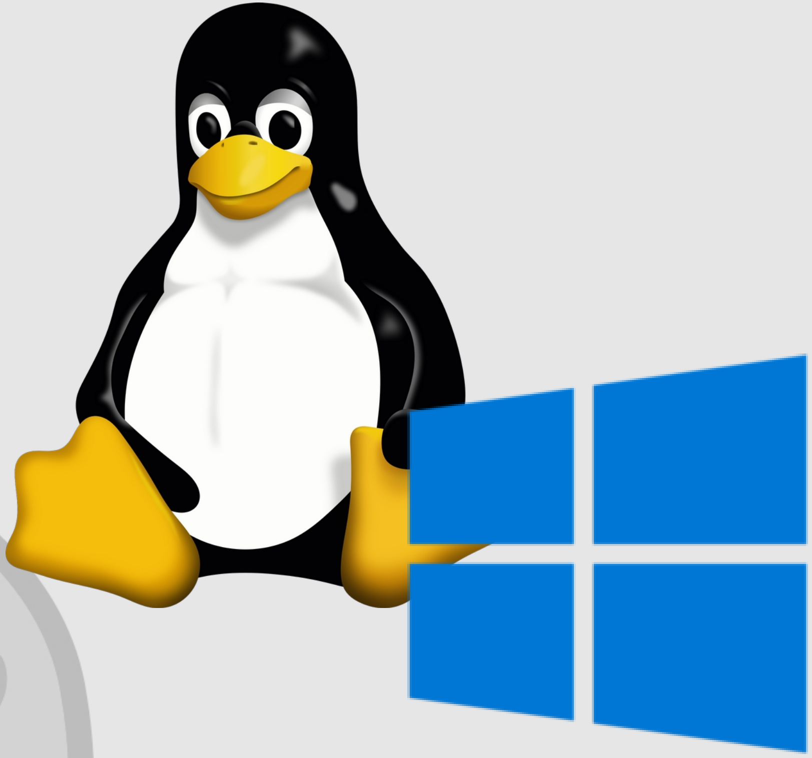
Ironclad: A formally verified OS kernel written in Ada

<streaksu@ironclad-os.org>

Mon, Apr 14

The context

- In the beginning: C based OSes, like Windows or Linux.
- Issues with scalability and widepread safety and security.
- Insuitability to safety critical operations and work.



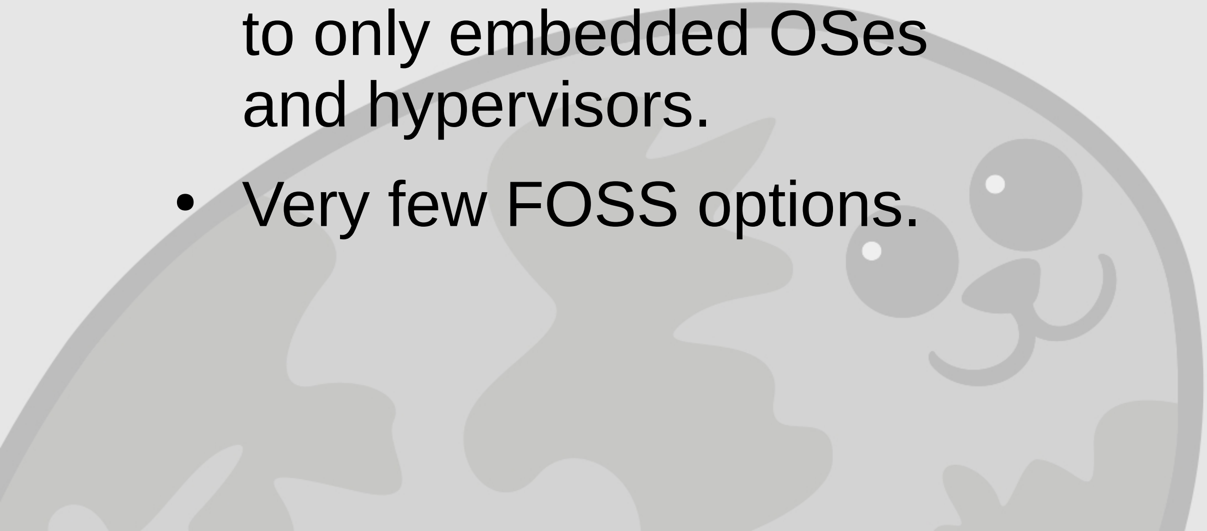
The context

- Lots of potential answers.

- Formal verification is underexplored and confined to only embedded OSes and hypervisors.
- Very few FOSS options.

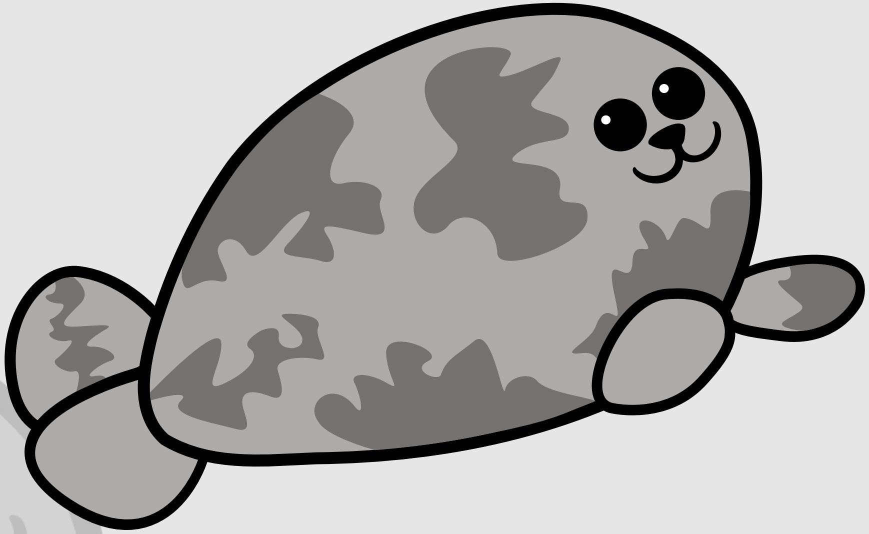
seL4

QNX



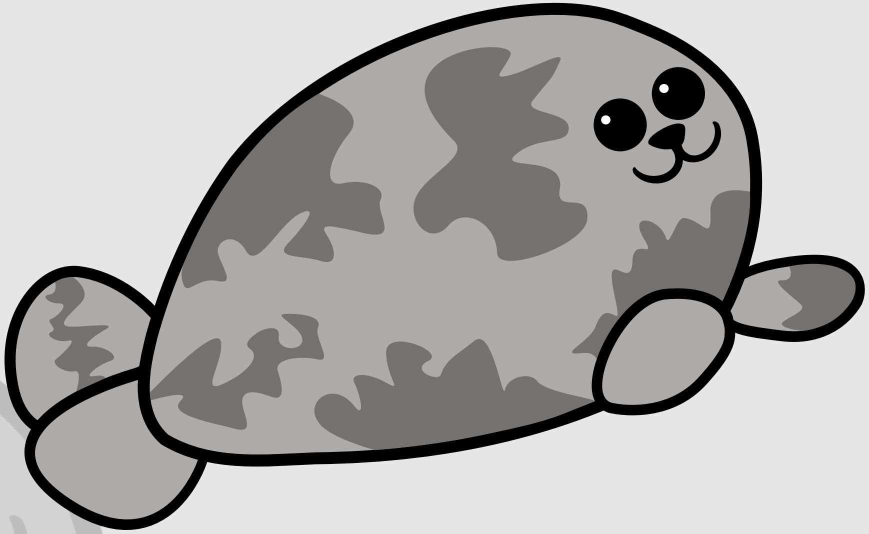
Enter Ironclad

- Introduction to formal verification.
- What is SPARK and how we use it for Ironclad.
- What we do that other systems don’t.



