Ironclad

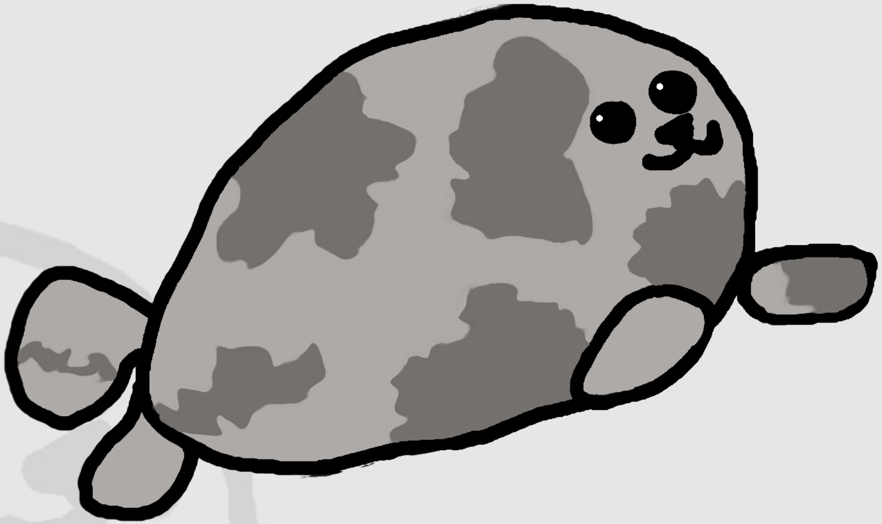
A formally verified OS kernel written in SPARK and Ada.

<streaksu@protonmail.com>

Ada Monthly Meetup - Sat, Nov 4, 2023

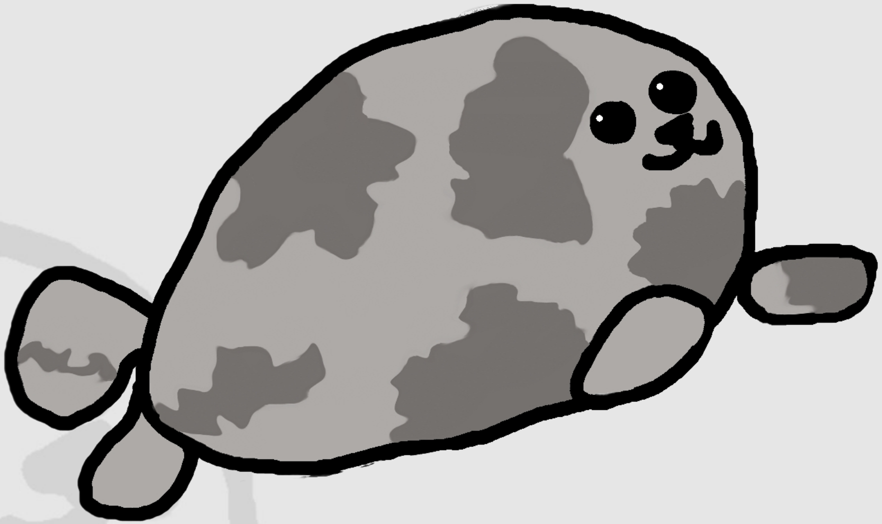
So what is Ironclad?

- POSIX-compatible formally-verified (to an extent) kernel.
- Hard real-time facilities and flexible scheduling.
- Highly portable.
- Free as in freedom.

