2020-05-06

Software implementation for RF breakdown protection and recovery

Bartosz Bielawski (BE-RF)
Rolf Wegner (BE-RF)

188th Machine Protection Panel Meeting (Injectors): Special meeting on Linac4 RFQ protection

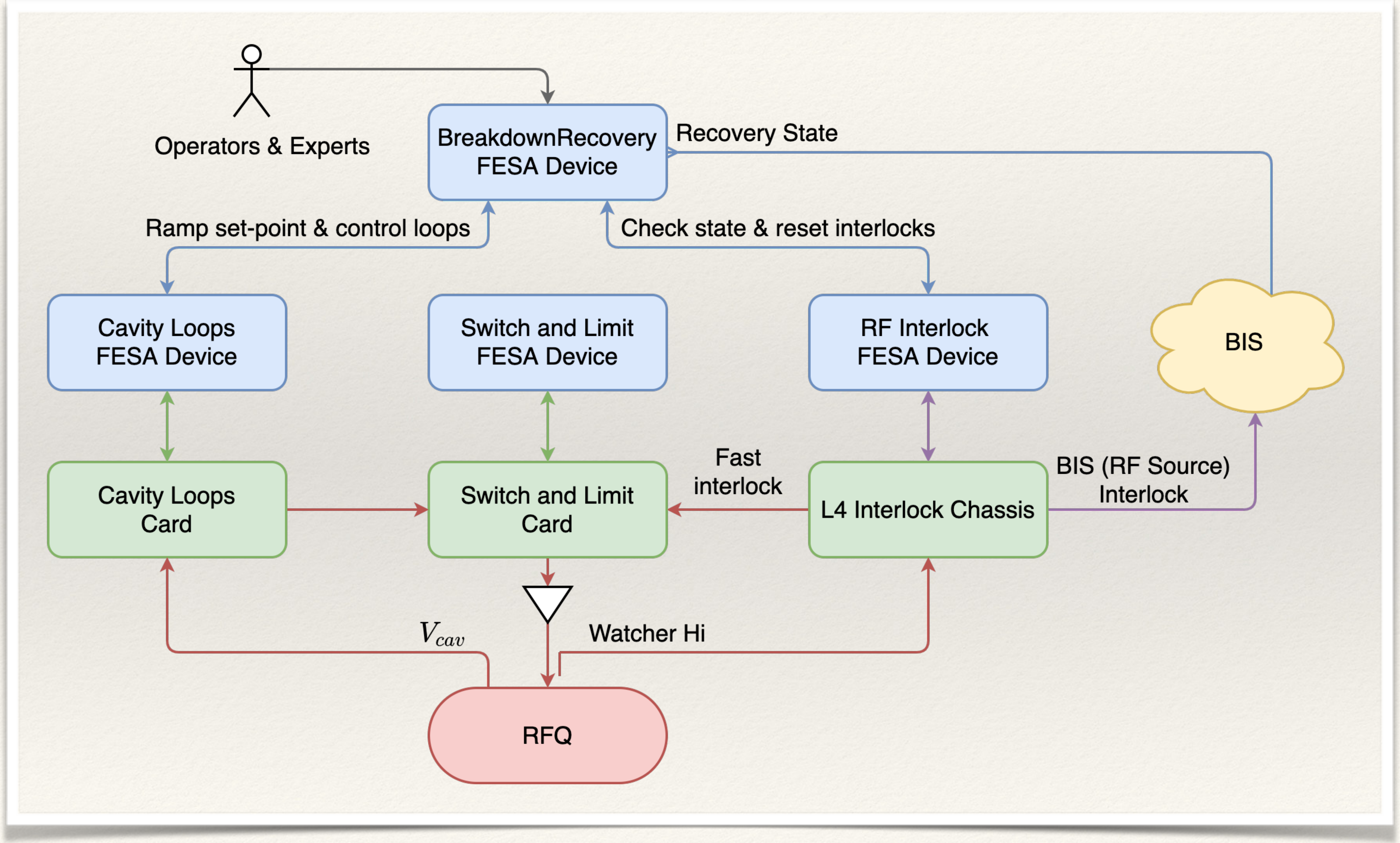
Outline

- Objectives,
- Structure,
- States & Transitions,
- Recovery ramps,
- Interfaces,
- Modifications of the control system,
- Other details

