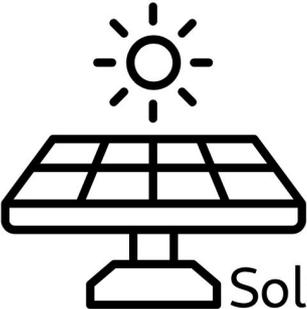


# User-directed Assembly Code Transformations Enabling Efficient Batteryless Arduino Applications

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

[ckraemer7@gatech.edu](mailto:ckraemer7@gatech.edu)

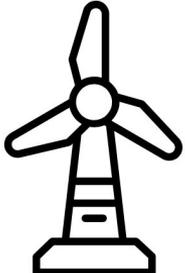
# No Batteries = Energy Harvesting



Solar



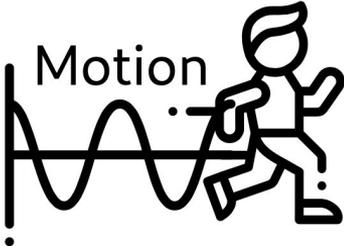
Radio Waves



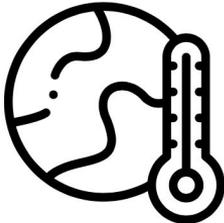
Wind



Interaction

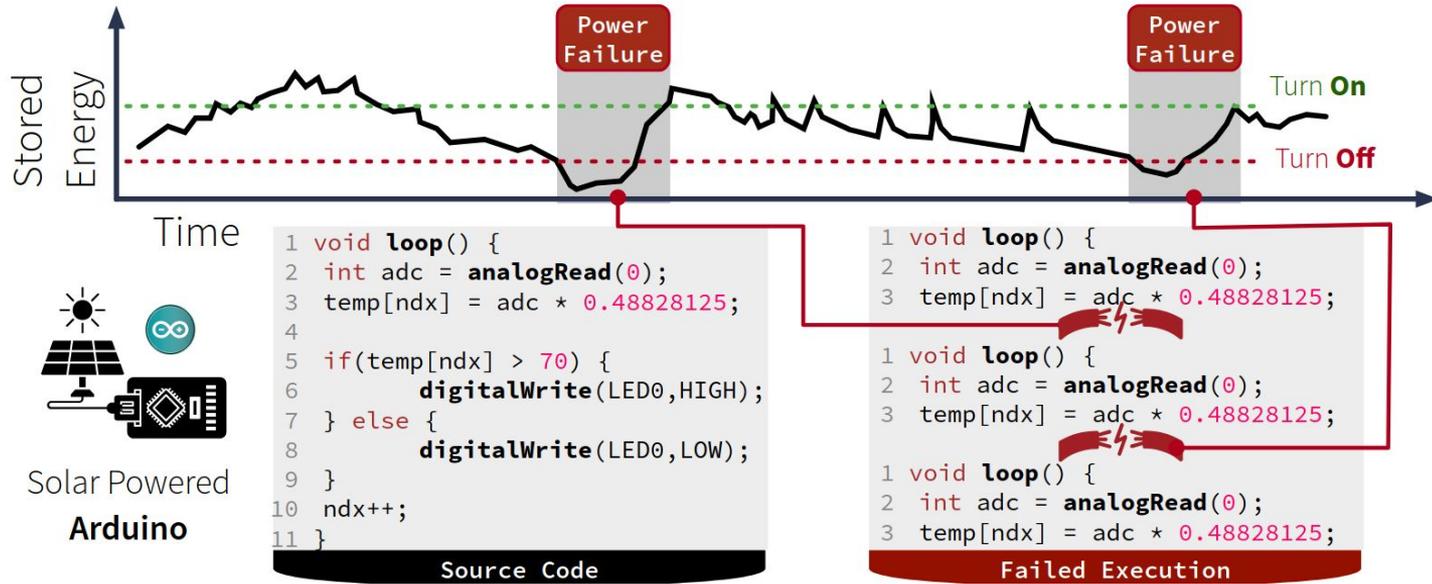


Motion



Thermal

# 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

# Targeting the Maker Ecosystem

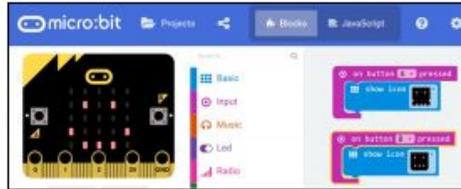
## Arduino (est. 2005)

```
Arduino IDE / Arduino 1.8.12
Arduino
28 //
29
30 int sensorPin = A0; // select the input pin for the p
31 int ledPin = 13; // select the pin for the LED
32 int sensorValue = 0; // variable to store the value con
33
34 void setup() {
35   // declare the ledPin as an OUTPUT:
36   pinMode(ledPin, OUTPUT);
37 }
38
39 void loop() {
40   // read the value from the sensor:
41   sensorValue = analogRead(sensorPin);
42   // turn the ledPin on
43   digitalWrite(ledPin, HIGH);
44   // stop the program for  $\approx$  milliseconds:
45   delay(sensorValue);
46   // turn the ledPin off:
47   digitalWrite(ledPin, LOW);
48   // stop the program for  $\approx$  milliseconds:
49   delay(1000);
50 }
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

Compiled (C/C++)



## Makecode (est. 2017)



Source to Source Translation  
(Blocks/JS/Python)



## CircuitPython (est. 2017)

```
1 import board
2 import digitalio
3 import time
4
5 led = digitalio.DigitalInOut(board.LED)
6 led.direction = digitalio.Direction.OUTPUT
7
8 while True:
9     led.value = True
10    time.sleep(0.5)
11    led.value = False
12    time.sleep(0.5)
13
```

Interpreted On-Device (Python)  
Virtualized Instructions



Which gives us  
the largest  
potential  
userbase?

->Arduino

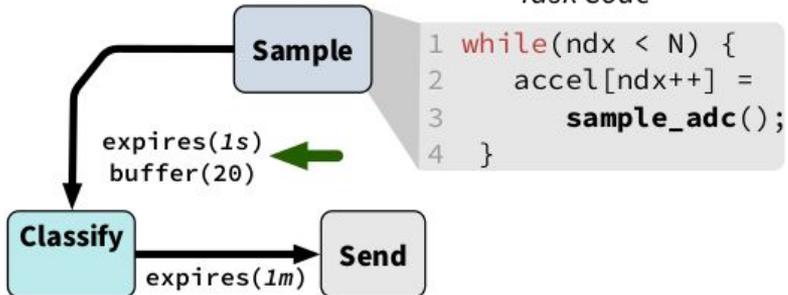
# Existing Infrastructure(Related Work)

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

## Task-graph behavior defining annotations

*Mayfly, InK, CatNap*

*Task Code*



```
1 while(ndx < N) {  
2   accel[ndx++] =  
3     sample_adc();  
4 }
```

## 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() ←
```