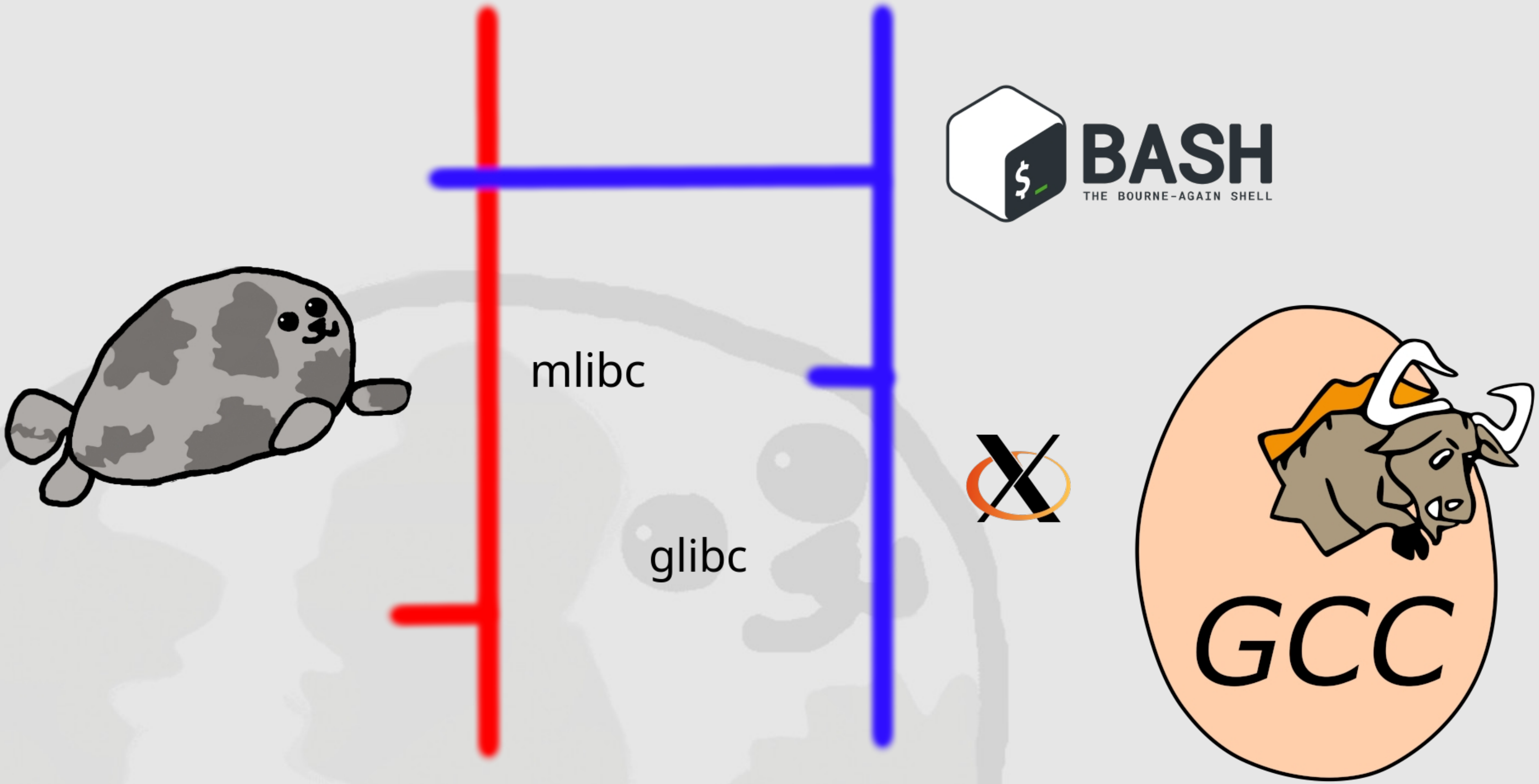


What are the goals of Ironclad?

- A highly secure architecture.
- Hard real-time suitability without compromising general purpose computing.
- Doing so keeping in mind POSIX compatibility.

