RedLeaf: Isolation and Communication in a Safe Operating System

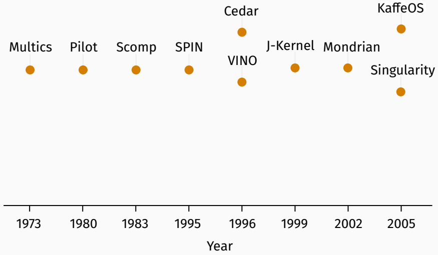
Vikram Narayanan[1], Tianjiao Huang[1], David Detweiler[1], Dan Appel[1], Zhaofeng Li[1], Gerd Zellweger[2], Anton Burtsev[1]

OSDI ’20

[1]University of California, Irvine

[2]VMware Research

History of Isolation



- Isolation of kernel subsystems
  - Final report of Multics (1976)
  - Scomp (1983)
- Systems remained monolithic
  - Isolation was expensive

Isolation mechanisms

- Hardware Isolation
  - Segmentation (46 cycles)[1]
  - Page table isolation (797 cycles)[2]
  - VMFUNC (396 cycles)[3]
  - Memory protection keys (20-26 cycles)[4]
- Language based isolation
  - Compare drivers written (DPDK-style) in a safe high-level language (C, Rust, Go, C#, etc.)[5]
  - Managed runtime and Garbage collection (20-50% overhead on a device-driver workload)

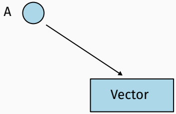
[1] L4 Microkernel: Jochen Liedtke
[2] https://sel4.systems/About/Performance/
[3] Lightweight Kernel Isolation with Virtualization and VM Functions, VEE 2020
[4] Hodor: Intra-process isolation for high-throughput data plane libraries
[5] The Case for Writing Network Drivers in High-Level Programming Languages, ANCS 2019

Traditional Safe languages vs Rust

Java, C# etc.

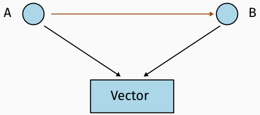
Traditional Safe languages vs Rust

Java, C# etc.



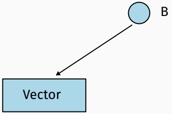
Traditional Safe languages vs Rust

Java, C# etc.



Traditional Safe languages vs Rust

Java, C# etc.



Traditional Safe languages vs Rust

Java, C# etc.

Vector

Traditional Safe languages vs Rust

Java, C# etc.

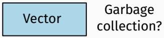
Vector

Garbage
collection?

Traditional Safe languages vs Rust

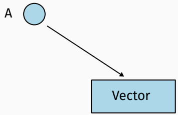
Rust

Java, C# etc.

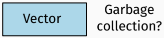


Traditional Safe languages vs Rust

Rust

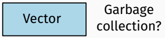


Java, C# etc.

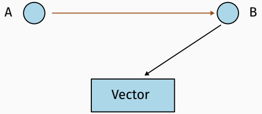


Traditional Safe languages vs Rust

Java, C# etc.

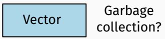


Rust

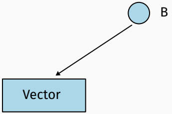


Traditional Safe languages vs Rust

Java, C# etc.

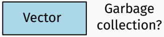


Rust

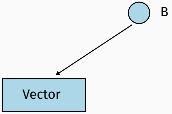


Traditional Safe languages vs Rust

Java, C# etc.



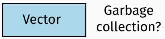
Rust



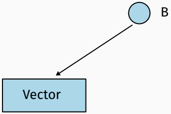
- Linear types

Traditional Safe languages vs Rust

Java, C# etc.



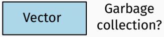
Rust



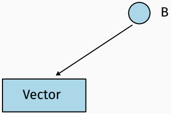
- Linear types
- Enforces type and memory safety

Traditional Safe languages vs Rust

Java, C# etc.



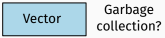
Rust



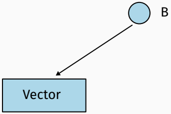
- Linear types
- Enforces type and memory safety
- Statically checked at compile time

Traditional Safe languages vs Rust

Java, C# etc.



Rust

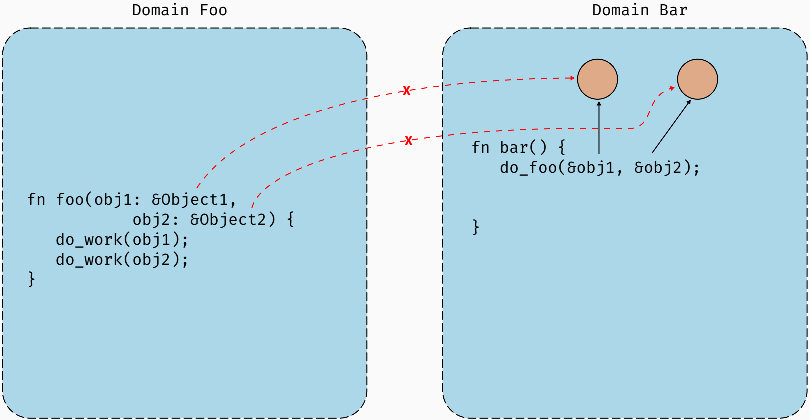


- Linear types
- Enforces type and memory safety
- Statically checked at compile time
- Safety without runtime garbage collection overhead

Language-based isolation - Rust

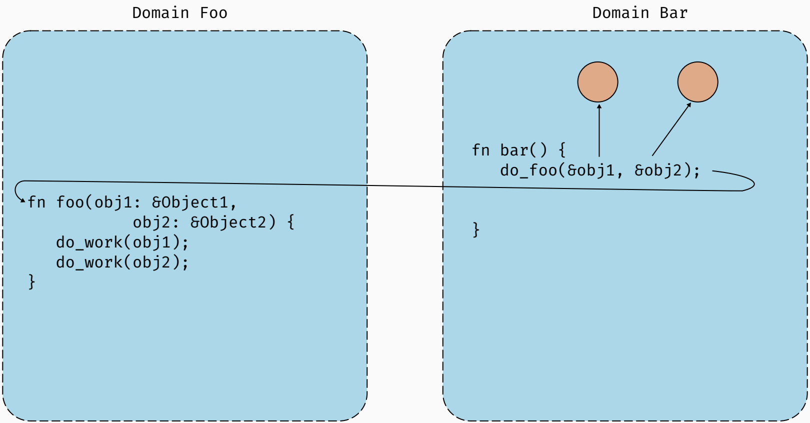
- Mostly use Rust as a drop-in replacement for C
- Numerous possbilities
    - Fault Isolation
    - Transparent device-driver recovery
    - Safe Kernel extensions
    - Fine-grained capability-based access control etc.

Fault isolation in Language-based systems



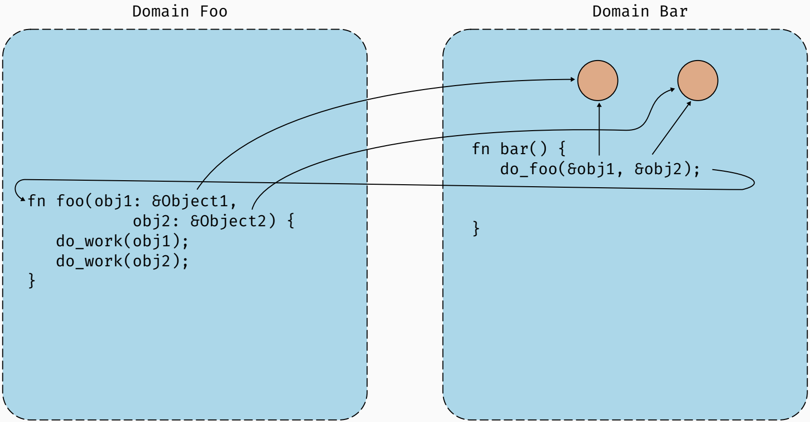
- e.g., SPIN (using Modula-3 pointers)

Fault isolation in Language-based systems



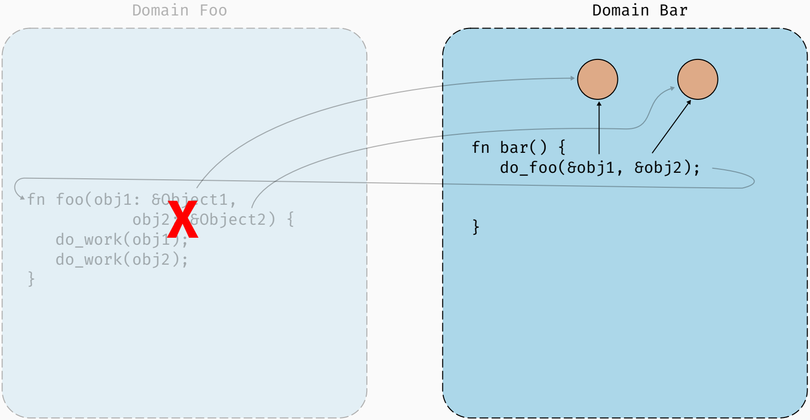
- e.g., SPIN (using Modula-3 pointers)

