Microsoft Research
Faculty Summit
2014 15TH ANNUAL

The Hidden Challenges of Intermittent Execution

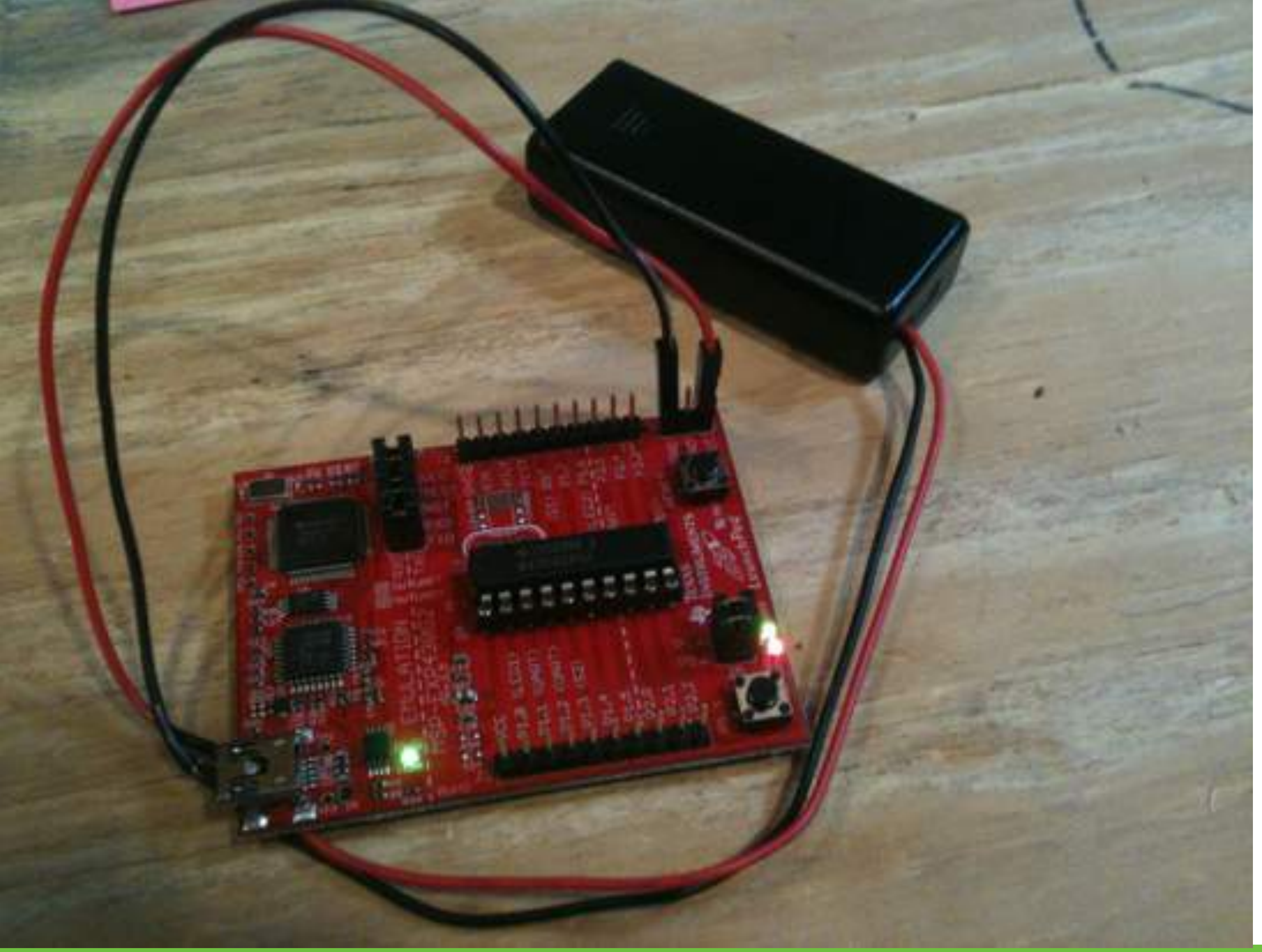
Brandon Lucia l Microsoft Research

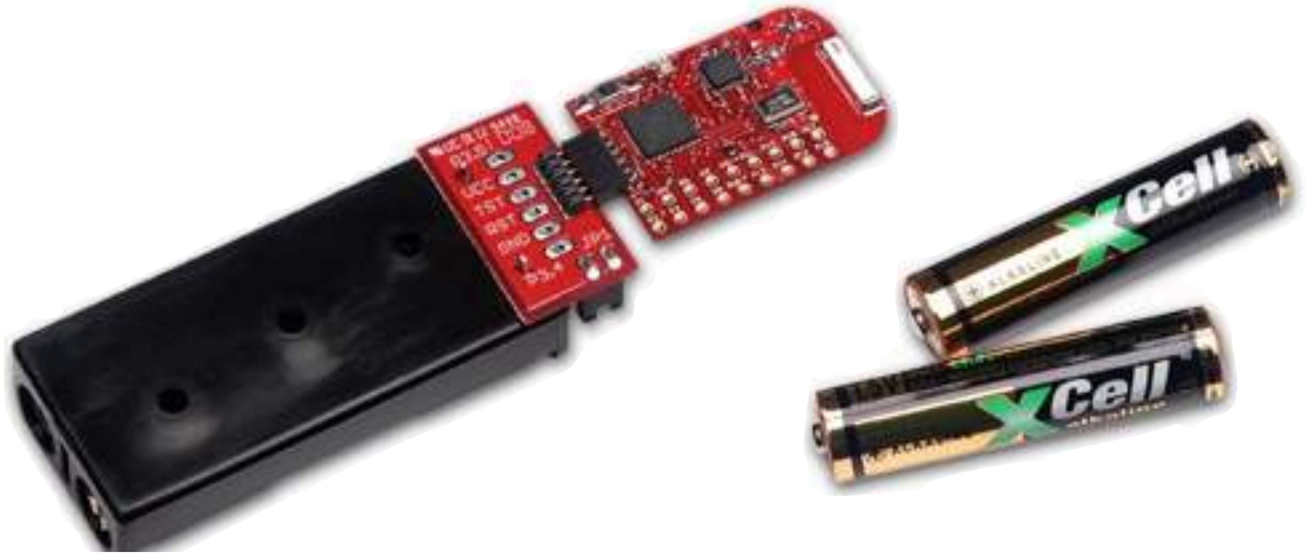
work done in cooperation with
Ben Ransford | University of Washington

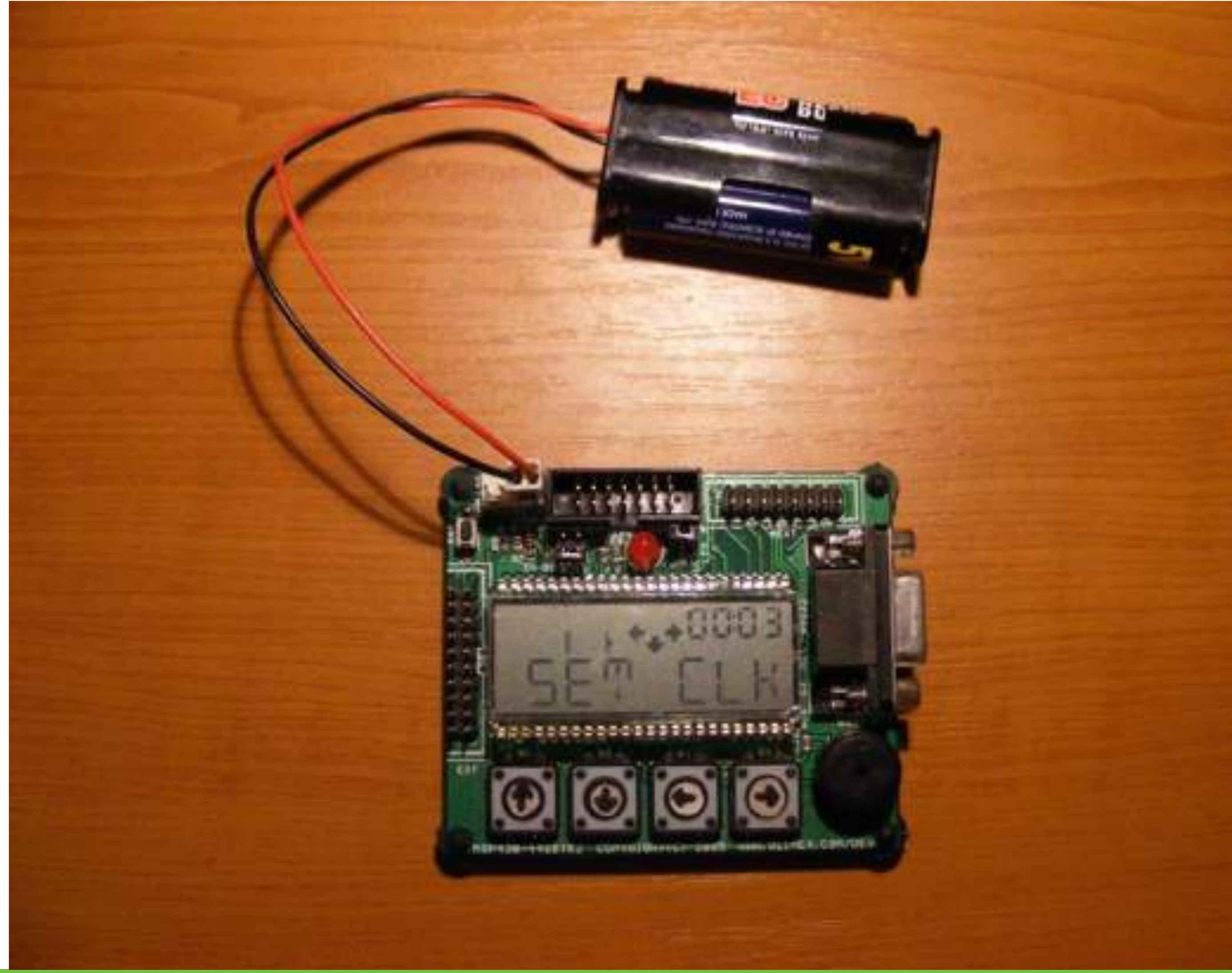
Microsoft

Emerging Devices Everywhere

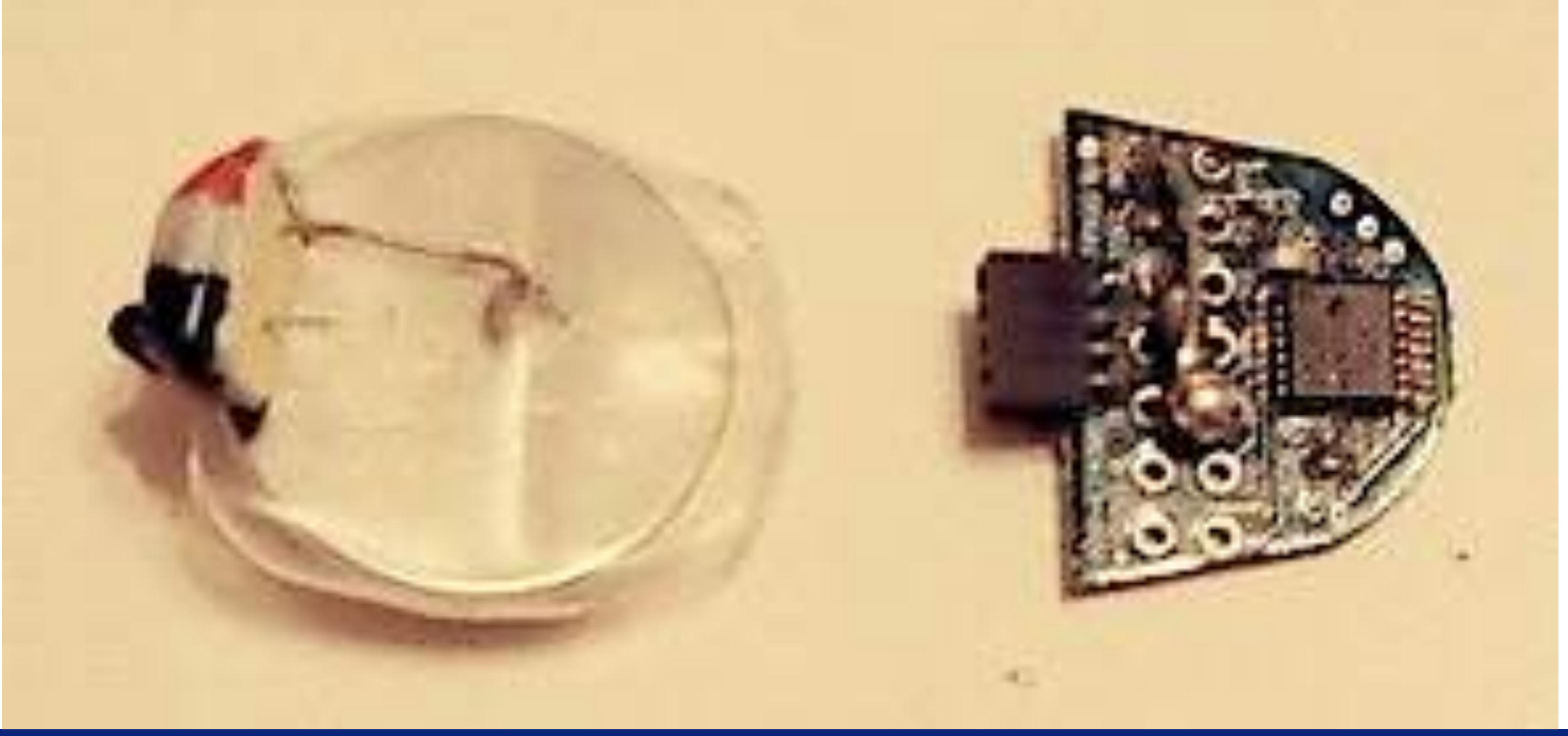
Wearables, sensors, "Internet of Things", medical implants, environmental monitoring, security, computational art, music, interactive clothing, smart furniture, smart carpets

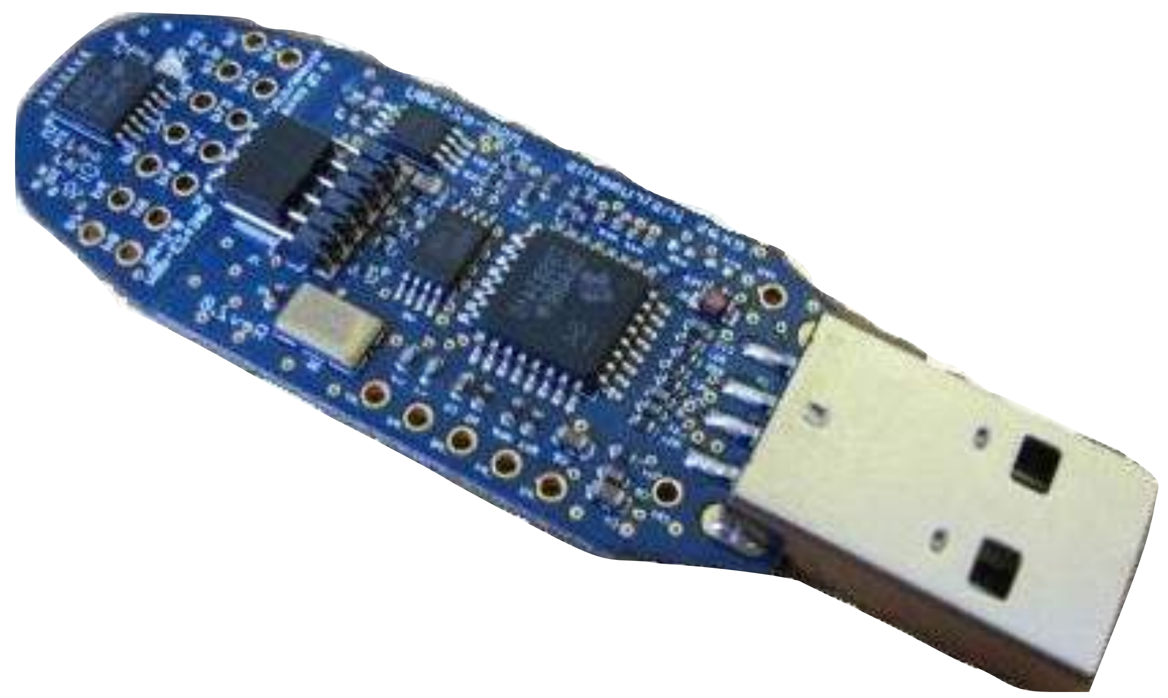


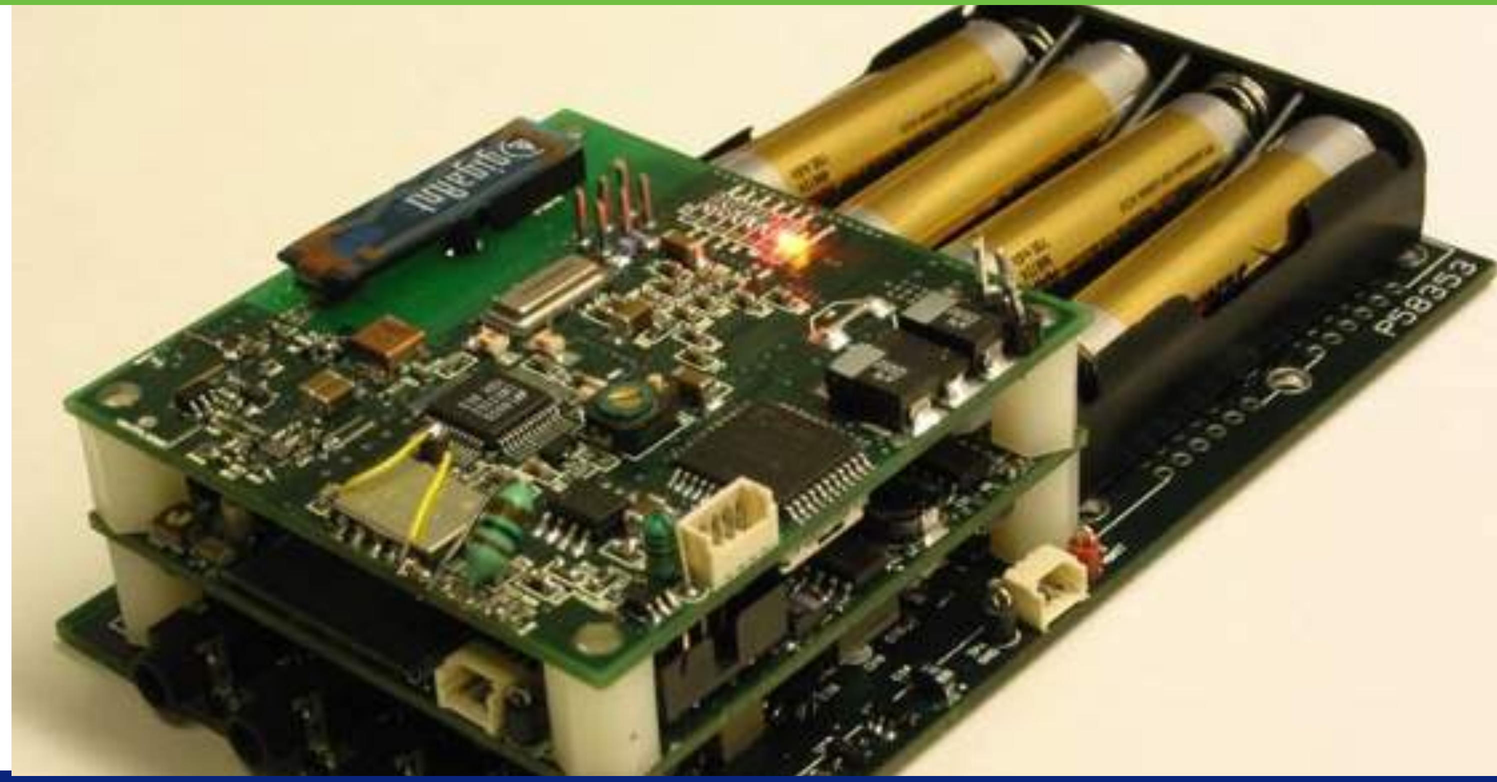


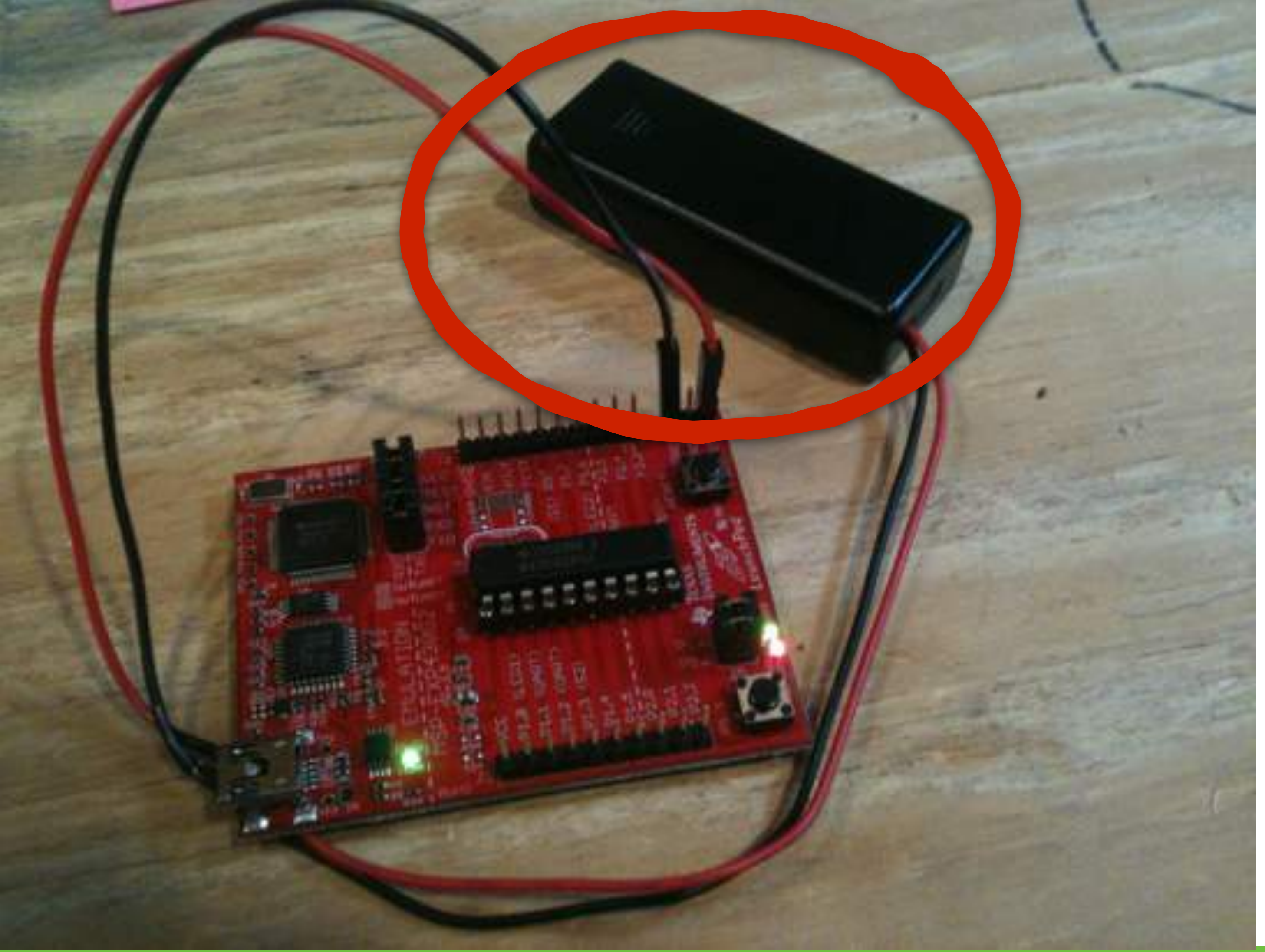


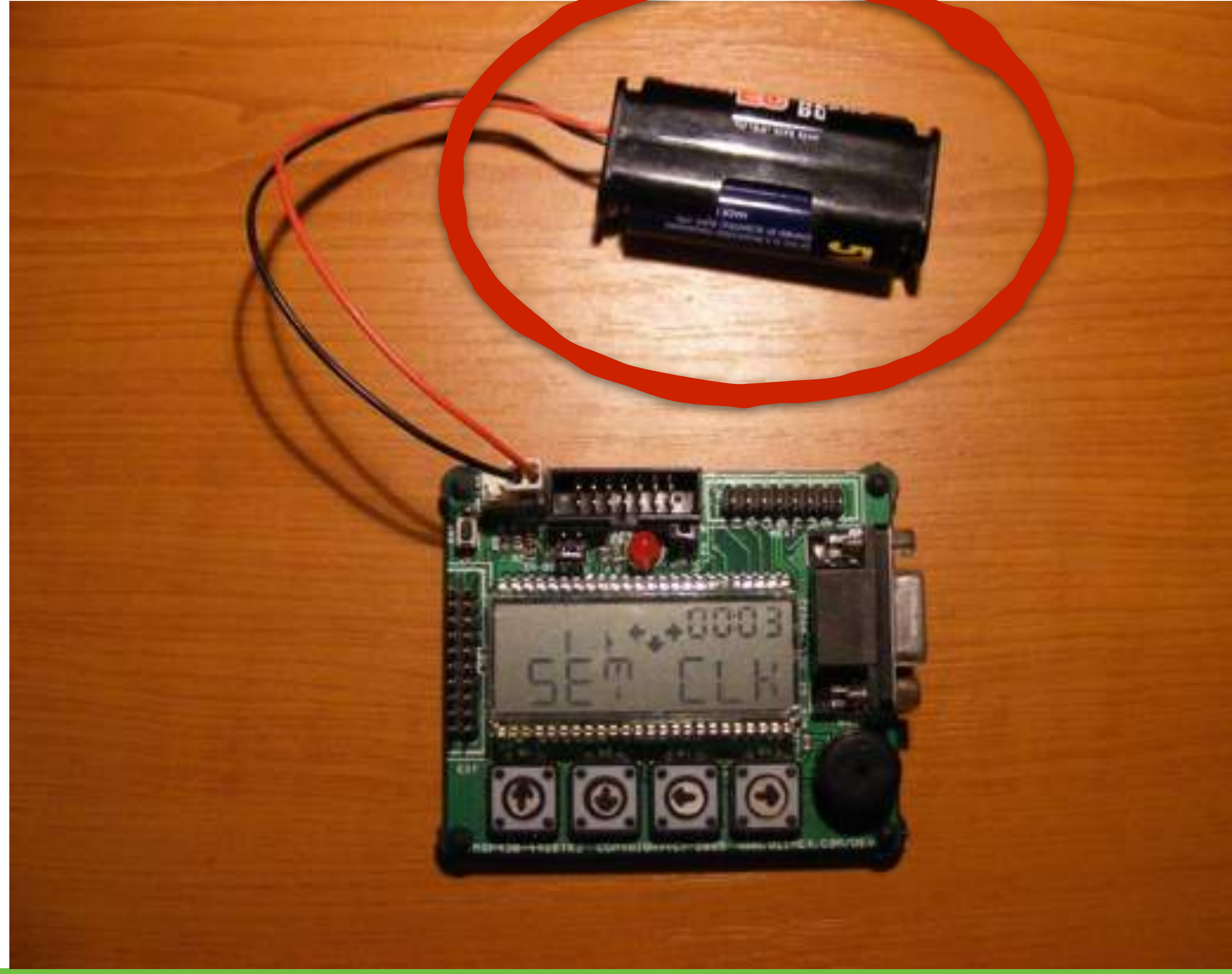
Tiny Computers Everywhere!