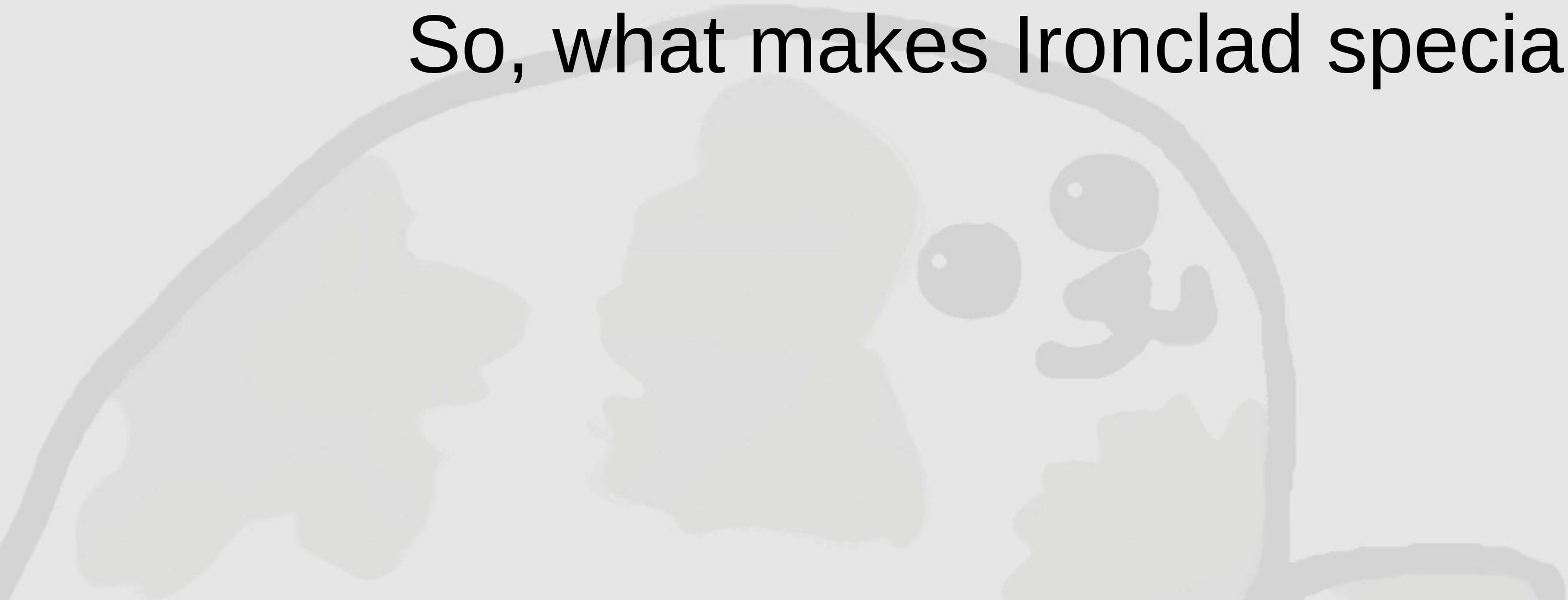


Operating system architecture

