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

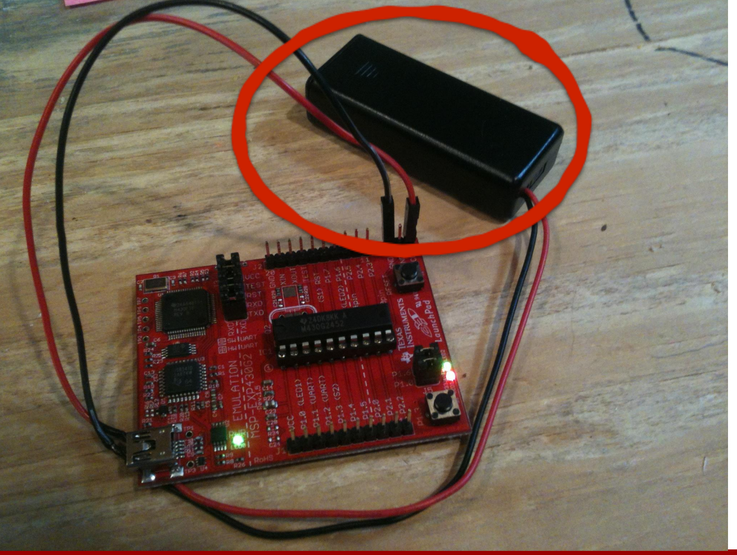
BS Students: Graham Harvey, Mark McElwaine, Hannah Tomio

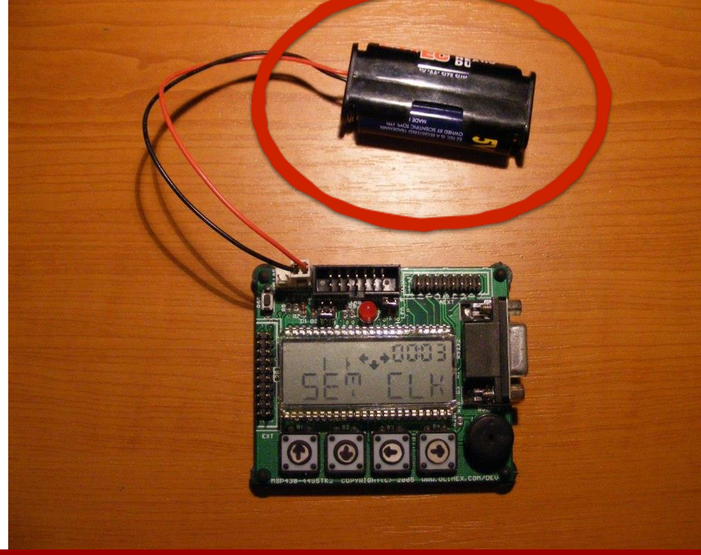
Industry Collaborators: Zac Manchester (Harvard/KickSat)
Alanson Sample (Disney Research Pittsburgh)



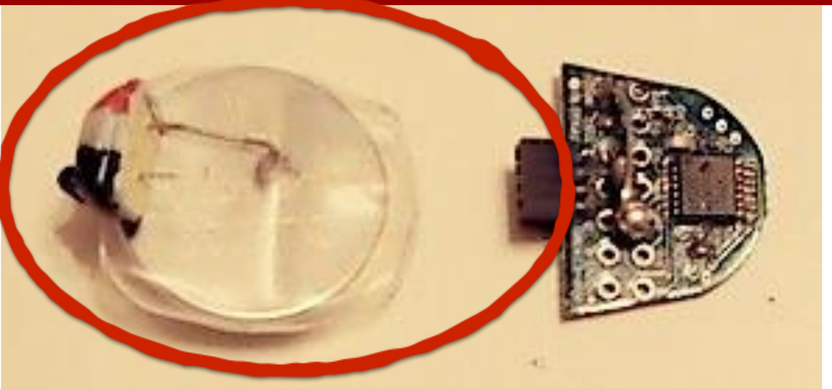
Emerging Devices Everywhere

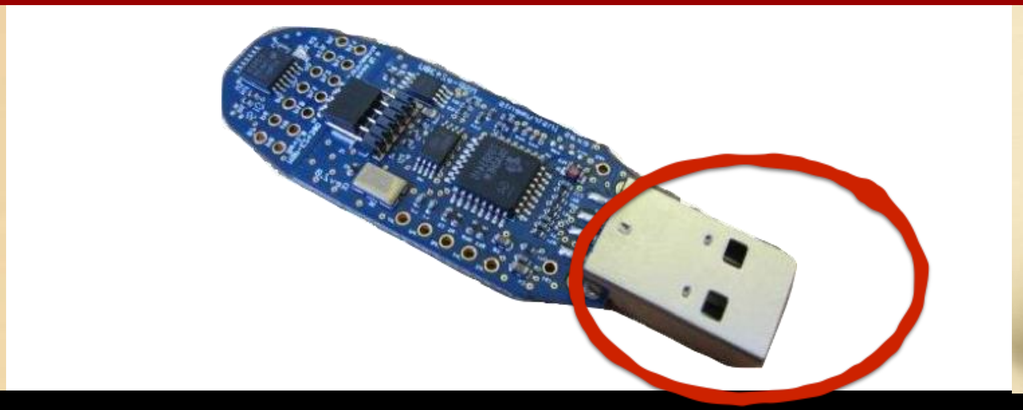
Wearables, sensors, “Internet of Things”, tiny satellites, medical implants, environmental monitoring, security, computational art, interactive clothing, smart furniture, smart carpets, smart toilet paper