Fault isolation in Language-based systems



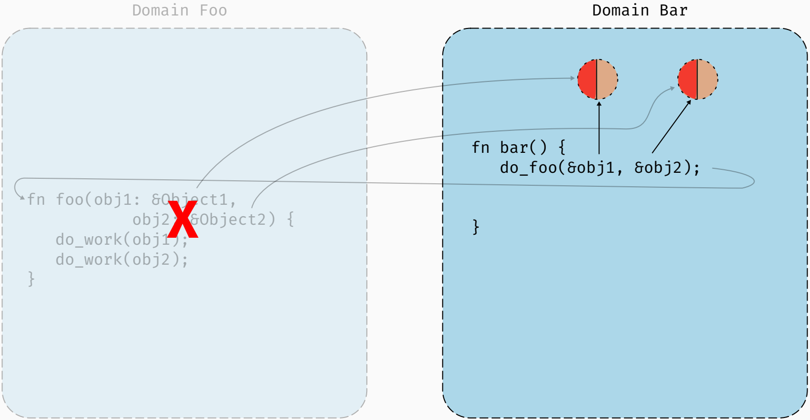
- e.g., SPIN (using Modula-3 pointers)

Fault isolation in Language-based systems



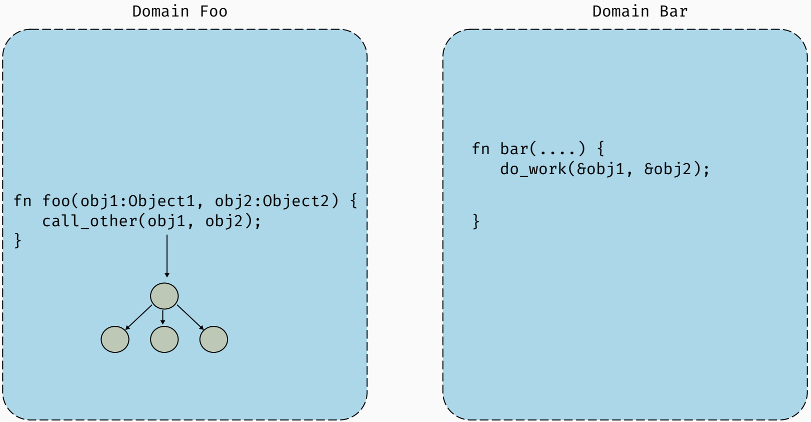
- e.g., SPIN (using Modula-3 pointers)

Fault isolation in Language-based systems



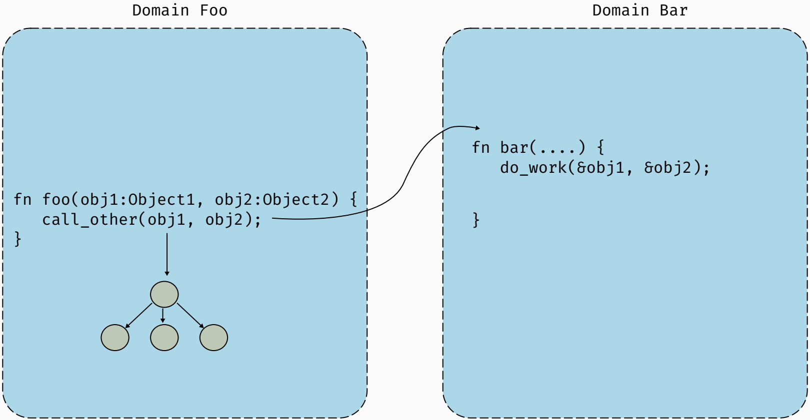
- e.g., SPIN (using Modula-3 pointers)

