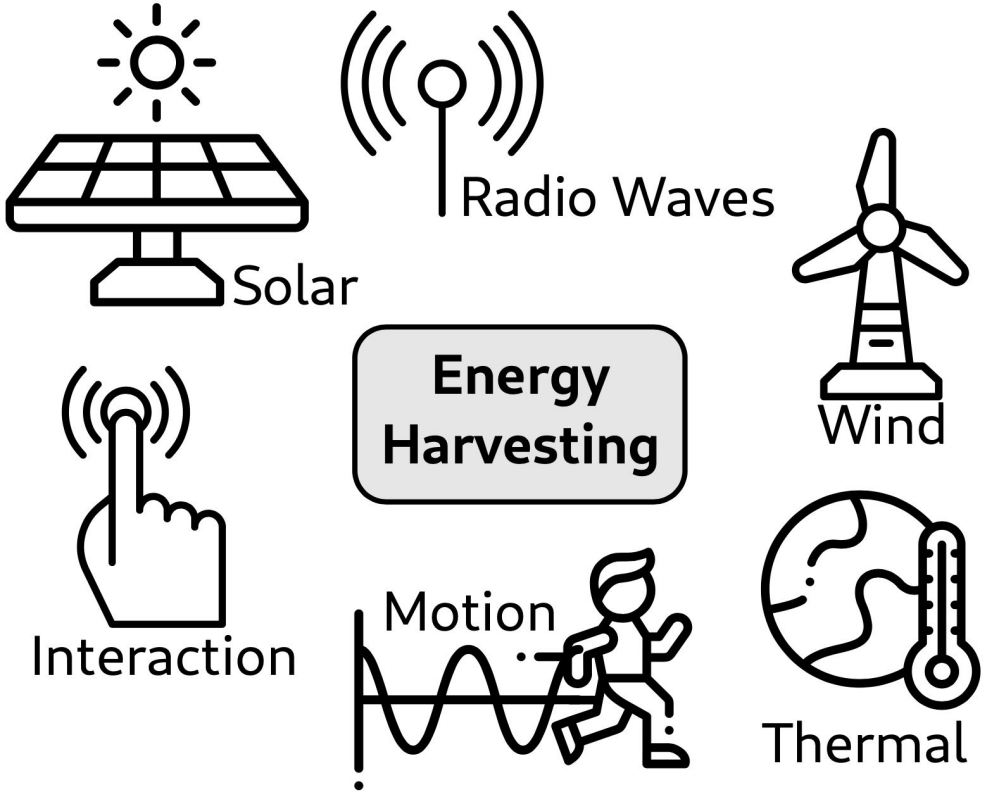
User-directed Assembly Code Transformations Enabling Efficient Batteryless Arduino Applications

Chris Kraemer, William Gelder, Josiah Hester
October 7th, 2024

ckraemer7@gatech.edu

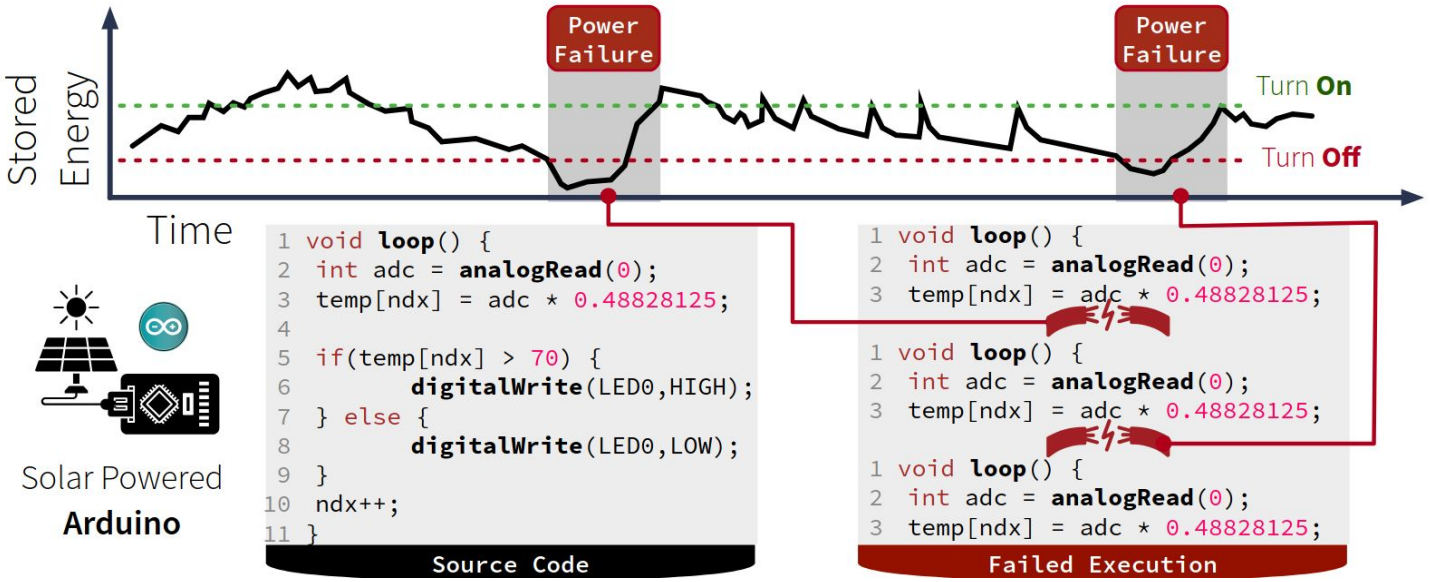
Georgia Tech

No Batteries = Energy Harvesting



ckraemer7@gatech.edu

Energy Harvesting/Intermittent Computing



A Program's execution must be stitched together across power failures because the power coming from energy harvesters is low and dynamic

ckraemer7@gatech.edu

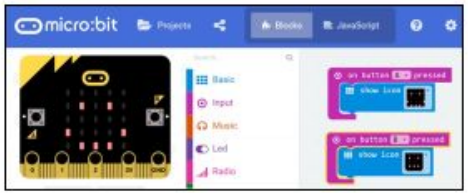
Georgia Tech

Targeting the Maker Ecosystem

Arduino (est. 2005)

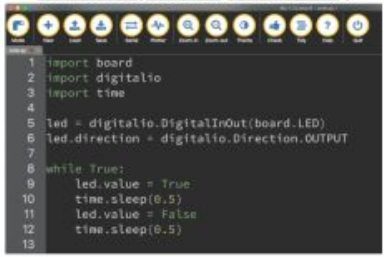
Compiled (C/C++)

Makecode (est. 2017)



Source to Source Translation
(Blocks/JS/Python)

CircuitPython (est. 2017)



```
import board
import digitalio
import time

led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT

while True:
    led.value = True
    time.sleep(0.5)
    led.value = False
    time.sleep(0.5)
```

Interpreted On-Device (Python)
Virtualized Instructions

Which gives us the largest potential userbase?

->Arduino

ckraemer7@gatech.edu

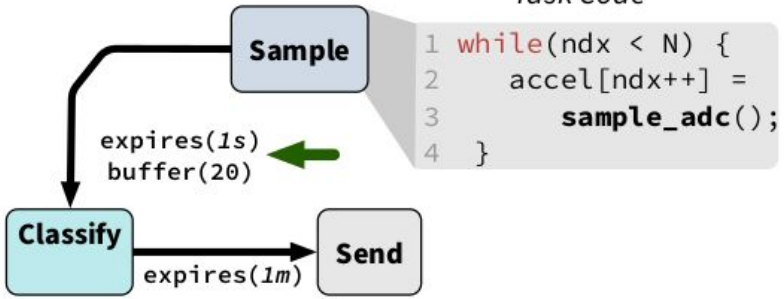
Georgia Tech

Existing Infrastructure(Related Work)

Well… Why don't these enable a battery-free maker ecosystem?

