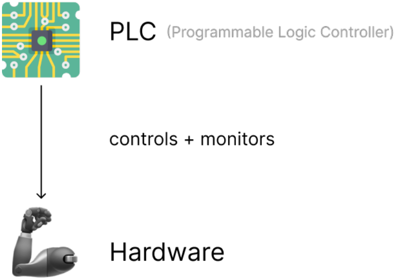
Have you ever heard of ML?

Codrin Iftode
@ CERN OpenLab

August 12, 2024

Device monitoring - Easy



PLC (Programmable Logic Controller)

controls + monitors

Hardware

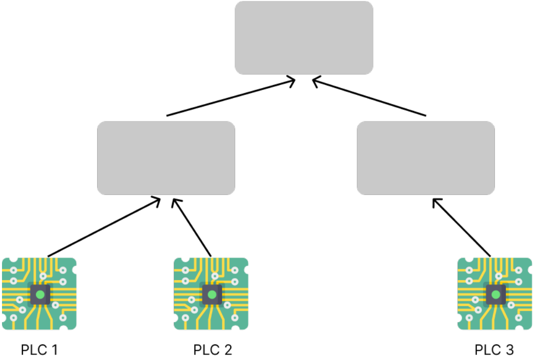
Icon attribution: https://www.flaticon.com/free-icons/chip

Device monitoring - Hard

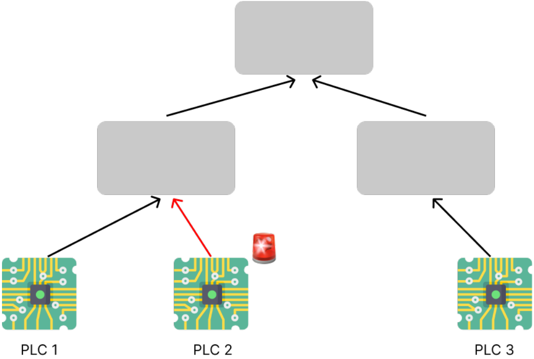
At CERN, there are **more than 2000** PLCs.