What is Ironclad?

- POSIX-compatible partially formally-verified kernel.
- Hard real-time and GP capable.
- ~100% Ada / SPARK code.
- Free as in freedom.



The most common question: why not Rust?

- Weak specifications for now (Ferrocene doesn’t really help).

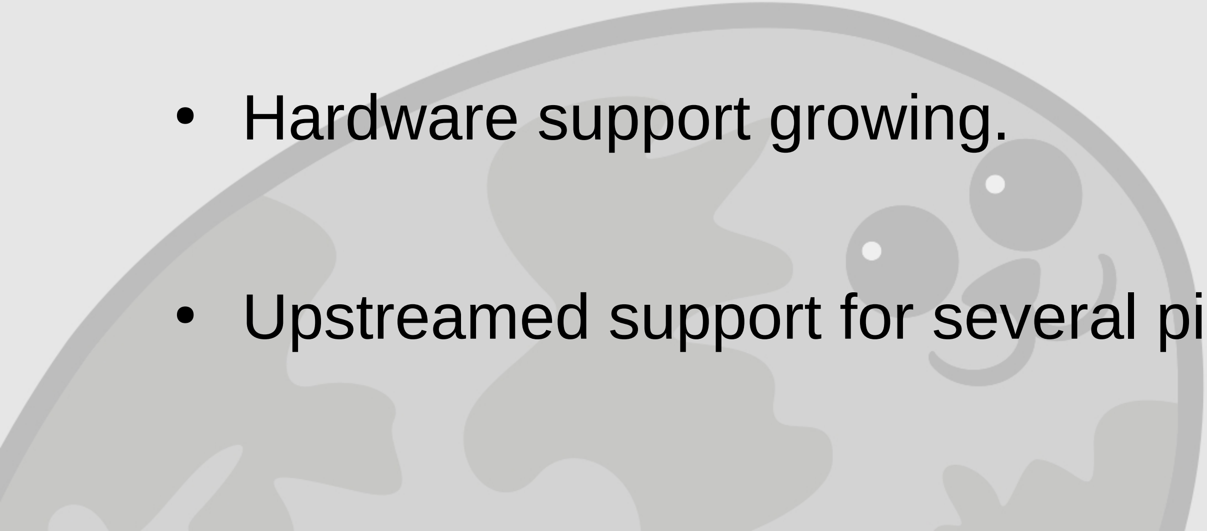
- Very lacking formal verification tooling.

History

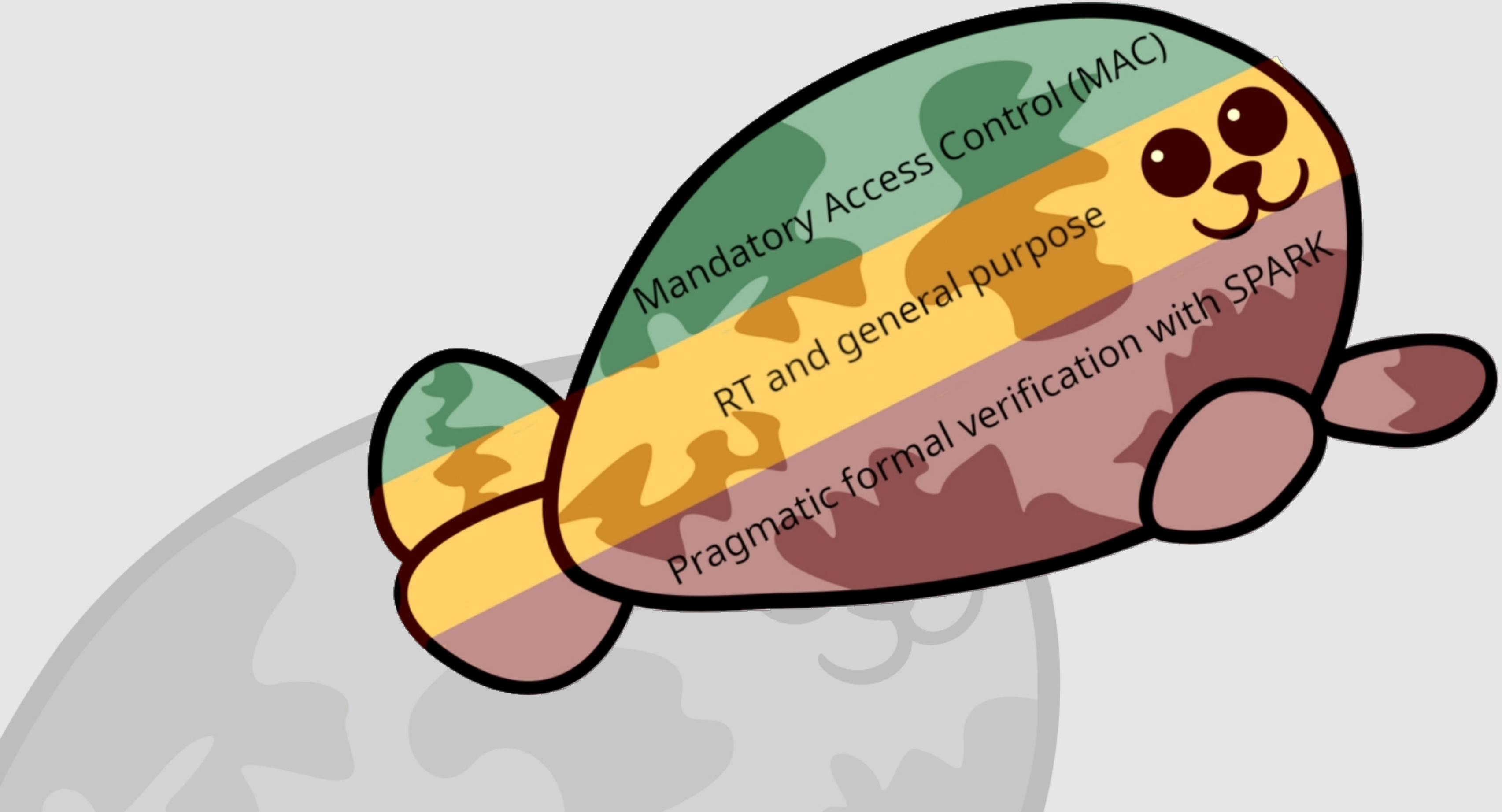
Ironclad is only a kernel

Where Ironclad is today

- Pretty small developer team.
- Gloire being the biggest and only FOSS distribution, and a growing community.
- Hardware support growing.
- Upstreamed support for several pieces of software.



