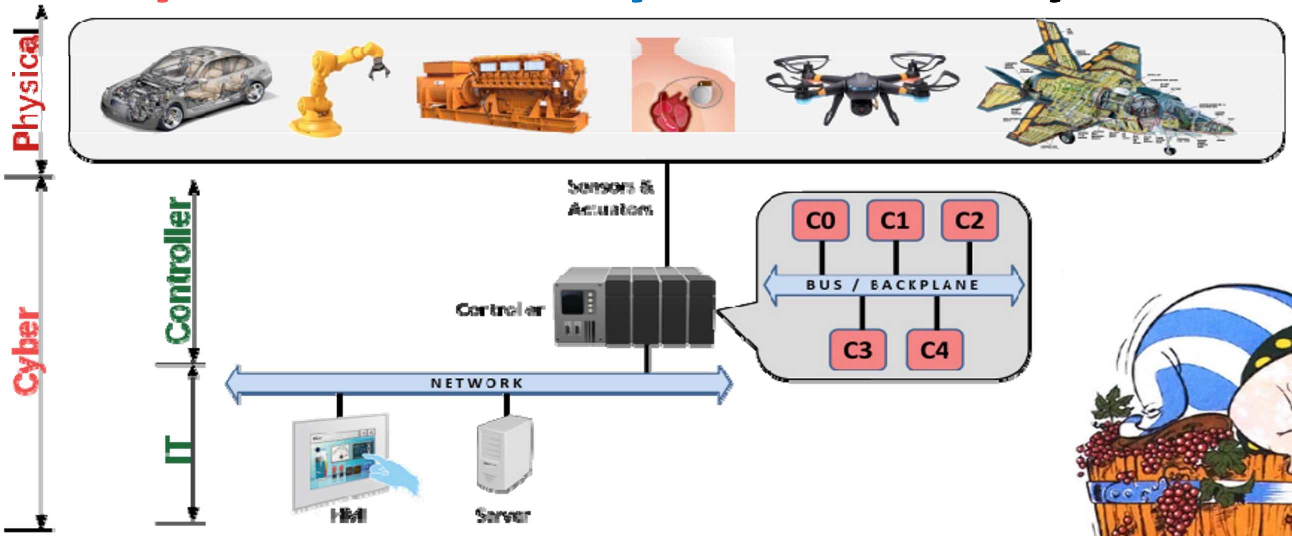
# Task 14
# SubProcess BFT++:
# Robust Cyber Attack Resilience for Production Industrial Control Systems

Dr. Sukarno Mertoguno
Dr. Bo Feng
**Muhammad Faraz Karim**

Georgia Institute of Technology

# Cyber Physical Systems




**Subsystems:**

- Physical Subsystems
- Cyber Subsystems
  - IT Space
  - Controller Space (our focus)

# CPS Security Space

**IT Space :**

- Monitoring & intrusion detection is relatively easier due to predictability of CPS operation - industry already working on this space
- Encryption & authentication – many researchers & industry already working on this space

**Controller Space :**

- Knowledge/Model dependent (e.g. digital twin, intrusion detection at controller bus level, etc) - many researchers & industry are working on this
- Encryption & authentication – researchers & industry already working on this space
  - limited computing capacity,
  - authentication is very relevant,
  - but encryption is less so in majority of applications (data is low-level, state dependent & temporal/short-lifetime)
- **Mechanism** (knowledge independent) – our focus

# Leveraging Key Properties of CPS

## Periodicity

- Continuous observe and control loop (scan cycle, usually ~1-100 Hz)


**Periodicity provides tolerance for loss of input**

- Sensitive to latency variations
- Not performing open-ended, general-purpose tasks like IT

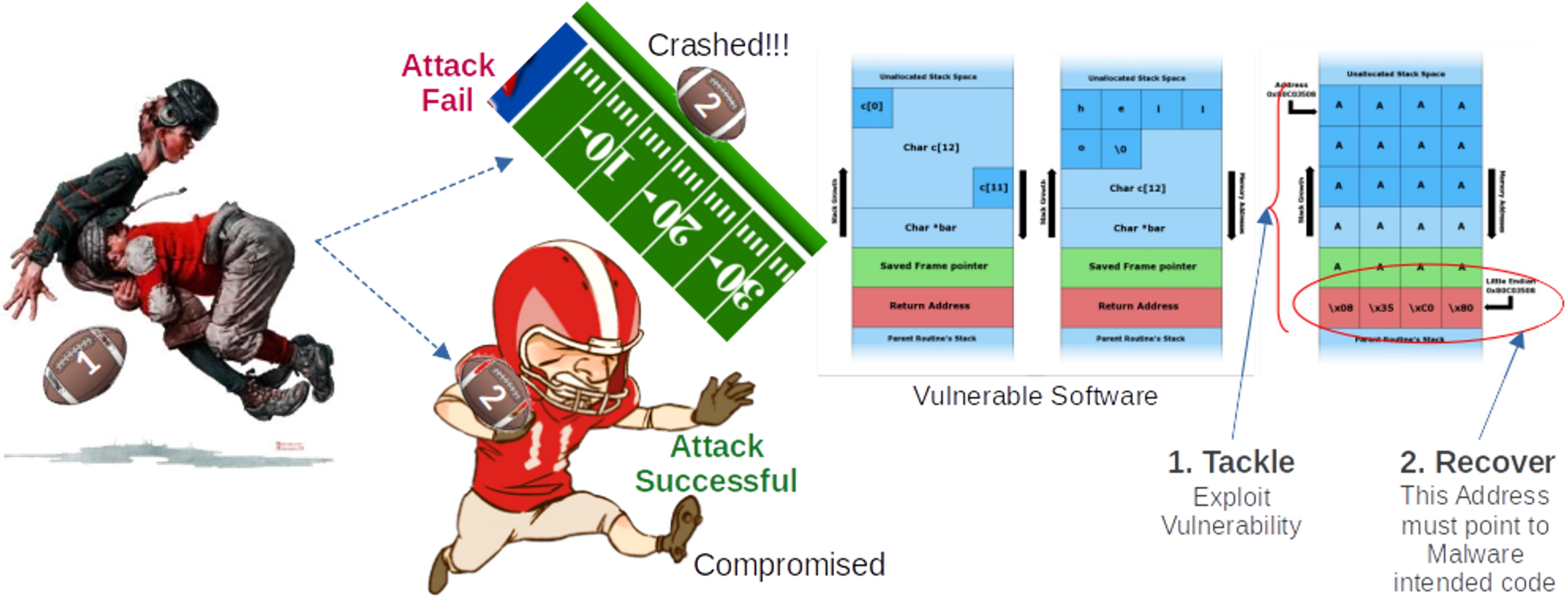
## Inertia

- Physical systems have ***inertia***
- Effect: can tolerate some bad cycles and still maintain stability
  - Missed output
  - Wrong output (sensor blip, etc.)
- In context of cyber attack:

