Programming Challenges of Intermittent Energy-harvesting Devices

Brandon Lucia | Assistant Professor, CMU ECE

ABSTRACT Research Group - https://intermittent.systems

with PhD Students: Alexei Colin, Kiwan Maeng, Emily Ruppel, Vignesh Balaji
Graham Gobieski, Milijana Surbatovich

MS Students: Preeti Murthy, Amanda Marano, Neil Ryan, Devon White, Chris Meredith

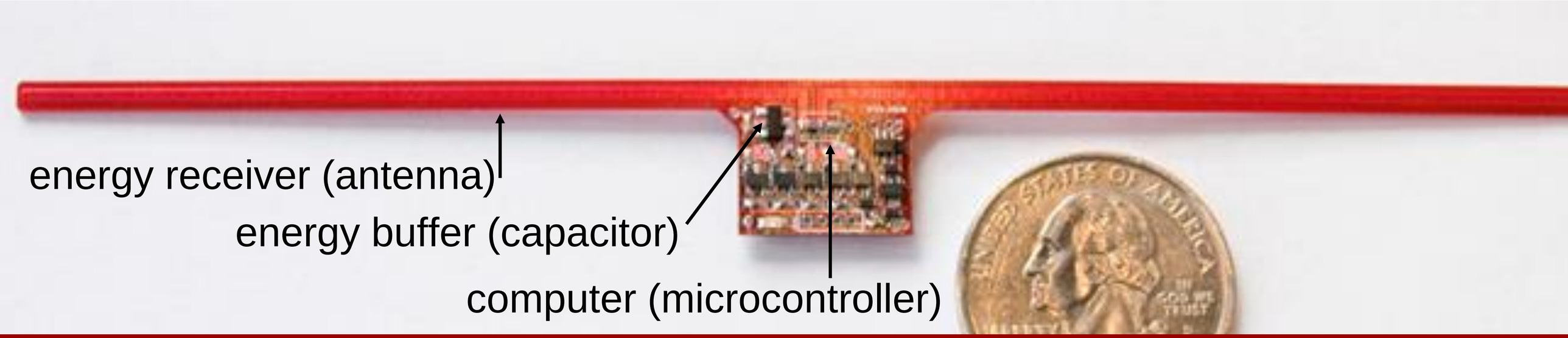
BS Students: Graham Harvey, Mark McElwaine, Hannah Tomio

Industry Collaborators: Zac Manchester (Harvard/KickSat)
Alanson Sample (Disney Research Pittsburgh)

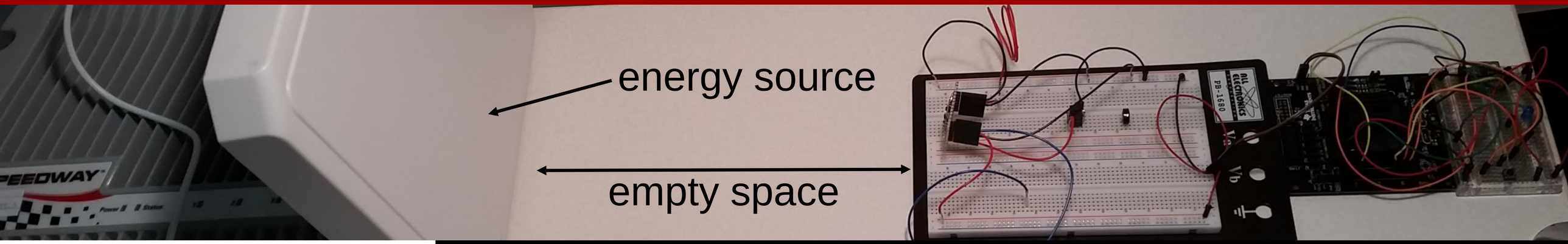


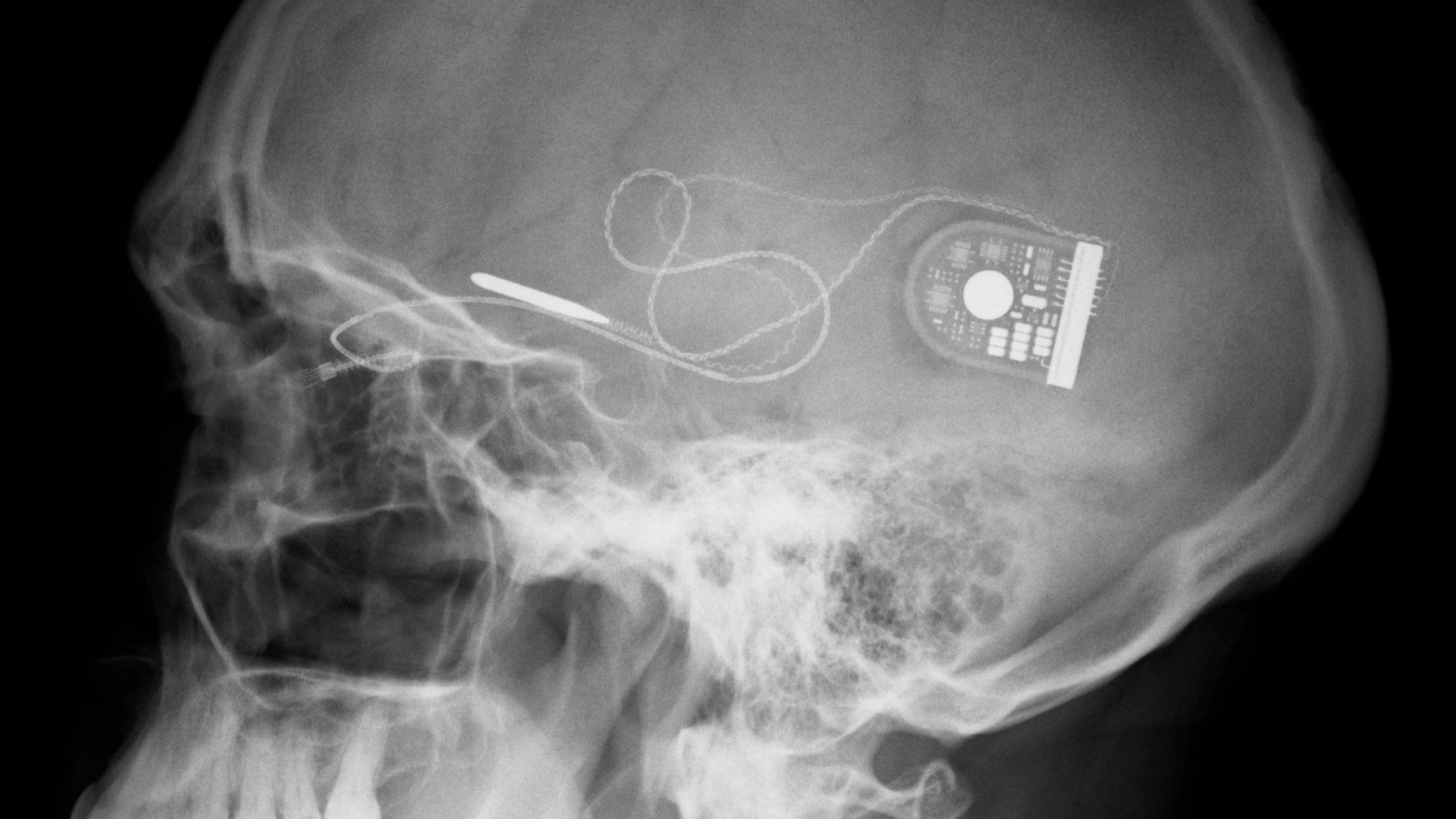
Emerging Hardware/Software Devices Everywhere

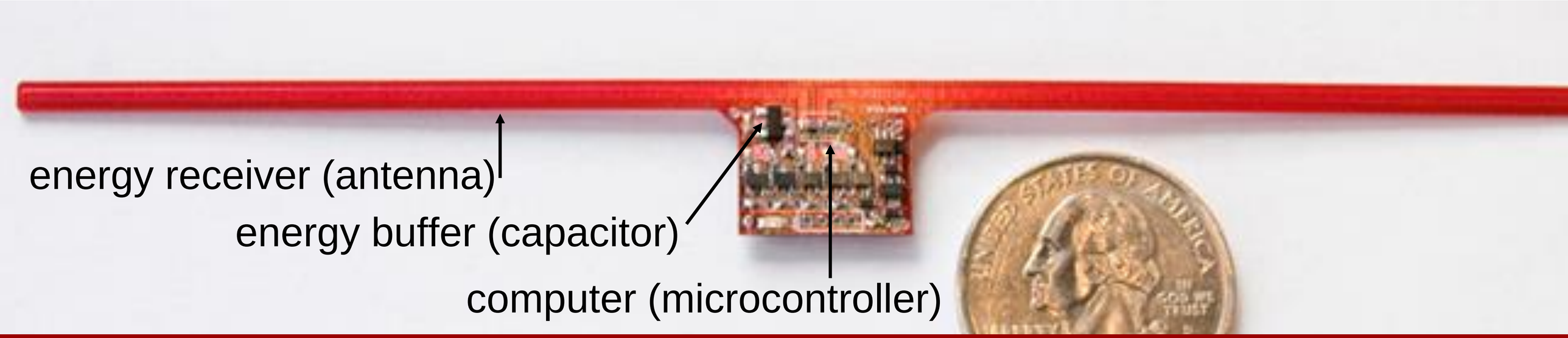
Wearables, sensors, “Internet of Things”, tiny satellites, medical implants, environmental monitoring, security, computational art, interactive clothing, smart furniture, smart carpets, smart toilet paper



Energy harvesting untethers devices



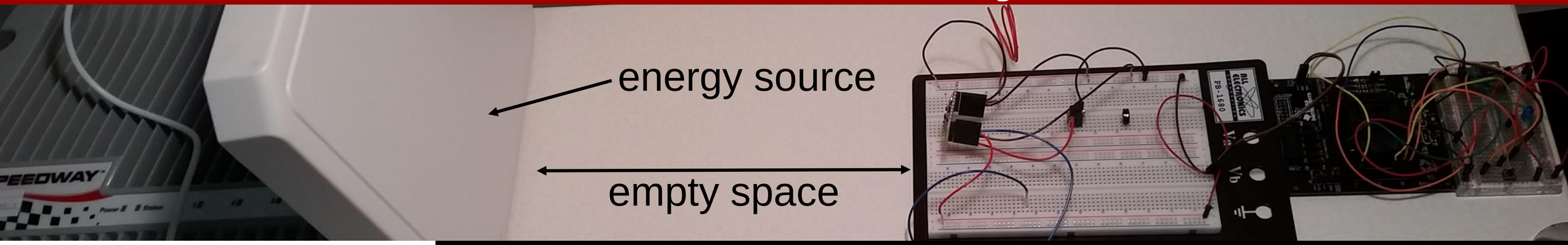




…but energy is **intermittent**

—and as a result—

software becomes **inherently unreliable.**



The Intermittent Execution Model

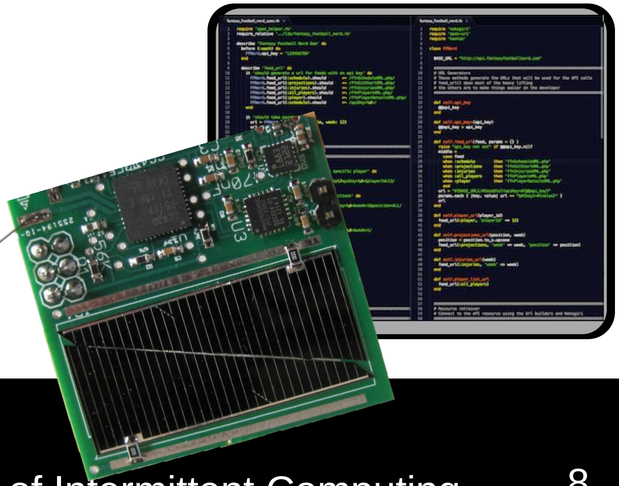
Reasoning About Intermittently-powered Devices