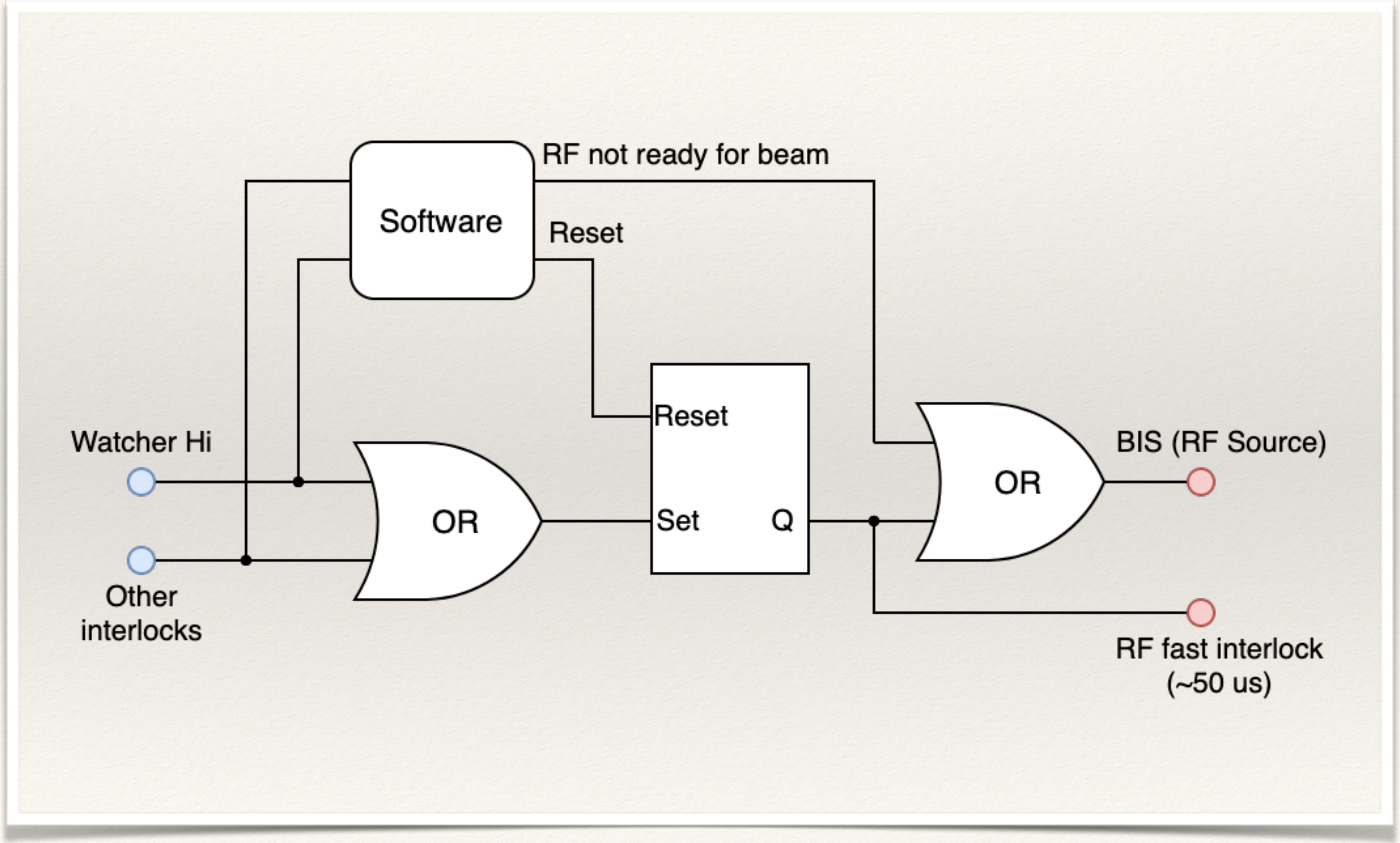
Objectives

- Protect the RFQ (done by HW),
- Decrease downtime due to single breakdowns,
- Automate detection of breakdown clusters,
- Automate recovery after breakdown clusters,
- Provide information about RFQ breakdown rates.