**Inertia provides some natural fault tolerance**

- Not immediately uncorrectable
- How long is system-dependent

# Process of a cyber exploits




**Successful attack requires:**

1. Success on derailing targeted program --> targeted program loses control
2. Success on capturing control --> attacker controls program execution

# Traditional Fault Tolerance

Many systems already employ some type of fault tolerance for physical and random failures:

Redundancy with voting/consensus

Quad Redundant Control (QRC)

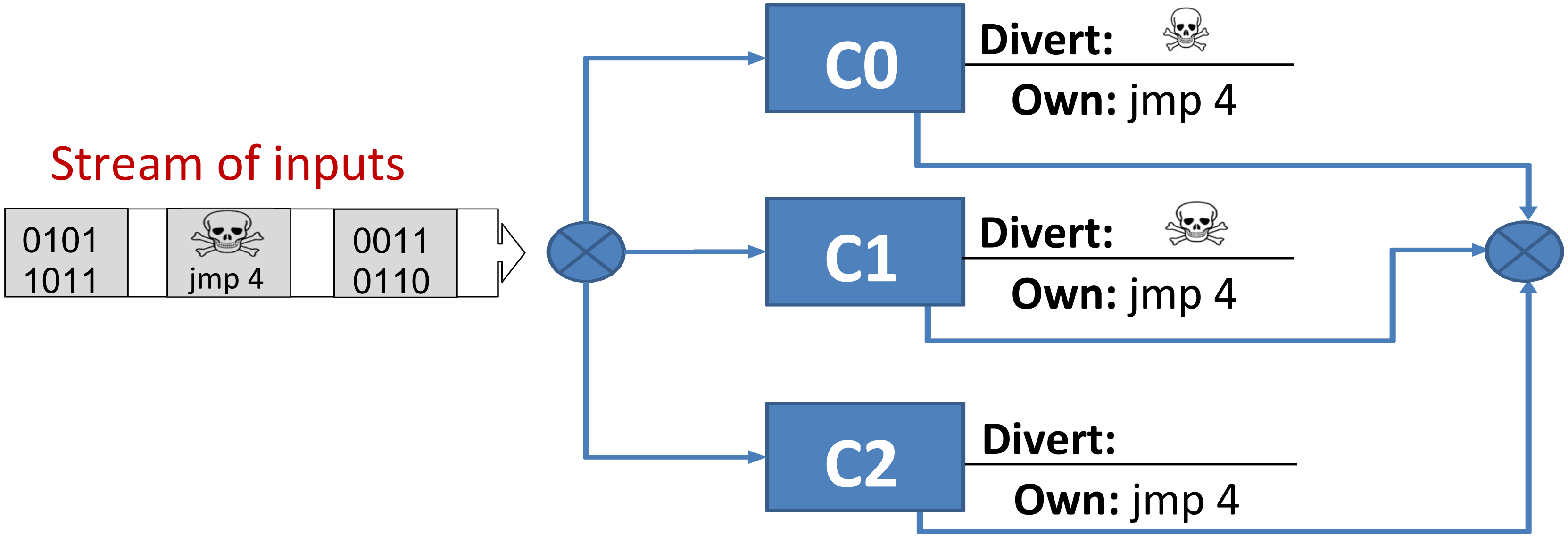
Byzantine Fault Tolerance (BFT)

[BFT allows] systems to continue to work correctly even when there are software errors. Not all errors are survivable; our approach **cannot mask a software error that occurs at all replicas**. However, it can mask errors that occur independently at different replicas, including non-deterministic software errors

Challenge:

**Build Cyber-Attack Tolerance on Traditional Fault Tolerance**

# Traditional Fault Tolerance



# Types of Diversity

- Execution level diversity
  - Same algorithm, same source code
  - Diversifying compiler (DARPA-CRASH)
  - Binary diversifying transformer (ONR, DARPA-CFAR)
- Algorithmic diversity
  - Different algorithm → different source code
  - Exp.: sort → quick sort, bubble sort, merge sort & all sort of sort stuffs.

