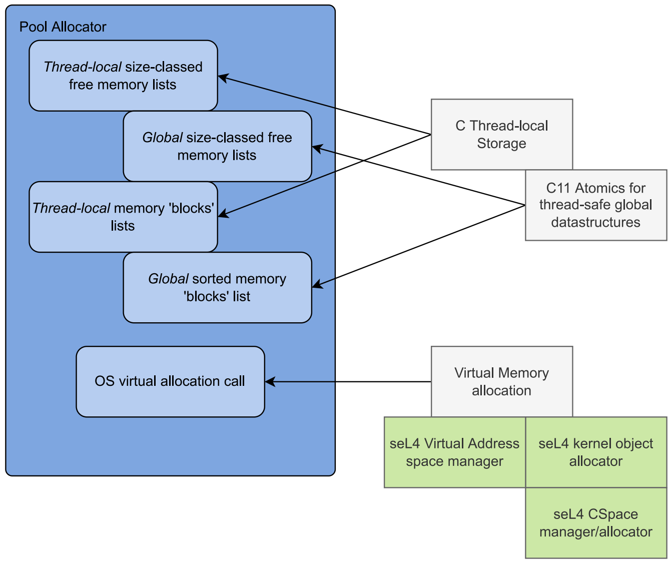


- GC strongly relies on assumptions made from the usual Pony message-passing model (which might not have been enforcable across seL4 protection domains)
- Async I/O involves a special single 'pinned' actor + monolithic-OS-specific async I/O mechanisms (e.g. epoll/kqueue)

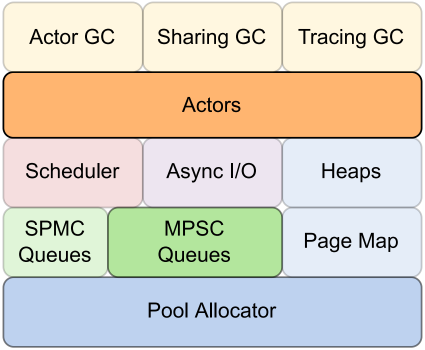
Pony Components - Pool Allocator



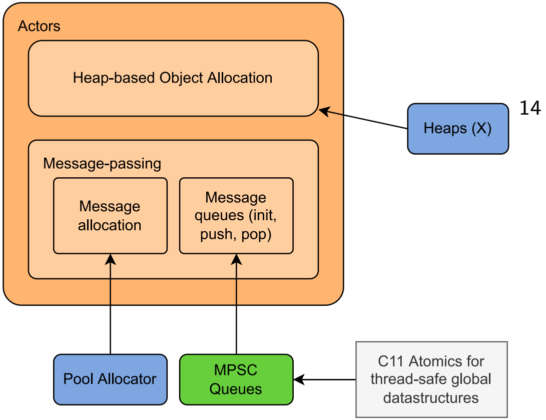
Porting Pony's Pool Allocator


Pool Allocator
Thread-local size-classed free memory lists
Global size-classed free memory lists
Thread-local memory 'blocks' lists
Global sorted memory 'blocks' list
OS virtual allocation call
C Thread-local Storage
C11 Atomics for thread-safe global datastructures
Virtual Memory allocation
seL4 Virtual Address space manager
seL4 kernel object allocator
seL4 CSpace manager/allocator

Pony Components - Actors

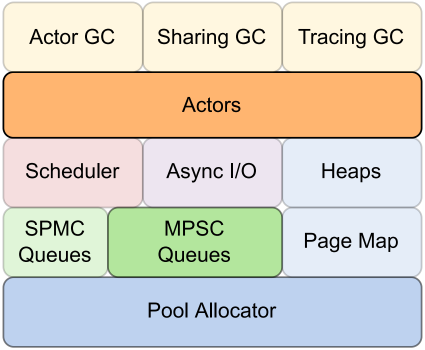


Porting Actor Communication


Actors
Heap-based Object Allocation
Message-passing
Message allocation
Message queues (init, push, pop)
Heaps (X)
14
Pool Allocator
MPSC Queues
C11 Atomics for thread-safe global datastructures

[14](X) indicating that Heaps have yet to be ported

Pony Components ported so far



'Key Observations' during research

1. Pony in its current form[15]
   assumes a single address-space of execution, and this is a strong root of its object capability model.