Task-graph behavior defining annotations

Mayfly, InK, CatNap



Checkpoint Boundary Markers

DINO

```
1 void main(){
2   s = rd_sensor();
3   DINO_task()
4   c = classify(s);
5   upd_stats(c)
6   DINO_task()}
```

Behavior Defining Annotations

TICS

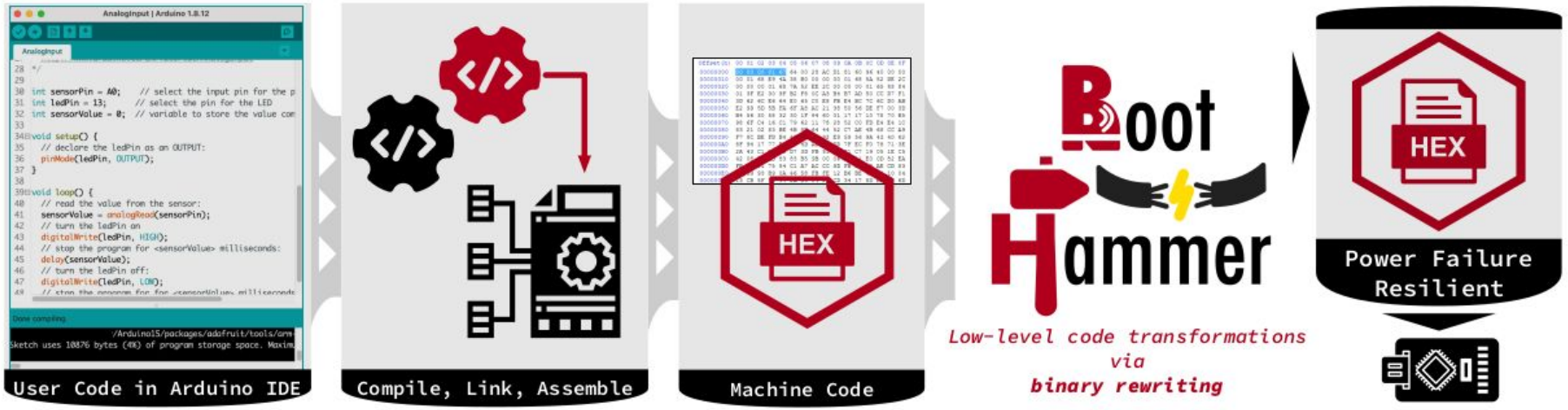
```
1 @expires_after=5 seconds
2 int temp;
3 ...
4 temp @= read_sensor();
5 @expires(temp){
6   if(temp>30)
7     blink_leds();
8 }
```


ckraemer7@gatech.edu

Georgia Tech

This Work: BootHammer

BootHammer produces intermittent code by modifying the incoming hex file and puts the user in charge of checkpointing via user instrumentation

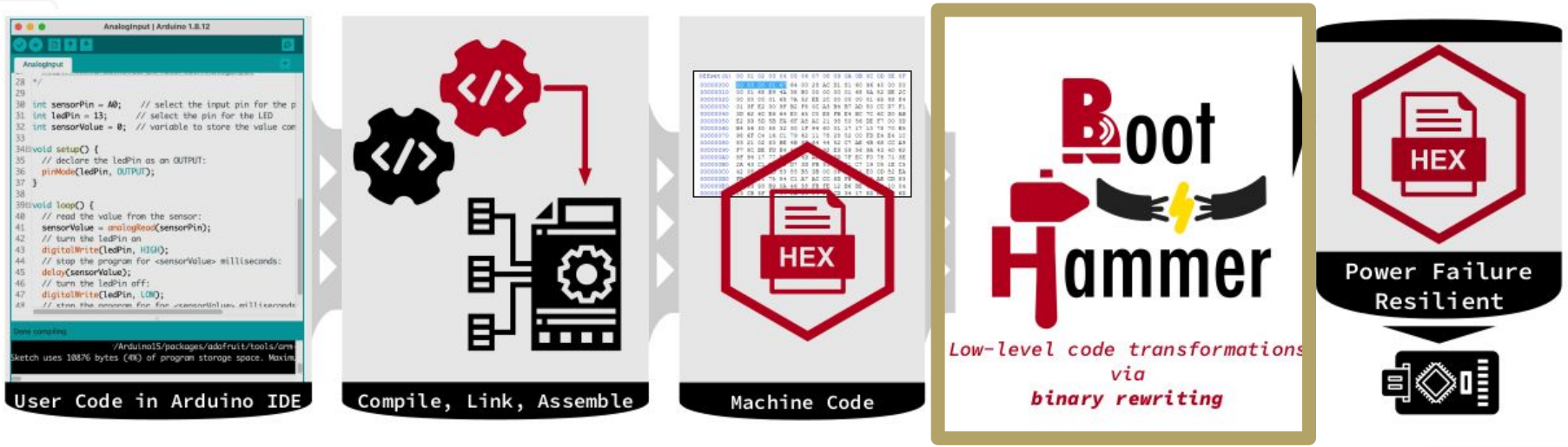


ckraemer7@gatech.edu

Georgia Tech

This Work: BootHammer

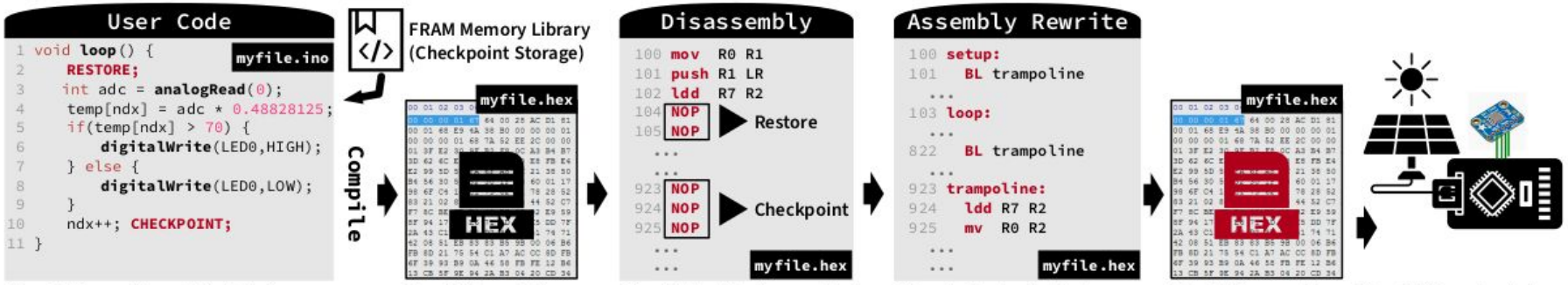
BootHammer produces intermittent code by modifying the incoming hex file and puts the user in charge of checkpointing via user instrumentation



ckraemer7@gatech.edu

Georgia Tech

BootHammer Pipeline



Step 1: User writes code in Arduino, (optionally) annotates with `restore` and `checkpoint` locations. Clicks the "compile/verify" button.

Step 2: Compilation happens as normal.

Step 3: Hex file disassembled to ease analysis for checkpoint insertion.

Step 4: Checkpoint/restore insertion via Assembly rewriting with FRAM library.

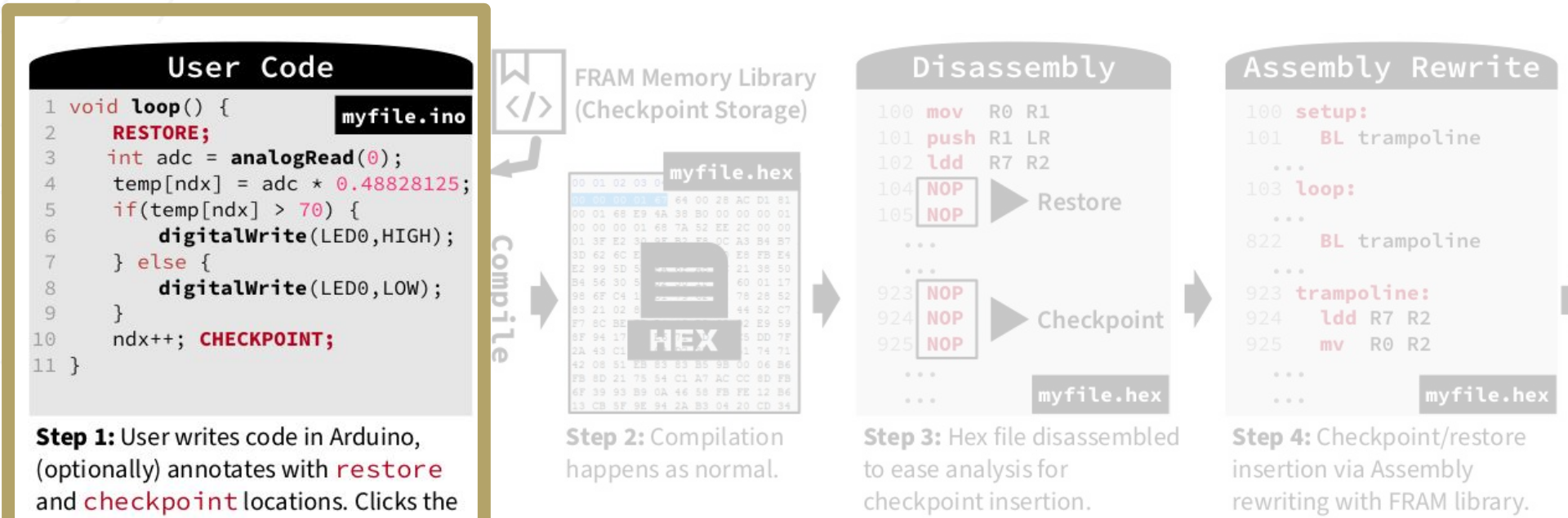
Step 5: Reassembly with included checkpoints/restore.