**BFT++ assumes Execution Level Diversity**

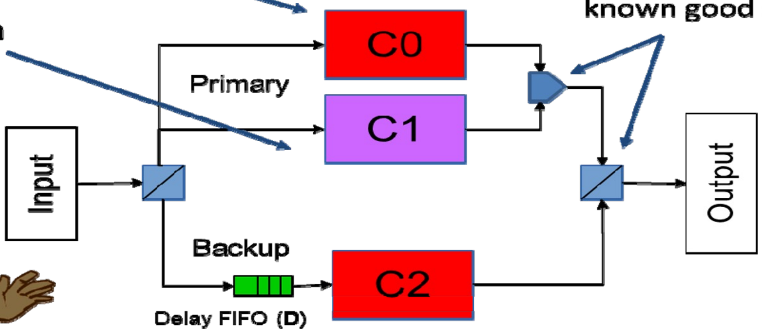
# Key Elements of BFT++

**Techniques work together to provide resiliency against cyber attack-induced failures**

**Artificial diversity**
Protects controllers from being owned

**Controller recovery**
Detects variance and restores controller(s) to known good state

**Engineered FastCrash**
*Limits time spent in corrupted state*

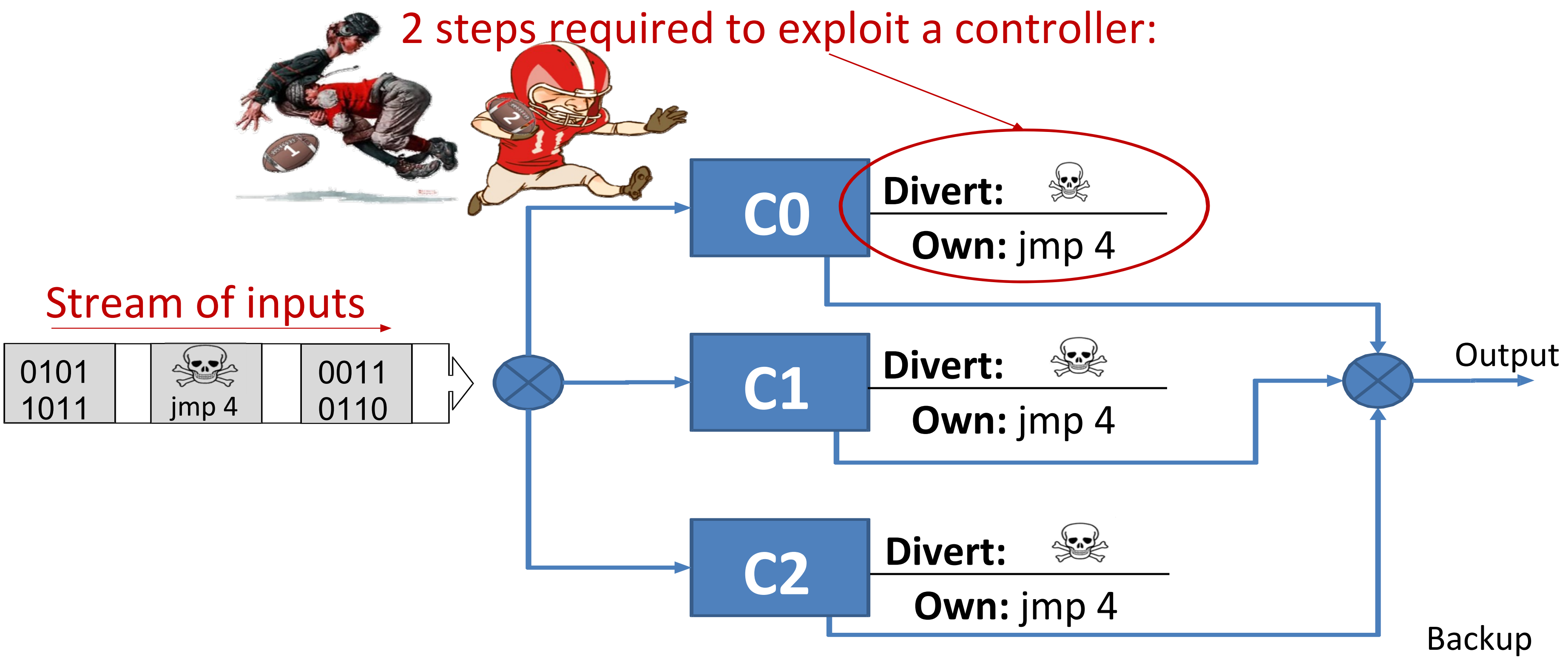



**Delayed input sharing**
Keeps backup in clean state and isolates malicious trigger

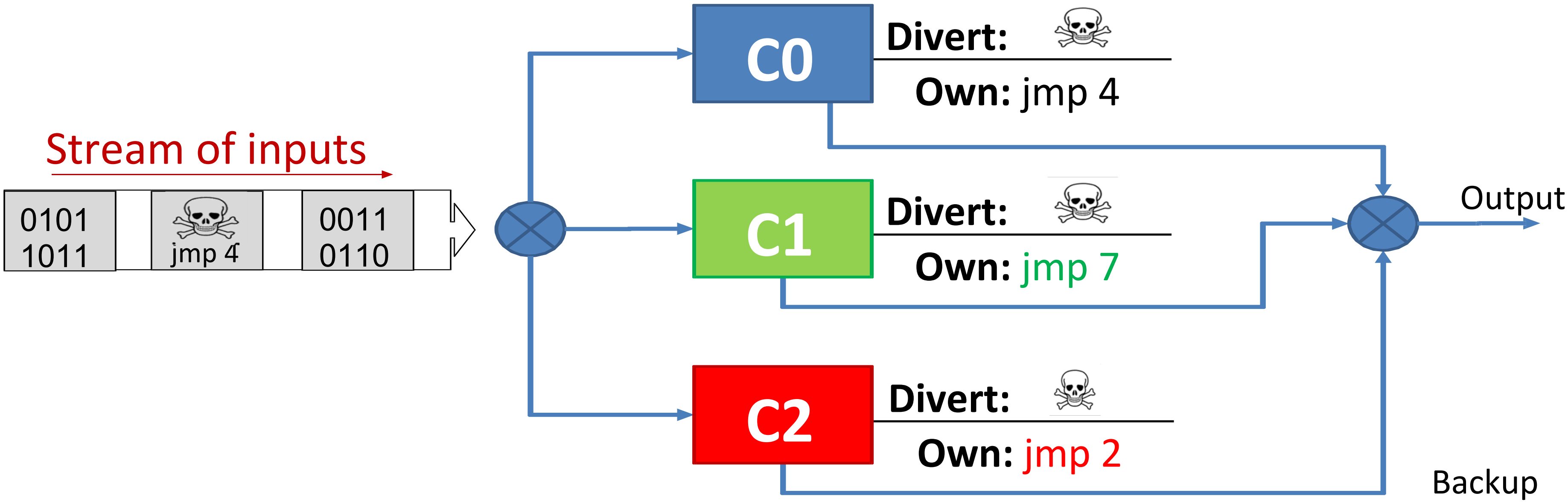
**Successful attack requires:**

1. Success on derailing targeted program  targeted program loses control
2. Success on capturing control  attacker controls program execution

# Example: controller resilience



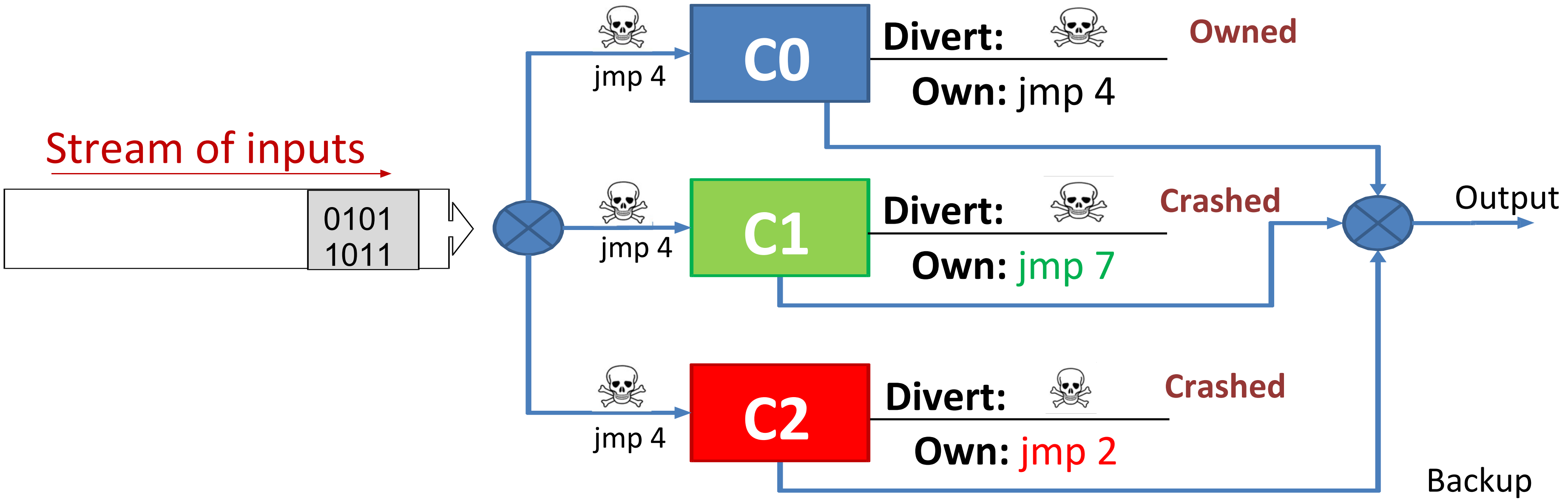
# Artificial Diversity




**Successful attack requires:**

- Success on derailing targeted program  targeted program loses control
- Success on capturing control  attacker controls program execution