What makes Ironclad special

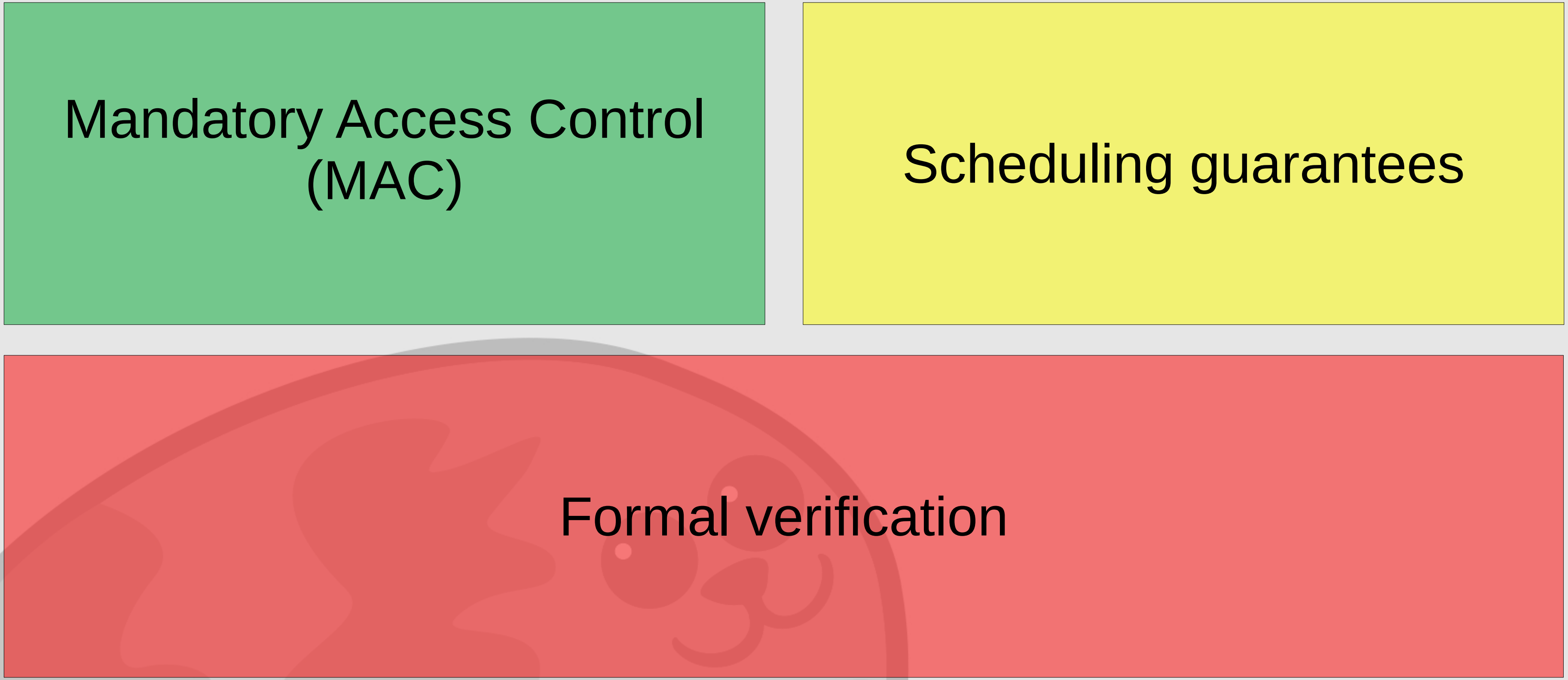


Mandatory Access Control (MAC)

RT and general purpose

Pragmatic formal verification with SPARK

Formal verification is a foundation

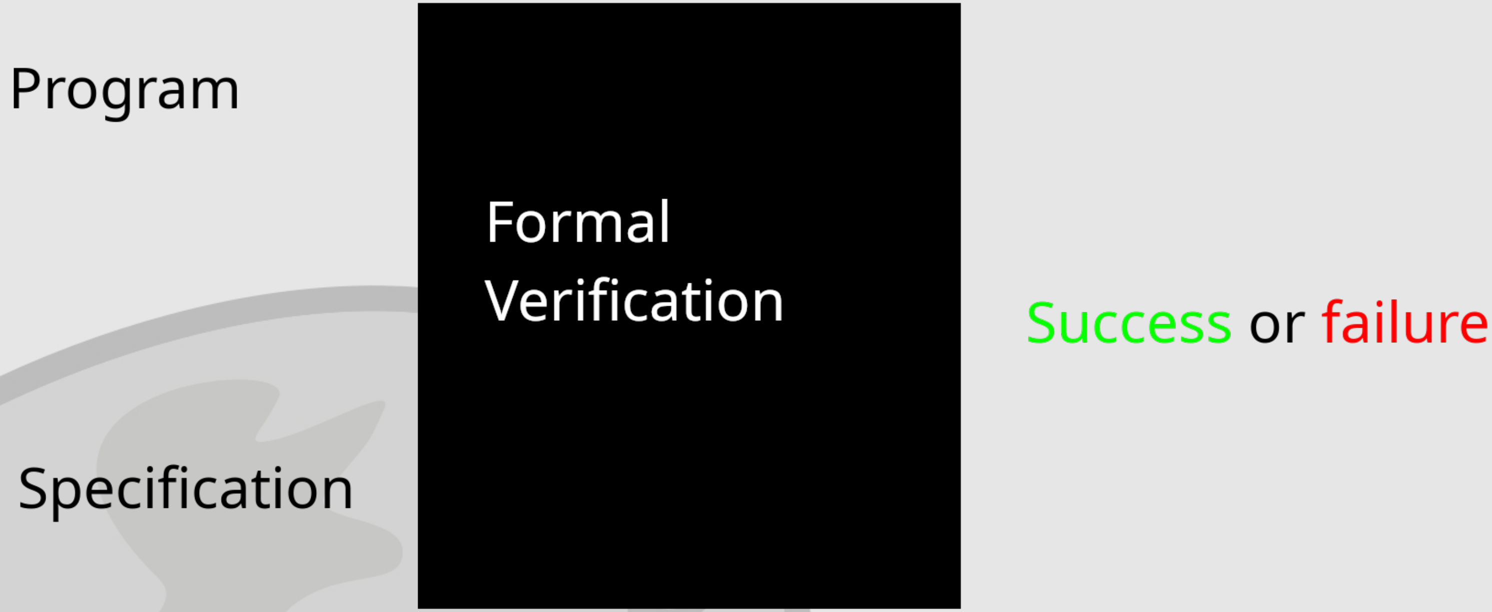


Mandatory Access Control (MAC)

Scheduling guarantees

Formal verification

What is formal verification, anyways?



Program

Specification

Formal Verification

Success or failure

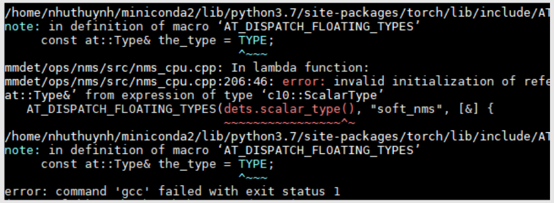
What is formal verification, anyways?