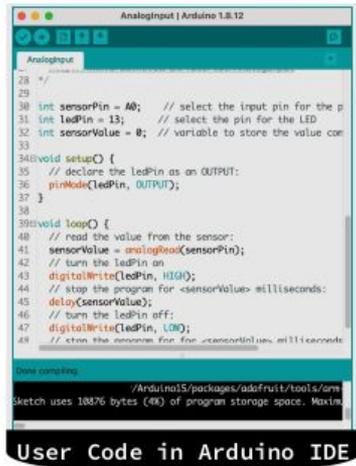
## Behavior Defining Annotations

*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 }
```

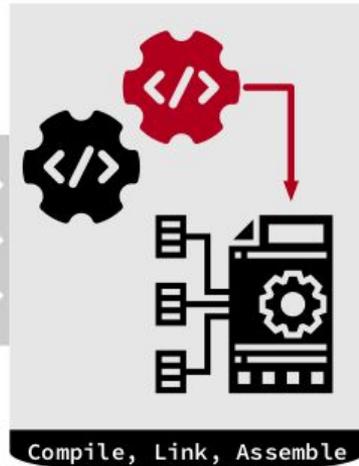
# This Work: BootHammer

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



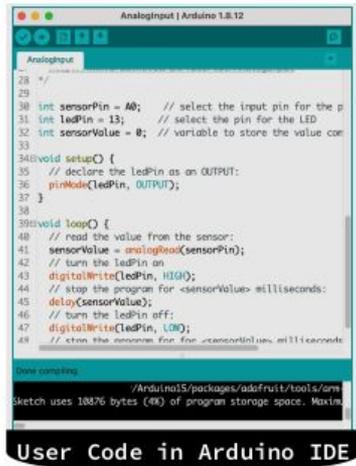
```
28 //
29 //
30 int sensorPin = A0; // select the input pin for the p
31 int ledPin = 13; // select the pin for the LED
32 int sensorValue = 0; // variable to store the value com
33
34 void setup() {
35 // declare the ledPin as an OUTPUT:
36 pinMode(ledPin, OUTPUT);
37 }
38
39 void loop() {
40 // read the value from the sensor:
41 sensorValue = analogRead(sensorPin);
42 // turn the ledPin on
43 digitalWrite(ledPin, HIGH);
44 // stop the program for <sensorValue> milliseconds:
45 delay(sensorValue);
46 // turn the ledPin off:
47 digitalWrite(ledPin, LOW);
48 // then the sensor for the <sensorInterval> milliseconds
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

User Code in Arduino IDE



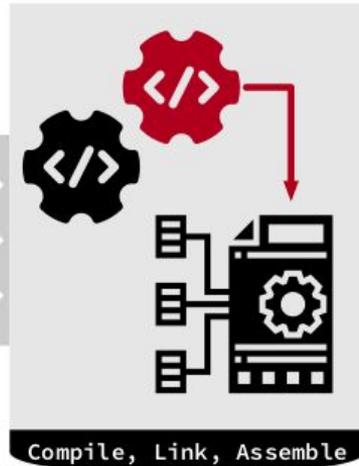
# This Work: BootHammer

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



```
28 //
29 //
30 int sensorPin = A0; // select the input pin for the p
31 int ledPin = 13; // select the pin for the LED
32 int sensorValue = 0; // variable to store the value com
33
34 void setup() {
35 // declare the ledPin as an OUTPUT:
36 pinMode(ledPin, OUTPUT);
37 }
38
39 void loop() {
40 // read the value from the sensor:
41 sensorValue = analogRead(sensorPin);
42 // turn the ledPin on
43 digitalWrite(ledPin, HIGH);
44 // stop the program for sensorValue milliseconds:
45 delay(sensorValue);
46 // turn the ledPin off:
47 digitalWrite(ledPin, LOW);
48 // turn the sensor on for sensorValue milliseconds:
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

User Code in Arduino IDE



**Boot  
Hammer**

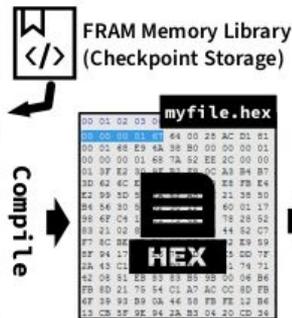
Low-level code transformations  
via  
binary rewriting



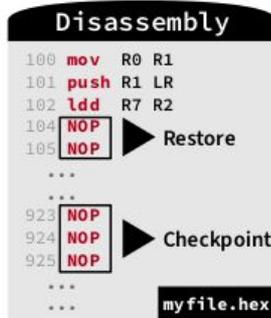
# BootHammer Pipeline

```
1 void loop() {
2   RESTORE;
3   int adc = analogRead(0);
4   temp[ndx] = adc * 0.48828125;
5   if(temp[ndx] > 70) {
6     digitalWrite(LED0,HIGH);
7   } else {
8     digitalWrite(LED0,LOW);
9   }
10  ndx++; CHECKPOINT;
11 }
```

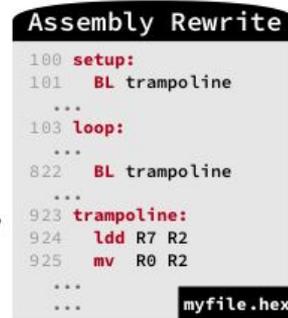
**Step 1:** User writes code in Arduino, (optionally) annotates with **restore** and **checkpoint** locations. Clicks the "compile/verify" button.



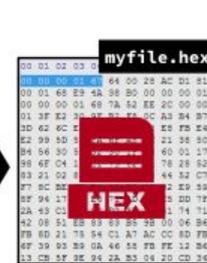
**Step 2:** Compilation happens as normal.



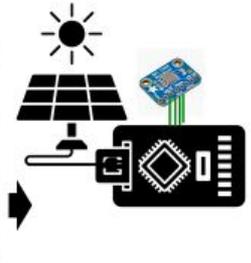
**Step 3:** Hex file disassembled to ease analysis for checkpoint insertion.



**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!

# BootHammer Pipeline: User Code Instrumentation

## User Code

```
1 void loop() { myfile.ino
2   RESTORE;
3   int adc = analogRead(0);
4   temp[ndx] = adc * 0.48828125;
5   if(temp[ndx] > 70) {
6     digitalWrite(LED0,HIGH);
7   } else {
8     digitalWrite(LED0,LOW);
9   }
10  ndx++; CHECKPOINT;
11 }
```

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



FRAM Memory Library  
(Checkpoint Storage)

Compile



**Step 2:** Compilation happens as normal.

## Disassembly

```
100 mov R0 R1
101 push R1 LR
102 ldd R7 R2
...
104 NOP
105 NOP
...
923 NOP
924 NOP
925 NOP
...
myfile.hex
```

Restore

Checkpoint

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

## Assembly Rewrite

```
100 setup:
101 BL trampoline
...
103 loop:
...
822 BL trampoline
...
923 trampoline:
924 ldd R7 R2
925 mv R0 R2
...
myfile.hex
```

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

# BootHammer Pipeline: Compilation

## User Code