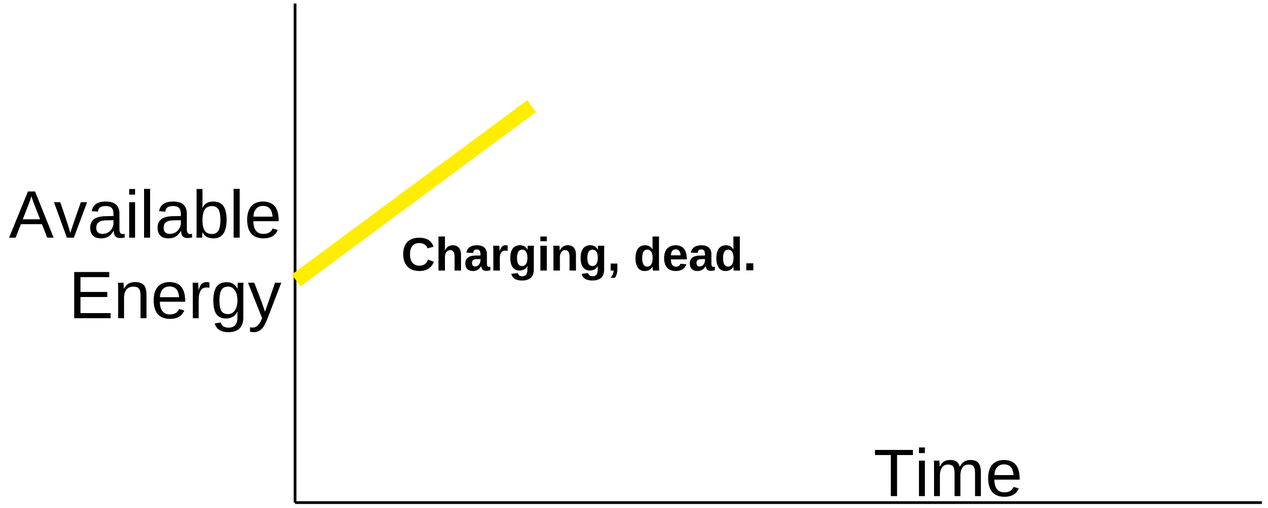
Designing for Intermittence

System Support for Reliable Intermittence

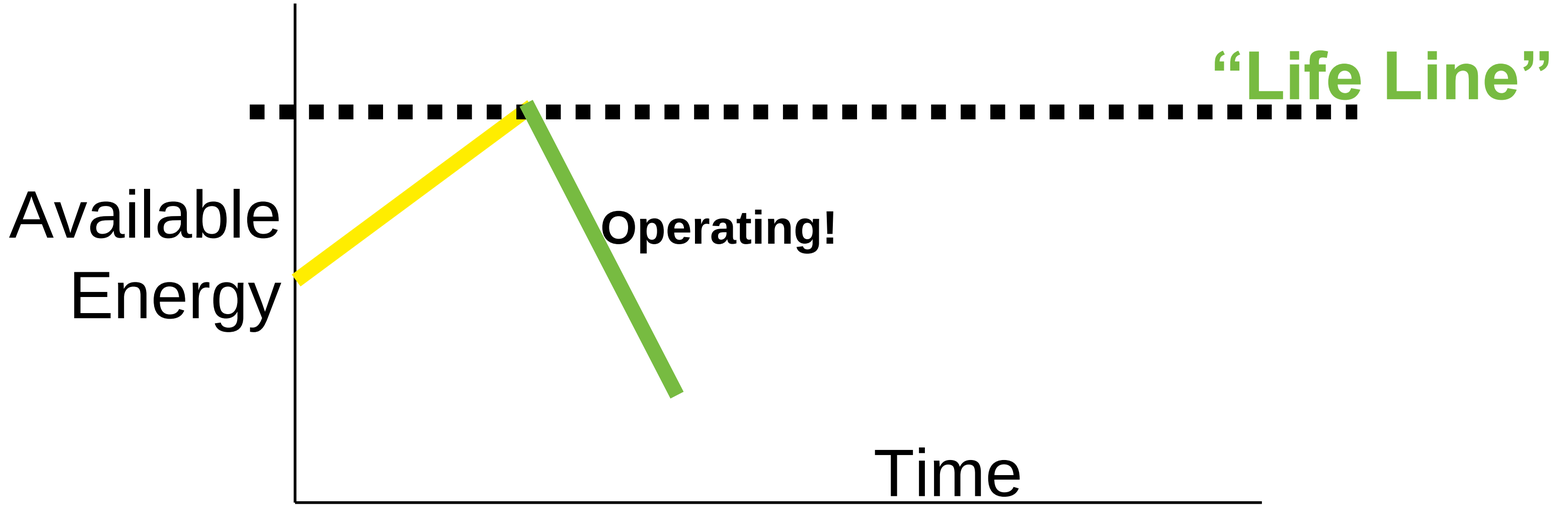
Debugging Intermittence

Toolchain Support for Understanding Intermittence

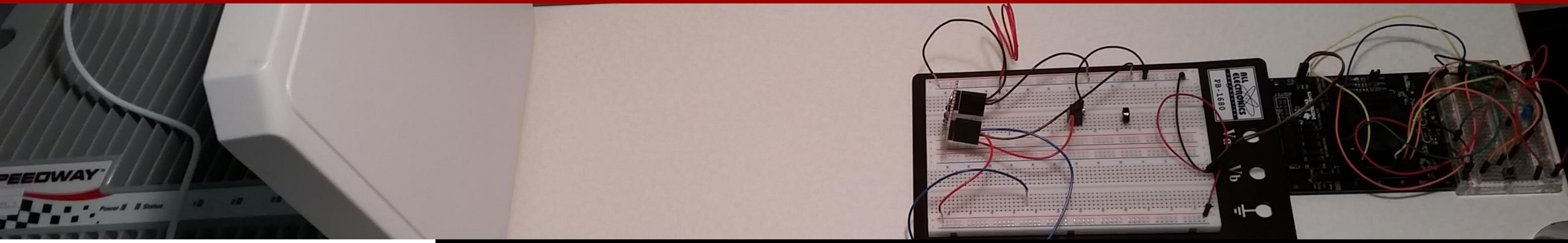


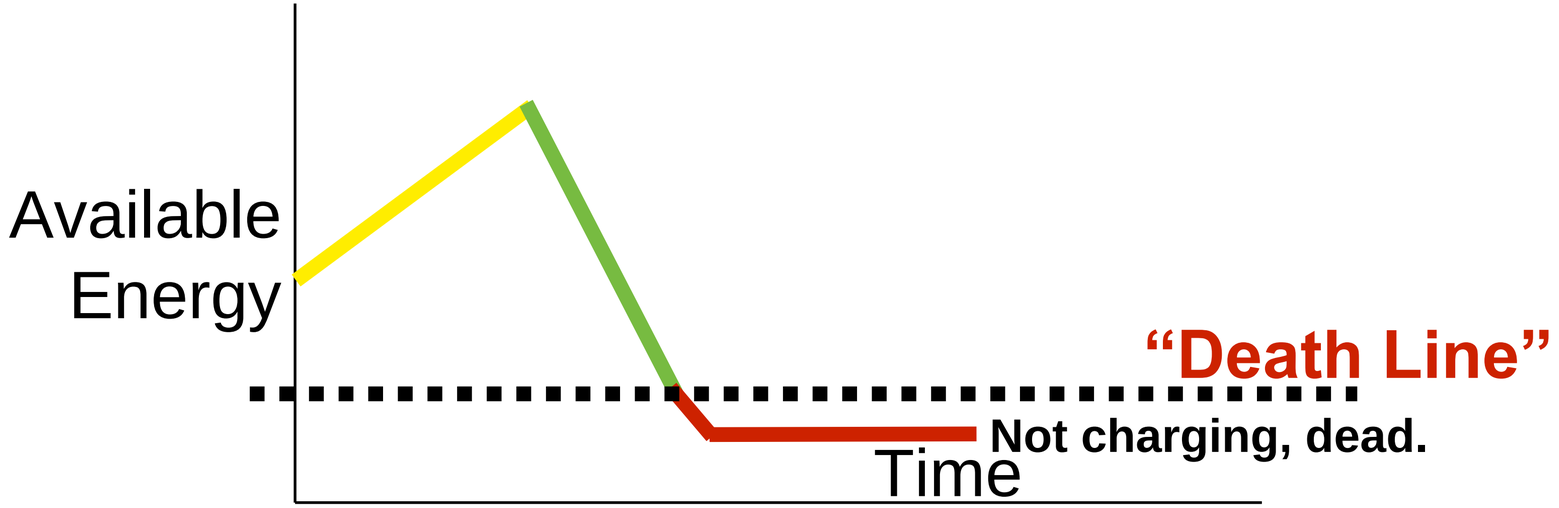


Running on intermittent energy

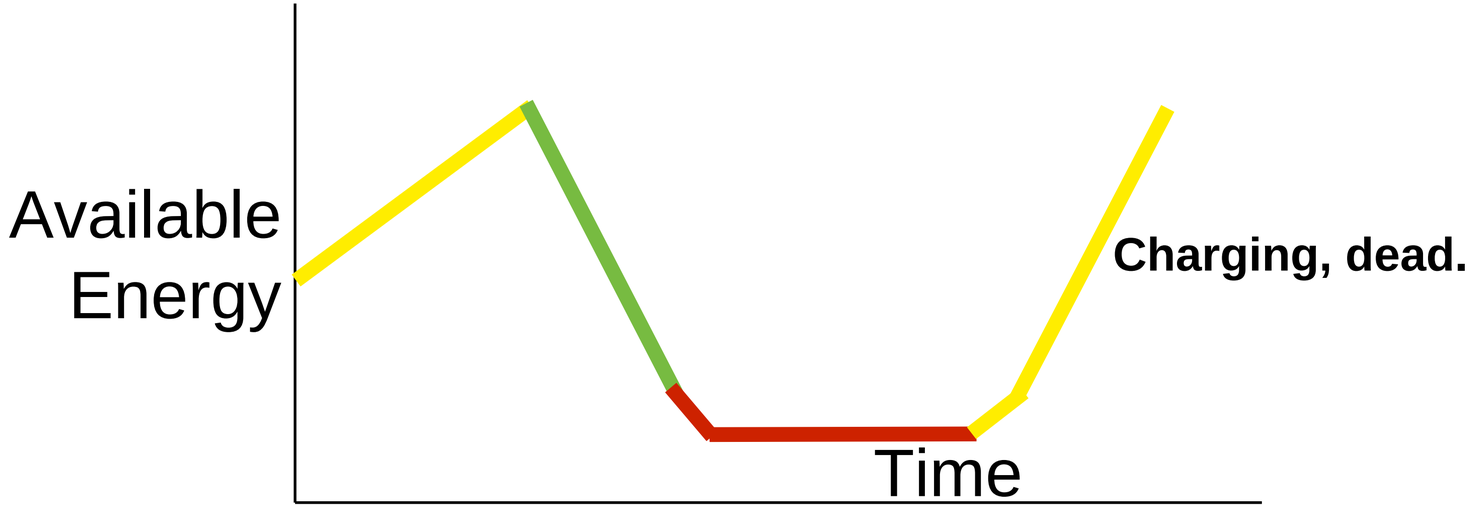


Running on intermittent energy

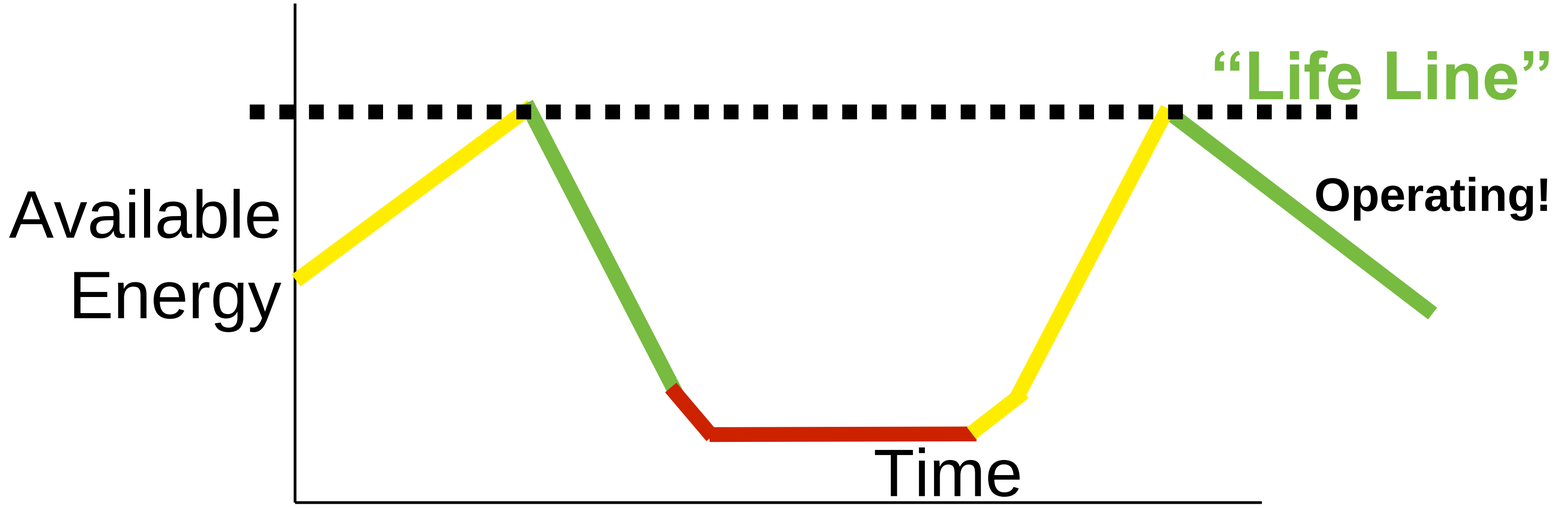




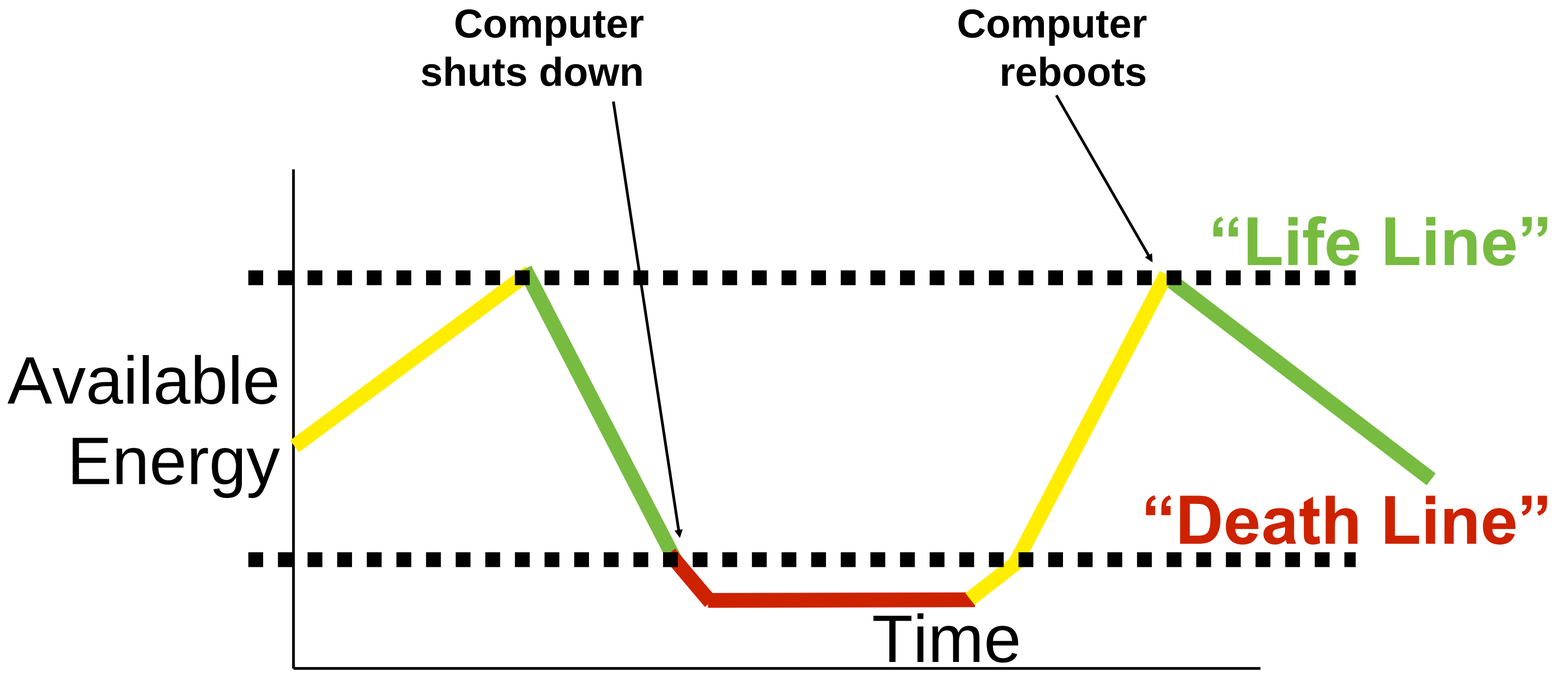
Running on intermittent energy

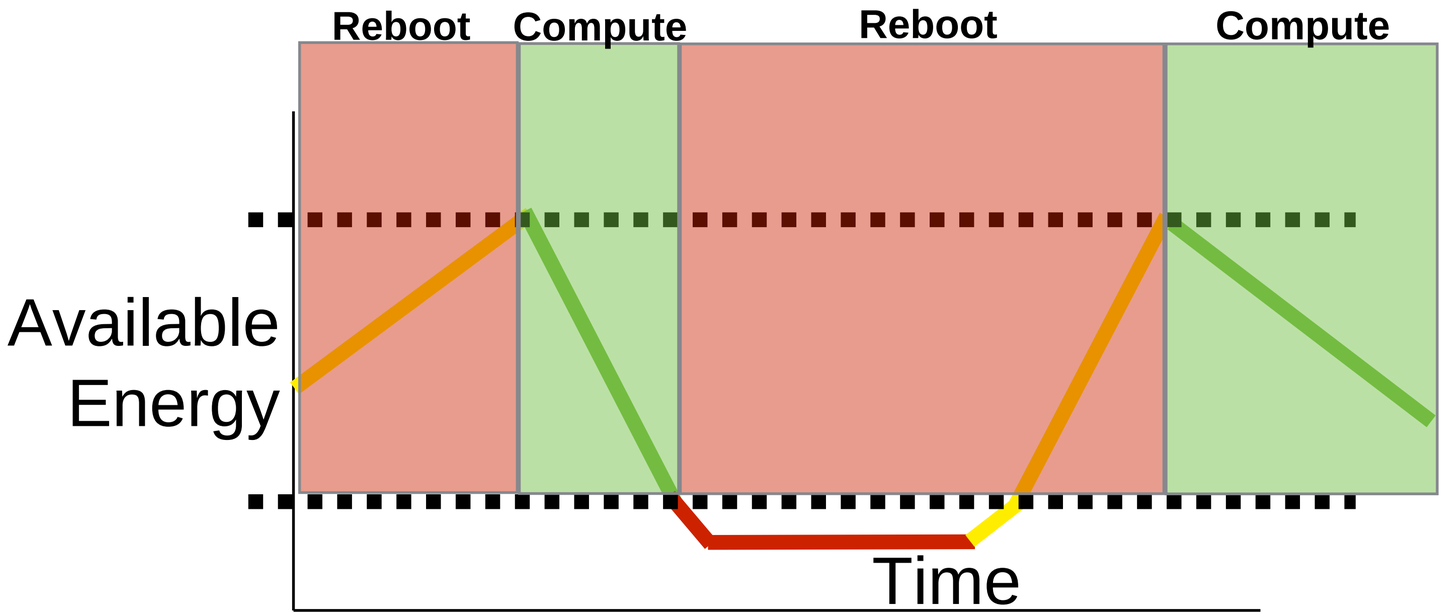


Running on intermittent energy



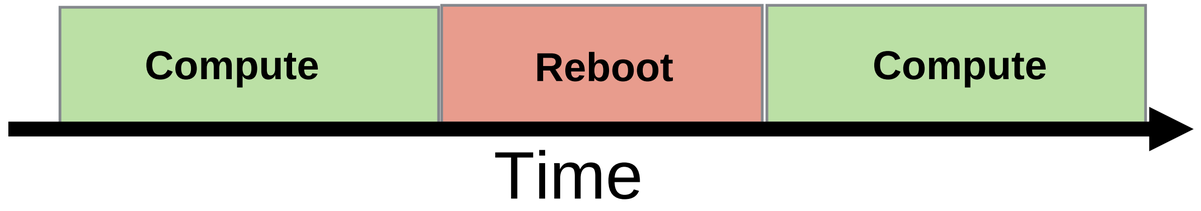
Running on intermittent energy


Computer
shuts down
Computer
reboots
“Life Line”
Available
Energy
“Death Line”
Time


Reboot
Compute
Reboot
Compute
Available
Energy
Time

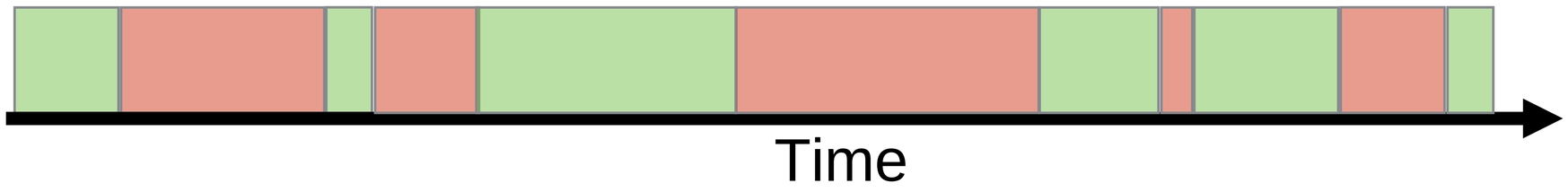
The Intermittent Execution Model

[PLDI ’15]



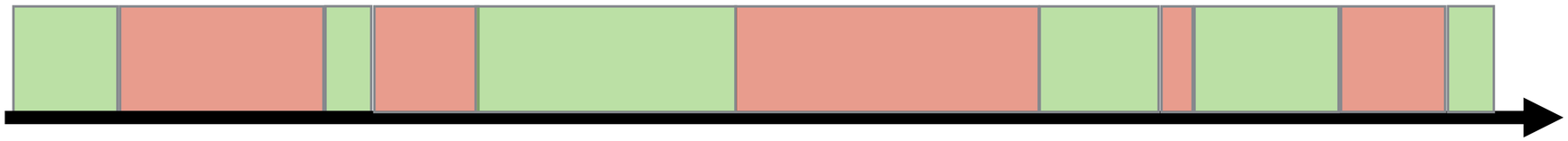
The Intermittent Execution Model

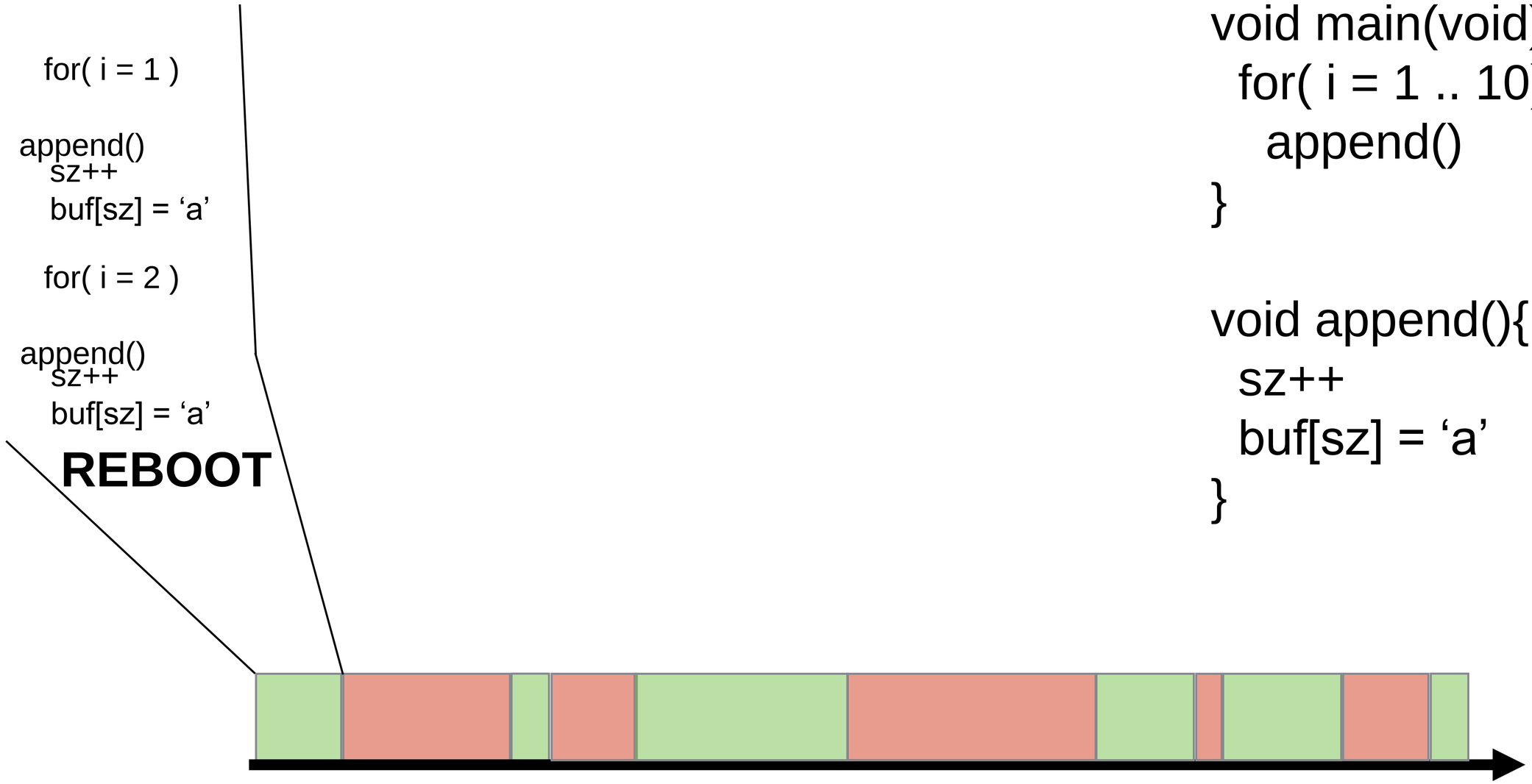
[PLDI ’15]



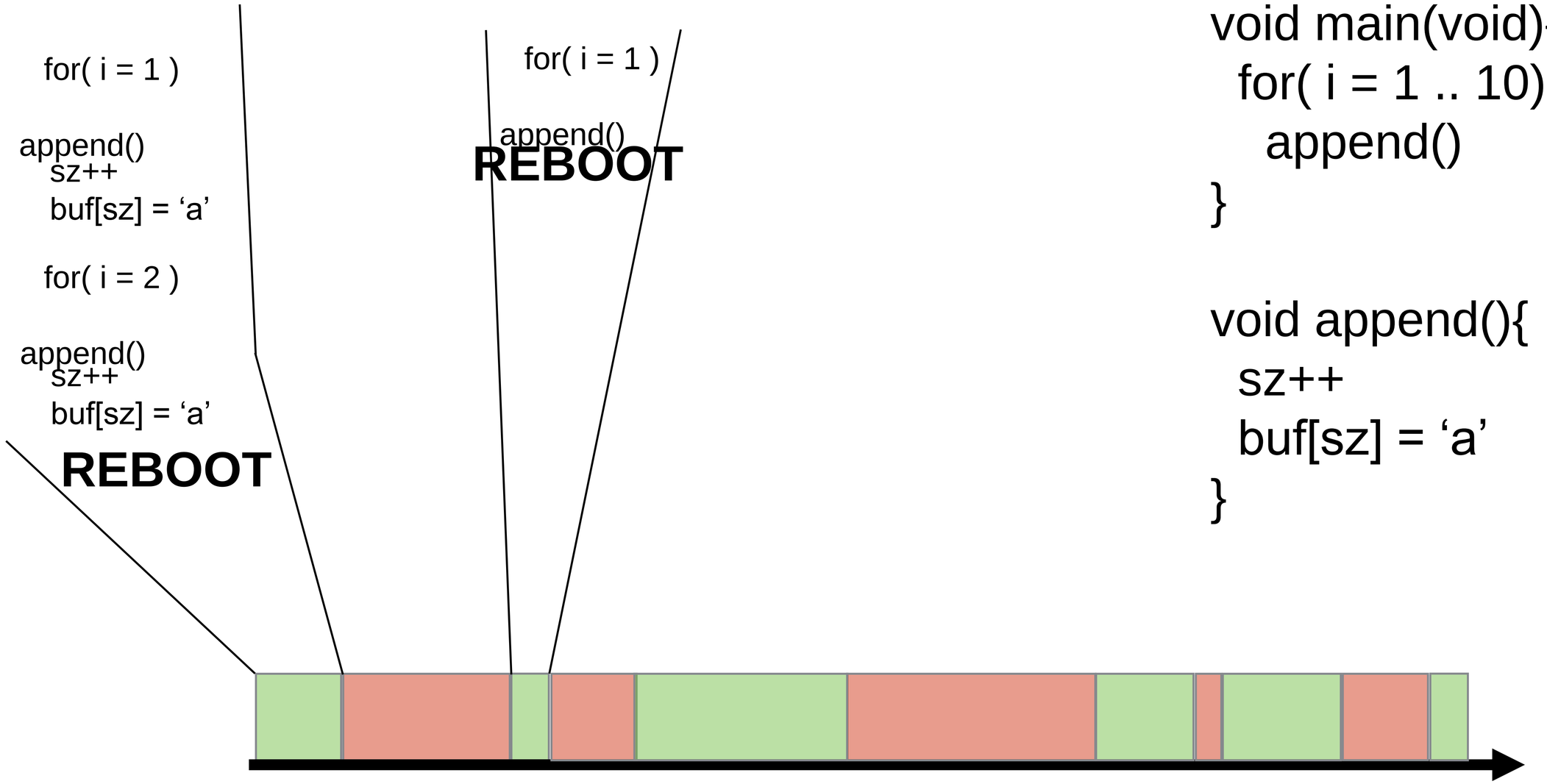
Goal: Run programs that take longer than one green box

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

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

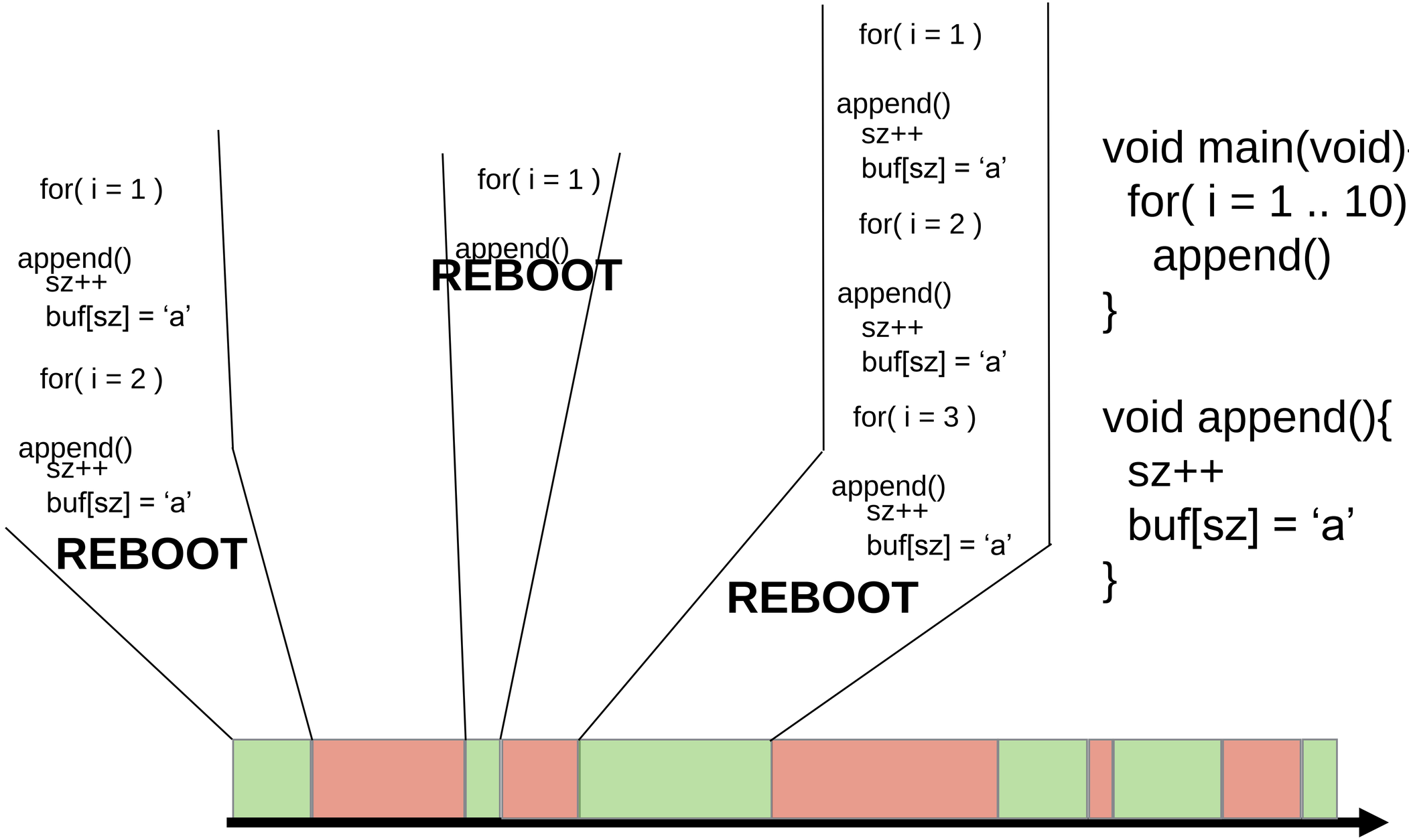





```
void main(void){
  for( i = 1 .. 10)
    append()
}

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

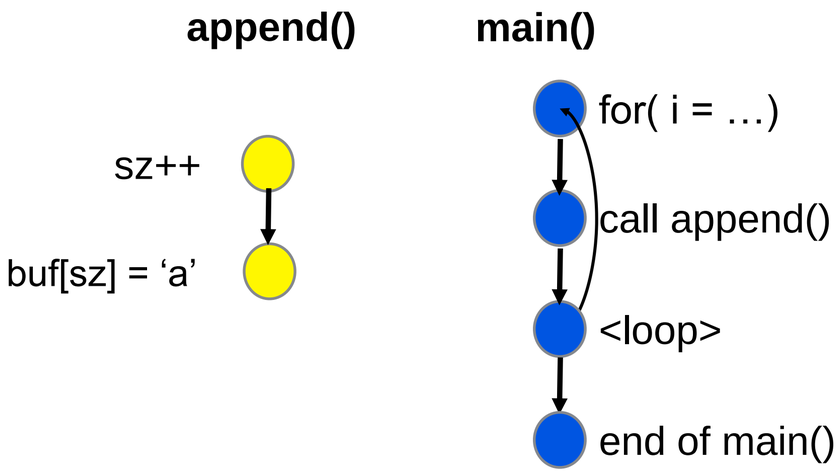