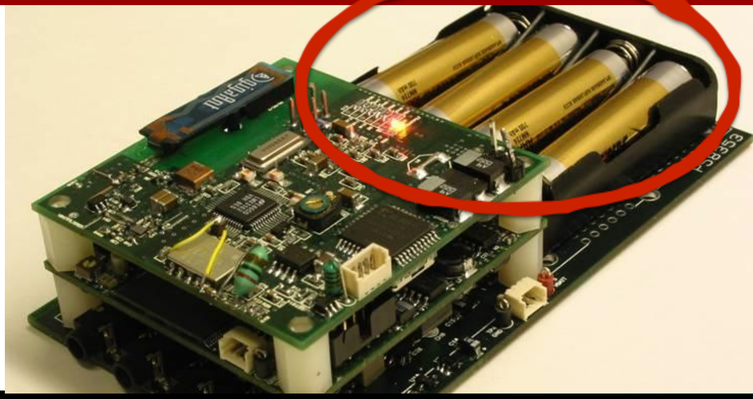


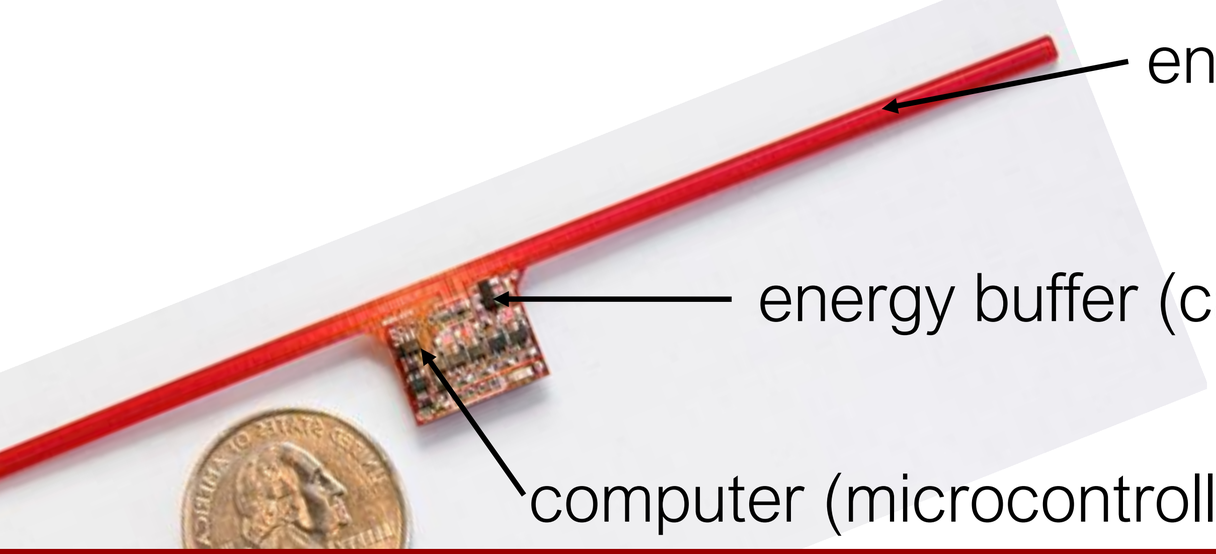


Tethered to a power source

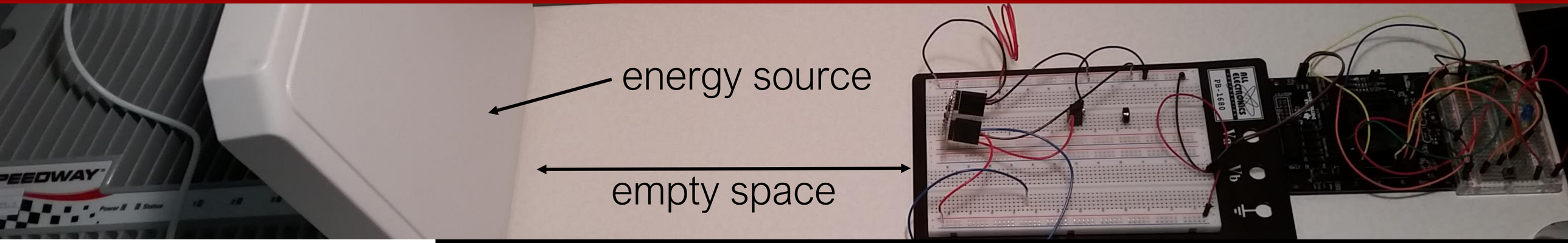


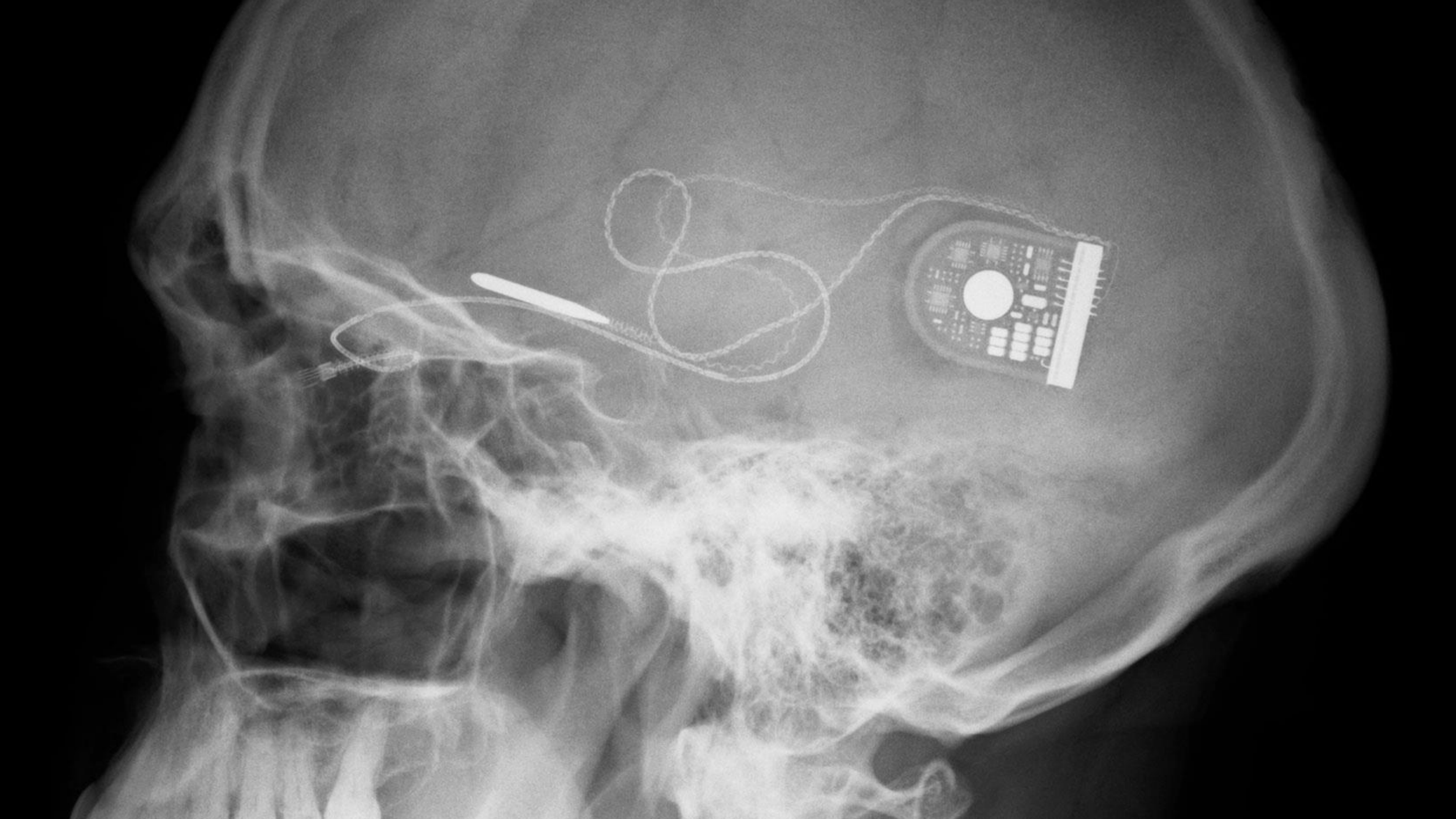


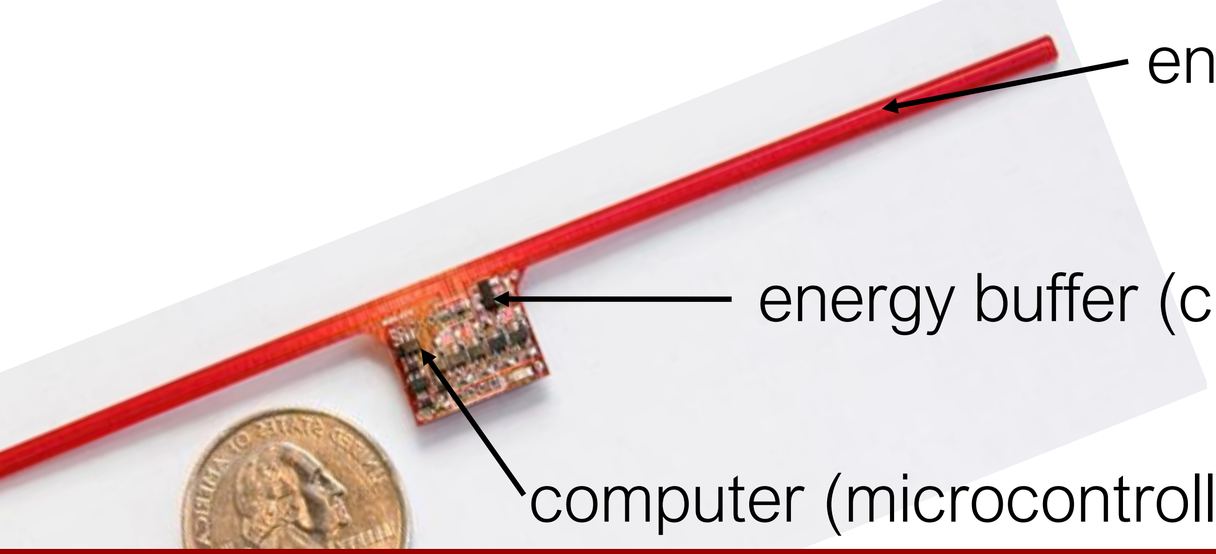




Energy harvesting untethers devices



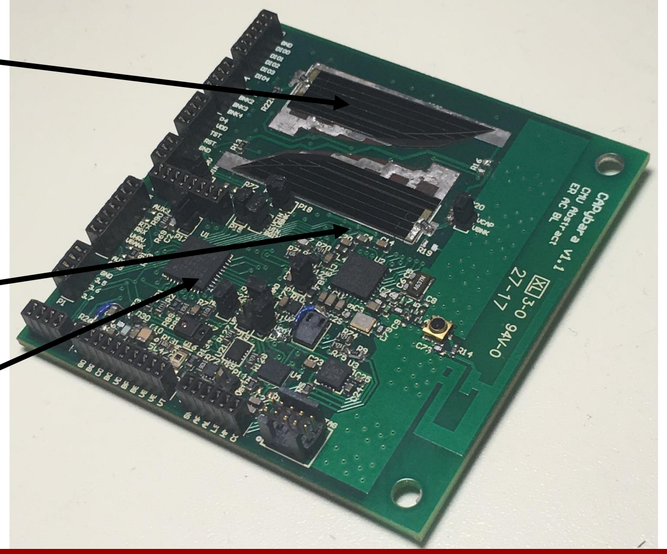




energy receiver

energy buffer (capacitor)

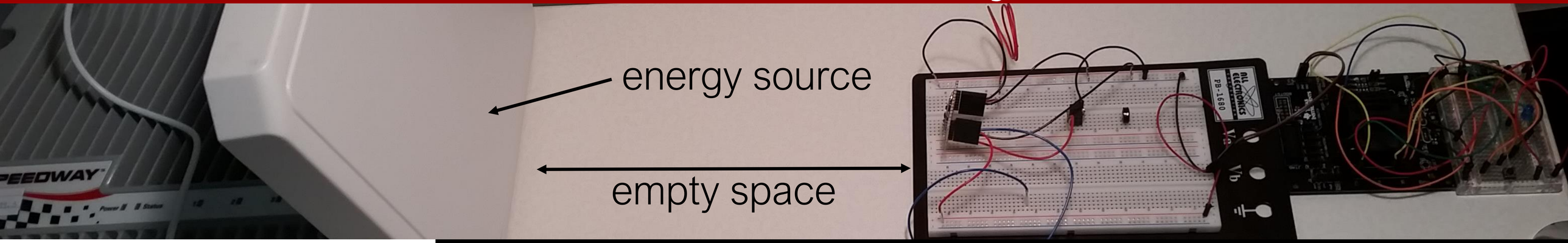
computer (microcontroller)



…but energy is **intermittent**

—and as a result—

software becomes **inherently unreliable.**



energy source

empty space

The Intermittent Execution Model

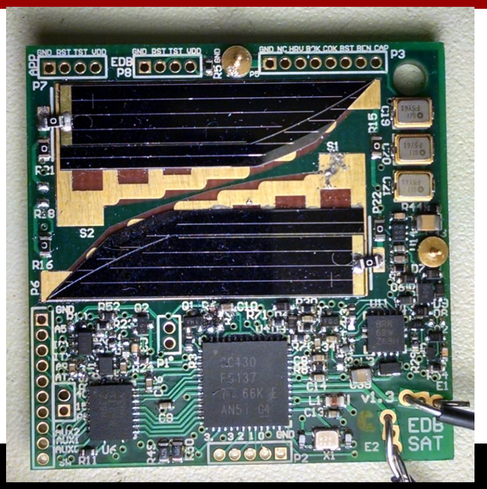
Reasoning About Intermittently-powered Devices