Error management, in **real-time??**

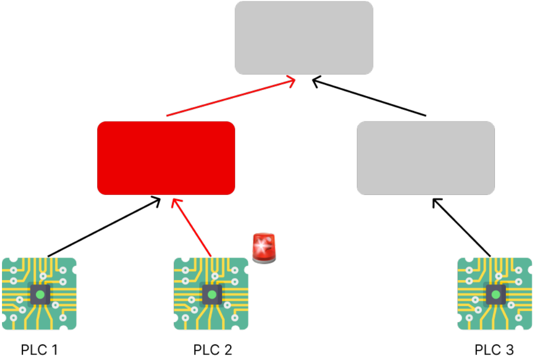
A solution: tree structure



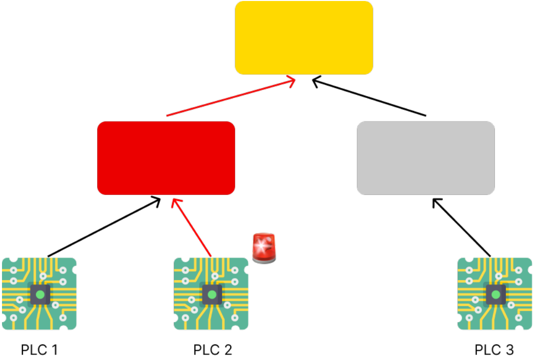
A solution: tree structure



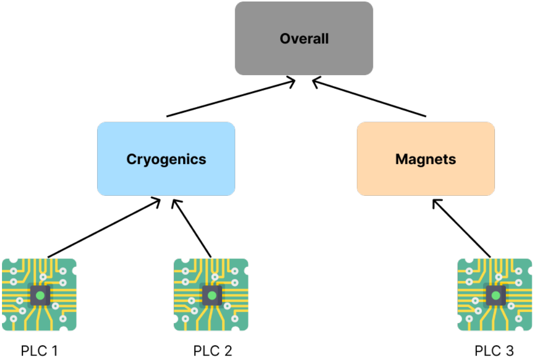
A solution: tree structure



A solution: tree structure



A solution: tree structure



Objectives

- Lightweight UI tool

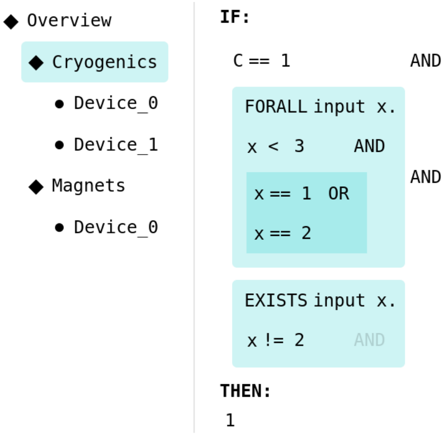
Objectives

- Lightweight UI tool
- Distributed system to run the tree

Objectives

- Lightweight UI tool
- Distributed system to run the tree
- Generic code for wider use cases

Designing a node (UI)



◆ Overview

◆ Cryogenics

● Device_0

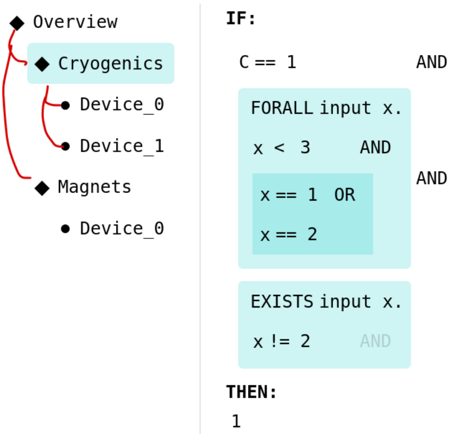
● Device_1

◆ Magnets

● Device_0

```
IF:
 C == 1                AND
 FORALL input x.
  x < 3       AND
  x == 1  OR           AND
  x == 2
 EXISTS input x.
  x != 2      AND
THEN:
 1
```

Designing a node (UI)



- ◆ Overview
  - ◆ Cryogenics
    - ● Device_0
    - ● Device_1
  - ◆ Magnets
    - ● Device_0

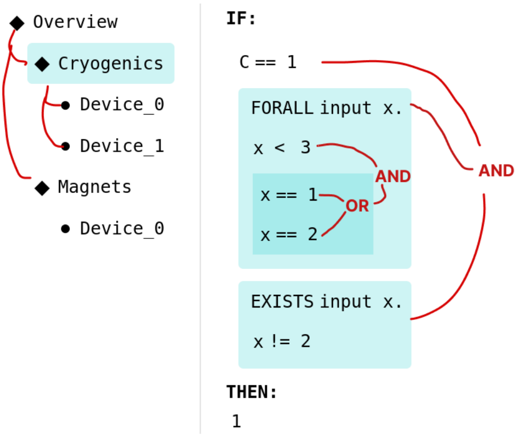
IF:

```
C == 1                  AND
  FORALL input x.
   x < 3       AND
    x == 1  OR          AND
    x == 2
  EXISTS input x.
   x != 2      AND
```

THEN:

1

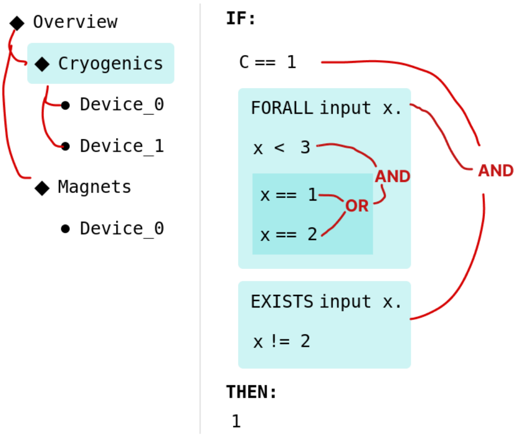
Designing a node (UI)



- ◆ Overview
  - ◆ Cryogenics
    - ● Device_0
    - ● Device_1
  - ◆ Magnets
    - ● Device_0

```
IF:

  C == 1

  FORALL input x.
    x < 3
    x == 1
    x == 2

  EXISTS input x.
    x != 2

THEN:
  1
```