```
void main(void){
  for( i = 1 .. 10)
    append()
}

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



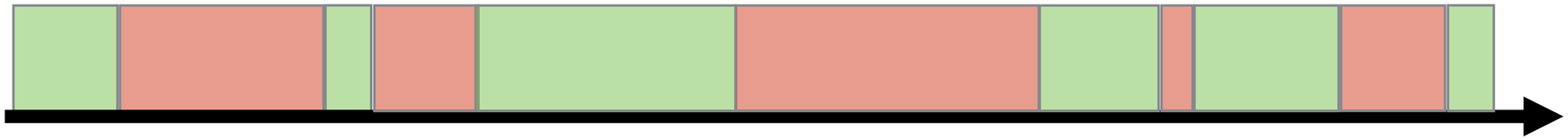

```
void main(void){
  for( i = 1 .. 10)
    append()
}

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

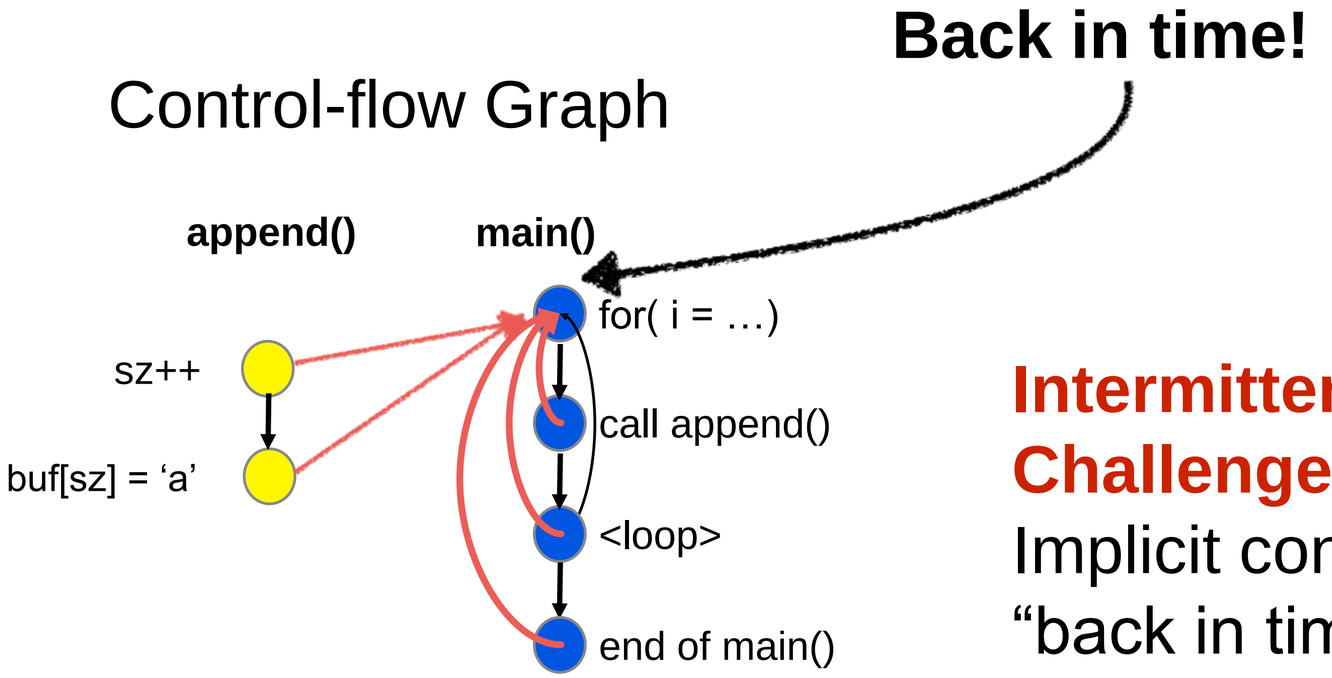
Control-flow Graph




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

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

We can model intermittence as a control-flow problem

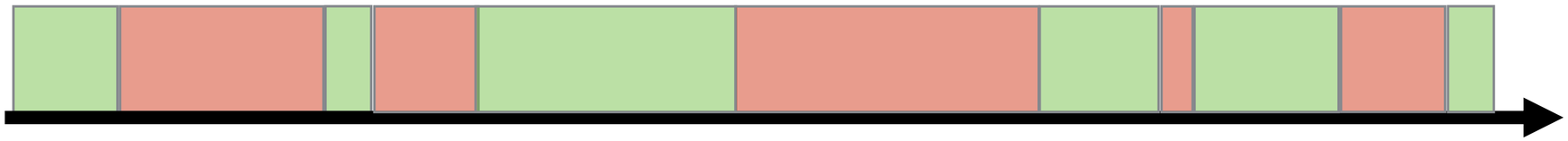




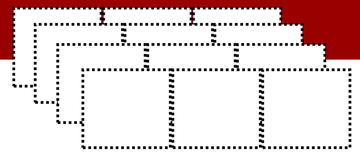
Intermittent Programming Challenge:

Implicit control-flow “back in time” on reboot.

Failure Induces **Implicit** Control-flow

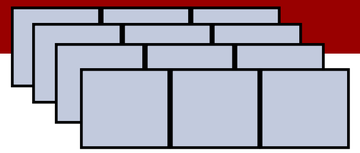


Mixture of Volatile & Non-volatile State



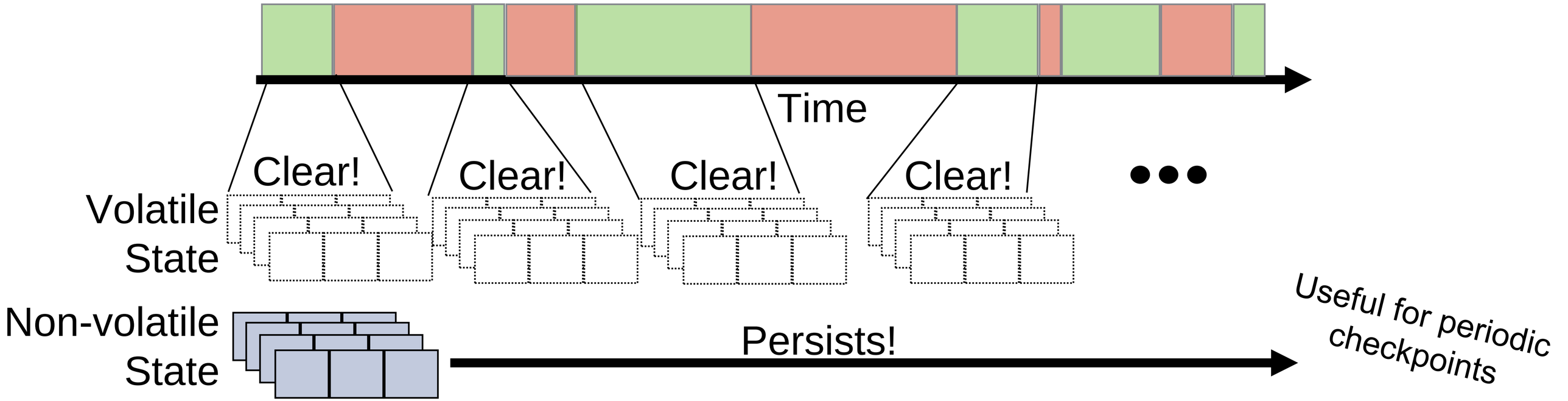
volatile memory (e.g., DRAM, SRAM registers)

Access latencies growing similar w/ new technology

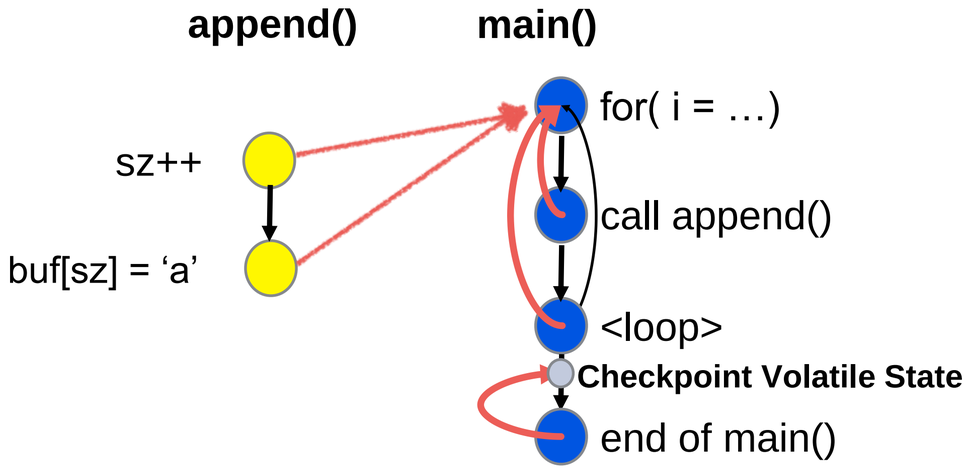


non-volatile memory (e.g., Flash, FRAM)

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



Control-flow Graph



Use NV memory to capture values of registers, stack, and global variables

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

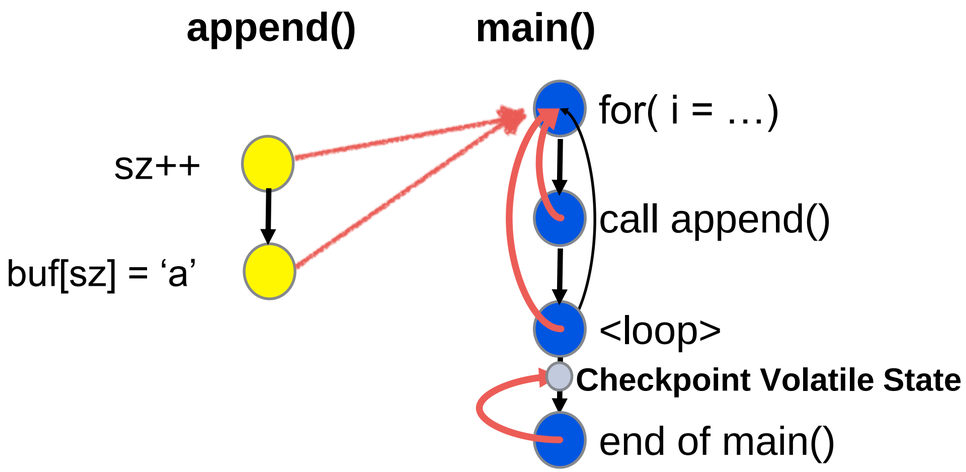
Benjamin Ransford
Department of Computer Science
University of Massachusetts Amherst
ransford@cs.umass.edu

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

Prior work (e.g., Mementos) **overlooked** the need to **version** non-volatile memory with checkpoints

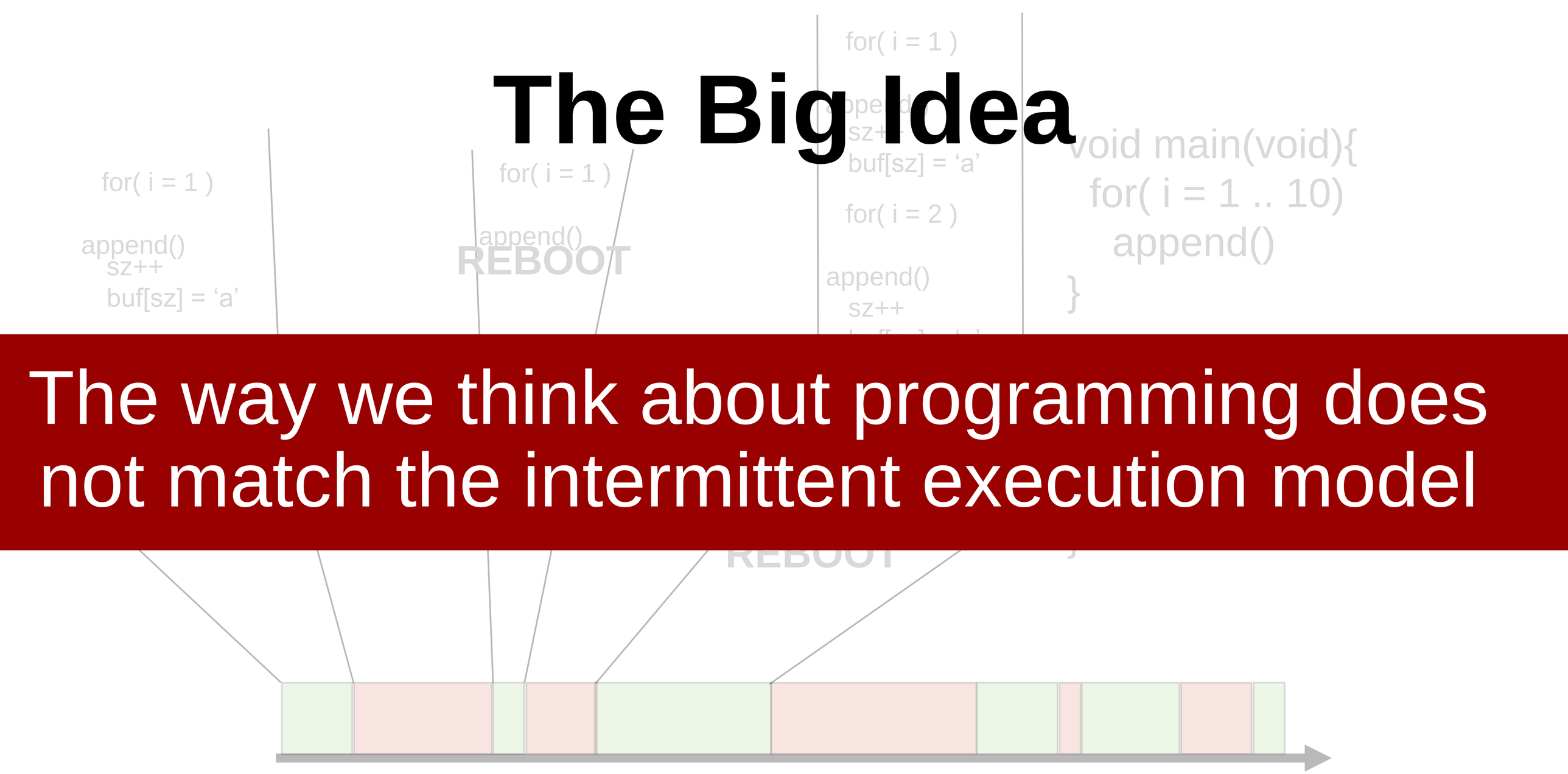
Control-flow Graph



Intermittent Programming Challenge:
Implicit control-flow
“back in time” on reboot.

Checkpoint introduce **more complex** implicit control-flow

The Big Idea

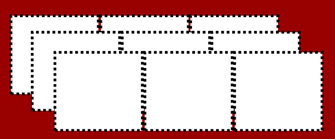


The way we think about programming does not match the intermittent execution model

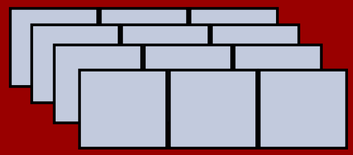
Challenge: *Intermittence Bugs*

Software that is **correct** with continuous power can be **incorrect** due to intermittent execution

especially when we combine non-volatile and volatile memory!

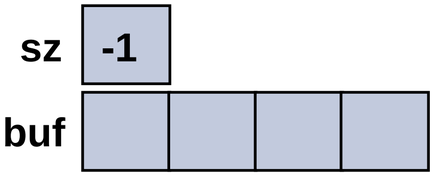


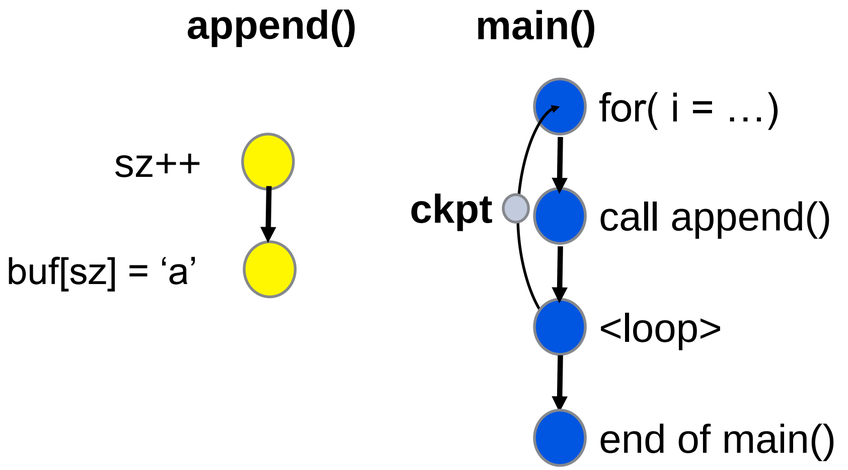
+



Intermittence Bug: Out-of-thin-air Values

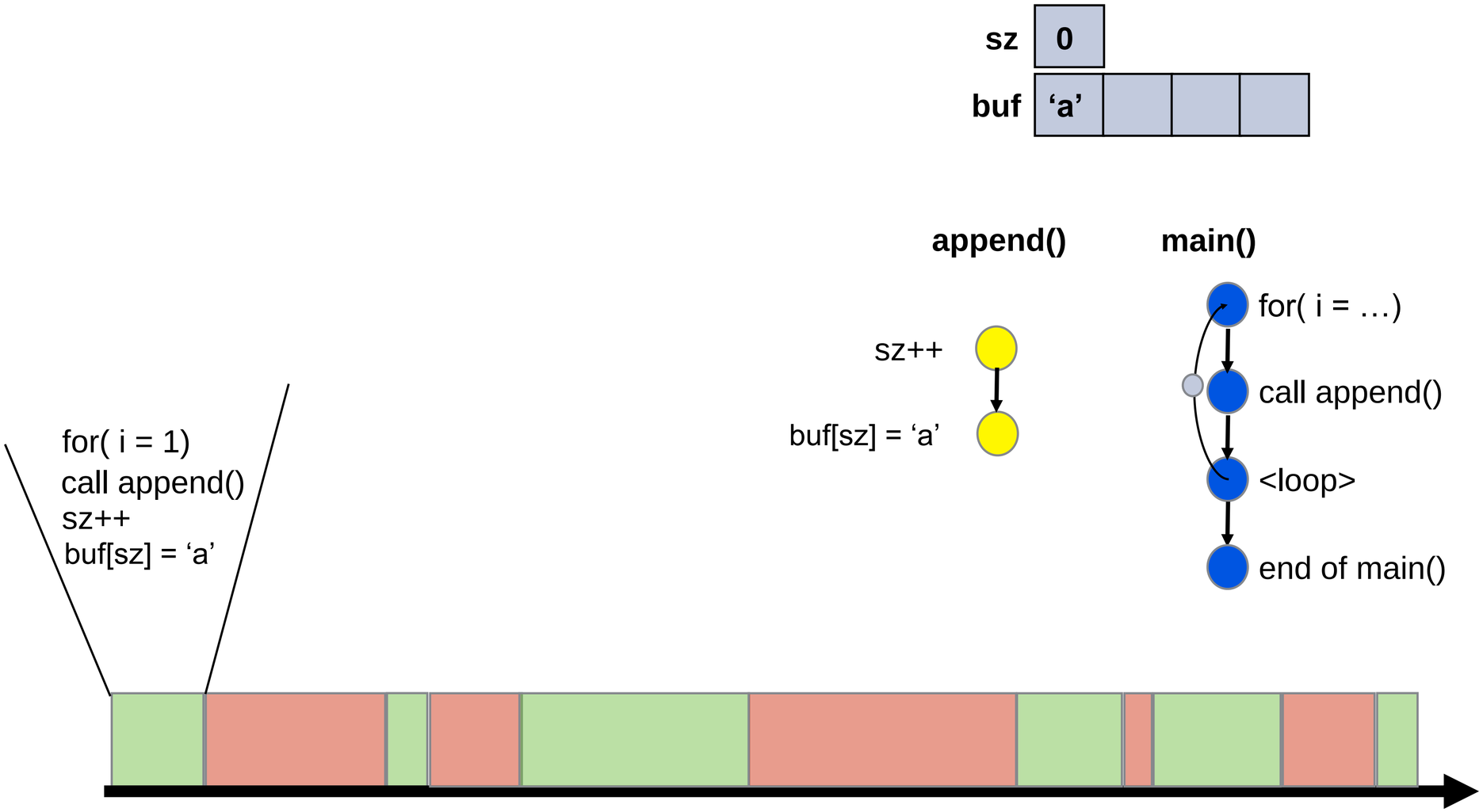
[MSPC ’14; PLDI ‘15]





Intermittence Bug: Out-of-thin-air Values

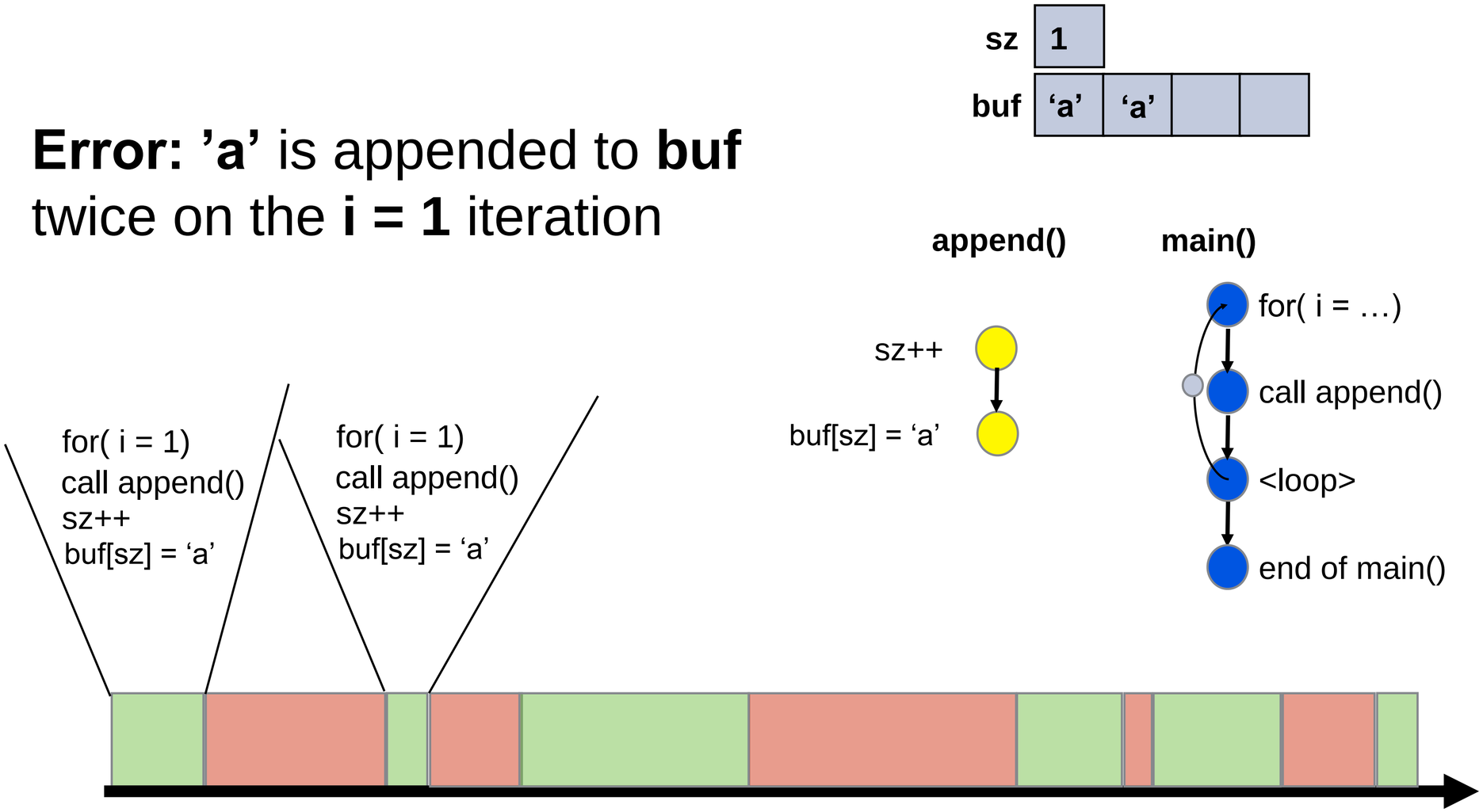
[MSPC ’14; PLDI ‘15]



Intermittence Bug: Out-of-thin-air Values

[MSPC ’14; PLDI ‘15]

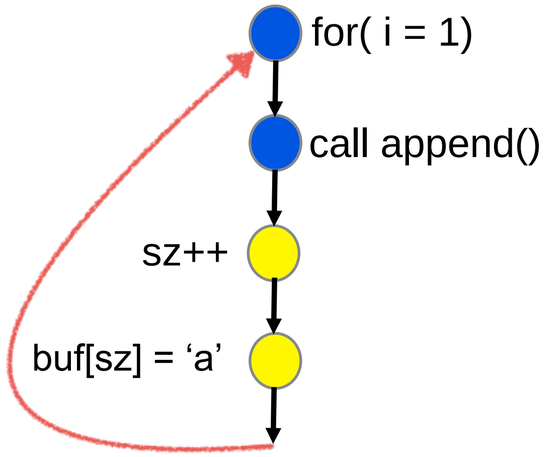
Error: ’a’ is appended to **buf** twice on the **i = 1** iteration

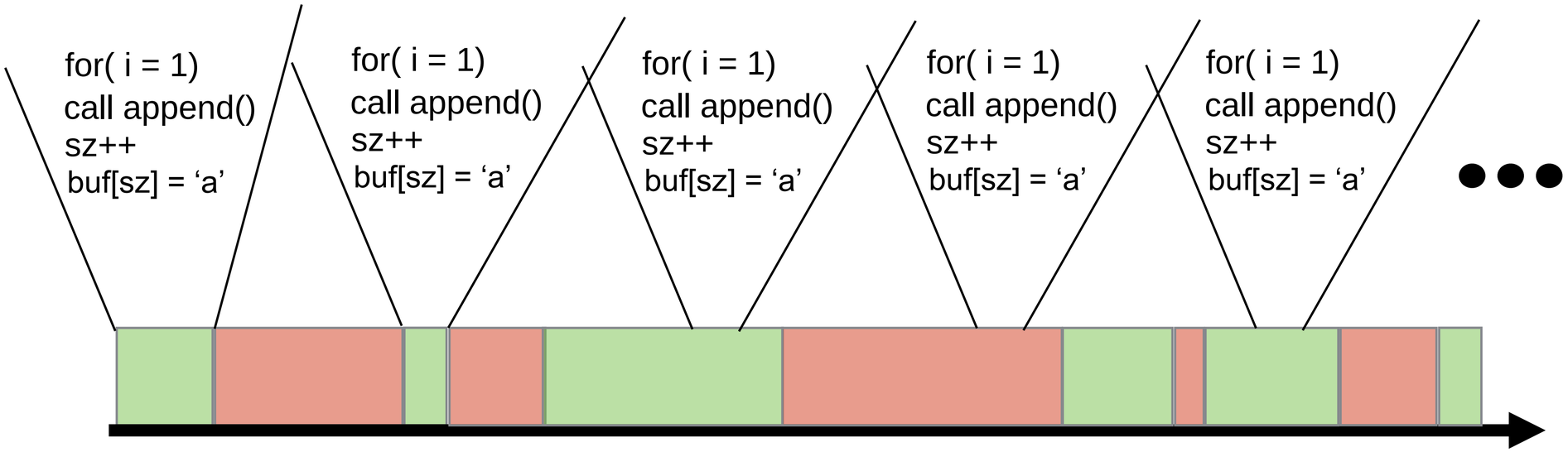


Intermittence Bug: Out-of-thin-air Values

[MSPC ’14; PLDI ‘15]

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

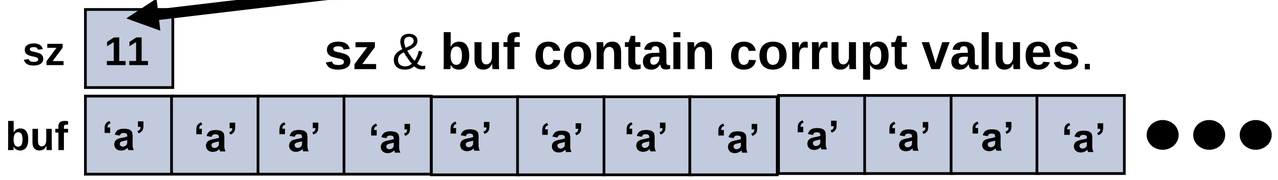




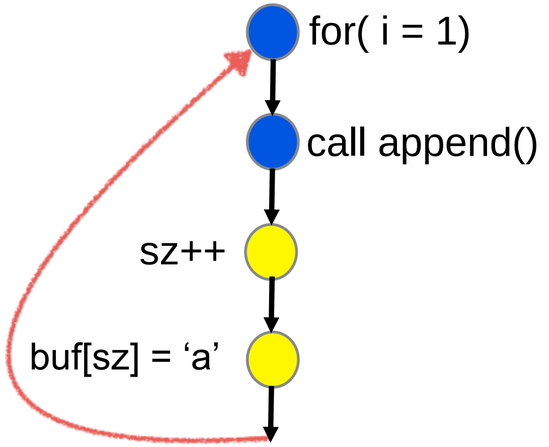
Intermittence Bug: Out-of-thin-air Values

[MSPC ’14; PLDI ‘15]

“Out of thin air” value not permitted by any continuous execution!



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



Application failures can depend solely on the *availability of wireless power* ***and*** *capacitor behavior*!

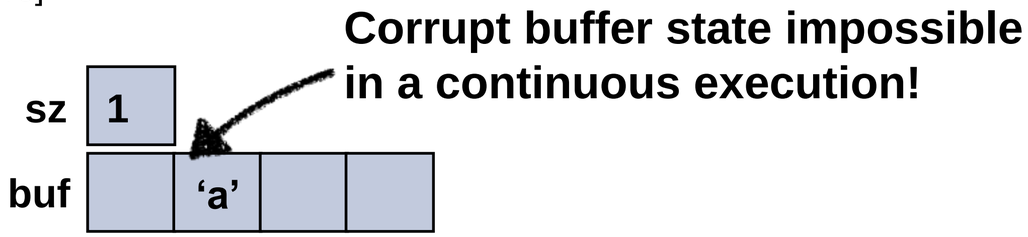
Intermittence Bug: Structural Inconsistency

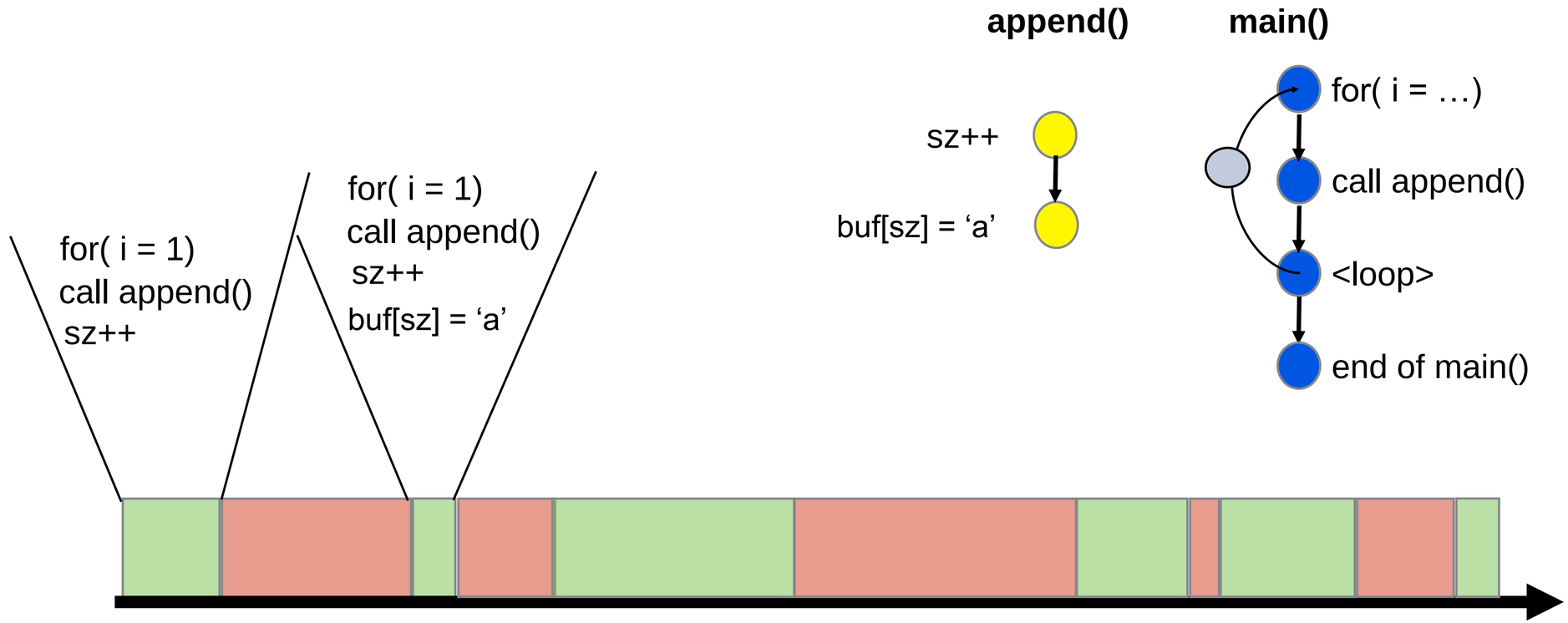
[MSPC ’14; PLDI ’15; Colin OOPSLA ‘16]

Reboot **violates atomicity** of

