CHECKING IS BELIEVING: EVENT-AWARE PROGRAM ANOMALY DETECTION IN CYBER-PHYSICAL SYSTEMS

Long Cheng, *Member, IEEE,* Ke Tian, Danfeng (Daphne) Yao, *Senior Member, IEEE,* Lui Sha, *Fellow, IEEE,* and Raheem A. Beyah, *Senior Member, IEEE*

UNIVERSITY OF
South Carolina

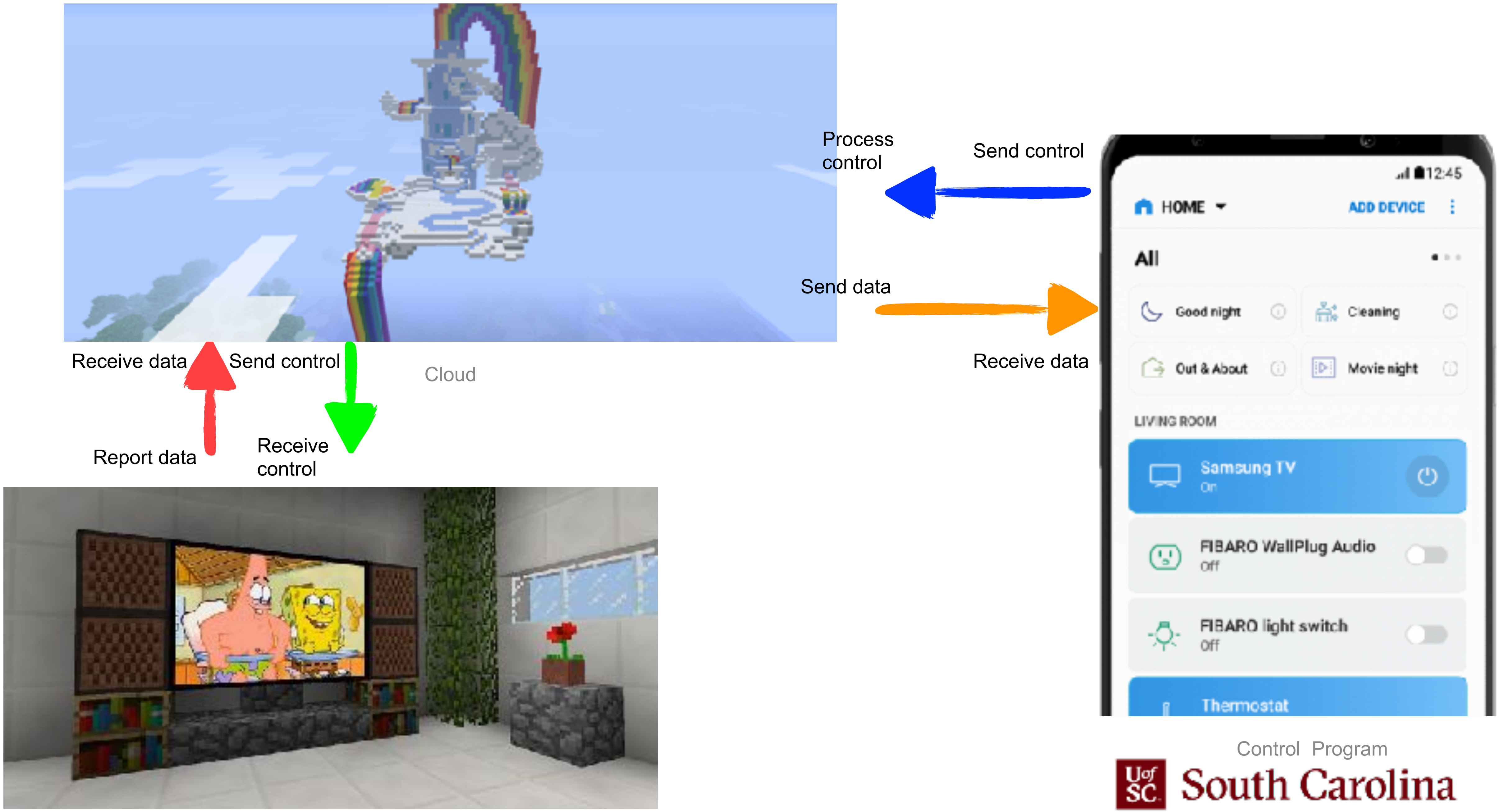
Cyber-physical systems (CPS):