   Memory addresses can be used as unique identifiers, and are unforgeable due to its memory allocation model (and assumption that the allocator is correct and never hands out overlapping memory).

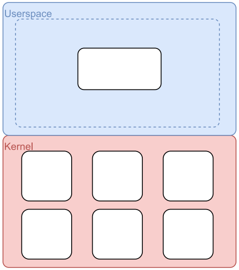
2. seL4 IPC does not support reliable non-blocking sending.

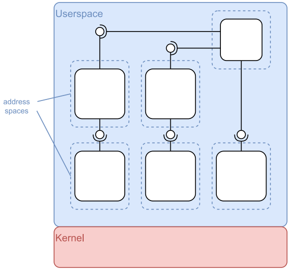
   an `NBSend` call to an Endpoint silently drops messages if a thread is not ready and blocked waiting on the other side of the Endpoint.

[15]Work has been done on a distributed version of Pony where distributed object identity is addressed - see Sebastian Blessing, *A String of Ponies: Transparent Distributed Programming with Actors* 2013

Discoveries and Conclusion

- Pony object capabilities reliant on single address-space model in current implementation. seL4 confined component communication somewhat breaks this.
- seL4 confined component communication also breaks Pony's unbounded queue assumptions (have to be able to handle failure)
    - Embedded memory constraints also don't help





Discoveries and Conclusion

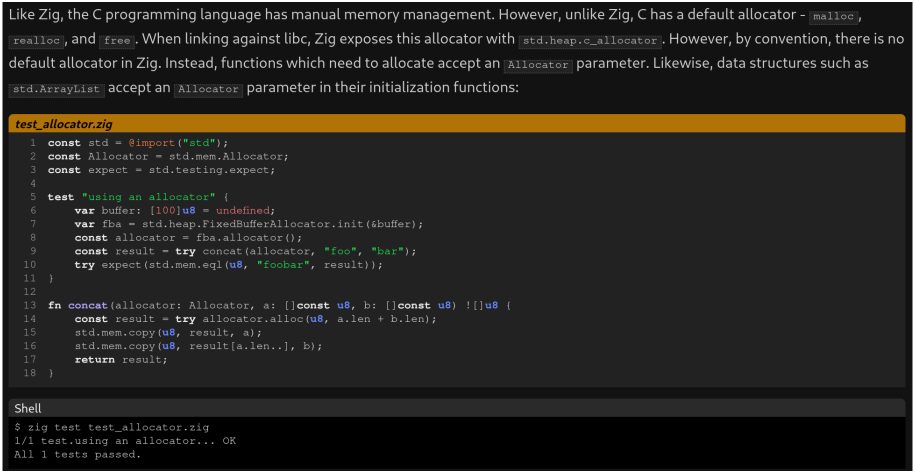
- Pony in its current form lacks any execution 'compartments' a la Secure EcmaScript, which might better match seL4's confined components model
  - Also no dynamic code evaluation/`eval()`
- seL4 capability transfer requires synchronous IPC - Pony communication model is inherently asynchronous
- "Ambient authority" / global rights often assumed for memory allocation in OCap languages
- Modifications to Pony / new language? / further research/work required!

Ideas for future work focus

- For a new or candidate language - implications of object model (GC? heap/stack?)
- Re-evaluate Singularity "shared heap" idea? (May be relatable to Dala's safe/unsafe heaps?)
- Language with memory allocation authority - heap, stack (e.g. `import mymodule with stackalloc(512)`)
    - Zig is actually somewhat of an example of this (see next slide)
- Examine distributed ocap models further. E has many features for this (NearRef, FarRef, Promise Pipelining) - may be design coupled to its single-threaded 'vat' model though.