# Artificial Diversity




Effect: 1 owned, others crashed

# Controller Recovery

- If we do not need to save controller state:

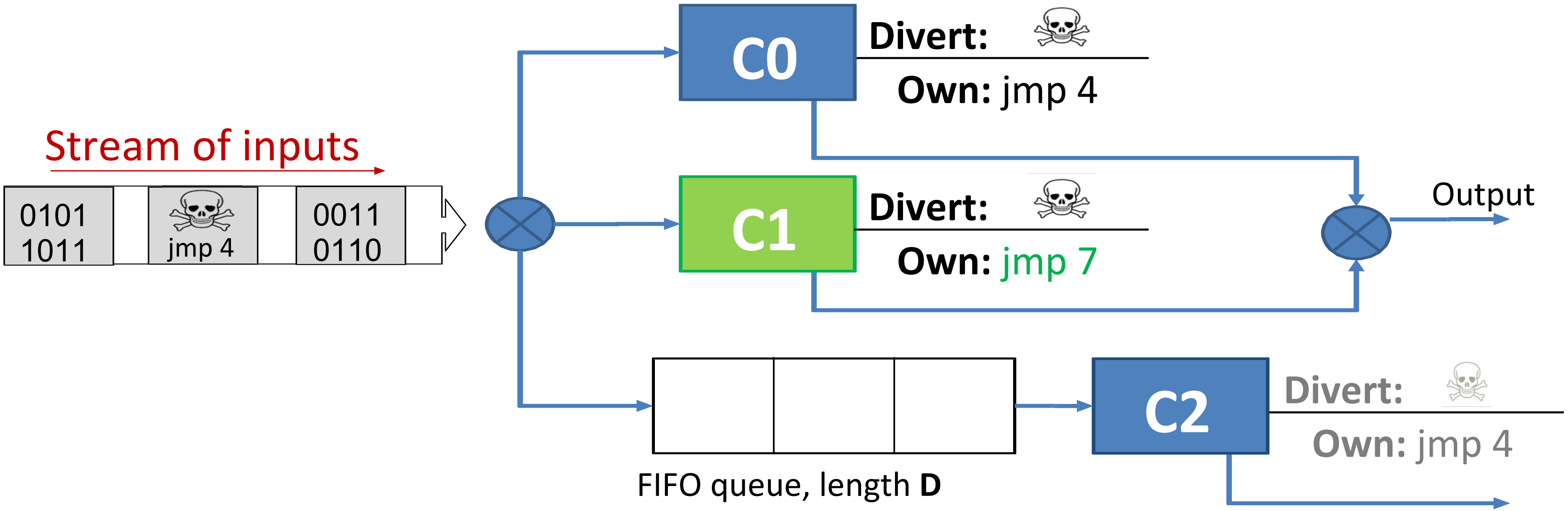
⇨ Restore from a cold backup

- If we need to restore with state, need a hot/warm backup

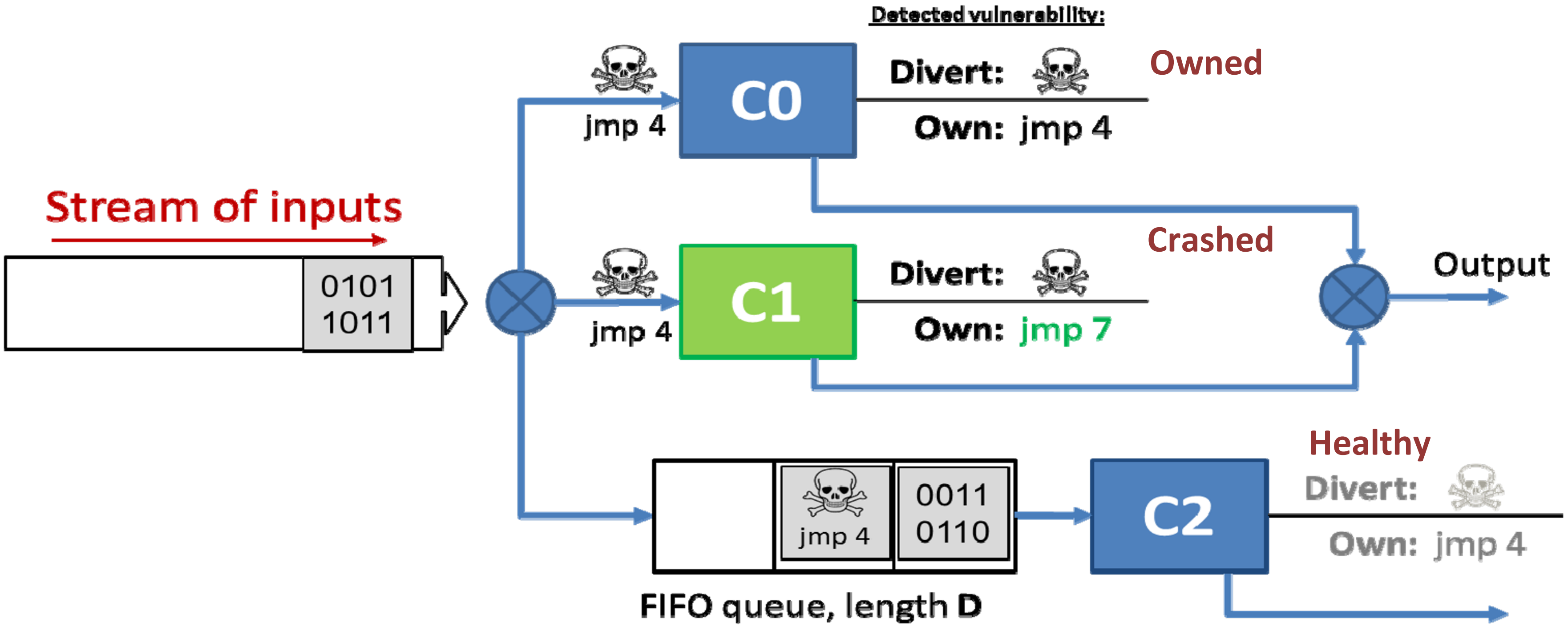
⇨ But how can we keep a hot backup that does not crash or get owned?