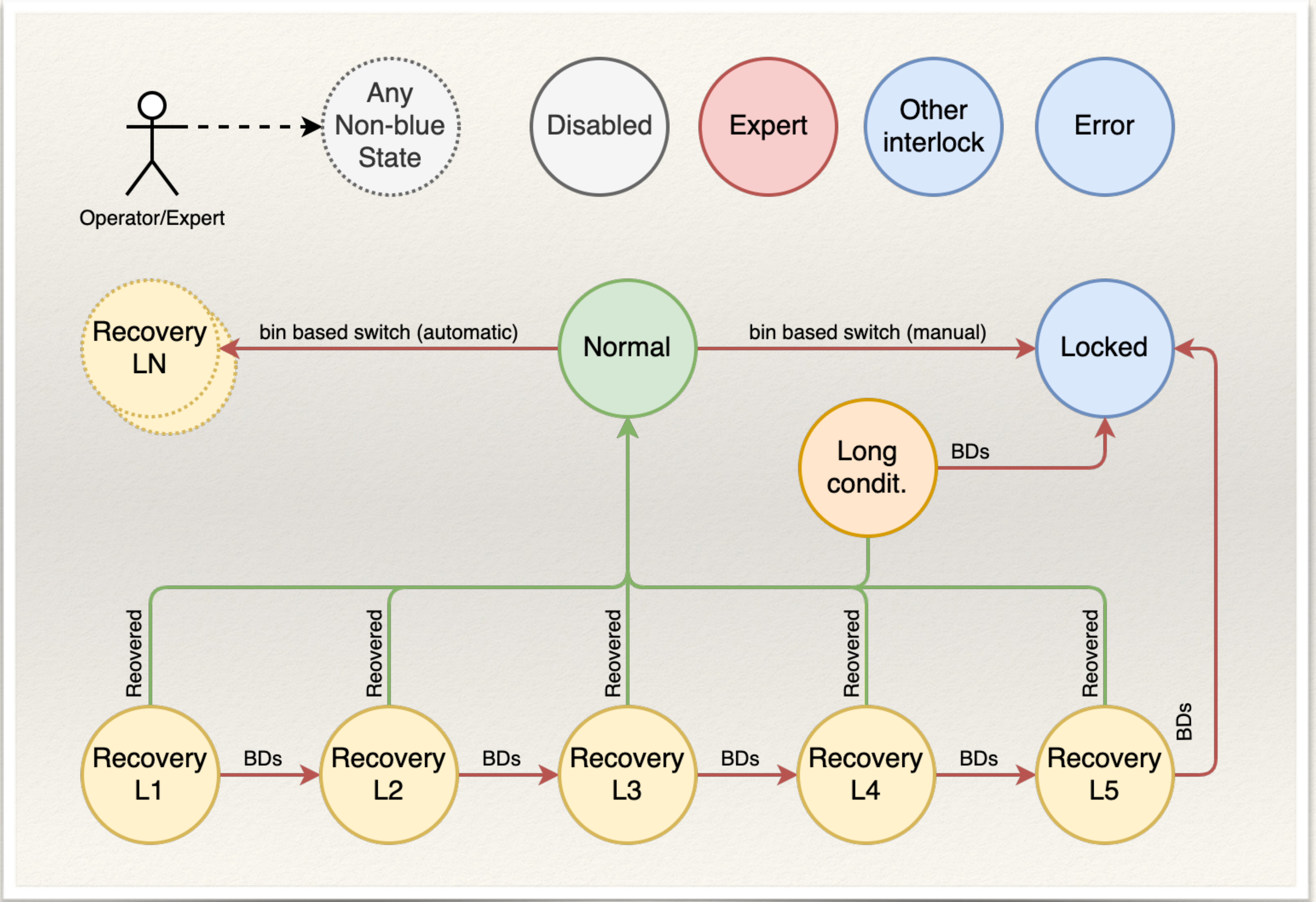
Structure: general



Structure: PLCs & Interlocks

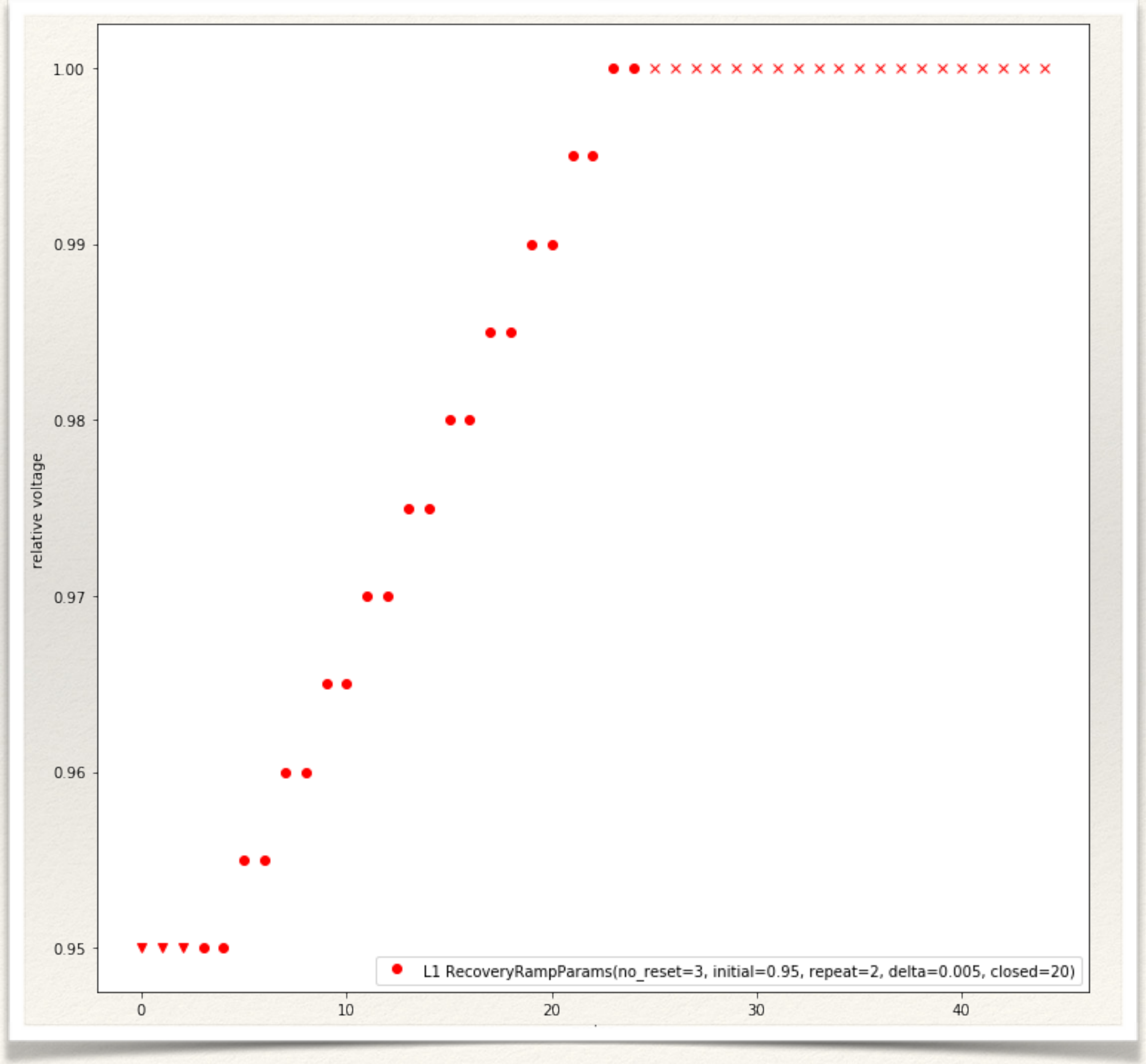


States & Transitions

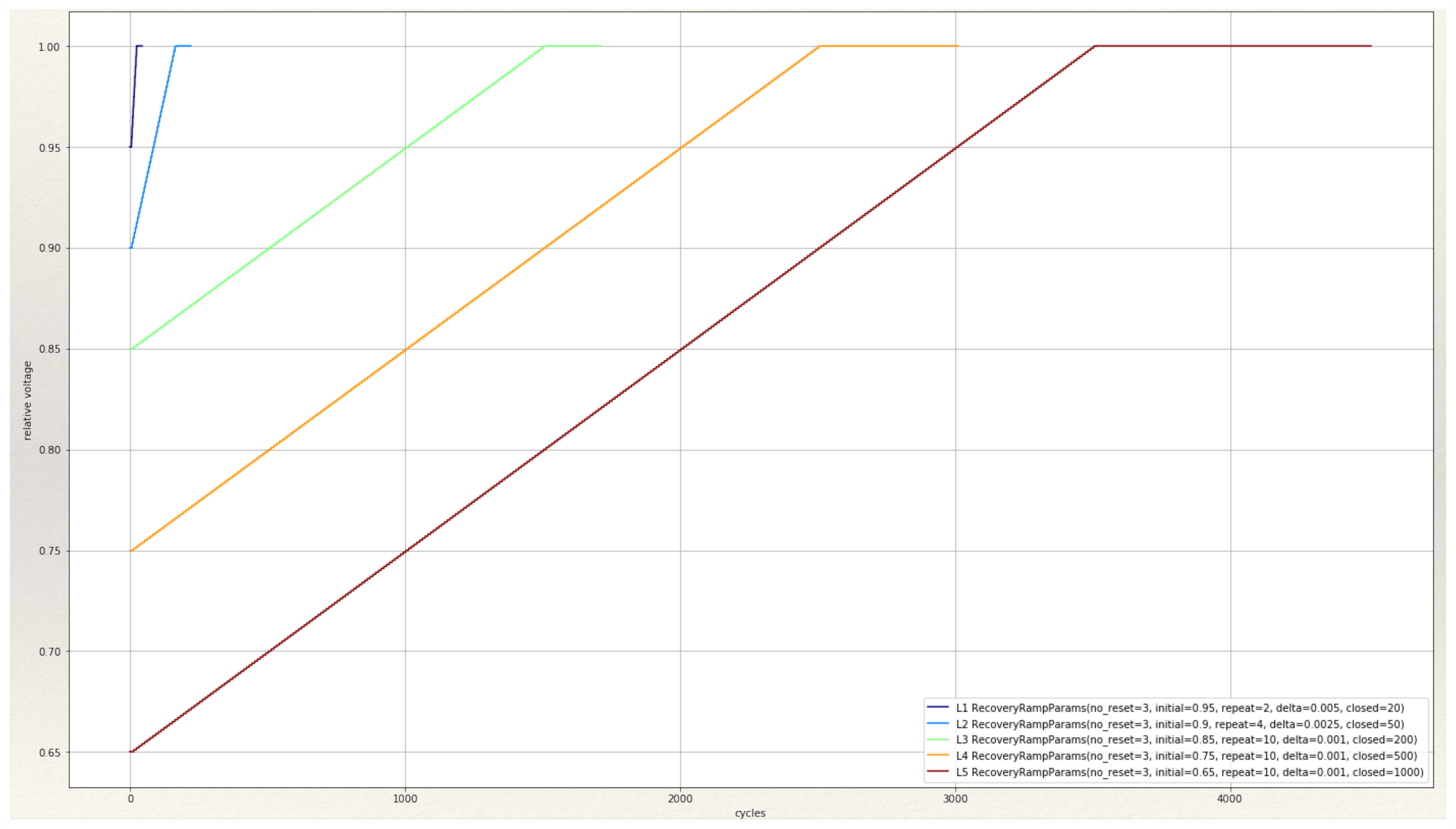


Recovery ramps (L1, parameters)

- Number of recovery levels: 5 + conditioning,
- Parameters:
  - initial delay (▼),