- Consists of computational elements and physical components
- Emphasizes the tightly coupled integration of computational components and physical world

Internet of Things (IoT):

- Emphasizes on the connection of things with networks.

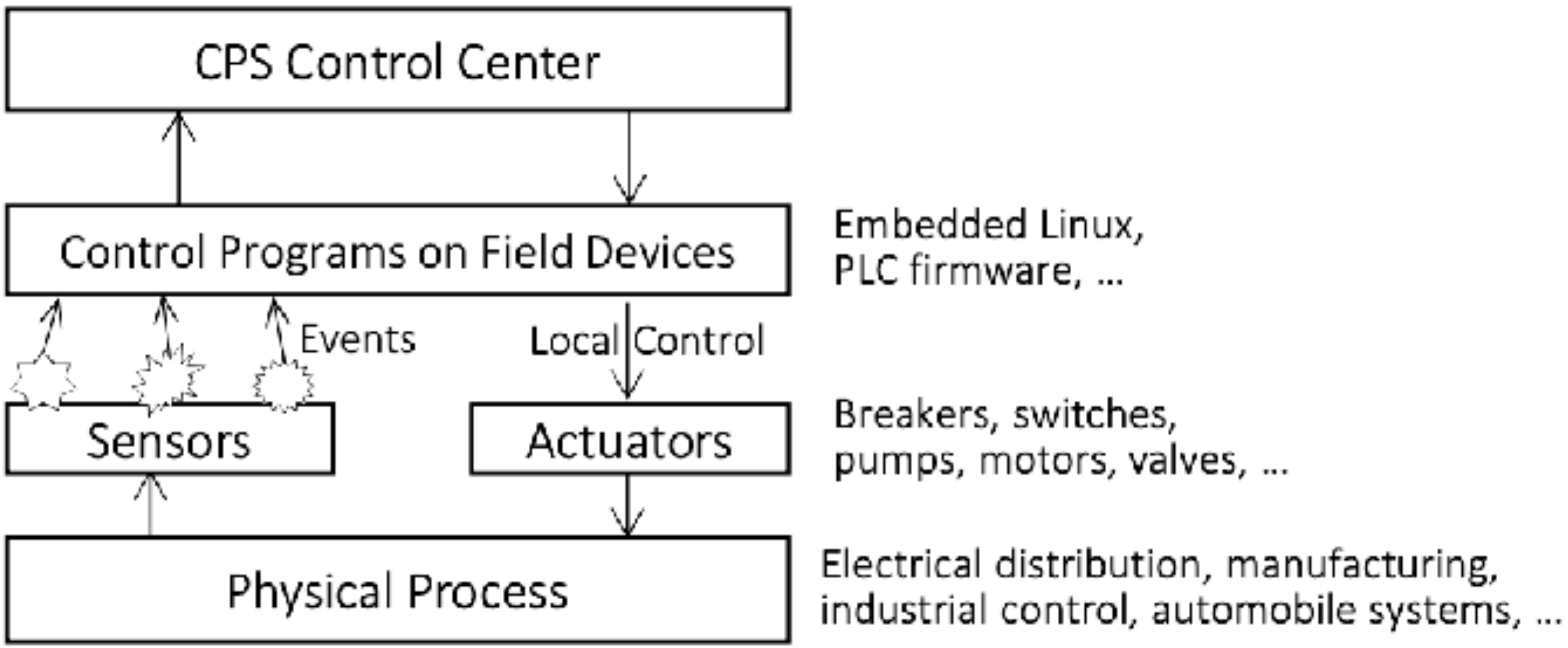
If an IoT system interacts with the physical world via sensors/actuators, we can also classify it as a CPS.