  - Must maintain a known good state,
  - check-pointing but it is expensive,
  - or may be not for LEGACY stuffs

# Delayed Input Sharing



# Delayed Input Sharing

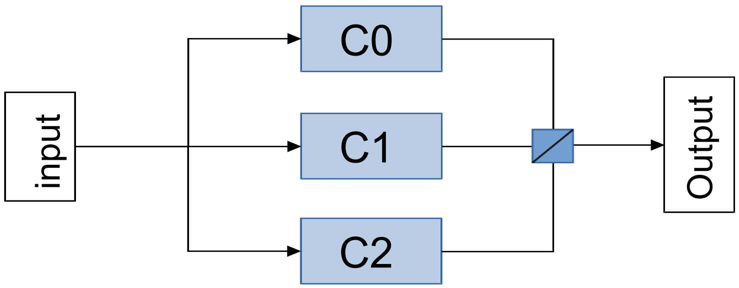



Effect: 1 owned / 1 crashed, but C1's crash trigger is sitting in FIFO queue for C2

# Existing BFT++ variants

**BFT++ v1 (Vanilla) – NRL**

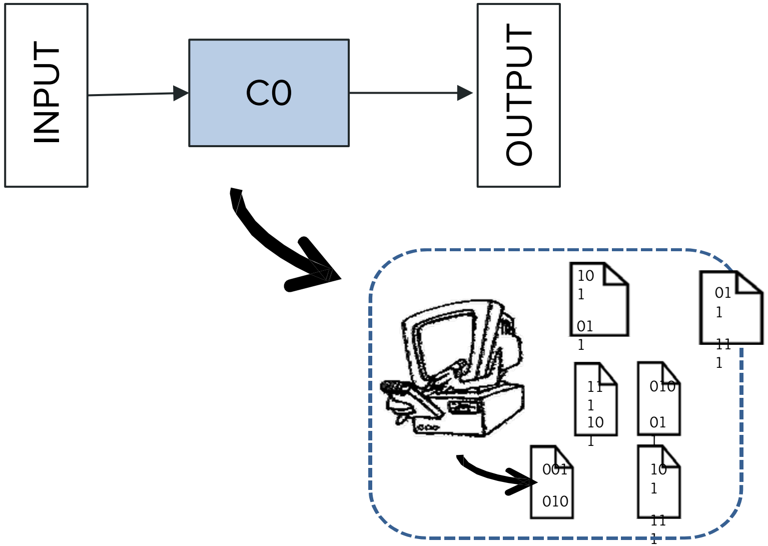
- Multiple replicated devices with artificial software diversity to detect attacks
- A device replica with delayed input to promptly recover from attacks
- **More robust security guarantee**
- **Less service disruptions**
- **Higher cost (due to device replica)**



# Existing BFT++ variants

**BFT++ (YOLO) – Columbia University**

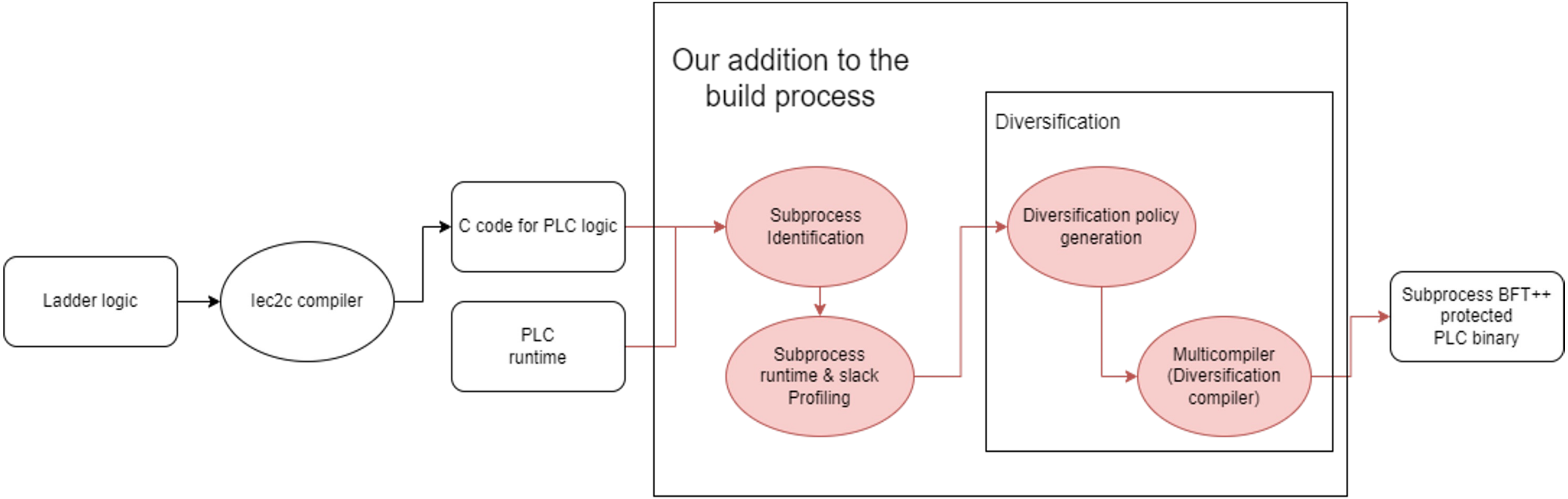
- Firmware diversification (to probabilistically prevent and detect attacks)
- Frequent reset (to recover from attacks)
- **Lower cost**
- **Probabilistic security guarantee**
- **More disruptions (due to frequent reset)**



# SubProcessBFT++

- Goal:
  - **Robust defense**: as robust as the Vanilla variant
  - **Low cost**: comparable to the YOLO variant
- Approach:
  - Operate on the subprocess level (the previous variants of BFT++ operate on the whole program)
  - For each subprocess, determine to duplicate (similar to vanilla variant) or randomize (similar to YOLO variant)
  - Diversify each subprocess according to profile & available slack