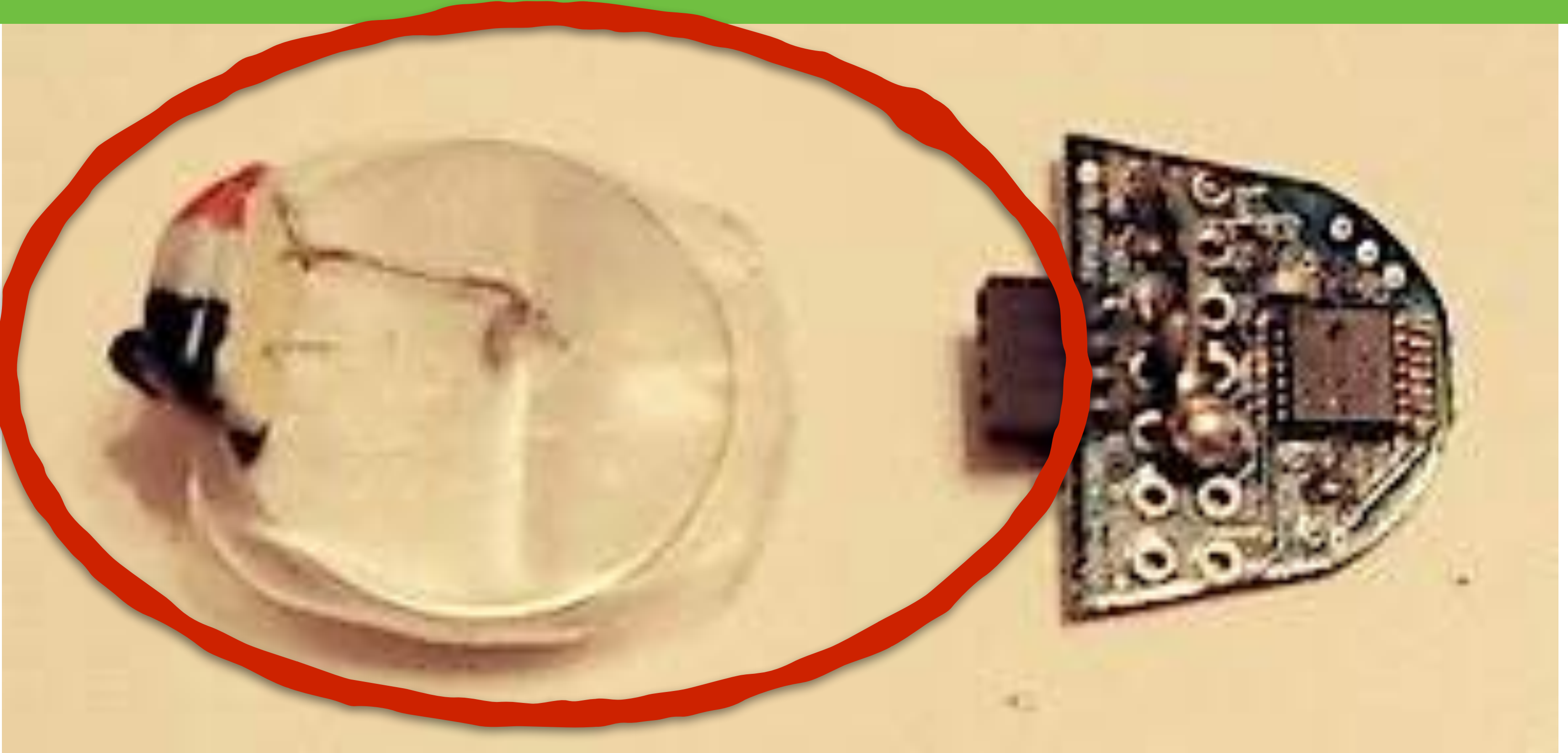


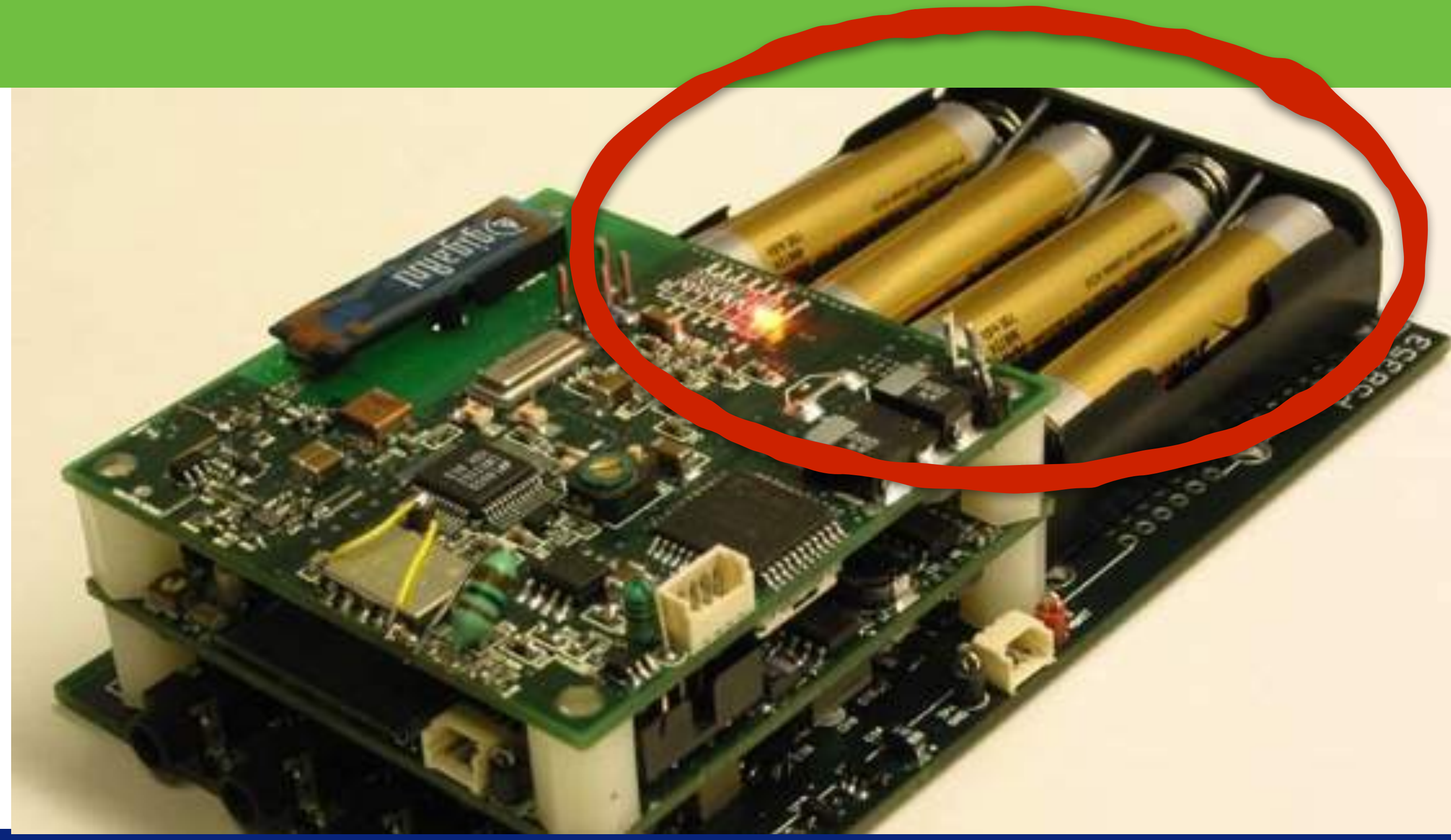


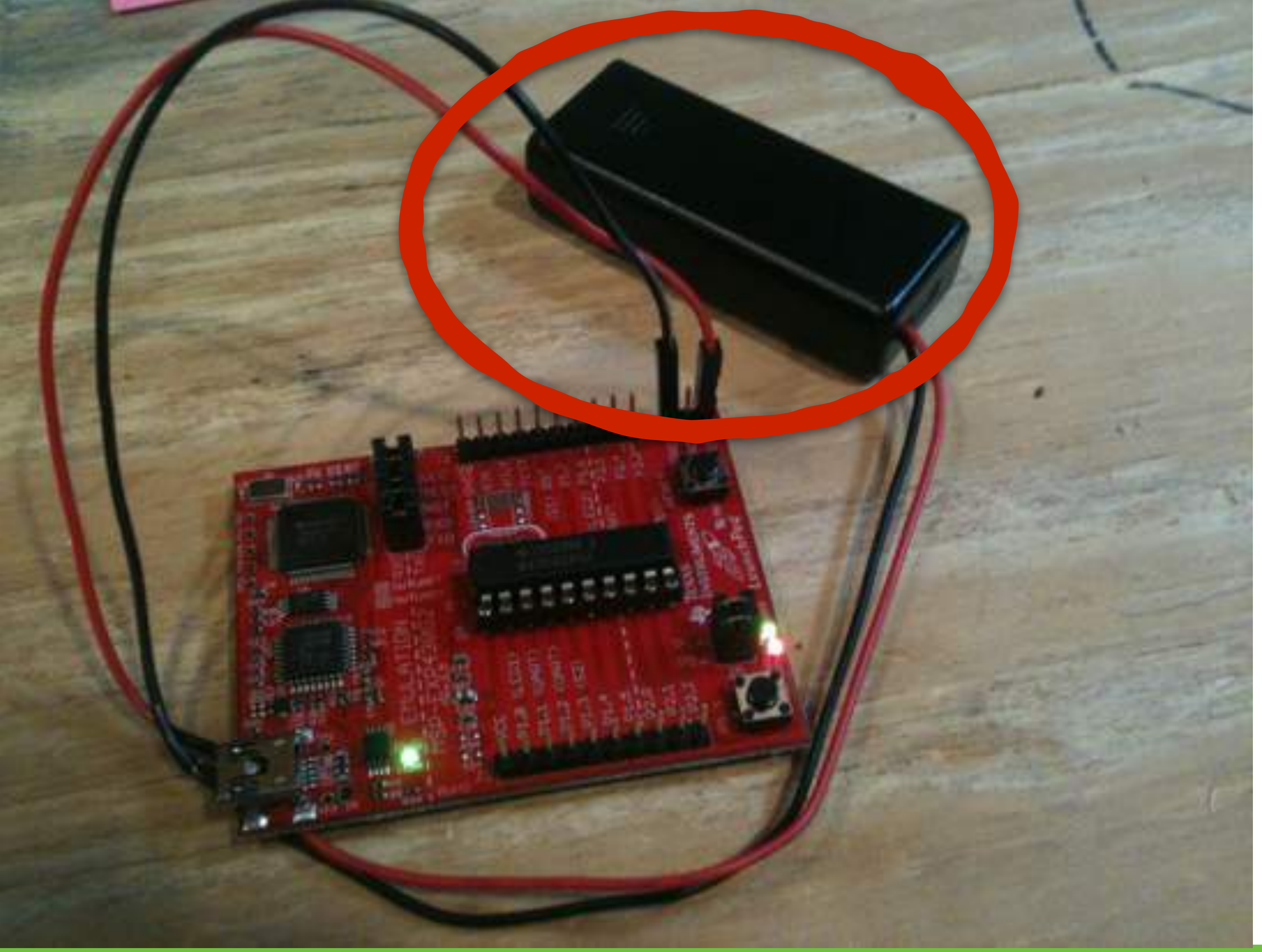


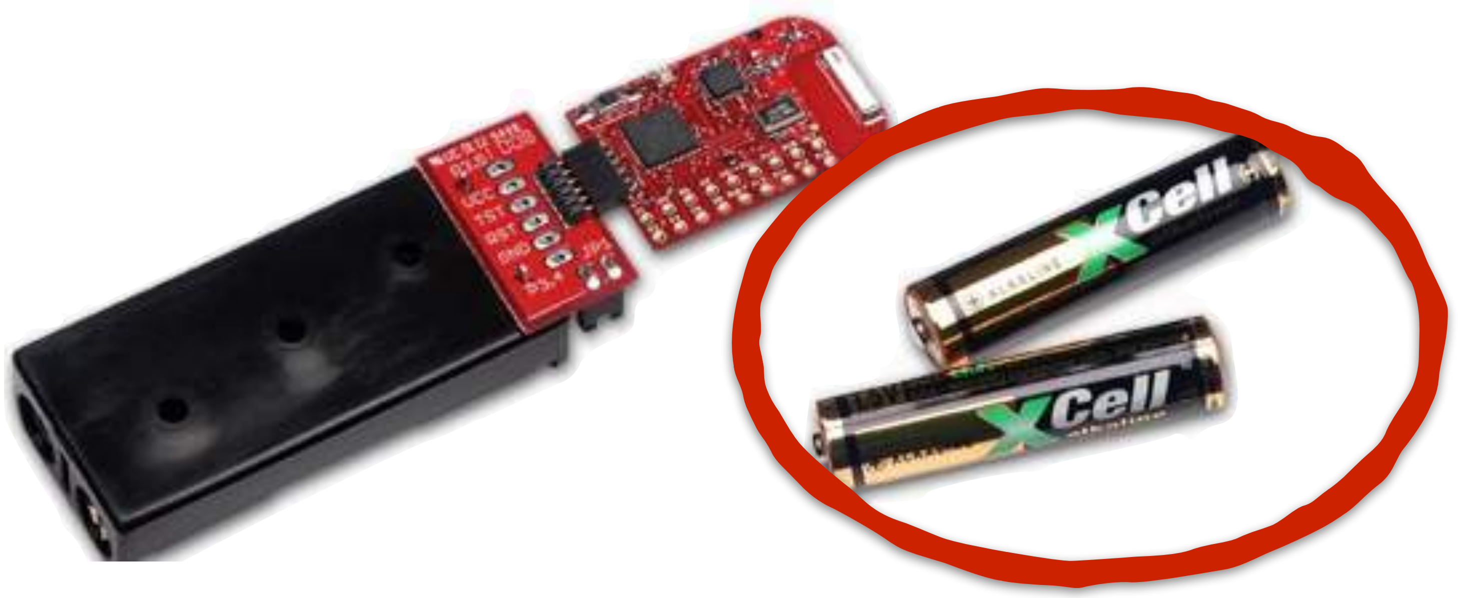


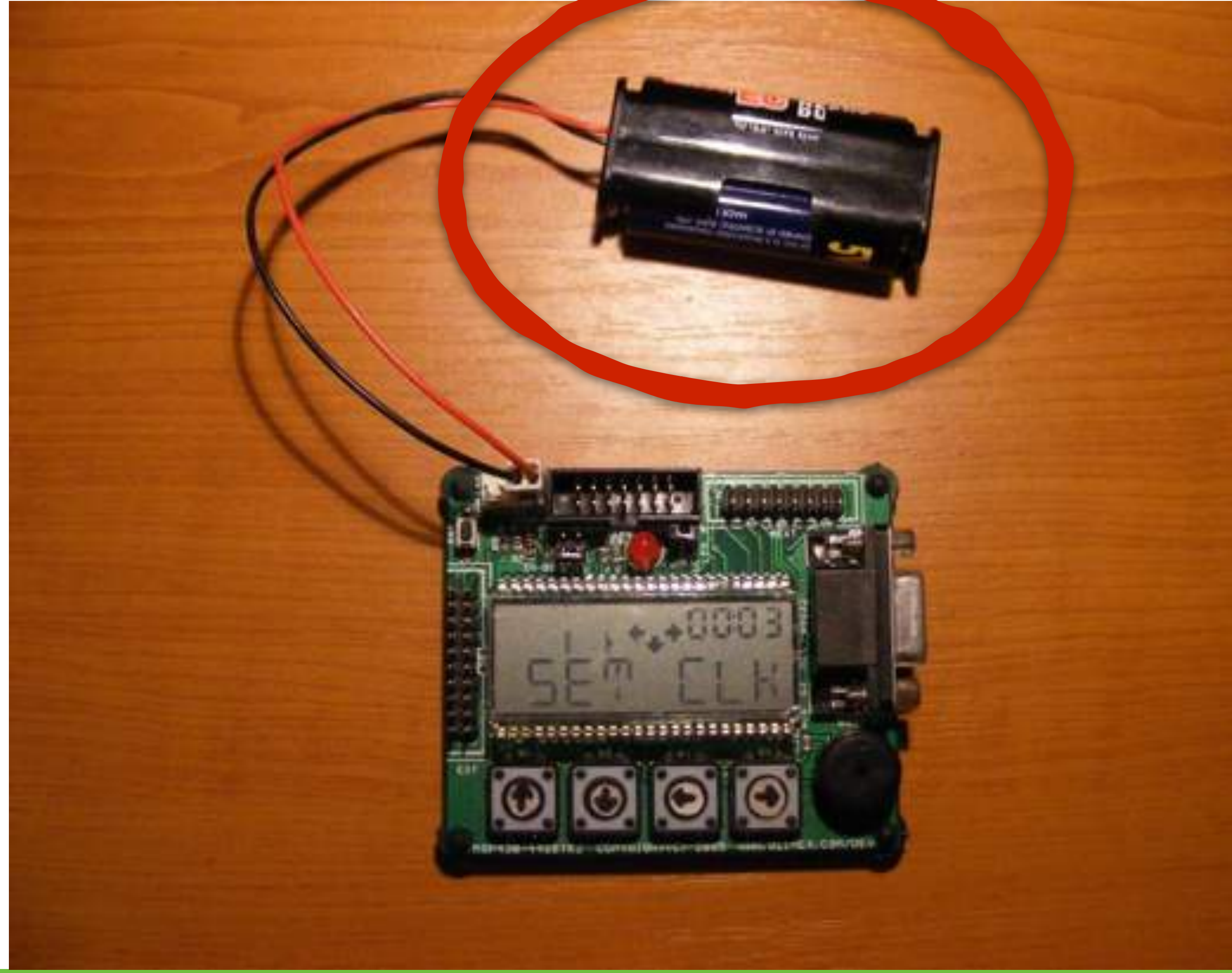
Systems expect **reliable power**



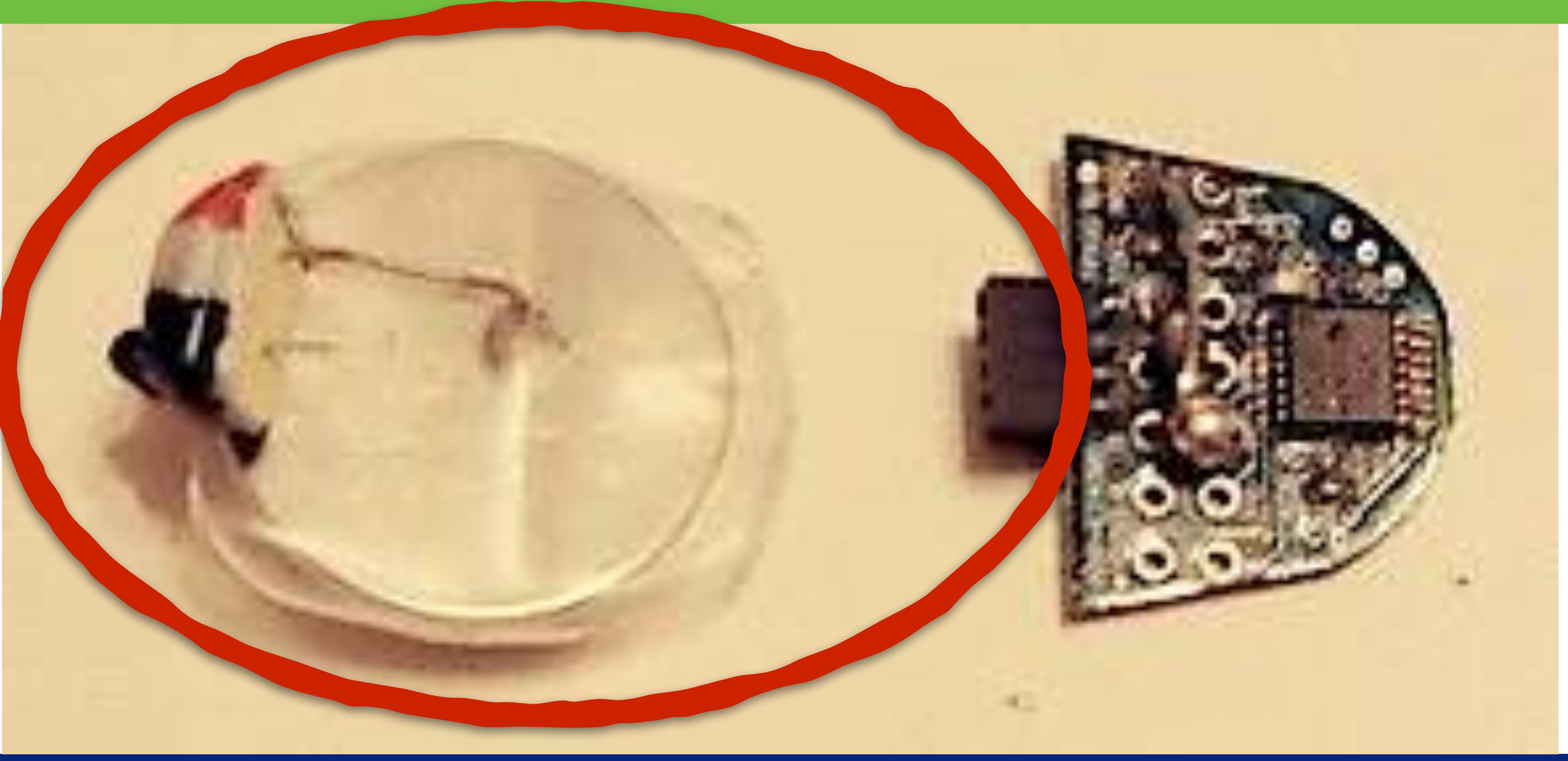


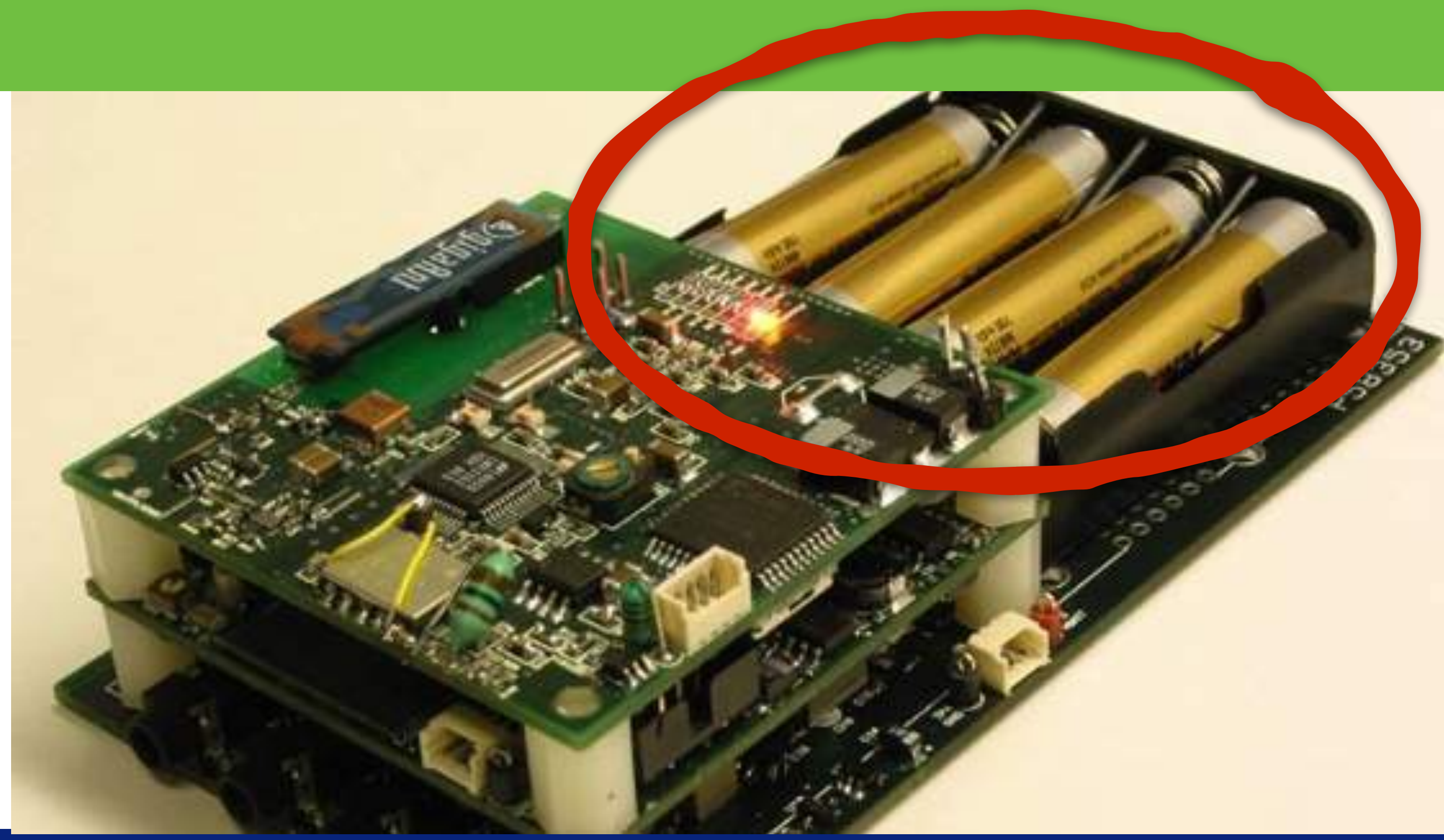






Tethered to a power source





Energy harvesting untethers devices

...but energy is **intermittent!**

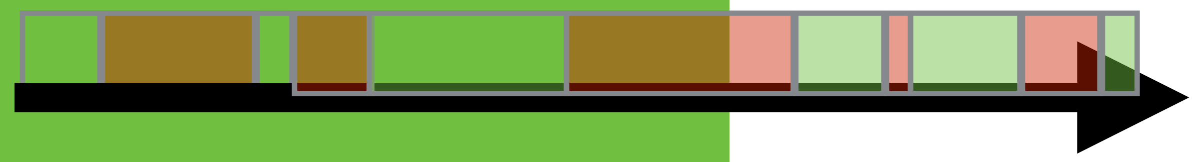
The Intermittent Execution Model

Thinking About Intermittently-powered Devices

Intermittence & Data Consistency

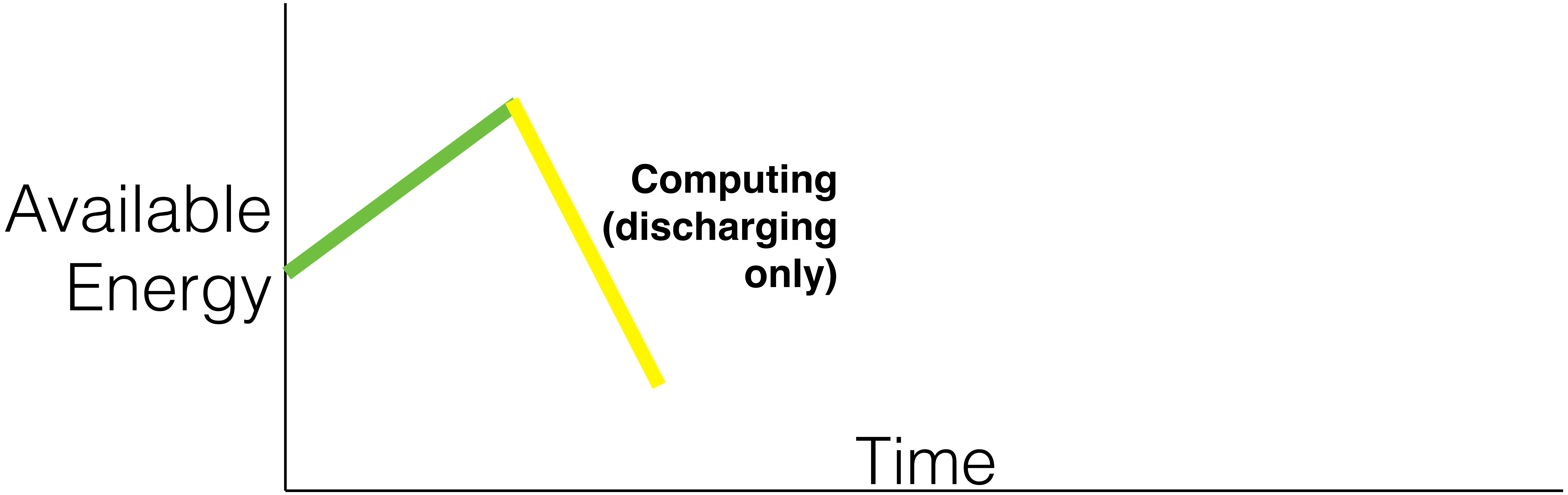
Hidden Problems Caused by Intermittence

Designing for Intermittence

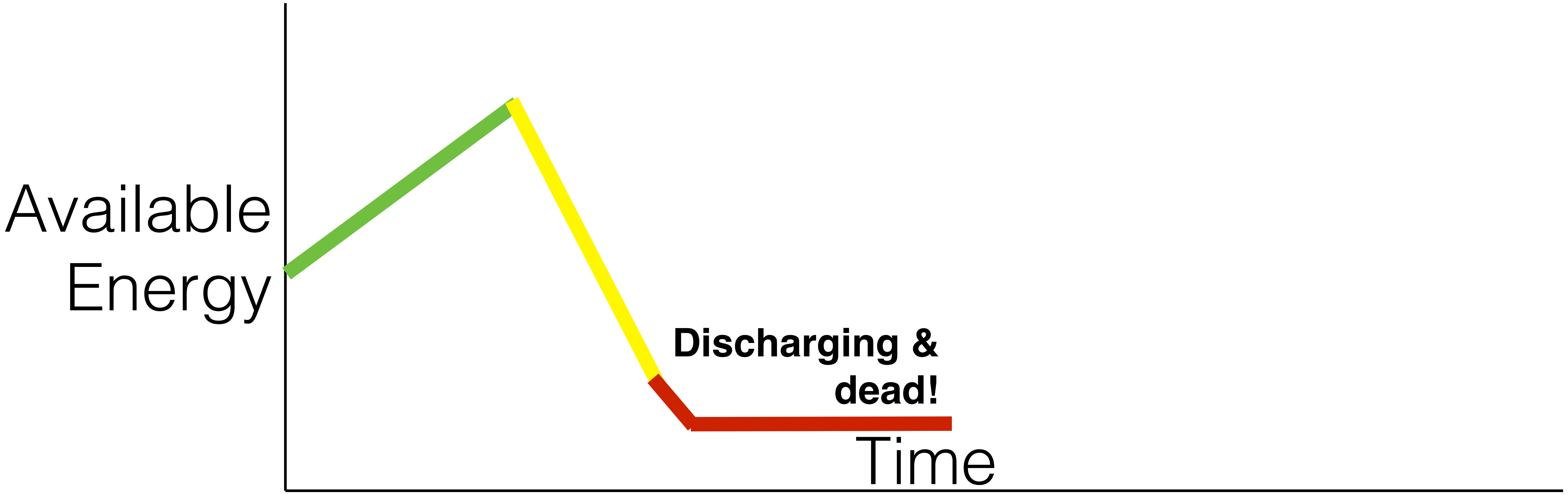


Strategies for Coping with Intermittence

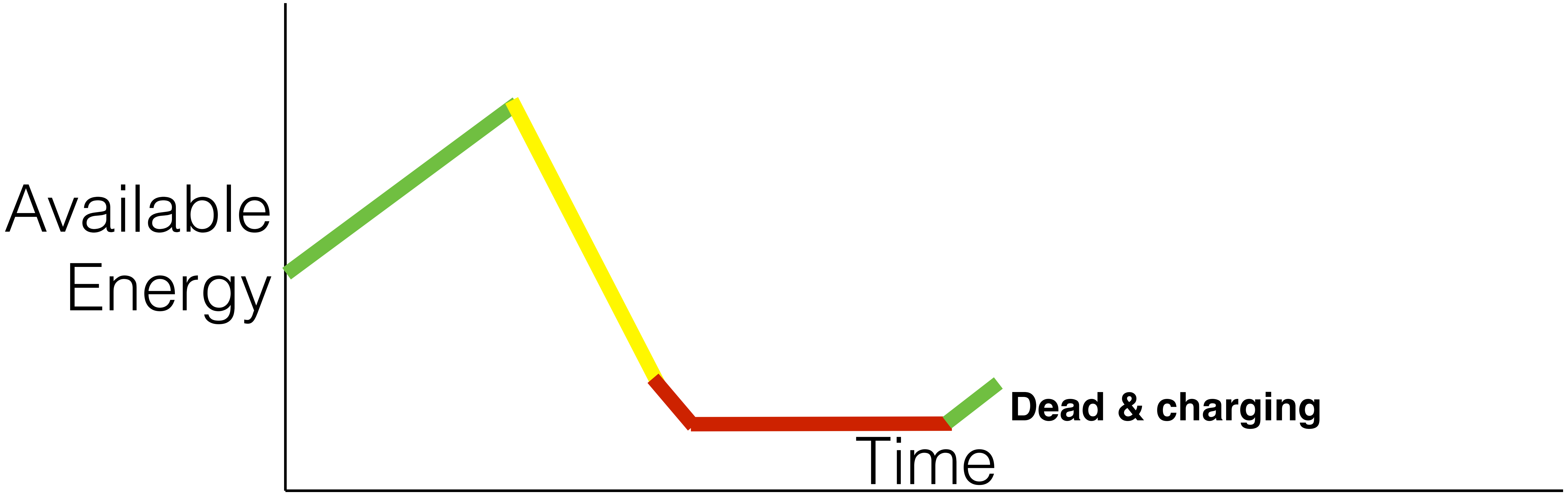
Running on intermittent energy