C program

C Specification

C Compiler's lexer + parser (frontend)

Success or



```
/home/nhuthuynh/miniconda2/lib/python3.7/site-packages/torch/lib/include/AT
note: in definition of macro ‘AT_DISPATCH_FLOATING_TYPES’
     const at::Type& the_type = TYPE;
                                ^~~~
mmdet/ops/nms/src/nms_cpu.cpp: In lambda function:
mmdet/ops/nms/src/nms_cpu.cpp:206:46: error: invalid initialization of refe
at::Type&’ from expression of type ‘c10::ScalarType’
   AT_DISPATCH_FLOATING_TYPES(dets.scalar_type(), "soft_nms", [&] {
                              ~~~~~~~~~~~~~~~~^~
/home/nhuthuynh/miniconda2/lib/python3.7/site-packages/torch/lib/include/AT
note: in definition of macro ‘AT_DISPATCH_FLOATING_TYPES’
     const at::Type& the_type = TYPE;
                                ^~~~
error: command 'gcc' failed with exit status 1
```

What is formal verification, anyways?

Boolean value

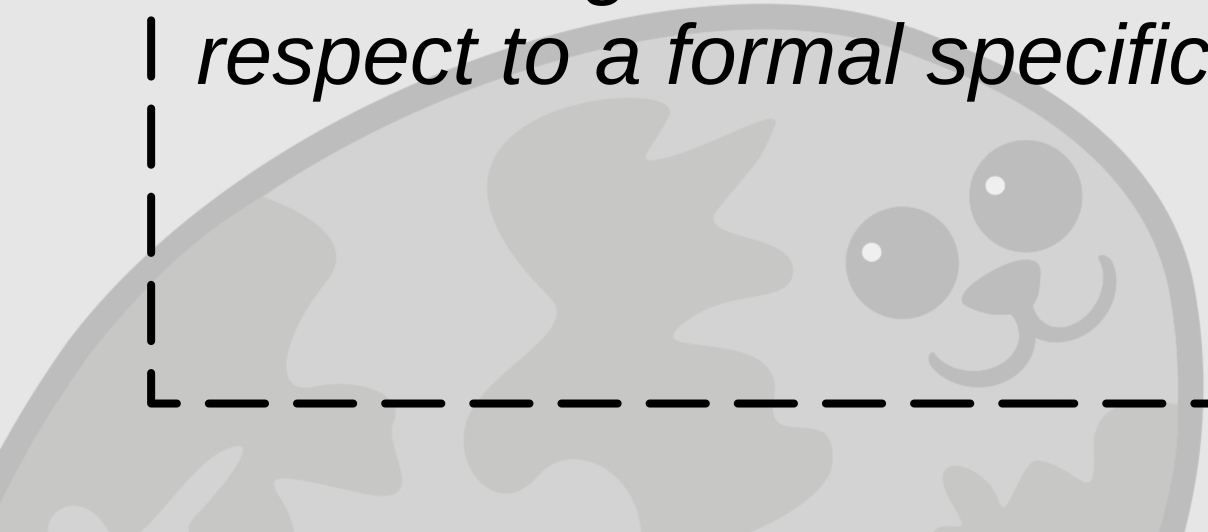
```
if (Input == True)
return Success;
else
return Failure;
```

Only true
gets past

Success or Failure

What is formal verification, anyways?

Checking the correctness of an input with respect to a formal specification (using math).



What is formal verification, anyways?

Ironclad code

Abscence of runtime errors (AoRTE)

Package specific contracts

GNATprove's SPARK

(why3 as SMT solver)

Success or failure

Why don’t we do it all the time?

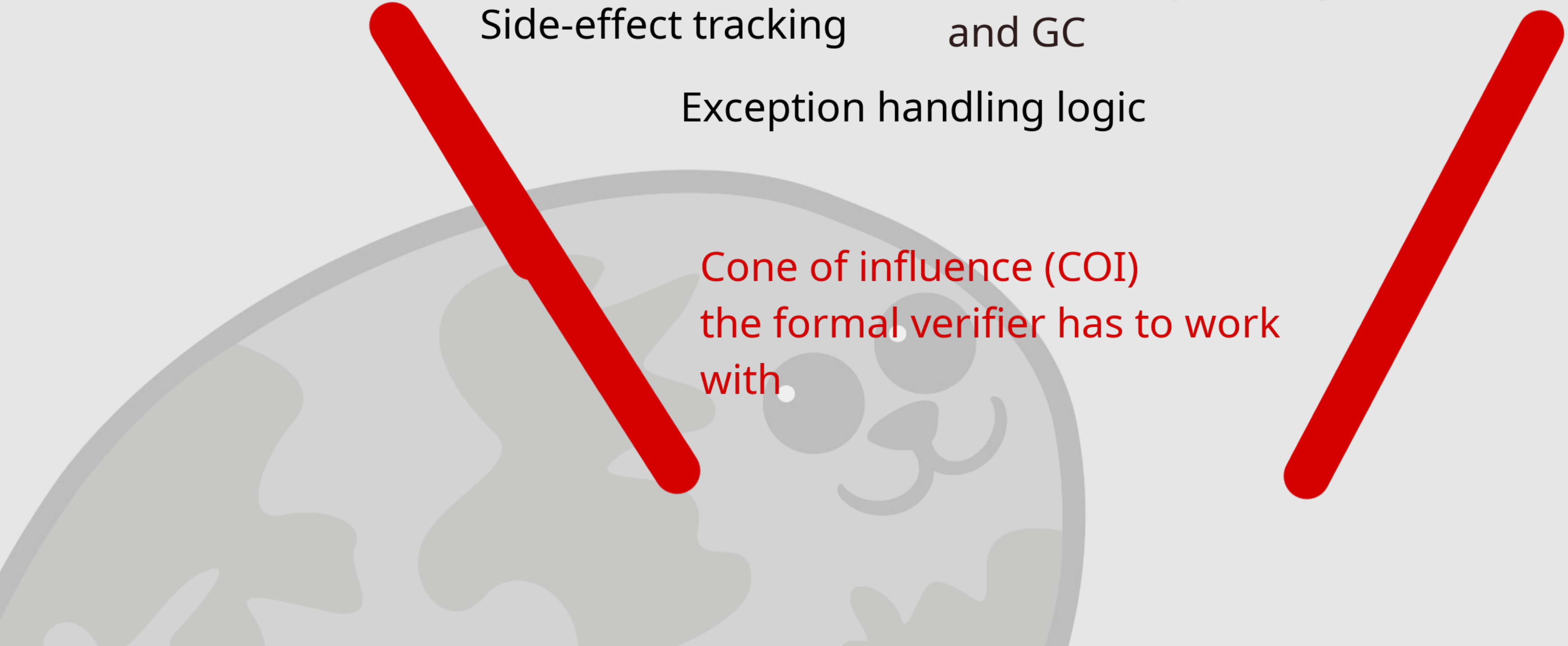
Variable moving semantics

Side-effect tracking

Corroutines and threading

Memory management and GC

Exception handling logic



Cone of influence (COI) the formal verifier has to work with

Why don’t we do it all the time?