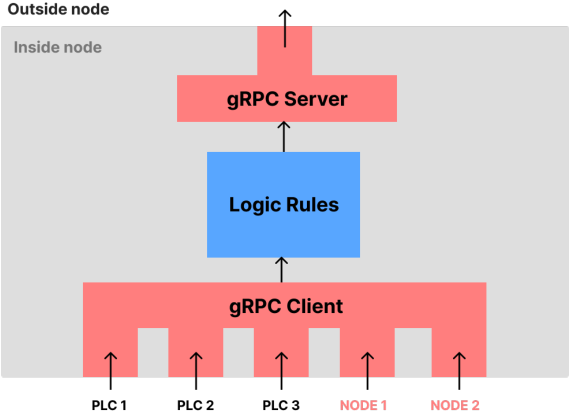
Designing a node (UI)



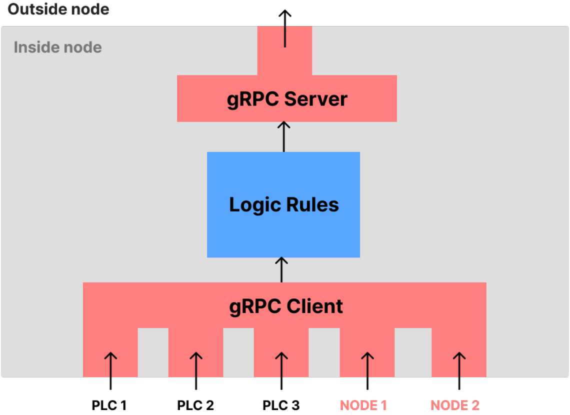
```
IF:

 C == 1

  FORALL input x.

  x < 3

   x == 1
   x == 2

  EXISTS input x.

  x != 2

THEN:
 1
```

We developed a tree state management library for React.

Running a node

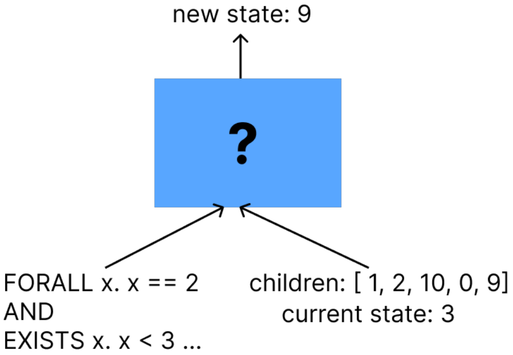


Running a node



Nodes talk over gRPC $\Longrightarrow$ distributed system

But where is the ML?



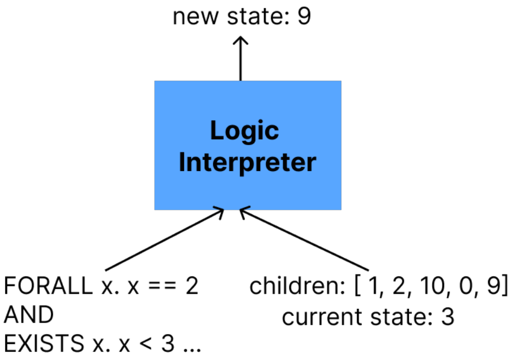
new state: 9

?

FORALL x. x == 2
AND
EXISTS x. x < 3 ...

children: [1, 2, 10, 0, 9]
current state: 3

But where is the ML?



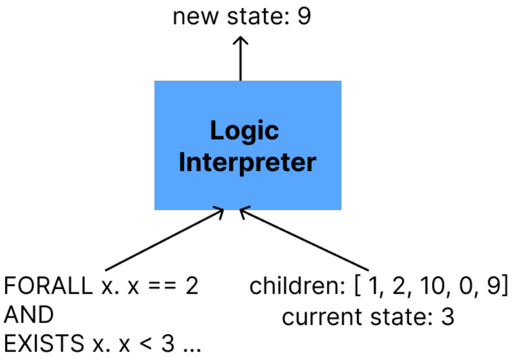
new state: 9

Logic Interpreter

FORALL x. x == 2
AND
EXISTS x. x < 3 ...