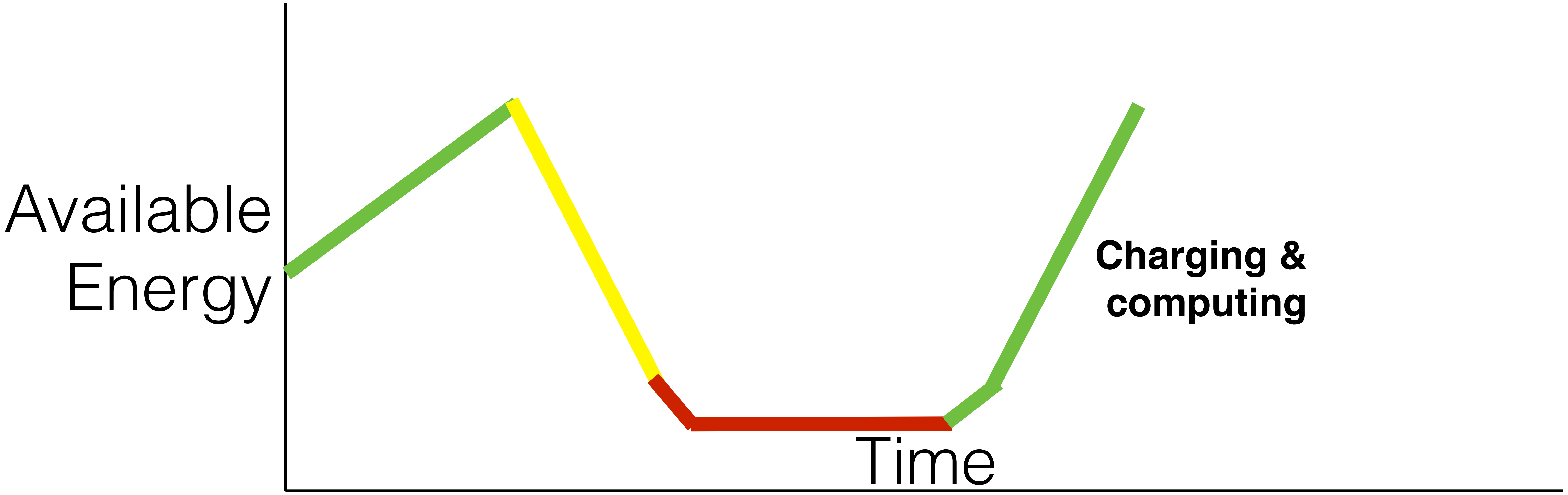
Running on intermittent energy



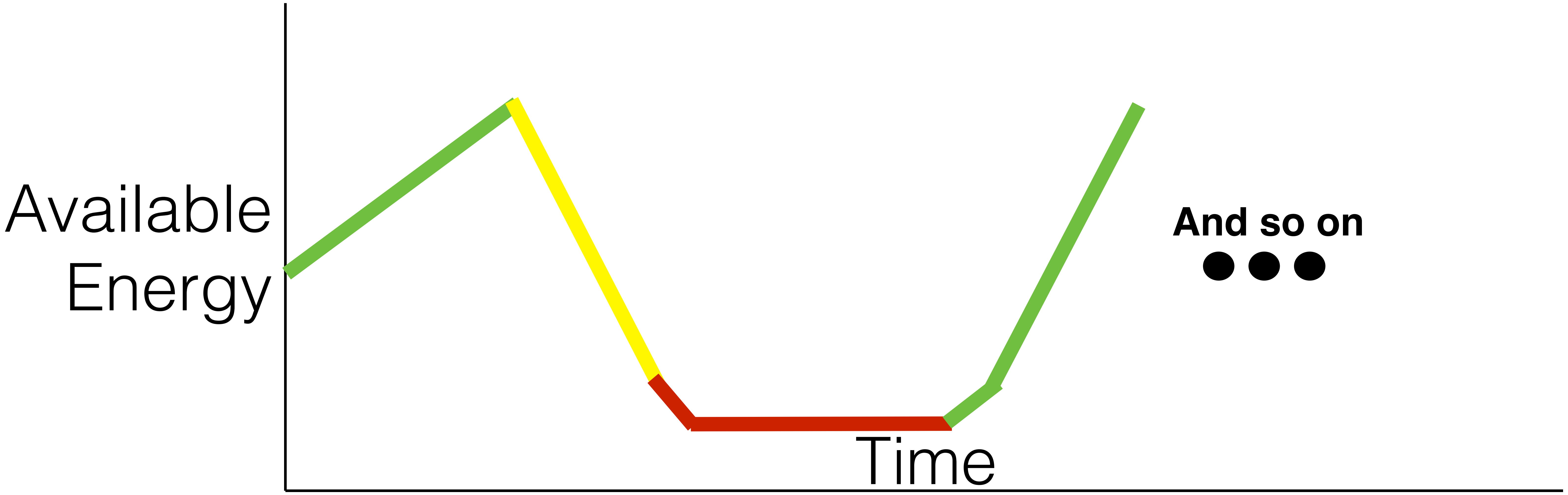
Running on intermittent energy



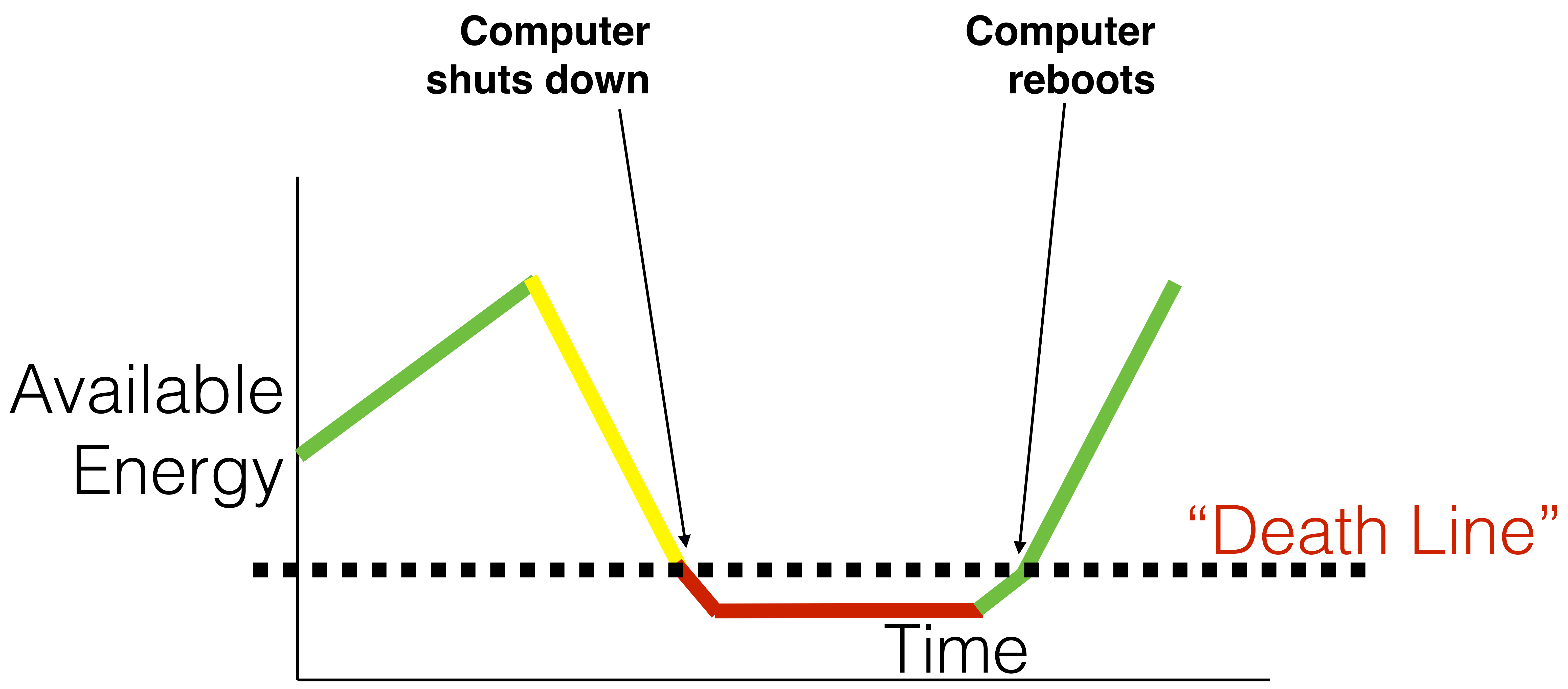
Running on intermittent energy

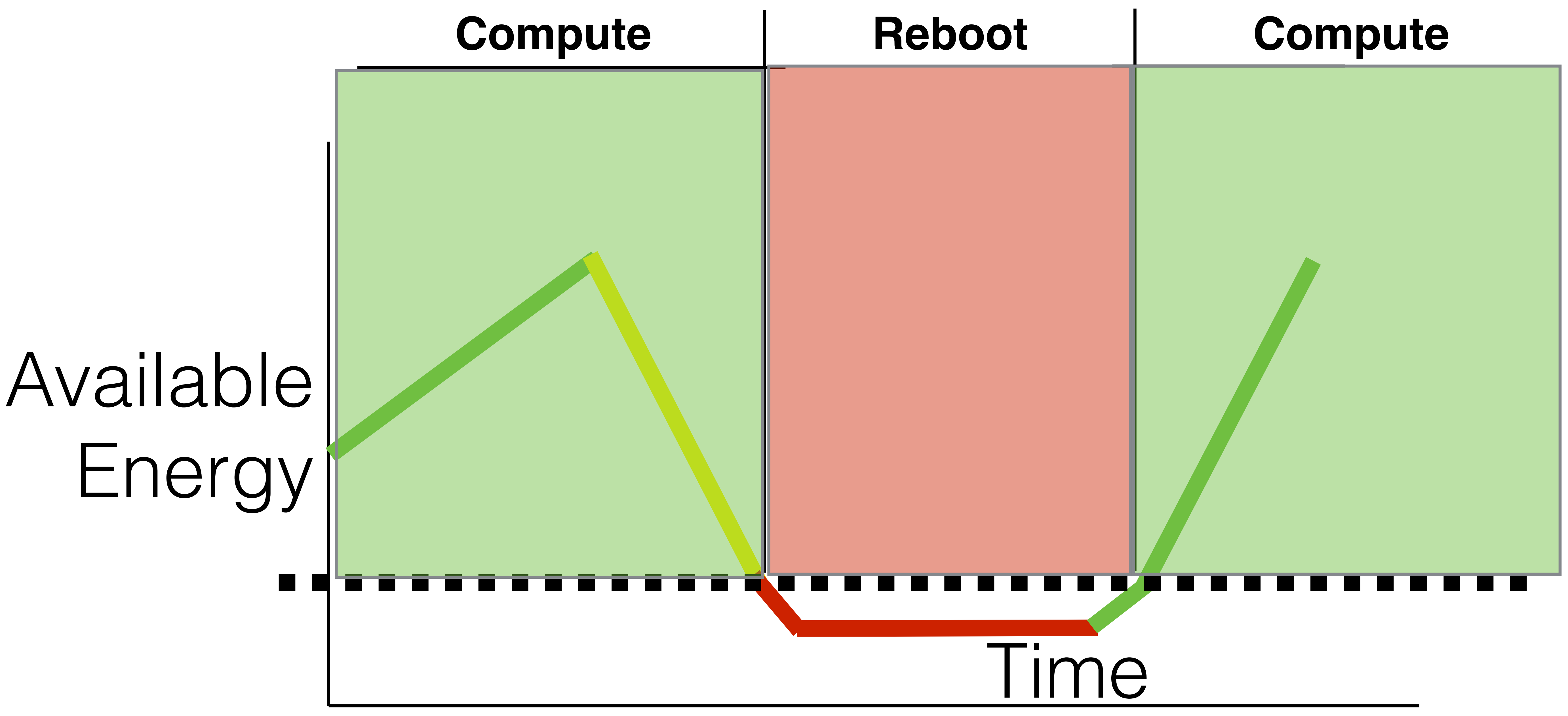


Running on intermittent energy

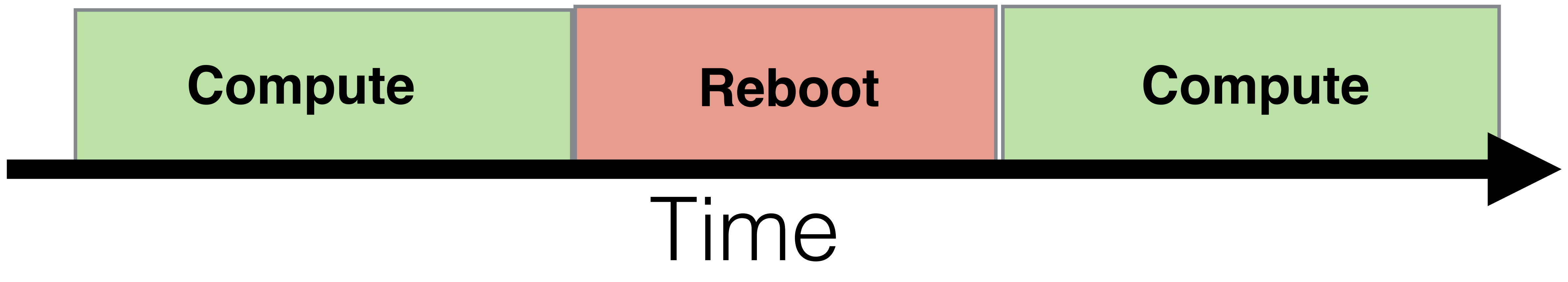


Running on intermittent energy

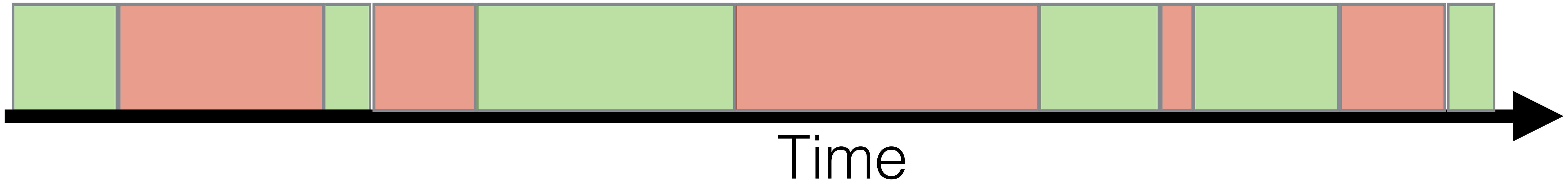

Computer
shuts down
Computer
reboots
Available
Energy
"Death Line"
Time


Compute
Reboot
Compute
Available Energy
Time

The Intermittent Execution Model

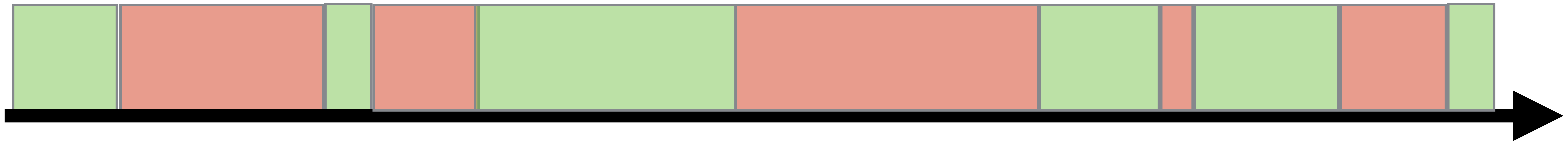


The Intermittent Execution Model

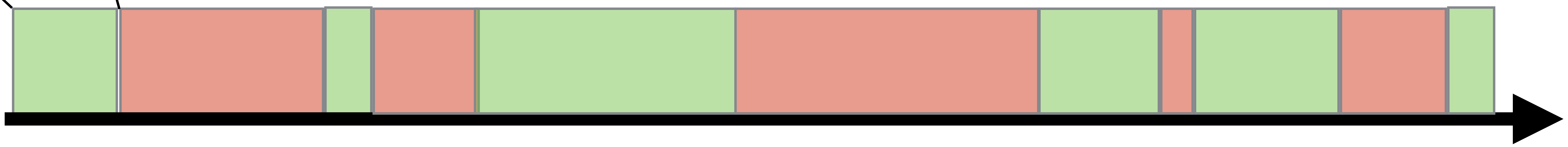


Goal: Run programs that take longer than one green box

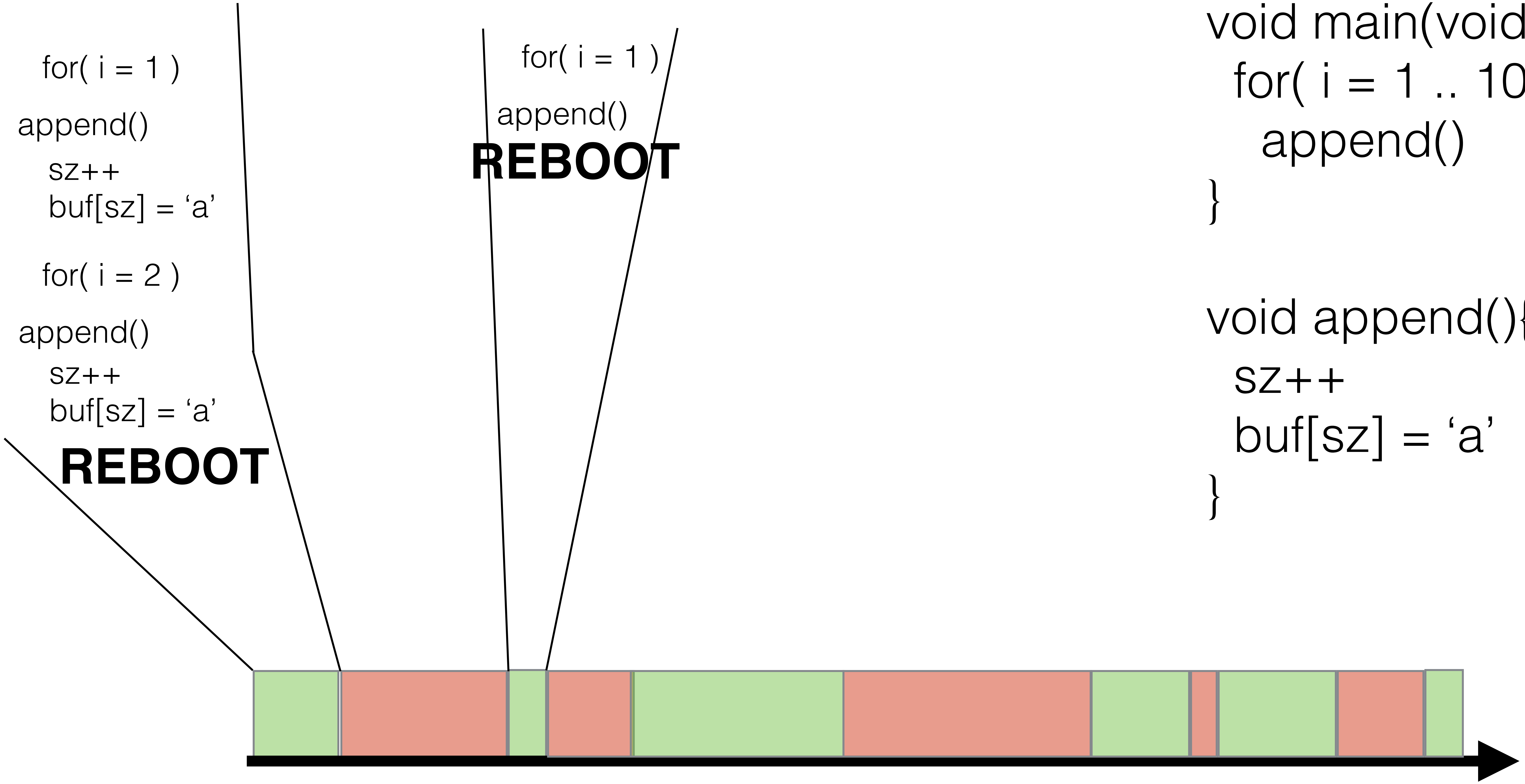
```
void main(void){
  for( i = 1 .. 10)
    append()
}

void append(){
  sz++
  buf[sz] = 'a'
}
```