```
sz++; buf[sz] = 'a'
```

causing update to **wrong entry** in `buf`.



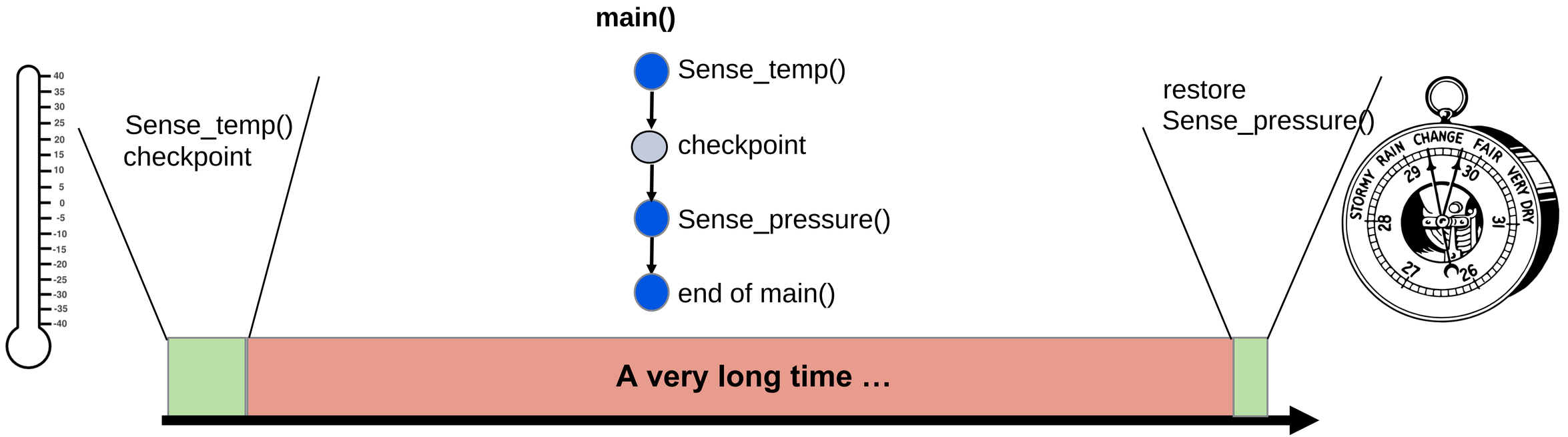


Intermittence Bug: I/O Atomicity & Timeliness

[Colin OOPSLA '16, Maeng OOPSLA '17, Hester et al SenSys '17]

Checkpoint between I/O operations that should execute together leaves I/O inconsistent

No atomicity control w/ automatic checkpoints



Inconsistent data read at different moments in time

Useless, stale data read a long time ago

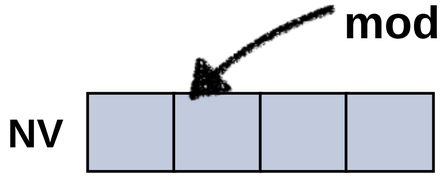
Intermittence Bug: Livelock, Non-termination, and Energy-Provisioning

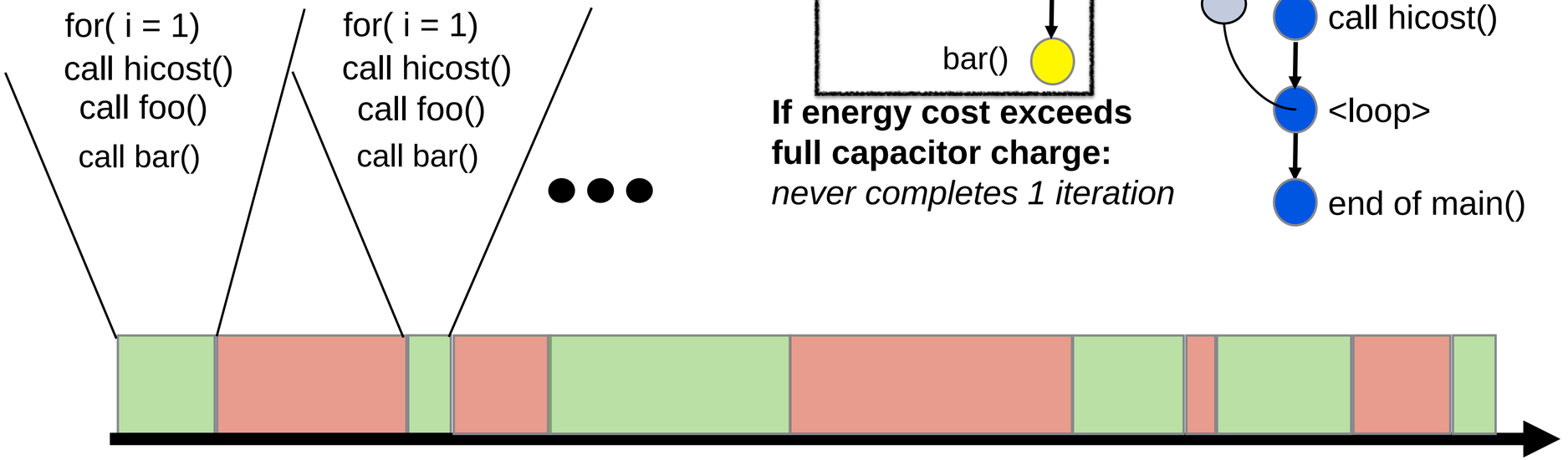
[Colin, et al. CASES '15, ASPLOS '16]

Energy cost atomic span exceeds energy capacity

Code that is **correct** with continuous power **reboots indefinitely** on intermittent power.

Affected code need not even modify NV storage!



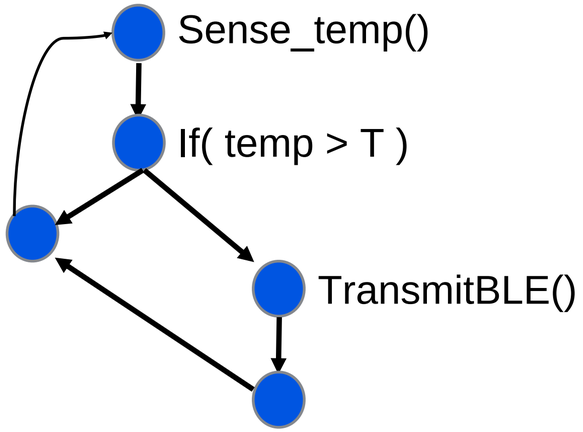


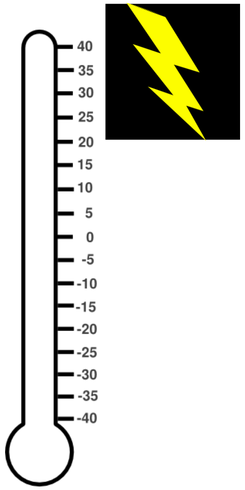
If energy cost exceeds full capacitor charge:
never completes 1 iteration

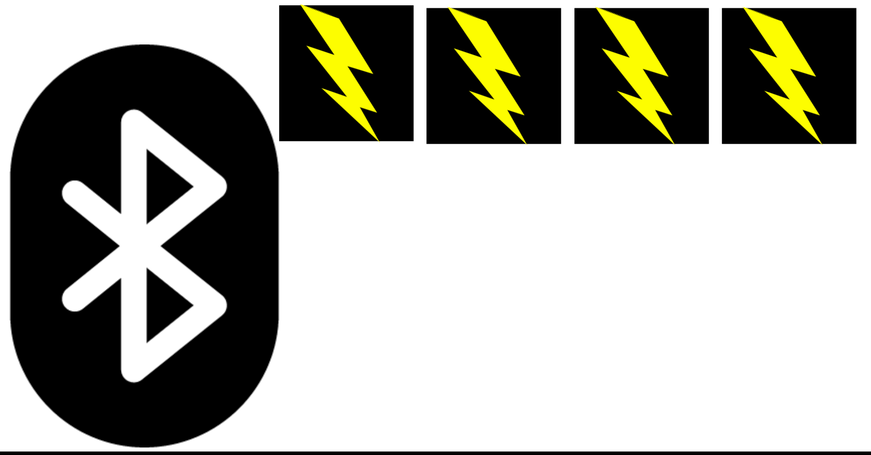
hicost() **main()**

Intermittence "Bug": Reactive vs. Burst Operation

Different operations have different energy requirements because of different hardware activation



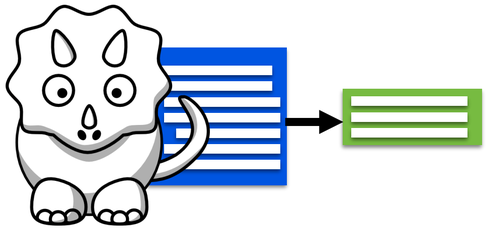




Need *sampling coverage* for sensing, long *burst* for BLE, *reactive* operation for asynchronous events.

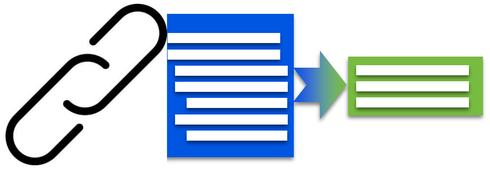
Reliable Intermittent Executi

Intermittent Programming Models



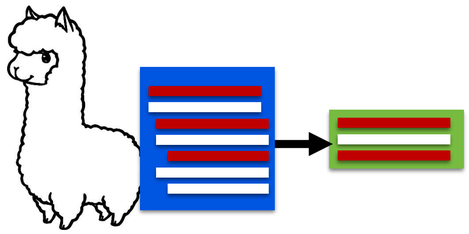
Dino

Arbitrary Task-based Intermittent Programming [PLDI'16]



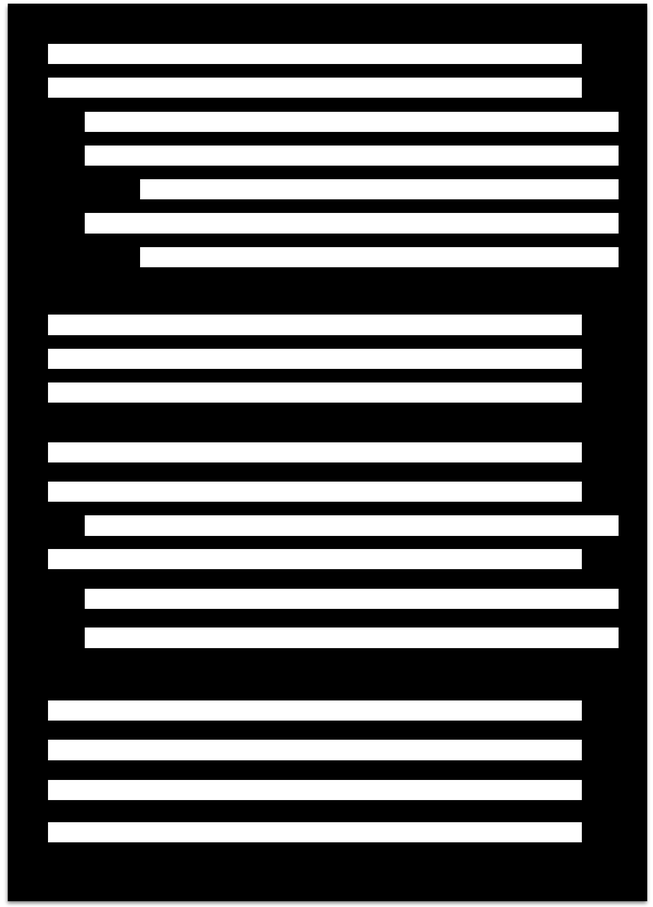
Chain

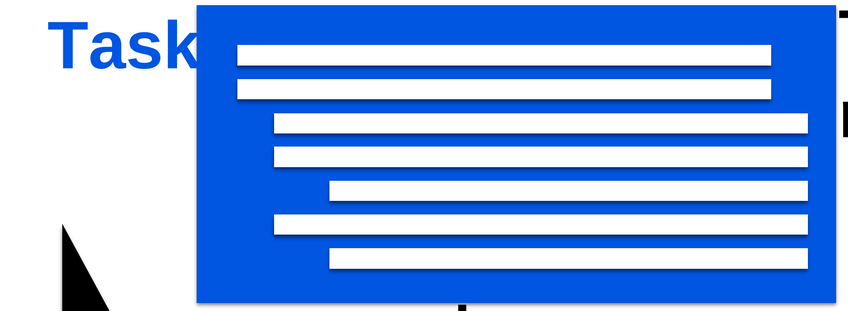
Task-based Intermittence w/ Static Data Duplication [OOPSLA'16]



Alpaca

Task-based Intermittence w/ Dynamic Privatization [OOPSLA'17]





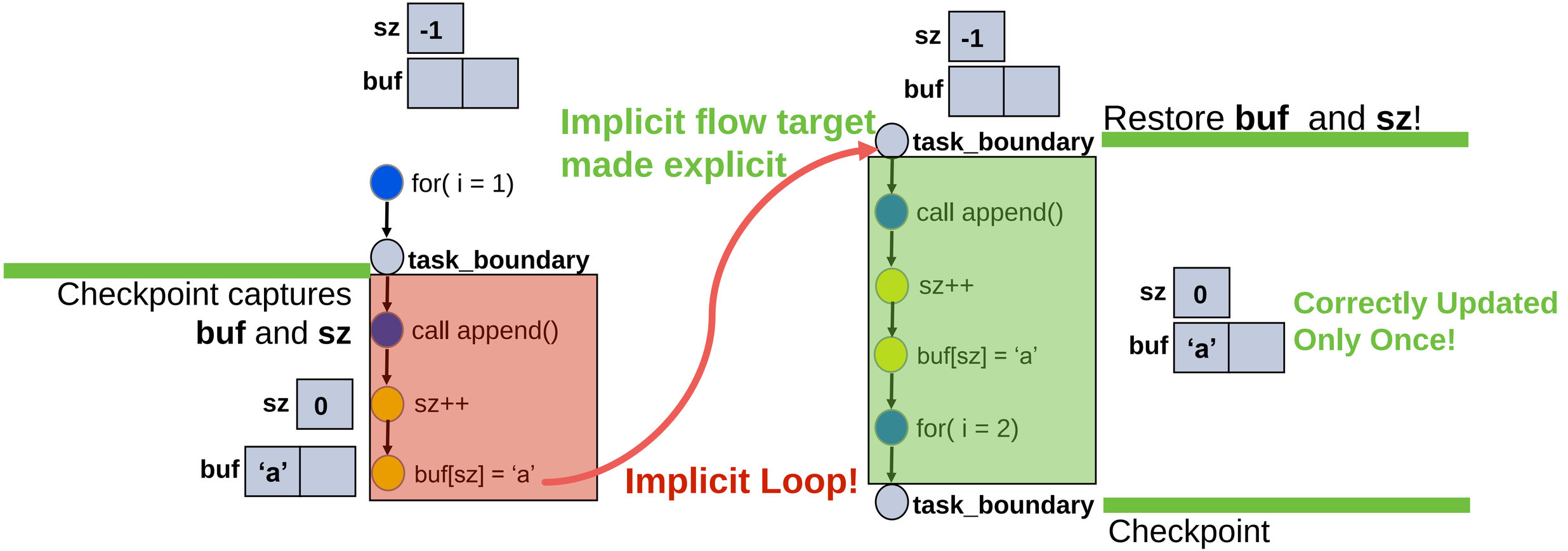
Tasks have *atomic* all-or-nothing semantics

No *implicit* control-flow!

Task boundaries **selectively checkpoint** *volatile* **and** *non-volatile* **state**

Task-atomic Execution in **DINO**

[PLDI ’15]



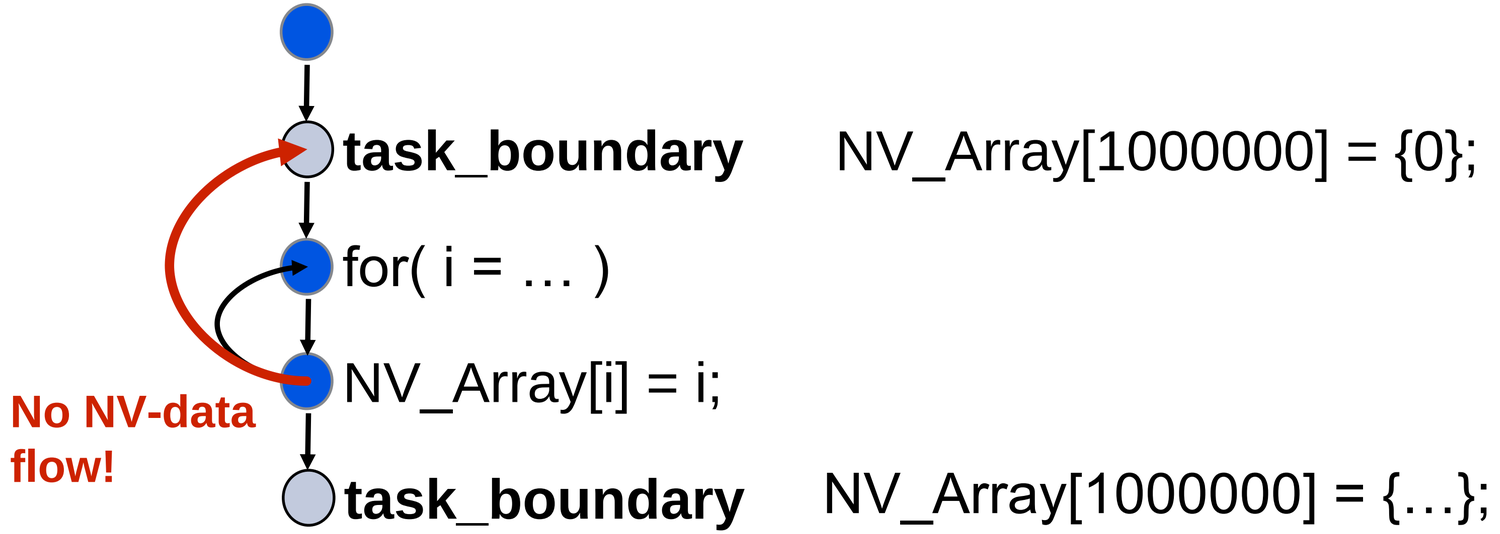
DINO Eliminates OoTA Values

[PLDI ’15]

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

  NV_Array[i] = i;

}
task_boundary
```