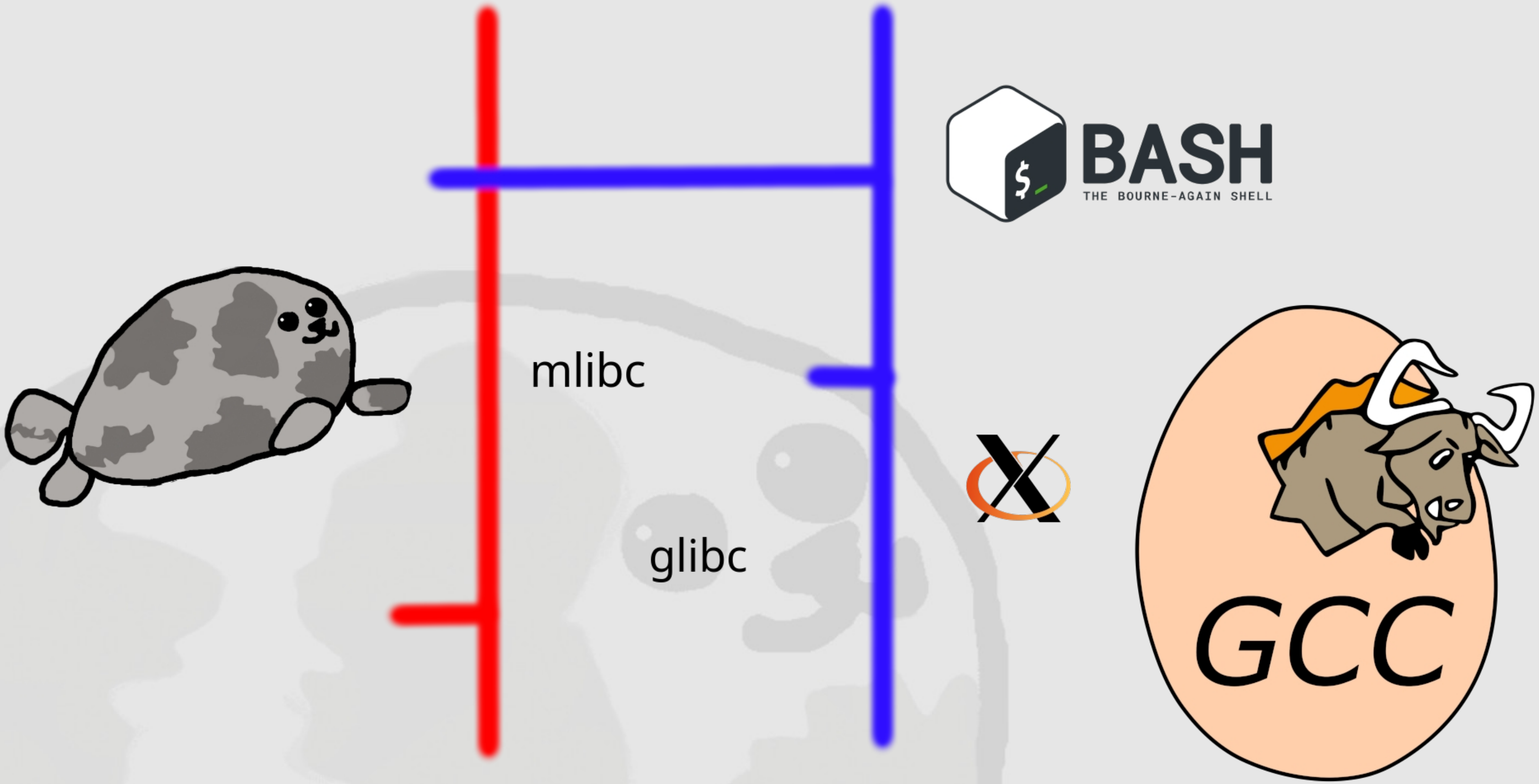


Operating system architecture