```
op() {  
  // myfile.ino  
  // CORE;  
  adc = analogRead(0);  
  p[ndx] = adc * 0.48828125;  
  if (temp[ndx] > 70) {  
    digitalWrite(LED0,HIGH);  
  } else {  
    digitalWrite(LED0,LOW);  
  }  
  ++; CHECKPOINT;  
}
```

myfile.ino



FRAM Memory Library  
(Checkpoint Storage)

Compile

myfile.hex

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	00	00	01	67	64	00	28	AC	D1	81					
00	01	68	E9	4A	38	B0	00	00	00	01					
00	00	00	01	68	7A	52	EE	2C	00	00					
01	3F	E2	30	8F	B7	F8	0C	A3	B4	B7					
3D	62	6C	E					E8	FB	E4					
E2	99	5D	5					21	38	50					
B4	56	30	5					60	01	17					
98	6F	C4	1					78	28	52					
83	21	02	8					44	52	C7					
E7	8C	BE						42	E9	59					
8F	94	17						5	DD	7F					
2A	43	C1						1	74	71					
42	08	51	EB	83	83	B5	9B	00	06	B6					
FB	8D	21	75	54	C1	A7	AC	CC	8D	FB					
6F	39	93	B9	0A	46	58	FB	FE	12	B6					
13	CB	5F	9E	94	2A	B3	04	20	CD	34					

HEX

Step 2: Compilation happens as normal.

## Disassembly

```
100 mov R0 R1  
101 push R1 LR  
102 ldd R7 R2  
104 NOP  
105 NOP  
...  
...  
923 NOP  
924 NOP  
925 NOP  
...  
...  
myfile.hex
```

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

## Assembly Rewrite

```
100 setup:  
101 BL trampoline  
...  
103 loop:  
...  
822 BL trampoline  
...  
...  
923 trampoline:  
924 ldd R7 R2  
925 mv R0 R2  
...  
...  
myfile.hex
```

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

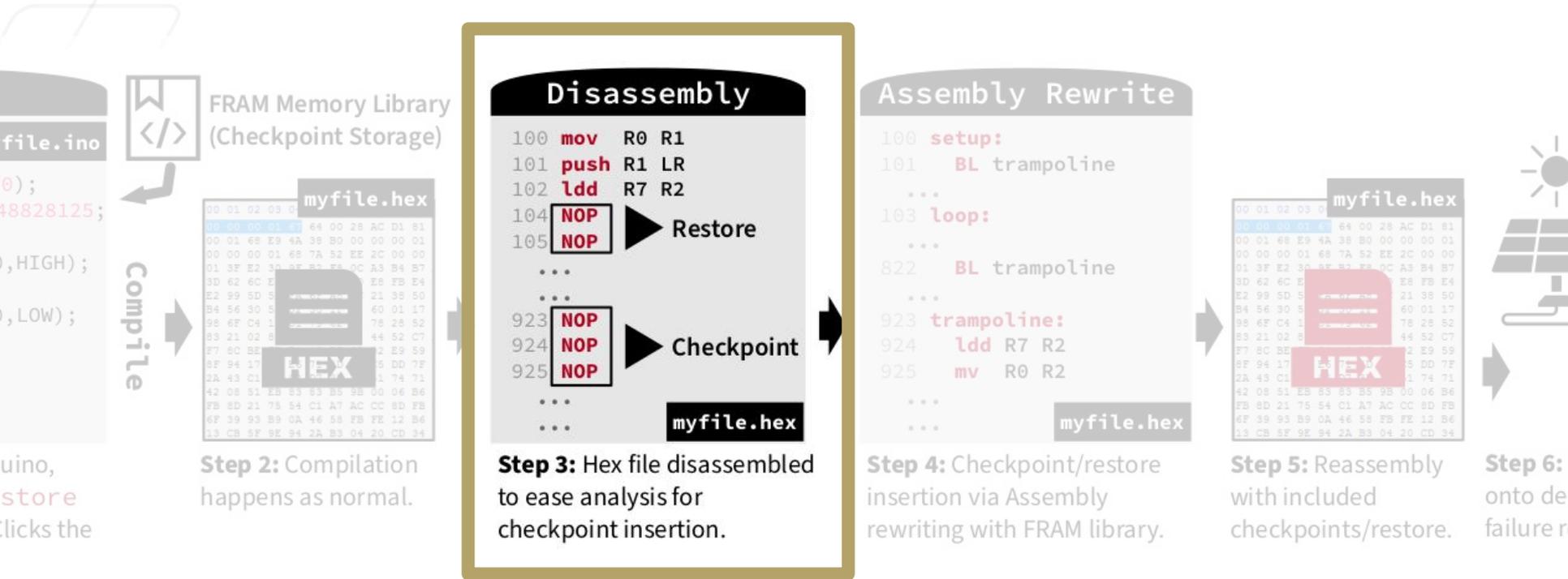
myfile.hex

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	00	00	01	67	64	00	28	AC	D1	81					
00	01	68	E9	4A	38	B0	00	00	00	01					
00	00	00	01	68	7A	52	EE	2C	00	00					
01	3F	E2	30	8F	B7	F8	0C	A3	B4	B7					
3D	62	6C	E					E8	FB	E4					
E2	99	5D	5					21	38	50					
B4	56	30	5					60	01	17					
98	6F	C4	1					78	28	52					
83	21	02	8					44	52	C7					
E7	8C	BE						42	E9	59					
8F	94	17						5	DD	7F					
2A	43	C1						1	74	71					
42	08	51	EB	83	83	B5	9B	00	06	B6					
FB	8D	21	75	54	C1	A7	AC	CC	8D	FB					
6F	39	93	B9	0A	46	58	FB	FE	12	B6					
13	CB	5F	9E	94	2A	B3	04	20	CD	34					

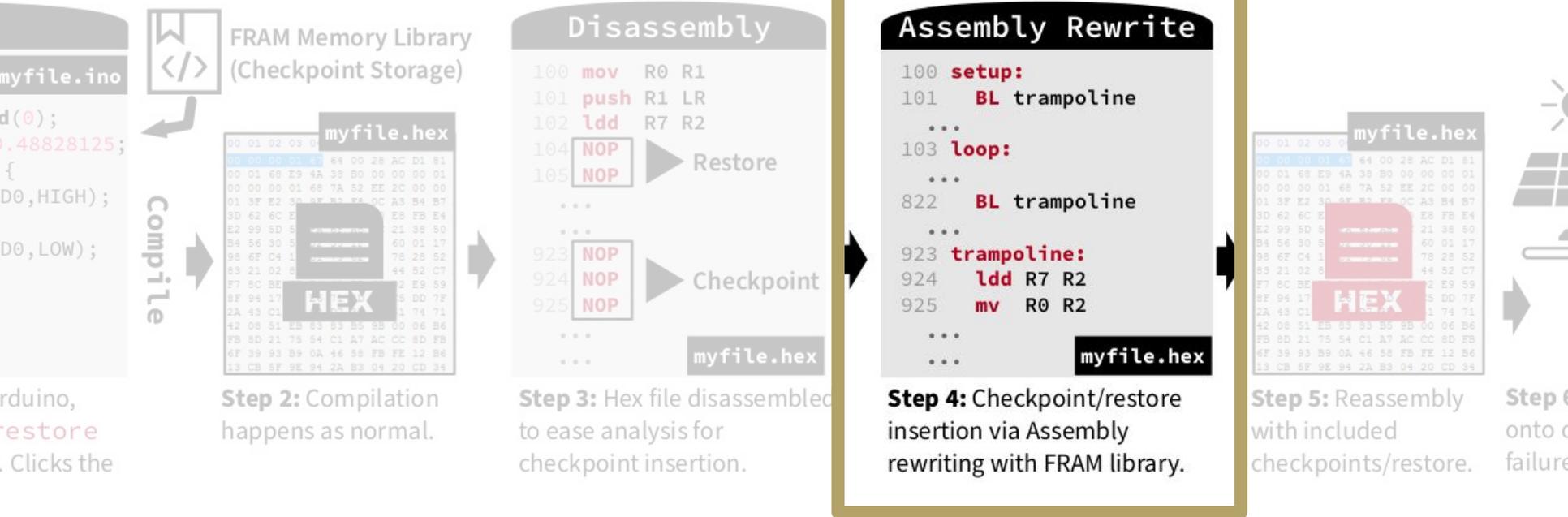
HEX

Step 5: Ready for verification with include checkpoints.

# BootHammer Pipeline: Disassembly



# BootHammer Pipeline: Assembly Transformation



# BootHammer Pipeline: Reassembly and Flash

FRAM Memory Library  
Checkpoint Storage)



Step 2: Compilation  
appears as normal.

### Disassembly