Key Question in DINO:
What non-volatile data must we checkpoint?

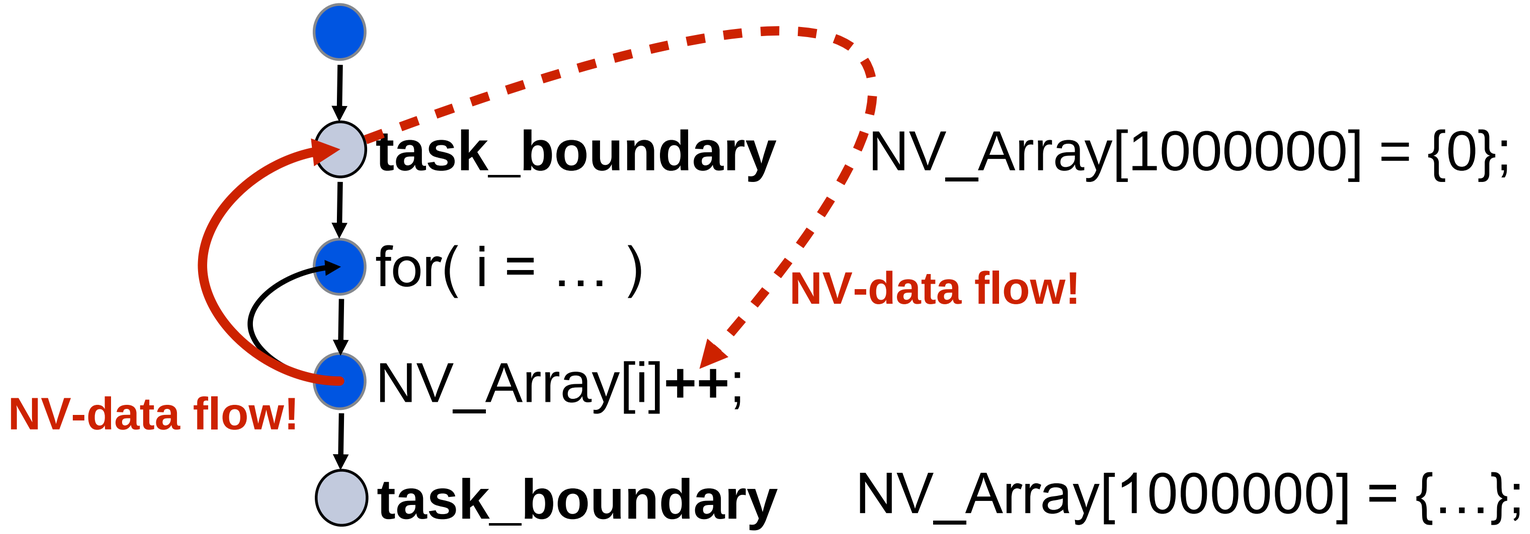
```
int NV_Array[1000000];
task_boundary
for( i = 0; i < 1000000; i++){

  NV_Array[i] = i;

}
task_boundary
```



Insight: No versioning! “future” write not observed in “past”

Selective Versioning Makes Tasks Idempotent

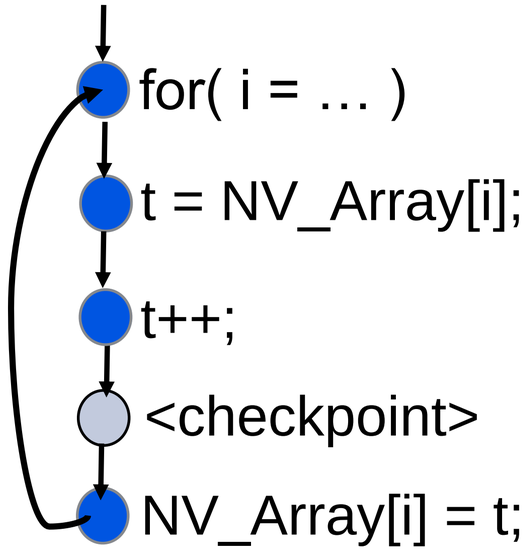
```
int NV_Array[1000000];
task_boundary
for( i = 0; i < 1000000; i++){

  NV_Array[i]++;

}
task_boundary
```



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

Selective Versioning Makes Tasks Idempotent

Ratchet inserts checkpoints
To break WAR dependences



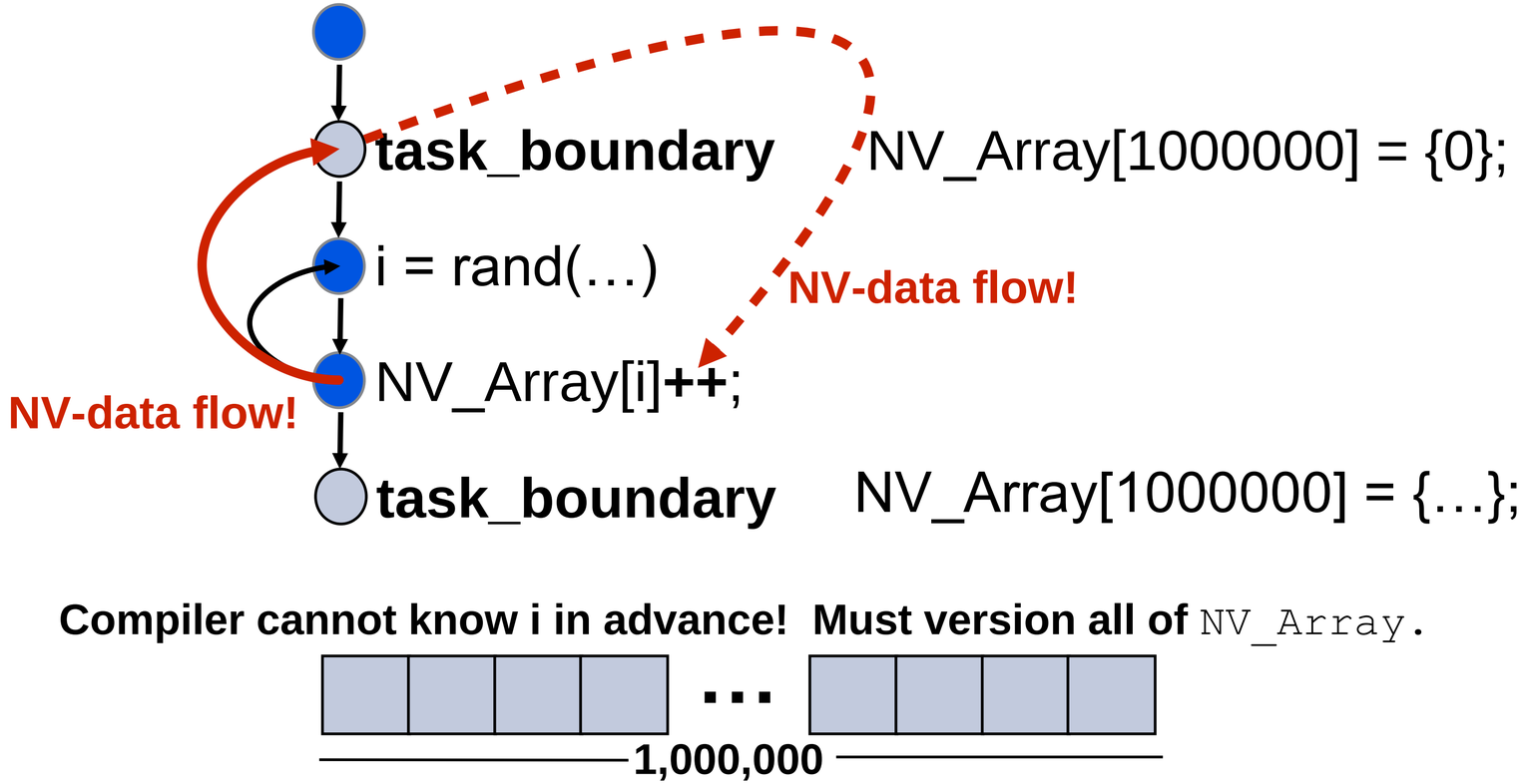
What to checkpoint here?
Assumptions about memory?

Ratchet [Hicks 2016]

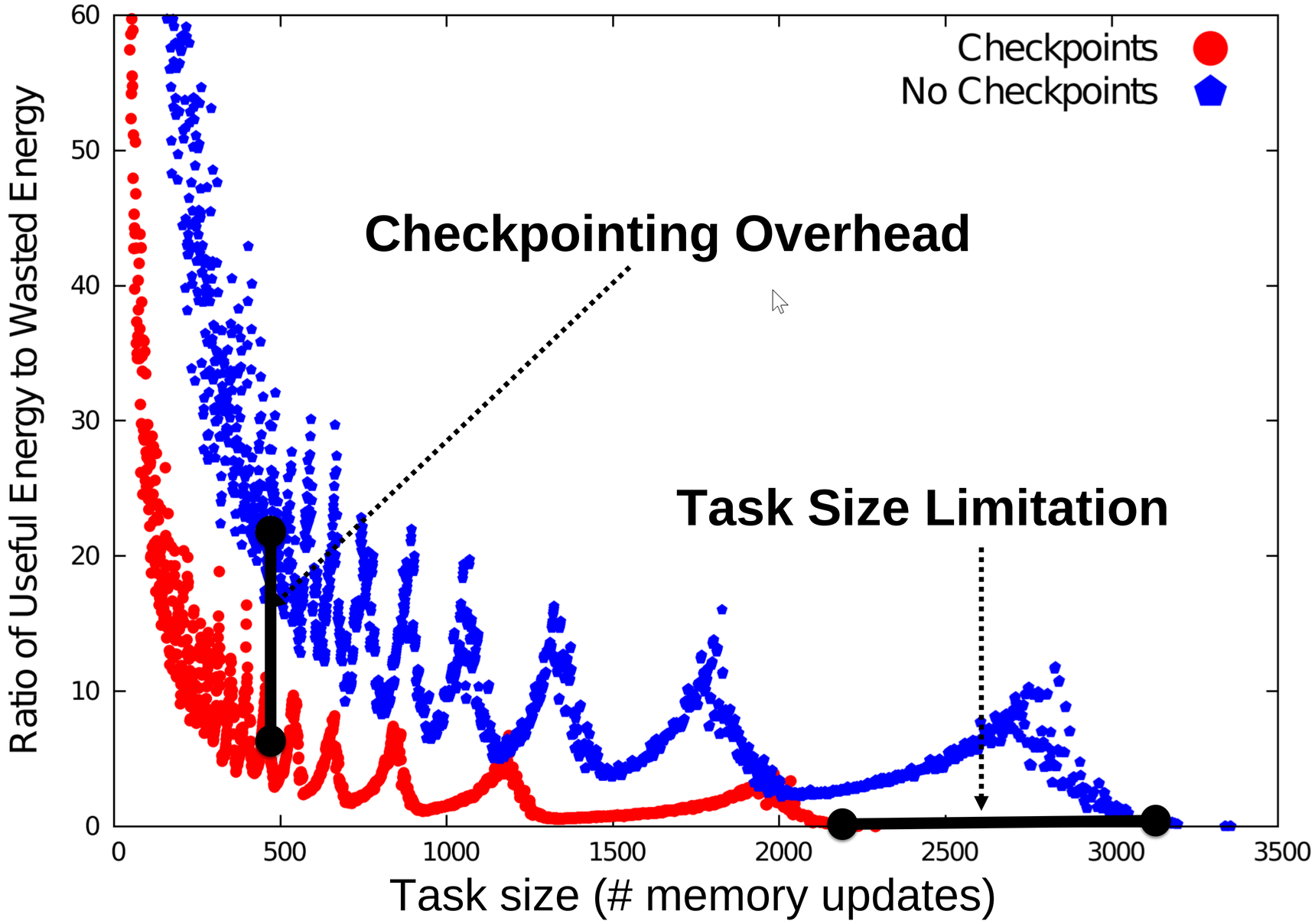
Insight: Write-After-Read dependences violate idempotenc

Alternative Scheme: Idempotent Checkpointing

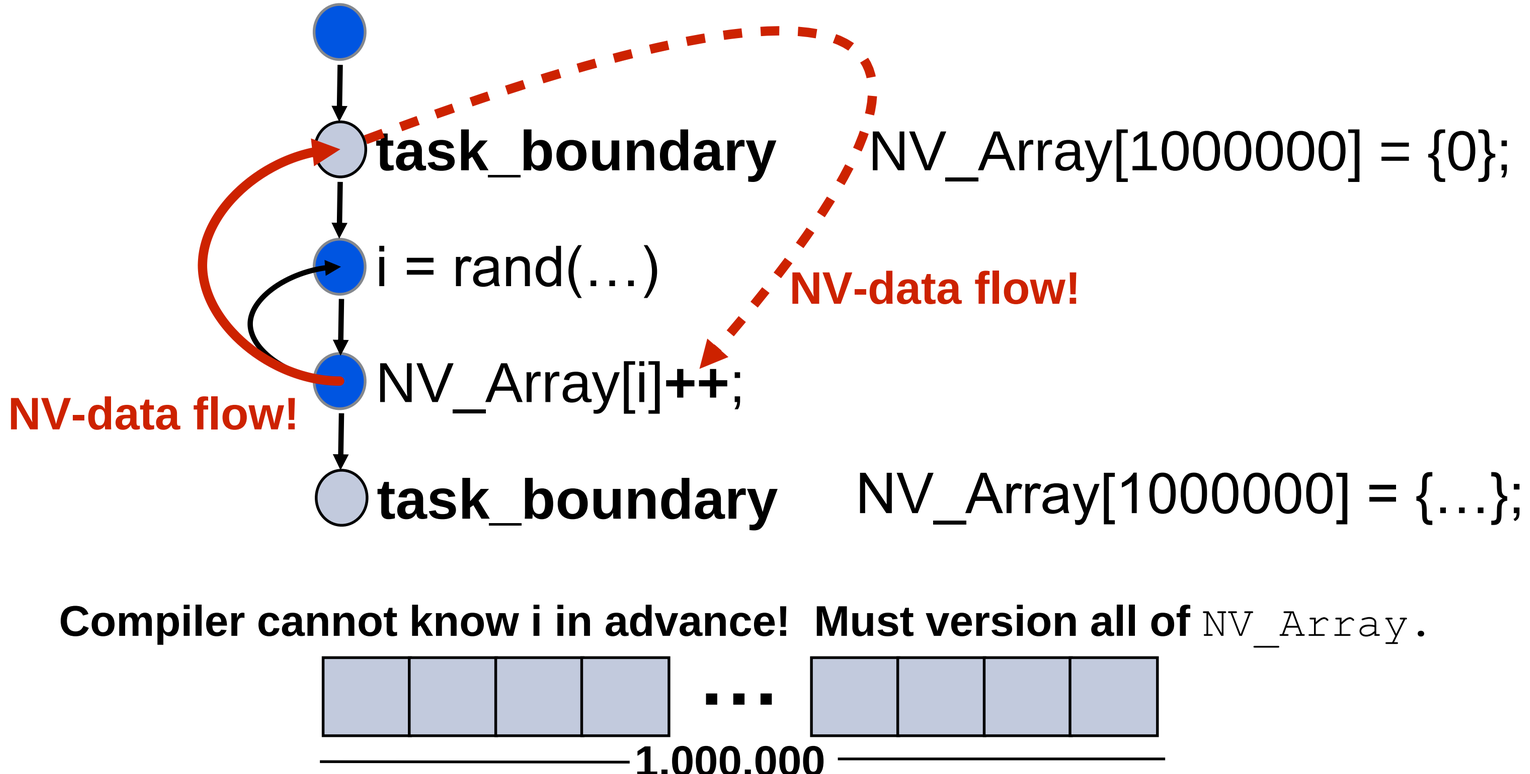
```
int NV_Array[1000000];
task_boundary
i = rand(1000000);

  NV_Array[i]++;

}
task_boundary
```



Fundamental limitation of checkpointing:
Checkpoints must keep volatile and possibly updated NV state consistent across reboots.