Language-based isolation: Deep copy



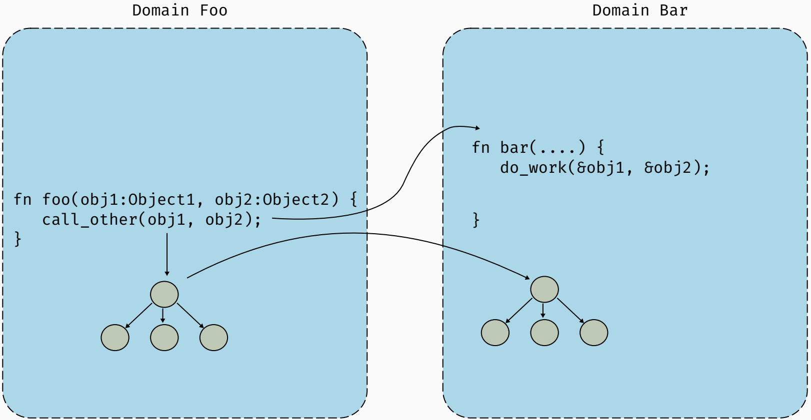
- e.g., J-Kernel, KaffeOS

Language-based isolation: Deep copy



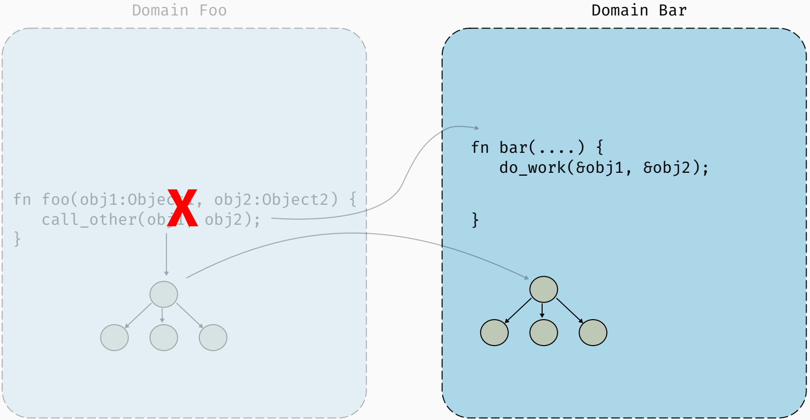
- e.g., J-Kernel, KaffeOS

Language-based isolation: Deep copy



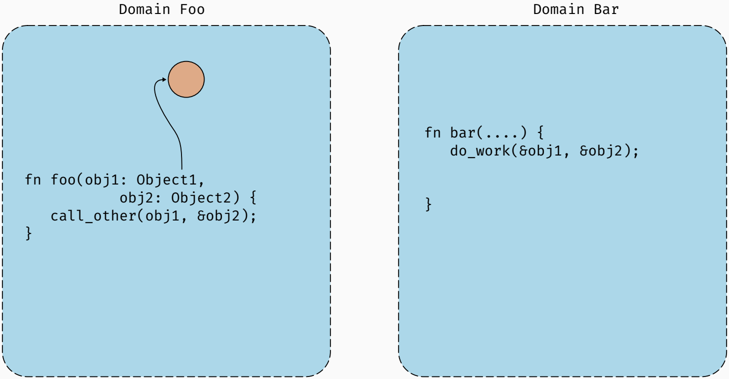
- e.g., J-Kernel, KaffeOS

Language-based isolation: Deep copy



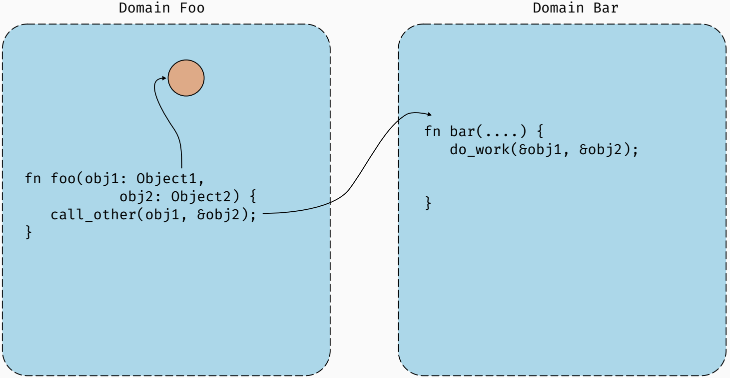
- e.g., J-Kernel, KaffeOS

Language-based isolation: Capabilities



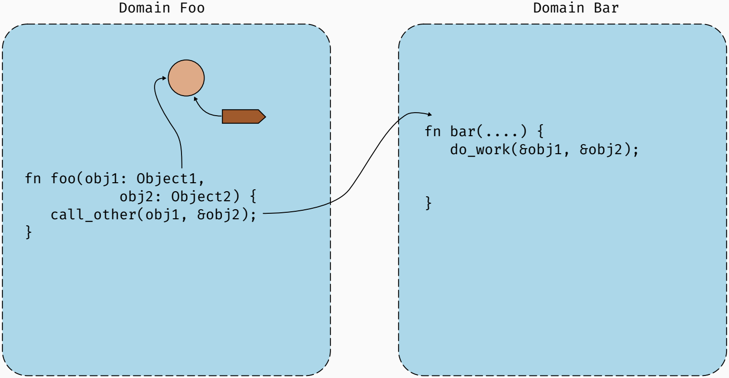
- e.g., J-Kernel

Language-based isolation: Capabilities



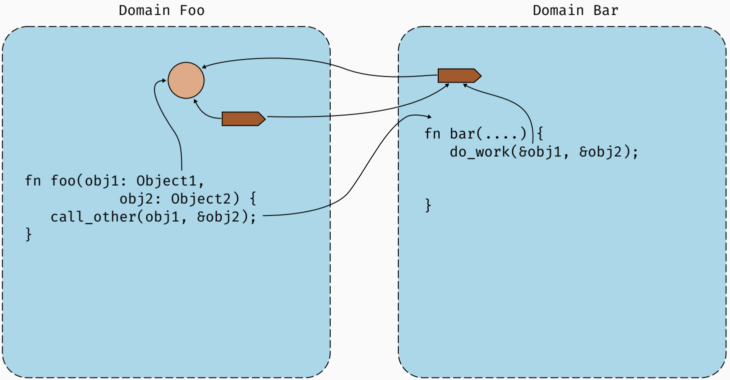
- e.g., J-Kernel

Language-based isolation: Capabilities



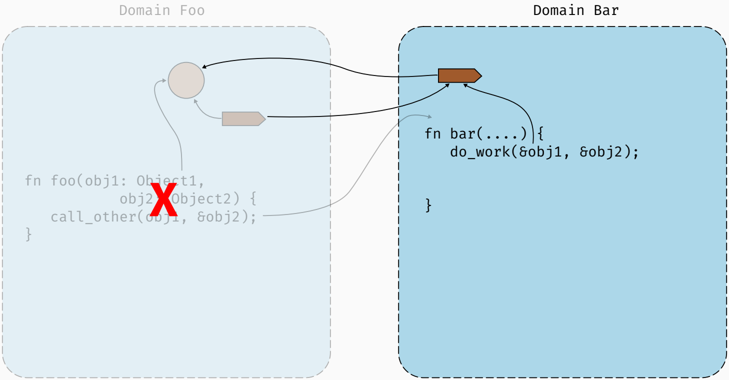
- e.g., J-Kernel

Language-based isolation: Capabilities



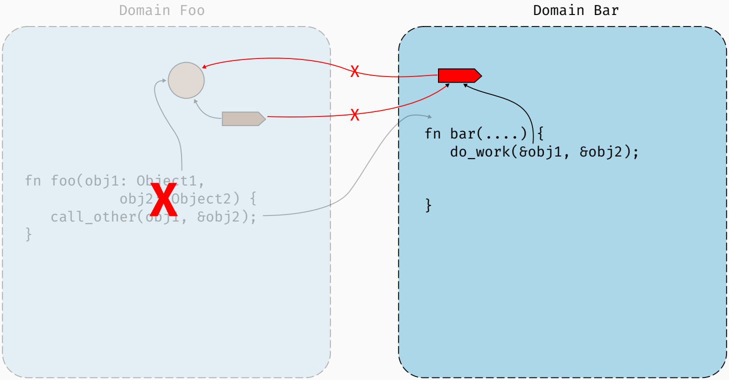
- e.g., J-Kernel

Language-based isolation: Capabilities



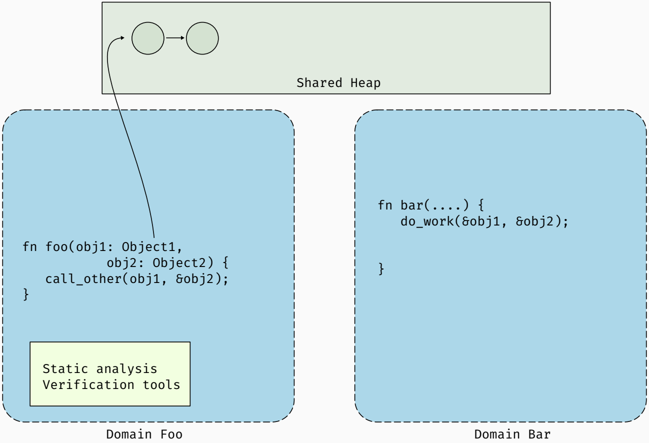
- e.g., J-Kernel

Language-based isolation: Capabilities



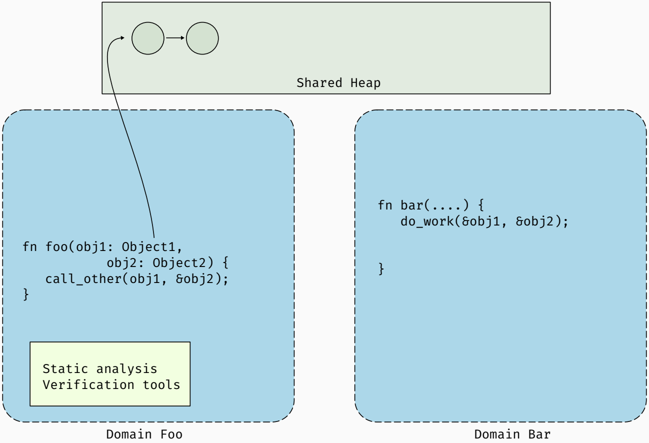
- e.g., J-Kernel

Language-based isolation: Singularity



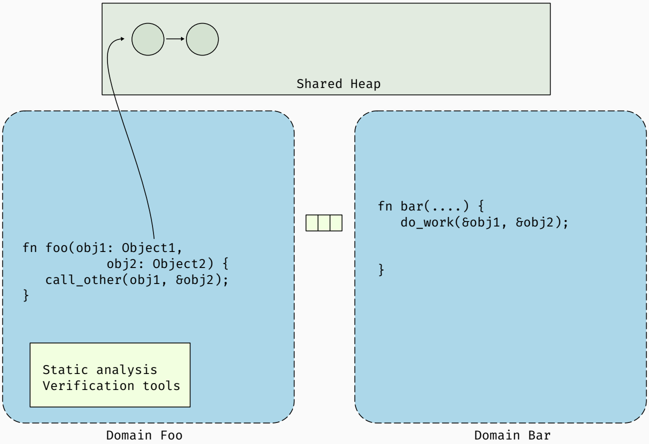
- Statically enforced ownership discipline

Language-based isolation: Singularity



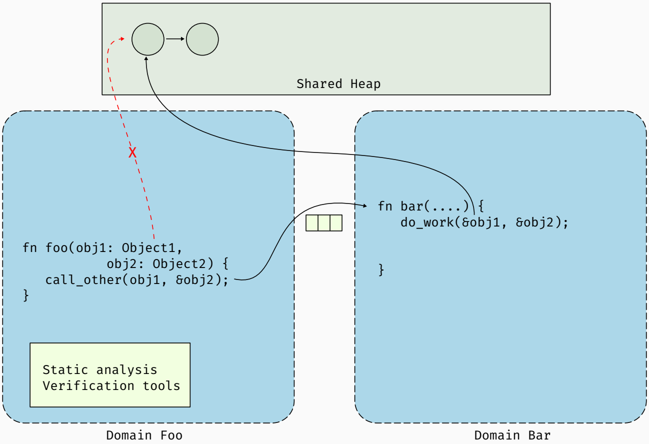
- Statically enforced ownership discipline
- Single ownership model

Language-based isolation: Singularity



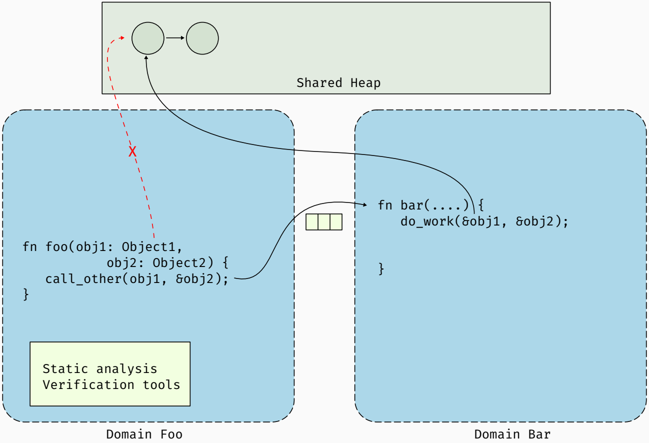
- Statically enforced ownership discipline
- Single ownership model