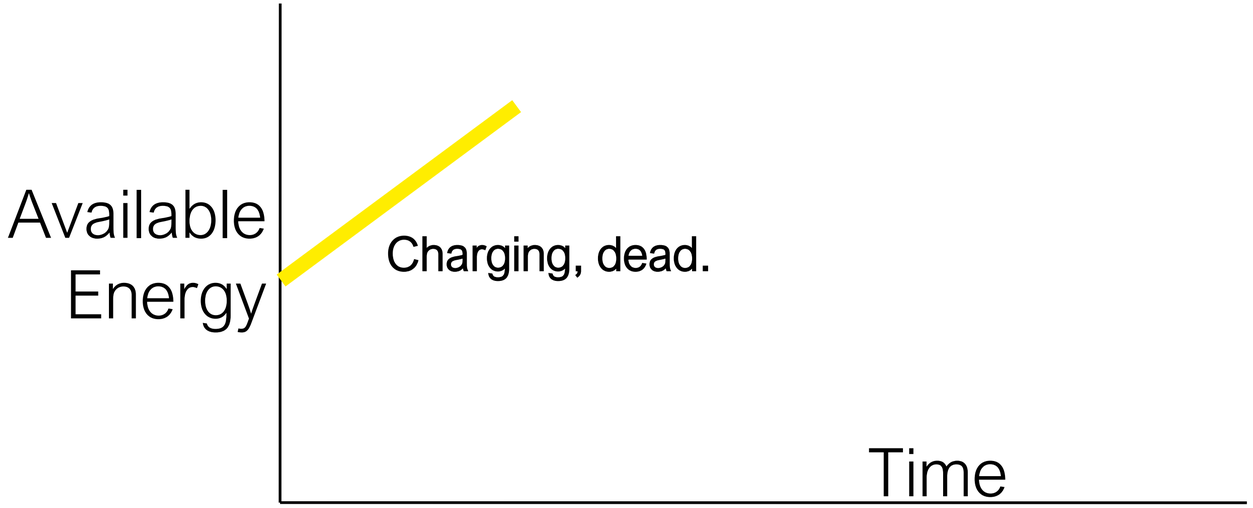
Designing for Intermittence

System Support for Reliable Intermittence

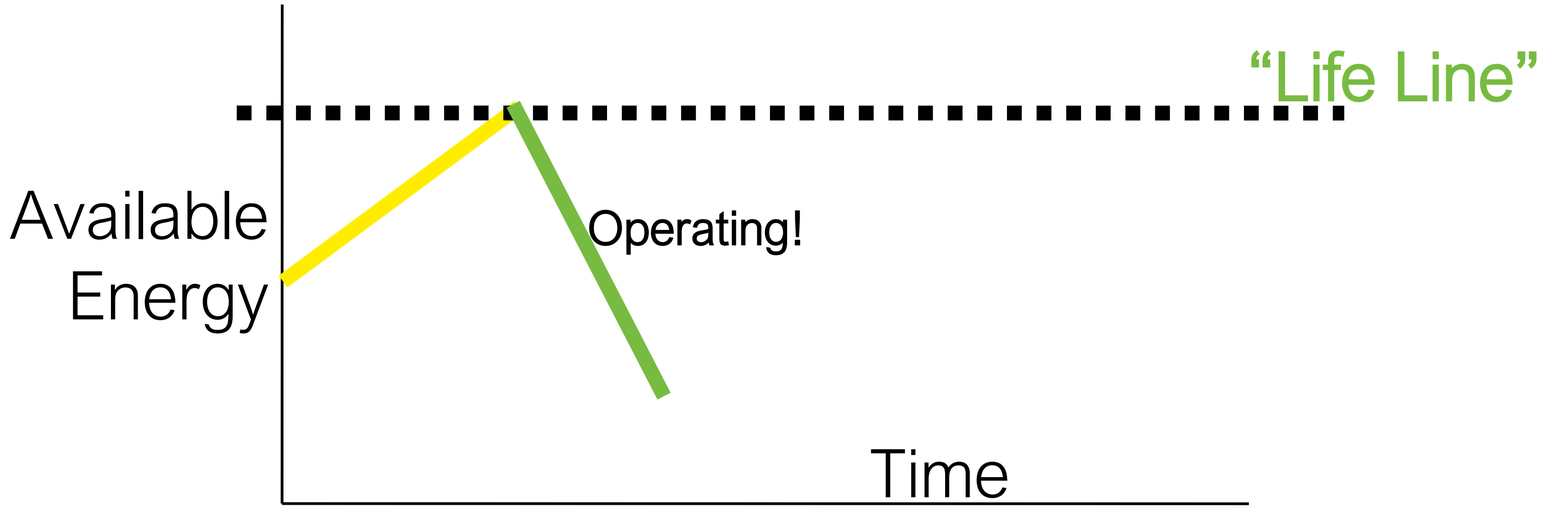
Intermittent Computing Platforms

Hardware, Programming Models, System Software

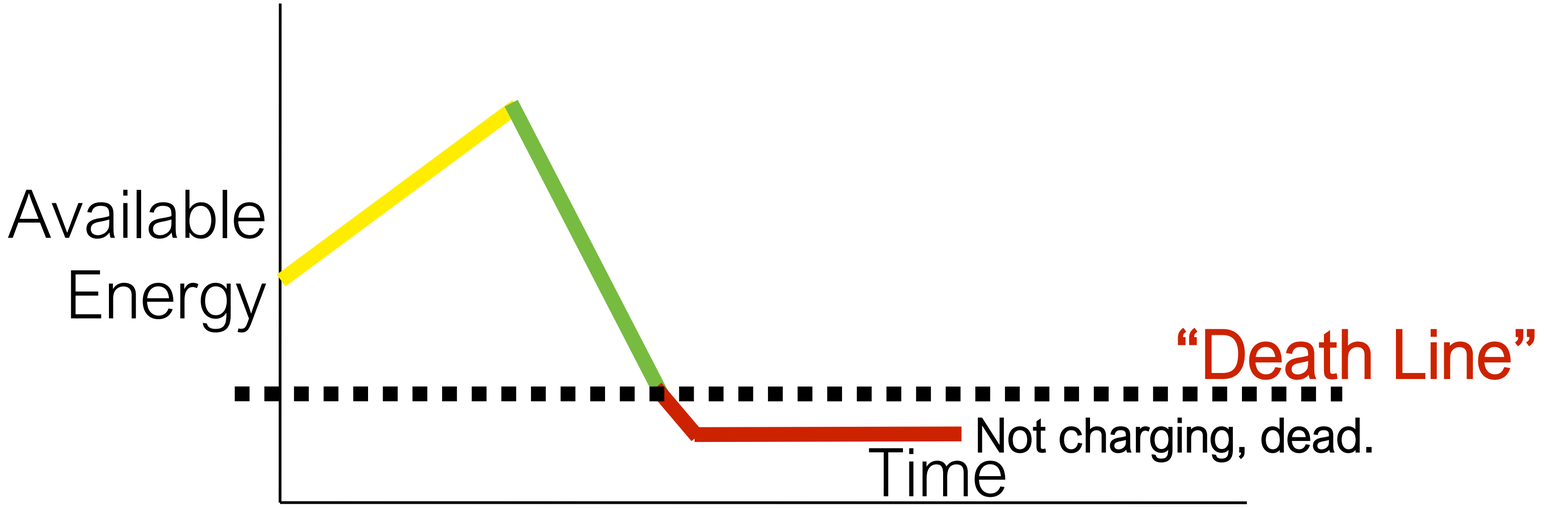




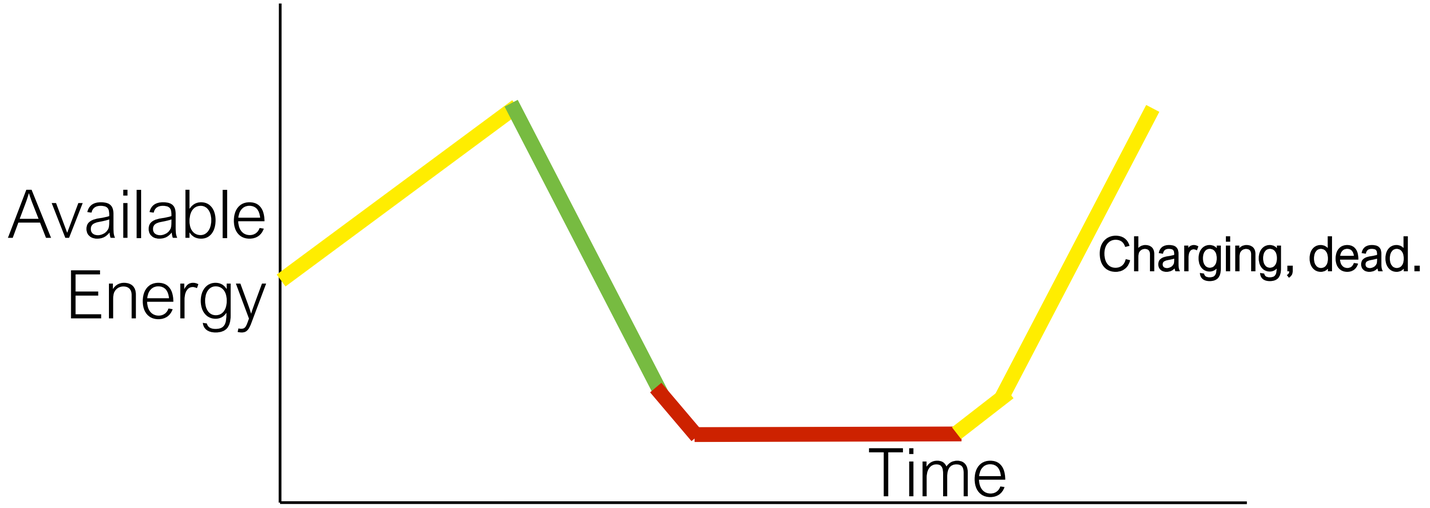
Running on intermittent energy



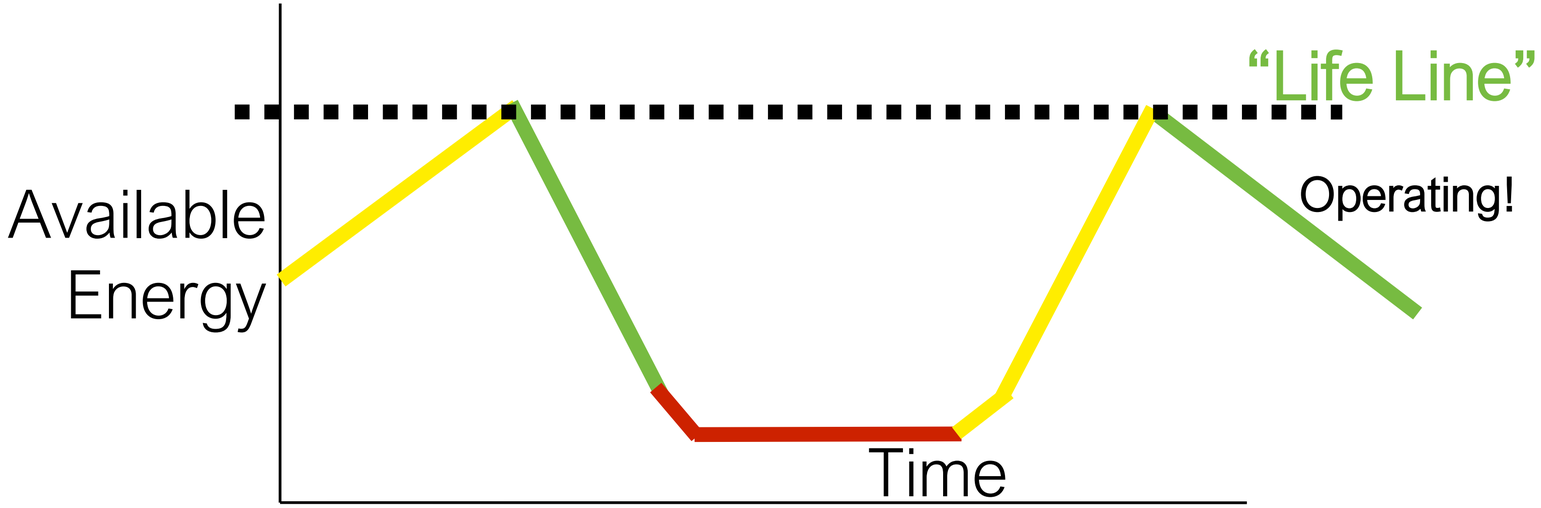
Running on intermittent energy



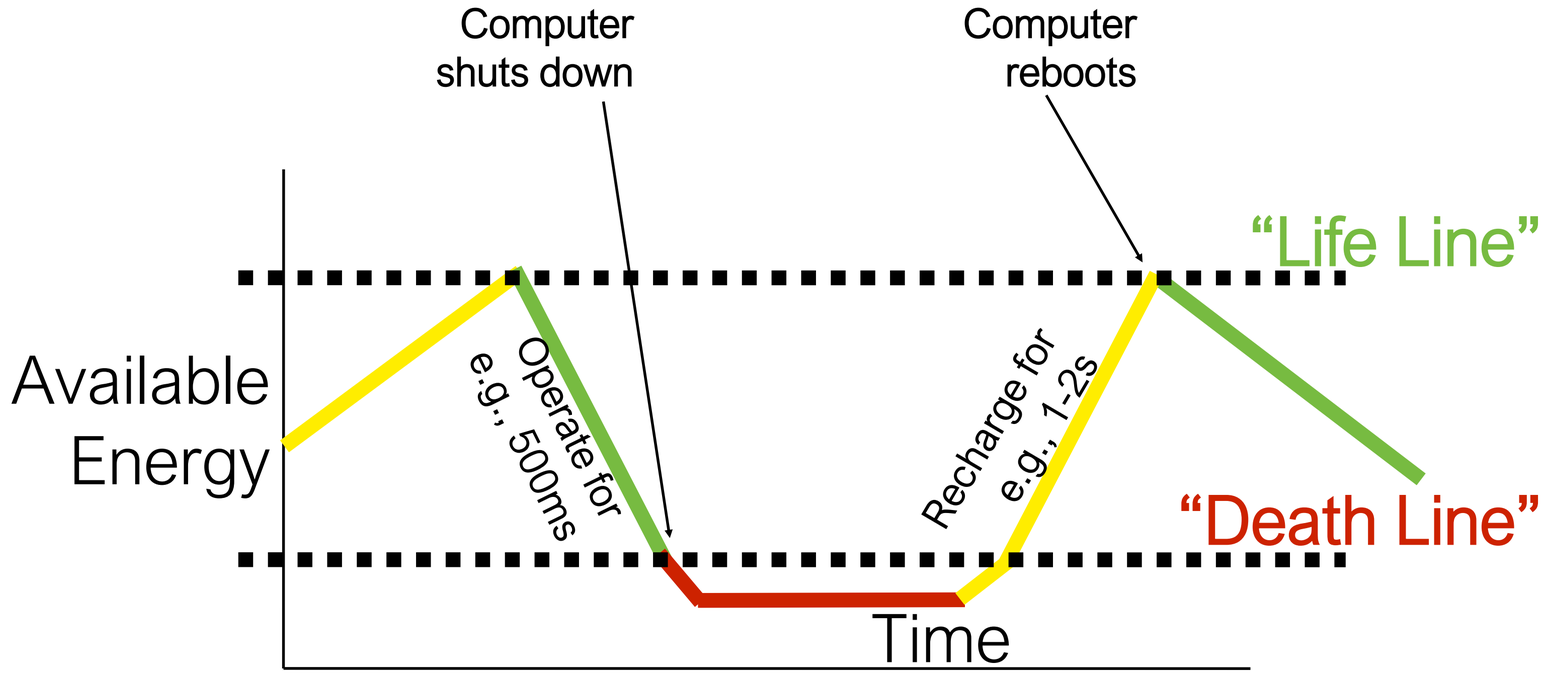
Running on intermittent energy



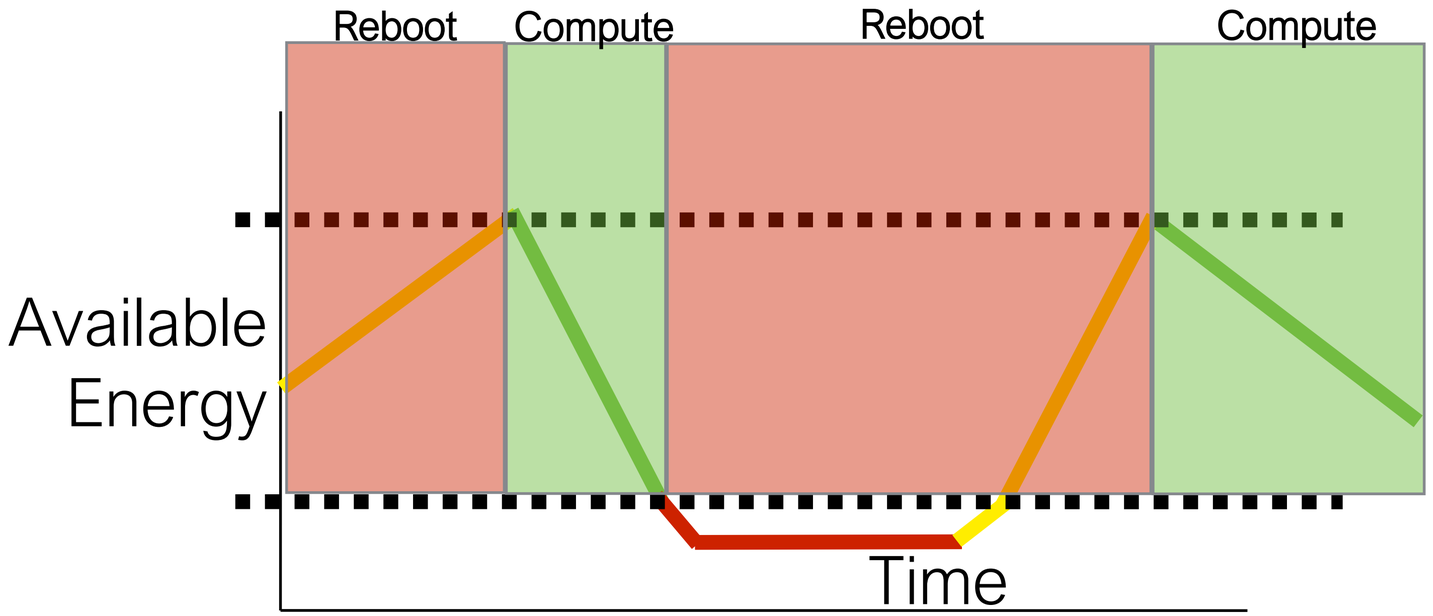
Running on intermittent energy



Running on intermittent energy

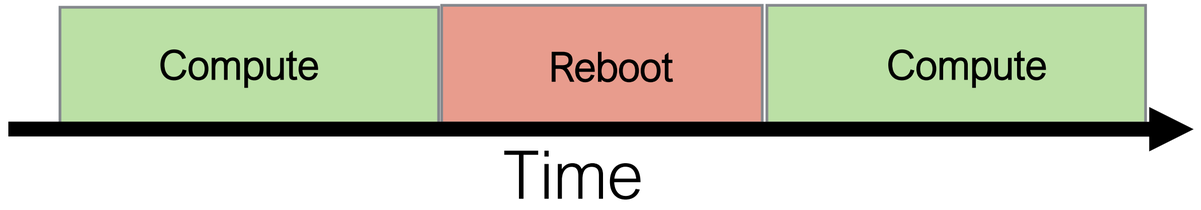


Intermittent Operation on Harvested Energy


Reboot
Compute
Reboot
Compute
Available
Energy
Time

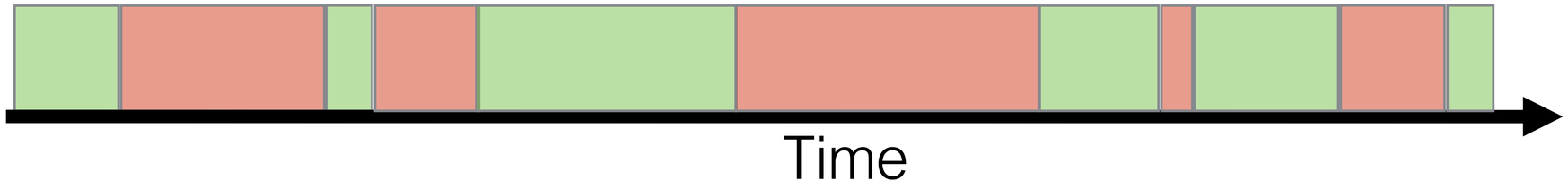
The Intermittent Execution Model

[PLDI '15]



The Intermittent Execution Model

[PLDI '15]