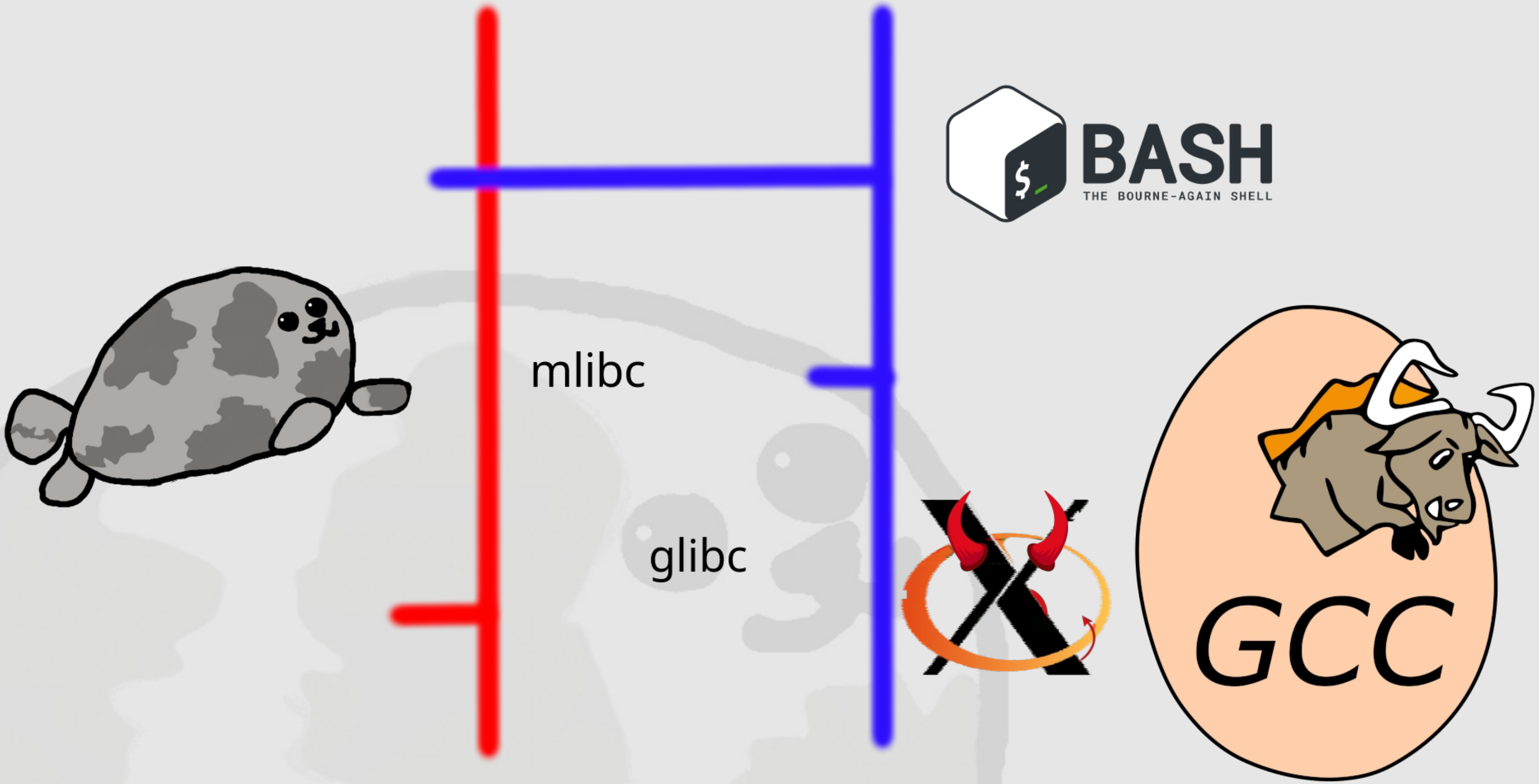
So, what makes Ironclad special?