Checkpoints
No Checkpoints
Checkpointing Overhead
Task Size Limitation
Ratio of Useful Energy to Wasted Energy
Task size (# memory updates)
0
10
20
30
40
50
60
0
500
1000
1500
2000
2500
3000
3500

```
int NV_Array[1000000];
task_boundary
i = rand(1000000);

  NV_Array[i]++;

}
task_boundary
```



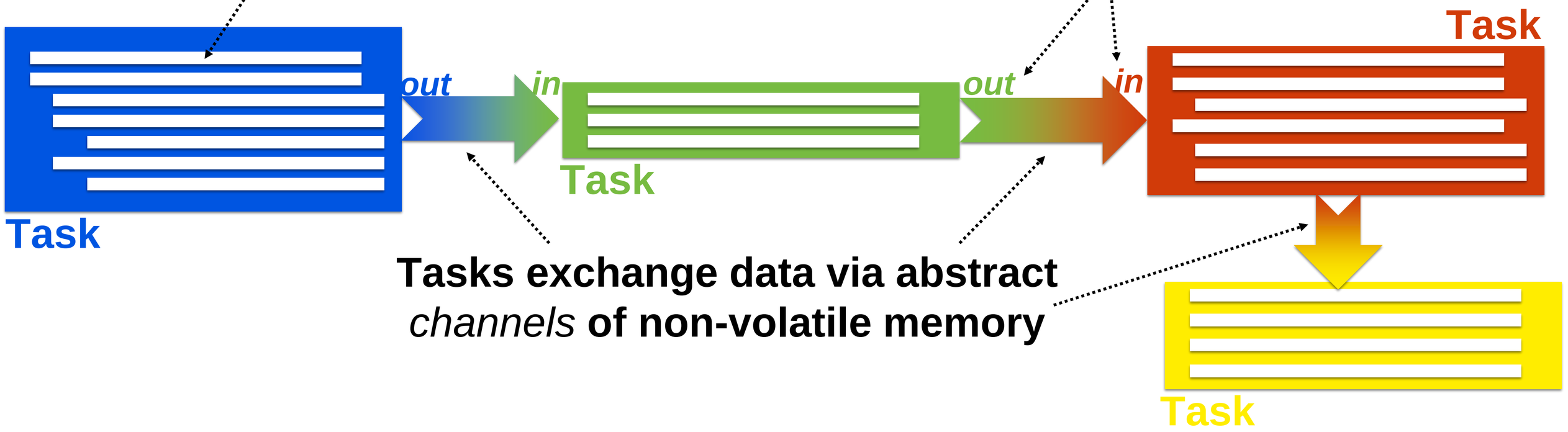
Programming Challenge:

"How much checkpointing overhead will I have?"

breaks the compiler abstraction boundary

Programmer explicitly defines tasks and task-flow graph in code

Channels *statically multiversion* data separating inputs from outputs

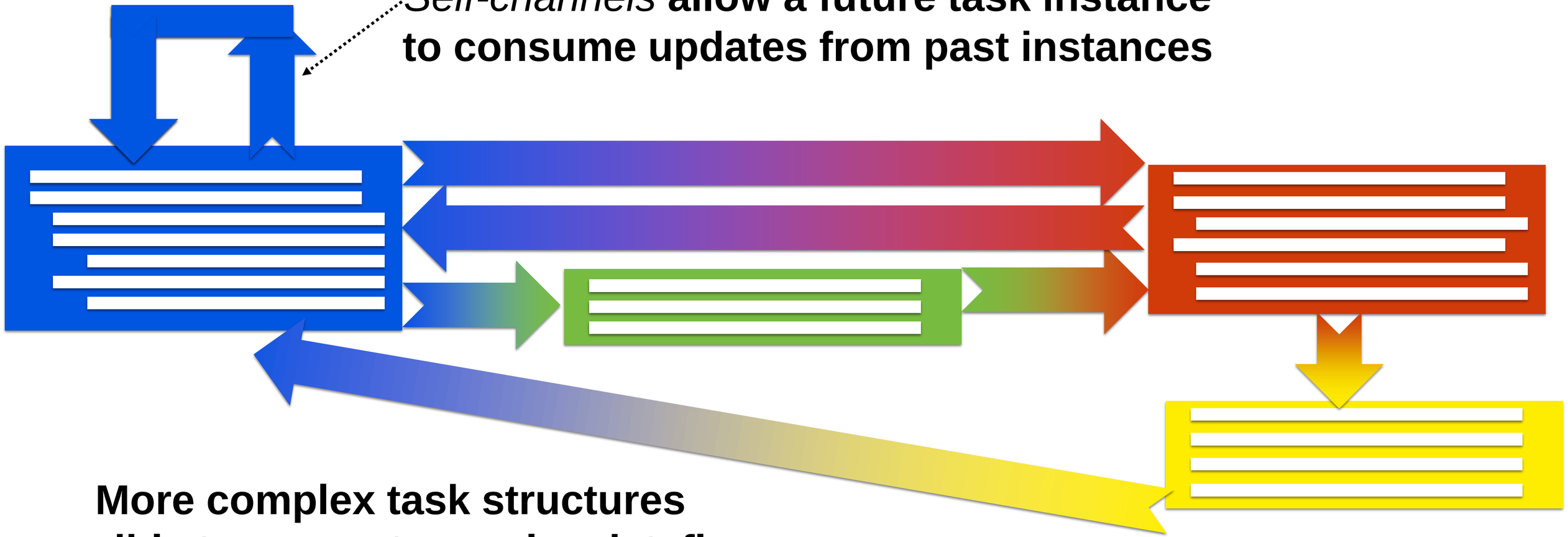


Tasks exchange data via abstract *channels* of non-volatile memory

Chain: Static data multi-versioning

[Colin, et al OOPSLA ’16]

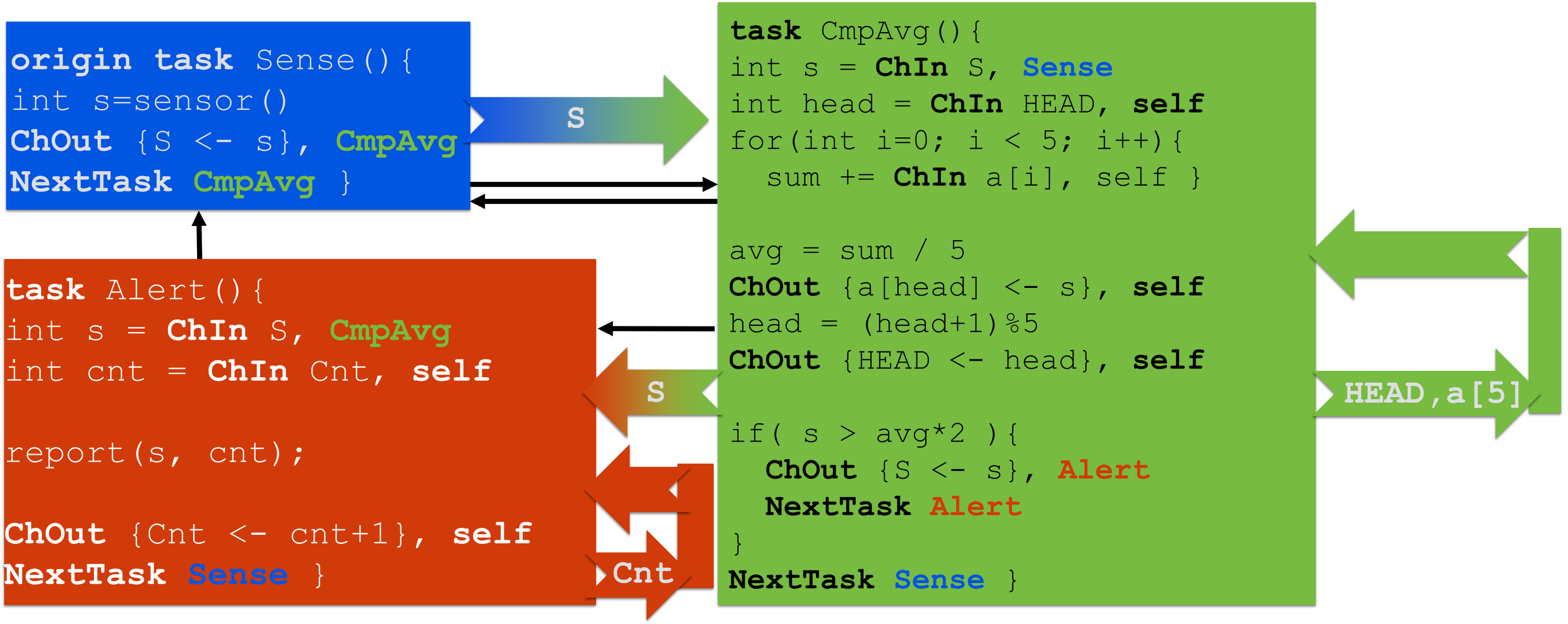
Self-channels **allow a future task instance to consume updates from past instances**



More complex task structures possible to support complex dataflow

Chain: Static data multi-versioning

[Colin, et al OOPSLA ’16]



Chain: Static data multi-versioning

[Colin, et al OOPSLA '16]

```
origin task Sense(){
 S=read_sensor()
NextTask CmpAvg }
```

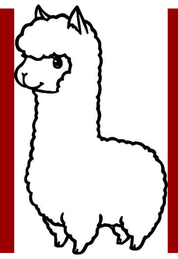
```
task CmpAvg(){
 for(int i=0; i < 5; i++){
   sum += A[i] }
 avg = sum / 5
 A[head] = S
 head = (head+1)%5

 if( S > avg*2 ){
   NextTask Alert
 }
NextTask Sense }
```

```
task Alert(){
 report(S, Cnt);
 Cnt++
NextTask Sense }
```

Function-as-task means no need to checkpoint stack & regs

```
Shared S, Cnt,
       A[5],head
```

Alpaca selectively privatizes task-shared variables



Alpaca: Tasks with Dynamic Privatization

[Maeng, et al OOPSLA ’17]

```
origin task Sense(){
S=read_sensor()
NextTask CmpAvg }
```

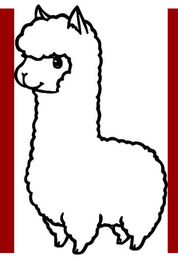
```
task Alert(){
priv_Cnt = Cnt
report(S, Cnt);
priv_Cnt++
2PCommit(Cnt) && NextTask
Sense }
```

```
Shared S, Cnt,
       A[5],head
```

```
task CmpAvg(){
priv_head = head
for(int i=0; i < 5; i++){
    sum += A[i] }
avg = sum / 5
priv_A[priv_head] = A[head]
markDirty(A[priv_head])
priv_A[priv_head] = S
priv_head = (priv_head+1)%5
if( S > avg*2 ){
  2PCommit(dirty(A), head)
  && NextTask Alert
}
2PCommit(dirty(A), head)
&& NextTask Sense }
```

Only privatize data read then written (e.g., S not privatized)

Array entries privatized on use



Alpaca: Tasks with Dynamic Privatization

[Maeng, et al OOPSLA '17]

```
origin task Light(){
light=light_sensor()
NextTask CmpAvg }
```

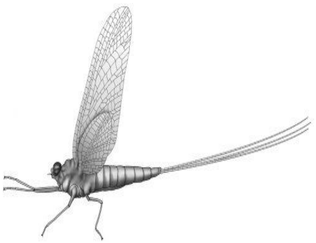
```
task isWet(light,moist,…){
  //Decide if the device
  //is wet
}
```

```
task Moist(){
moist=moisture_sensor()
}
```

```
Light->isWet (expires 10s)
Moist->isWet (expires 4m)
```

New idea: Timely Execution requiring stale inputs to expire

Requires intermittence-safe time-keeping (remanance timekeeping)

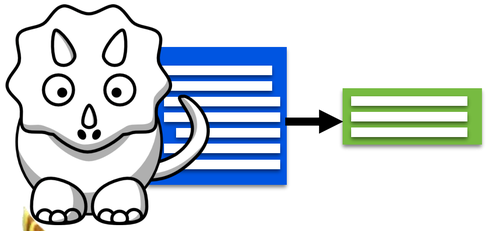


Mayfly: Timely Intermittent Execution

[Hester, et al SenSys ’17]

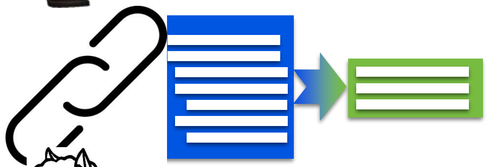
Electrical & Computer ENGINEERING

Mementos [Ransford, et al ‘11]

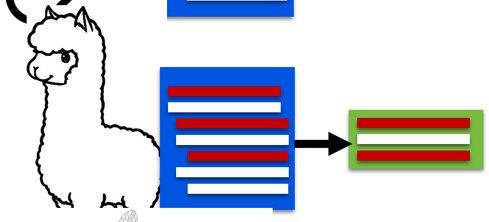


Dino

Ratchet [Hicks ‘16]



Chain



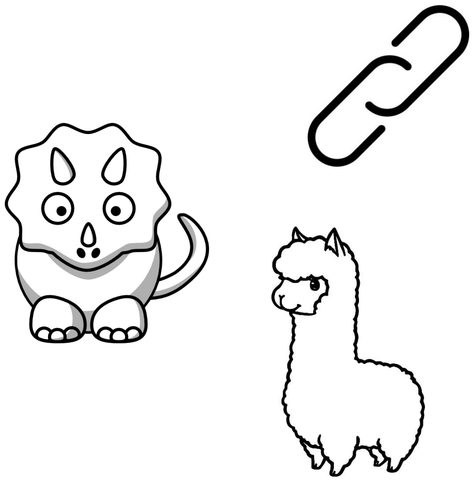
Alpaca

Mayfly [Hester et al ‘17]

Hardware assumptions?

Intermittence Bug: Out-of-thin-air Values
Intermittence Bug: Structural Inconsistenc
Intermittence Bug: IO Atomicity/Timelines
Intermittence Bug: Livelock/Progress
Intermittence Bug: Reactivity/Bursts

DINO, Chain, Alpaca Programming Model



Task, Channel Definitions

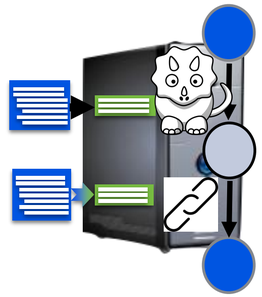
Task-aware Compiler

Task Data-flow Analysis

Link to Task Runtime

Channels, Versioning, Privatization

Runtime Library



DINO: Checkpoint & Recovery

Chain: Task Management

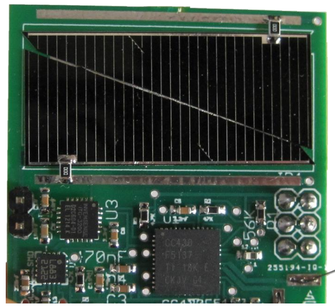
Alpaca: Privatization/Commit

Prototype Implementations

Energy-harvesting Platform Prototypes

WISP5 Energy-harvesting Device

RF Power Setup



Custom PCBs

Applications

Activity Recognition (accelerometer+ML)

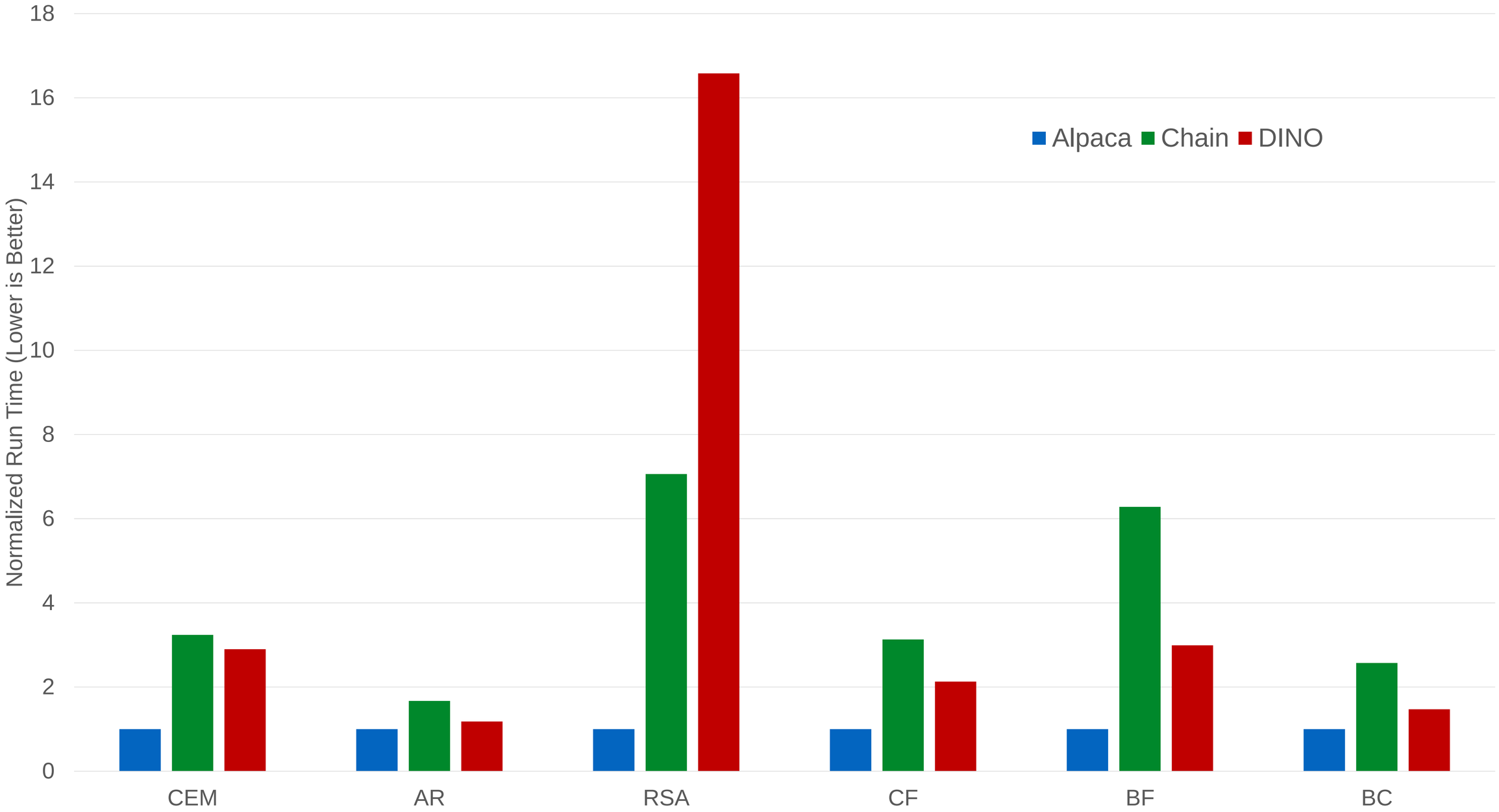
MIDI Natural User Interface

Cold-chain Equipment Monitor

Multi-granular Sensor Log

Key Result:

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


Normalized Run Time (Lower is Better)
18
16
14
12
10
8
6
4
2
0
Alpaca
Chain
DINO
CEM
AR
RSA
CF
BF
BC

Use Alpaca, Chain and DINO for your research!

http://intermittent.systems | http://github.com/CMUAbstract

https://cmuabstract.github.io/alpaca-landing-page/

Alpaca

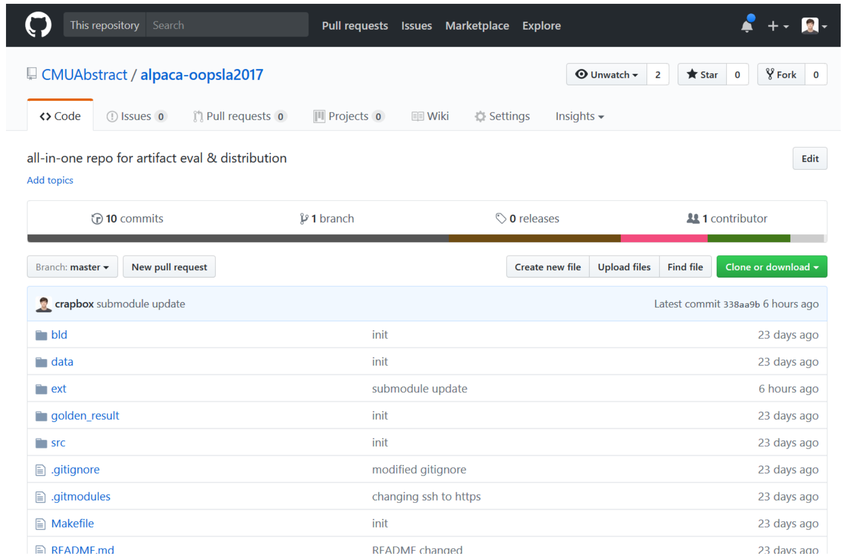
A programming language for writing reliable software for intermittent, energy-harvesting computer systems

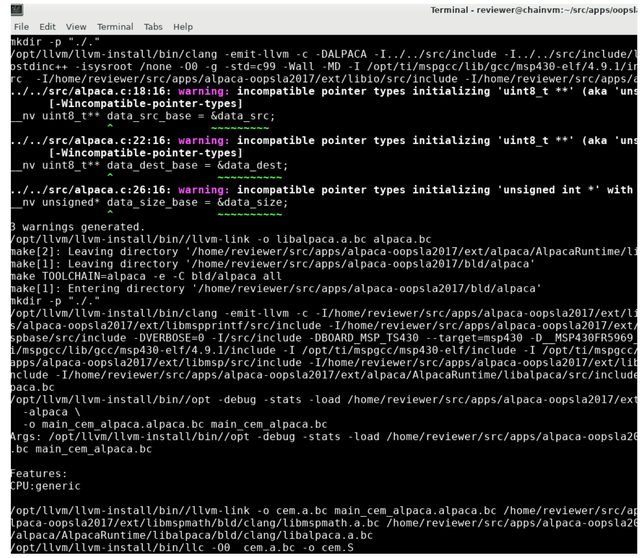
Download VM (10.67GB)

VM contains Alpaca and other previous state-of-the-art systems (Chain, DINO)
UserID: reviewer PW: review.