Language-based isolation: Singularity



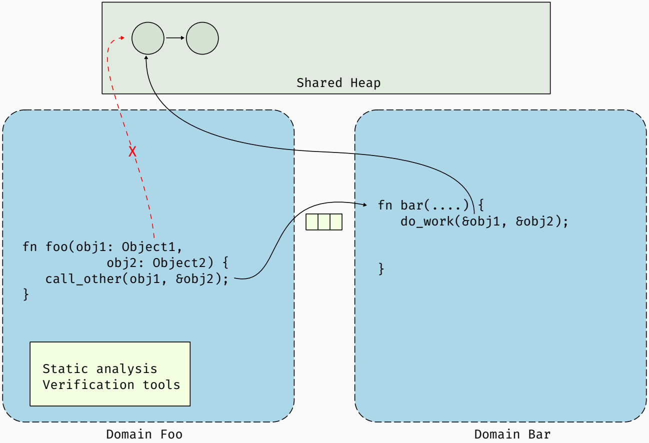
- Statically enforced ownership discipline
- Single ownership model

Language-based isolation: Singularity



- Statically enforced ownership discipline
- Single ownership model
- Static analysis and verification tools (Sing#)

Language-based isolation: Singularity

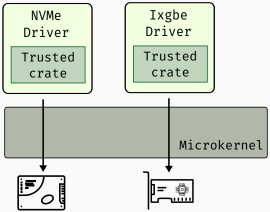


- Statically enforced ownership discipline
- Single ownership model
- Static analysis and verification tools (Sing#)
- Reusing the moved object return an error
- zero-copy

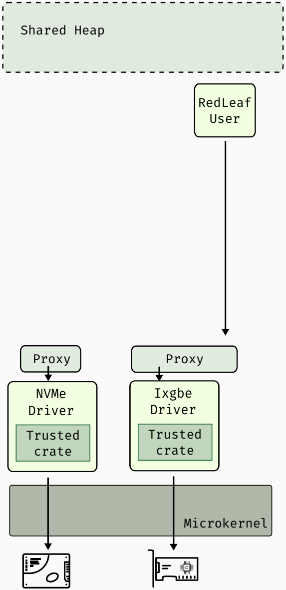
RedLeaf

Architecture

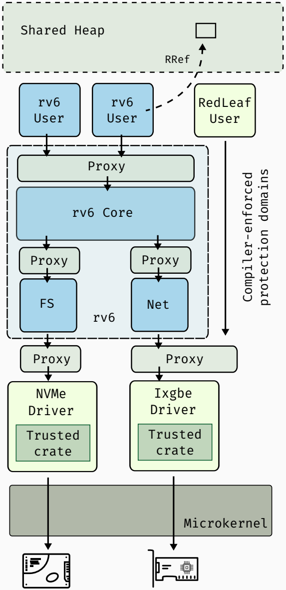
Architecture



Architecture



Architecture



Fault Isolation

- After a domain crash:

Fault Isolation

- After a domain crash:
  - Unwind all threads running inside

Fault Isolation

- After a domain crash:
    - Unwind all threads running inside
    - Subsequent invocations return error

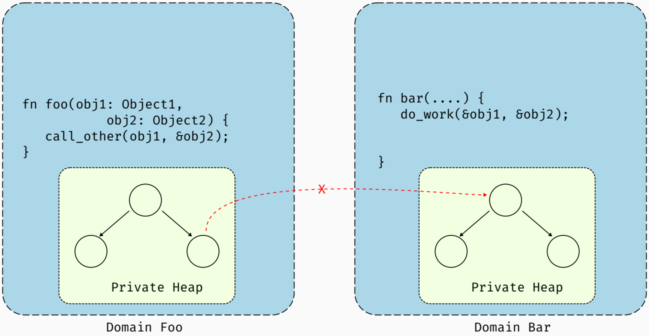
Fault Isolation

- After a domain crash:
  - Unwind all threads running inside
  - Subsequent invocations return error
  - All resources are deallocated

Fault Isolation

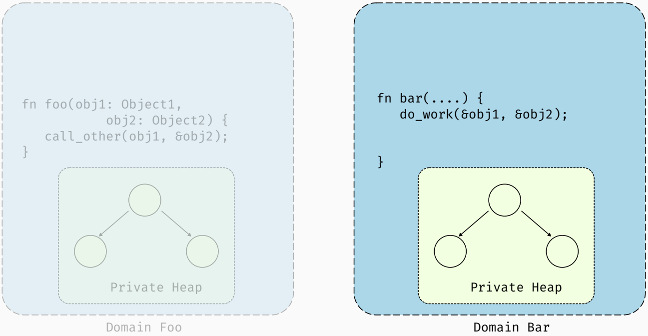
- After a domain crash:
  - Unwind all threads running inside
  - Subsequent invocations return error
  - All resources are deallocated
  - Other threads continue execution

Heap Isolation



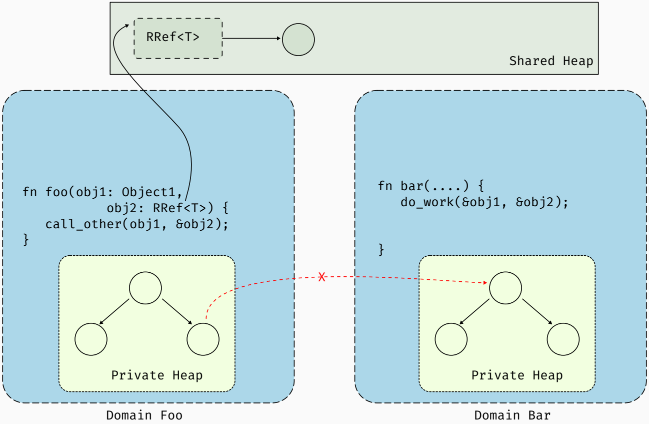
- Domains never hold pointers into other domains

Heap Isolation



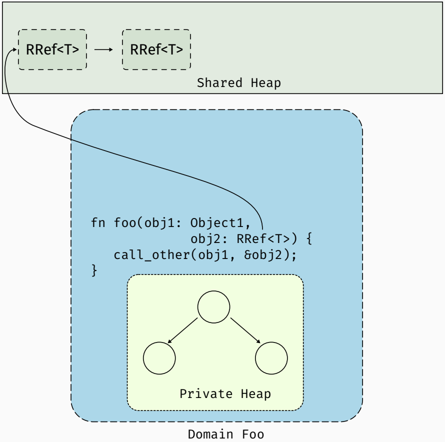
- Domains never hold pointers into other domains

Heap Isolation



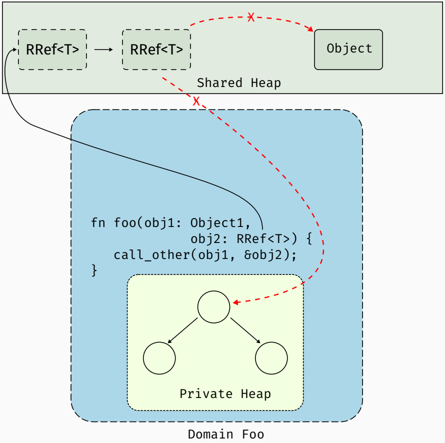
- Domains never hold pointers into other domains
- Special shared heap for passing objects between domains

Exchangeable types



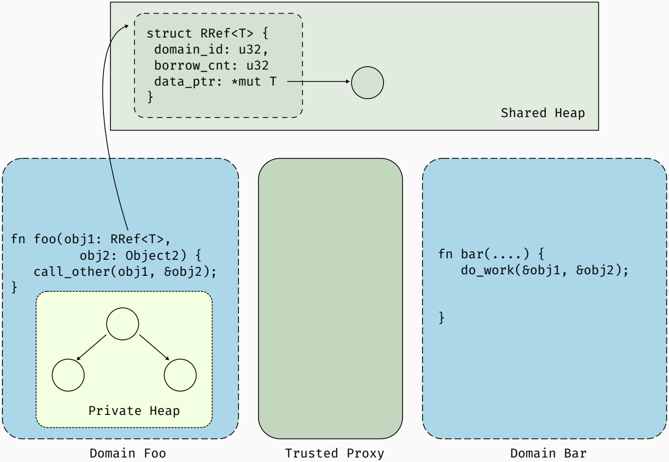
- Objects in shared heap can only be exchangeable types