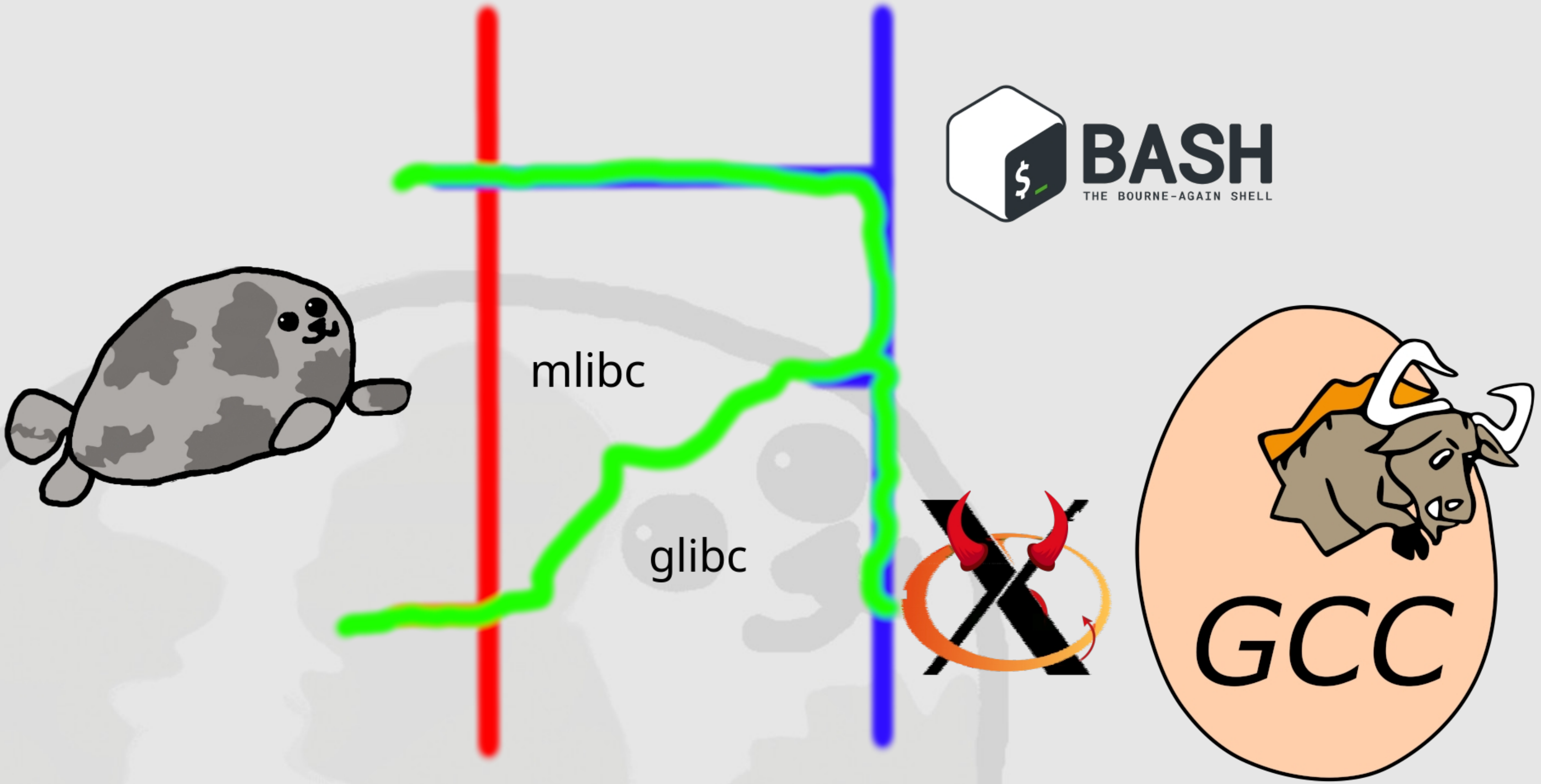
Mandatory Access Control (MAC)



Mandatory Access Control (MAC)