# SubprocessBFT++ Workflow



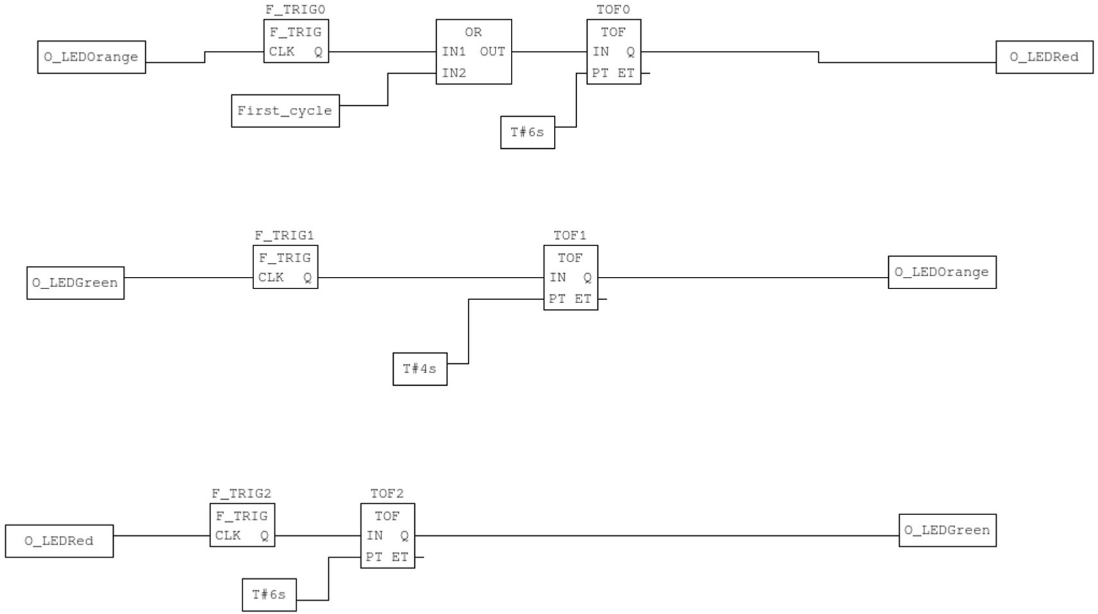

Subprocess BFT++ workflow

# Subprocess

- A single component of the ladder logic of a PLC program.

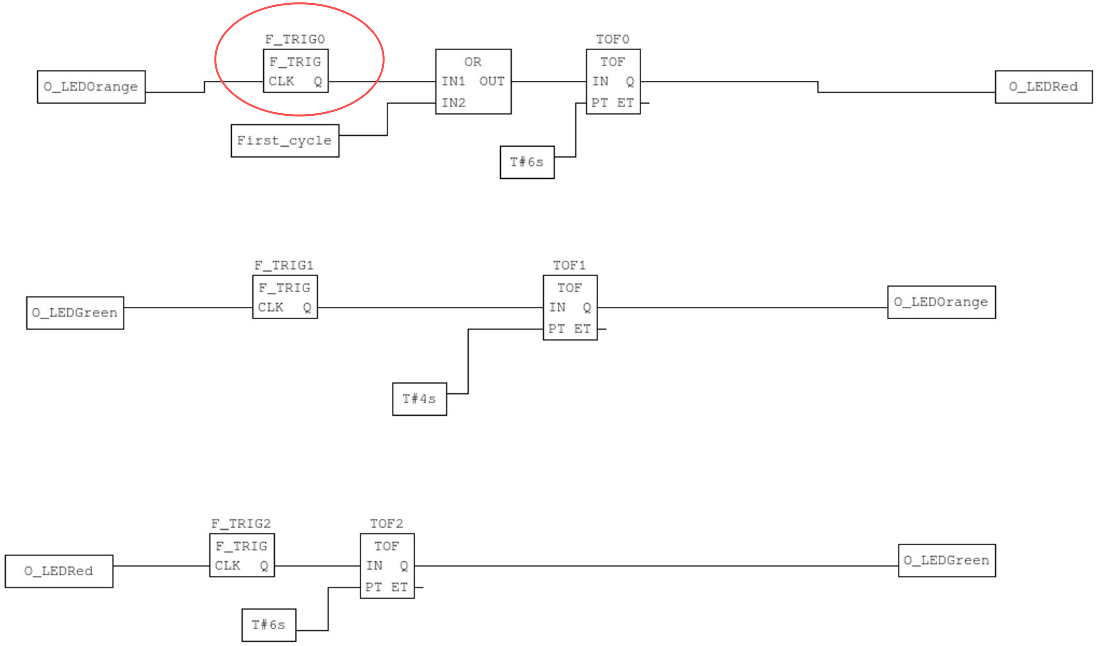
# Subprocess

- A single component of the ladder logic of a PLC program.



# Subprocess

- A single component of the ladder logic of a PLC program.



# Translated c-code

```
// Code part
void TRAFIC_LIGHT_body__(TRAFIC_LIGHT *data__) {
  // Initialise TEMP variables

  __SET_VAR(data__->F_TRIG0.,CLK,,__GET_LOCATED(data__->O_LEDORANGE,));
  F_TRIG_body__(&data__->F_TRIG0);
  __SET_VAR(data__->,_TMP_OR67_OUT,,OR__BOOL__BOOL(
    (BOOL)__BOOL_LITERAL(TRUE),
    NULL,
    (UINT)2,
    (BOOL)__GET_VAR(data__->F_TRIG0.Q,),
    (BOOL)__GET_LOCATED(data__->FIRST_CYCLE,)));
  __SET_VAR(data__->TOF0.,IN,,__GET_VAR(data__->_TMP_OR67_OUT,));
  __SET_VAR(data__->TOF0.,PT,,__time_to_timespec(1, 0, 6, 0, 0, 0));
  TOF_body__(&data__->TOF0);
  __SET_LOCATED(data__->,O_LEDRED,,__GET_VAR(data__->TOF0.Q,));
  __SET_VAR(data__->F_TRIG1.,CLK,,__GET_LOCATED(data__->O_LEDGREEN,));
  F_TRIG_body__(&data__->F_TRIG1);
  __SET_VAR(data__->TOF1.,IN,,__GET_VAR(data__->F_TRIG1.Q,));
  __SET_VAR(data__->TOF1.,PT,,__time_to_timespec(1, 0, 4, 0, 0, 0));
  TOF_body__(&data__->TOF1);
  __SET_LOCATED(data__->,O_LEDORANGE,,__GET_VAR(data__->TOF1.Q,));
  __SET_VAR(data__->F_TRIG2.,CLK,,__GET_LOCATED(data__->O_LEDRED,));
  F_TRIG_body__(&data__->F_TRIG2);
  __SET_VAR(data__->TOF2.,IN,,__GET_VAR(data__->F_TRIG2.Q,));
  __SET_VAR(data__->TOF2.,PT,,__time_to_timespec(1, 0, 6, 0, 0, 0));
  TOF_body__(&data__->TOF2);
  __SET_LOCATED(data__->,O_LEDGREEN,,__GET_VAR(data__->TOF2.Q,));
  __SET_LOCATED(data__->,FIRST_CYCLE,,__BOOL_LITERAL(FALSE));
```