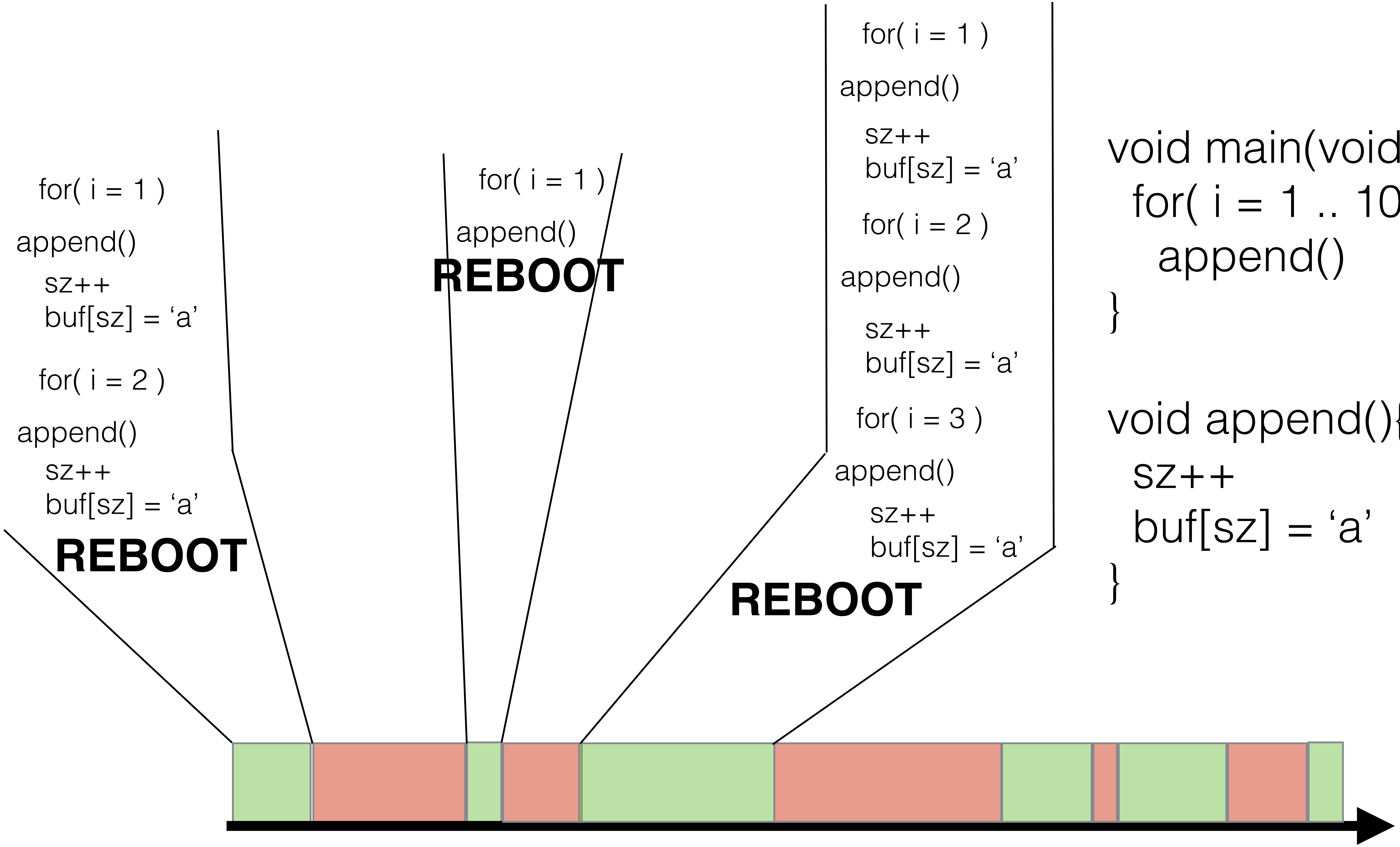
for(i = 1)
append()
sz++
buf[sz] = ‘a’
for(i = 2)
append()
sz++
buf[sz] = ‘a’
REBOOT



```
void main(void){
  for( i = 1 .. 10)
    append()
}

void append(){
  sz++
  buf[sz] = ‘a’
}
```

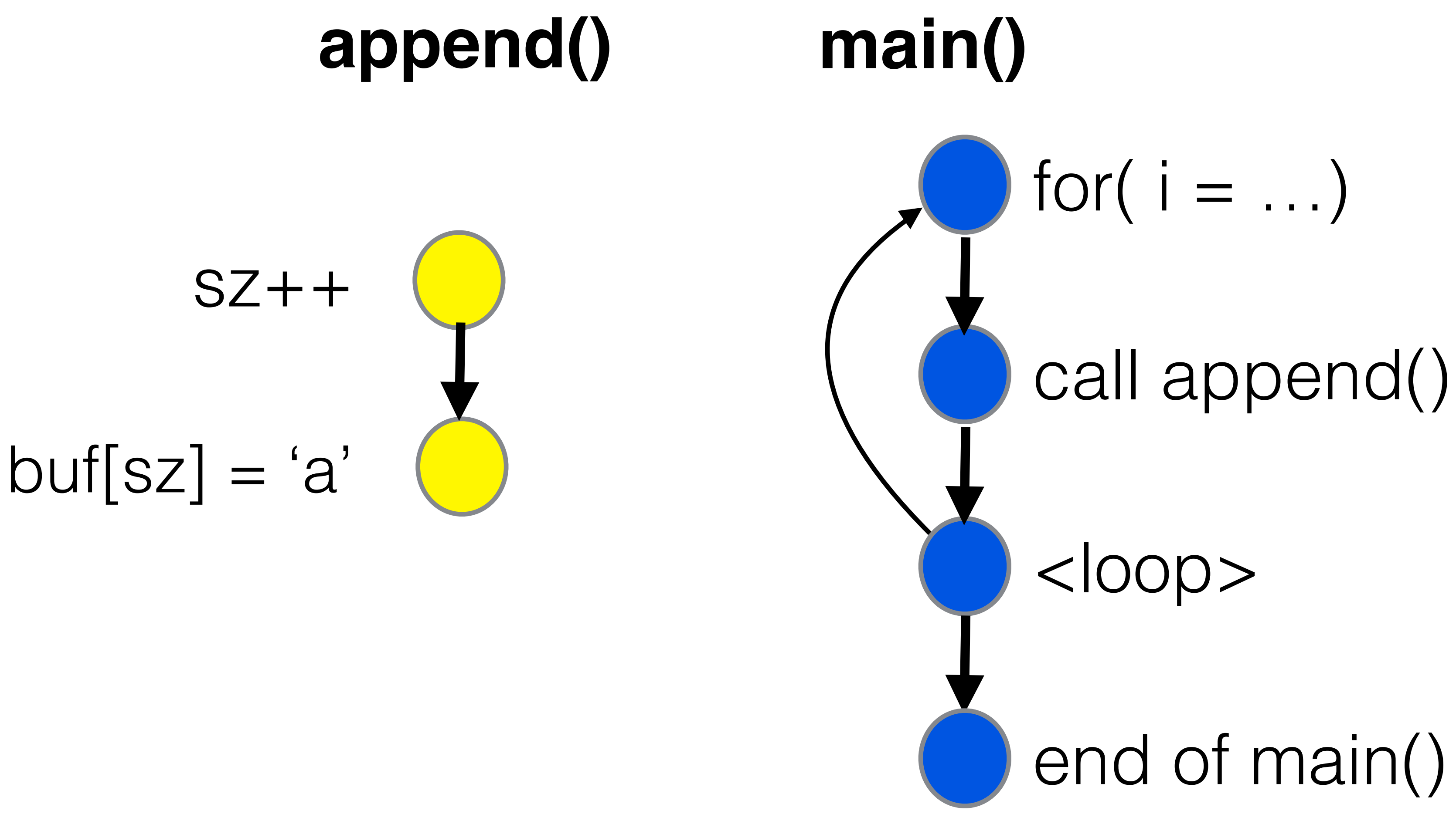



```
void main(void){
  for( i = 1 .. 10)
    append()
}

void append(){
  sz++
  buf[sz] = ‘a’
}
```



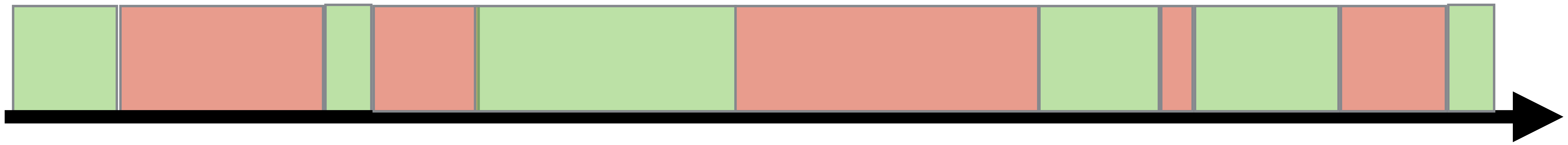

```
void main(void){
  for( i = 1 .. 10)
    append()
}

void append(){
  sz++
  buf[sz] = ‘a’
}
```

Control-flow Graph

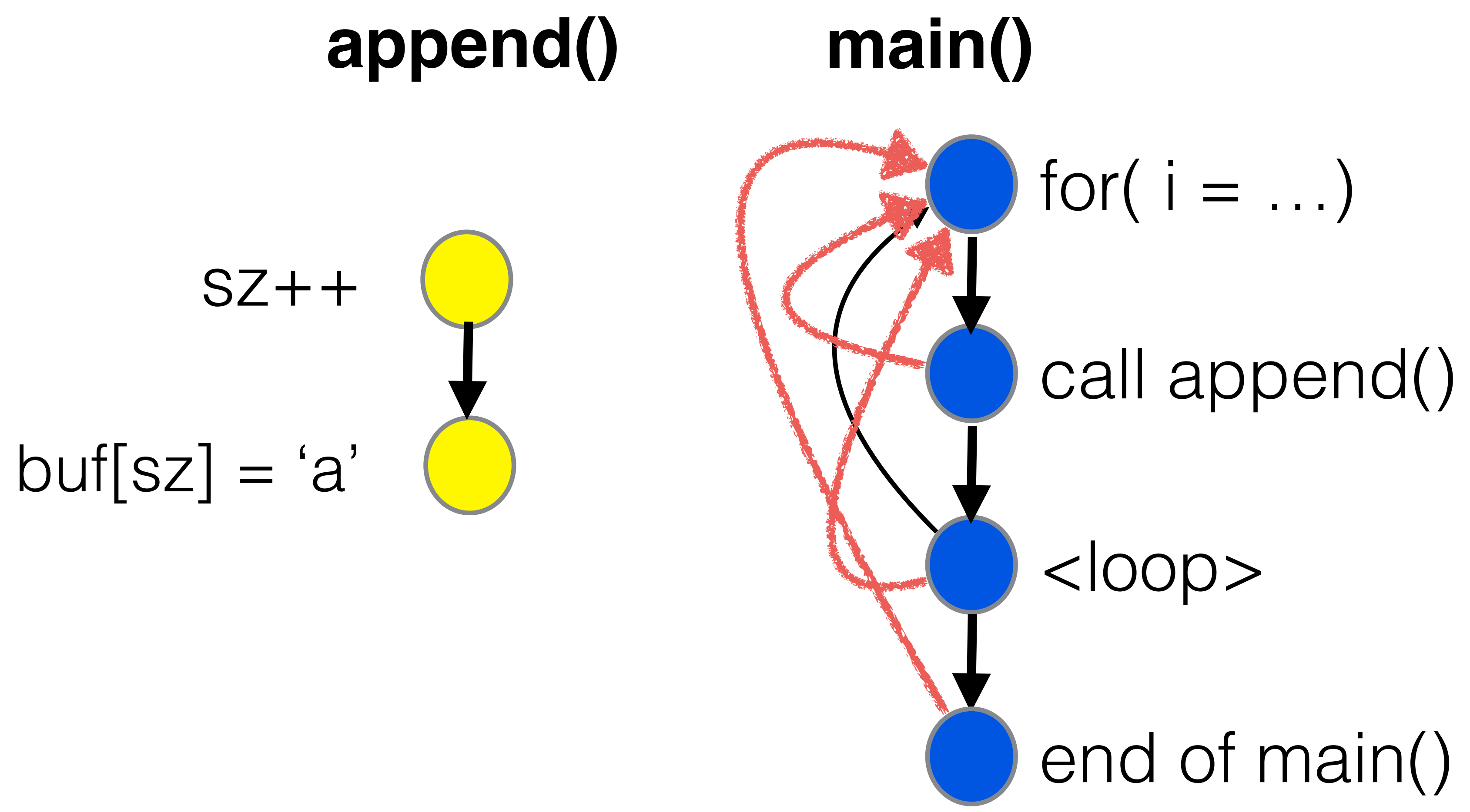



```
void main(void){
  for( i = 1 .. 10)
    append()
}

void append(){
  sz++
  buf[sz] = ‘a’
}
```

We can model intermittence as a control-flow problem

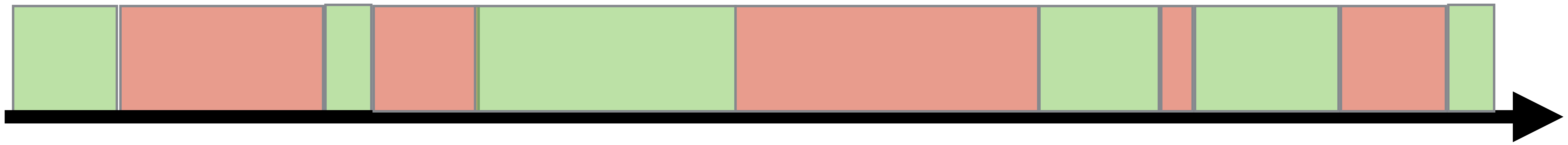


Control-flow Graph

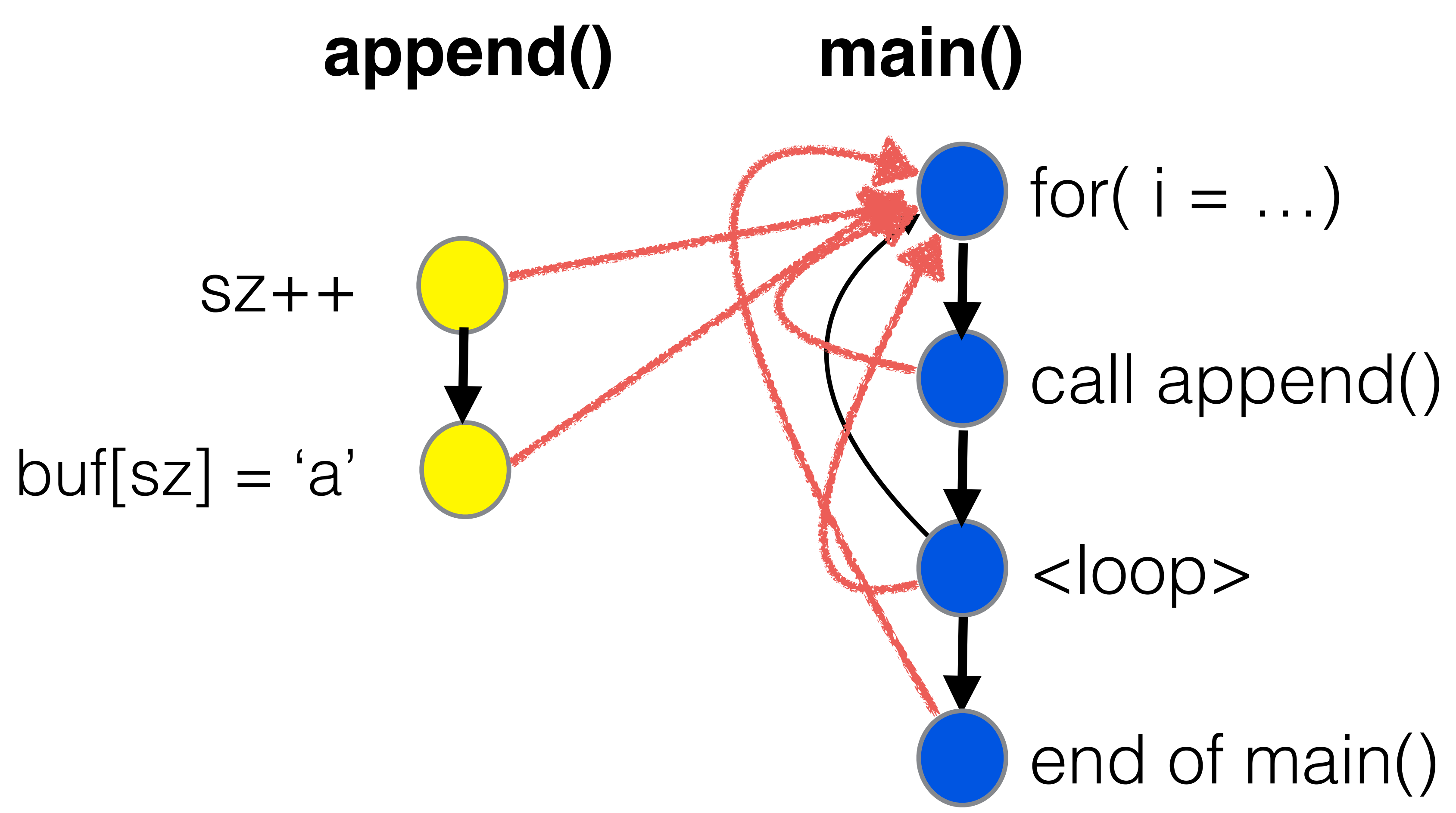



```
void main(void){
  for( i = 1 .. 10)
    append()
}

void append(){
  sz++
  buf[sz] = ‘a’
}
```

Failure Induces
Implicit Control-flow

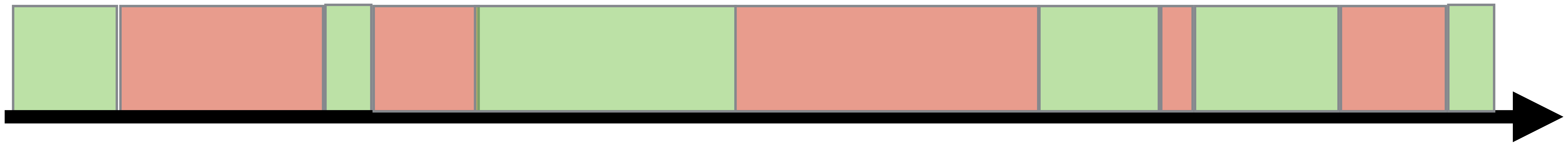


Control-flow Graph



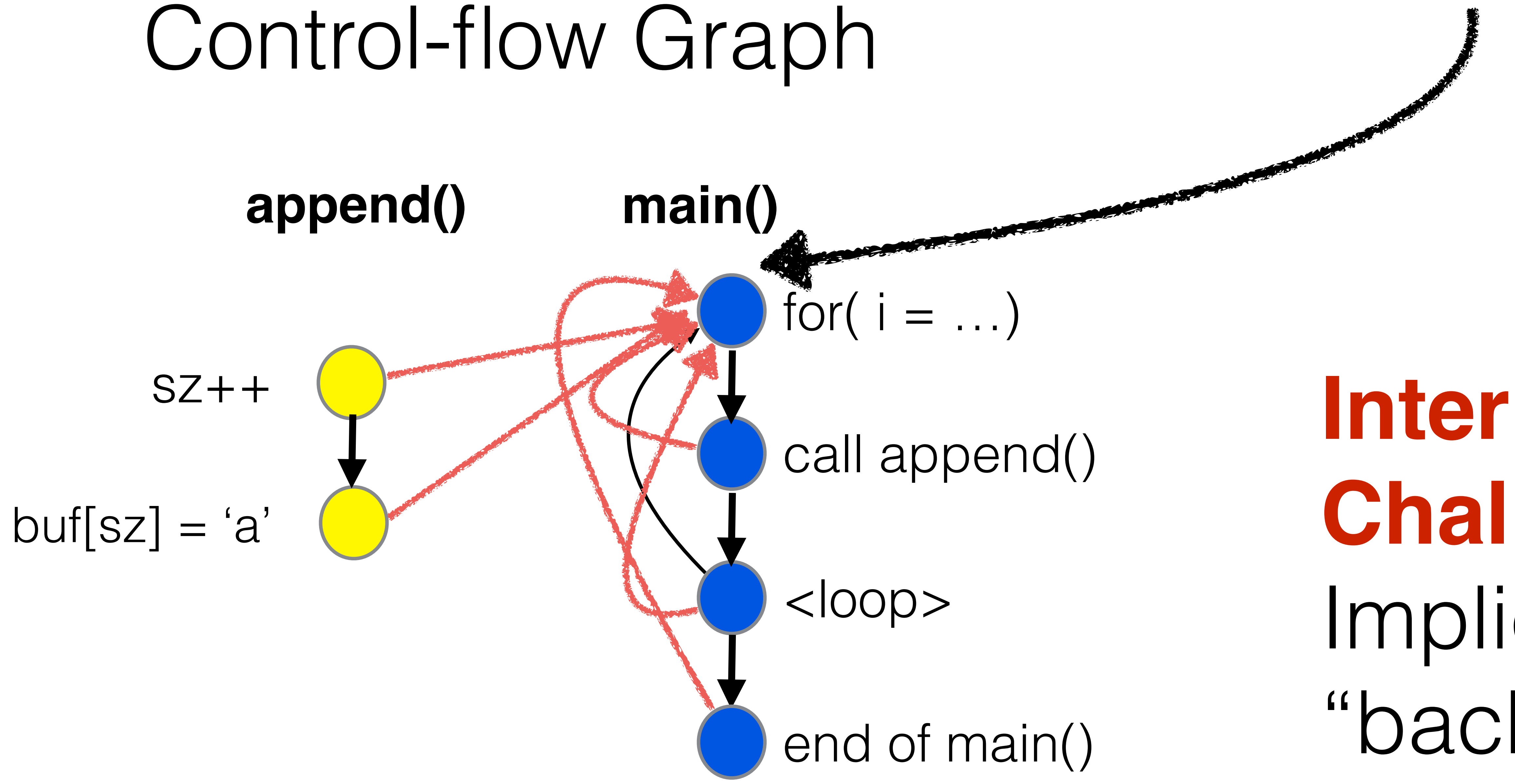

```
void main(void){
  for( i = 1 .. 10)
    append()
}

void append(){
  sz++
  buf[sz] = ‘a’
}
```

Failure Induces
Non-Local, Implicit Control-flow



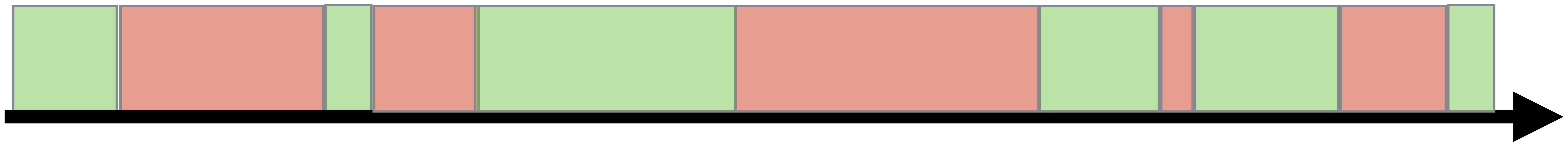
Back in time!