Simplified Memory

Variable semantics

Simplified Side-effects

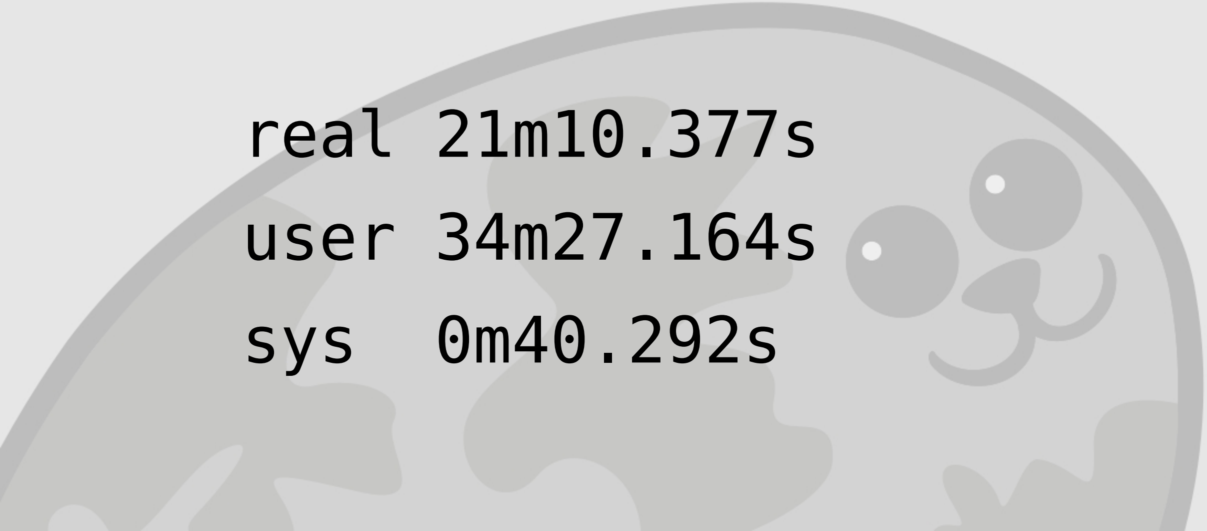
Cone of influence (COI)
the formal verifier has to work with

Why don’t we do it all the time?

You need programming language subsets!

Why don’t we do it all the time?

- Extremely expensive to do in terms of labour and compute as the formal core (the part is formally checked) grows.



```
real 21m10.377s
user 34m27.164s
sys  0m40.292s
```

Enter SPARK

- Subset of Ada with a long list of successes on aerospace, transportation, MIC...
- GNATProve as biggest public, fully FOSS checker.

SPARK’s requirements

- Much stricter scope for side effects.
- Much more restrictive access types (pointers).
- No backward GOTOs.
- No exception handling.
- No controlled types (handicaps a bit the type system).

SPARK's requirements

- Much more restrictive access types (pointers) and a primitive **borrow checker** means Ada becomes more like Rust

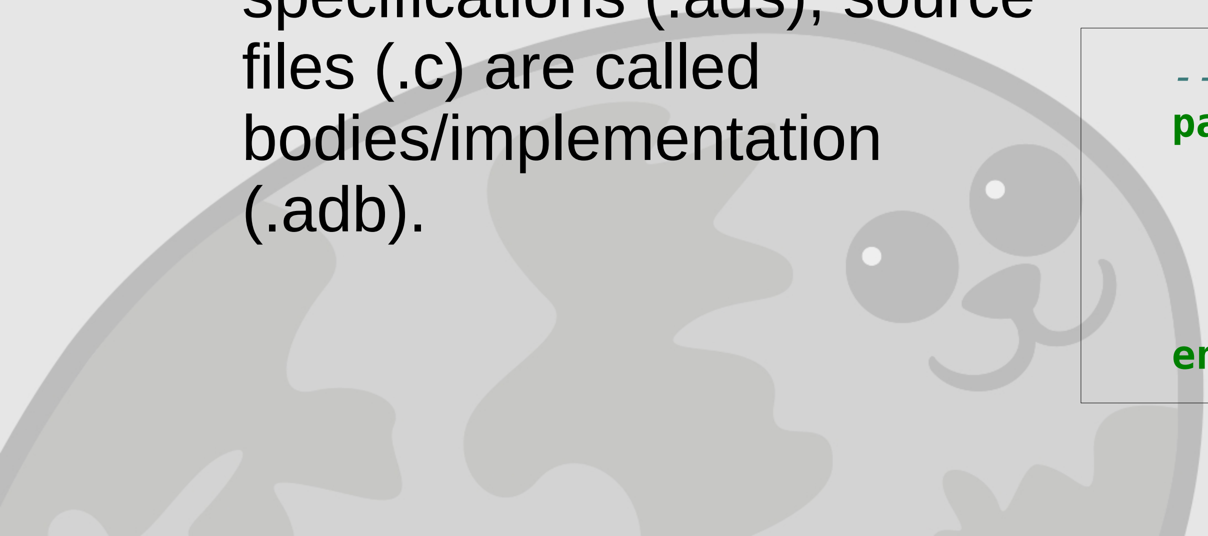
```
type Gen_Int_Acc is access all Integer;

V  : aliased Integer := 15;

-- This is a Move
X3 : Gen_Int_Acc := V'Access;

-- This is a Move
X4 : Gen_Int_Acc := X3;

--  This is an Allocation. GNATprove will flag
--  this as a leak because implicit deallocation
--  is not possible
X3 : Gen_Int_Acc := new Integer'(15);
```

Quick tangent: Ada’s source hierarchy

- Ada uses headers, like C/C++.
- Headers (.h) are called specifications (.ads), source files (.c) are called bodies/implementation (.adb).

```
--  lib-messages.ads.
package Lib.Messages is
    procedure Print;
end Lib.Messages;
```

```
--  lib-messages.adb.
package body Lib.Messages is
    procedure Print is
    begin
        Put_Line ("Hello!");
    end Print;
end Lib.Messages;
```