```
100 mov R0 R1
101 push R1 LR
102 ldd R7 R2
...
104 NOP
105 NOP
...
923 NOP
924 NOP
925 NOP
...
myfile.hex
```

Restore

Checkpoint

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

### Assembly Rewrite

```
100 setup:
101 BL trampoline
...
103 loop:
...
822 BL trampoline
...
923 trampoline:
924 ldd R7 R2
925 mv R0 R2
...
myfile.hex
```

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

### myfile.hex

```
00 01 02 03 04
00 00 00 01 67 64 00 28 AC D1 81
00 01 68 E9 4A 38 B0 00 00 00 01
00 00 00 01 68 7A 52 EE 2C 00 00
01 3F E2 30 8F B2 FC A3 B4 B7
3D 62 6C E E8 FB E4
E2 99 5D 5 21 38 50
B4 56 30 5 60 01 17
98 6F C4 1 78 28 52
83 21 02 8 44 52 C7
E7 8C 8E 2 E9 59
8F 94 17 3 DD 7F
2A 43 C1 1 74 71
42 08 51 EB 83 83 B5 9B 00 06 B6
FB 8D 21 75 54 C1 A7 AC CC 8D FB
6F 39 93 B9 0A 46 58 FB FE 12 B6
13 CB 5F 9E 94 2A B3 04 20 CD 34
```

Step 5: Reassembly  
with included  
checkpoints/restore.

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

# BootHammer Evaluation

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

BC-Bitcount

BF-Blowfish

CF-Cuckoo Filtering

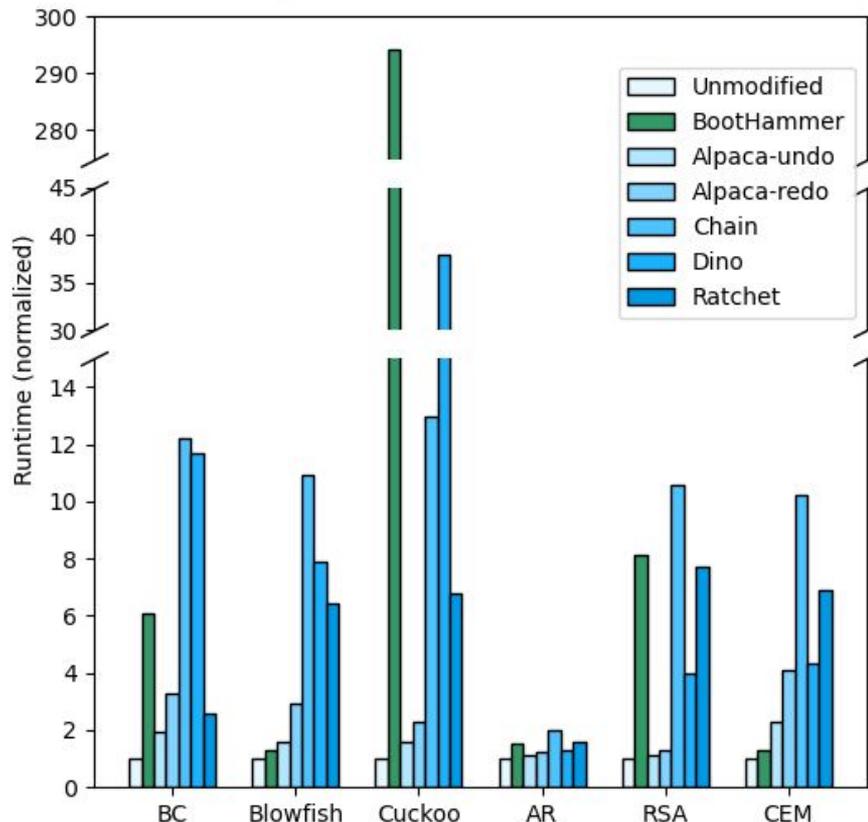
AR-Activity Recognition

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

Strategic BootHammer vs State-of-the-Art



# 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

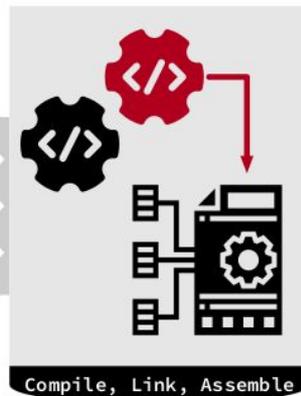
# BootHammer and the Future Ahead

- 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



```
28 //
29
30 int sensorPin = A0; // select the input pin for the p
31 int ledPin = 13; // select the pin for the LED
32 int sensorValue = 0; // variable to store the value ca
33
34 void setup() {
35 // declare the ledPin as an OUTPUT:
36 pinMode(ledPin, OUTPUT);
37 }
38
39 void loop() {
40 // read the value from the sensor:
41 sensorValue = analogRead(sensorPin);
42 // turn the ledPin on
43 digitalWrite(ledPin, HIGH);
44 // stop the program for <sensorValue> milliseconds:
45 delay(sensorValue);
46 // turn the ledPin off:
47 digitalWrite(ledPin, LOW);
48 // stop the program for <sensorValue> milliseconds:
49 delay(sensorValue);
50 }
51
52 // Arduino IDE
53 Sketch uses 10876 bytes (40% of program storage space. Max is 262144 bytes.)
```

User Code in Arduino IDE



# 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