Control-flow Graph

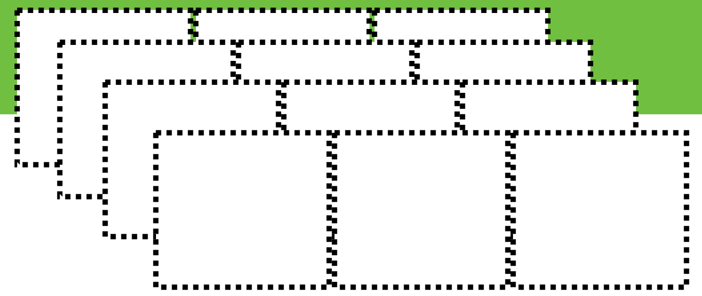


Intermittent Execution Challenge #1:
Implicit, non-local control-flow “back in time” on reboot.

Failure Induces
Non-Local, Implicit Control-flow

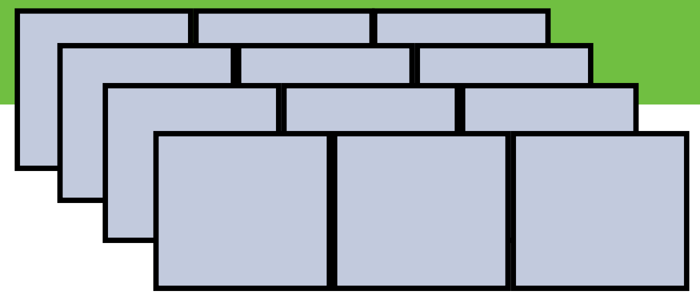


Mixture of Volatile & Non-volatile State



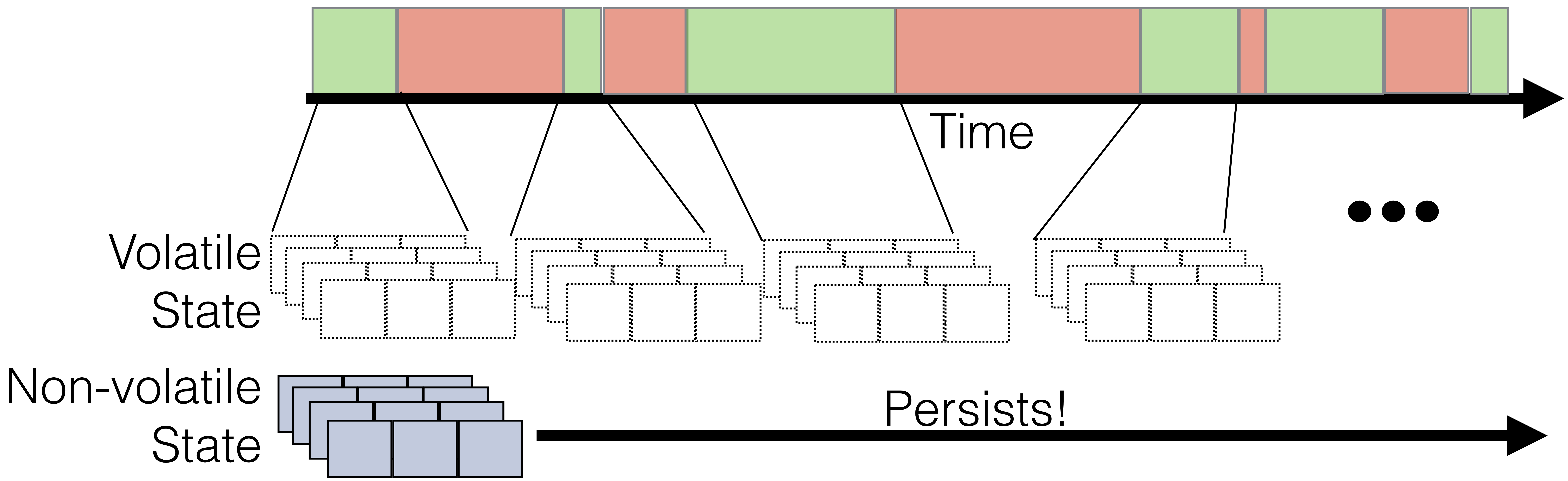
volatile memory (e.g., DRAM, SRAM registers)

Access latencies growing similar w/ new technology



non-volatile memory (e.g., Flash, FRAM)

Reboots **clear** volatile state and **preserve** non-volatile state



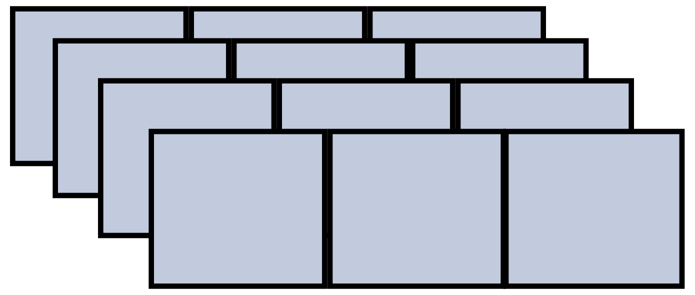
[ASPLOS ’11]

Mementos: System Support for Long-Running Computation on RFID-Scale Devices

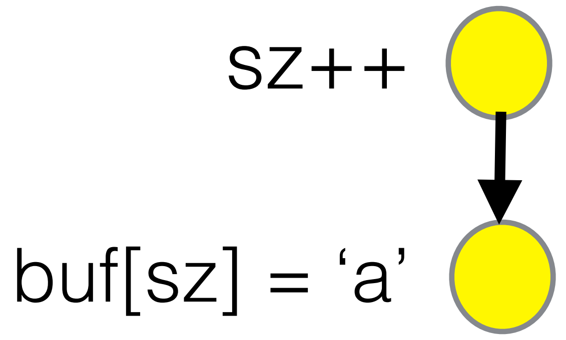
Benjamin Ransford
Department of Computer Science
University of Massachusetts Amherst
ransford@cs.umass.edu

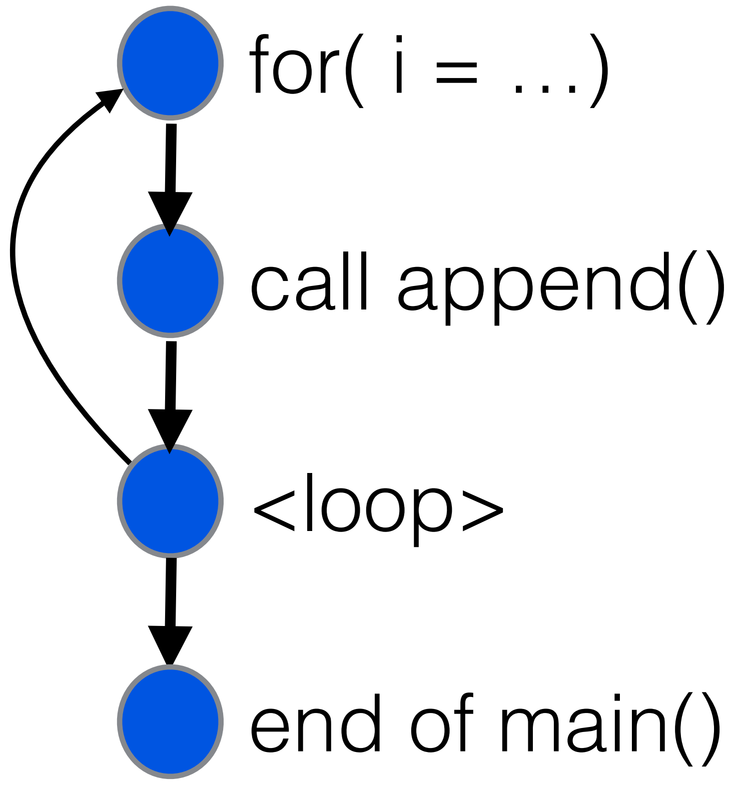
Jacob Sorber
Institute for Security, Technology, and Society
Dartmouth College
jacob.m.sorber@dartmouth.edu

Kevin Fu
Department of Computer Science
University of Massachusetts Amherst
kevinfu@cs.umass.edu



Assume non-volatile storage on device
[Flash & FRAM on TI MSP430]





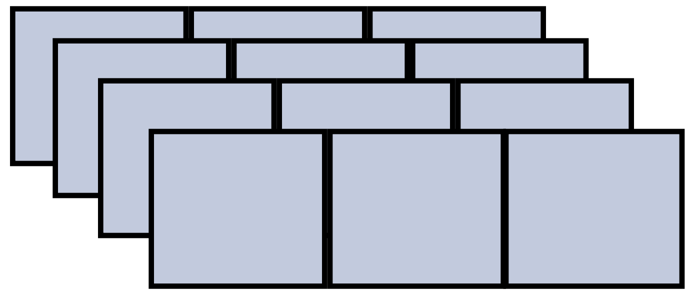
[ASPLOS '11]

Mementos: System Support for Long-Running Computation on RFID-Scale Devices

Benjamin Ransford
Department of Computer Science
University of Massachusetts Amherst
ransford@cs.umass.edu

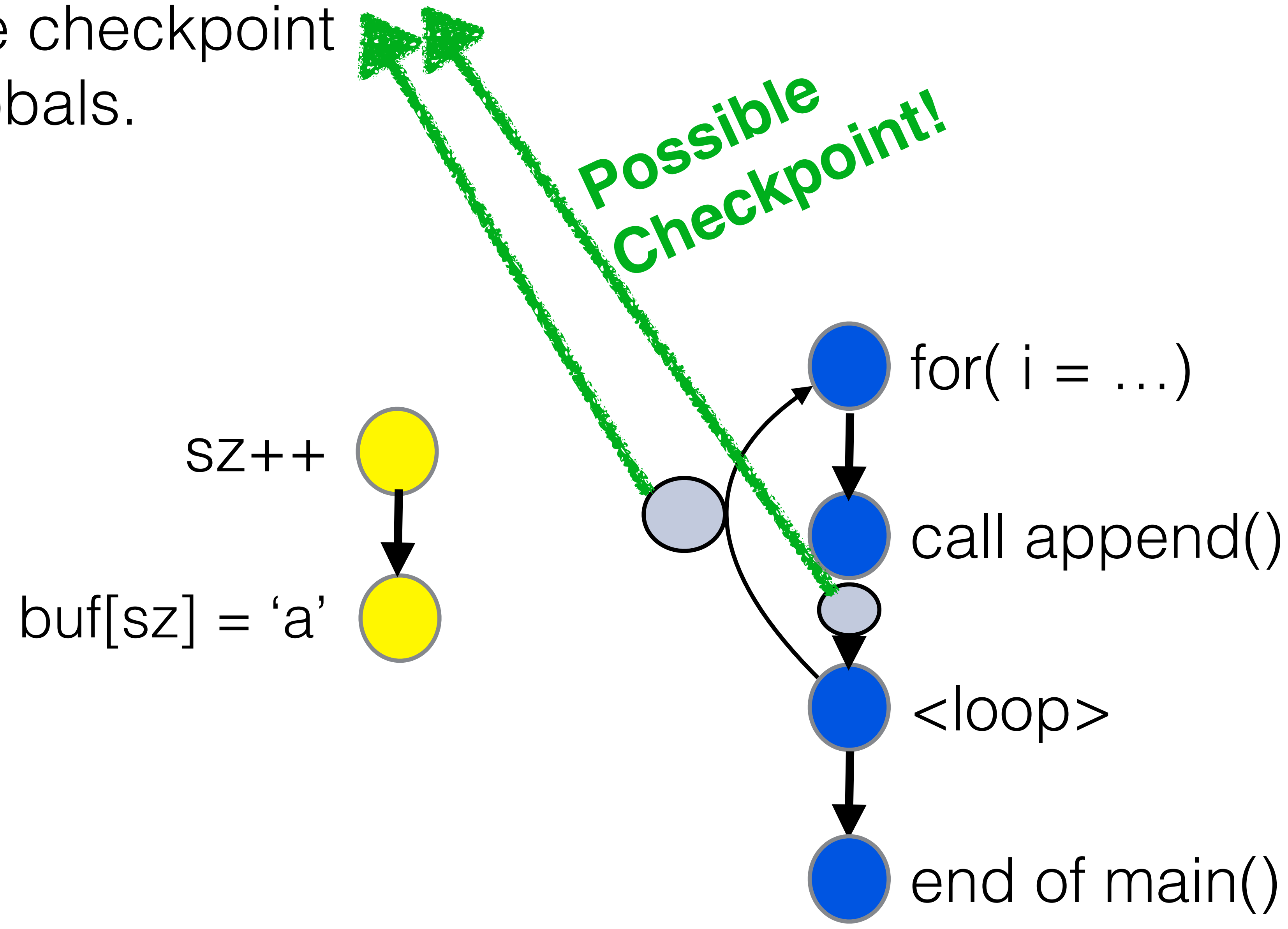
Jacob Sorber
Institute for Security, Technology, and Society
Dartmouth College
jacob.m.sorber@dartmouth.edu

Kevin Fu
Department of Computer Science
University of Massachusetts Amherst
kevinfu@cs.umass.edu



Checkpoints determined dynamically

Periodically store checkpoint of regs, stack, globals.



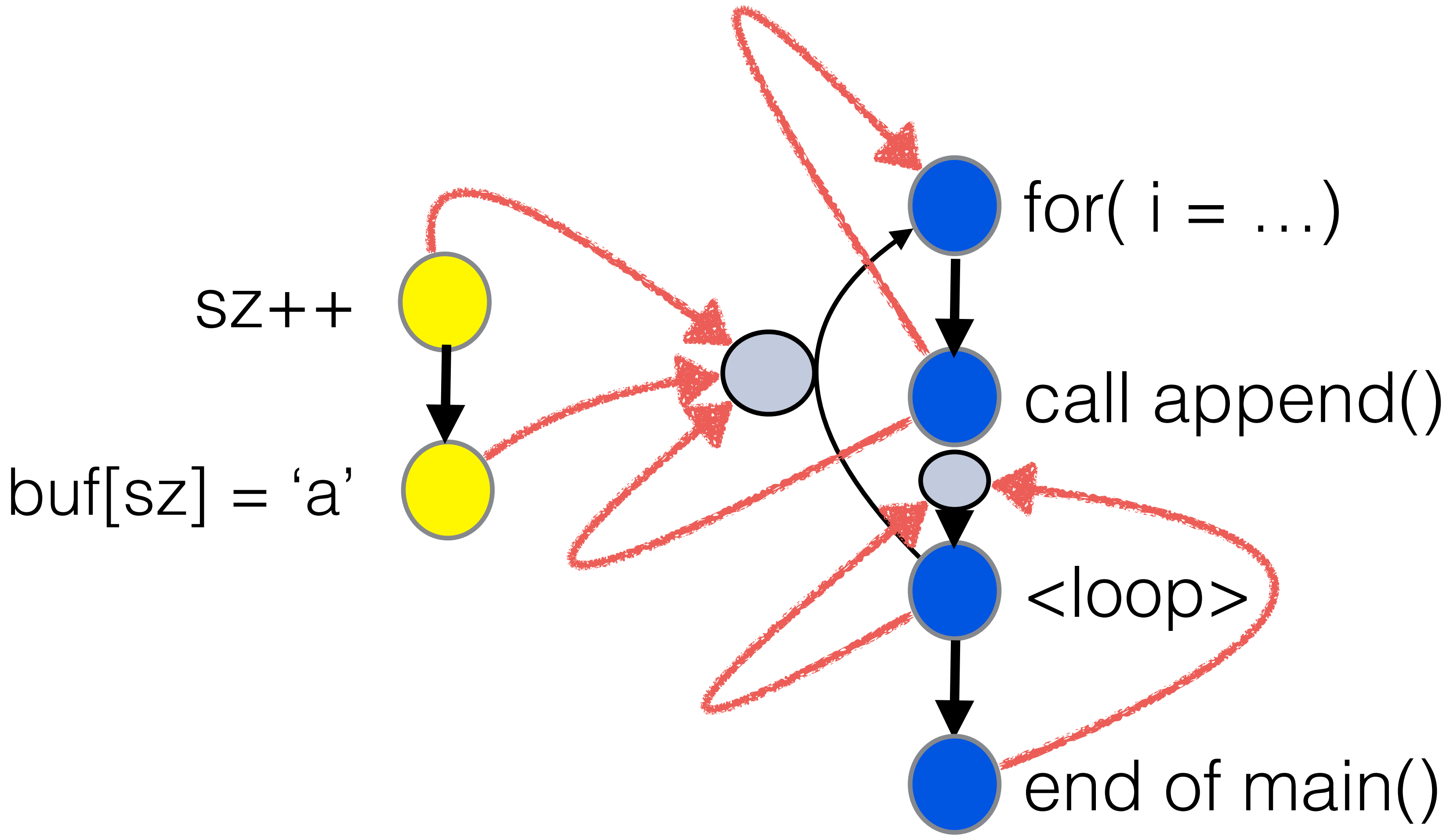
[ASPLOS '11]

Mementos: System Support for Long-Running Computation on RFID-Scale Devices

Benjamin Ransford
Department of Computer Science
University of Massachusetts Amherst
ransford@cs.umass.edu

Jacob Sorber
Institute for Security, Technology, and Society
Dartmouth College
jacob.m.sorber@dartmouth.edu

Kevin Fu
Department of Computer Science
University of Massachusetts Amherst
kevinfu@cs.umass.edu

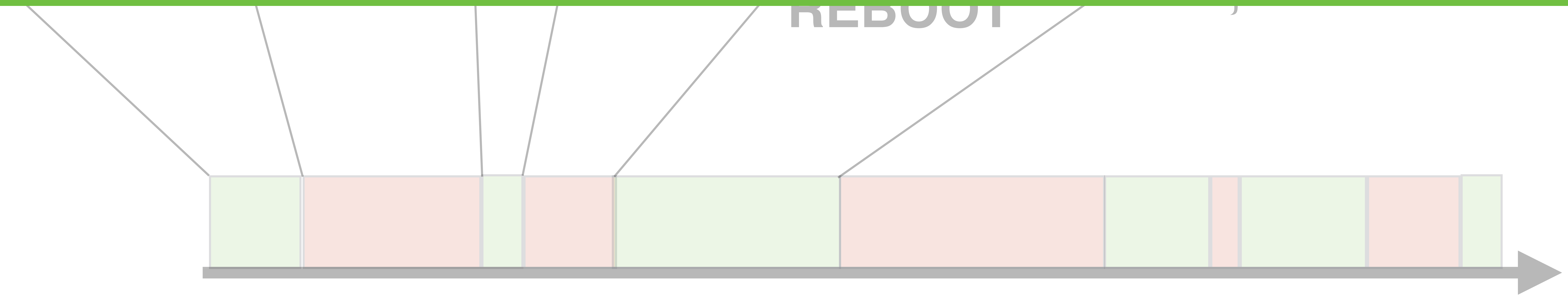


Still have Intermittent Execution Challenge #1

(Implicit, non-local control-flow)

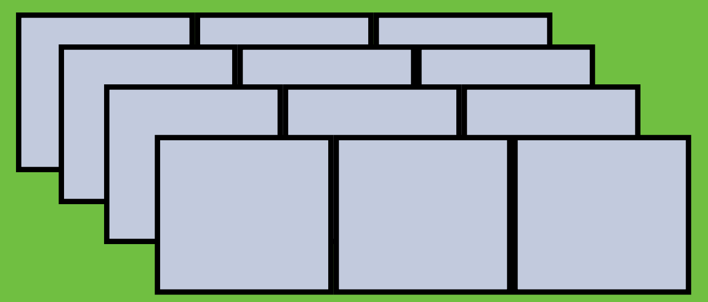
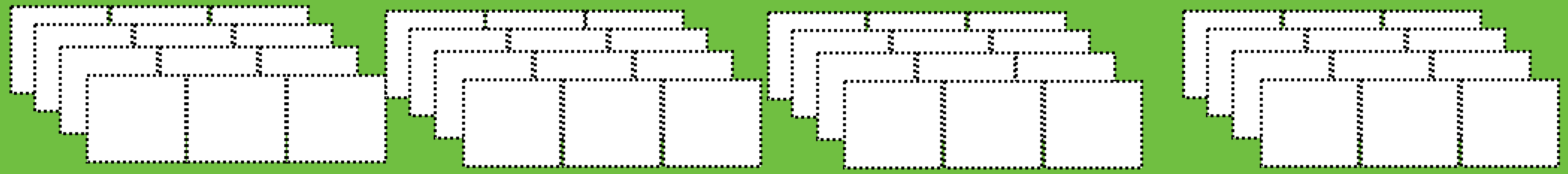
The Big Idea

The way we think about programs does not match the intermittent execution model



The Big Idea

The way we think about programs does not match the intermittent execution model

Especially when we combine non-volatile  and volatile memory! 

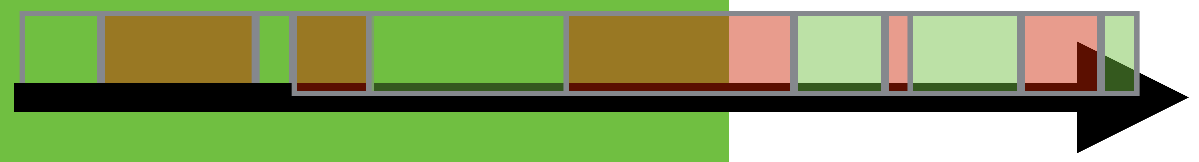
The Intermittent Execution Model

Thinking About Intermittently-powered Devices

Intermittence & Data Consistency

Hidden Problems Caused by Intermittence

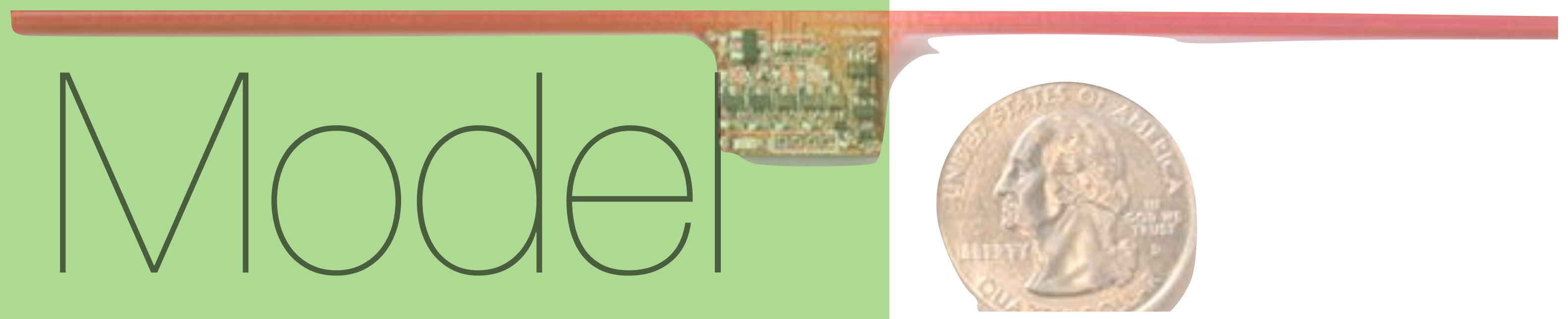
Designing for Intermittence



Strategies for Coping with Intermittence

The Intermittent Execution Model

Thinking About Intermittently-powered Devices

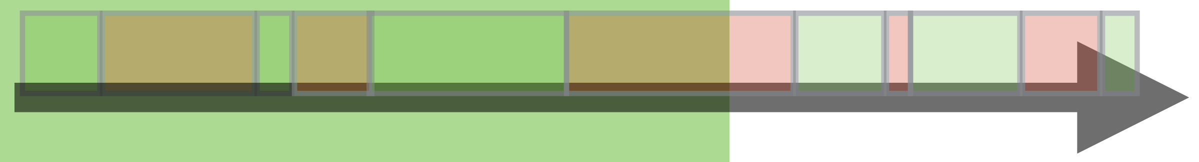


Intermittence & Data Consistency

Hidden Problems Caused by Intermittence

Designing for Intermittence

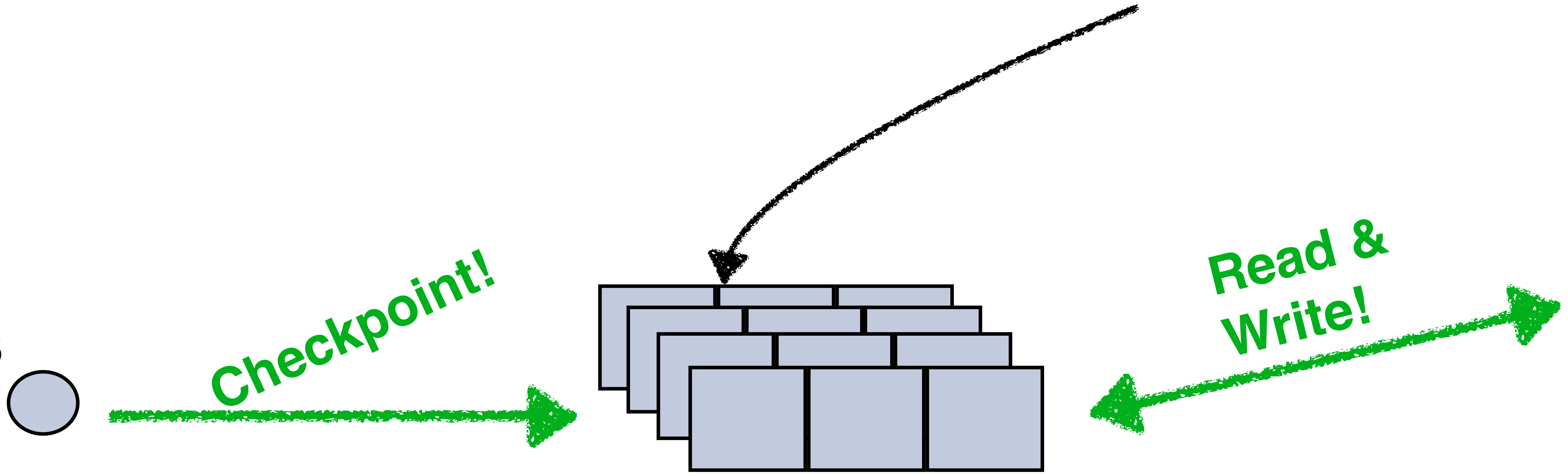
Strategies for Coping with Intermittence



Why does **Intermittent Execution Challenge #1** matter?