Step 6: Binary loaded onto device. Now Power failure resilient!

ckraemer7@gatech.edu

Georgia Tech

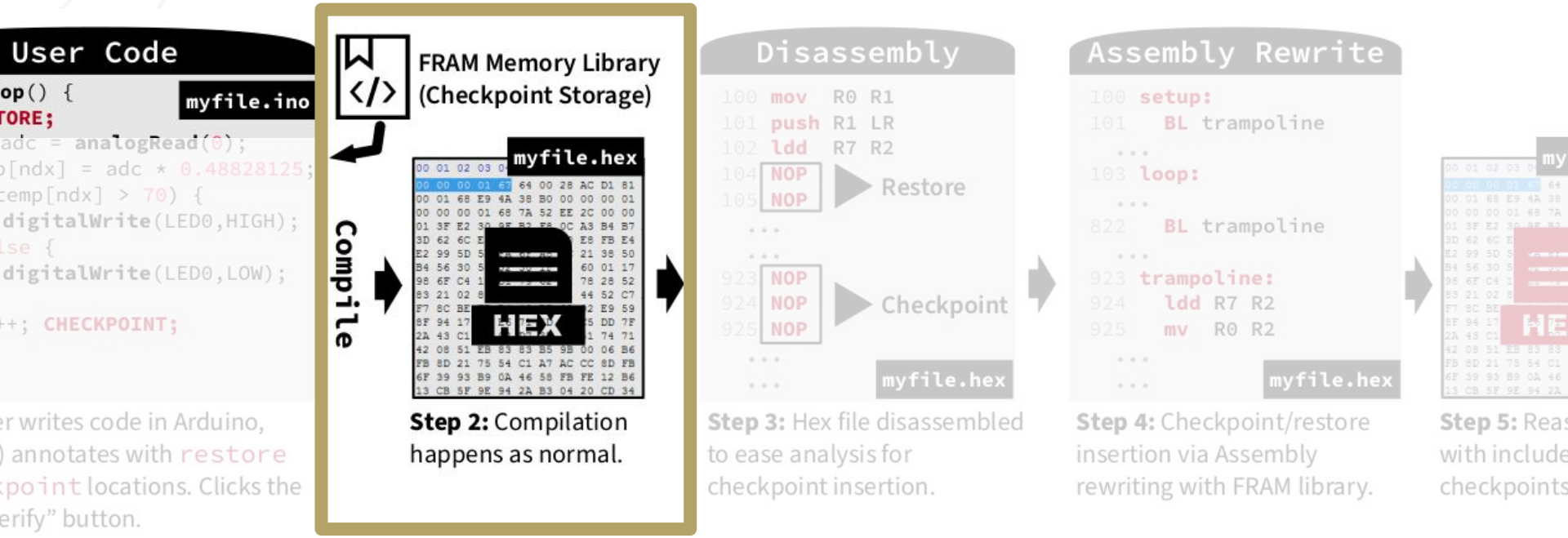
BootHammer Pipeline: User Code Instrumentation



ckraemer7@gatech.edu

Georgia Tech

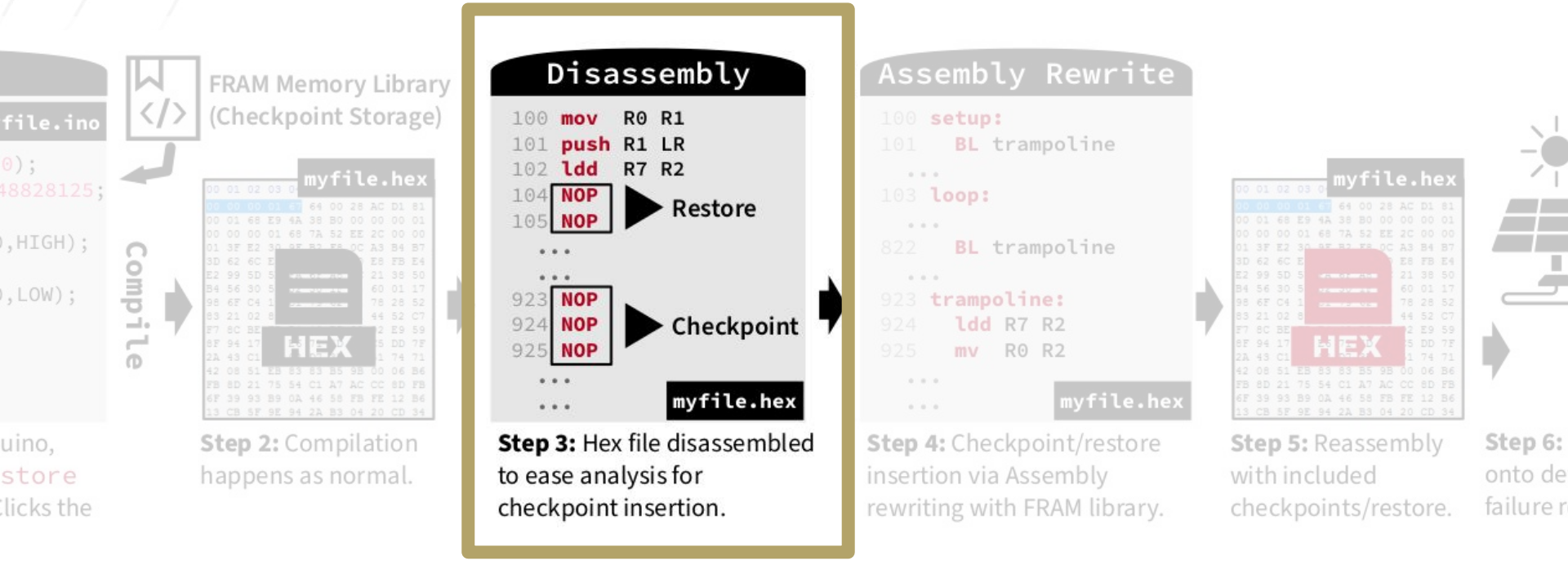
BootHammer Pipeline: Compilation



ckraemer7@gatech.edu

Georgia Tech

BootHammer Pipeline: Disassembly



Step 2: Compilation happens as normal.

Step 3: Hex file disassembled to ease analysis for checkpoint insertion.