  - initial voltage:
    - relative, absolute
  - repetitions,
  - step delta,
  - repetitions with feedback (×).



Recovery ramps (L1-L5)



Interfaces

- Hardware:
  - Interlock to BIS (Source RF):
    - SW control over “RF not ready”,
    - HW control from all the other interlocks,
- Software (FESA):
  - Command property to request state transition,
  - Status property: current state and suggested state,
  - Diagnostic interfaces for RF and OP.

Modifications of the control system

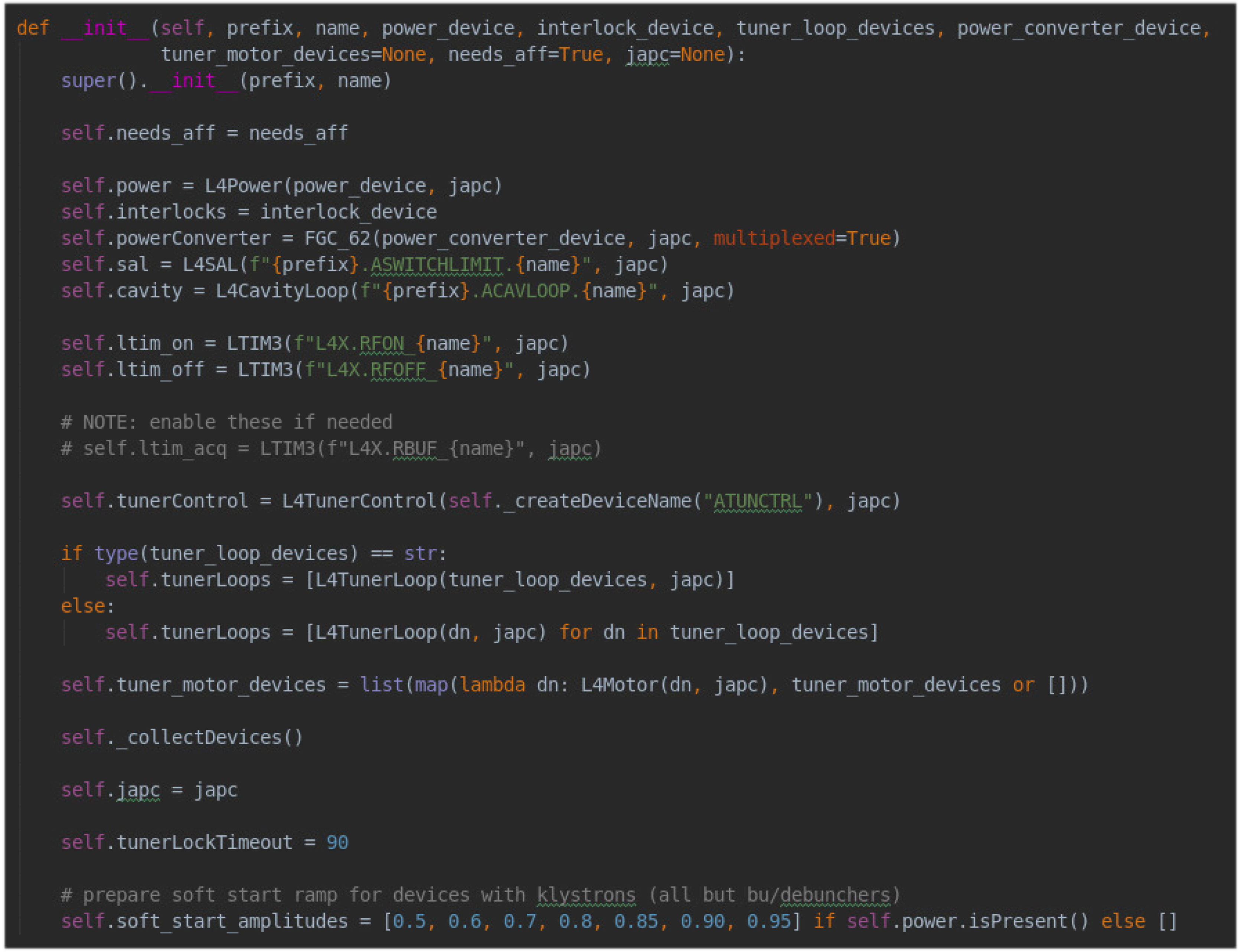
- PLCs:
  - Wiring of the Watcher High (reflected power trigger) interlock,
  - Update of Interlock PLC to include “RF not ready”,
- Software:
  - Implementation of regular readouts, (async. 2 s → sync. 1.2 s),
  - Update of CavityLoops (non-muxed SP scaling & loops enable),
  - Update of RF Sequencer to be aware of the BD recovery class,
- Optional:
  - Beam stopper control (BE-OP).