Can put `sz` & `buf` in Non-Volatile storage

registers,
stack vars,
global vars



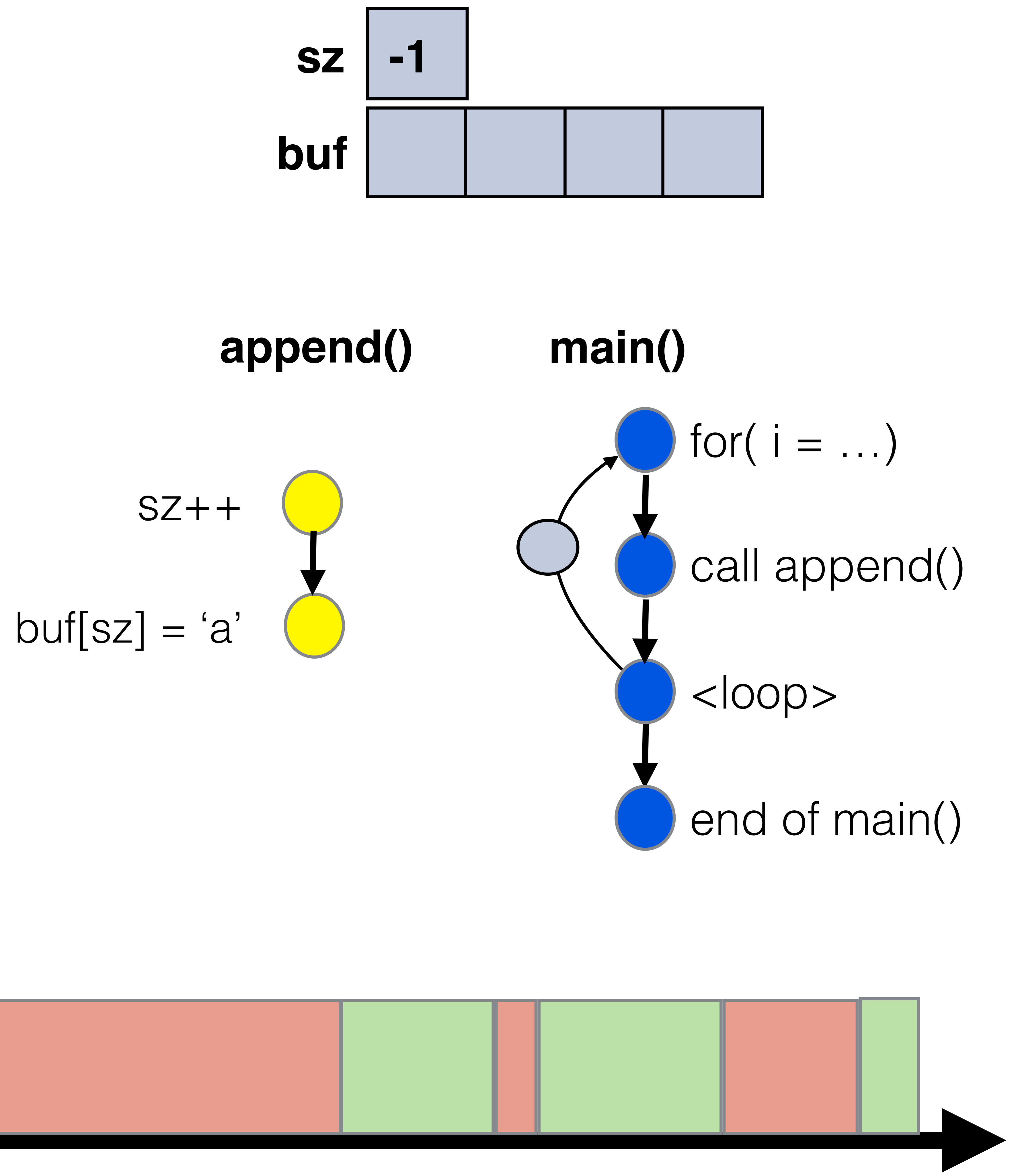

```
NV sz, buf;
void append(){
   sz++
  buf[sz] = 'a'
}
```

Non-volatile storage records checkpoints periodically!

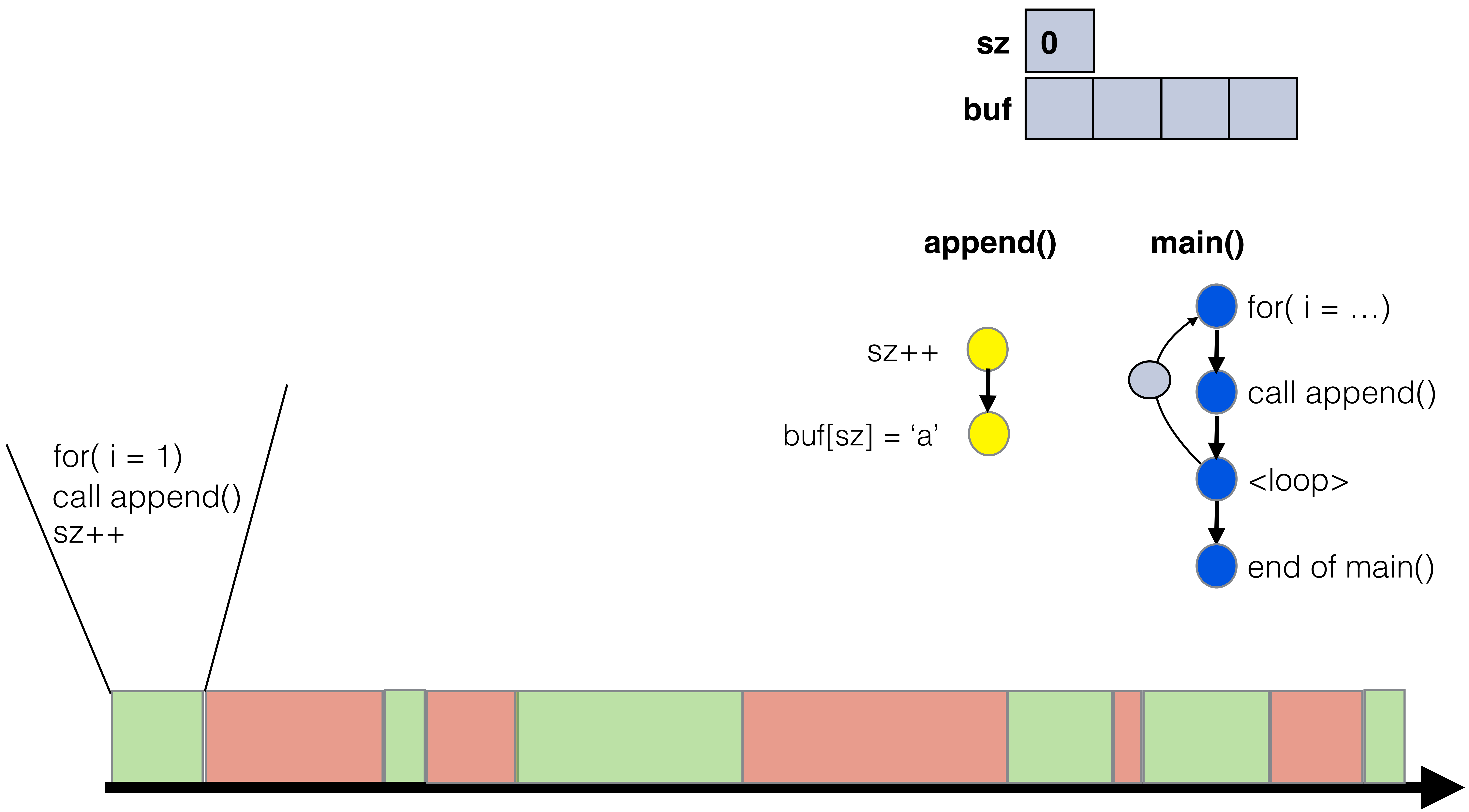
Non-volatile storage is continuously accessible in user code.

IEC#1 + Non-Volatile Storage = Data Consistency Violations!

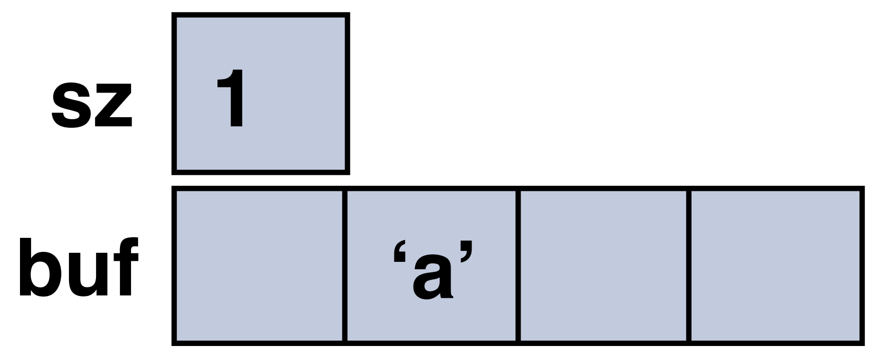
Intermittent Execution Challenge #2: Hidden Atomicity Violations



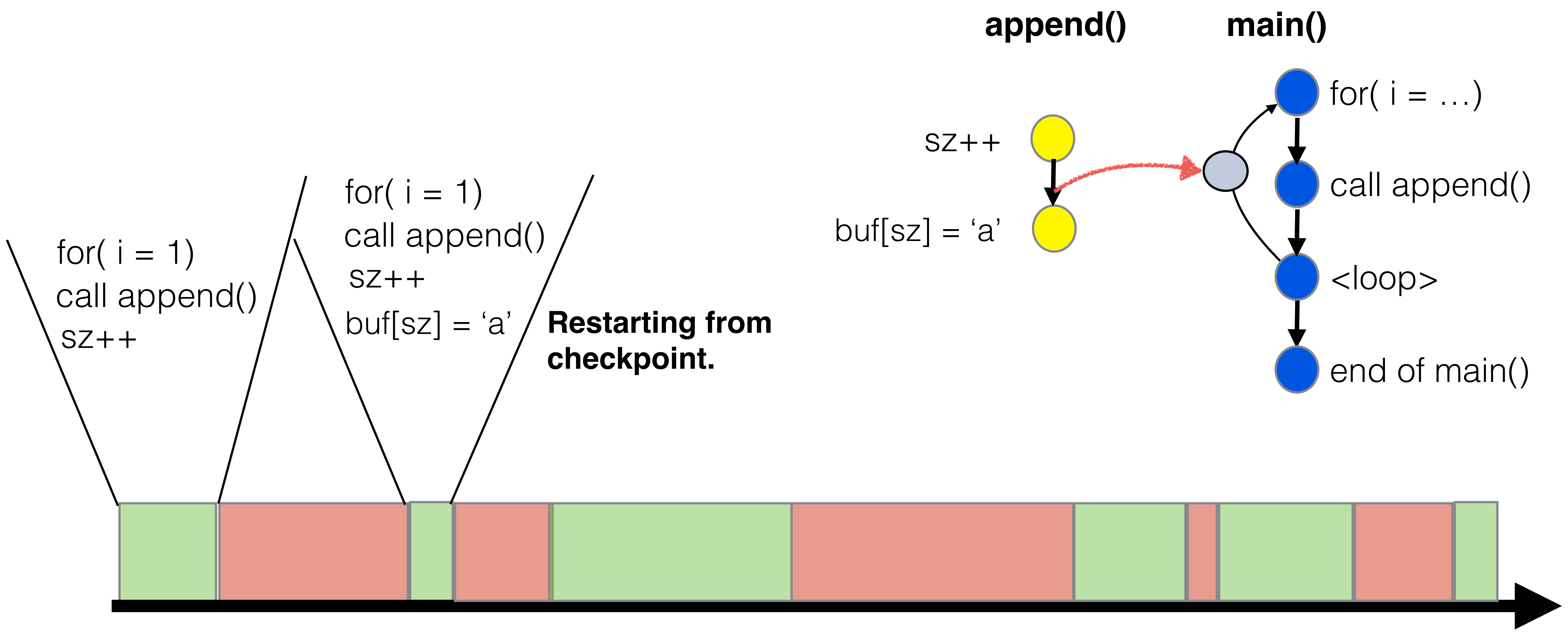
Intermittent Execution Challenge #2: Hidden Atomicity Violations



Intermittent Execution Challenge #2: Hidden Atomicity Violations

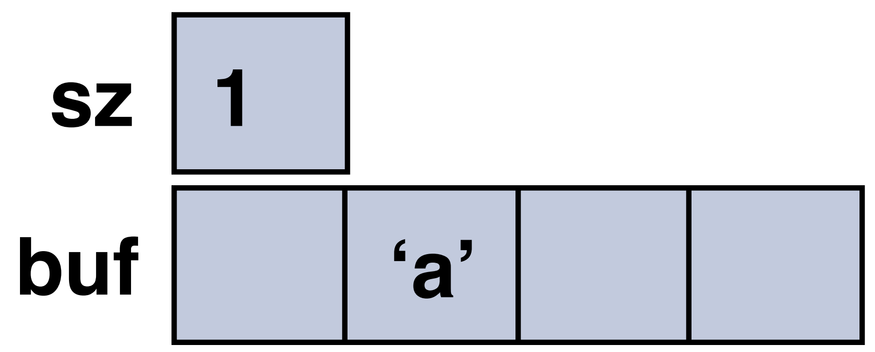


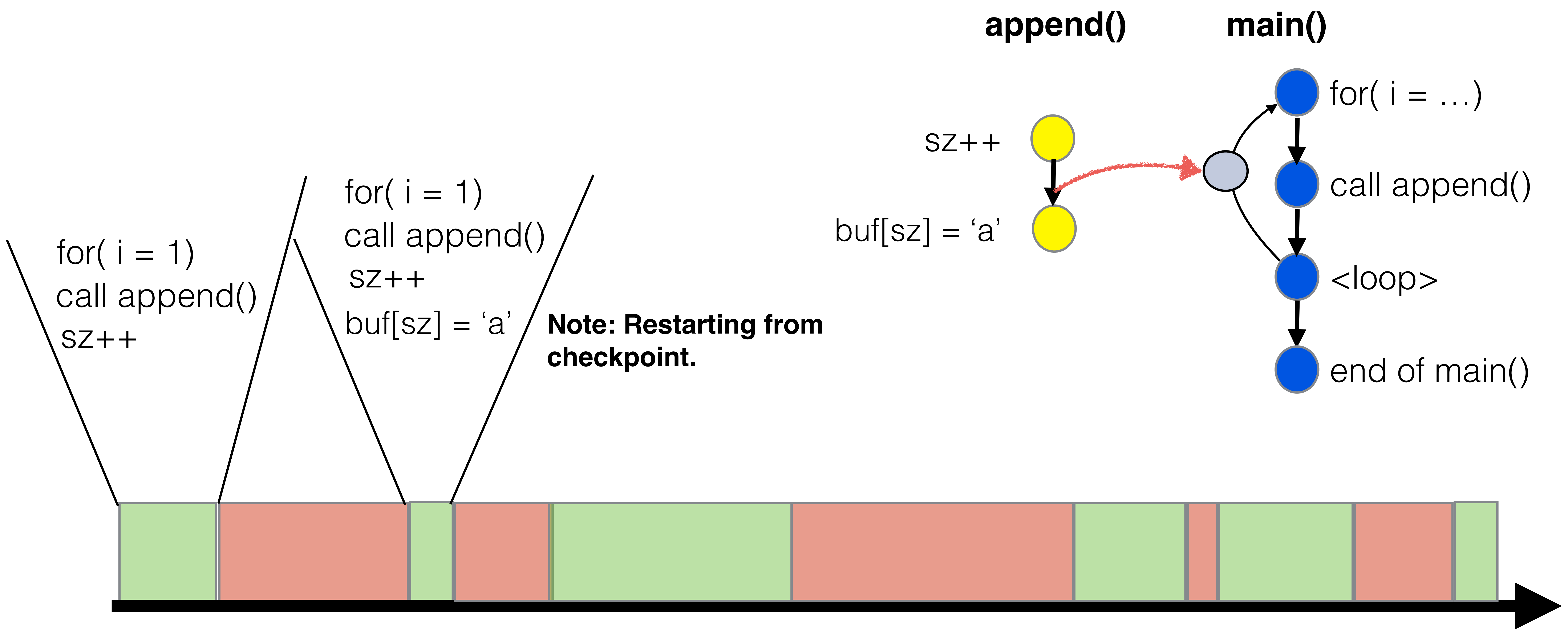
Error: ’a’ is appended to the wrong entry in **buf**



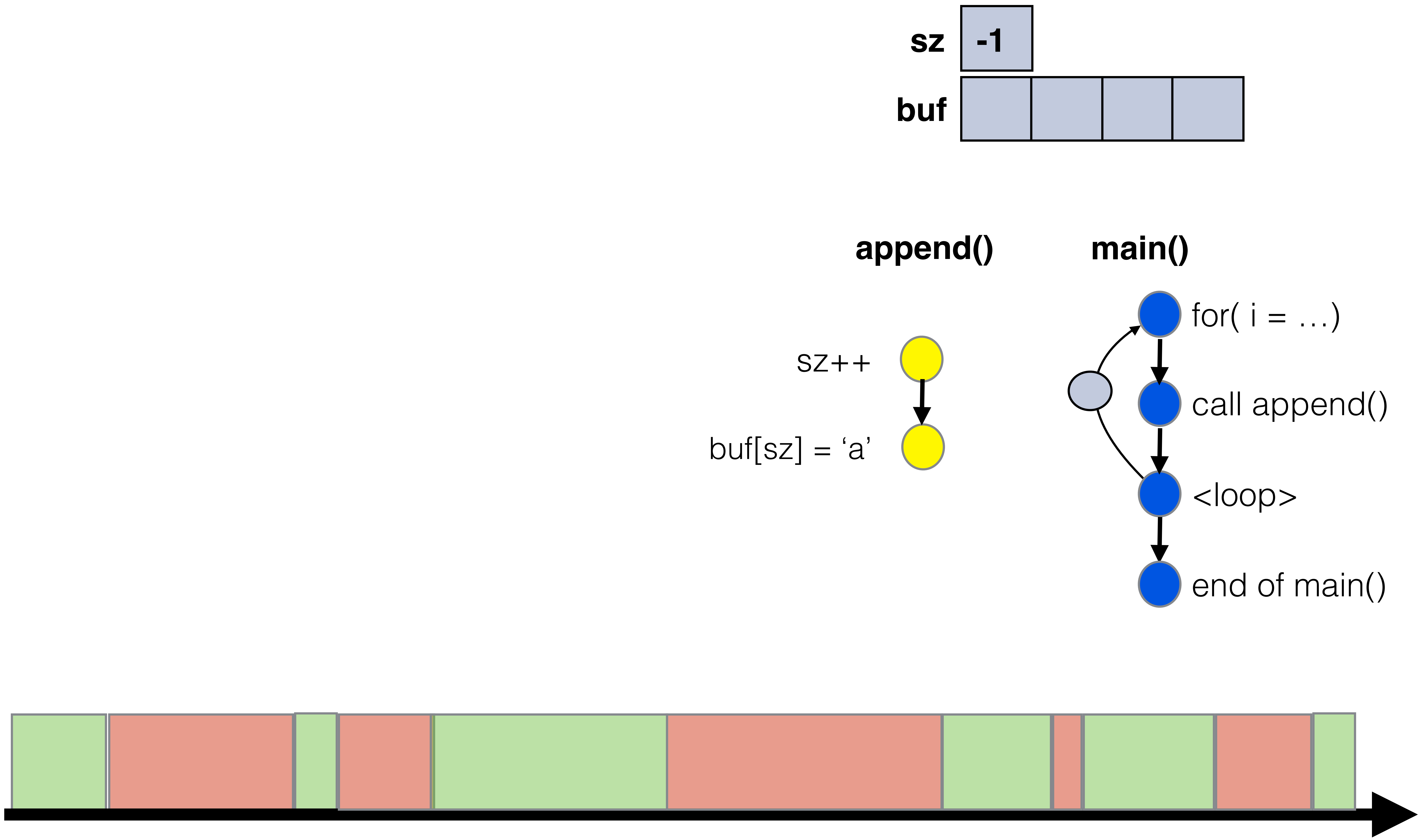
Intermittent Execution Challenge #2: Hidden Atomicity Violations

Atomicity Violation leaves **sz** and **buf** mutually inconsistent

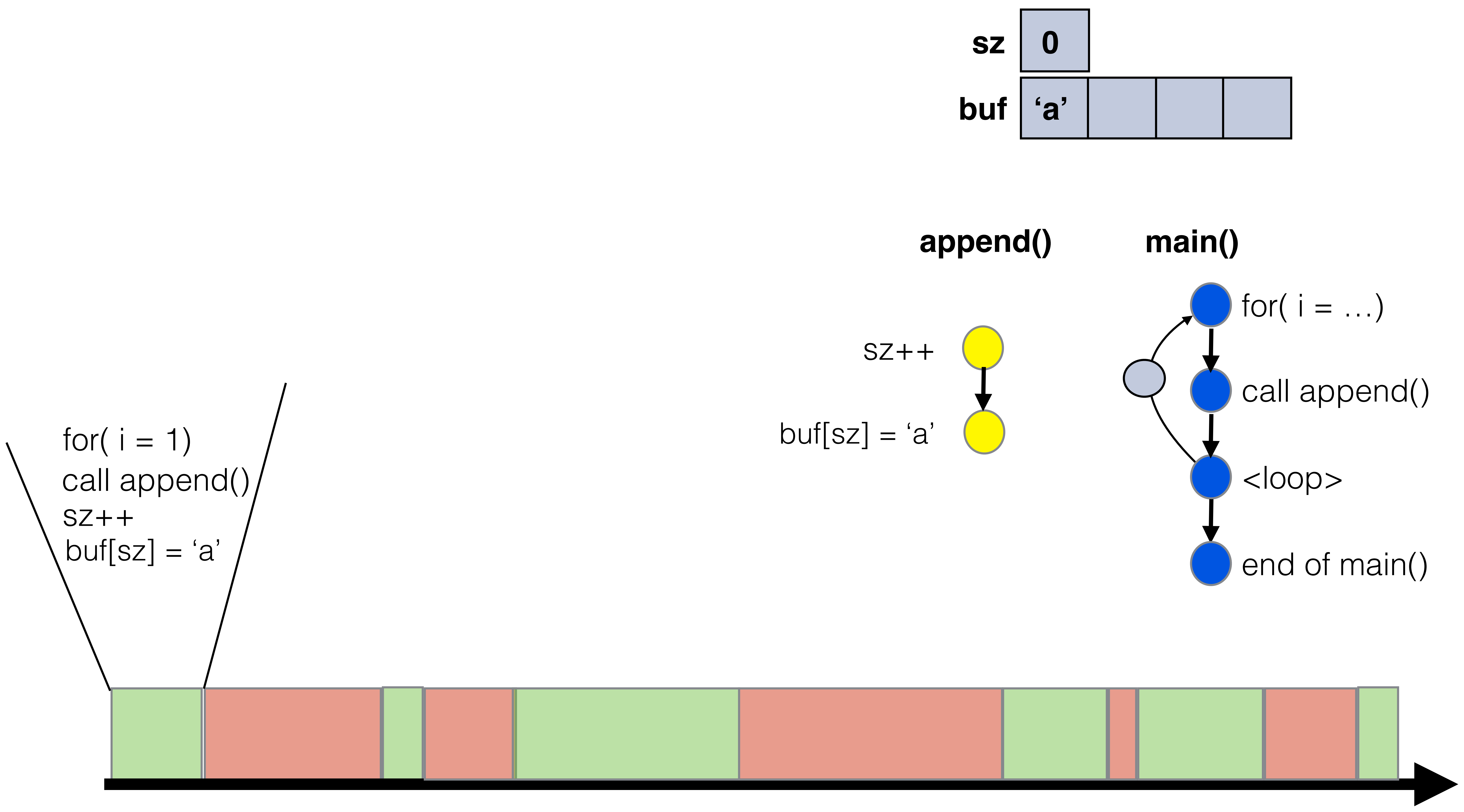




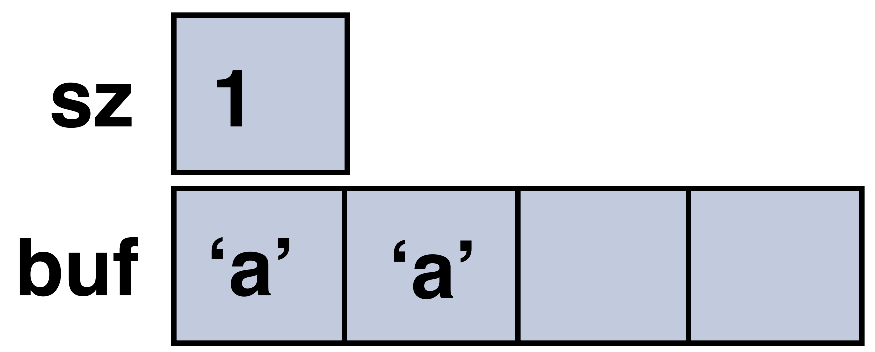
Intermittent Execution Challenge #3: Hidden Loops (“Idempotence Violations”)