Pull source code from Github

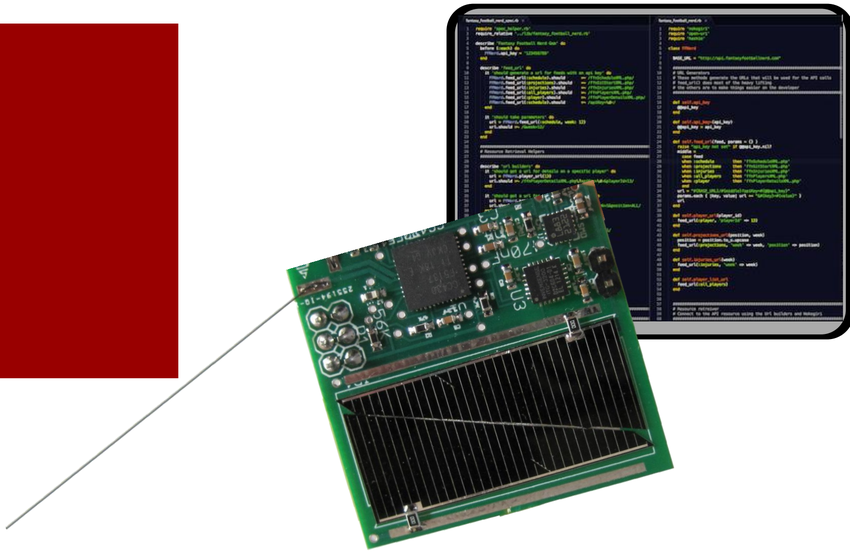
https://github.com/CMUAbstract/alpaca-oopsla2017





Debugging Intermittence

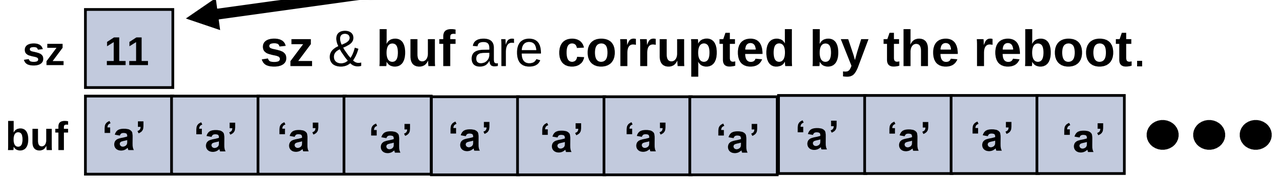
Toolchain Support for Understanding Intermittence



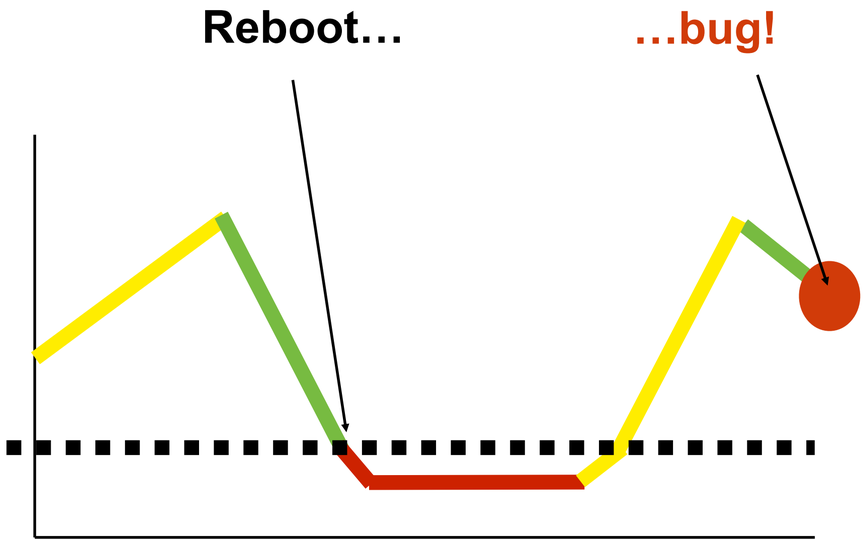
Intermittence Bugs Lead to Application Errors

[Colin, et al. CASES '15, ASPLOS '16, IEEE Micro Top Picks '16]

"Out of thin air" value not permitted by any continuous execution!

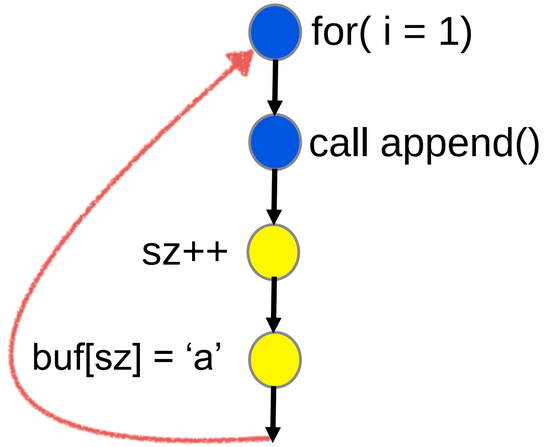


sz & **buf** are **corrupted by the reboot**.



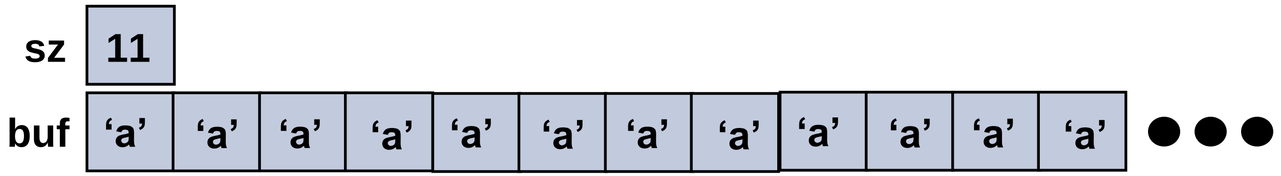
Failure manifests on intermittent power

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



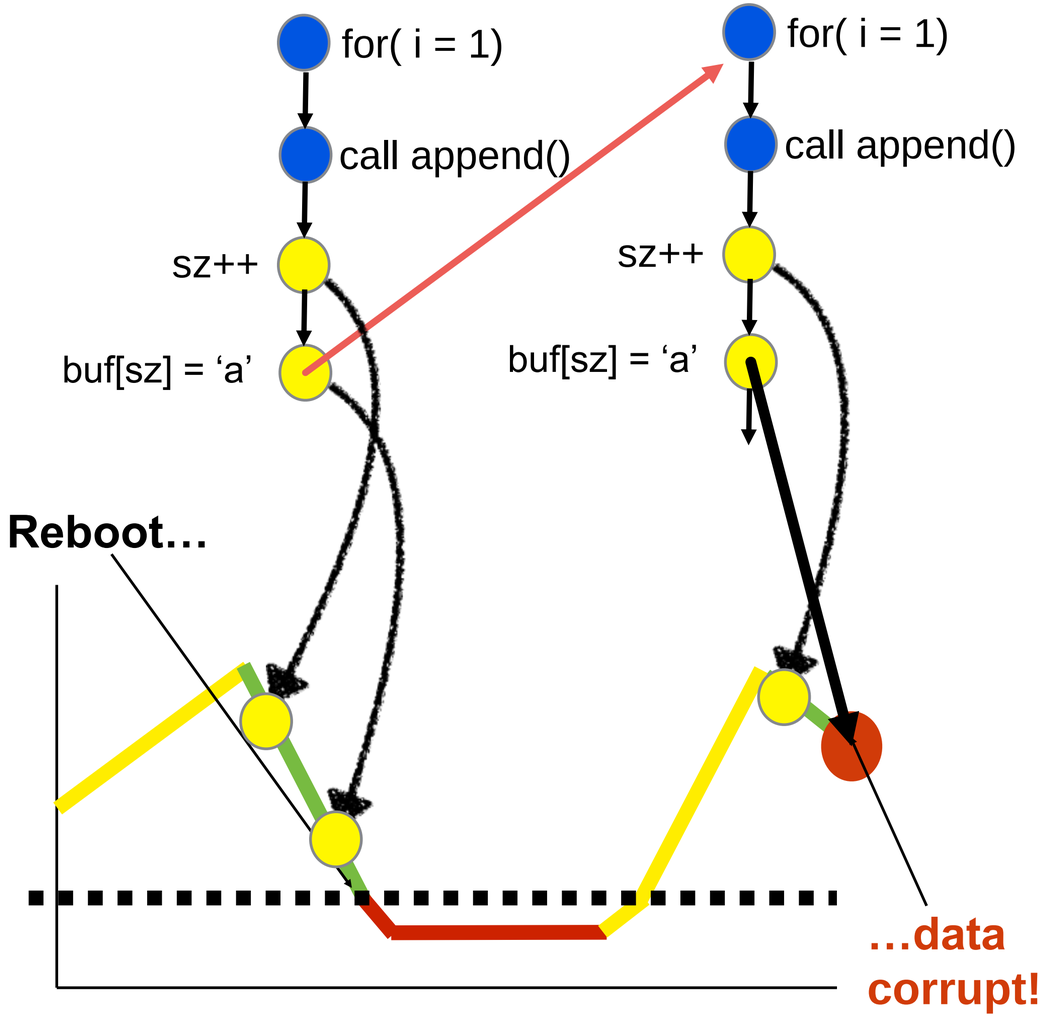
How Do We Debug *Intermittence Bugs*?

[Colin, et al. CASES '15, ASPLOS '16, IEEE Micro Top Picks '16]



Key Need: Correlate *program events* **with the device's** *energy state.*

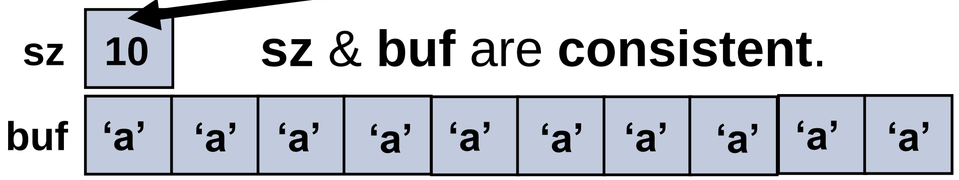
"What was happening when the power failed that led to weird behavior?"

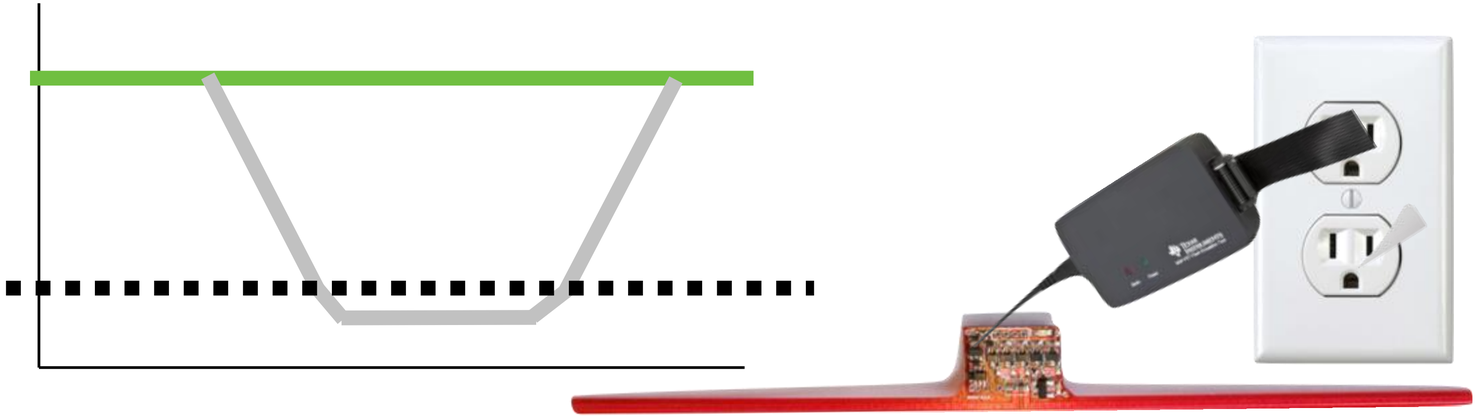


Energy-interference Precludes Conventional Debuggers

[Colin, et al. CASES '15, ASPLOS '16, IEEE Micro Top Picks '16]

The buggy, "out of thin air" value never shows up with continuous power!





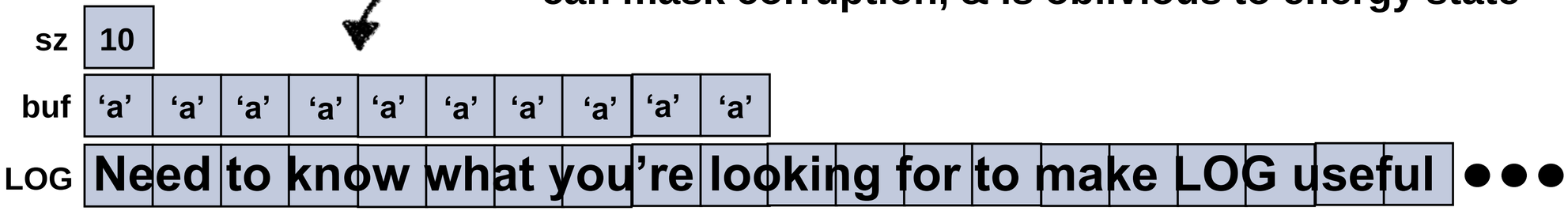
Debugger *provides* **power,**
masks **intermittence**

No assertions
No break points
No tracing
No interactive debugging
No energy tracing
No debug-time I/O

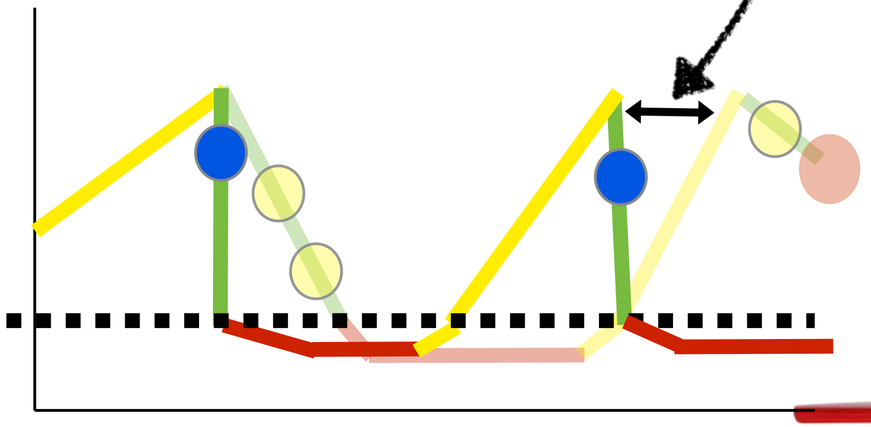
Energy-interference Complicates Instrumentation

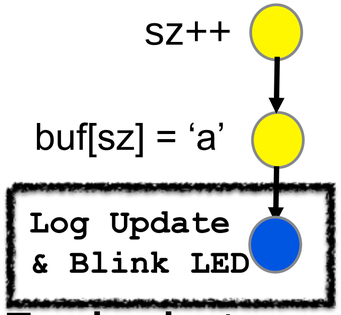
[Colin, et al. CASES '15, ASPLOS '16, IEEE Micro Top Picks '16]

Instrumentation changes energy behavior, can mask corruption, & is oblivious to energy state

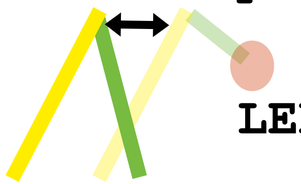


"Energy-interference gap"





Tracing instrumentation



Tracing code *consumes energy & changes behavior*

`printf`: UART or other I/O + msg handling code

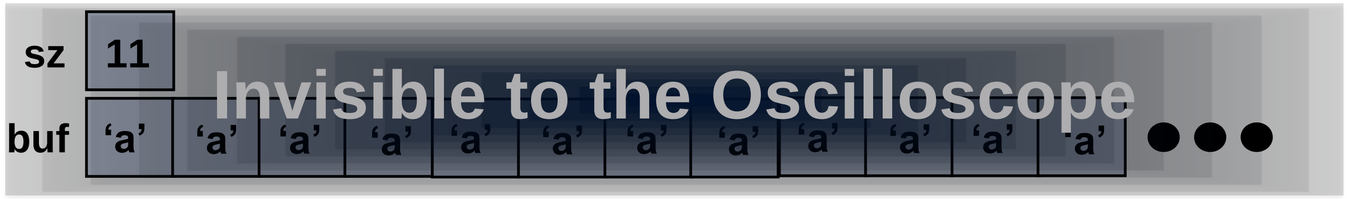
`LED`: up to **5x** increase in current draw w/ LED

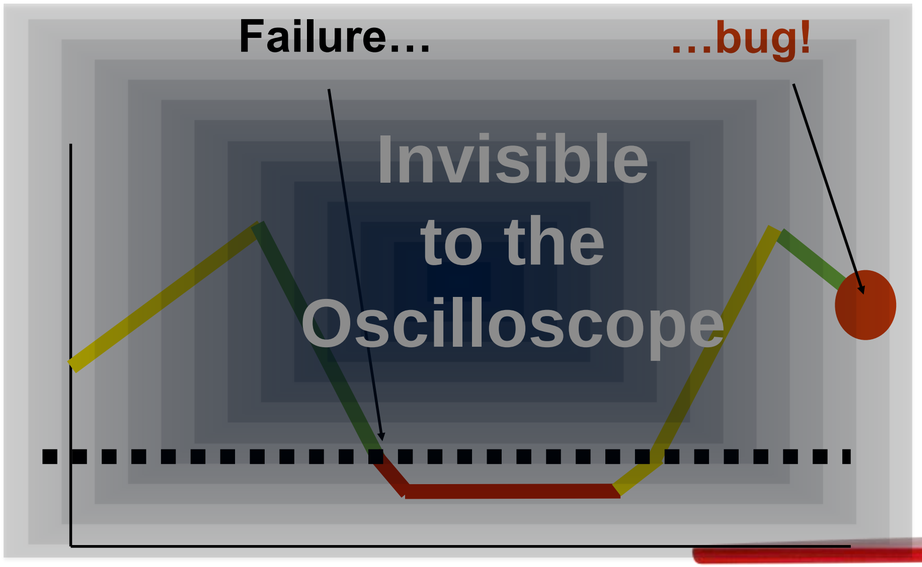
`Logs`: eats NV storage + high access cost

Existing Energy-interference-free Tools are Inadequate

[Colin, et al. CASES '15, ASPLOS '16, IEEE Micro Top Picks '16]

Corrupt data is device-internal and does not necessarily affect external signals





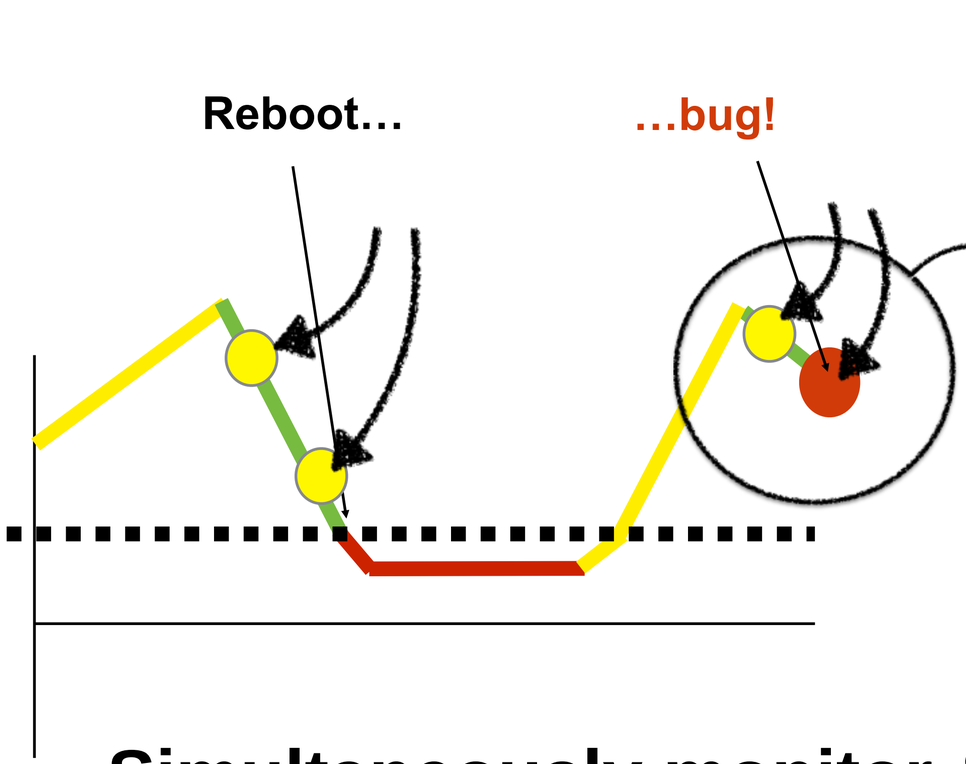
Oscilloscope: *expensive* **and** *limited to external signals*

No assertions
No break points
No tracing
No interactive debugging
No energy tracing
No debug-time I/O

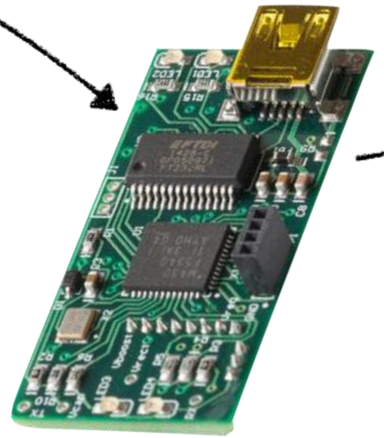
EDB: An Energy-interference-free Debugging Toolchain

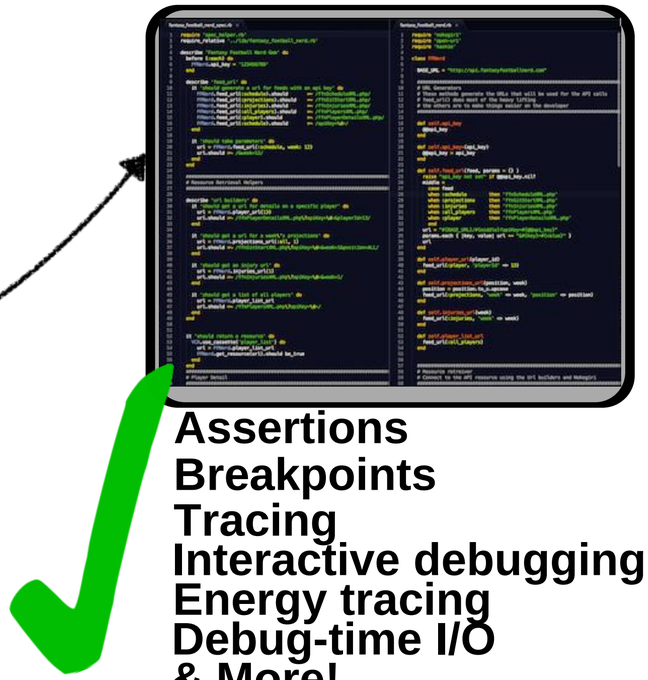
[Colin, et al. CASES '15, ASPLOS '16, IEEE Micro Top Picks '16]

We built *energy-interference-free* support for *debugging and monitoring* intermittent, energy-harvesting computers.