```
1 #include <DHT11.h>
2 #include <Adafruit_GFX.h>
3 #include <Adafruit_ShpMem.h>
4 #define SHARP_SCK 30
5 #define SHARP_MOSI 29
6 #define SHARP_SS 1
7 Adafruit_ShpMem display(SHARP_SCK, SHARP_MOSI, SHARP_SS, 144, 168);
8 #define BLACK 0
9 #define WHITE 1
10 int minorHalfSize; // 1/2 of lesser of display width or height
11 DHT11 dht11(A10);
12 void setup() {
13   Serial.begin(9600);
14   delay(1000);
15   display.begin();
16   display.clearDisplay();
17 }
18 void loop() {
19   int avg_temp = 0;
20   int count = 0;
21   while(1){
22     int temperature = dht11.readTemperature();
23     if (temperature != DHT11::ERROR_CHECKSUM && temperature != DHT11::ERROR_TIMEOUT)
24     {
25       display.setCursor(0,0);
26       display.setRotation(2);
27       display.clearDisplay();
28       display.setTextSize(2);
29       display.setTextColor(BLACK);
30       display.print("Sample: "); display.println(count, DEC);
31       display.setCursor(0,18);
32       display.print("Temperature: ");
33       display.print(temperature);
34       display.println(" C");
35       display.setCursor(0,50);
36       display.println("Avg Temp: ");
37       avg_temp = ((avg_temp * count) + temperature) / (count+1);
38       count++;
39       display.print(avg_temp);
40       display.println(" C");
41     }
42     else
43     {
44       Serial.println(DHT11::getErrorString(temperature));
45     }
46     for(int j=0; j<4; j++) {
47       display.refresh();
48       delay(500); // 1/2 sec delay
49     }
50 }
```

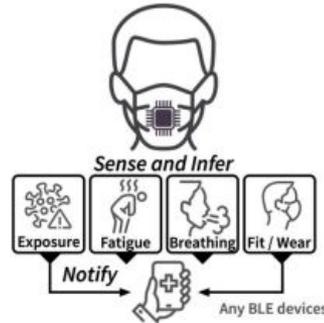
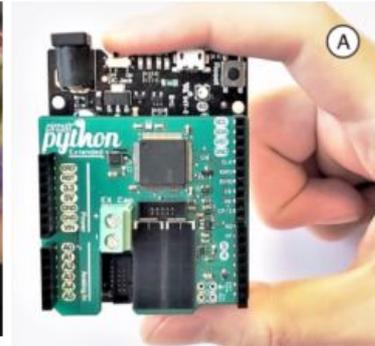
# BootHammer at a Glance

...After

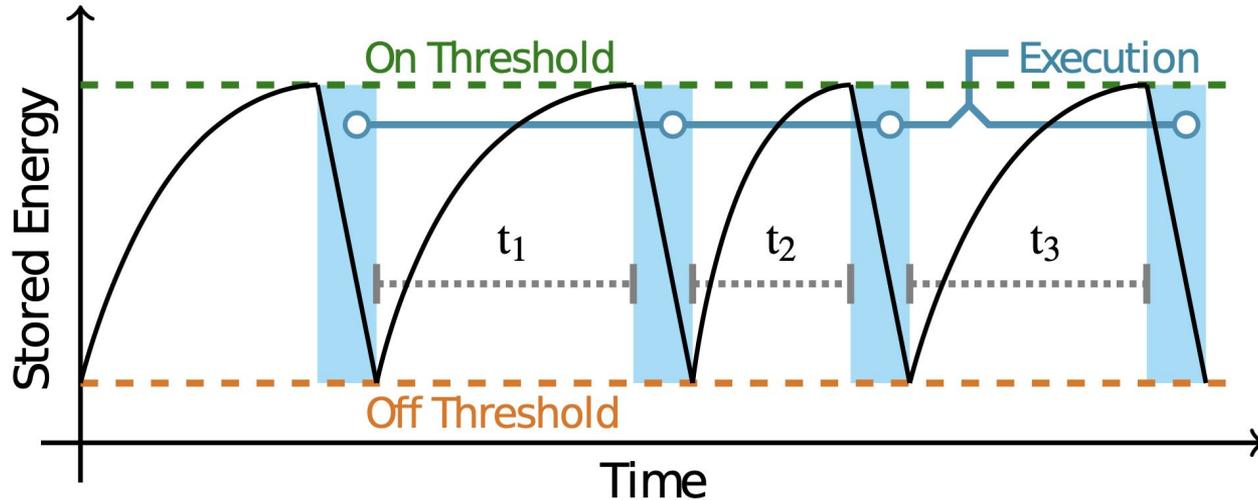
```
UserStudy_SmartGarden.ino
1 #include <DHT11.h>
2 #include <framstackAdaSPI.h>
3 #include <Adafruit_GFX.h>
4 #include <Adafruit_SSharpMem.h>
5 #define SHARP_SCK 30
6 #define SHARP_MOSI 29
7 #define SHARP_SS 1
8 Adafruit_SSharpMem display(SHARP_SCK, SHARP_MOSI, SHARP_SS, 144, 168);
9 #define BLACK 0
10 #define WHITE 1
11 dDHT11 dht11(A10);
12 void setup() {
13     Serial.begin(9600);
14     delay(1000);
15     display.begin();
16     display.clearDisplay();
17     getDeviceID();
18     RESTORE
19 }
20 void loop() {