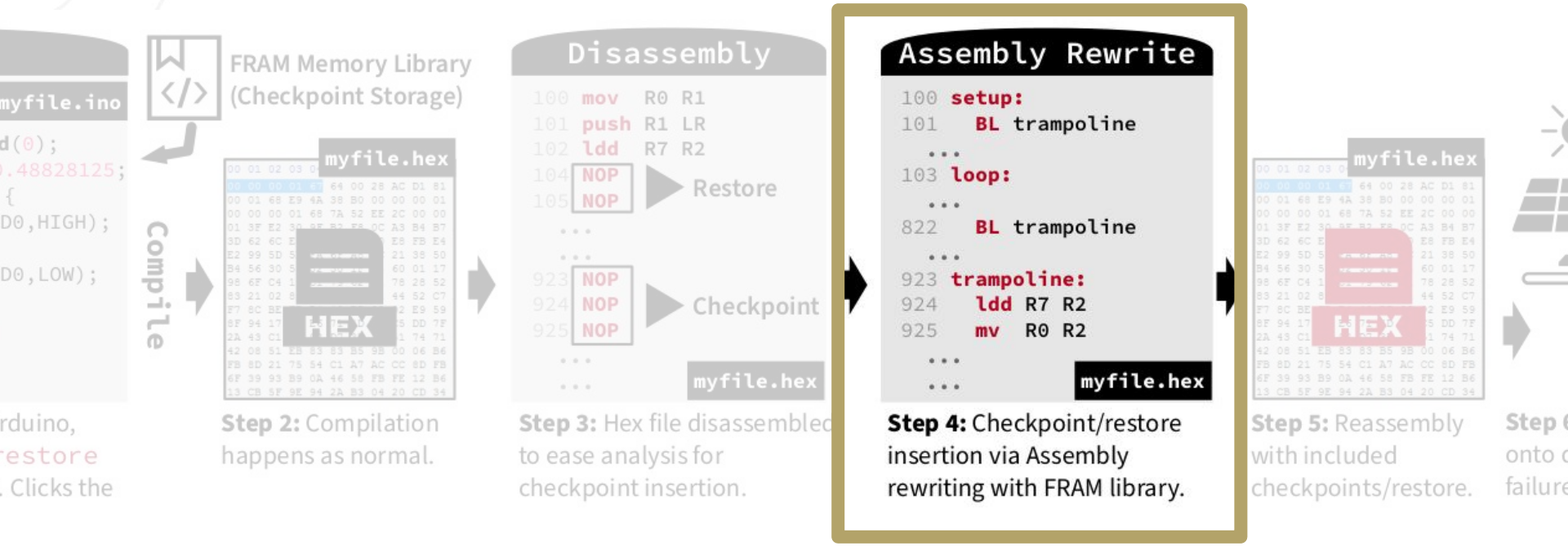
Step 4: Checkpoint/restore insertion via Assembly rewriting with FRAM library.

Step 5: Reassembly with included checkpoints/restore.

ckraemer7@gatech.edu

Georgia Tech

BootHammer Pipeline: Assembly Transformation



Step 2: Compilation happens as normal.

Step 3: Hex file disassembled to ease analysis for checkpoint insertion.

Step 4: Checkpoint/restore insertion via Assembly rewriting with FRAM library.

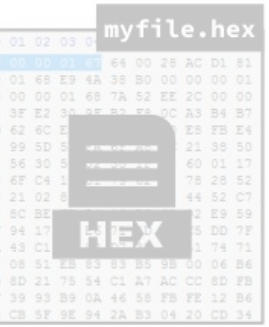
Step 5: Reassembly with included checkpoints/restore.

ckraemer7@gatech.edu

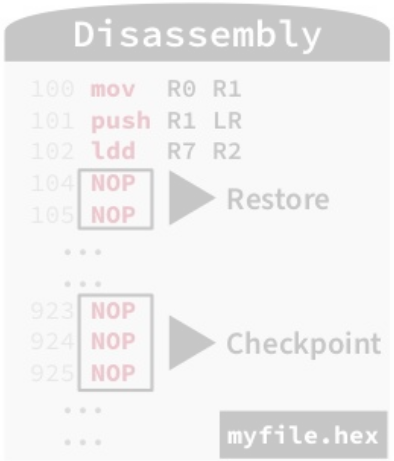
Georgia Tech

BootHammer Pipeline: Reassembly and Flash

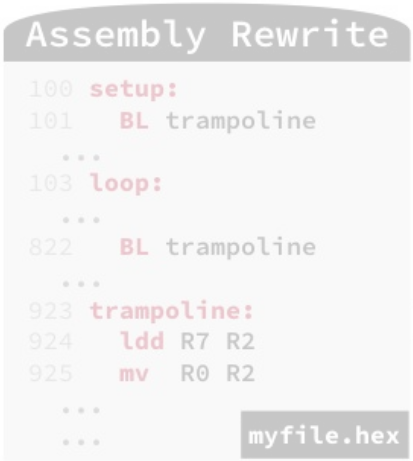
FRAM Memory Library
(Checkpoint Storage)



tep 2: Compilation
appens as normal.

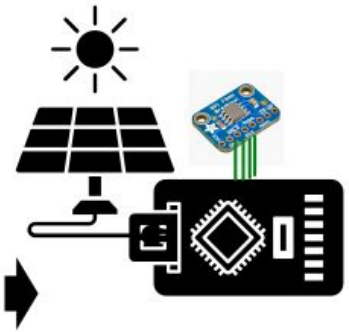


Step 3: Hex file disassembled to ease analysis for checkpoint insertion.



Step 4: Checkpoint/restore insertion via Assembly rewriting with FRAM library.

Step 5: Reassembly with included checkpoints/restore.



Step 6: Binary loaded onto device. Now Power failure resilient!

ckraemer7@gatech.edu

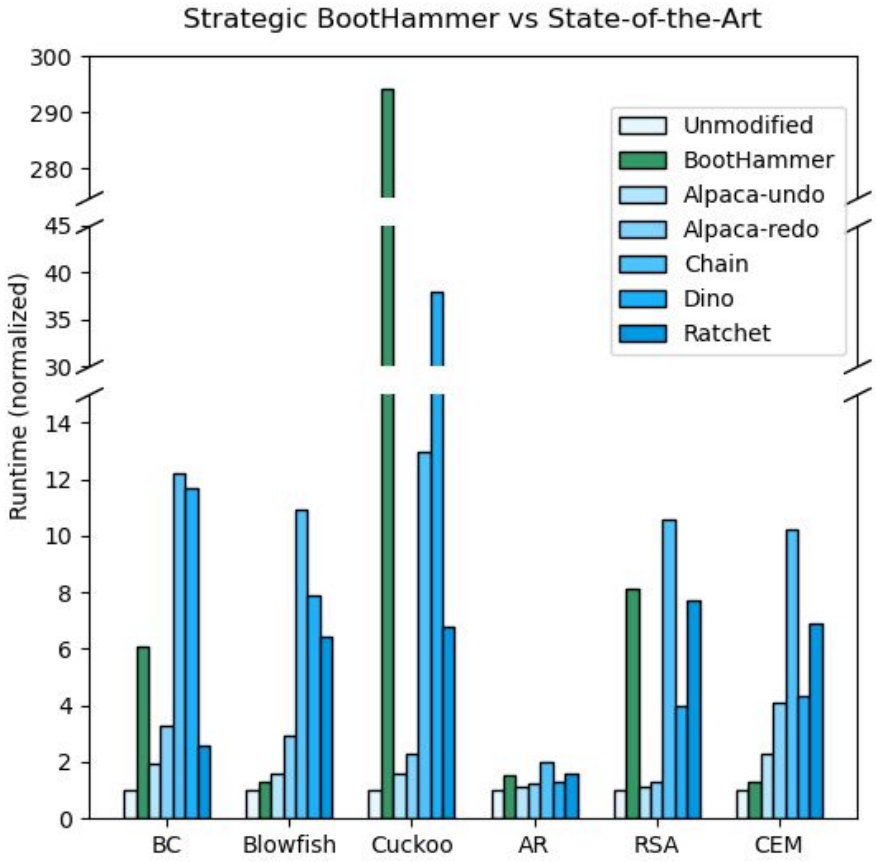
Georgia Tech

BootHammer Evaluation

Normalized runtime overhead comparison between BootHammer and State-of-the-Art intermittent computing techniques

BC-Bitcount
BF-Blowfish
CF-Cuckoo Filtering
AR-Activity Recognition
RSA-RSA Encryption
CEM-Cold Chain Equipment Monitoring

In general, and despite major hardware differences, BootHammer performs quite well compared with the State-of-the-Art



ckraemer7@gatech.edu

Georgia Tech

User Study

We evaluated BootHammer in a small scale user study of 9 participants

Participants were recruited via snowball sampling
- Undergrads and Grads
- Novice to Intermediate Experience with Arduino