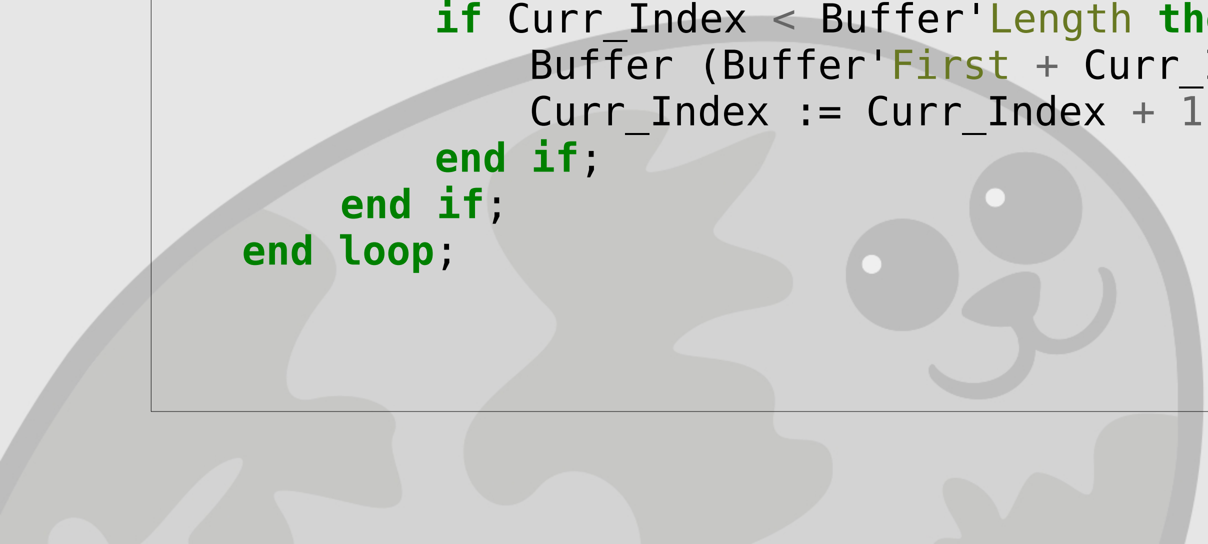
SPARK helps but it doesn’t do everything

```
--  Signature in package specification.
--  Set the user id associated with a process.
procedure Set_UID (Proc : PID; UID : Unsigned_32)
with Global => (In_Out => (Proc_Lock, Proc_Registry)),
     Pre    => Is_Valid (Proc) and UID >= 1000,
     Post   => Get_UID (Proc) = UID;
```

```
--  Implementation in package body.
procedure Set_UID (Proc : PID; UID : Unsigned_32) is
begin
   Registry (Proc).User := UID;
end Set_UID;
```

SPARK helps but it doesn’t do everything

```
for I in Devices_Data'Range loop
    pragma Loop_Invariant (Total <= Devices_Data'Length);

    if Devices_Data (I).Is_Present then
        Total := Total + 1;
        if Curr_Index < Buffer'Length then
            Buffer (Buffer'First + Curr_Index) := I;
            Curr_Index := Curr_Index + 1;
        end if;
    end if;
end loop;
```



SPARK helps but it doesn’t do everything

```
package Arch.Clocks with
    Abstract_State => (Monotonic_Clock_State, RT_Clock_State)
is
    procedure Initialize_Sources
        with Global => (Output => (Monotonic_Clock_State, RT_Clock_State));
end Arch.Clocks;
```

```
package body Arch.Clocks with
    Refined_State =>
        (RT_Clock_State => (Is_Initialized, RT_Timestamp_Seconds,
             RT_Timestamp_Nanoseconds, RT_Stored_Seconds,
             RT_Stored_Nanoseconds),
         Monotonic_Clock_State => (TSC_Tick_Resolution))
is
```

SPARK helps but it doesn’t do everything

```
procedure Read
    (Key         : FS_Handle;
     Ino         : File_Inode_Number;
     Offset      : Unsigned_64;
     Data        : out Operation_Data;
     Ret_Count   : out Natural;
     Is_Blocking : Boolean;
     Success     : out FS_Status)
is
     pragma Annotate
      (GNATprove, False_Positive, "precondition might fail",
      Reason => "No it does not");
...
```

SPARK is still pretty neat

SeL4, biggest formally verified operating system kernel

L4v: 61352 lines of code split among a lot of different languages, the main ones being C, Haskell, Ocaml...

Ironclad

Specifications and checking baked in the code.

Want to check? Run
`make check`

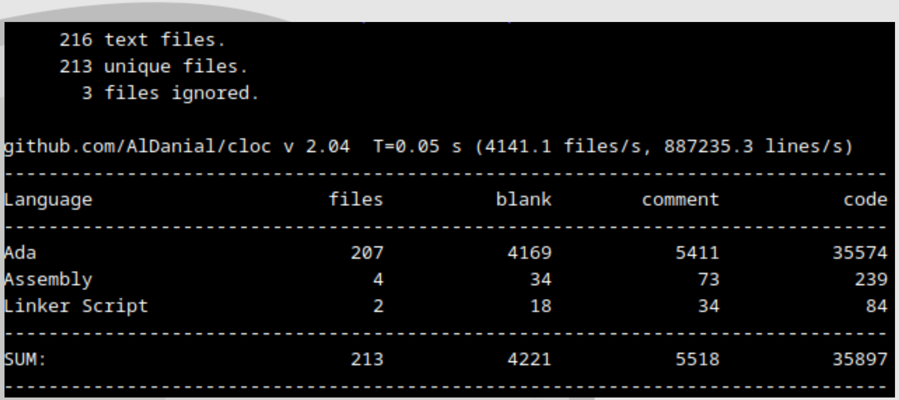
:)

The challenge of formal verification

“The seL4 team reports 20 person years for 10 000 source lines of C code”.