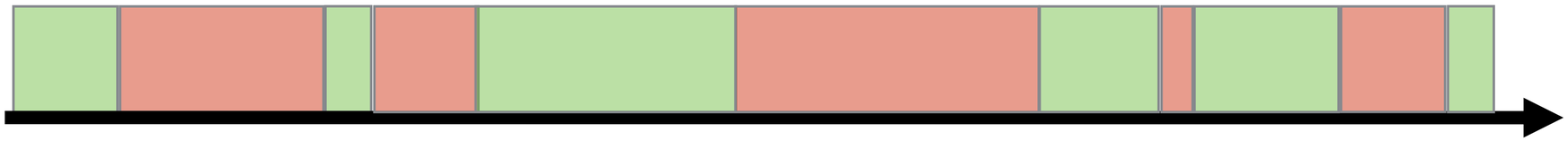


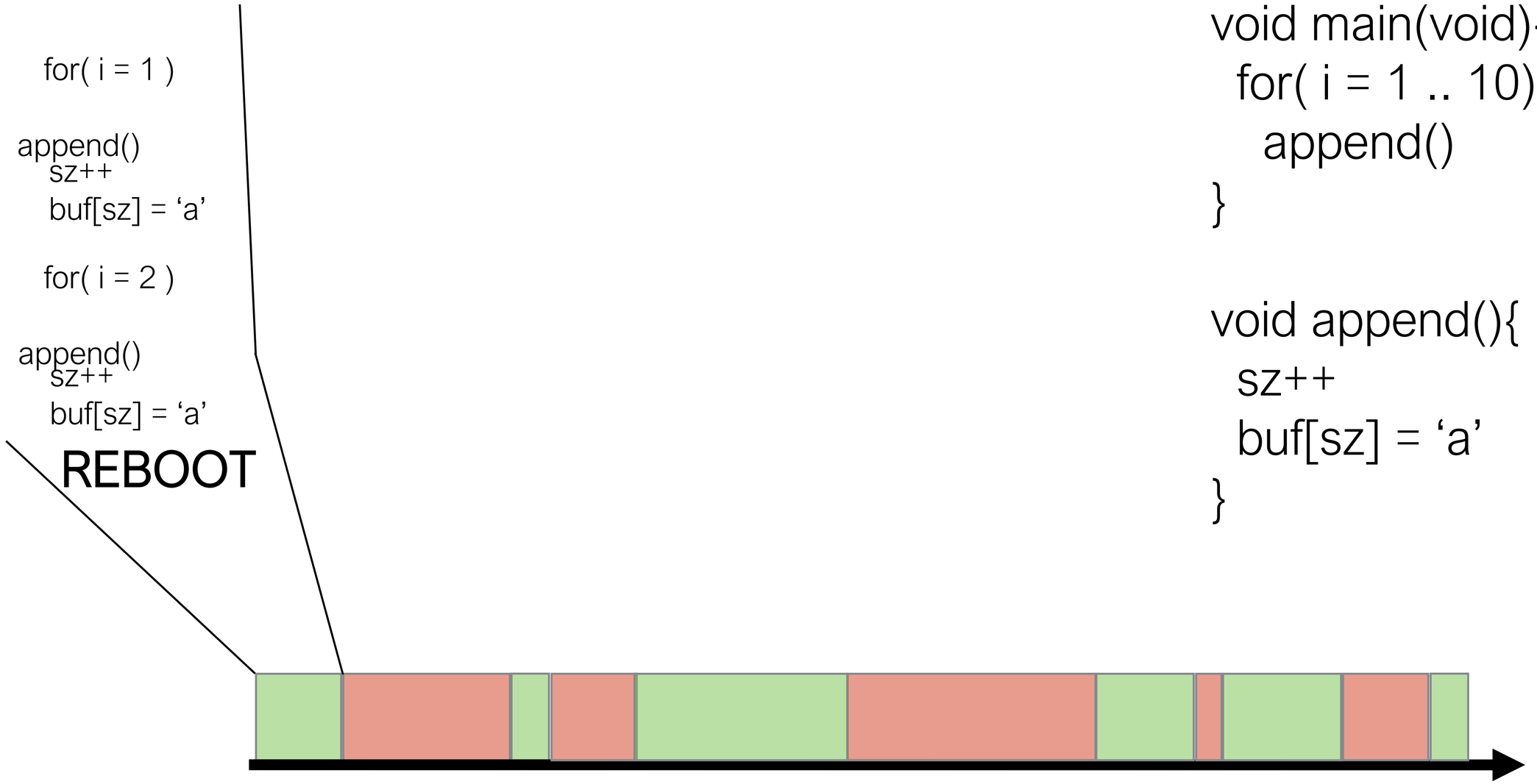
Goal: Run programs that take longer than one green box

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

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

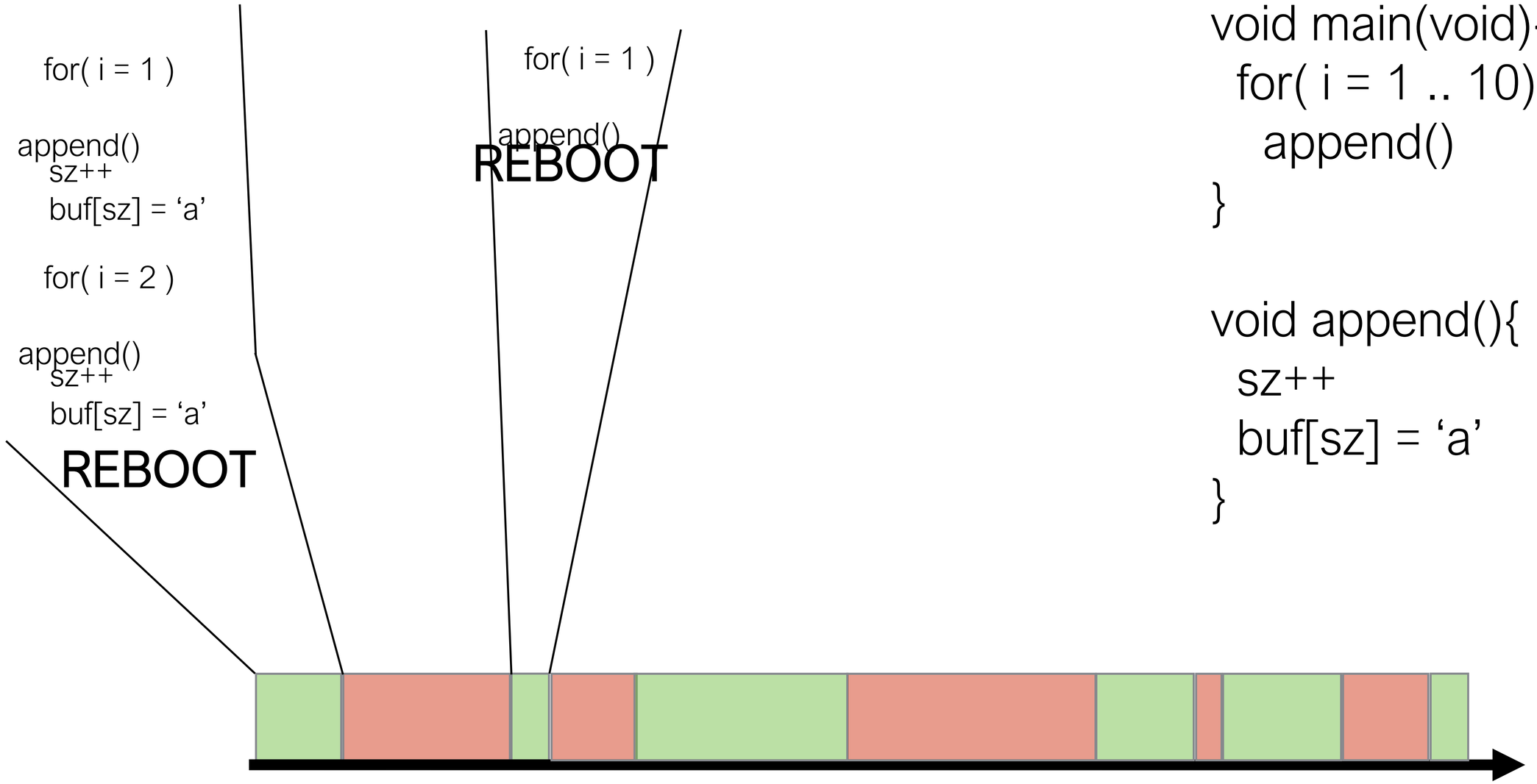
*/*Store 10 letters ‘a’ in the buffer*/*

*/*Store a letter ‘a’ in buf[sz] and increase sz by one*/*

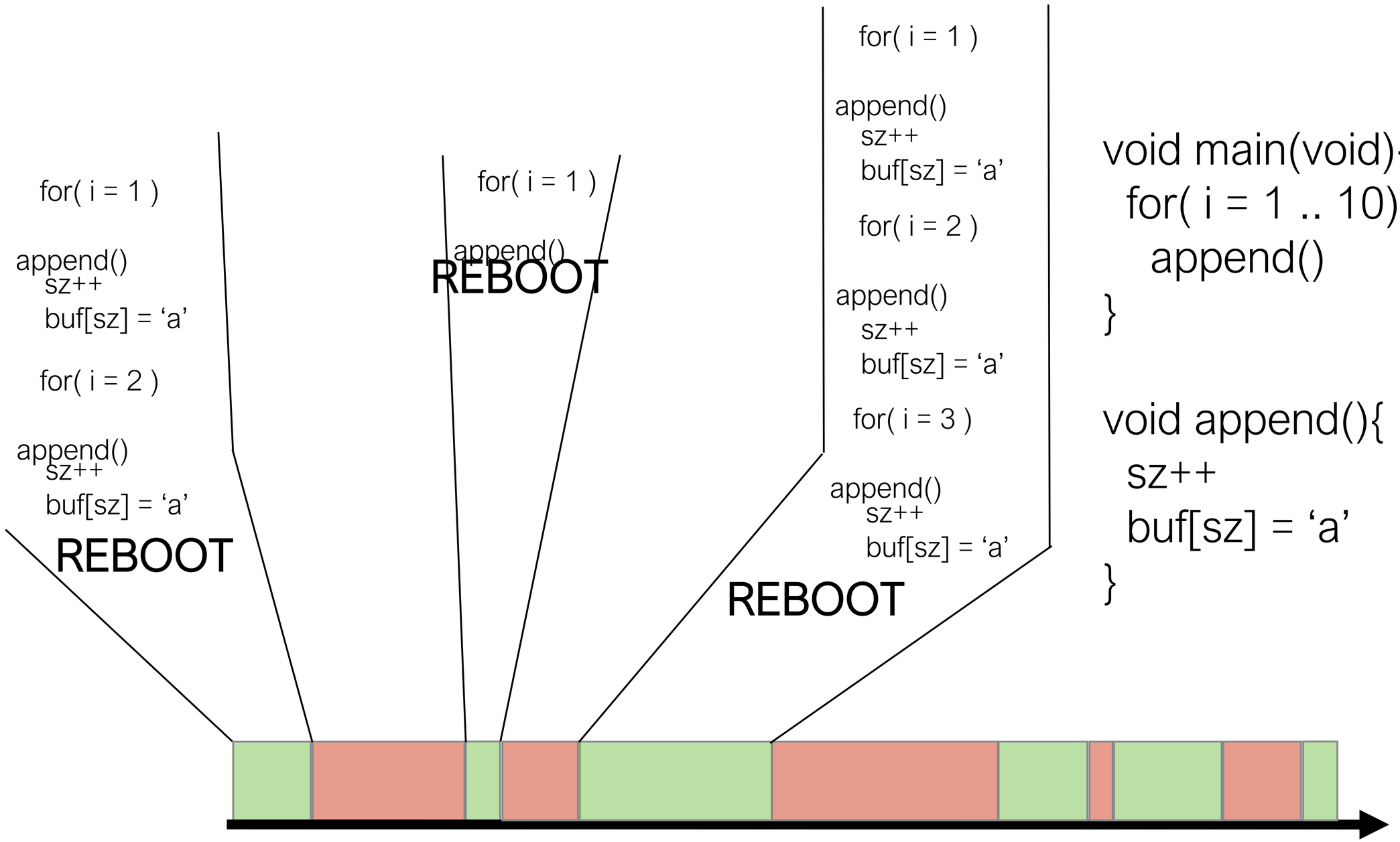





```
void main(void){
  for( i = 1 .. 10)
    append()
}

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

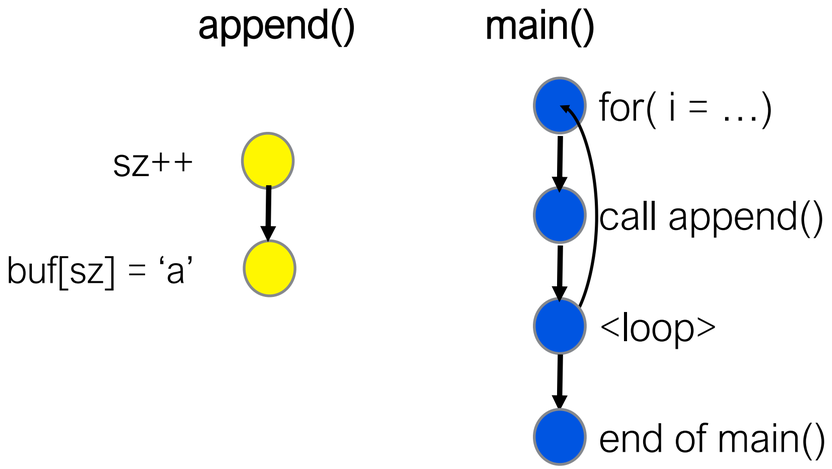



```
void main(void){
  for( i = 1 .. 10)
    append()
}

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



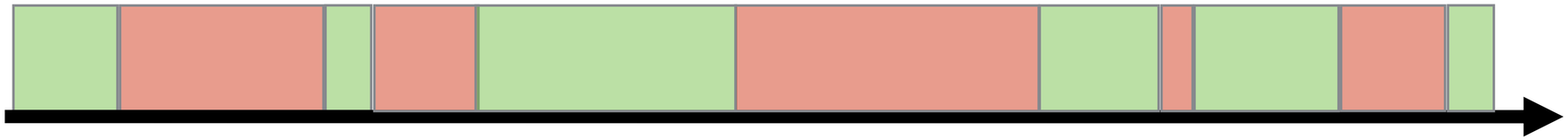

```
void main(void){
  for( i = 1 .. 10)
    append()
}

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

Control-flow Graph



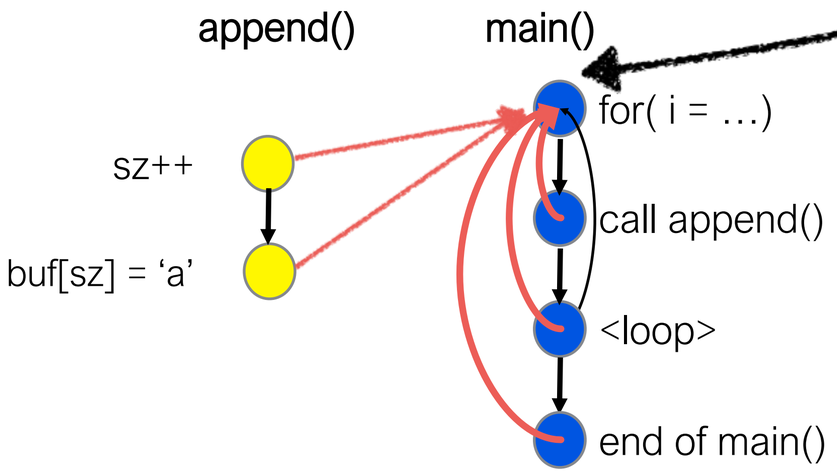

```
void main(void){
  for( i = 1 .. 10)
    append()
}

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

We can model intermittence as a control-flow problem



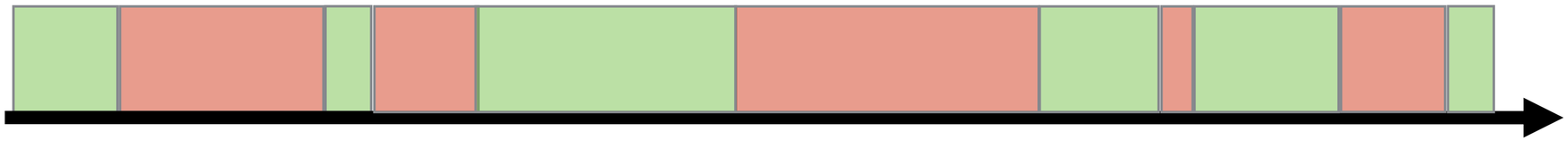
Back in time!