In the study participants provided likert responses to our research questions

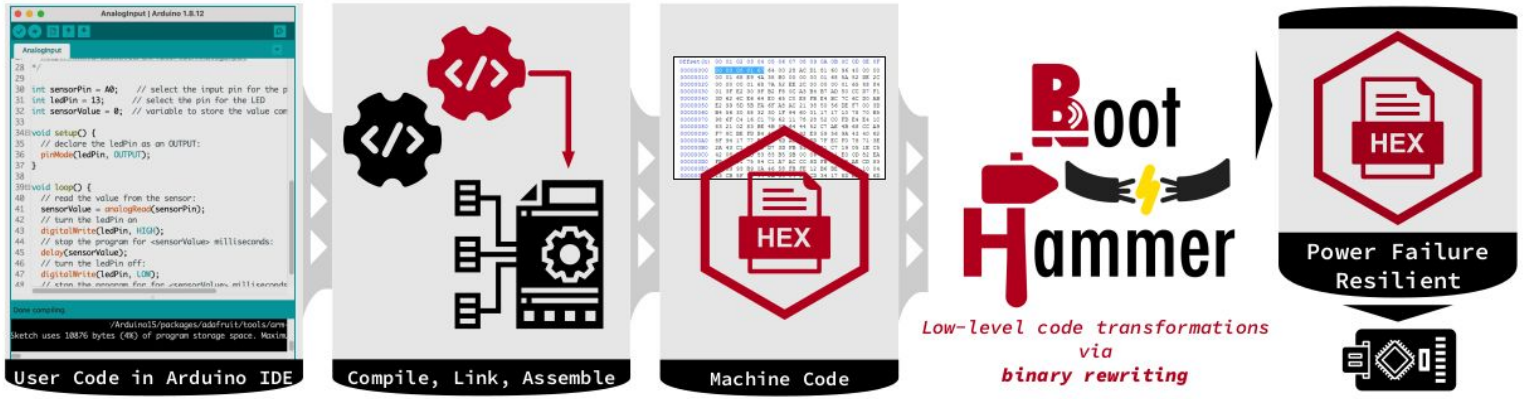
Does BootHammer increase a participant's confidence in their ability to write an intermittent program? 4.6/5

Does BootHammer increase a participant's interest in writing an intermittent/battery-less program? 3.5/5

ckraemer7@gatech.edu

BootHammer and the Future Ahead

- Maker ecosystems provide a unique opportunity to expand access to intermittent computing
- The world needs sustainable makers and BootHammer makes this possible
- BootHammer is a fully fledged disassembler and assembly rewriter based on programmer annotations



ckraemer7@gatech.edu

Georgia Tech

Questions?

Extra Slides

Use Case	Participant Description
Longitudinal e-cigarette study	"... useful for long term usage pattern detection... to try to see how people are quitting smoking."
Persistent health device status	"... health related devices... remember their last status."
Elephant enrichment data collection in zoos	"... when they place their trunk in the hole, it could play a different tone... we need (the device) to be wired all the time... maybe we could get away with doing solar."
Persistent smart home device status	"In South Africa, we have bad power problems... (smart devices) would forget all this data... Knowing I could build projects that can actually just remember states would be so useful."
Lowering barriers for remote sensing	" I think this also lowers the barrier of entry for doing longitudinal testing in remote locations... for months even."
Saving game state	"let's say this controller, it could actually keep track of the save when the power goes out"
Musicians recording audio remotely	"There's also a lot of musicians that record samples and stuff out in weird places'.
Interactive environments	"you could set up a smart space where it can sense what you're doing or your emotional states and generate music throughout your house...if you could run those on this type of stuff, then you could save whatever data you're collecting."

BootHammer at a Glance

Before…

UnModGarden.ino

```
#include <DHT11.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SharpMem.h>
#define SHARP_SCK  30
#define SHARP_MOSI 29
#define SHARP_SS   1
Adafruit_SharpMem display(SHARP_SCK, SHARP_MOSI, SHARP_SS, 144, 168);
#define BLACK 0
#define WHITE 1
int minorHalfSize; // 1/2 of lesser of display width or height
DHT11 dht11(A10);
void setup() {
  Serial.begin(9600);
  delay(1000);
  display.begin();
  display.clearDisplay();
}
void loop() {
  int avg_temp = 0;
  int count = 0;
  while(1){
    int temperature = dht11.readTemperature();
    if (temperature != DHT11::ERROR_CHECKSUM && temperature != DHT11::ERROR_TIMEOUT)
    {
        display.setCursor(0,0);
        display.setRotation(2);
        display.clearDisplay();
        display.setTextSize(2);
        display.setTextColor(BLACK);
        display.print("Sample: "); display.println(count, DEC);
        display.setCursor(0,18);
        display.print("Temperature: ");
        display.print(temperature);
        display.println(" C");
        display.setCursor(0,50);
        display.println("Avg Temp: ");
        avg_temp = ((avg_temp * count) + temperature) / (count+1);
        count++;
        display.print(avg_temp);
        display.println(" C");
    }
    else
    {
        Serial.println(DHT11::getErrorString(temperature));
    }
    for(int j=0; j<4; j++) {
      display.refresh();
      delay(500); // 1/2 sec delay
    }
  }
```


Georgia Tech

BootHammer at a Glance

…After

UserStudy_SmartGarden.ino

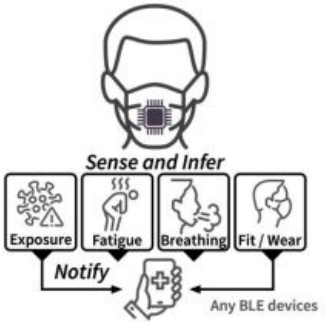
```
#include <DHT11.h>
#include <framstackAdaSPI.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SharpMem.h>
#define SHARP_SCK  30
#define SHARP_MOSI 29
#define SHARP_SS   1
Adafruit_SharpMem display(SHARP_SCK, SHARP_MOSI, SHARP_SS, 144, 168);
#define BLACK 0
#define WHITE 1
dDHT11 dht11(A10);
void setup() {
  Serial.begin(9600);
  delay(1000);
  display.begin();
  display.clearDisplay();
  getDeviceID();
  RESTORE
}
void loop() {
  int avg_temp = 0;
  int count = 0;
  while(1){
    int temperature = dht11.readTemperature();
    if (temperature != DHT11::ERROR_CHECKSUM && temperature != DHT11::ERROR_TIMEOUT)
    {
        display.setCursor(0,0);
        display.setRotation(2);
        display.clearDisplay();
        display.setTextSize(2);
        display.setTextColor(BLACK);
        display.print("Sample: "); display.println(count, DEC);
        display.setCursor(0,18);
        display.print("Temperature: ");
        display.print(temperature);
        display.println(" C");
        display.setCursor(0,50);
        display.println("Avg Temp: ");
        avg_temp = ((avg_temp * count) + temperature) / (count+1);
        count++;
        display.print(avg_temp);
        display.println(" C");
    }
    else
    {
        Serial.println(DHT11::getErrorString(temperature));
    }
    for(int j=0; j<4; j++) {
      display.refresh();
      delay(500); // 1/2 sec delay
    } // x4 = 2 second pause between rotations
    CHECKPOINT
  }

}
```


Georgia Tech

7th Generation Decision Making

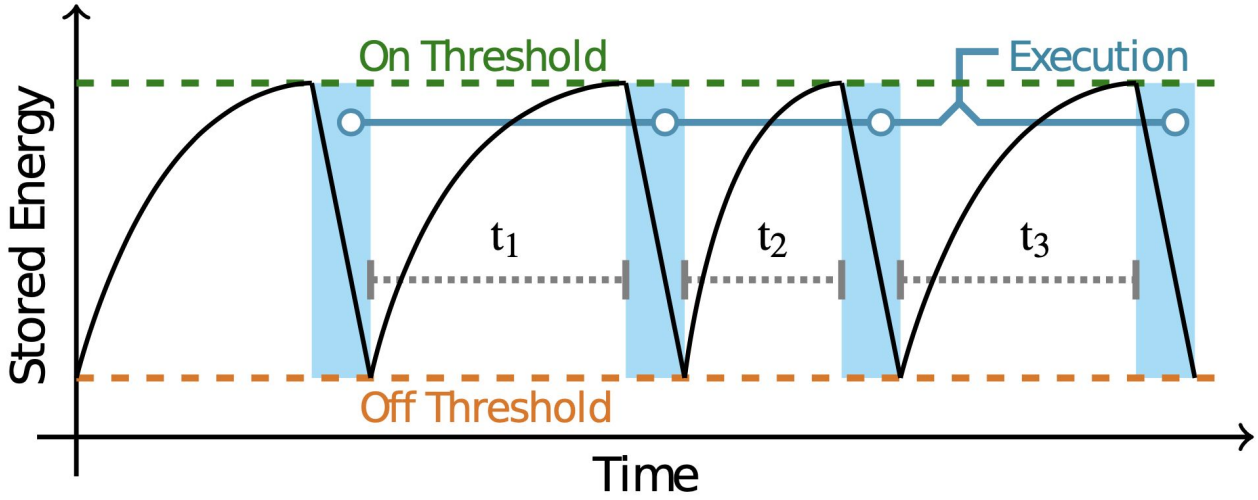
Build computer **systems** that are sustainable, resilient, and useful for the entire lifetime of the people, places, and things they support.