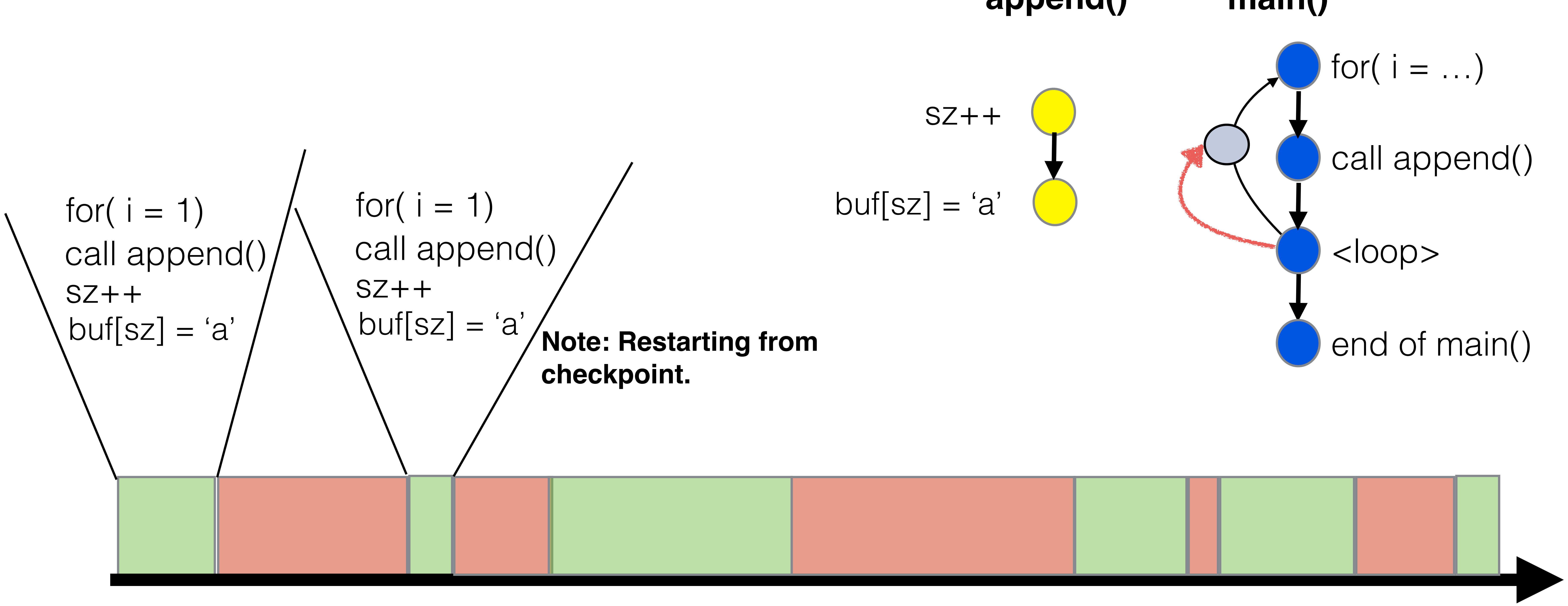
Intermittent Execution Challenge #3: Hidden Loops



Intermittent Execution Challenge #3: Hidden Loops (“Idempotence Violations”)

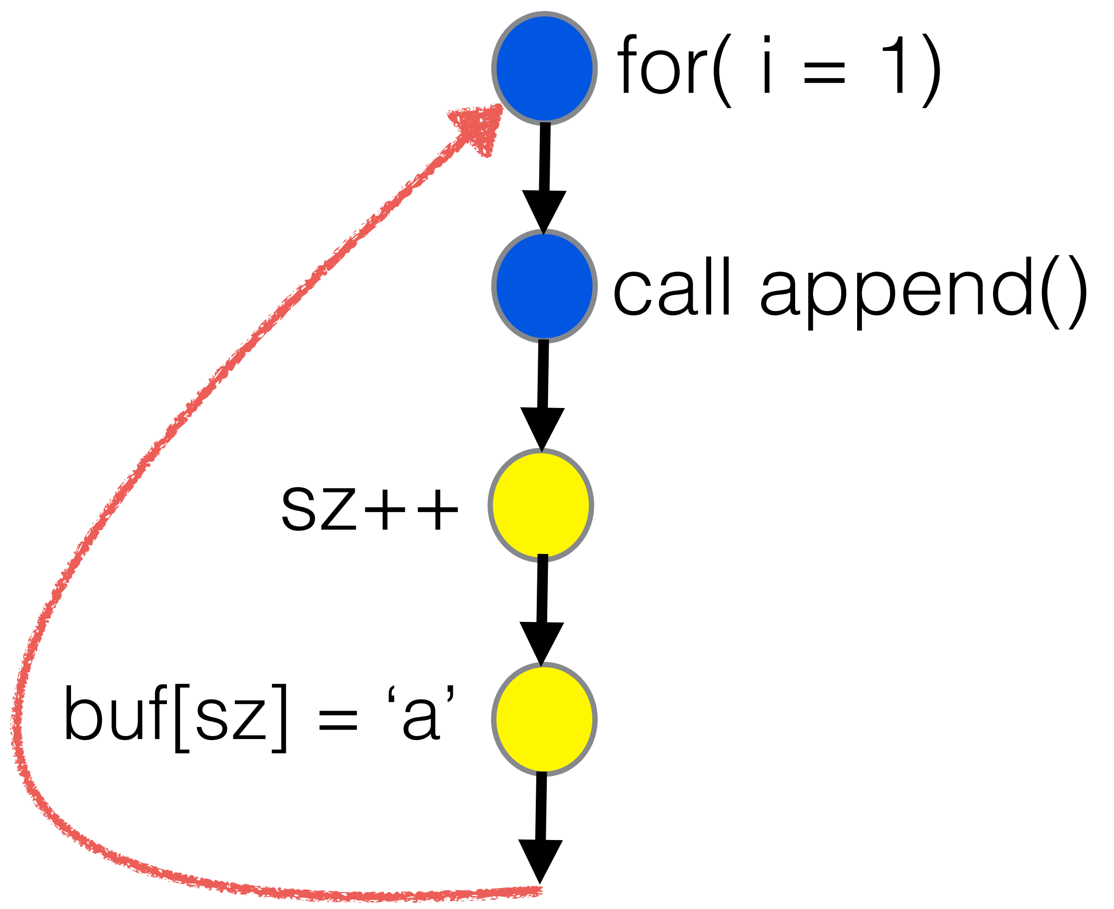


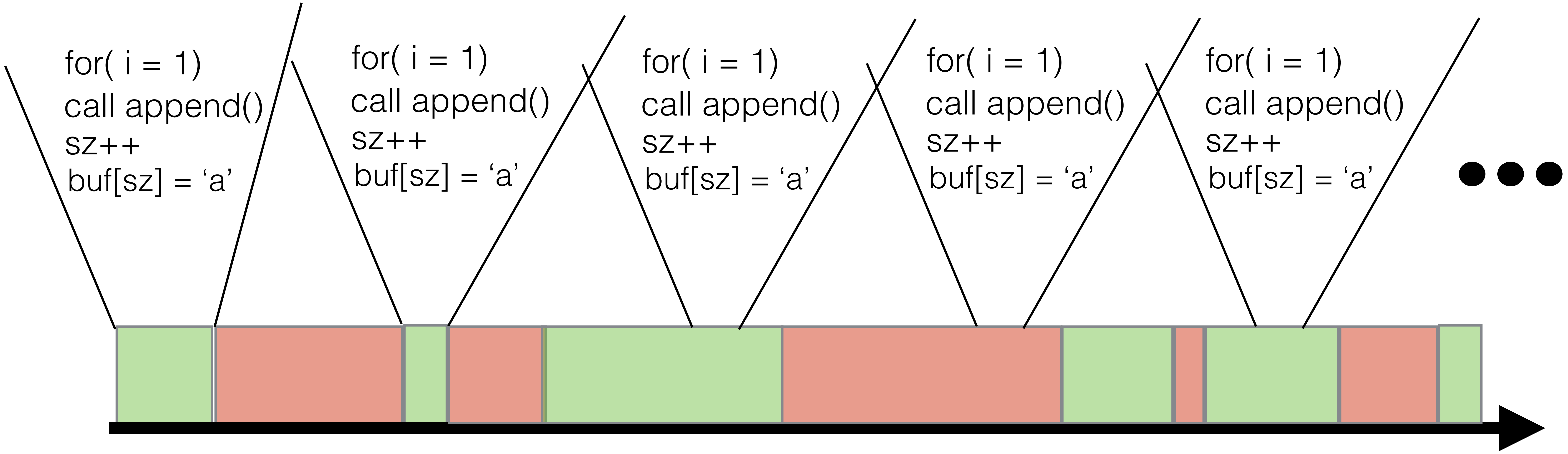
Error: ’a’ is appended to **buf** twice on the **i = 1** iteration



Intermittent Execution Challenge #3: Hidden Loops (“Idempotence Violations”)

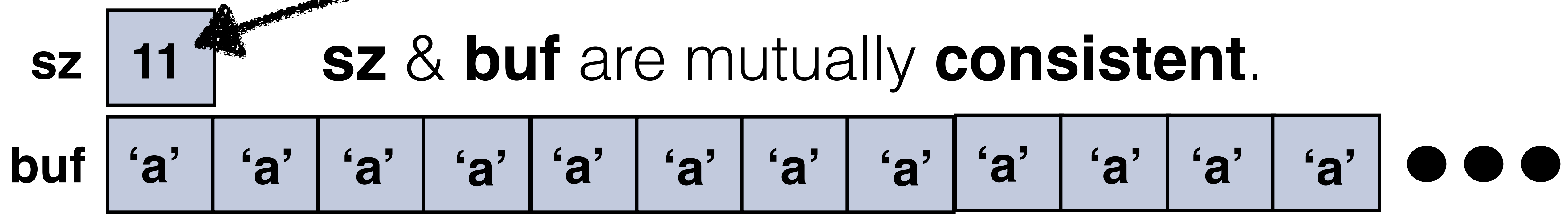
Stuck in an implicit loop for **i = 1**





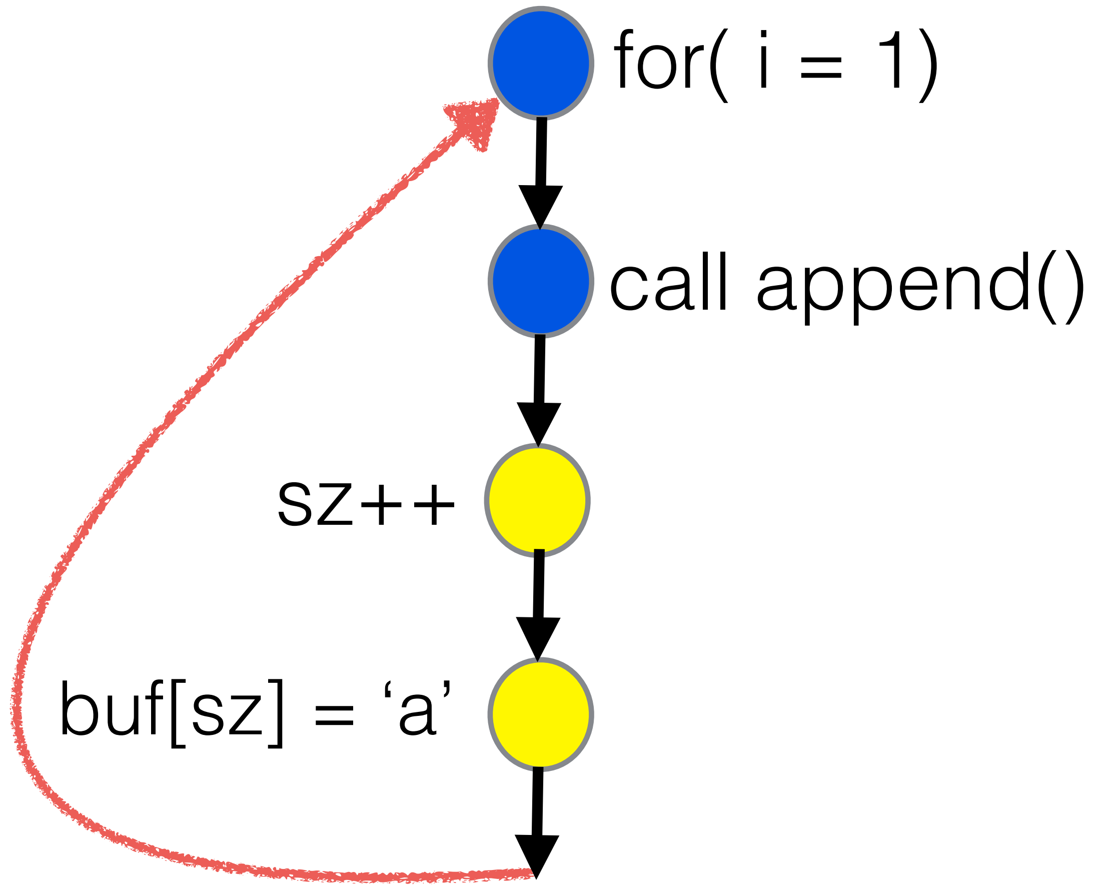
Intermittent Execution Challenge #3: Hidden Loops ("Idempotence Violations")

Value not permitted by any continuous execution!



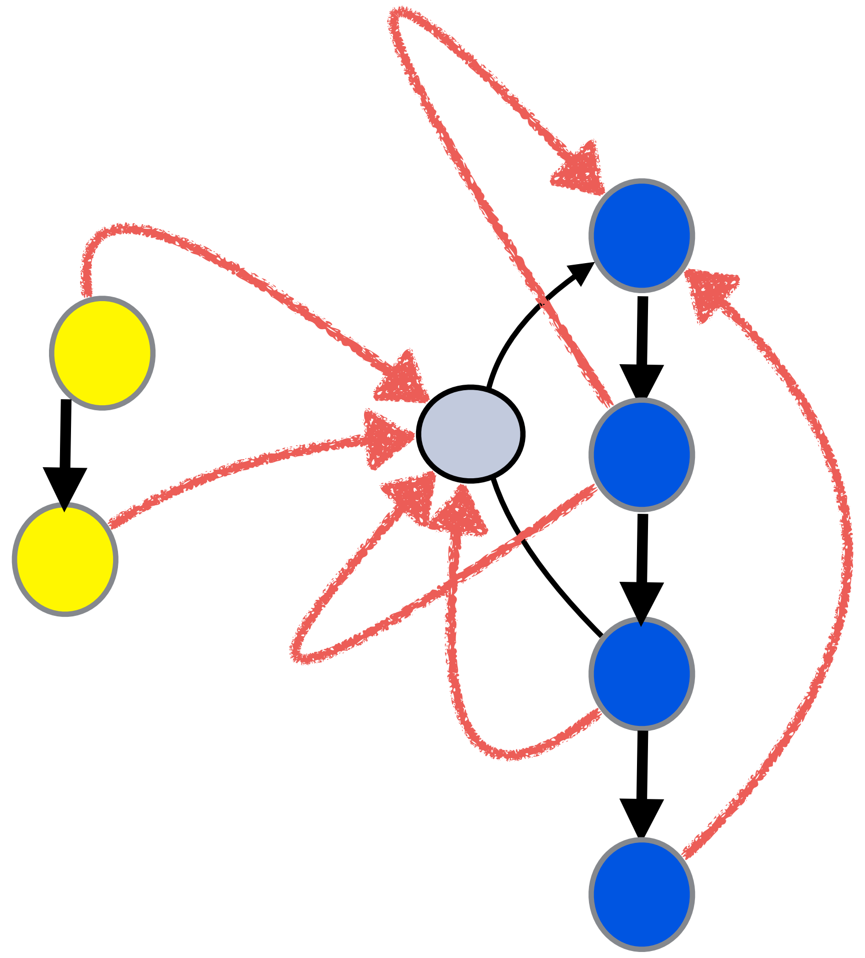
sz & **buf** are mutually **consistent**.

Stuck in an implicit loop for **i = 1**

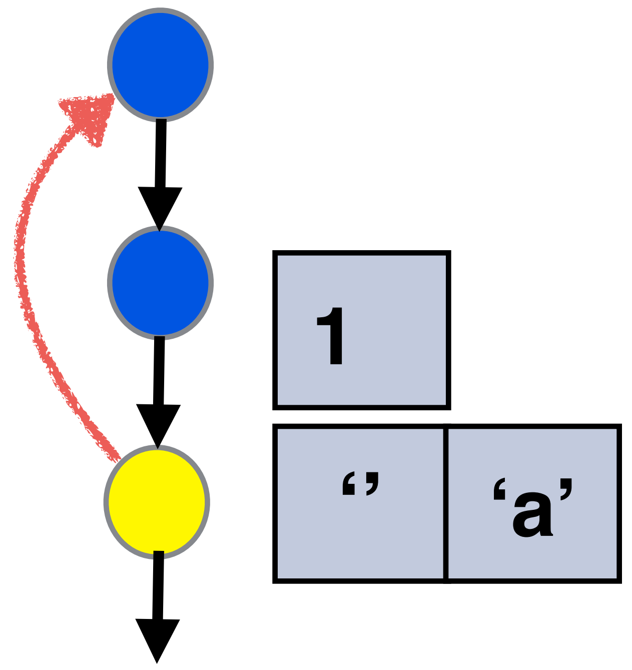


The data are inconsistent with the **checkpointed execution context**

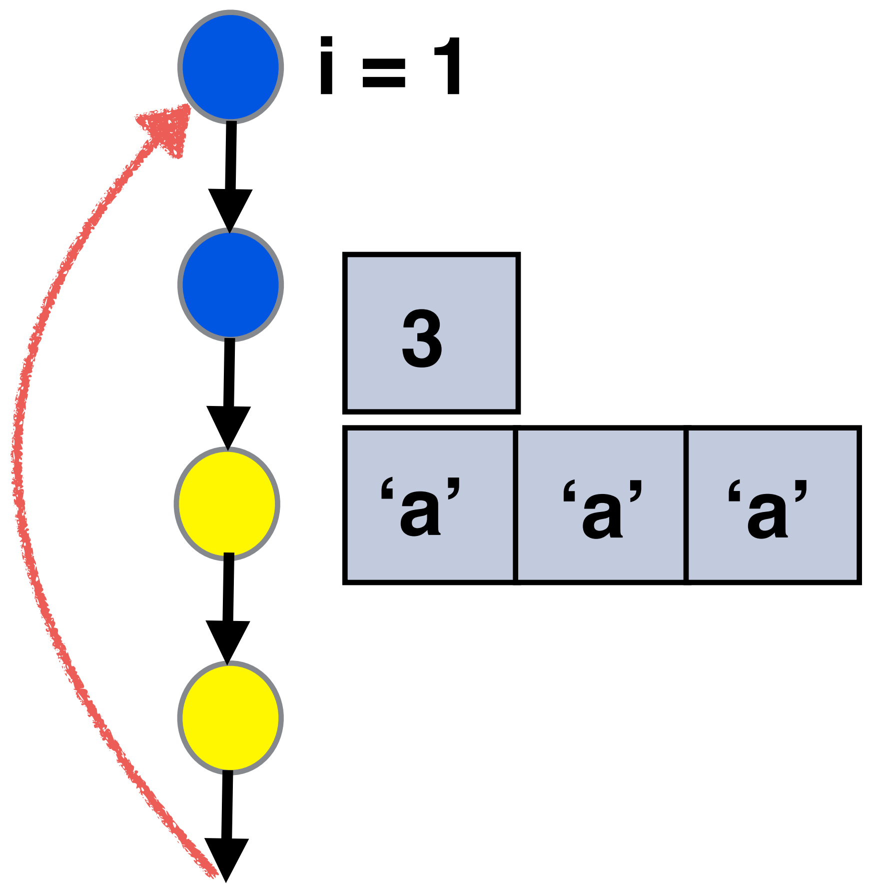
Intermittent Execution Challenge #1:
Implicit, non-local control flow



Intermittent Execution Challenge #2:
Hidden Atomicity Violations



Intermittent Execution Challenge #3:
Hidden Loops (“Idempotence Violations”)



The Intermittent Execution Model

Thinking About Intermittently-powered Devices

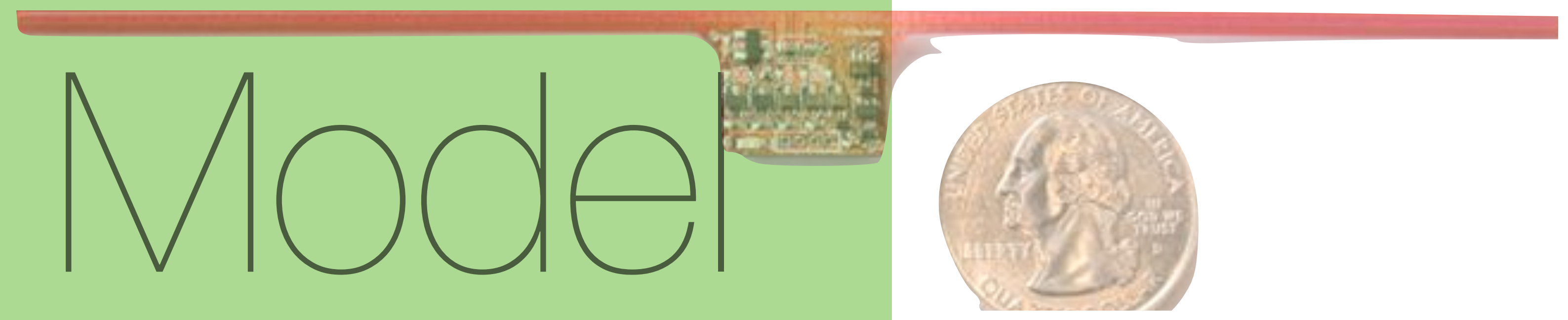
Intermittence & Data Consistency

Hidden Problems Caused by Intermittence

Designing for Intermittence

Strategies for Coping with Intermittence

The Intermittent Execution Model

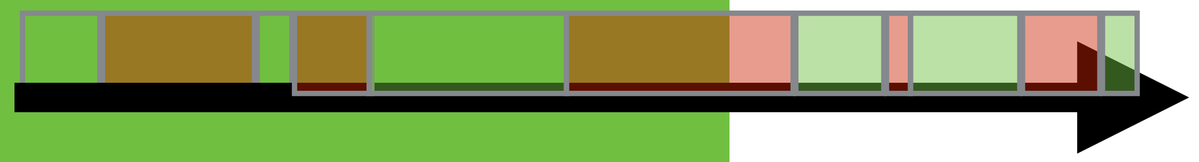


Thinking About Intermittently-powered Devices

Intermittence & Data Consistency

Hidden Problems Caused by Intermittence

Designing for Intermittence



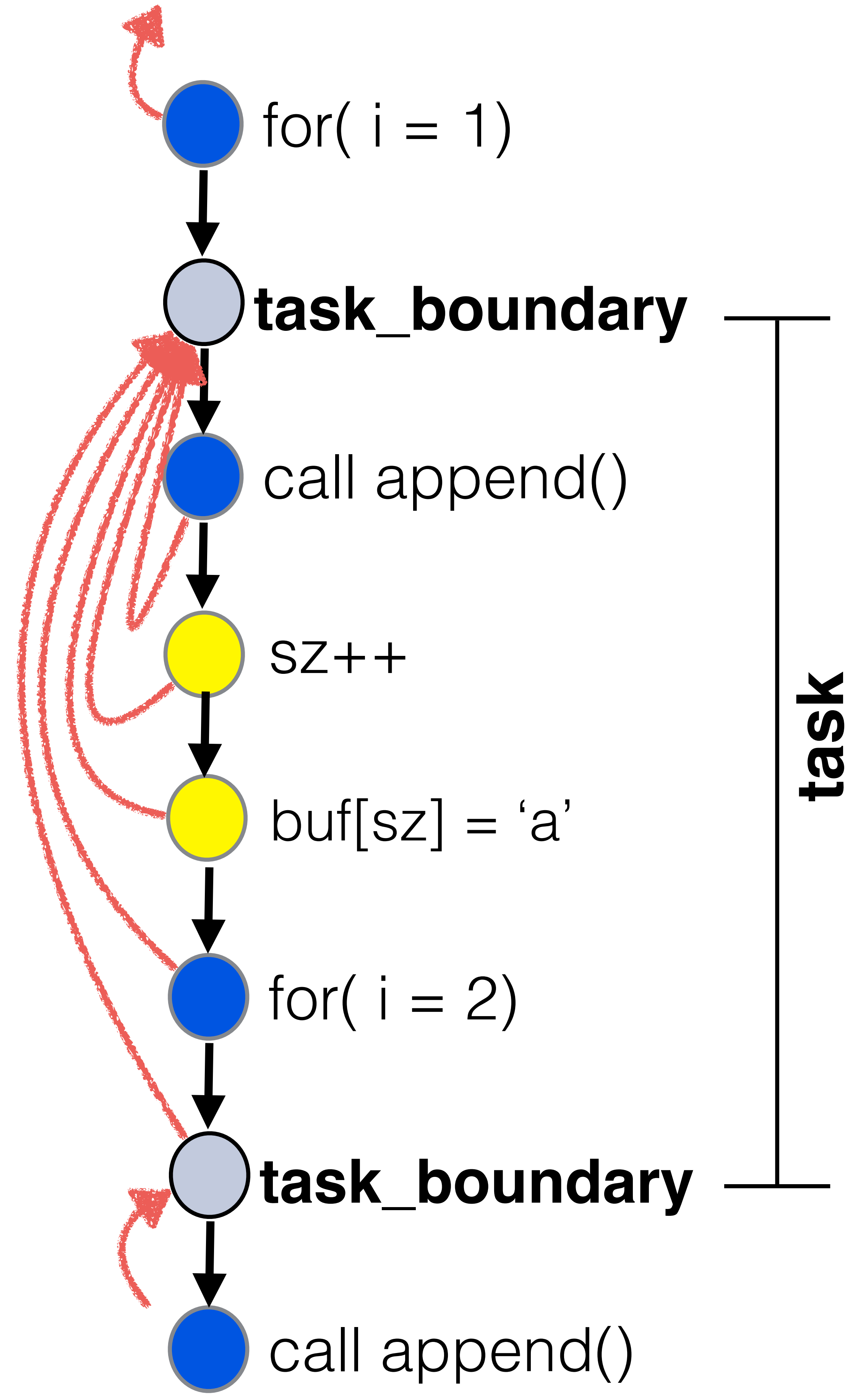
Strategies for Coping with Intermittence

```
void main(void){
  for( i = 1 .. 10)
 task_boundary()
    append()
}

void append(){
  sz++
  buf[sz] = 'a'
}
```

Statically defined task boundaries.