Physical System

South Carolina

ARCHITECTURE



ATTACKS

- Control-oriented attacks: exploit memory corruption vulnerabilities to diver a program’s control flows. e.g., malicious code injunction

- Data-oriented attacks: manipulate programs **internal data variables** without violating its control-flow integrity.

DATA ORIENTED ATTACKS AGAINST CONTROL PROGRAM

- Attacks on control branch

  if (push_event( ) == True)

- Attacks on control intensity

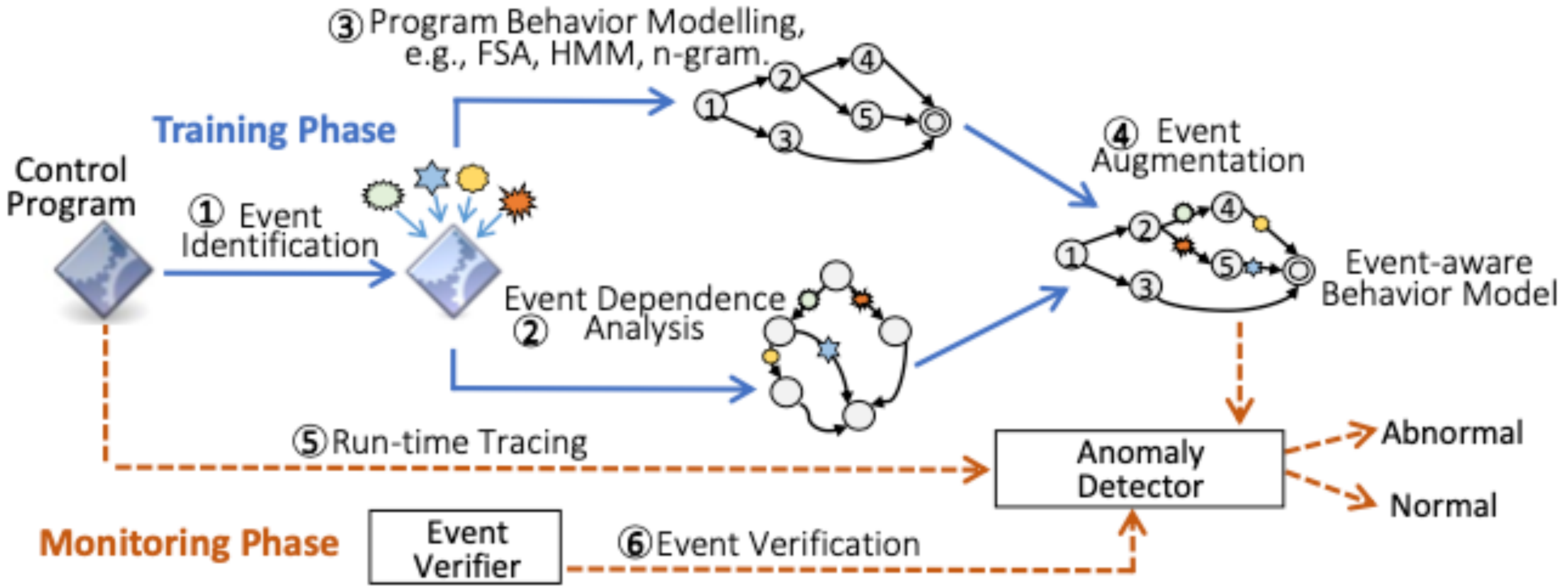
  steps = humidity - HUMIDITY_THRESHOLD

```
void loop(...) {
   readSensors(&pressure,&temperature,&humidity);
   recvRemoteCommand();/*buffer overflow
     vulnerability*/
   ...
   if(push_event()==True)/*Attack control branch*/
       push_syringe();
   else if (pull_event()==True)
       pull_syringe();
}
bool push_event() {
  //decide whether push_event is triggered
  if(humidity>HUMIDITY_THRESHOLD)
    return True;
  return False;
}
void push_syringe() {
  //calculate the steps value
  steps=humidity-HUMIDITY_THRESHOLD;
  for(int i=0; i<steps; i++){/*Attack control
    intensity*/
    digitalWrite(motorStepPin,HIGH);
    delayMicroseconds(usDelay);
    digitalWrite(motorStepPin,LOW);
  }
}
```