Exchangeable types



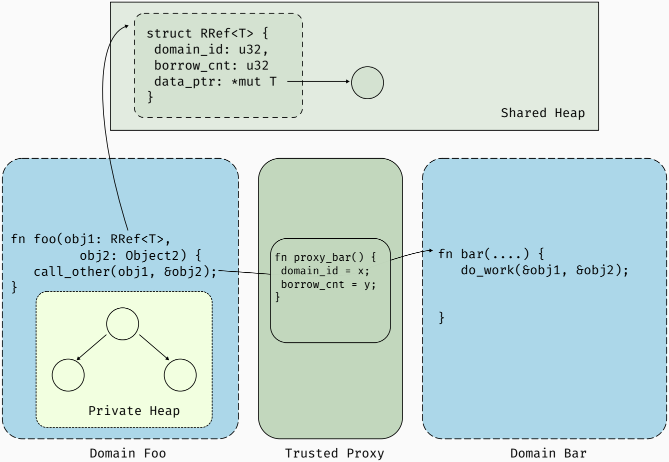
- Objects in shared heap can only be exchangeable types
- Cannot point to normal pointers in shared heap or private heap

Ownership tracking



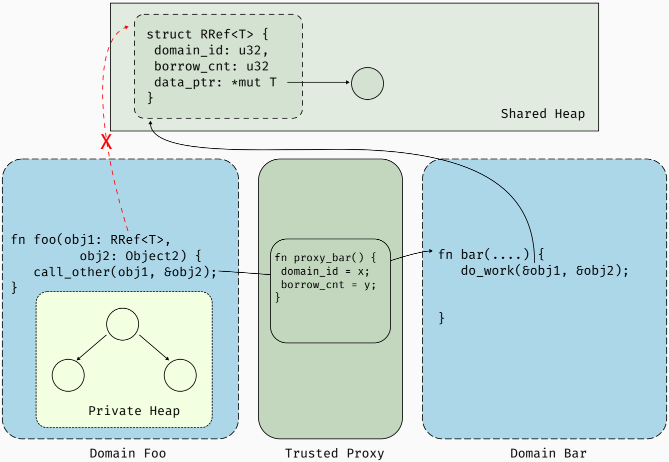
- RRef<T>'s can be passed between domains

Ownership tracking



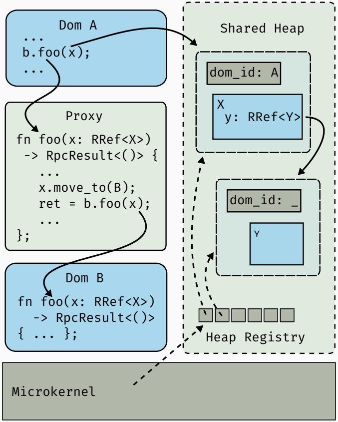
- `RRef<T>`'s can be passed between domains
- Metadata keeps track of owner domain and ref count

Ownership tracking



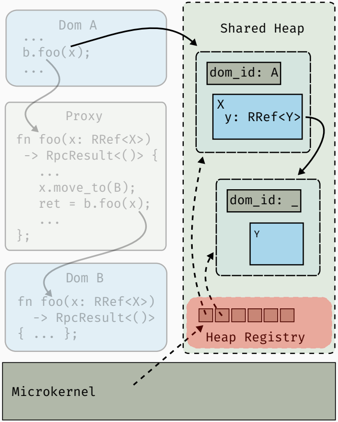
- `RRef<T>`'s can be passed between domains
- Metadata keeps track of owner domain and ref count
- Mediated through trusted proxies

Heap reclamation



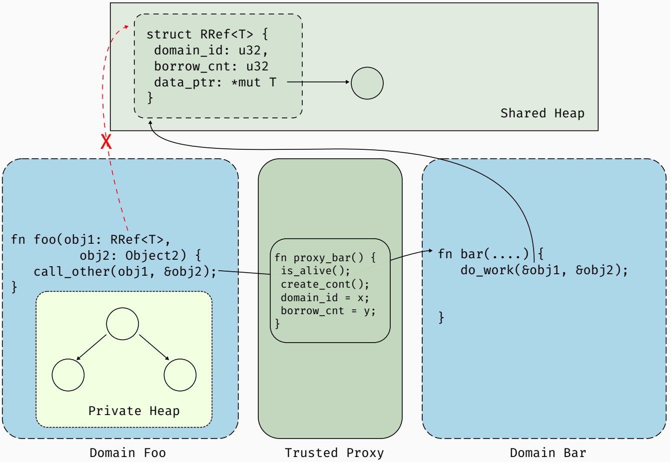
- Maintains a global registry of allocated objects

Heap reclamation



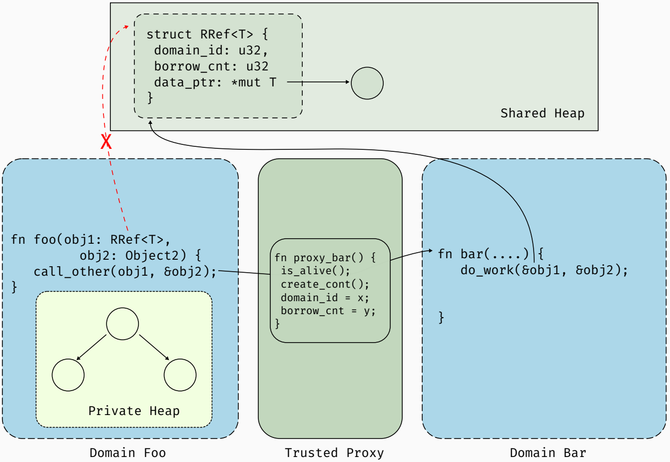
- Maintains a global registry of allocated objects
- On panic
  - Deallocate all objects owned by the crashing domain
  - Defer borrowed `RRef`'s until ref count is zero

Cross-domain call proxying



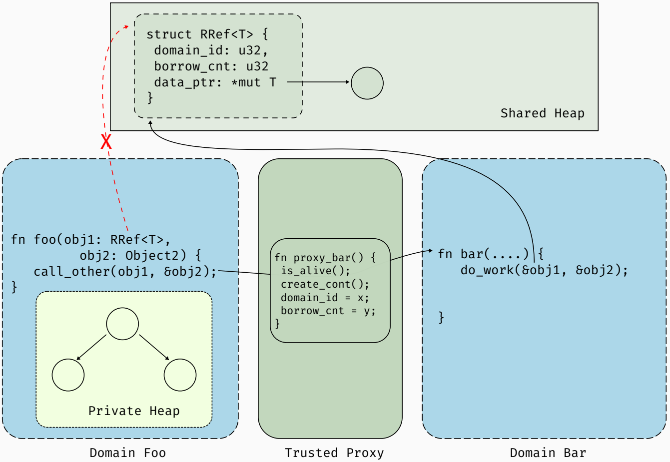
- Checks if domain is alive

Cross-domain call proxying



- Checks if domain is alive
- Creates continuation

Cross-domain call proxying



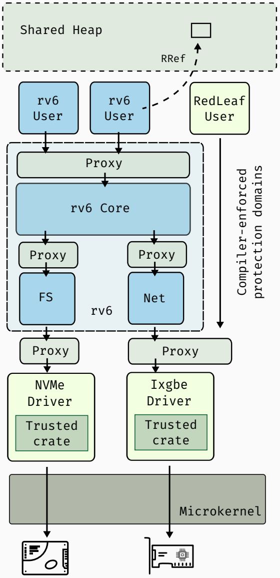
- Checks if domain is alive
- Creates continuation
- Moves ownership of all `RRef<T>`

Interface validation

- Validate domain interfaces
- Generate proxies to enforce ownership discipline
- e.g., Block Device domain Interface

```
pub trait BDev {
  fn read(&self, block: u32, data: RRef<[u8; BSIZE]>)
          -> RpcResult<RRef<[u8; BSIZE]>>;
  fn write(&self, block: u32, data: &RRef<[u8; BSIZE]>)
          -> RpcResult<()>;
}
```

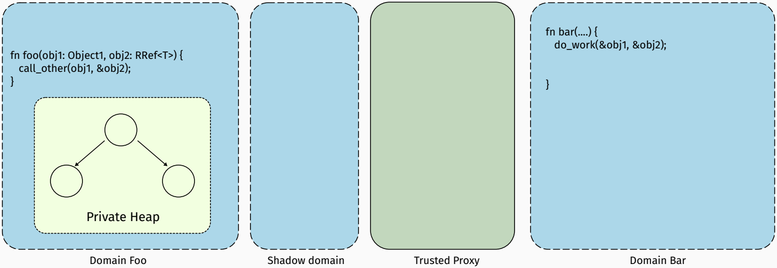
RedLeaf



- Heap isolation
- Exchangeable types
- Ownership tracking
- Interface validation
- Cross-domain call proxying