21     int avg_temp = 0;
22     int count = 0;
23     while(1){
24         int temperature = dht11.readTemperature();
25         if (temperature != DHT11::ERROR_CHECKSUM && temperature != DHT11::ERROR_TIMEOUT)
26         {
27             display.setCursor(0,0);
28             display.setRotation(2);
29             display.clearDisplay();
30             display.setTextSize(2);
31             display.setTextColor(BLACK);
32             display.print("Sample: "); display.println(count, DEC);
33             display.setCursor(0,18);
34             display.print("Temperature: ");
35             display.print(temperature);
36             display.println(" C");
37             display.setCursor(0,50);
38             display.println("Avg Temp: ");
39             avg_temp = ((avg_temp * count) + temperature) / (count+1);
40             count++;
41             display.print(avg_temp);
42             display.println(" C");
43         }
44         else
45         {
46             Serial.println(DHT11::getErrorString(temperature));
47         }
48         for(int j=0; j<4; j++) {
49             display.refresh();
50             delay(500); // 1/2 sec delay
51             // x4 = 2 second pause between rotations
52             CHECKPOINT
53         }
54     }
55 }
56
```

# 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.

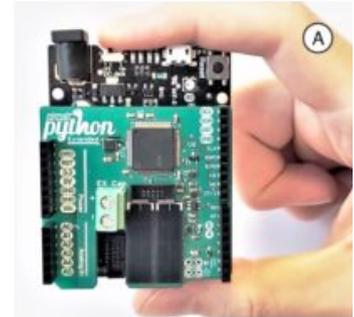
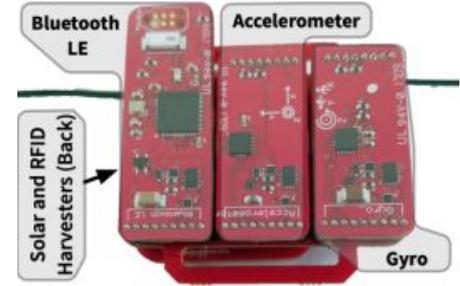


# Energy Harvesting/Intermittent Computing

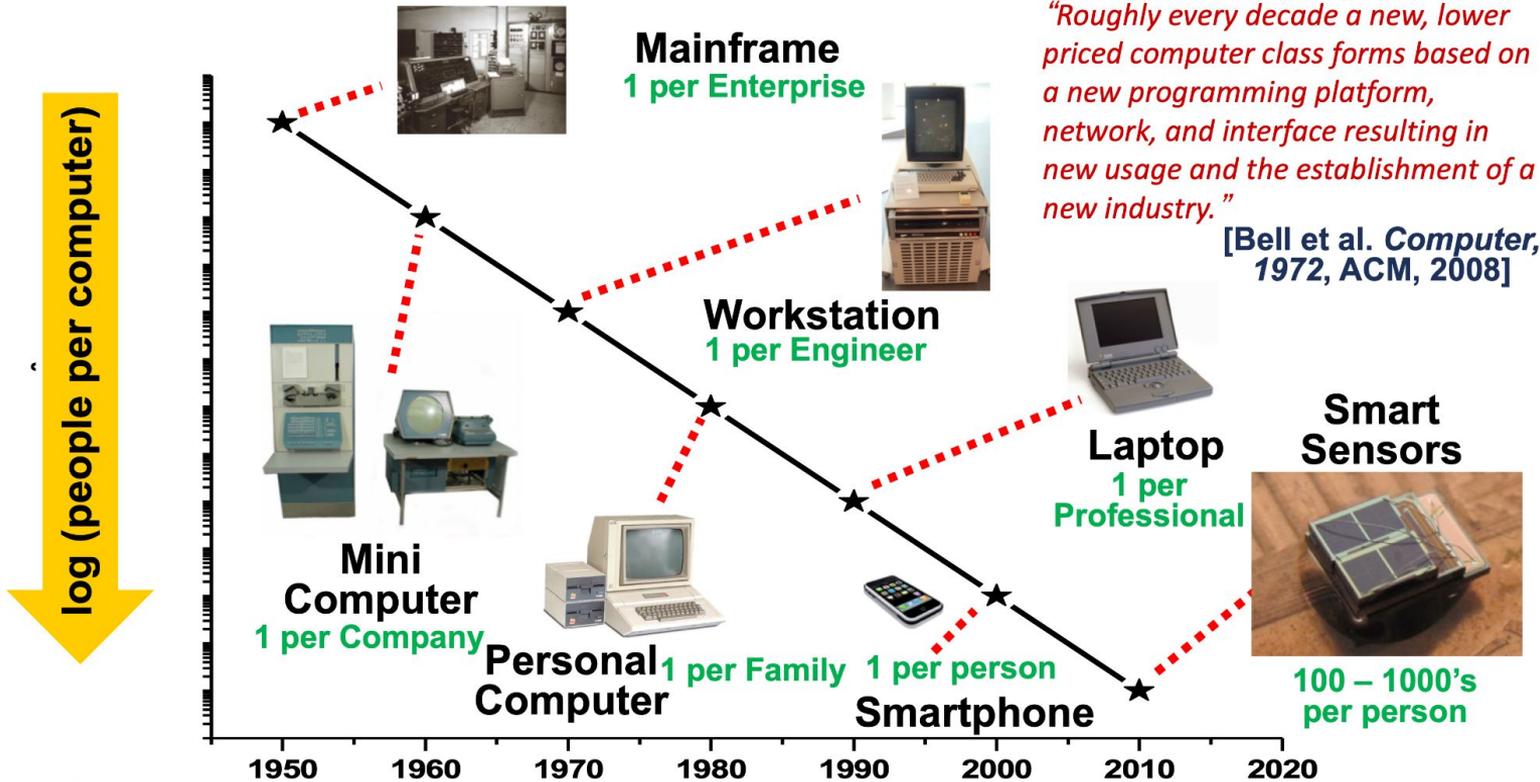


A Program's execution must be stitched together across power failures because the power coming from energy harvesters is low and dynamic

# Sustainable Computational “Things”

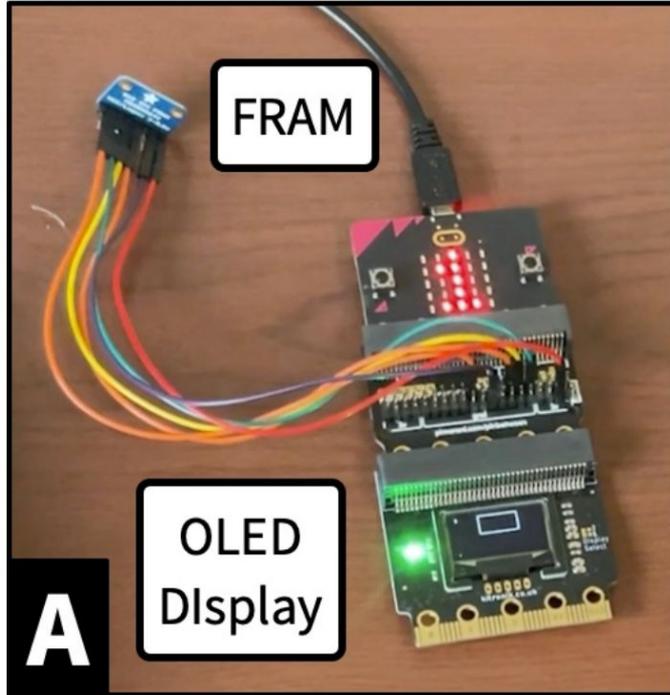


# Bell's Law of Computer Classes: A new computer class emerges roughly every decade

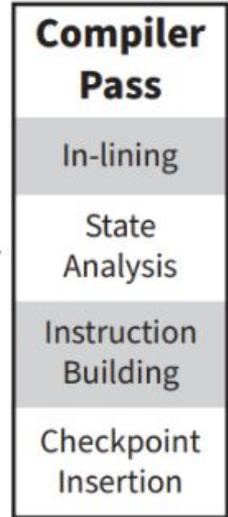




# Previous Work: Battery-free MakeCode



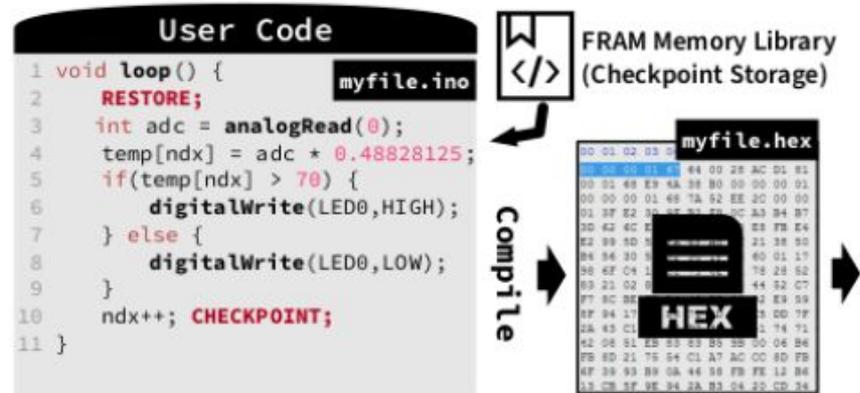
```
1 let count = 0
2 fram.init()
3 while(count < 10){
4     count++
5 }
```



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.

# Assembly Analysis

- BootHammer's "trick"
  - Arduino outputs its 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