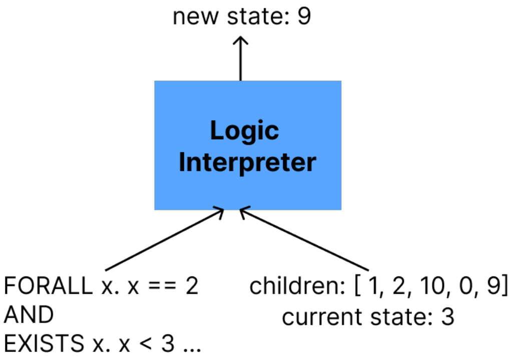
children: [1, 2, 10, 0, 9]
current state: 3

But where is the ML?



The language of the interpreter manipulates another language (logic).

But where is the ML?



new state: 9

Logic Interpreter

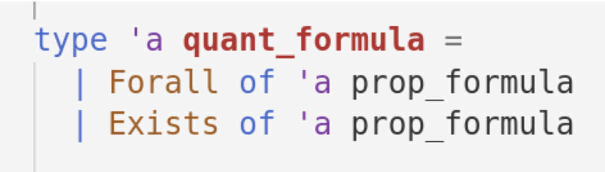
FORALL x. x == 2
AND
EXISTS x. x < 3 ...

children: [1, 2, 10, 0, 9]
current state: 3

The language of the interpreter manipulates another language (logic).

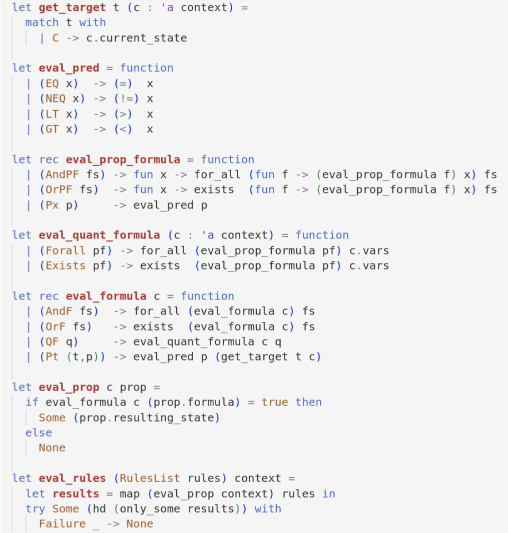
It is a meta-language! (ML)

Let's write an interpreter!



```
type 'a quant_formula =
  | Forall of 'a prop_formula
  | Exists of 'a prop_formula
```

Let's write an interpreter!



```
let get_target t (c : 'a context) =
  match t with
    | C -> c.current_state

let eval_pred = function
  | (EQ x)  -> (=)  x
  | (NEQ x) -> (!=) x
  | (LT x)  -> (>)  x
  | (GT x)  -> (<)  x

let rec eval_prop_formula = function
  | (AndPF fs) -> fun x -> for_all (fun f -> (eval_prop_formula f) x) fs
  | (OrPF fs)  -> fun x -> exists  (fun f -> (eval_prop_formula f) x) fs
  | (Px p)     -> eval_pred p

let eval_quant_formula (c : 'a context) = function
  | (Forall pf) -> for_all (eval_prop_formula pf) c.vars
  | (Exists pf) -> exists  (eval_prop_formula pf) c.vars

let rec eval_formula c = function
  | (AndF fs)  -> for_all (eval_formula c) fs
  | (OrF fs)   -> exists  (eval_formula c) fs
  | (QF q)     -> eval_quant_formula c q
  | (Pt (t,p)) -> eval_pred p (get_target t c)

let eval_prop c prop =
  if eval_formula c (prop.formula) = true then
    Some (prop.resulting_state)
  else
    None

let eval_rules (RulesList rules) context =
  let results = map (eval_prop context) rules in
  try Some (hd (only_some results)) with
    Failure _ -> None
```

Why OCaml?

- Rich type system $\Longrightarrow$ easy to use as meta-language
- Interpreter is only **30 lines of code**
- Used by industry leaders, such as Jane Street, Facebook, Microsoft, Bloomberg, and more.

Now and Future

- Lightweight UI tool
- Distributed system to run the tree (95%)
- Generic code
- Connect frontend to backend
- Expand the interpreter

Thank you!

CERN openlab

Thank you!

(learn OCaml, it's fun)