Ka Moamoa

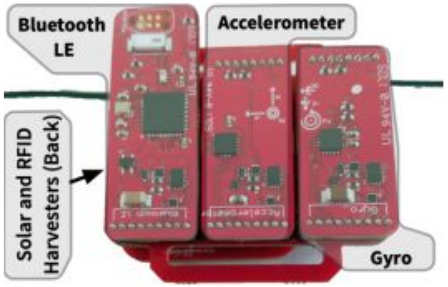
Georgia Tech

Energy Harvesting/Intermittent Computing



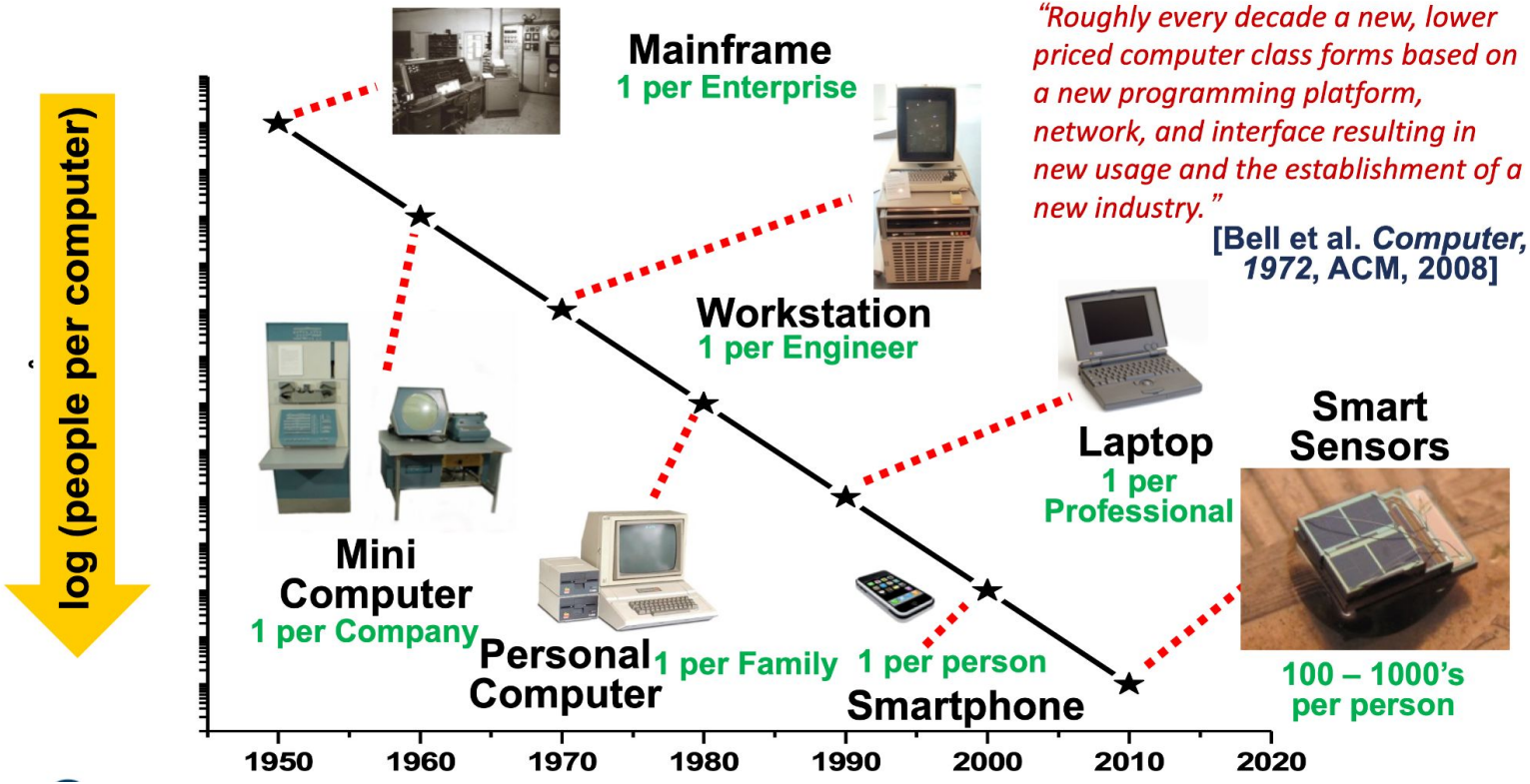
A Program's execution must be stitched together across power failures because the power coming from energy harvesters is low and dynamic

Sustainable Computational “Things”



Georgia Tech

Bell's Law of Computer Classes:
A new computer class emerges roughly every decade



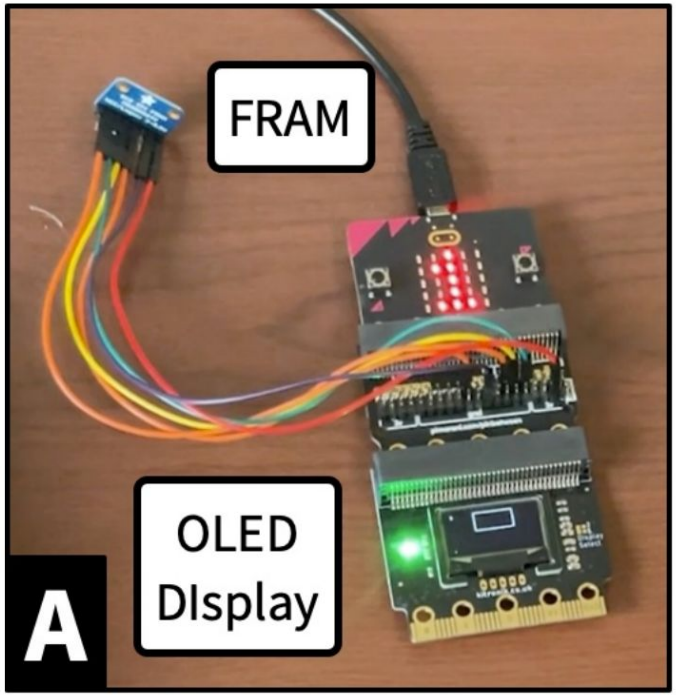
"Roughly every decade a new, lower priced computer class forms based on a new programming platform, network, and interface resulting in new usage and the establishment of a new industry."

[Bell et al. *Computer, 1972*, ACM, 2008]

Georgia Tech


EVEREADY
Energizer
SUPER
SONY
Panasonic
PHILIPS
Kodak
DURACELL
ULTRA
High-Tech
ALCALINE
HIGH ENERGY
Powerking
RAYOVAC
EXTRA DURATION

Previous Work: Battery-free MakeCode




```
1 let count = 0
2 fram.init()
3 while(count < 10){
4     count++
5 }
```

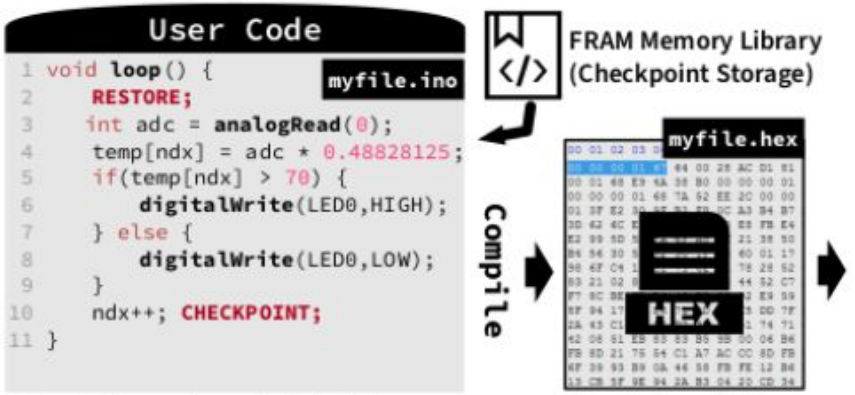
→

Compiler Pass
In-lining
State Analysis
Instruction Building
Checkpoint Insertion

Leverage the compiler to transform code to work battery-free

User Instrumentation

RESTORE and **CHECKPOINT** macros let the user decide where to checkpoint their code. The macros are placeholders made of BKPT and NOP instructions. These placeholders make space for 32-bit branch instructions



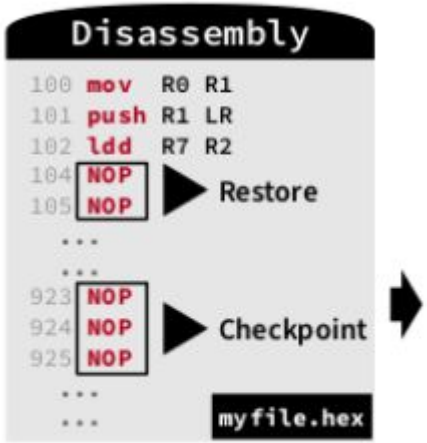
Step 1: User writes code in Arduino, (optionally) annotates with `restore` and `checkpoint` locations. Clicks the "compile/verify" button.