Don’t Sweat the Small Stuff: formal verification of C code without the pain

- NICTA and UNSW, Sydney, Australia



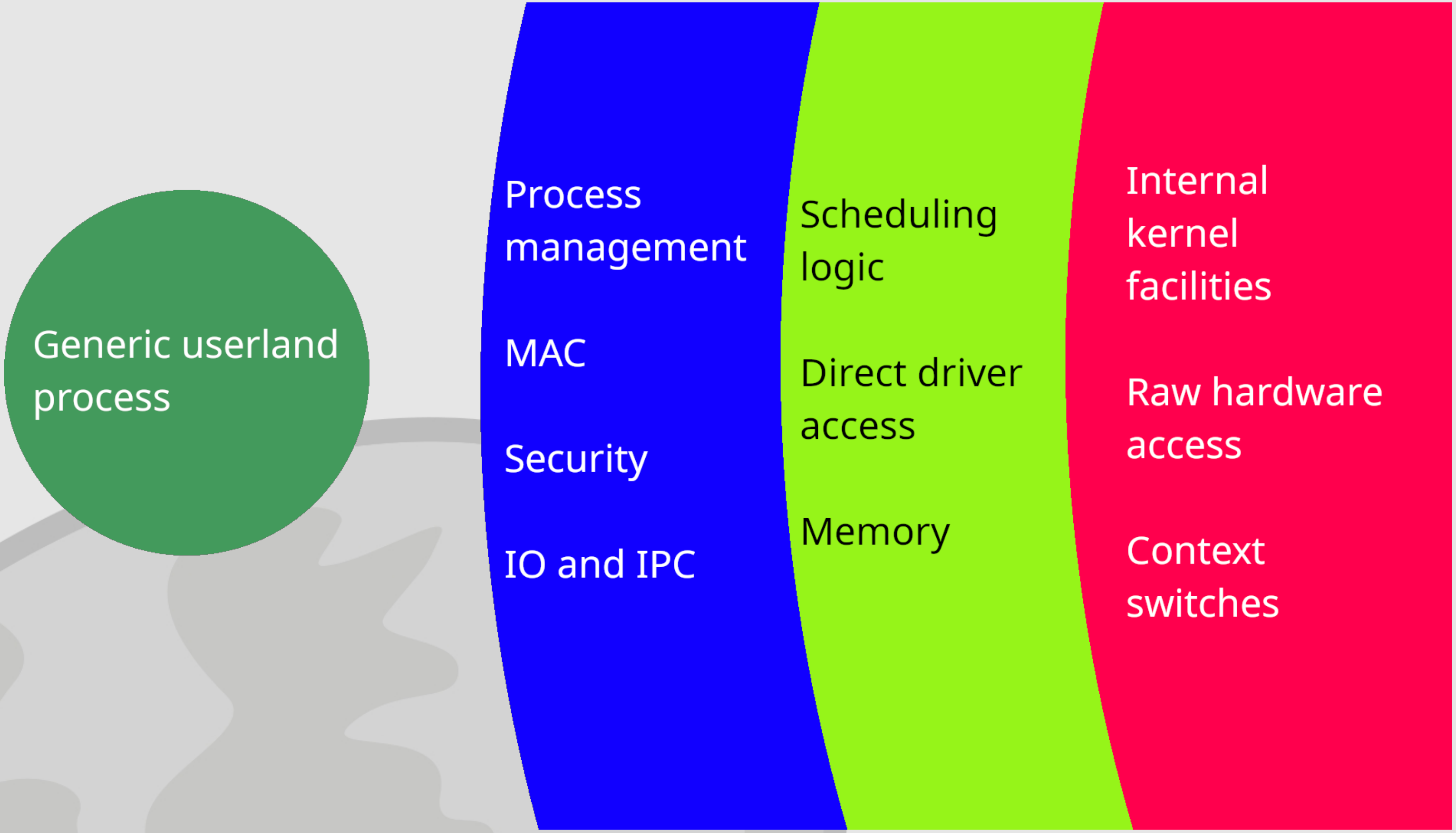
```
     216 text files.
     213 unique files.
       3 files ignored.

github.com/AlDanial/cloc v 2.04  T=0.05 s (4141.1 files/s, 887235.3 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Ada                            207           4169           5411          35574
Assembly                         4             34             73            239
Linker Script                    2             18             34             84
-------------------------------------------------------------------------------
SUM:                           213           4221           5518          35897
-------------------------------------------------------------------------------
```

The challenge of formal verification

- Scheduling code
- Inter-process and inter-thread communication.

- Scheduling code
- Inter-process and inter-thread communication.
- Cryptographic interfaces.
- POSIX interfaces.
- More complex IPC interfaces.
- Kernel level device drivers.
- Filesystem and VFS.
- Networking.

So we have to pick our battles



Generic userland process

Process management

MAC

Security

IO and IPC

Scheduling logic

Direct driver access

Memory

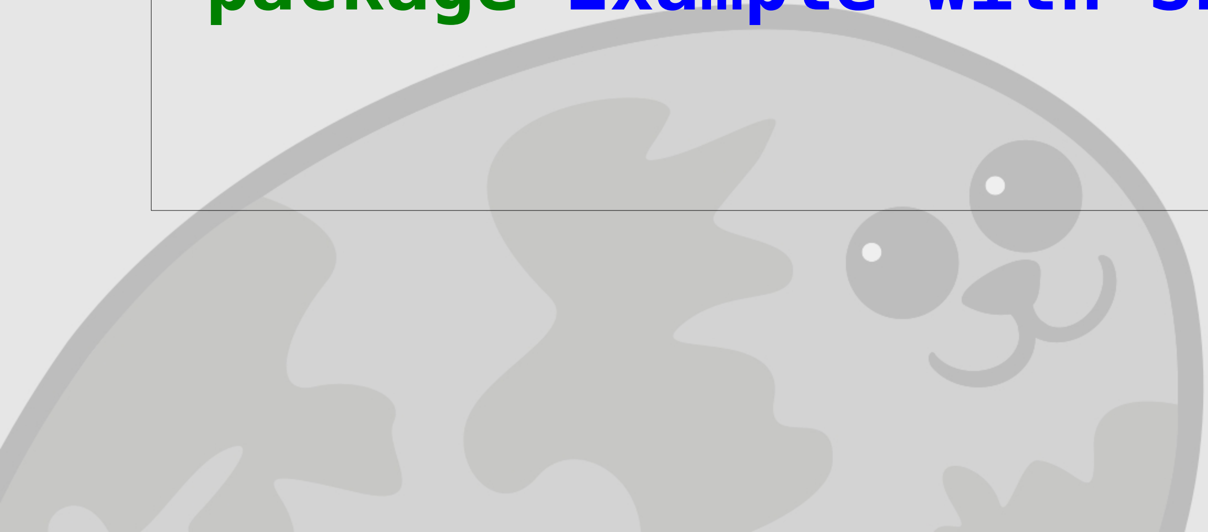
Internal kernel facilities

Raw hardware access

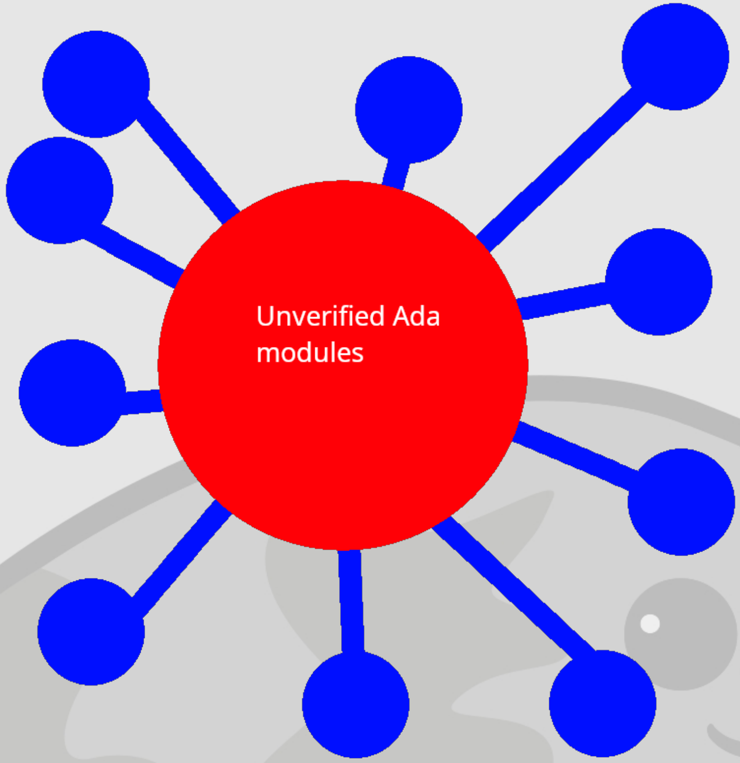
Context switches

Thankfully sometimes you can say no...