# Slack

- Time remaining after a task finishes before the next task or event is scheduled
- Greater usable slack allows for more subprocesses to be protected by replication in subprocessBFT++

$$UsableSlack = CycleTime - ReservedSlack - \sum SubprocessTime$$

- This Usable slack comes from the time remaining in the Scan Cycle after deducting the time allocated to all subprocess on the processor
- Users can also specify a reserve of Scan cycle for their systems to retain

# Slack

- Time remaining after a task finishes before the next task or event is scheduled
- Greater usable slack allows for more subprocesses to be protected by replication in subprocessBFT++

$$UsableSlack = CycleTime - ReservedSlack - \sum SubprocessTime$$

- This Usable slack comes from the time remaining in the **Scan Cycle** after deducting the time allocated to all subprocess on the processor
- Users can also specify a reserve of Scan cycle for their systems to retain

# Slack

- Time remaining after a task finishes before the next task or event is scheduled
- Greater usable slack allows for more subprocesses to be protected by replication in subprocessBFT++

$$UsableSlack = CycleTime - ReservedSlack - \sum SubprocessTime$$

- This Usable slack comes from the time remaining in the Scan Cycle after deducting the **time allocated to all subprocess** on the processor
- Users can also specify a reserve of Scan cycle for their systems to retain

# Slack

- Time remaining after a task finishes before the next task or event is scheduled
- Greater usable slack allows for more subprocesses to be protected by replication in subprocessBFT++

$$UsableSlack = CycleTime - ReservedSlack - \sum SubprocessTime$$

- This Usable slack comes from the time remaining in the Scan Cycle after deducting the time allocated to all subprocess on the processor
- Users can also specify **a percentage of Scan Cycle** for their systems to retain even after securing it with SubprocessBFT++

# OpenPLC and firmware profiling

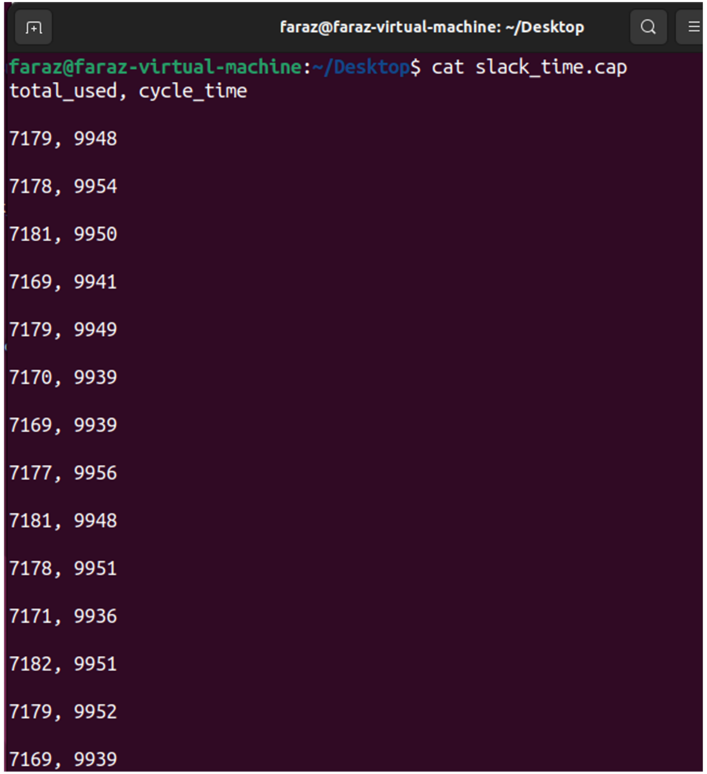
- We have edited the OpenPLC compilation process to profile the subprocesses and how much time they take on the MCU in a single cycle

- Allows us to measure in real time the amount of usable slack available in the system

# OpenPLC and firmware profiling

- First column: The sum time allocated to all subprocesses in a single cycle
- Second column: The total cycle time

** all measurements in microseconds*



# Slack Calculation

- Lets calculate the slack considering as a user we want to reserve 25% of the cycle slack

**All calculations in microseconds*

**Using averages of the last 500 values when run for 1 minute*

**Total Slack = (9948) - (7175) = 2773**

**Reserved Slack = 2773 * 0.25 = 693**

**Usable Slack = (2773) - (693)**

**= 2080 micro seconds**

```
faraz@faraz-virtual-machine: ~/Desktop
faraz@faraz-virtual-machine:~/Desktop$ cat slack_time.cap
total_used, cycle_time

7179, 9948

7178, 9954

7181, 9950

7169, 9941

7179, 9949

7170, 9939

7169, 9939

7177, 9956

7181, 9948

7178, 9951

7171, 9936

7182, 9951

7179, 9952

7169, 9939
```