Control-flow Graph

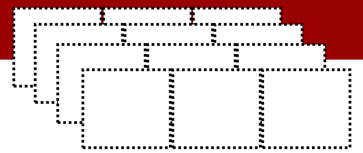


Intermittent Execution Challenge:
Implicit control-flow “back in time” on reboot.

Failure Induces
Implicit Control-flow

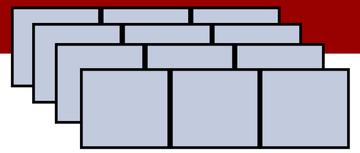


Mixture of Volatile & Non-volatile State



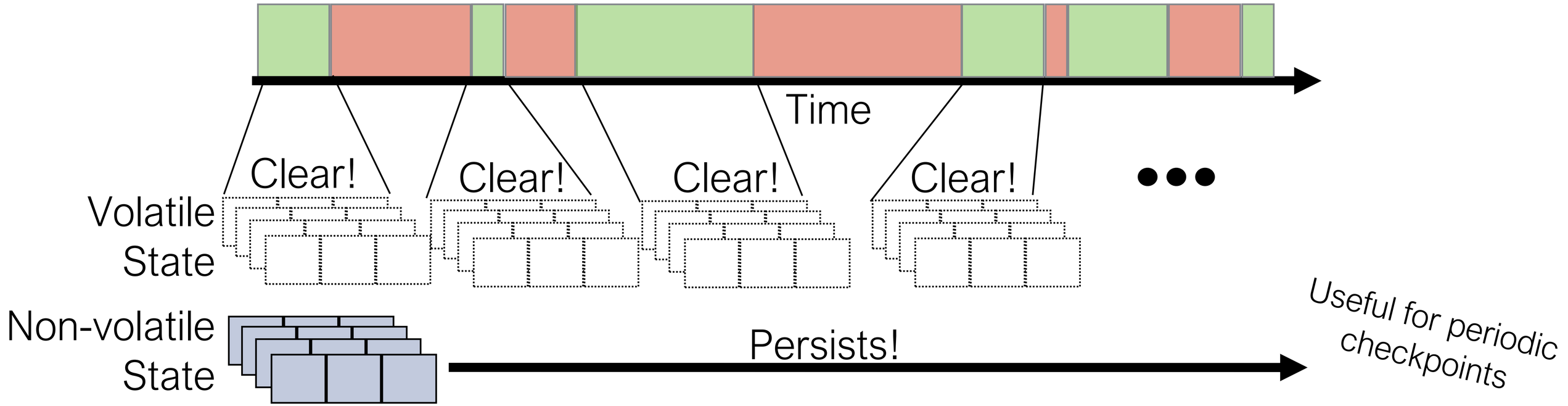
volatile memory (e.g., DRAM, SRAM registers)

Access latencies growing similar w/ new technology

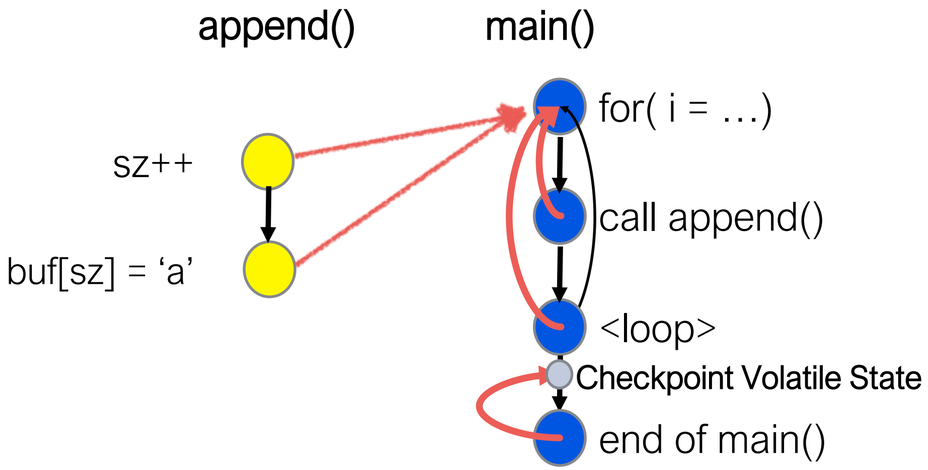


non-volatile memory (e.g., Flash, FRAM)

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

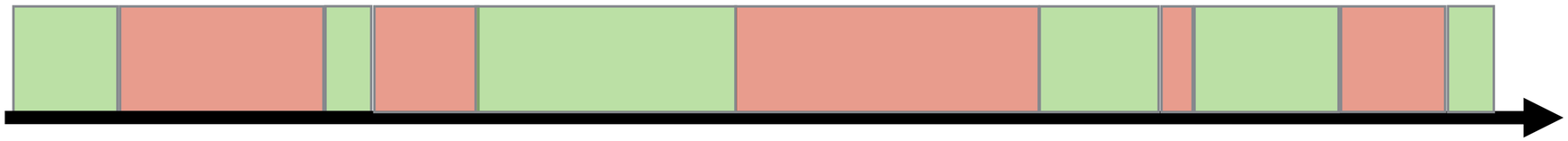


Control-flow Graph



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

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



The Big Idea

for(i = 1)

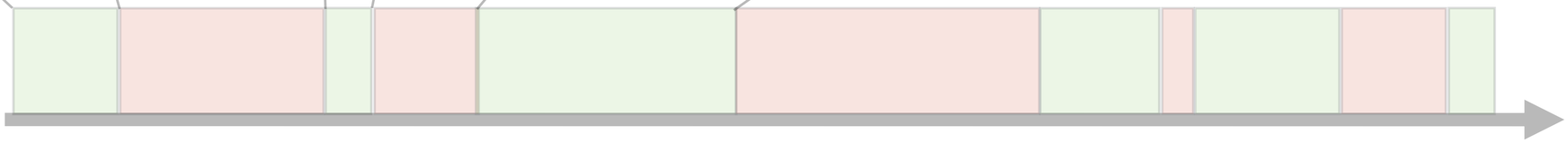
append()
sz++
buf[sz] = ‘a’

REBOOT

for(i = 2)

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

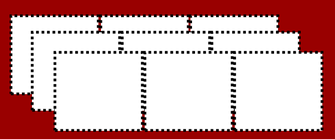
The way we think about software does work with the intermittent execution model



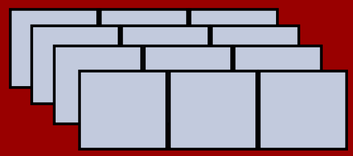
Challenge: *Intermittence Bugs*

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

especially when we combine non-volatile and volatile memory!

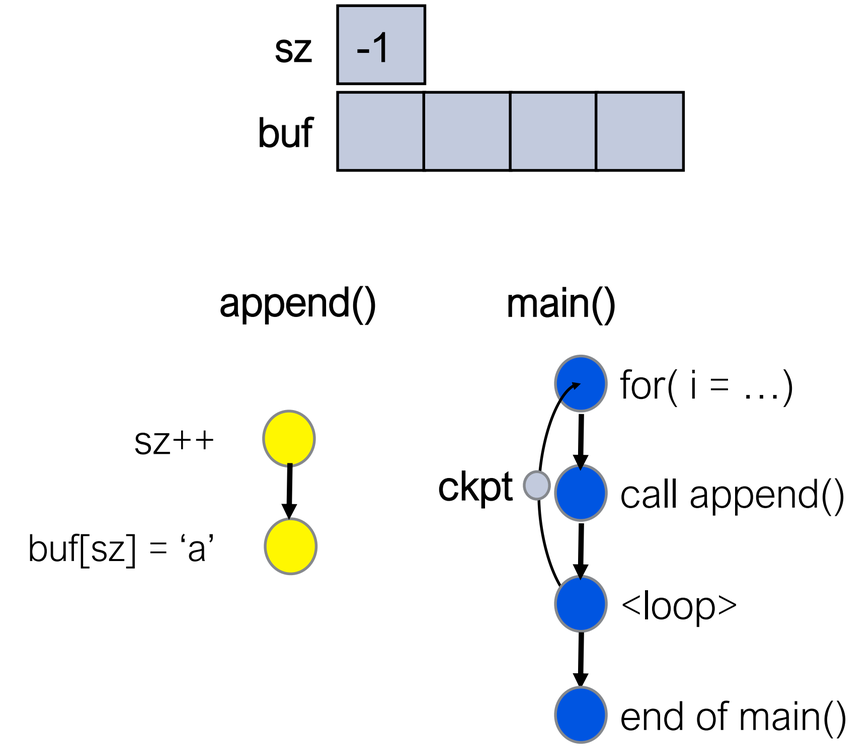


+



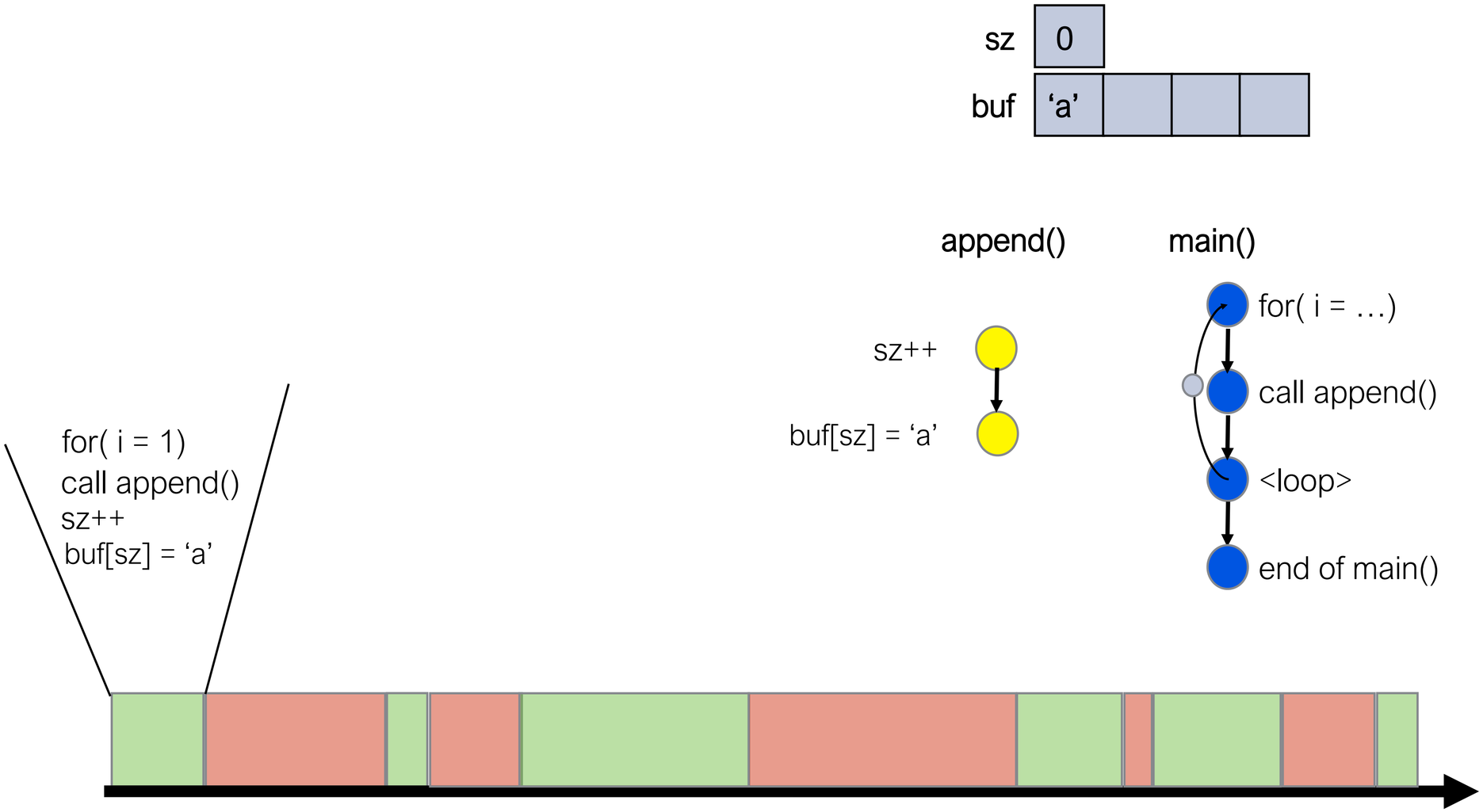
Intermittence Bug: Out-of-thin-air Values

[MSPC '14; PLDI '15]



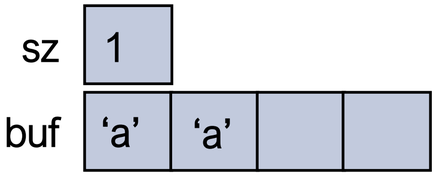
Intermittence Bug: Out-of-thin-air Values