Other details

- Parameters of recovery ramps and bins are configurable,
- Mandatory logging - interlocks are reset as they happen,
- Handling of special cases can still be redefined,
- Alarms for breakdowns and other special cases.

Thank you for your attention…

Questions?



```
def __init__(self, prefix, name, power_device, interlock_device, tuner_loop_devices, power_converter_device,
             tuner_motor_devices=None, needs_aff=True, japc=None):
    super().__init__(prefix, name)

    self.needs_aff = needs_aff

    self.power = L4Power(power_device, japc)
    self.interlocks = interlock_device
    self.powerConverter = FGC_62(power_converter_device, japc, multiplexed=True)
    self.sal = L4SAL(f"{prefix}.ASWITCHLIMIT.{name}", japc)
    self.cavity = L4CavityLoop(f"{prefix}.ACAVLOOP.{name}", japc)

    self.ltim_on = LTIM3(f"L4X.RFON_{name}", japc)
    self.ltim_off = LTIM3(f"L4X.RFOFF_{name}", japc)

    # NOTE: enable these if needed
    # self.ltim_acq = LTIM3(f"L4X.RBUF_{name}", japc)

    self.tunerControl = L4TunerControl(self._createDeviceName("ATUNCTRL"), japc)

    if type(tuner_loop_devices) == str:
        self.tunerLoops = [L4TunerLoop(tuner_loop_devices, japc)]
    else:
        self.tunerLoops = [L4TunerLoop(dn, japc) for dn in tuner_loop_devices]

    self.tuner_motor_devices = list(map(lambda dn: L4Motor(dn, japc), tuner_motor_devices or []))

    self._collectDevices()

    self.japc = japc

    self.tunerLockTimeout = 90

    # prepare soft start ramp for devices with klystrons (all but bu/debunchers)
    self.soft_start_amplitudes = [0.5, 0.6, 0.7, 0.8, 0.85, 0.90, 0.95] if self.power.isPresent() else []
```