MOTIVATION

A data-oriented attack could lead to an inconsistency between the physical context and program control flow.

WORKFLOW OF ORPHEUS



South Carolina

EVENT IDENTIFICATION

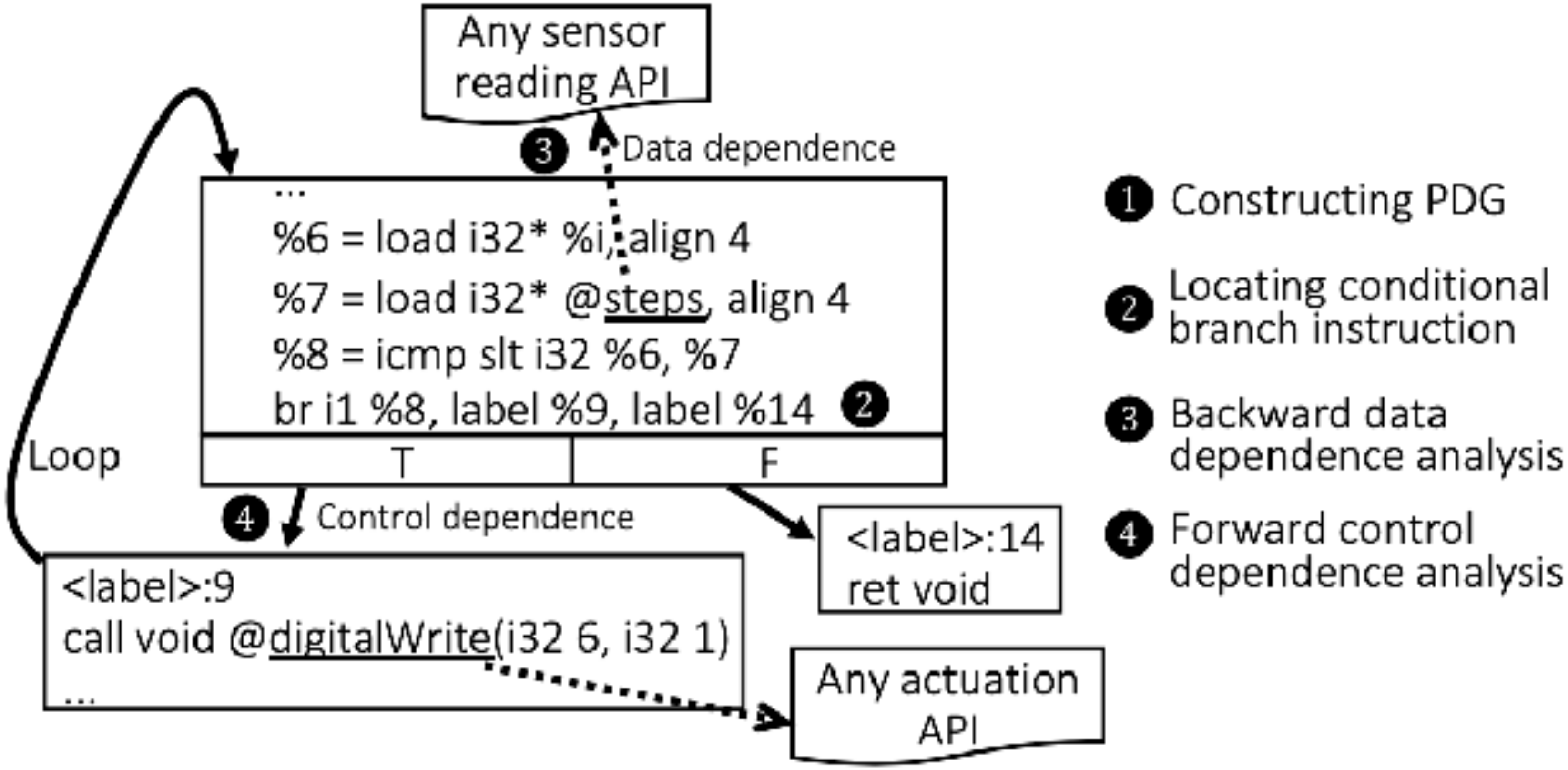
- Binary events
- Control intensity events / loops

```
void loop(...) {
   readSensors(&pressure,&temperature,&humidity);
   recvRemoteCommand();/*buffer overflow
     vulnerability*/
   ...
→  if(push_event()==True)/*Attack control branch*/
       push_syringe();
→  else if (pull_event()==True)
       pull_syringe();
}
bool push_event() {
  //decide whether push_event is triggered
  if(humidity>HUMIDITY_THRESHOLD)
    return True;
  return False;
}
void push_syringe() {
  //calculate the steps value
→ steps=humidity-HUMIDITY_THRESHOLD;
  for(int i=0; i<steps; i++){/*Attack control
    intensity*/
    digitalWrite(motorStepPin,HIGH);
    delayMicroseconds(usDelay);
    digitalWrite(motorStepPin,LOW);
  }
}
```