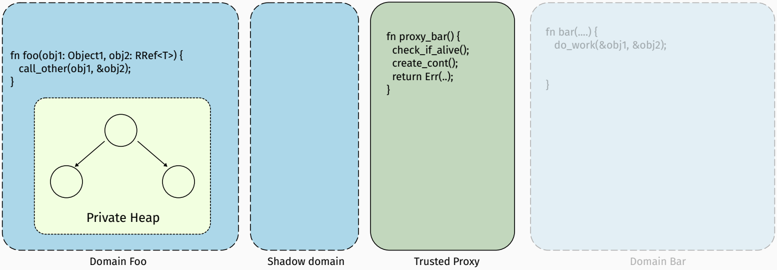
Device driver Recovery

Device driver Recovery



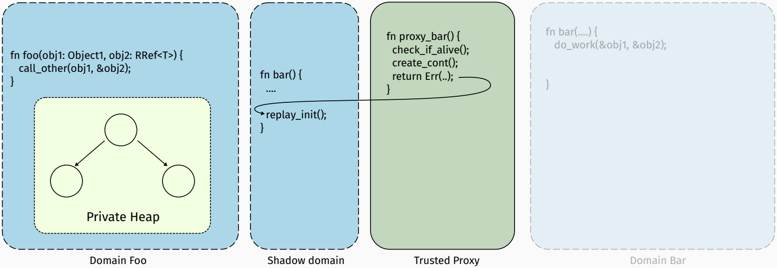
- Support transparent device driver recovery
- Wraps the interaface to expose an identical interface
- Interposes on all communication

Device driver Recovery



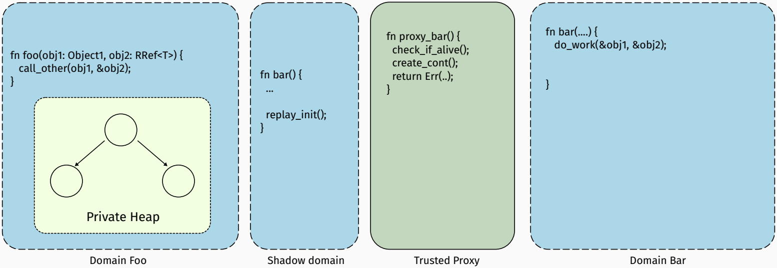
- Support transparent device driver recovery
- Wraps the interaface to expose an identical interface
- Interposes on all communication

Device driver Recovery



- Support transparent device driver recovery
- Wraps the interaface to expose an identical interface
- Interposes on all communication

Device driver Recovery



- Support transparent device driver recovery
- Wraps the interaface to expose an identical interface
- Interposes on all communication

Evaluation

System setup

- 2 x Intel E5-2660 v3 10-core CPUs at 2.60 GHz (Haswell EP)
- Disabled: Hyper-threading, Turbo boost, CPU Idle states
- Linux and DPDK benchmarks run on version 4.8.4
- RedLeaf benchmarks run on baremetal

Communication costs

Operation	Cycles
seL4	834
VMFUNC	169
VMFUNC-based call/reply invocation	396
RedLeaf cross-domain invocation	124
RedLeaf cross-domain invocation (passing an RRef)	141
RedLeaf cross-domain invocation via shadow	279
RedLeaf cross-domain via shadow (passing an RRef)	297

Language overheads: C vs Rust

- Hashtable - (FNV hash, open addressing, <8B, 8B>)

Language overheads: C vs Rust

- Hashtable - (FNV hash, open addressing, <8B, 8B>)
- C, Idiomatic Rust, C-style Rust,

Language overheads: C vs Rust

- Hashtable - (FNV hash, open addressing, <8B, 8B>)
- C, Idiomatic Rust, C-style Rust,
- C-style Rust: No higher order functions `usize, usize`

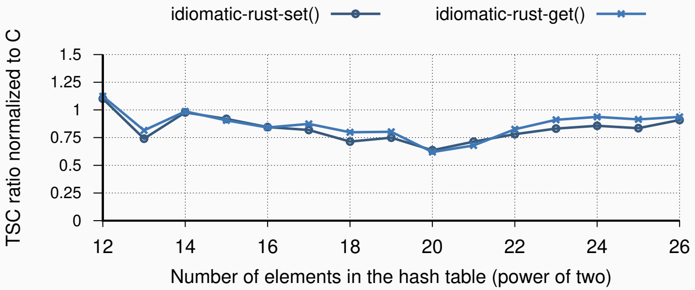
Language overheads: C vs Rust

- Hashtable - (FNV hash, open addressing, <8B, 8B>)
- C, Idiomatic Rust, C-style Rust,
- C-style Rust: No higher order functions `usize, usize`
- Idiomatic Rust - `Option<(usize, usize)>`

Language overheads: C vs Rust

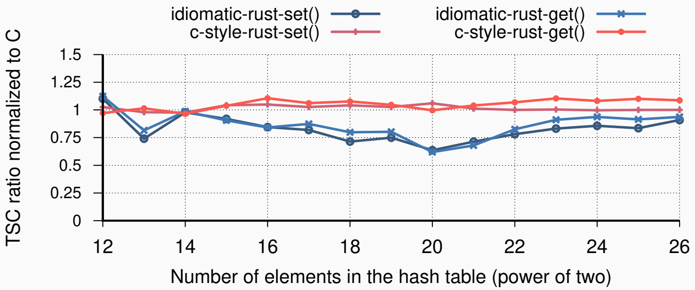
- Hashtable - (FNV hash, open addressing, <8B, 8B>)
- C, Idiomatic Rust, C-style Rust,
- C-style Rust: No higher order functions `usize, usize`
- Idiomatic Rust - `Option<(usize, usize)>`
- Vary the size (2^{12} to 2^{26} at 75% full)

Language overheads: C vs Rust



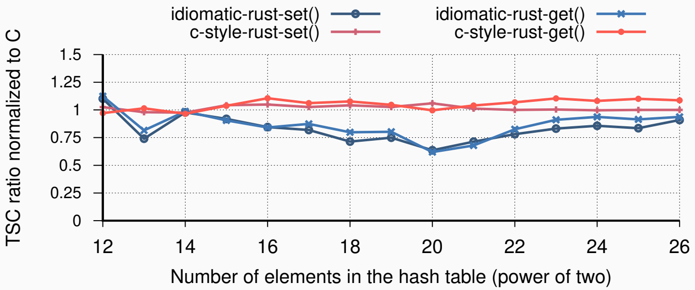
- Hashtable - (FNV hash, open addressing, <8B, 8B>)
- C, Idiomatic Rust, C-style Rust,
- C-style Rust: No higher order functions `usize, usize`
- Idiomatic Rust - `Option<(usize, usize)>`
- Vary the size (2^{12} to 2^{26} at 75% full)
- Idiomatic Rust - 25% overhead, C-style - Closer to C

Language overheads: C vs Rust



- Hashtable - (FNV hash, open addressing, <8B, 8B>)
- C, Idiomatic Rust, C-style Rust,
- C-style Rust: No higher order functions `usize, usize`
- Idiomatic Rust - `Option<(usize, usize)>`
- Vary the size (2^{12} to 2^{26} at 75% full)
- Idiomatic Rust - 25% overhead, C-style - Closer to C

Language overheads: C vs Rust

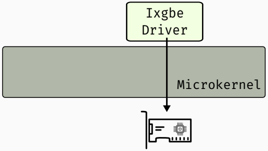


- Hashtable - (FNV hash, open addressing, <8B, 8B>)
- C, Idiomatic Rust, C-style Rust,
- C-style Rust: No higher order functions `usize, usize`
- Idiomatic Rust - `Option<(usize, usize)>`
- Vary the size (2^{12} to 2^{26} at 75% full)
- Idiomatic Rust - 25% overhead, C-style - Closer to C
- Cycles for a `set` on 2^{20} (C - 51, idrust - 81, cst-rust - 48)