[MSPC '14; PLDI '15]

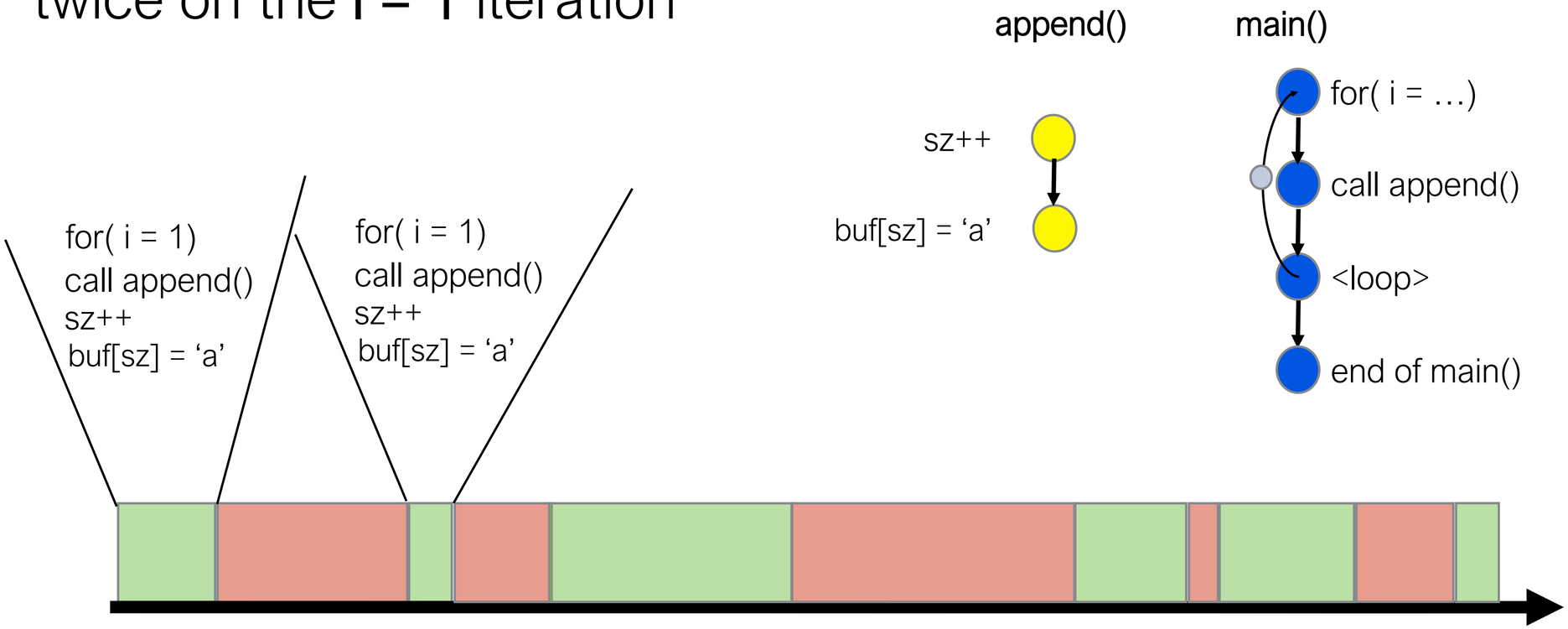


Intermittence Bug: Out-of-thin-air Values

[MSPC '14; PLDI '15]



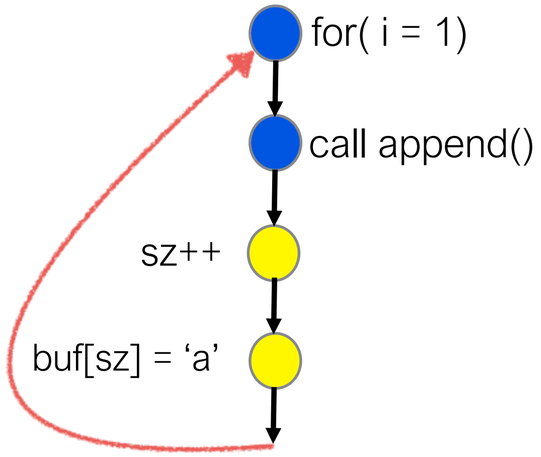
Error: 'a' is appended to **buf** twice on the **i = 1** iteration

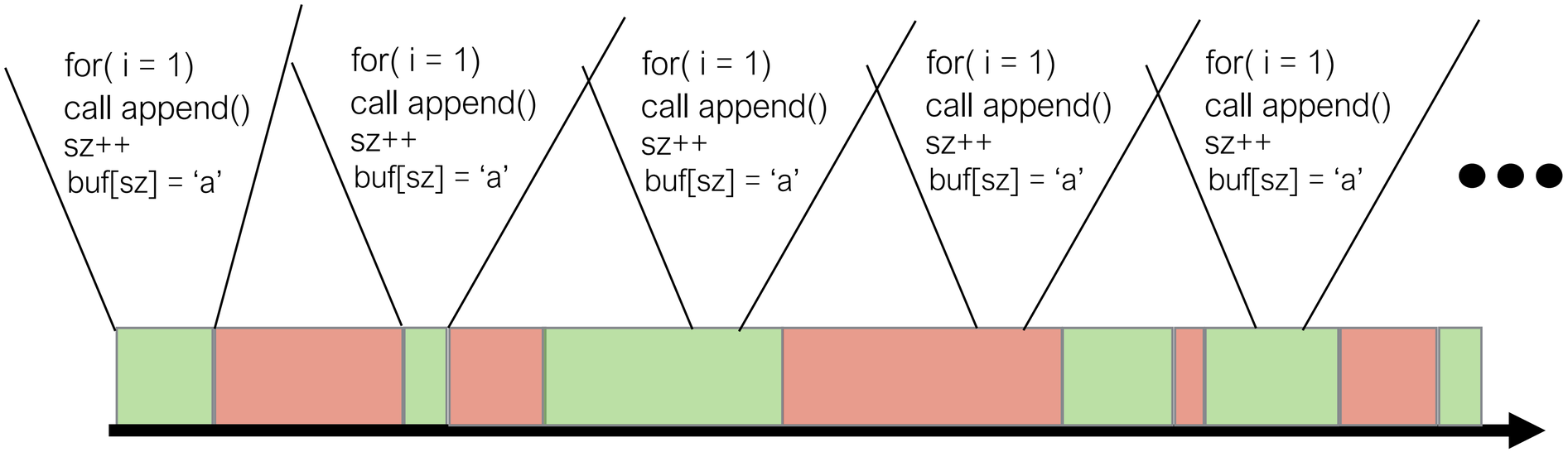


Intermittence Bug: Out-of-thin-air Values

[MSPC '14; PLDI '15]

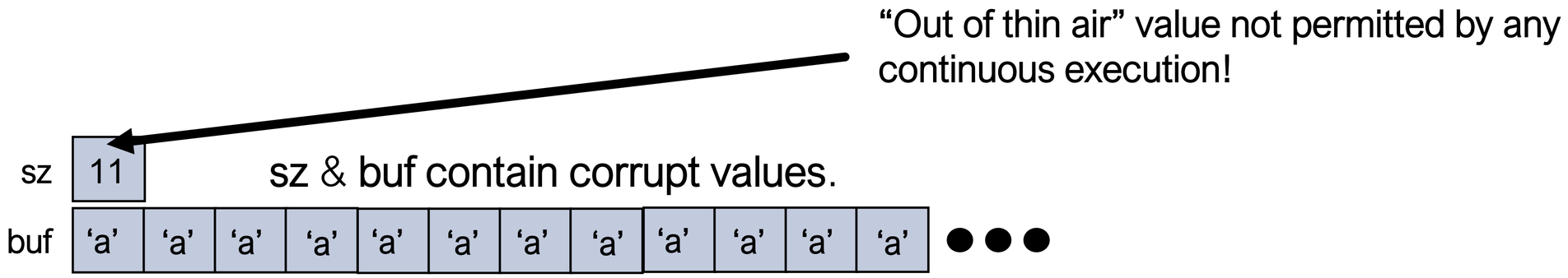
Stuck in an implicit loop for **i = 1**





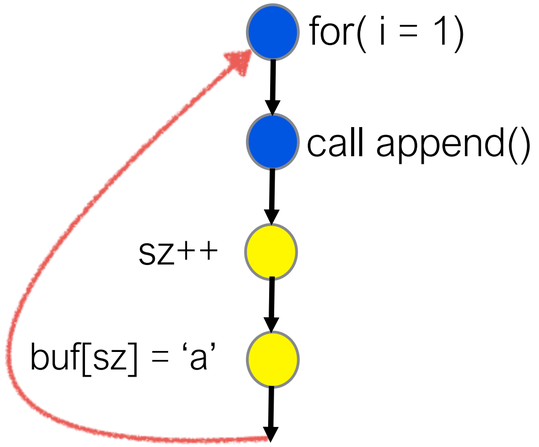
Intermittence Bug: Out-of-thin-air Values

[MSPC '14; PLDI '15]



Memory contents depend on the *availability of harvestable energy* ***and*** *capacitor size*!

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

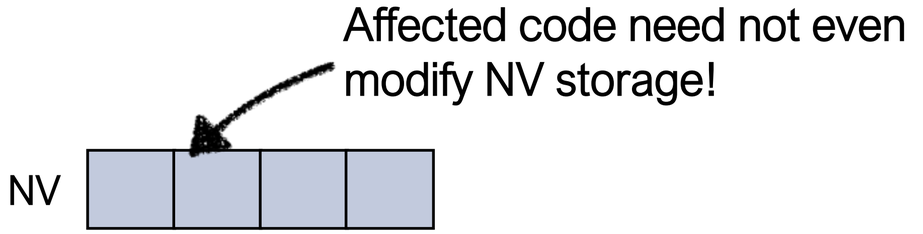


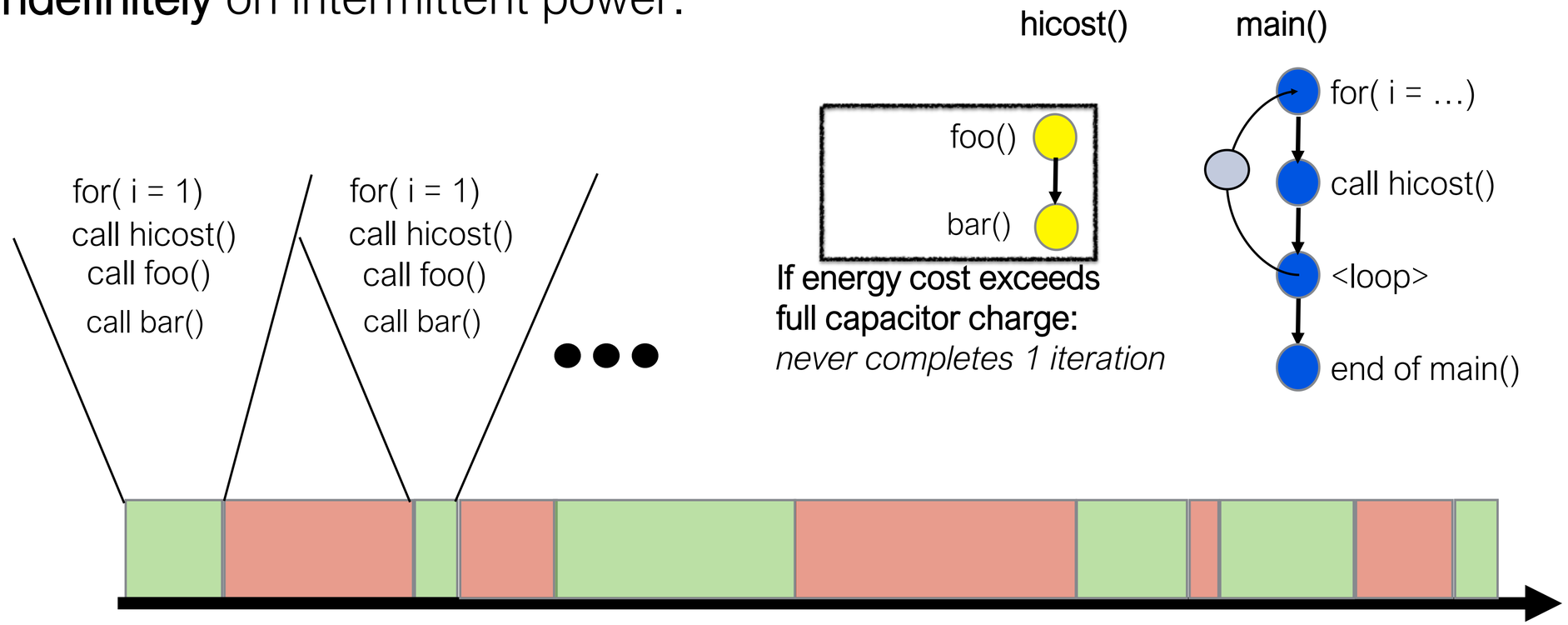
Intermittence Bug: Livelock & Non-termination

[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.