Step 2: Compilation happens as normal.

Georgia Tech

Assembly Analysis

- BootHammer's "trick"
  - Arduino output's it's code in a very predictable pattern
  - They all follow the setup() and loop() model
    - The program won't compile without these functions!
  - So, there must be branches to setup() and loop() from main()
  - Exploit this pattern to rediscover higher level information about the code
    - If I can find main(), I can find setup() and loop()
    - If I can find setup() and loop(), I can find all of the user's code



Step 3: Hex file disassembled to ease analysis for checkpoint insertion.

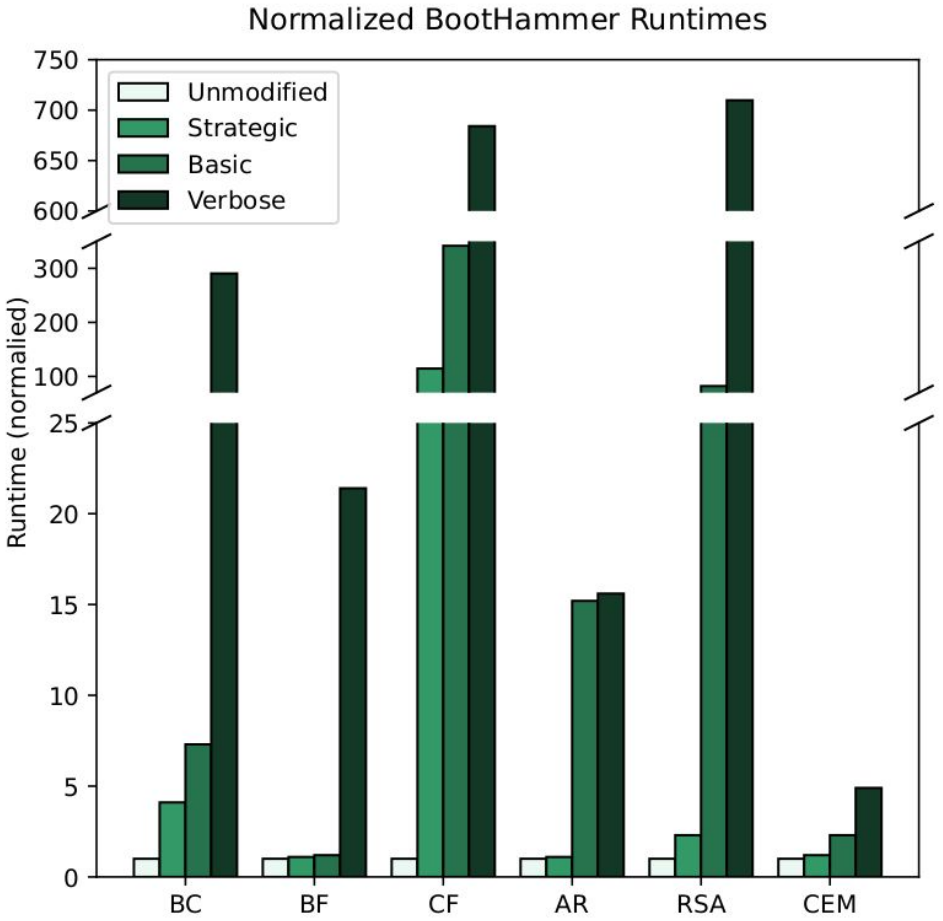
BootHammer Runtimes

Verbose: Checkpoint every function and every iteration of a loop

Basic: Checkpoint every function. Skip some iterations of a loop

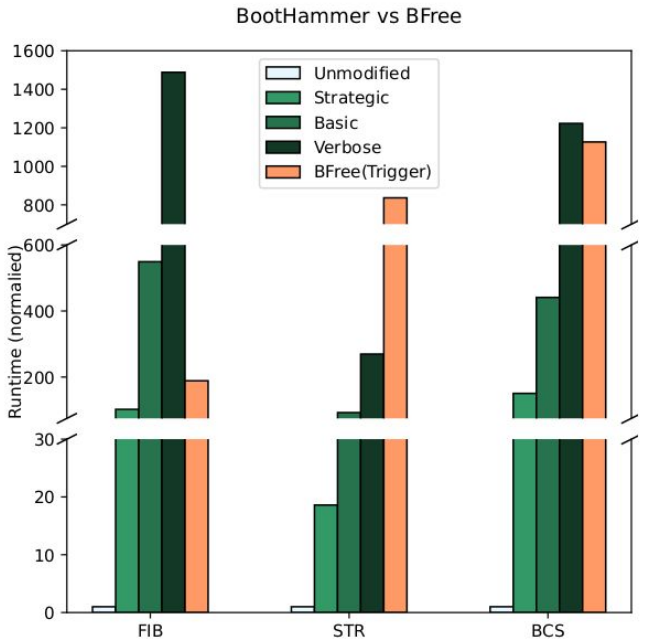
Strategic: Smart placement of checkpoints*

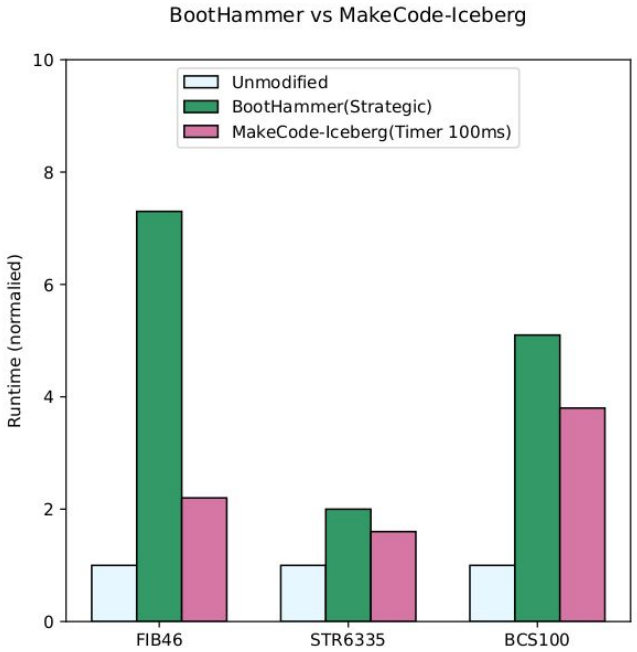
Unmodified: Native code



Georgia Tech

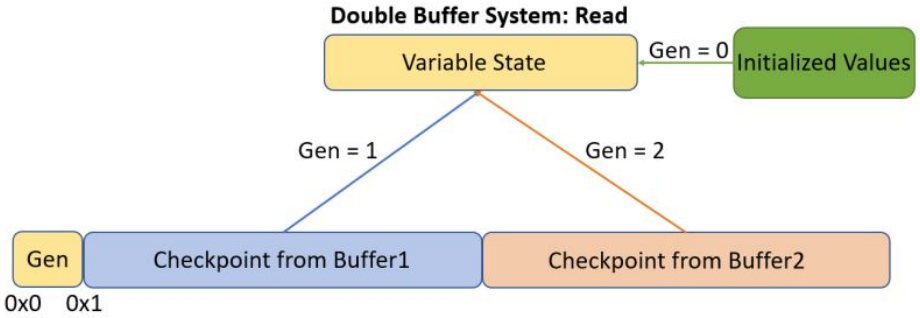
BootHammer VS MakeCode-Iceberg and BFree