Case Study: Device Drivers

Configurations

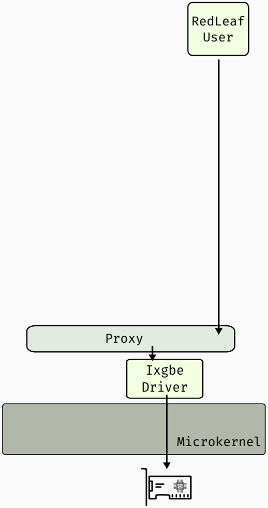
- redleaf-driver



Case Study: Device Drivers

Configurations

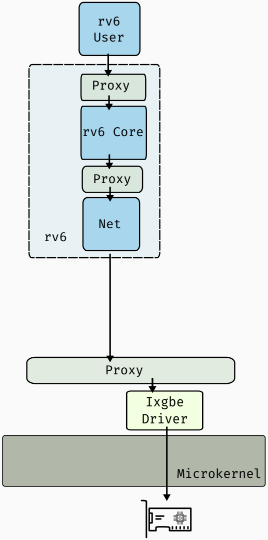
- redleaf-driver
- redleaf-domain



Case Study: Device Drivers

Configurations

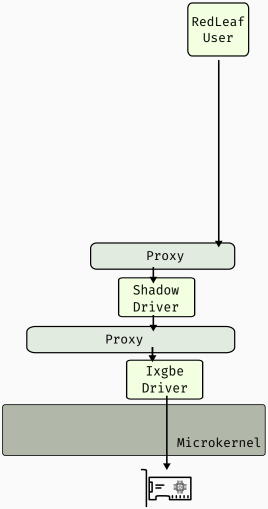
- redleaf-driver
- redleaf-domain
- rv6-domain



Case Study: Device Drivers

Configurations

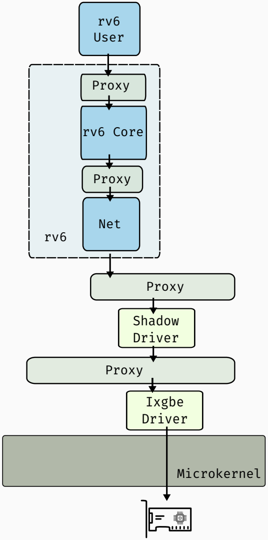
- redleaf-driver
- redleaf-domain
- rv6-domain
- redleaf-shadow



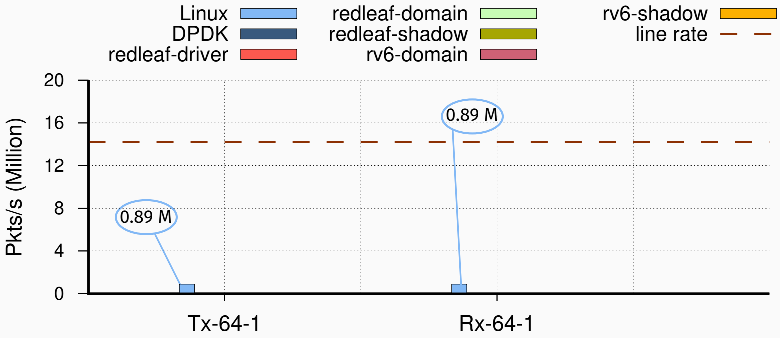
Case Study: Device Drivers

Configurations

- redleaf-driver
- redleaf-domain
- rv6-domain
- redleaf-shadow
- rv6-shadow

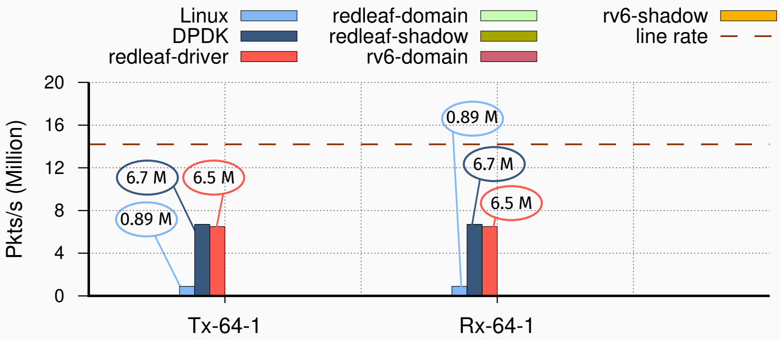


Ixgbe performance benchmark



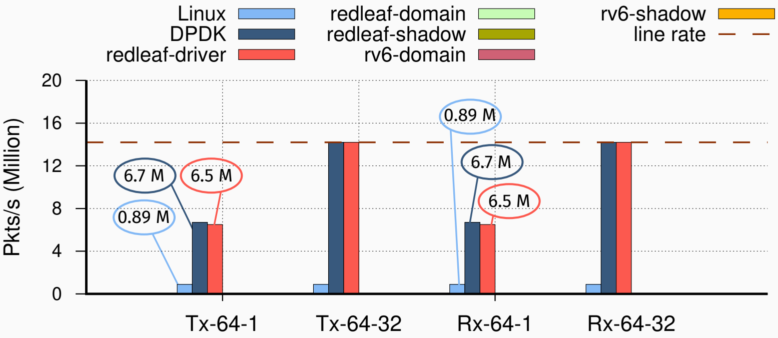
- Linux application (2921 cycles)

Ixgbe performance benchmark



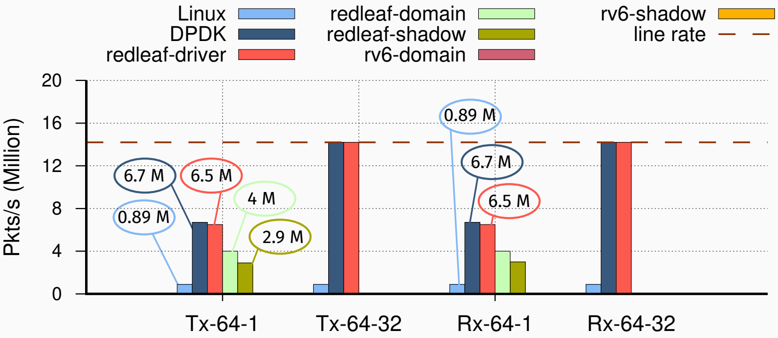
- Linux application (2921 cycles)
- DPDK (388 cycles) and Redleaf driver (400 cycles)

Ixgbe performance benchmark



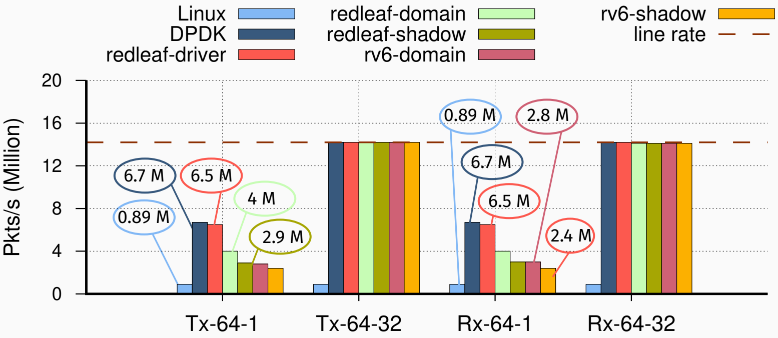
- Linux application (2921 cycles)
- DPDK (388 cycles) and Redleaf driver (400 cycles)

Ixgbe performance benchmark



- Linux application (2921 cycles)
- DPDK (388 cycles) and Redleaf driver (400 cycles)
- Redleaf-domain and Redleaf-shadow

Ixgbe performance benchmark



- Linux application (2921 cycles)
- DPDK (388 cycles) and Redleaf driver (400 cycles)
- Redleaf-domain and Redleaf-shadow
- Rv6-domain and shadow

Application Benchmarks

- Maglev load balancer

Application Benchmarks

- Maglev load balancer
- Network-attached key-value store

Application Benchmarks

- Maglev load balancer
- Network-attached key-value store
- a minimal webserver

Application benchmarks: Maglev

- Load balancer developed by Google

Application benchmarks: Maglev

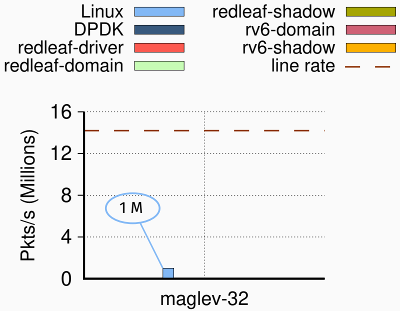
- Load balancer developed by Google
- Lookup or insert into the flow tracking table

Application benchmarks: Maglev

- Load balancer developed by Google
- Lookup or insert into the flow tracking table
- Compare C vs Rust Maglev versions