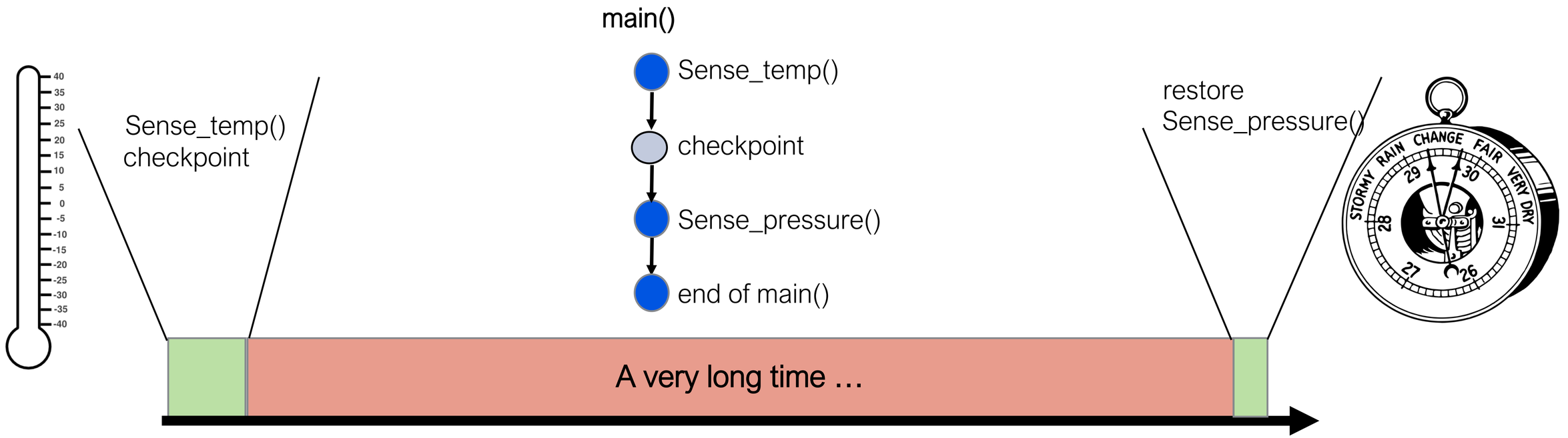


Intermittence Bug: I/O Atomicity

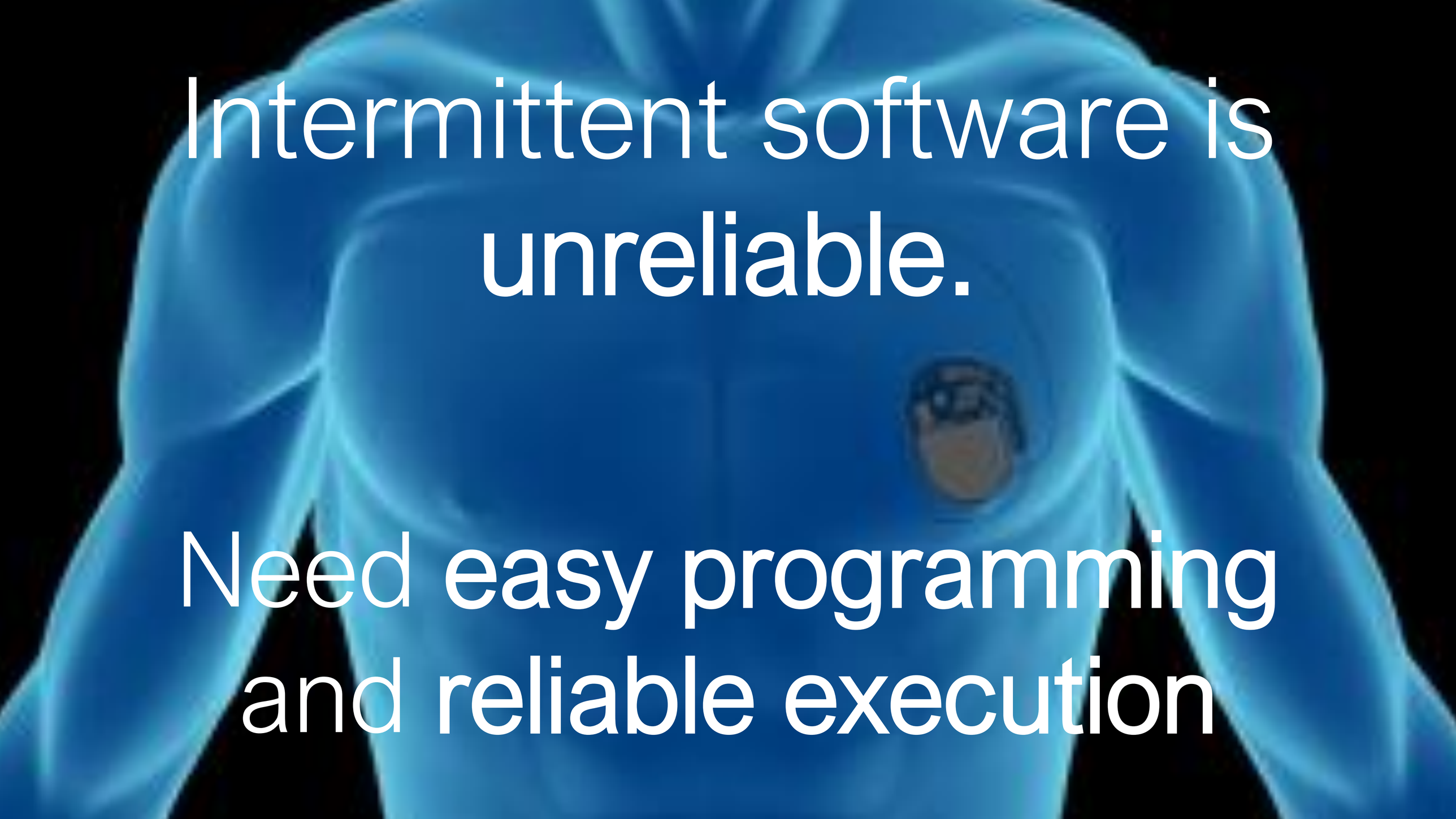
[Colin OOPSLA '16, Maeng OOPSLA '17, Hester et al SenSys '17]

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

No atomicity control w/ automatic checkpoints

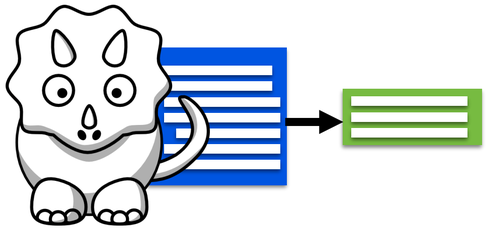


Inconsistent data read at different moments in time


Intermittent software is unreliable.
Need easy programming and reliable execution

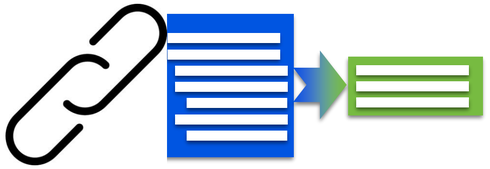
Reliable Intermittent Execution

Task-based Programming Models



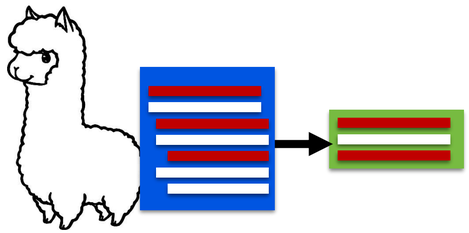
Dino

Arbitrary Task-based Intermittent Programming [PLDI'15]