Zig explicit allocator example

From `https://ziglang.org/documentation/0.11.0/#Memory`



Like Zig, the C programming language has manual memory management. However, unlike Zig, C has a default allocator - `malloc`, `realloc`, and `free`. When linking against libc, Zig exposes this allocator with `std.heap.c_allocator`. However, by convention, there is no default allocator in Zig. Instead, functions which need to allocate accept an `Allocator` parameter. Likewise, data structures such as `std.ArrayList` accept an `Allocator` parameter in their initialization functions:

test_allocator.zig

```zig
const std = @import("std");
const Allocator = std.mem.Allocator;
const expect = std.testing.expect;

test "using an allocator" {
    var buffer: [100]u8 = undefined;
    var fba = std.heap.FixedBufferAllocator.init(&buffer);
    const allocator = fba.allocator();
    const result = try concat(allocator, "foo", "bar");
    try expect(std.mem.eql(u8, "foobar", result));
}

fn concat(allocator: Allocator, a: []const u8, b: []const u8) ![]u8 {
    const result = try allocator.alloc(u8, a.len + b.len);
    std.mem.copy(u8, result, a);
    std.mem.copy(u8, result[a.len..], b);
    return result;
}
```

Shell

```
$ zig test test_allocator.zig
1/1 test.using an allocator... OK
All 1 tests passed.
```

Bibliography I

Agoric Inc. *Jessie, simple universal safe mobile code*. GitHub repository. 2018. URL: https://github.com/endojs/Jessie.

Sebastian Blessing. *A String of Ponies: Transparent Distributed Programming with Actors*. Available online at https://www.ponylang.io/media/papers/a_string_of_ponies.pdf or https://www.doc.ic.ac.uk/~scb12/publications/s.blessing.pdf. Masters Thesis. 2013.

Sylvan Clebsch. "'Pony': co-designing a type system and a runtime.". PhD thesis. Imperial College London, 2017.

Sylvan Clebsch, Juliana Franco, Sophia Drossopoulou, et al. "Orca: GC and Type System Co-Design for Actor Languages". In: *Proc. ACM Program. Lang.* 1.OOPSLA (Oct. 2017). DOI: 10.1145/3133896. URL: https://doi.org/10.1145/3133896.

Kiko Fernandez-Reyes, Isaac Oscar Gariano, James Noble, et al. "Dala: a simple capability-based dynamic language design for data race-freedom". In: *Proceedings of the 2021 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. 2021, pp. 1–17.

Norm Hardy. "The Confused Deputy: (Or Why Capabilities Might Have Been Invented)". In: *SIGOPS Oper. Syst. Rev.* 22.4 (Oct. 1988), pp. 36–38. ISSN: 0163-5980. DOI: 10.1145/54289.871709. URL: https://doi.org/10.1145/54289.871709.

Bibliography II

Gerwin Klein, June Andronick, Kevin Elphinstone, et al. "Comprehensive Formal Verification of an OS Microkernel". In: *ACM Trans. Comput. Syst.* 32.1 (Feb. 2014). ISSN: 0734-2071. DOI: 10.1145/2560537. URL: https://doi.org/10.1145/2560537.

Gerwin Klein, Kevin Elphinstone, Gernot Heiser, et al. "seL4: formal verification of an OS kernel". In: *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*. SOSP '09. New York, NY, USA: Association for Computing Machinery, Oct. 2009, pp. 207–220. ISBN: 978-1-60558-752-3. DOI: 10.1145/1629575.1629596. URL: https://doi.org/10.1145/1629575.1629596.

Adrian Mettler, David Wagner, and Tyler Close. "Joe-E: A Security-Oriented Subset of Java.". In: *Proceedings of the Network and Distributed System Security Symposium, NDSS 2010, San Diego, California, USA, 28th February - 3rd March 2010*. The Internet Society, Jan. 2010. URL: https://www.ndss-symposium.org/ndss2010/joe-e-security-oriented-subset-java.

Mark S. Miller. "Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control". PhD Thesis. Johns Hopkins University, May 2006.

Mark S. Miller, Tom Van Cutsem, and Bill Tulloh. "Distributed Electronic Rights in JavaScript". In: *ESOP'13 22nd European Symposium on Programming*. 2013.

Pony Developers. *Website for the Pony programming language*. Website. 2022. URL: https://www.ponylang.io/.