Application benchmarks: Maglev

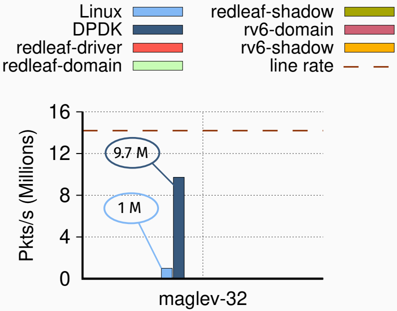
- Load balancer developed by Google
- Lookup or insert into the flow tracking table
- Compare C vs Rust Maglev versions



- Linux Application - Limited due to synchronous socket interface

Application benchmarks: Maglev

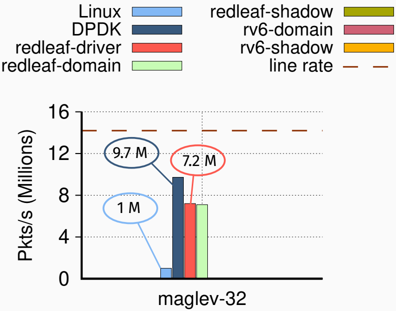
- Load balancer developed by Google
- Lookup or insert into the flow tracking table
- Compare C vs Rust Maglev versions



- Linux Application - Limited due to synchronous socket interface
- DPDK Application (batch of 32) - 9.7 Mpps per-core

Application benchmarks: Maglev

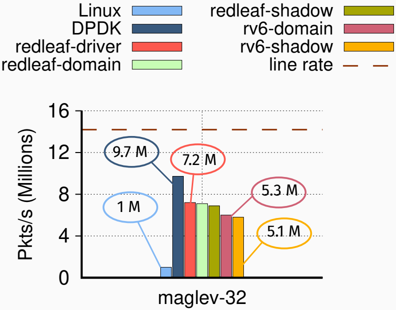
- Load balancer developed by Google
- Lookup or insert into the flow tracking table
- Compare C vs Rust Maglev versions



- Linux Application - Limited due to synchronous socket interface
- DPDK Application (batch of 32) - 9.7 Mpps per-core
- Redleaf-driver - 7.2 Mpps

Application benchmarks: Maglev

- Load balancer developed by Google
- Lookup or insert into the flow tracking table
- Compare C vs Rust Maglev versions



- Linux Application - Limited due to synchronous socket interface
- DPDK Application (batch of 32) - 9.7 Mpps per-core
- Redleaf-driver - 7.2 Mpps
- Rv6-domain - 5.3 Mpps, Rv6-shadow - 5.1 Mpps

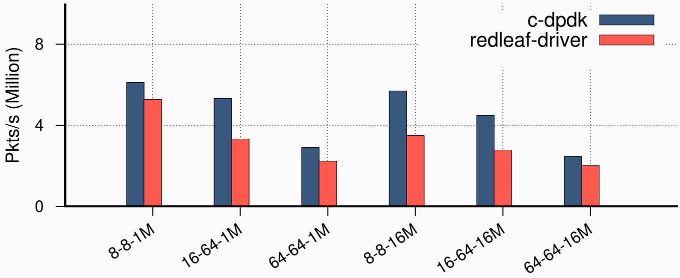
Application: Key Value Store

- Network attached key-value store

Application: Key Value Store

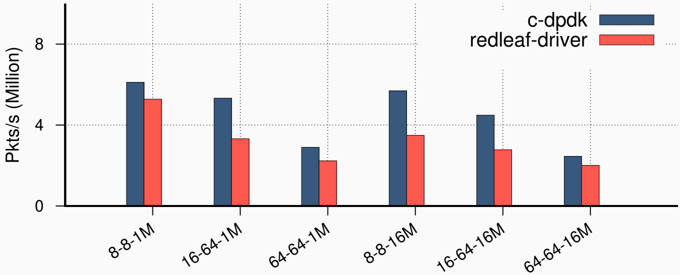
- Network attached key-value store
- FNV Hash with open addressing (linear probing)

Application: Key Value Store



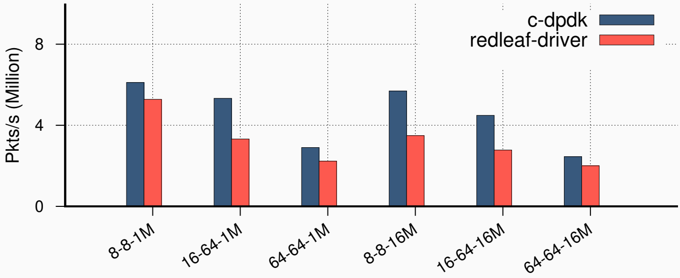
- Network attached key-value store
- FNV Hash with open addressing (linear probing)
- Rust (C-Style) vs a DPDK application

Application: Key Value Store



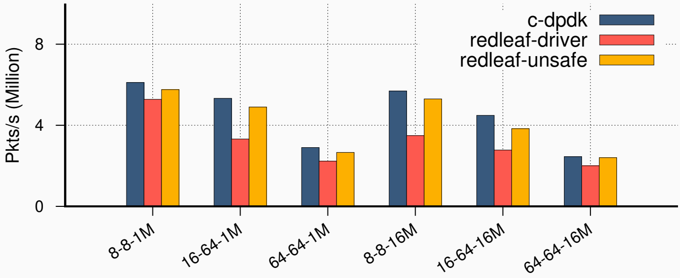
- Network attached key-value store
- FNV Hash with open addressing (linear probing)
- Rust (C-Style) vs a DPDK application
- Various Key value sizes (<8B, 8B>, <16B, 64B>, <64B, 64B>)

Application: Key Value Store



- Network attached key-value store
- FNV Hash with open addressing (linear probing)
- Rust (C-Style) vs a DPDK application
- Various Key value sizes (<8B, 8B>, <16B, 64B>, <64B, 64B>)
- Achieves 61-86% performance (`extend_from_slice() x 3`)

Application: Key Value Store



- Network attached key-value store
- FNV Hash with open addressing (linear probing)
- Rust (C-Style) vs a DPDK application
- Various Key value sizes (<8B, 8B>, <16B, 64B>, <64B, 64B>)
- Achieves 61-86% performance (`extend_from_slice() x 3`)
- With Unsafe Rust 85-94% performance of the C DPDK

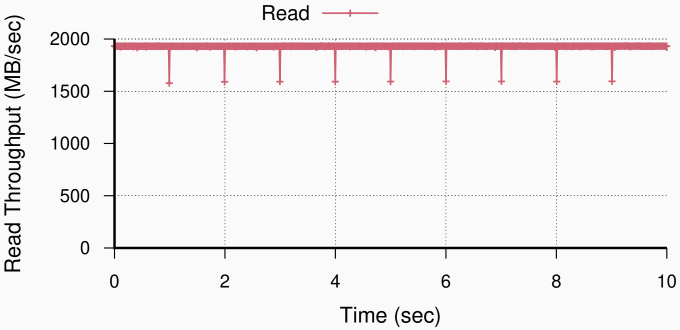
Device driver recovery

- Rv6 program <-> in-memory block device

Device driver recovery

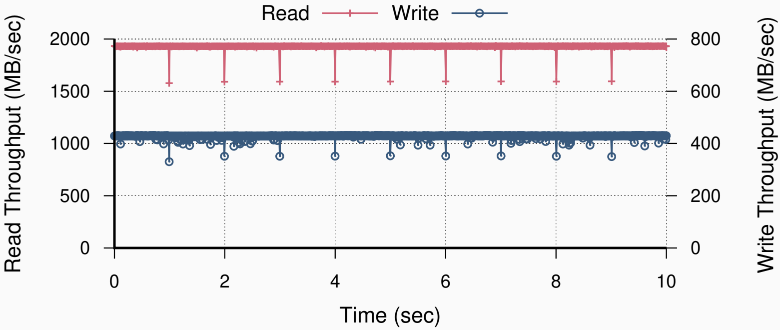
- Rv6 program <-> in-memory block device
- Trigger crash every second

Device driver recovery



- Rv6 program <-> in-memory block device
- Trigger crash every second
- Small drop in performance
    - Read: 2062 MB/s (restart), 2164 MB/s (without restart)

Device driver recovery



- Rv6 program <-> in-memory block device
- Trigger crash every second
- Small drop in performance
  - Read: 2062 MB/s (restart), 2164 MB/s (without restart)
  - Writes: 356 MB/s (restart), 423 MB/s (without restart)

Conclusion

- Heap isolation, exchangeable types, ownership tracking, interface validation, cross-domain call proxying
- Provides a collection of mechanisms for enabling isolation
- A step forward in enabling future system architectures
  - Secure kernel extensions, fine-grained access control, transparent recovery etc.

Source code

`https://mars-research.github.io/redleaf/`

Thank you!