EVENT DEPENDENCE ANALYSIS

Identify individual events that are involved in a control program.

Associate control-intensity event/loop with the whole loop that contains the sensor-driven control action.



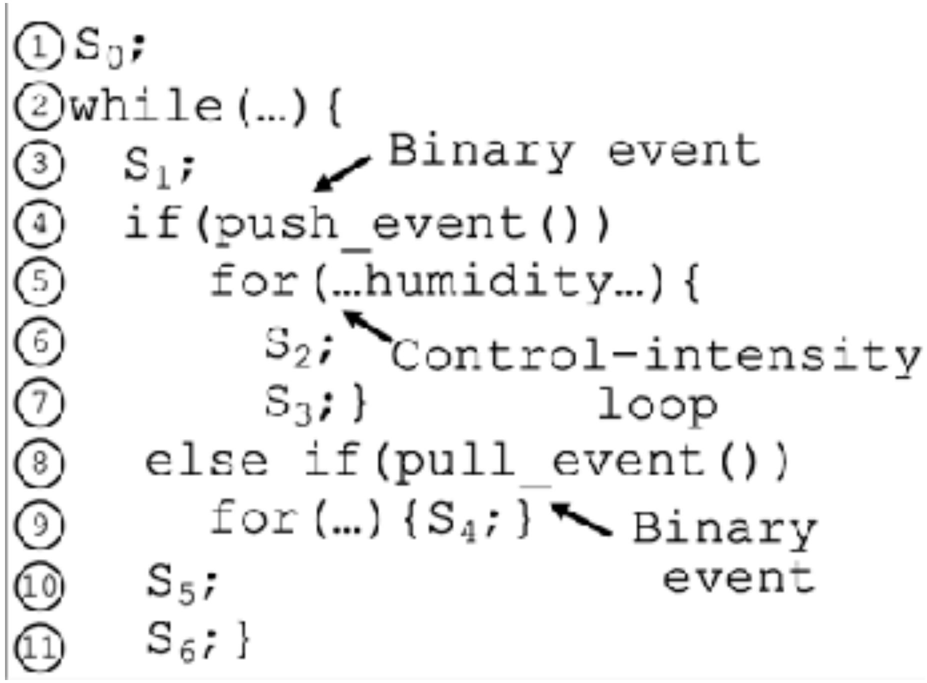
South Carolina

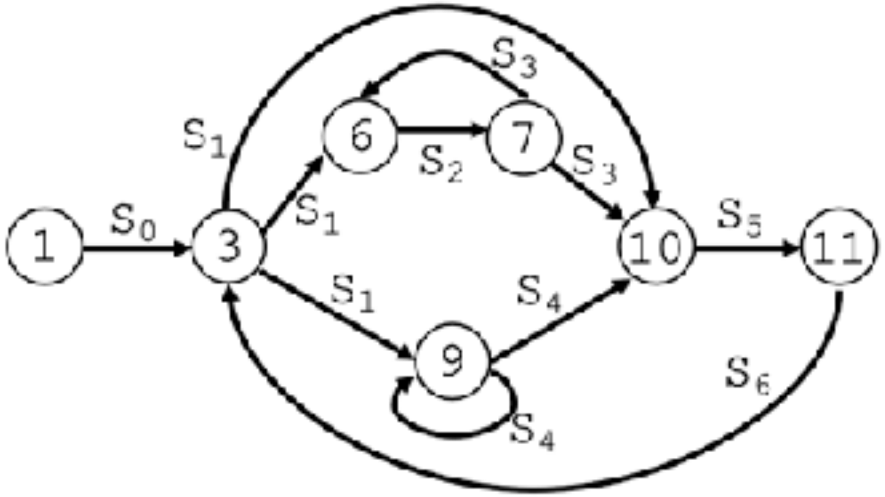
FINITE-STATE AUTOMATON MODEL

Trace the system calls and program counters made by a control program under normal execution.

Each distinct PC (*i.e.*, the return address of a system call) value indicates a different state of the FSA, so that invocation of same system calls from different places can be differentiated. Each system call corresponds to a state transition.