```
Disassembly
100 mov R0 R1
101 push R1 LR
102 ldd R7 R2
104 NOP
105 NOP
...
923 NOP
924 NOP
925 NOP
...
my file.hex
```

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

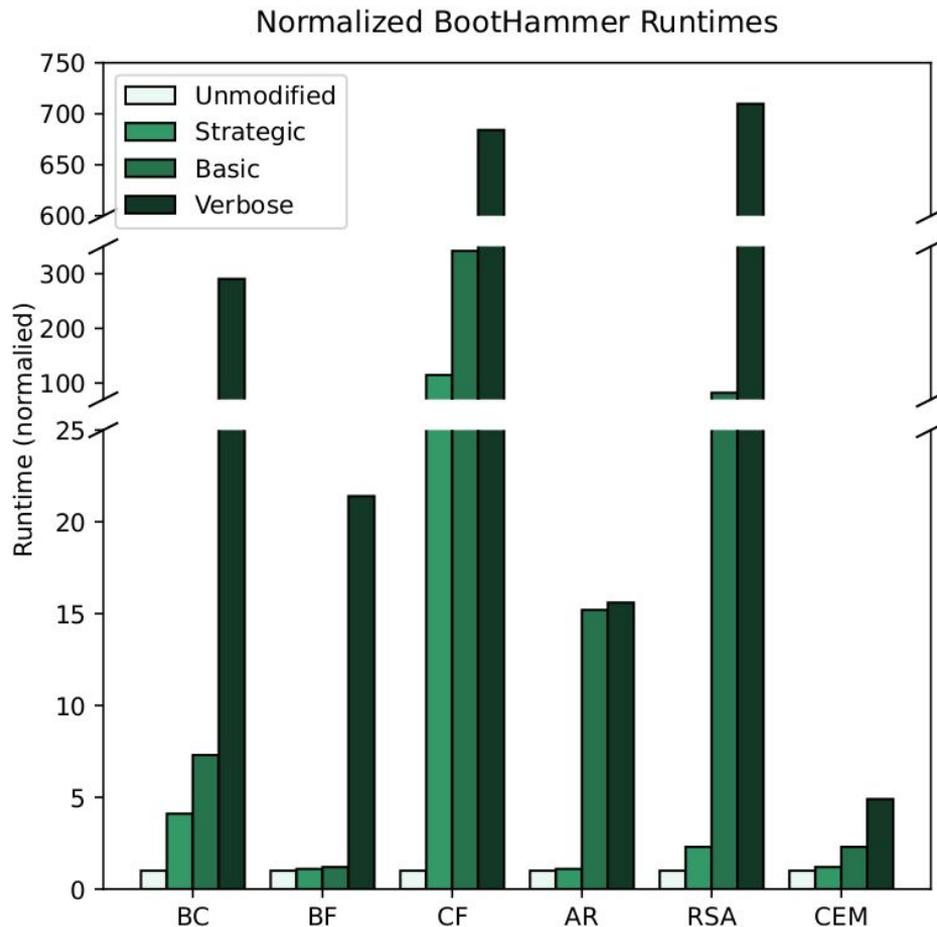
# BootHammer Runtimes

Verbose: Checkpoint every function and every iteration of a loop

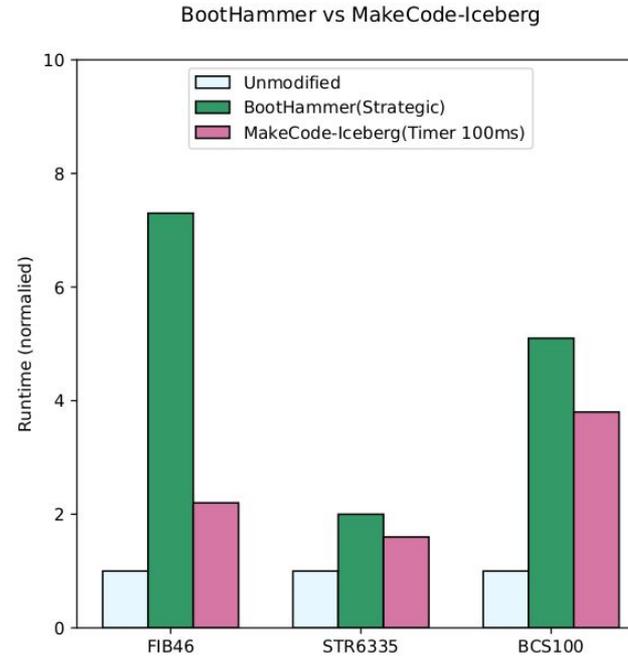
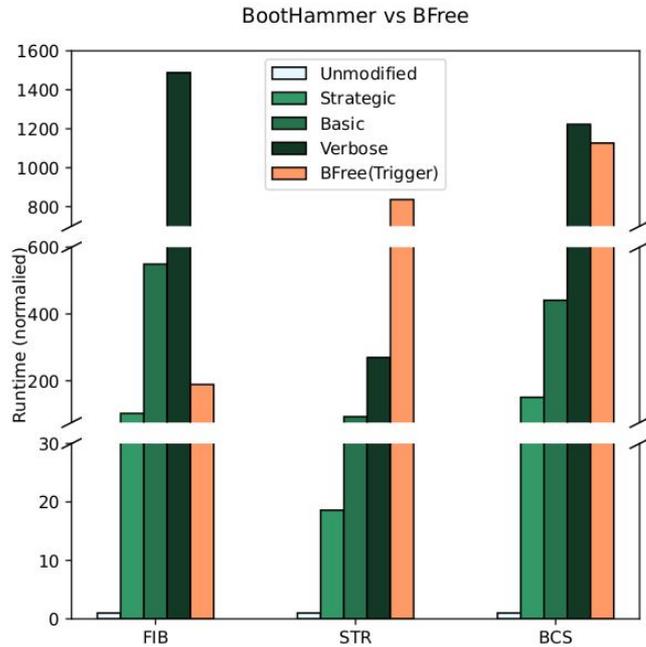
Basic: Checkpoint every function. Skip some iterations of a loop

Strategic: Smart placement of checkpoints\*

Unmodified: Native code

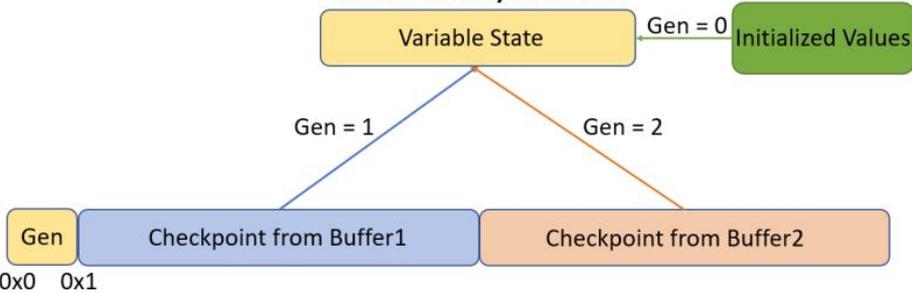


# BootHammer VS MakeCode-Iceberg and BFree



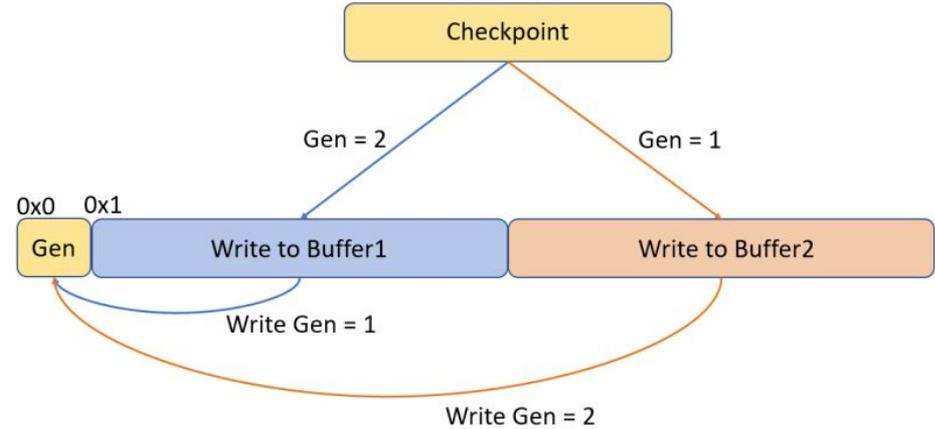