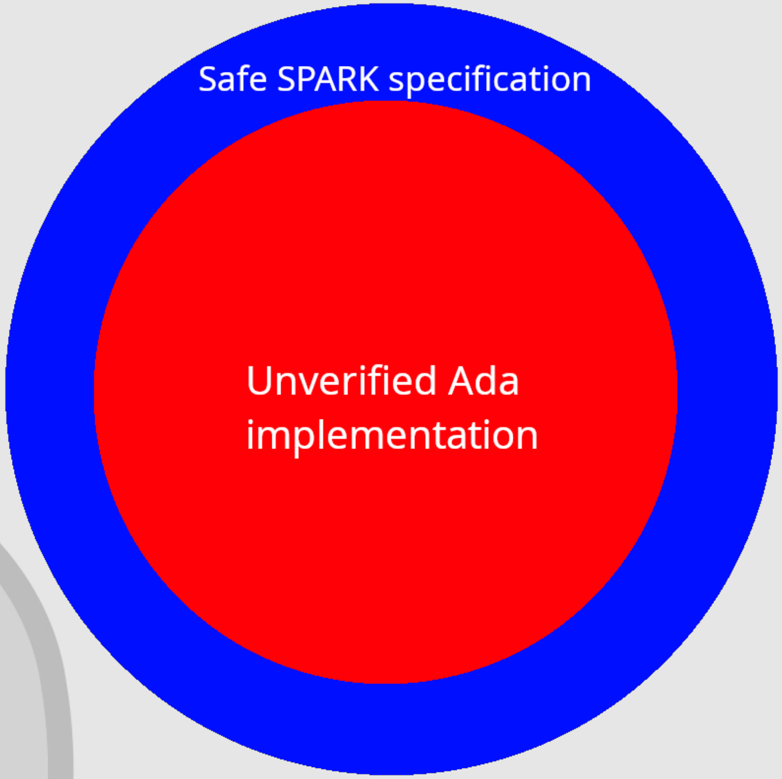
```
package Example with SPARK_Mode => Off
```



Thankfully Ada helps



Unverified Ada modules



Safe SPARK specification

Unverified Ada implementation

SPARK levels

- Bronze level
- Correct initialization and correct data flow.

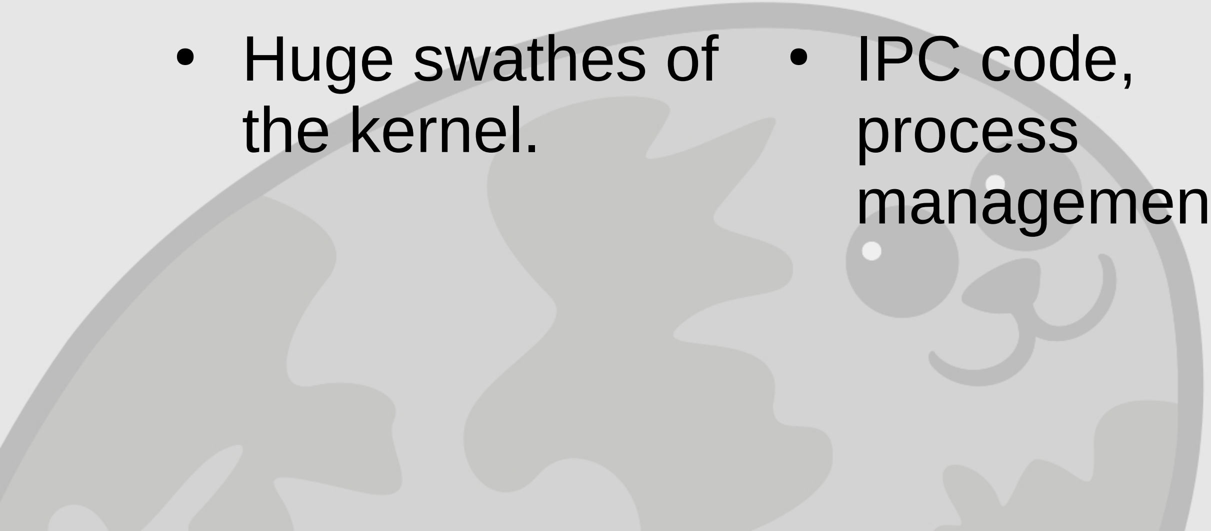
- Huge swathes of the kernel.

- Silver level
- Abscence of runtime errors (AoRTE).

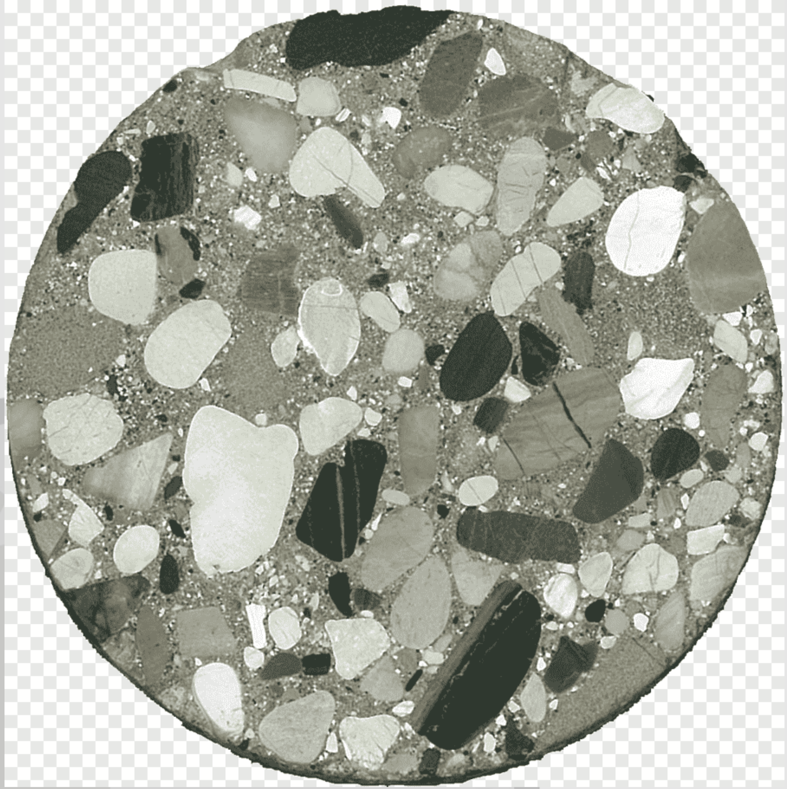
- IPC code, process management.

- Gold level
- Proof of integrity and correctness according to specs.

- Cryptographic and Mandatory Access Control (MAC) code.



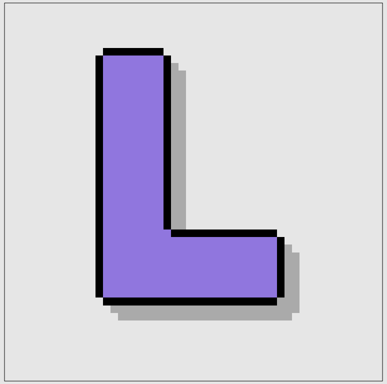
SPARK levels



Follow the progress, check the source code, or download distributions at

<https://ironclad-os.org>

Thanks to



Thanks to

- Mintsuki <https://github.com/mintsuki>
- Lucretia <https://github.com/lucretia>
- Ineiev <https://savannah.gnu.org/users/ineiev>
- Irvise <https://github.com/Irvise>
- The Managarm Project <https://github.com/managarm>