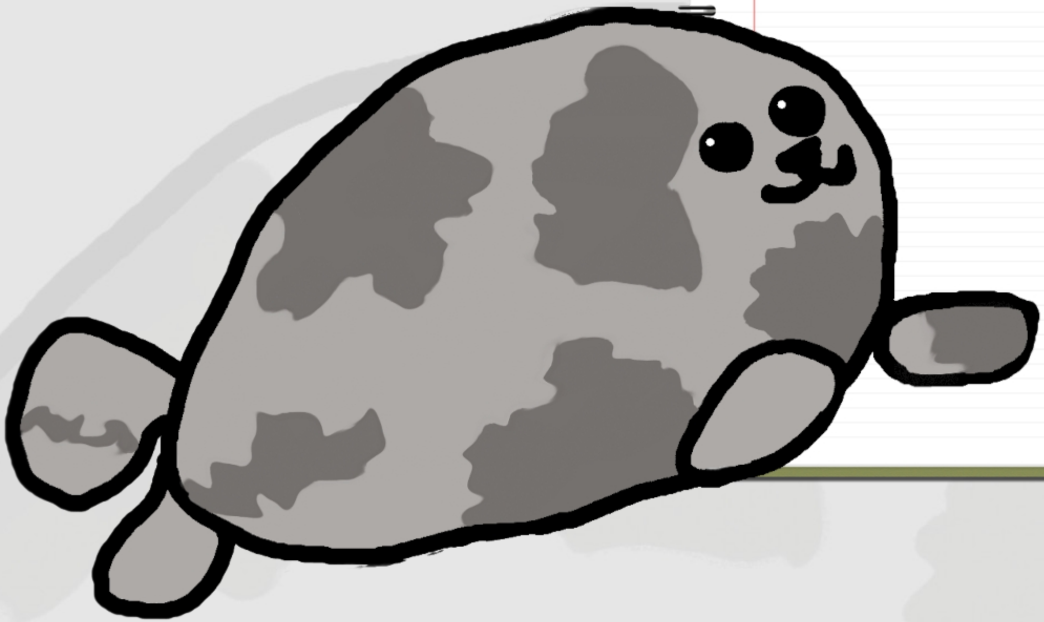
Mandatory Access Control (MAC)



Mandatory Access Control (MAC)

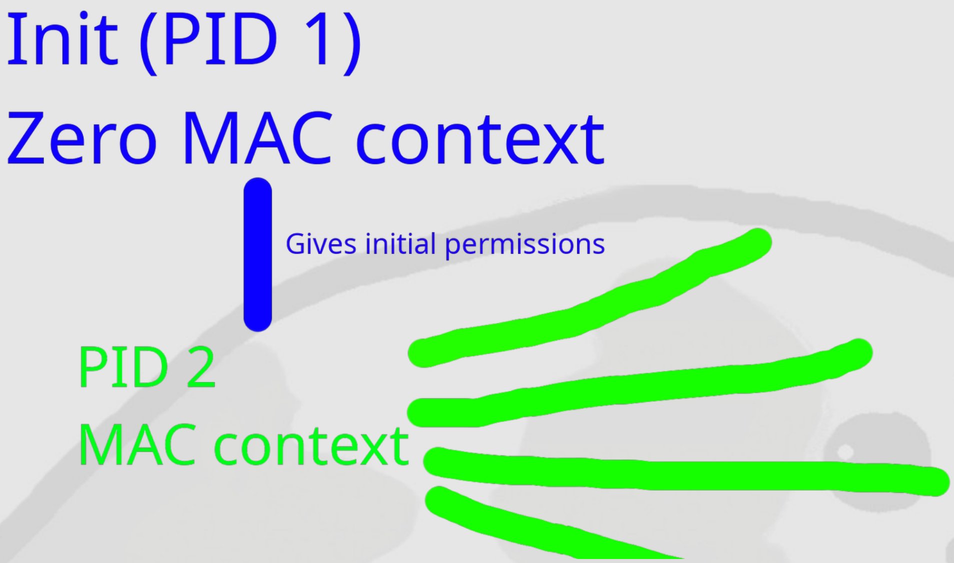
Mandatory Access Control (MAC)

Things X can do:
- Allocate memory
- Do inter process communication
- Access entropy resources
- Open files read only.