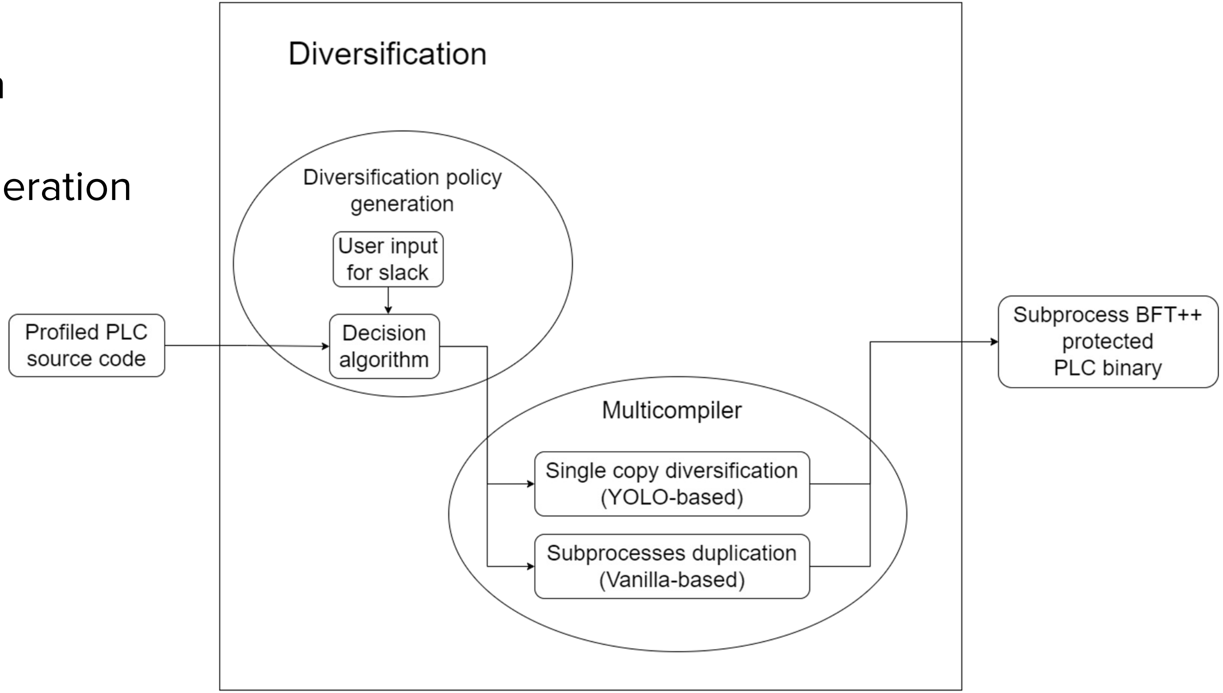
# Discussion

- The greater the value of the slack with respect to the total time occupied by subprocesses the more we are able to use the duplication technique which has a greater level of security

- Replication not possible for all subprocesses as we do not have enough usable slack

# Diversification

- Subprocess identification
- Slack profiling
- Diversification policy generation
- Diversification



# Diversification by multicompiler

- Multicompiler is our choice of diversification tool as it works on source code build by Michael Franz in UCI

- LLVM-based compiler to create artificial software diversity to protect software from code-reuse attacks.

- However, it works only for x86 targets. We ported it to ARM, a popular architecture for PLCs

# Prototyping with OpenPLC

- Open-source Programmable Logic Controller development environment

- Widely used in industrial, home automation, and Internet of Things.

- Can produce PLC programs for a wide range of hardware, from Raspberry Pi to cloud servers

- Very practical for automating legacy systems since it can run on a range of hardware, and does not require great processing power

OPENPLC

TO A MORE
OPEN FUTURE

# Prototyping with OpenPLC

- Our decision algorithm runs when OpenPLC compiles the firmware and decides which part of the system is protected by which methodology

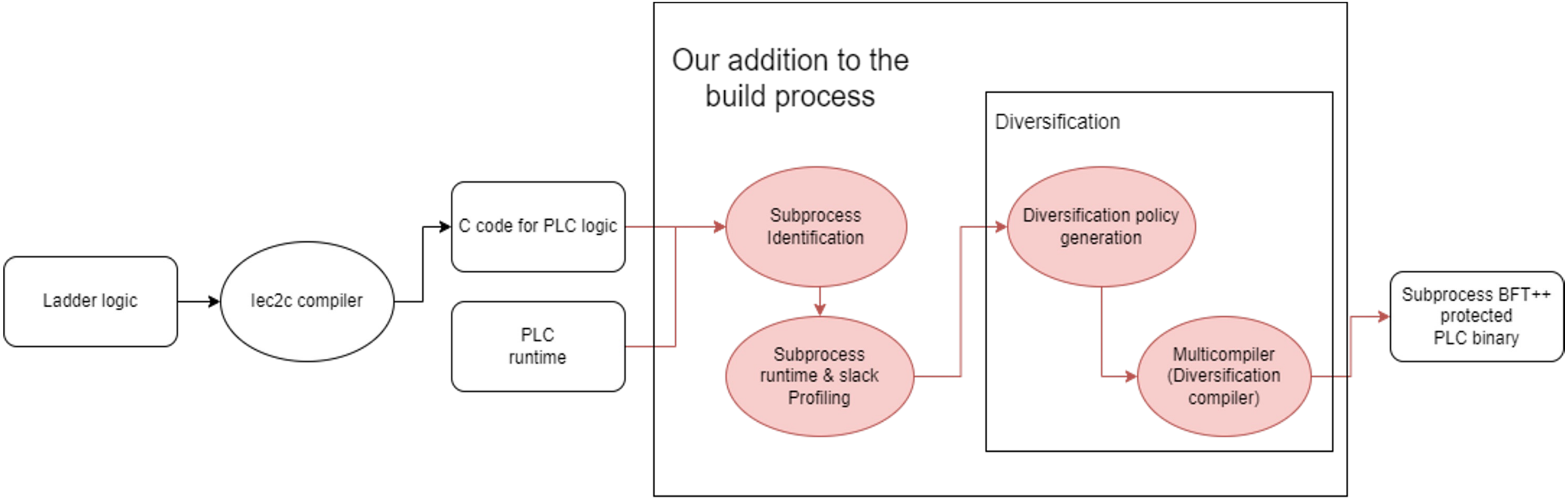
- The OpenPLC compilation workflow is also edited to allow for compilation using multicompiler

- Our experiments used the Arduino NanoRP2040Connect

OPENPLC

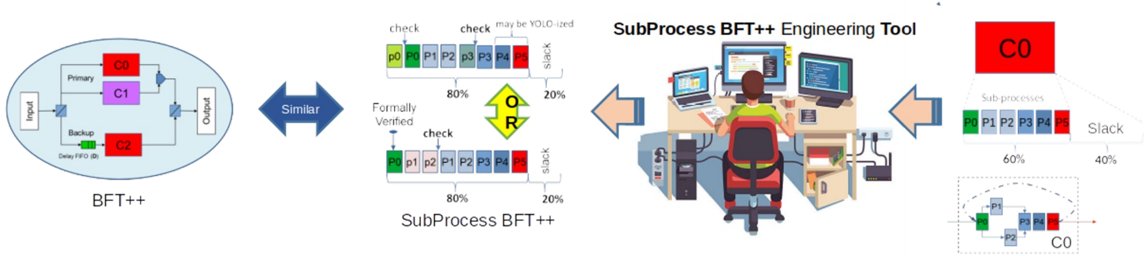
TO A MORE
OPEN FUTURE

# SubprocessBFT++ Workflow




Subprocess BFT++ workflow

# Potential of SubprocessBFT++




- Significantly widen the applicability of BFT++ and provide resilience against direct cyber-attack
- Providing cyber attack resilience for application which cannot afford device redundancy, alleviate the need for redundant device in SubprocessBFT++
- Provide a degree of user control over the security to cost ratio
- Layered defence Automated isolation of offending data, which can be communicated to other system components, e.g. SCATOPSY, RAM2., to prevent repeat attack.
- Integration into OpenPLC design environment for ease of deployment and dissemination.
- Discussion with Siemens for potential integration of SubprocessBFT++ into with their PLCs for the purpose of commercialization.

# Thank you

# QRC++

- Practically the same thing as BFT++
- Only difference being instead of 3 it has a total of 4 redundant copies