FINITE-STATE AUTOMATON MODEL





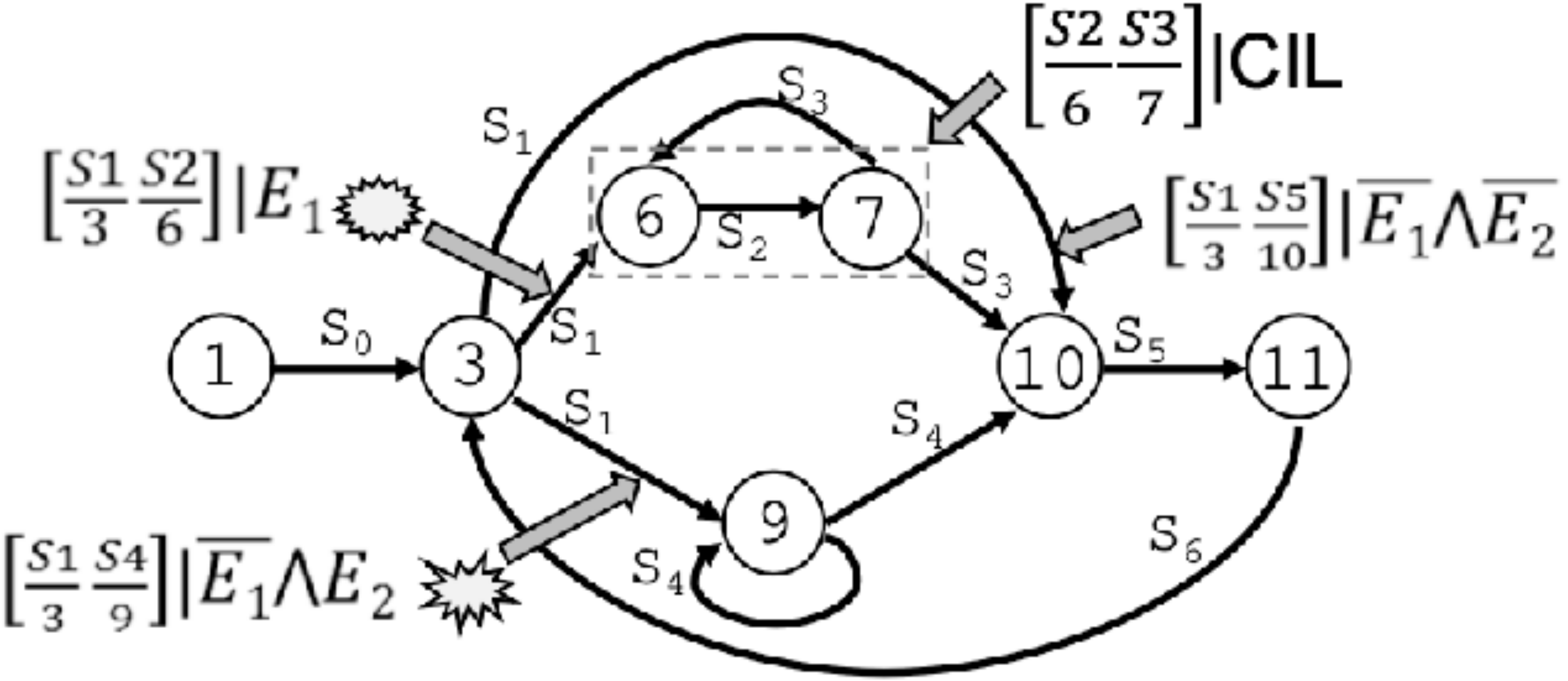
$S_0, \ldots, S_5$ denote system calls

$$\begin{array}{cccccc} S_0 & S_1 & S_4 & S_4 & S_5 & S_6 \\ \hline 1 & 3 & 9 & 9 & 10 & 11 \end{array}$$

$$\begin{array}{ccccccc} S_0 & S_1 & S_5 & S_6 & S_1 & S_5 & S_6 \\ \hline 1 & 3 & 10 & 11 & 3 & 10 & 11 \end{array}$$

$$\begin{array}{cccccccc} S_0 & S_1 & S_2 & S_3 & S_2 & S_3 & S_5 & S_6 \\ \hline 1 & 3 & 6 & 7 & 6 & 7 & 10 & 11 \end{array}$$

EFSA



E_1	pull_event
E_2	push_event
CIL	Control-intensity event/loop

Carolina

Anomaly Detection

1. **Event-independent state transition:** For each intercepted system call, check if there exists an outgoing edge labelled with the system call name from the current state in FSA. If not, an anomaly is detected.
2. **Event-dependent state transition:**
    1. Basic state-transition checking
    2. Check whether a specific physical event associated with this state transition is observed in the physical domain.

IMPLEMENTATION

- main experimental platform: Raspberry Pi 2 with Sense HAT.
- Dynamic tracing: strace-4.13
- Event Identification and Dependence Analysis: Low Level Virtual Machine (LLVM) compiler infrastructure

South Carolina

CASE STUDY

- Solard: an open source controller for boiler and house heating system. Control decisions are made when to turn on or off of heaters by periodically detecting sensor events.
- SyringePump: the control program takes remote user commands via serial connection, and translates the input values into control signals to the actuator.

Training:

Collect execution traces of Solard and SyringePump using training scripts that attempt to simulate possible sensor inputs of the control programs.

Detecting:

The remote user command corrupts the humidity sensor value to be 48.56rH

True

```
→ if(push_event()==True)/*Attack control branch*/
      push_syringe();
→ else if (pull_event()==True)
      pull_syringe();
```

```
bool push_event() {
  //decide whether push_event is triggered
  if(humidity>HUMIDITY_THRESHOLD)
    return True;
  return False;
}
```

humidity = 48.56 HUMIDITY_THRESHOLD = 40

return True

South Carolina

LIMITATIONS

- Bare-metal CPS Devices: Cannot to utilize existing tracing facilities to collect system call traces.
- Time Constraints
- Limitation of Detection Capability