Mandatory Access Control (MAC)

Mandatory Access Control (MAC)




```
type Capabilities is record
   Can_Change_Scheduling : Boolean;
   Can_Spawn_Others      : Boolean;
   ...
end record;

type Context is record
   Action  : Enforcement;
   Caps    : Capabilities;
   Limits  : Limit_Arr;
   Filters : Filter_Arr (1 .. 30);
end record;
```

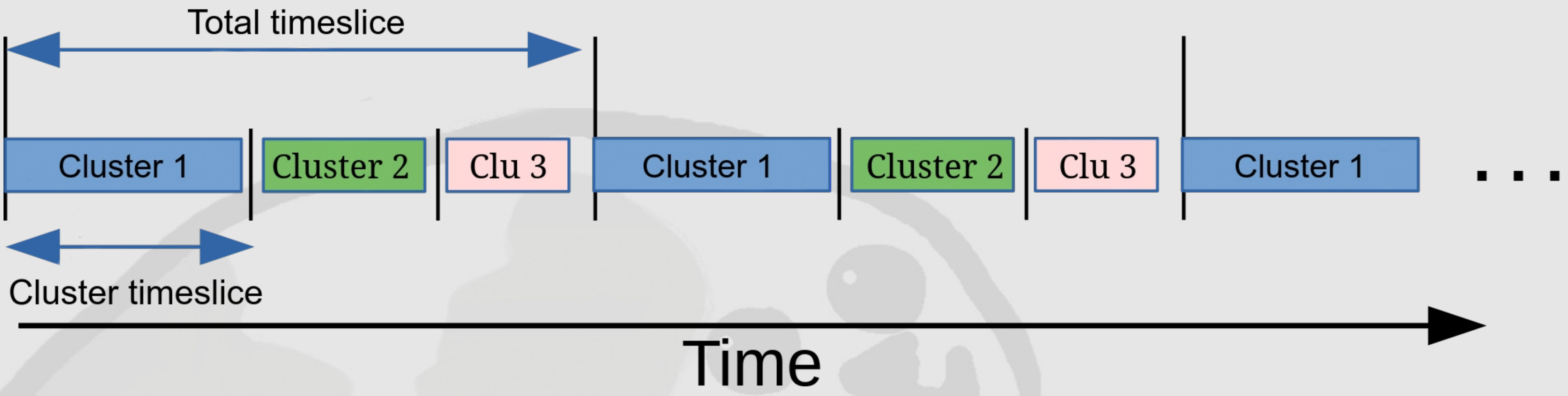
Scheduling

- Processes: Owns a memory map, threads, open files.

- Thread-cluster: Groups threads regardless of process and coordinates them.

- Threads: Basic unit of processor execution, has a set of registers and stack.

Scheduling


Total timeslice
Cluster 1
Cluster 2
Clu 3
Cluster 1
Cluster 2
Clu 3
Cluster 1
. . .
Cluster timeslice
Time

Formal verification

- 3 tiers.
- Architectural code that is difficult to verify or can be reasonably verified to a lesser standard.
- Easily verifiable architecture-independent code.

Formal verification

```
--  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;
```

```
procedure Set_UID (Proc : PID; UID : Unsigned_32) is
begin
   Registry (Proc).User := UID;
end Set_UID;
```

How do these benefits extend to userland?

```c
// Set the real and effective user Ids
// to 1000.
int err = setuids(1000, 1000);
```

```asm
mov $59, %rax
mov $1000, %rdi
mov $1000, %rsi
syscall # <- Straight to Ironclad!
```

- We never have to leave formally verified code!

Limitations of POSIX and userland verification

```
ssize_t read(int fd, void *buffer, size_t count);
```

Global state is not properly encapsulated!

Limitations of POSIX and userland verification

```
ssize_t read(int fd, void *buffer, size_t count);
                    // FD is a socket.
                    // FD is non blocking, so no waiting.
                    // And this goes all the way up the chain...
```

Global state is not properly encapsulated!

What’s next for Ironclad

- Finishing the last bells and wistles to get Xorg and a proper desktop environment to run.
- Do a port to riscv-based boards, like the visionfive series.
- Expand the existing networking to more network cards.

Follow the progress, check the source code, or download distributions at https://ironclad.cx

Thanks to

AdaCore

Thanks to

- Mintsuki <https://github.com/mintsuki>
- Lucretia <https://github.com/lucretia>
- Ineiev <https://savannah.gnu.org/users/ineiev>