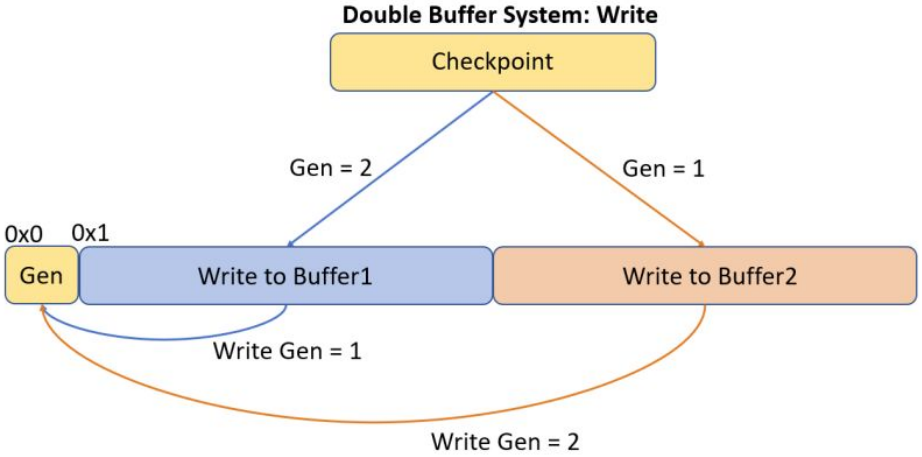


Georgia Tech

Double Buffering the Checkpoints in Non-volatile Storage



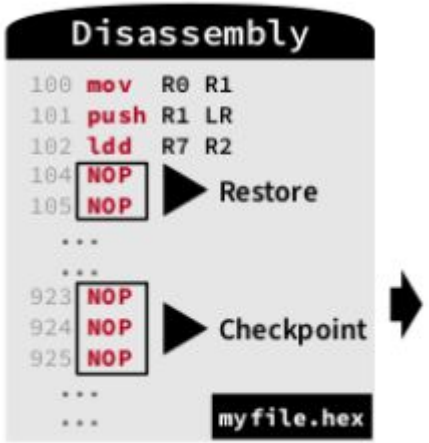
(a) Read operation for the double buffer system



(b) Write operation for the double buffer system

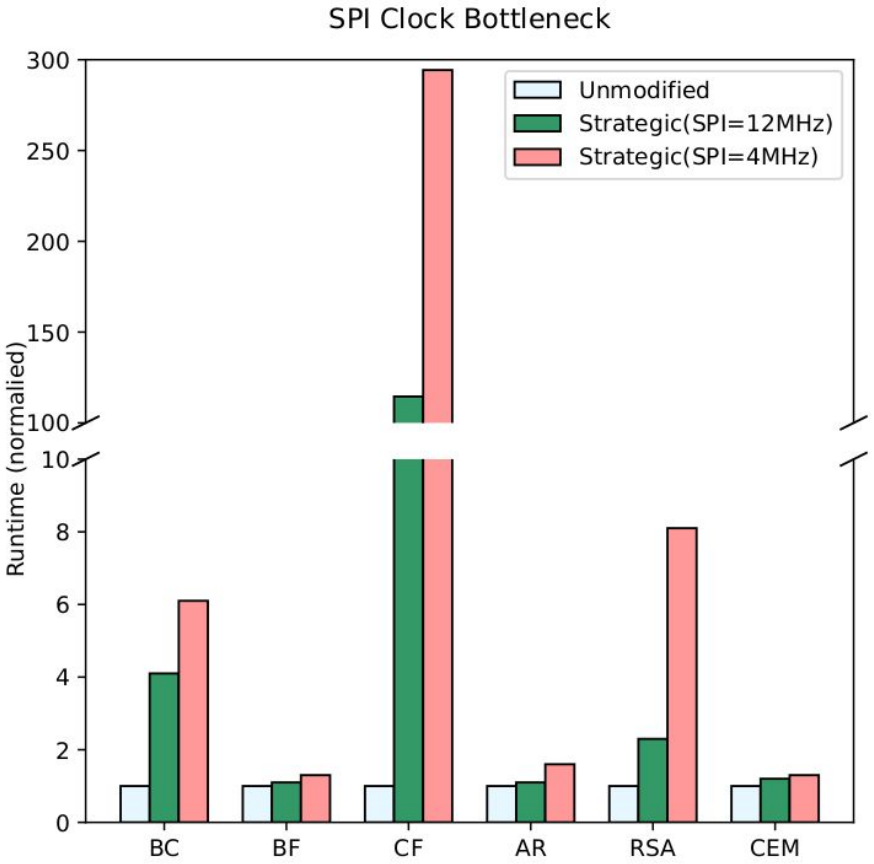
Disassembly

- Modeled after objdump disassembly
  - From a high level:
  - Parse each line of the hex file(ihex format)
  - Disassemble the line one halfword at a time, taking into account the few 32 bit instructions
  - State machine, moving left to right through the halfword
    - Identify registers, immediates, and other values used
    - Build branches and identify destination addresses
    - Build a list of Instruction objects made of these characteristics



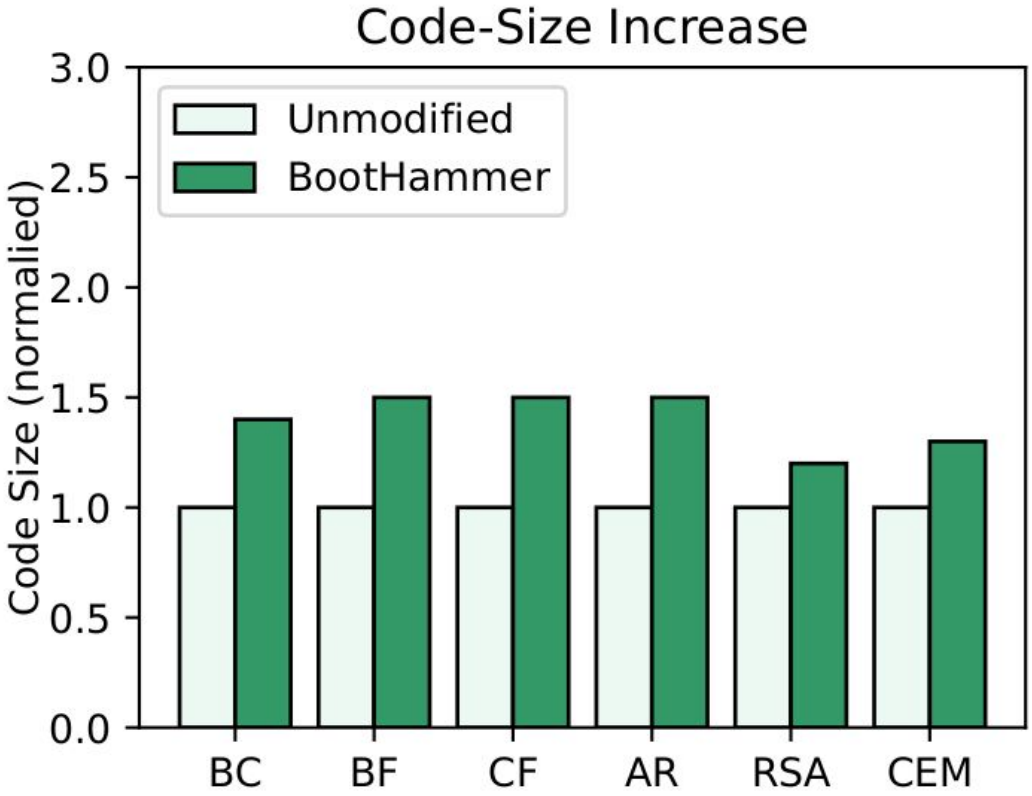
Step 3: Hex file disassembled to ease analysis for checkpoint insertion.

SPI Clock(BootHammer)



Georgia Tech

Code Size Increase



Georgia Tech

Runtime

Benchmark	BOOTHAMMER Rewriting Time(s)
BC	0.0094
BF	0.0106
CF	0.0098
AR	0.013
RSA	0.0102
CEM	0.0278

BX LR Problem

Not all functions push variables to the stack

CHECKPOINT is a NOP and 2 BKPT instructions

The NOP is replaced with a push(), the BKPTs are replaced with a branch to the trampoline, and BX LR is replaced with a pop()