Dynamically defined tasks

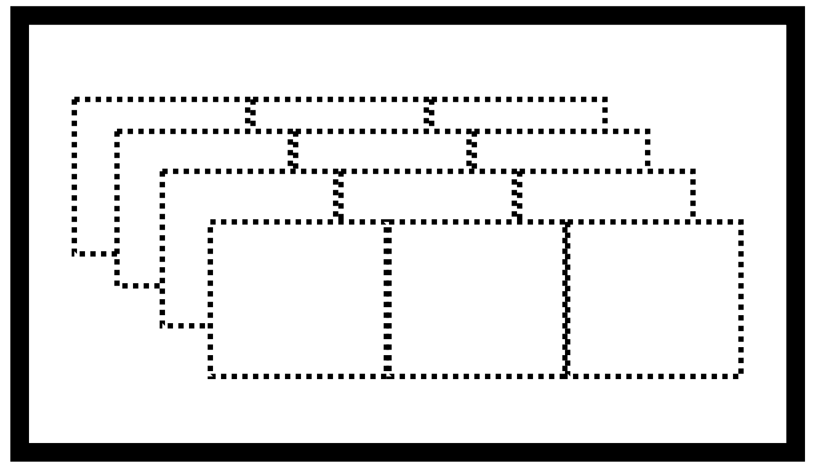
Task boundaries are **checkpoints**

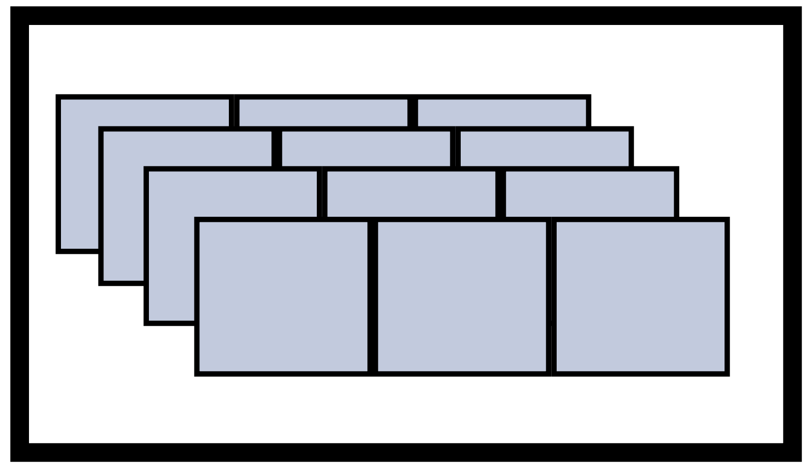


Key Advantage: Target of failure-induced flows now explicit in code!

Task-based Programming Makes Implicit Flow Explicit

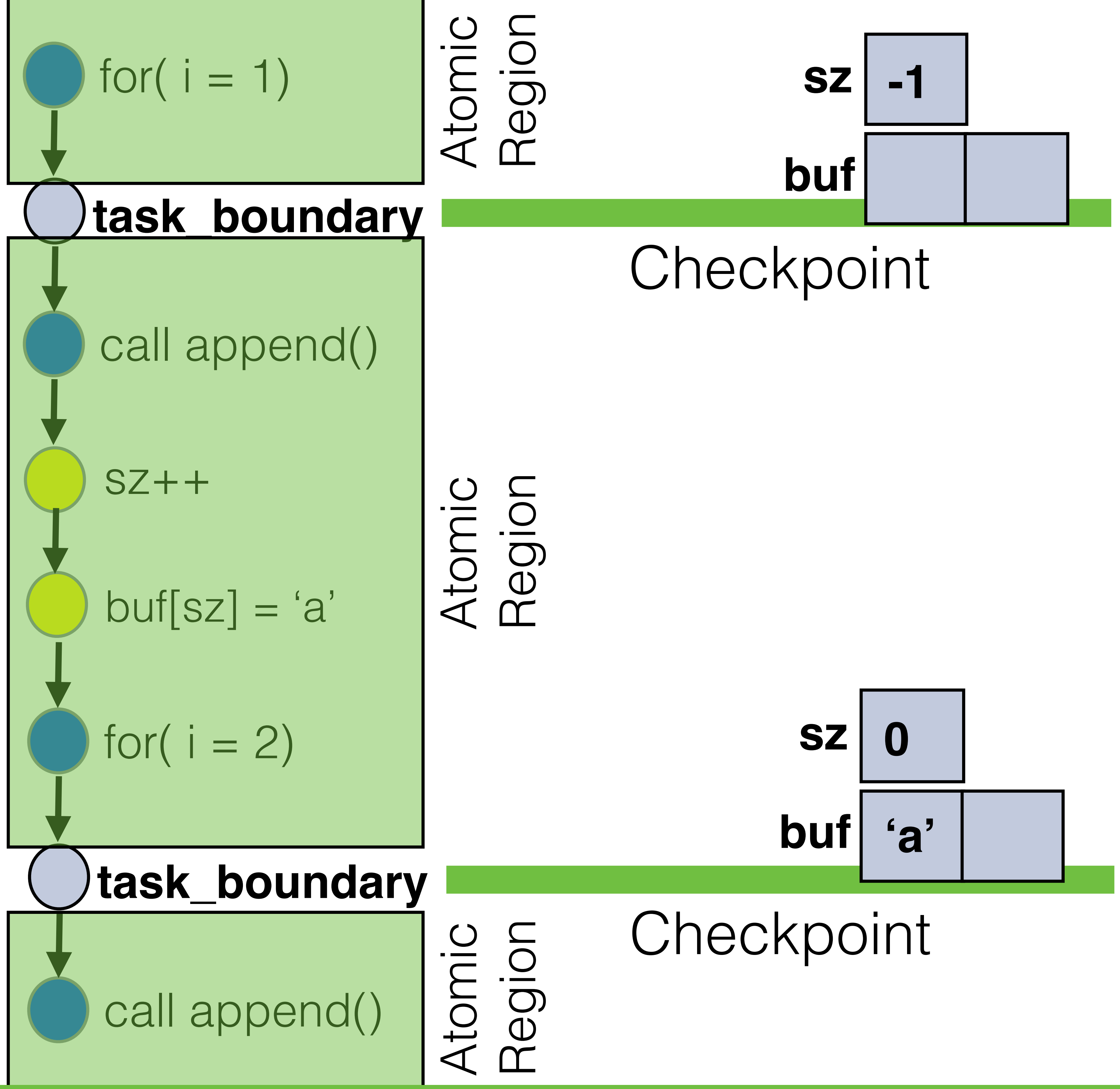
Checkpoint volatile state



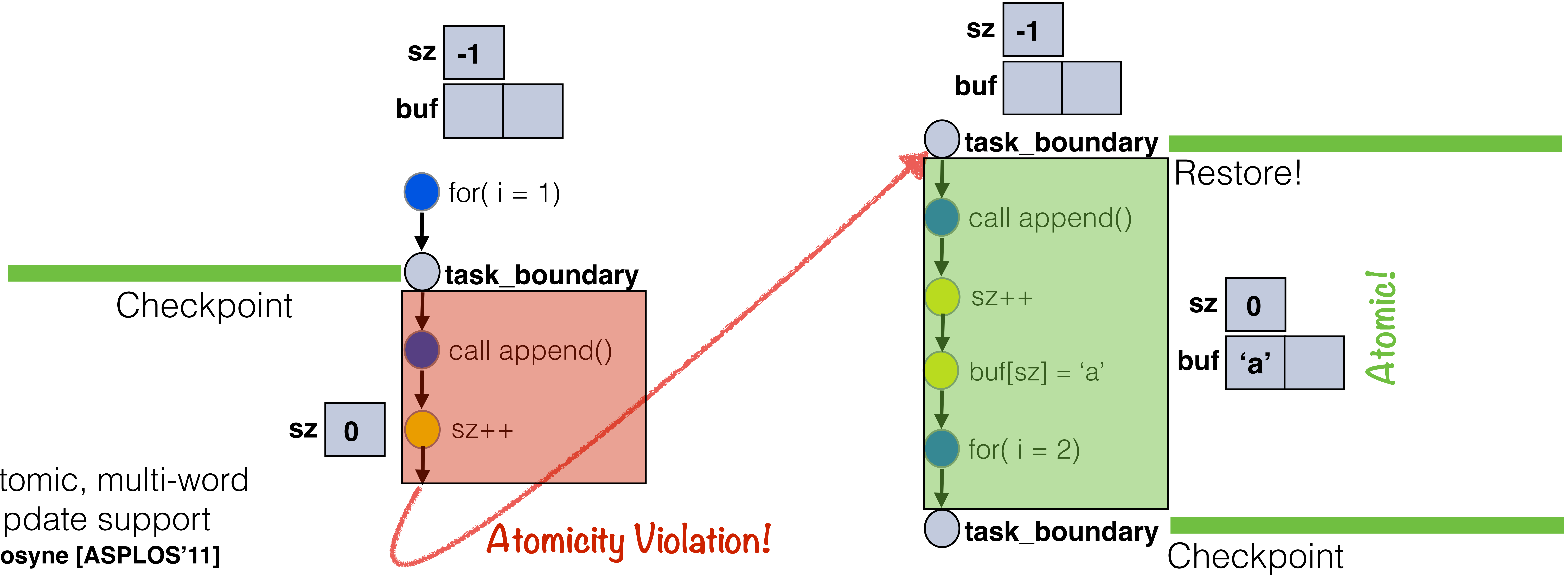


Version non-volatile state

Key Advantage: task-atomicity w.r.t. **volatile** and **non-volatile** data



Task-atomic Execution Model

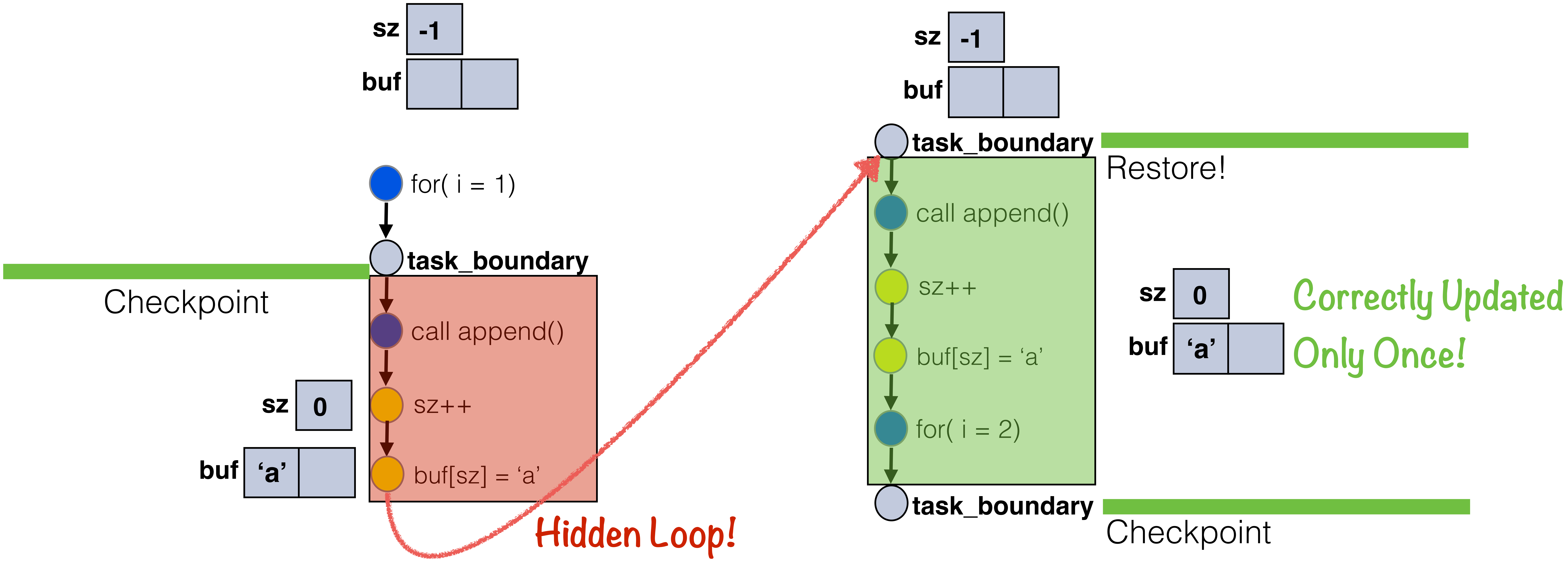


cf. Atomic, multi-word
NV update support

Mnemosyne [ASPLOS'11]

NV-Heaps [ASPLOS'11]

Eliminates Hidden Atomicity Violations!



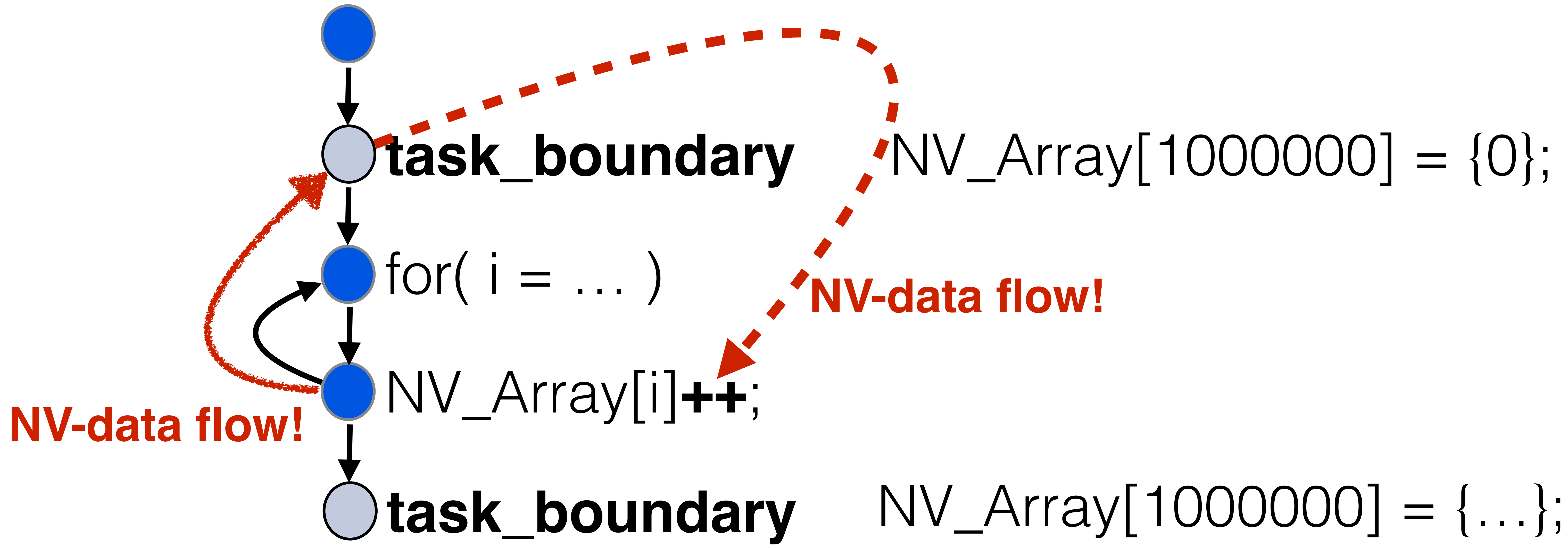
Eliminates Hidden Loops, Too!

```
int NV_Array[1000000];
task_boundary
for( i = 0; i < 1000000; i++){

  NV_Array[i]++;

}
task_boundary
```

What non-volatile data do we need to version here?

Selective Versioning with Data-flow Analysis

```
int NV_Array[1000000];
task_boundary
for( i = 0; i < 1000000; i++){

  NV_Array[i]++;

}
task_boundary
```



Insight: Must version data! Task **reads & writes** same NV data

Selective Versioning with Data-flow Analysis

Task-based Programming Model

```
int NV_Array[1000000];
task_boundary
for( i = 0; i < 1000000; i++){

  NV_Array[i] = i;

}
task_boundary
```

Task-aware Compiler

LLVM COMPILER INFRASTRUCTURE

Task Data-flow Analysis

Link to Task Runtime

Task-atomic Runtime Library

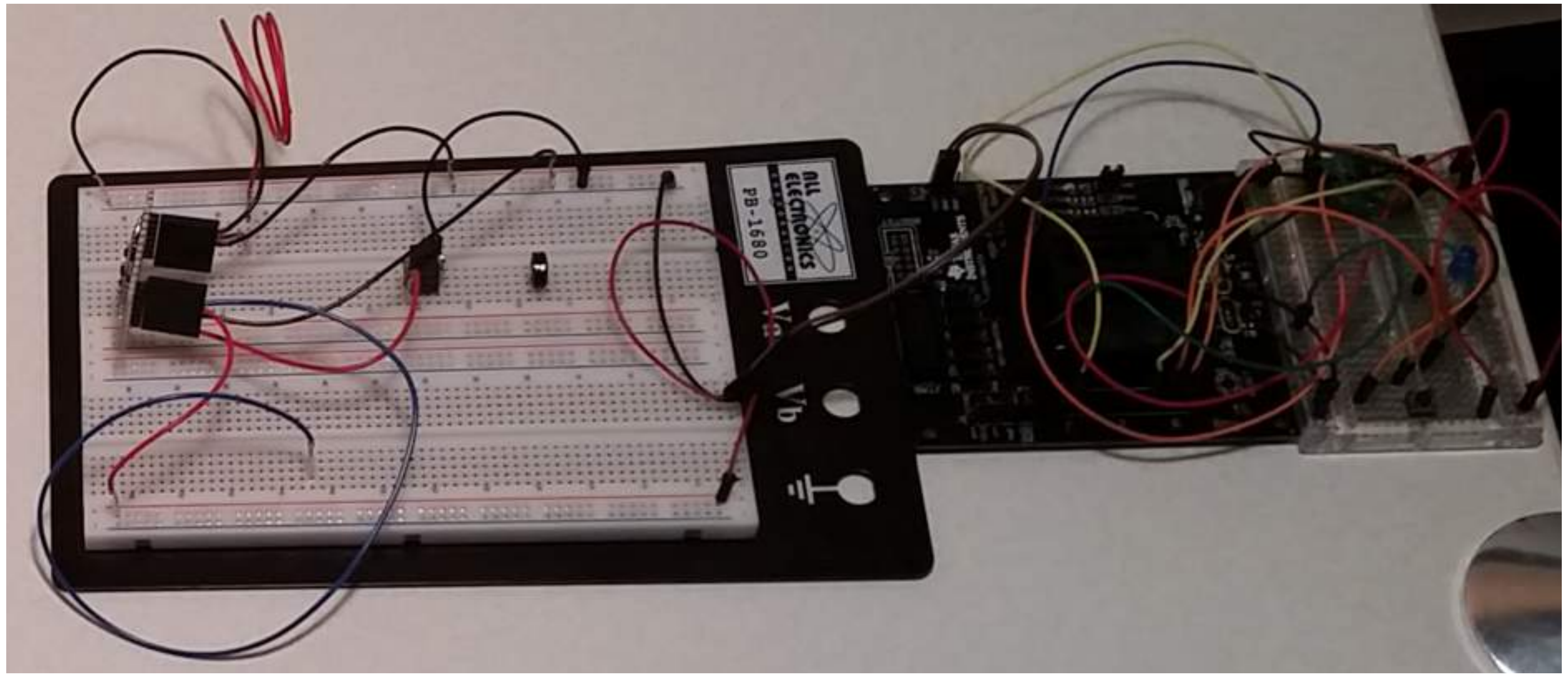
Checkpoint & Recovery

Data Versioning

Prototype Implementation

Energy-harvesting Platform Prototypes

WISP5 from Univ. of Washington



Custom prototype board

Applications

Activity Recognition (accelerometer+ML)

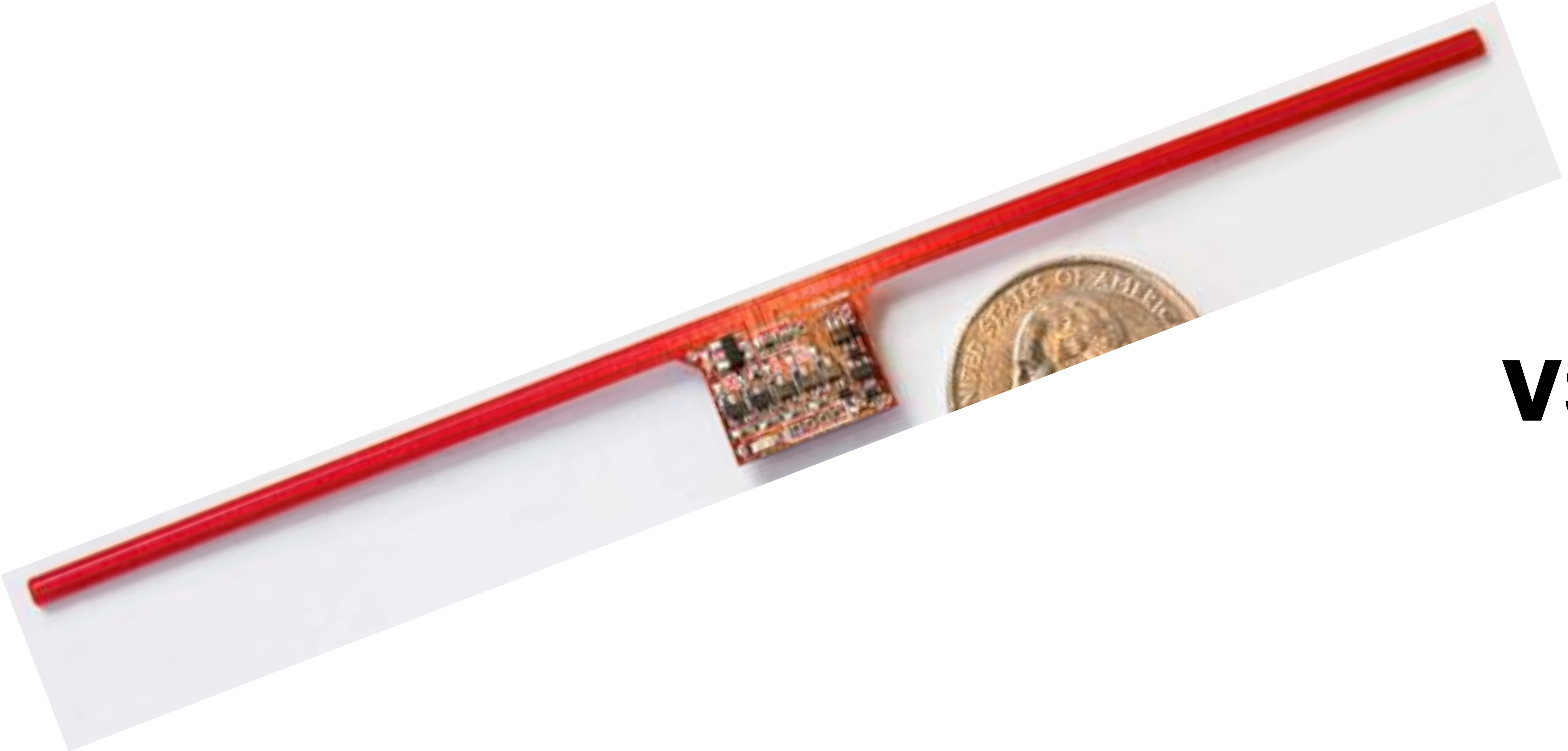
Cold-chain Equipment Monitor

Multi-granular Sensor Log

Key Result:
Applications suffer **errors** & **failures** without our system support for tasks

Activity Recognition

e.g., fall detection, tremor detection



vs.

Activity Recognition

e.g., fall detection, tremor detection

Activity Recognition

e.g., fall detection, tremor detection

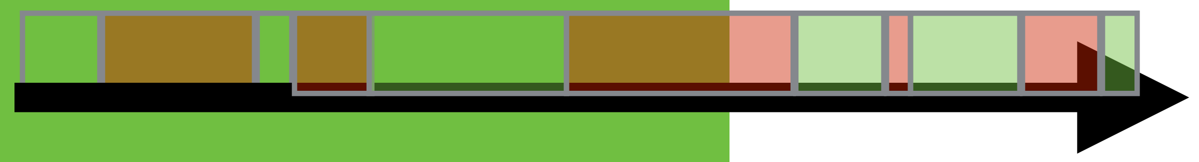
The Intermittent Execution Model

Thinking About Intermittently-powered Devices

Intermittence & Data Consistency

Hidden Problems Caused by Intermittence

Designing for Intermittence



Strategies for Coping with Intermittence

Microsoft Research Faculty Summit 2014 15TH ANNUAL

The Hidden Challenges of Intermittent Execution

Brandon Lucia | Microsoft Research

work done in cooperation with
Ben Ransford | University of Washington

Microsoft