Energy-interference-free debugger



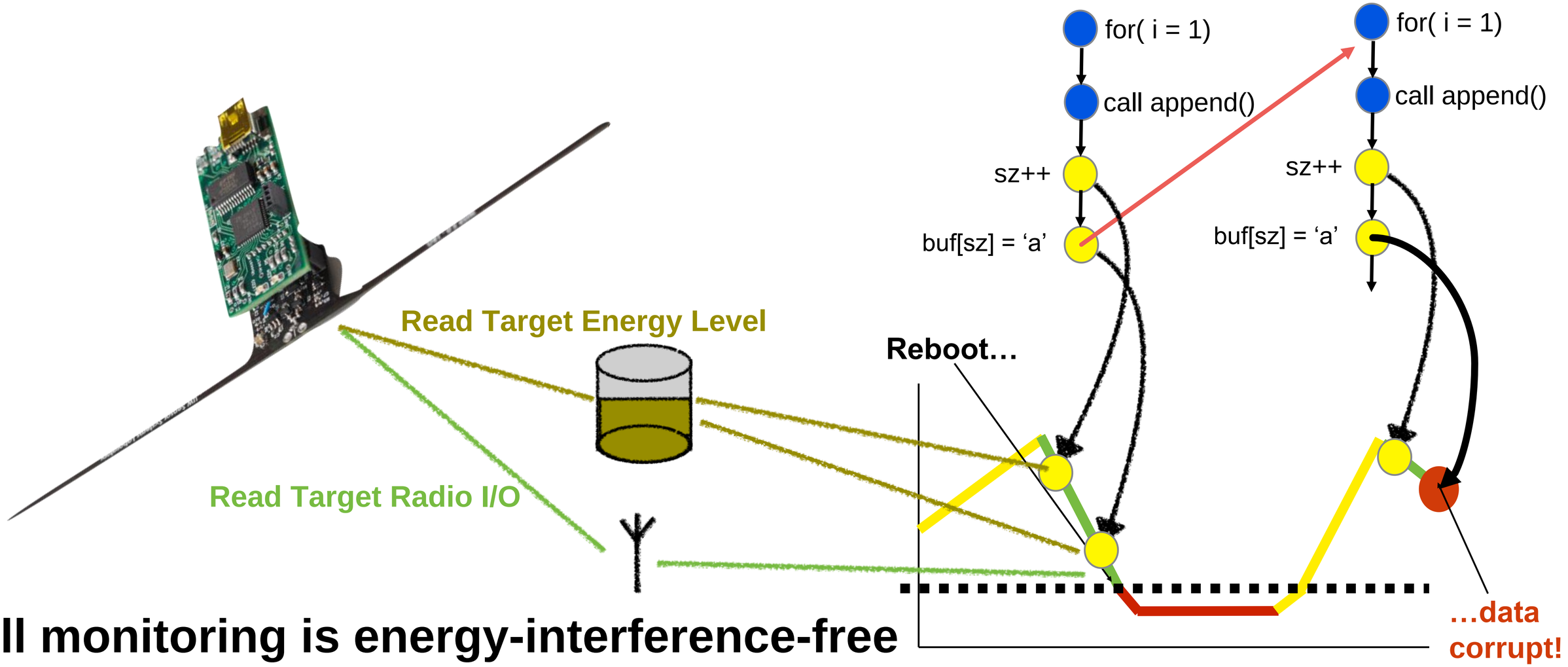


Assertions
Breakpoints
Tracing
Interactive debugging
Energy tracing
Debug-time I/O
& More!

Simultaneously monitor & manipulate *program state* and *energy state*.

Monitoring Target Energy & Device State

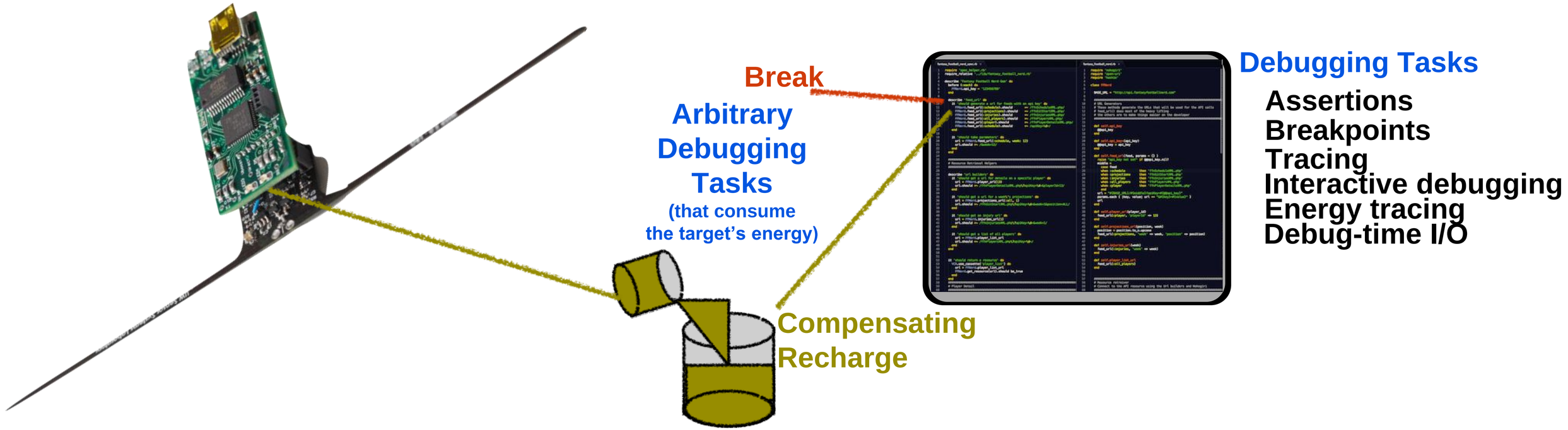
[Colin, et al. CASES ’15, ASPLOS ’16, IEEE Micro Top Picks ‘16]



All monitoring is energy-interference-free

Manipulating Energy State of the Device

[Colin, et al. CASES ’15, ASPLOS ’16, IEEE Micro Top Picks ‘16]

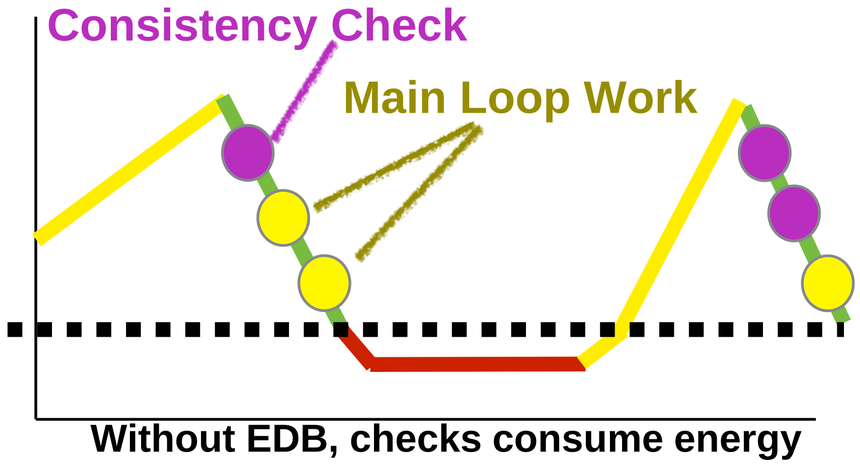


We charge **or** discharge **the device to** compensate **for expensive debugging tasks.**

Case Study: Expensive Consistency Checks Can Kill You

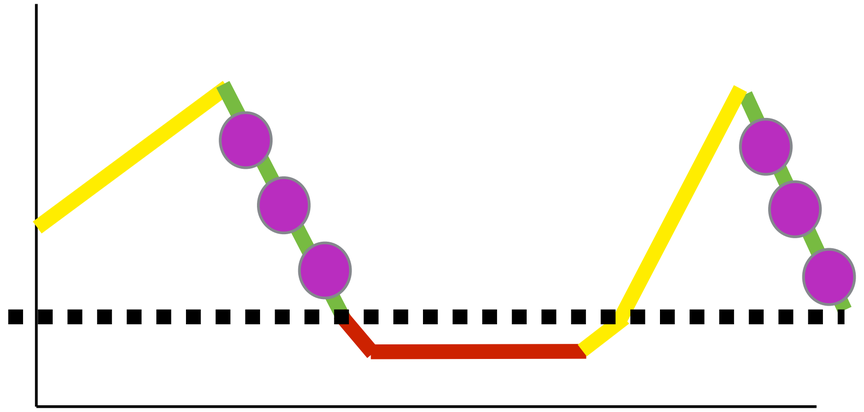
[Colin, et al. CASES '15, ASPLOS '16, IEEE Micro Top Picks '16]

One list element to check



Without EDB, checks consume energy

Many list elements to check



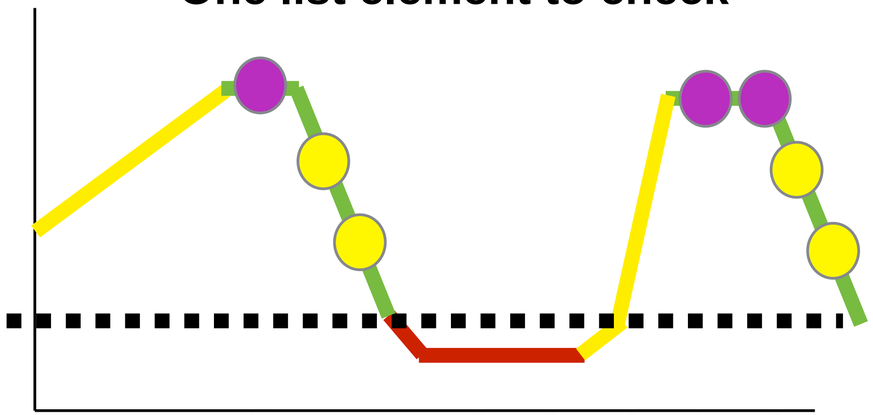
Eventually, checks prevent progress.

```
insert_element()

foreach(elem e){
  assert(e.next.prev == e) }
```

Case Study: EDB Enables Expensive Checks

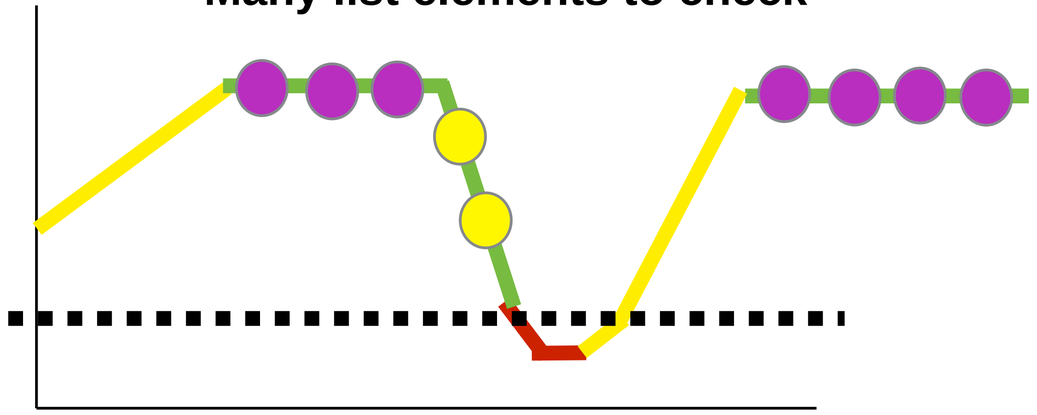
[Colin, et al. CASES ’15, ASPLOS ’16, IEEE Micro Top Picks ‘16]

One list element to check



EDB provides energy for the check only

Many list elements to check

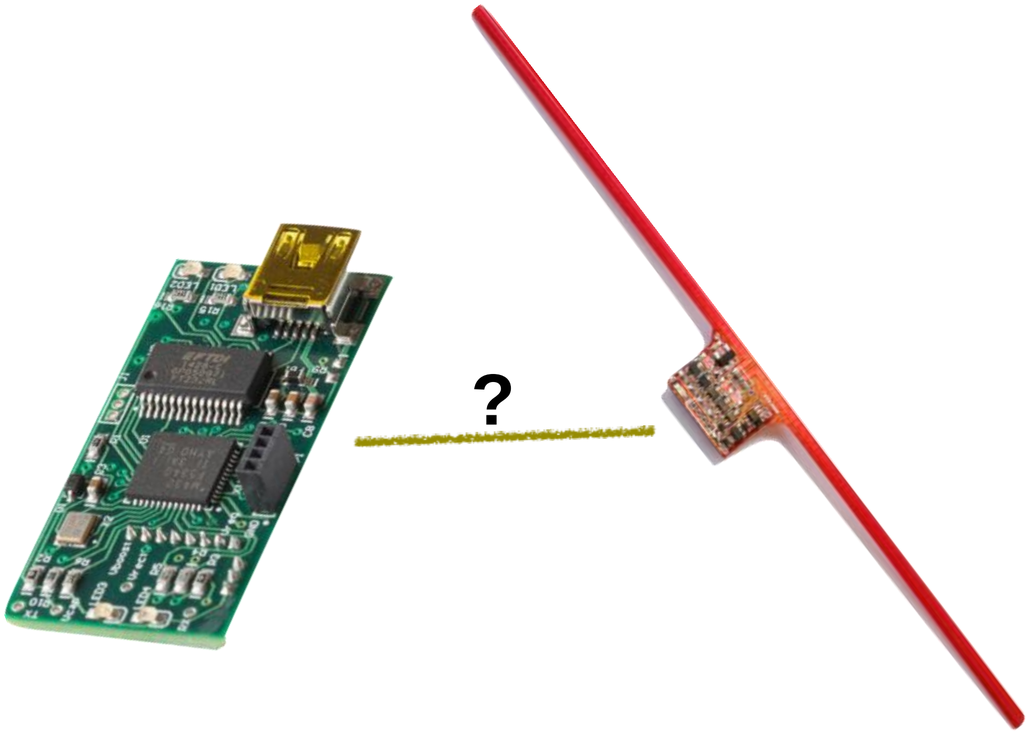


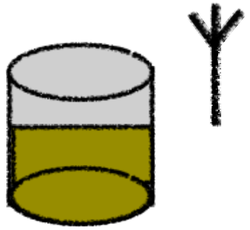
Eliminating energy interference enables expensive checks

```
insert_element()

foreach(elem e){
  assert(e.next.prev == e) }
```

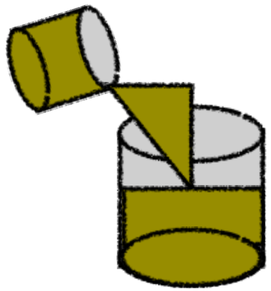
How Energy-interference-free?

[Colin, et al. CASES ’15, ASPLOS ’16, IEEE Micro Top Picks ‘16]





Passive monitoring E interference: ~0.2% of active power
(worst case)



Compensating charge error: ~4.3% of total E
(in a 47uF cap)

From the circuits up, EDB is energy-interference-free

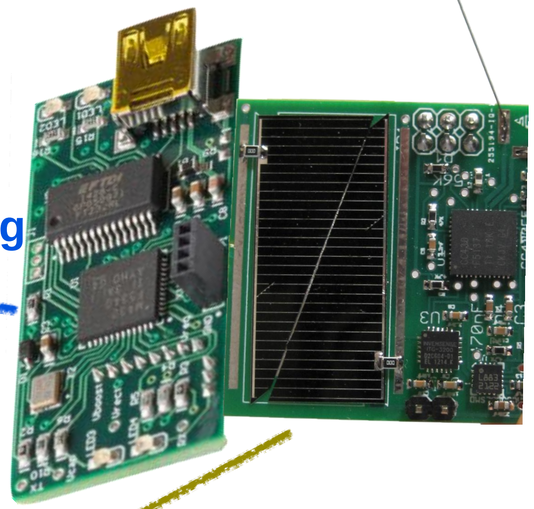
Integrated EDB Support for Deployed Diagnostics

[Colin, et al. CASES ’15, ASPLOS ’16, IEEE Micro Top Picks ‘16]

EDB can be powered to make test framework reliable

I/O Tracing

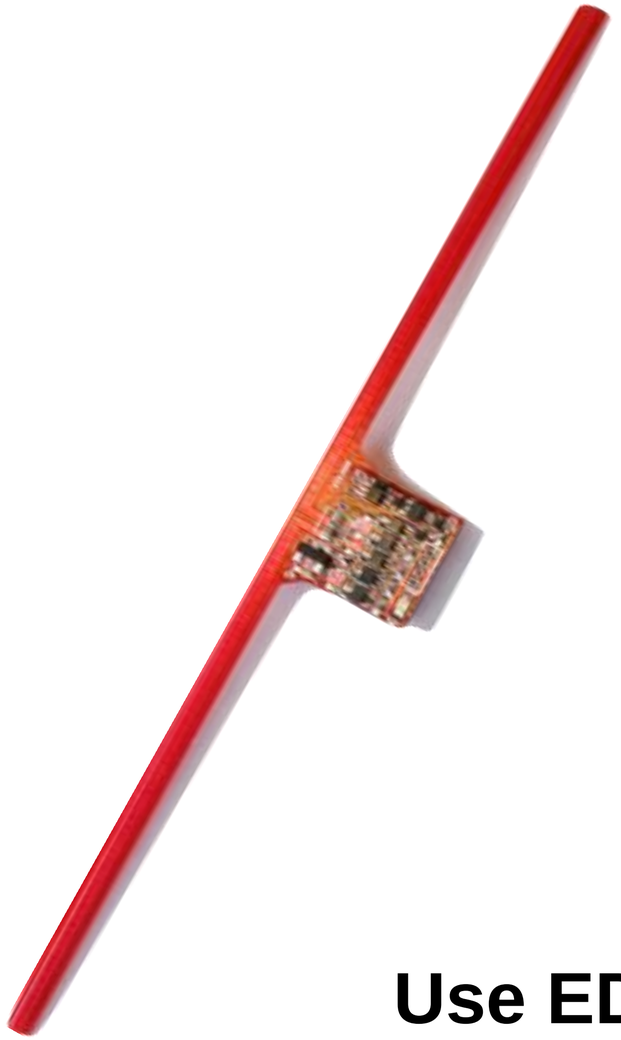
Program Tracing

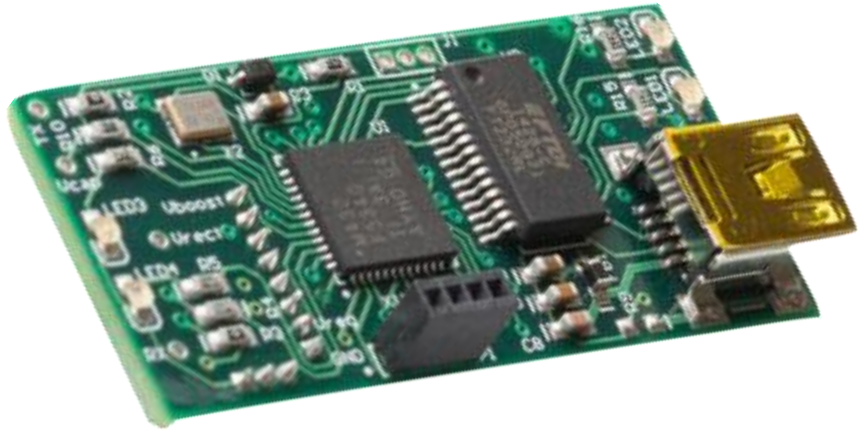


Energy Tracing & Manipulation

Integrating **EDB with an application board’s design enables beta testing & diagnostics in a** realistic target environment

EDB is freely available to researchers!





Use EDB for your research! Email us for hardware!

http://intermittent.systems | http://github.com/CMUAbstract/edb-rat

The Intermittent Execution Model

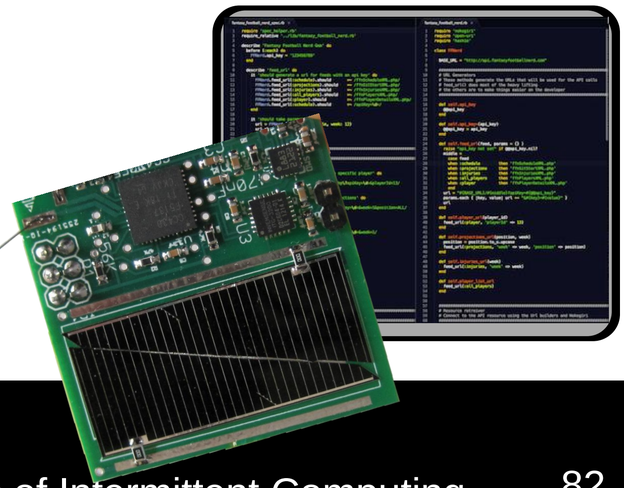
Reasoning About Intermittently-powered Devices

Designing for Intermittence

System Support for Reliable Intermittence

Intermittence Debugging

Toolchain Support for Understanding Intermittence

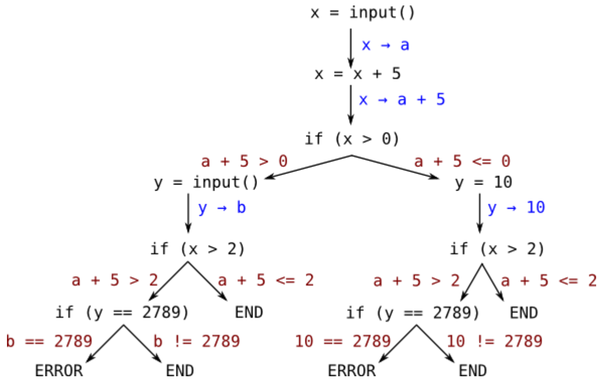


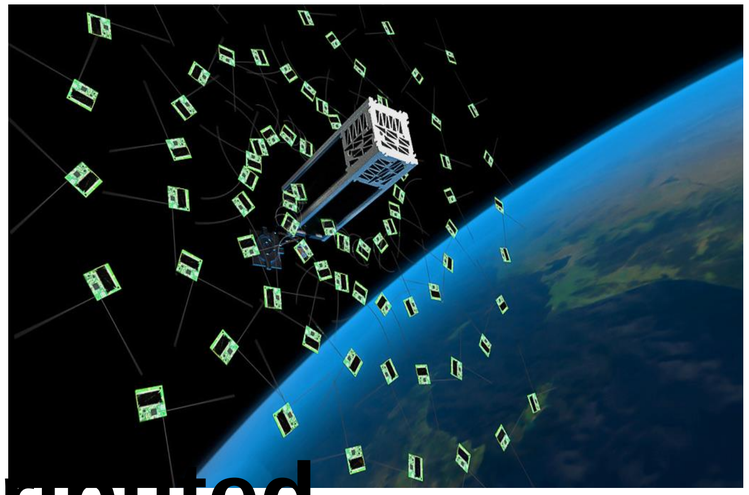
Future Challenges

Intermittent Computing Research Directions

Intermittent Verification: Is my program correct?

Key challenge: modeling intermittent behavior





Energy Modeling: How much energy does my code need?

Key challenge: platform energy varies

Distributed Coordination: Can devices cooperate?

Key challenge: communication & dist. state

Solving the System Challenges of Intermittent Energy-harvesting Devices

Brandon Lucia | Assistant Professor, CMU ECE

ABSTRACT Research Group - https://intermittent.systems

with PhD Students: Alexei Colin, Kiwan Maeng, Emily Ruppel, Vignesh Balaji
Graham Gobieski, Milijana Surbatovich

MS Students: Preeti Murthy, Amanda Marano, Neil Ryan, Devon White, Chris Meredith

BS Students: Graham Harvey, Mark McElwaine, Hannah Tomio

Industry Collaborators: Zac Manchester (Harvard/KickSat)
Alanson Sample (Disney Research Pittsburgh)