# Double Buffering the Checkpoints in Non-volatile Storage

Double Buffer System: Read



(a) Read operation for the double buffer system

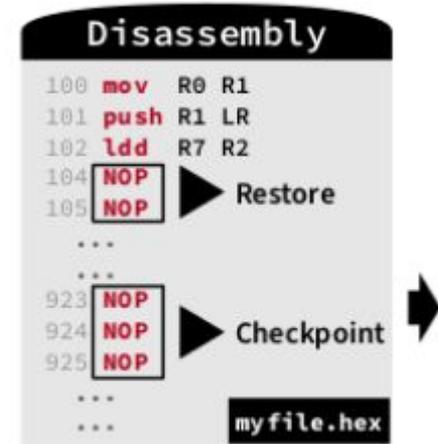
Double Buffer System: Write



(b) Write operation for the double buffer system

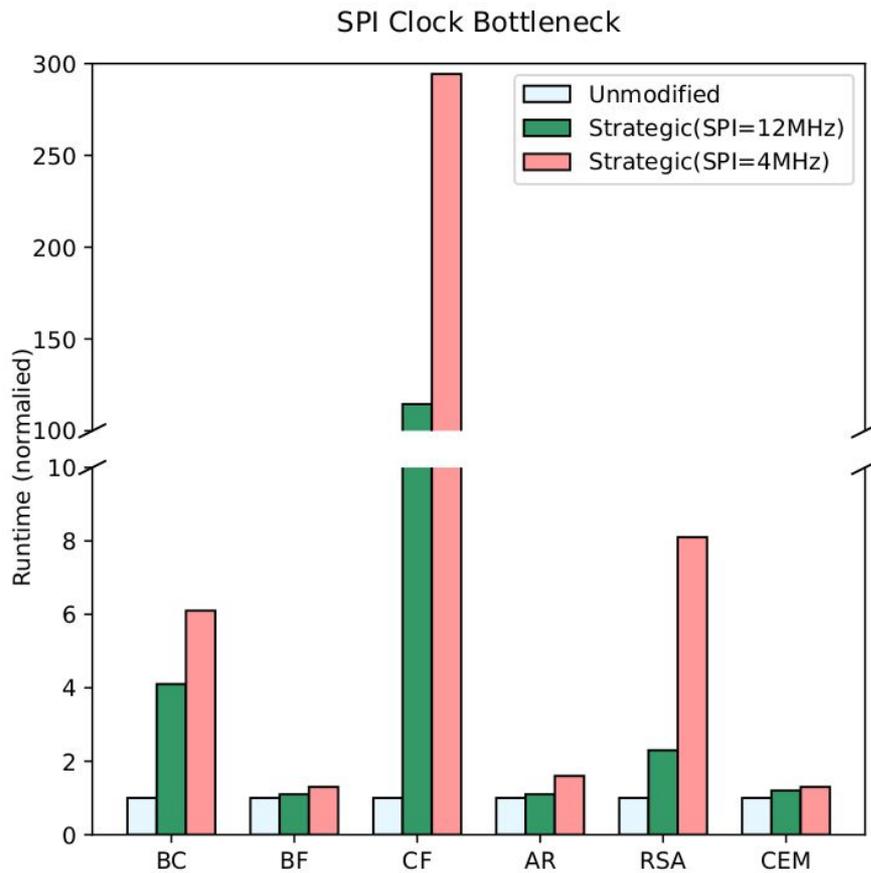
# Disassembly

- 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

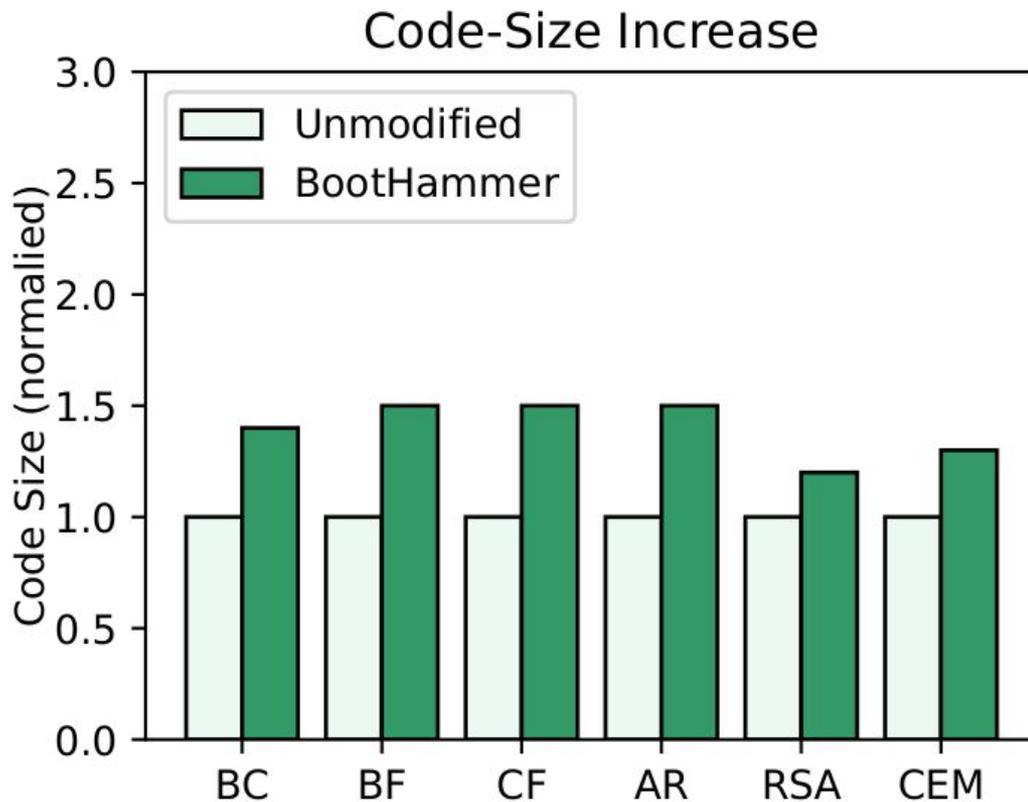


**Step 3:** Hex file disassembled to ease analysis for checkpoint insertion.

# SPI Clock(BootHammer)



# Code Size Increase



# Runtime

---

<b>Benchmark</b>	<b>BOOTHAMMER Rewriting Time(s)</b>
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()