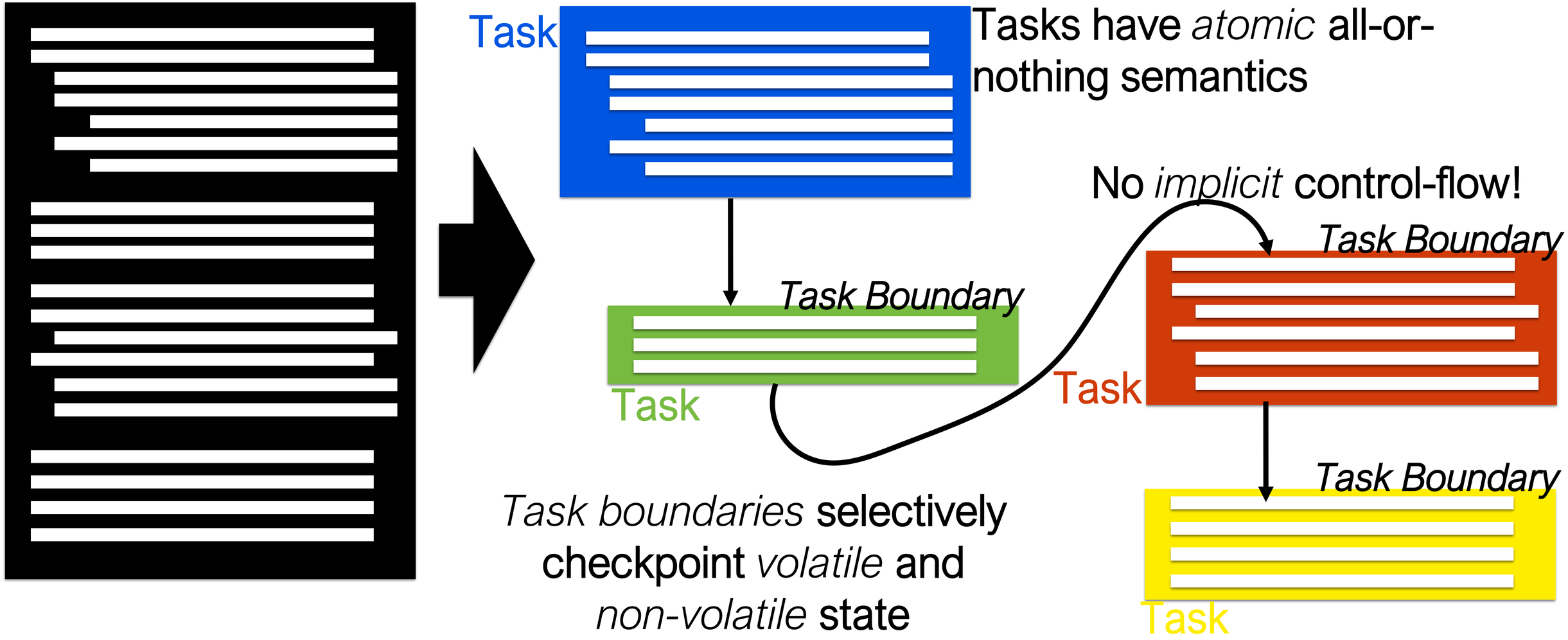
Chain

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



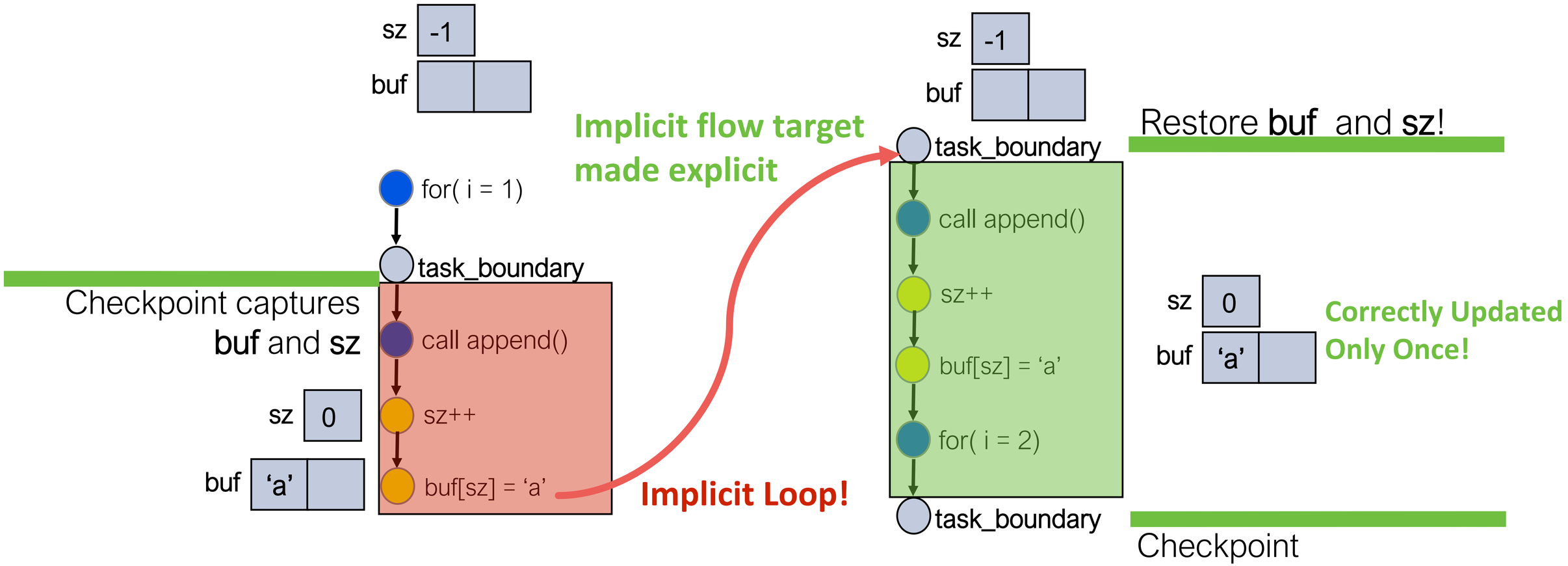
Alpaca

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



Task-atomic Execution in **DINO**

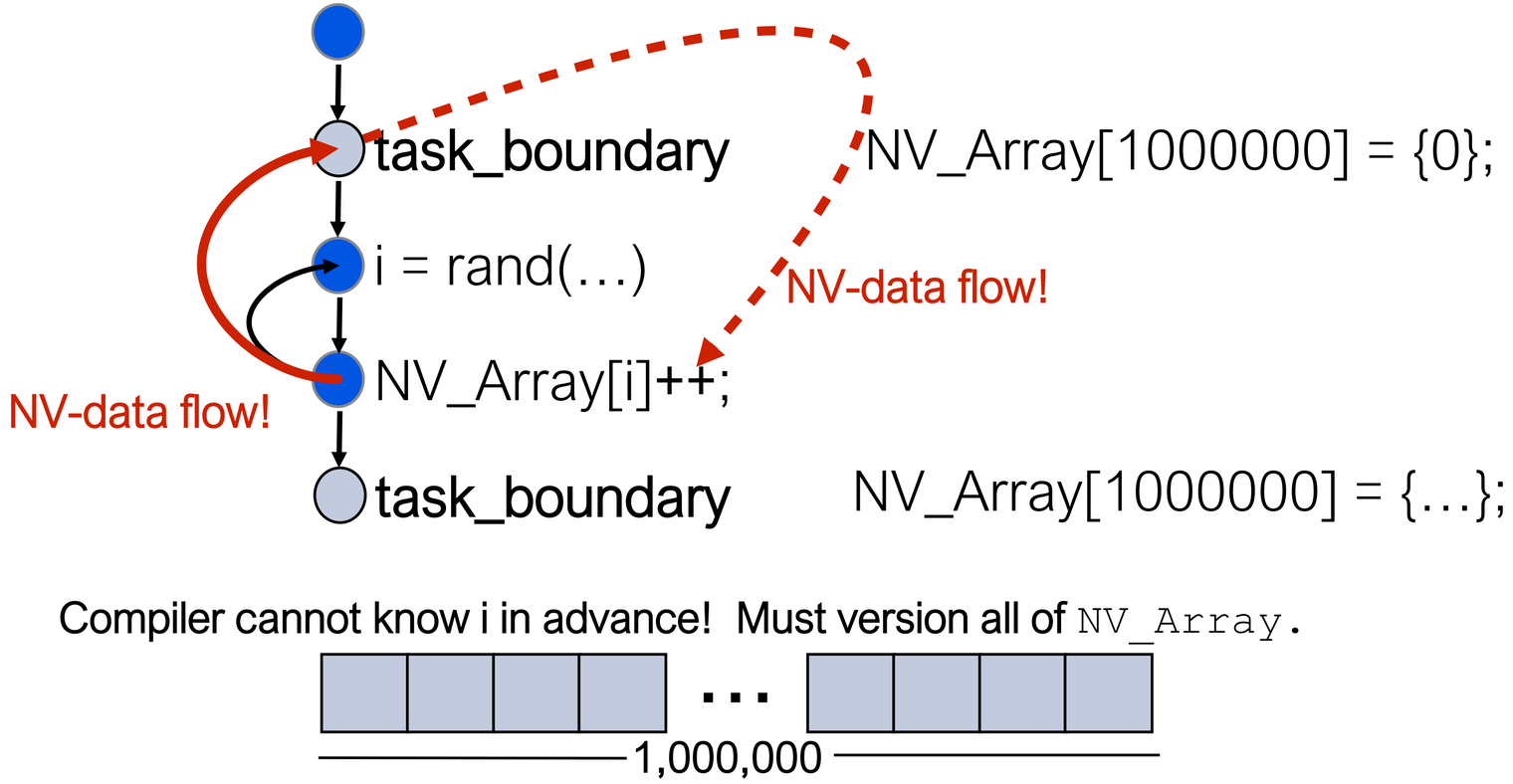
[PLDI '15]



DINO Eliminates OoTA Values

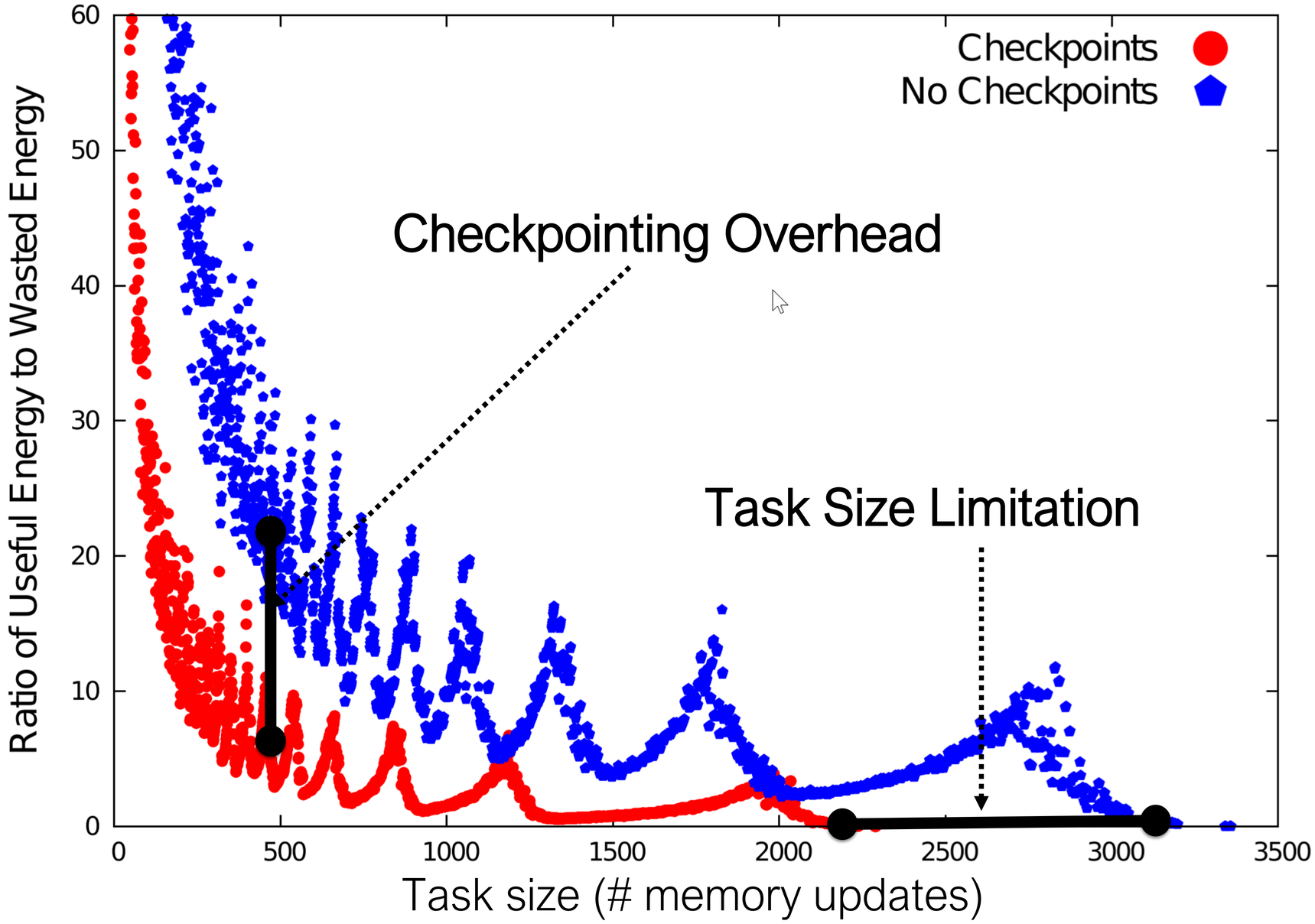
[PLDI ’15]

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

  NV_Array[i]++;

task_boundary
```



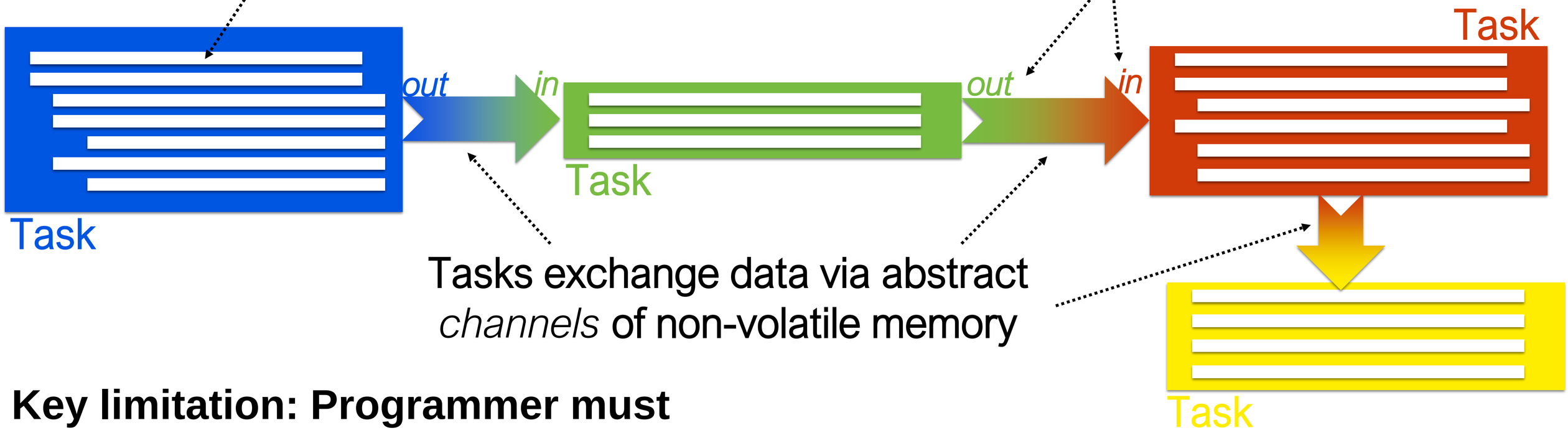
What data to include in checkpoint?

All NV data that **may** be accessed before a reboot, **plus volatile**.


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

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

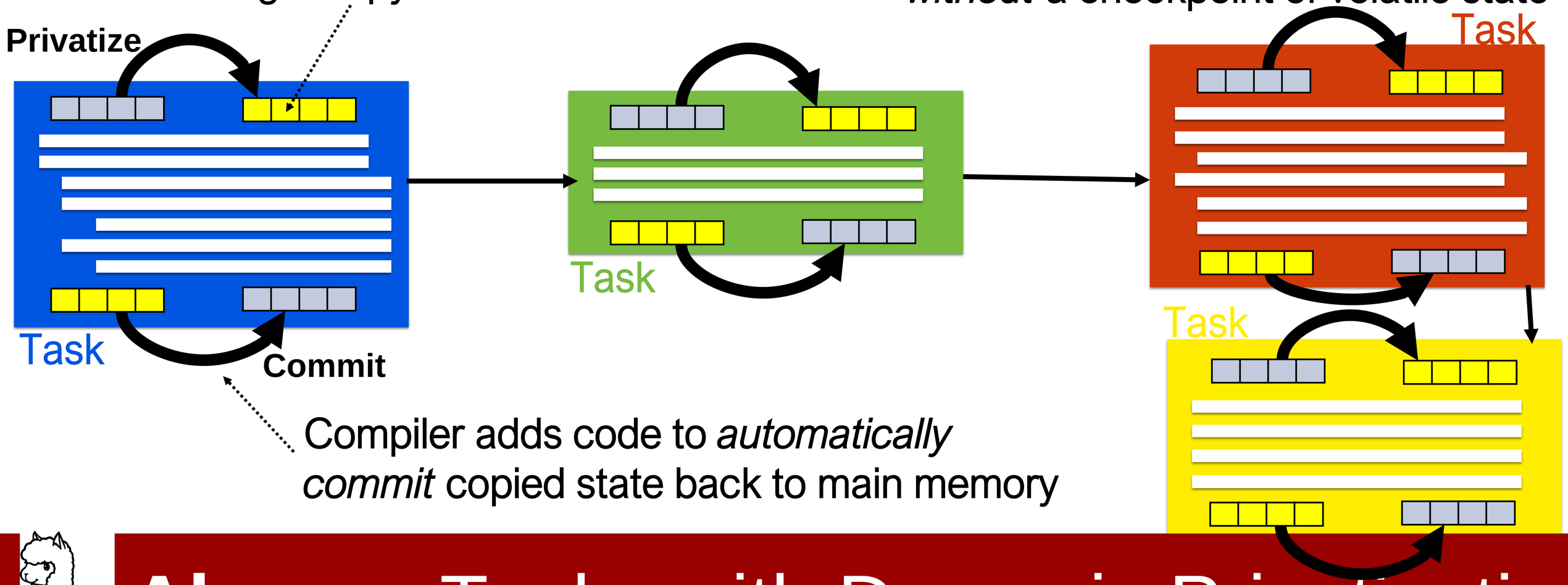
Key limitation: Programmer must learn new programming model

Chain: Static data multi-versioning

[Colin, et al OOPSLA '16]

Compiler *automatically privatizes* data creating a copy safe to use in a task.

Alpaca tasks are restartable *without* a checkpoint of volatile state

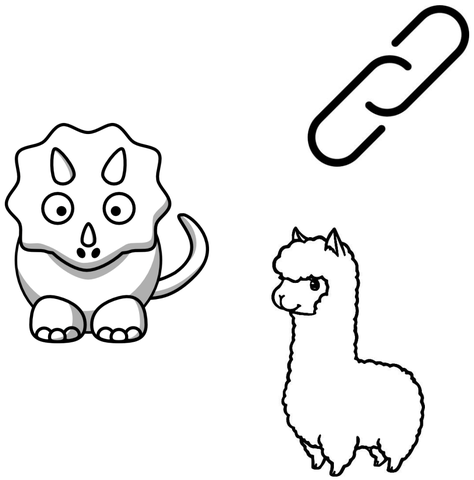


Compiler adds code to *automatically commit* copied state back to main memory

Alpaca: Tasks with Dynamic Privatization

[Maeng, et al OOPSLA '17]

DINO, Chain, Alpaca Programming Model



Task, Channel Definitions

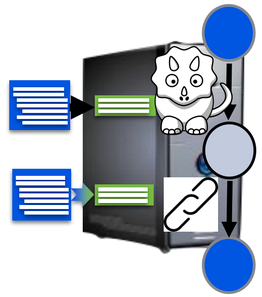
Task-aware Compiler

Task Data-flow Analysis

Link to Task Runtime

Channels, Versioning, Privatization

Runtime Library



DINO: Checkpoint & Recovery

Chain: Task Management

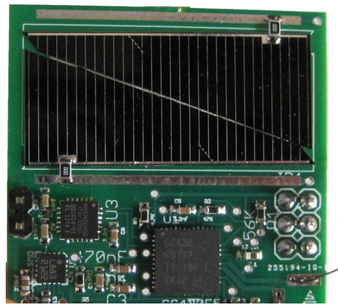
Alpaca: Privatization/Commit

Prototype Implementations

Energy-harvesting Platform Prototypes

WISP5 Energy-harvesting Device

RF Power Setup



Custom PCBs

Applications

Activity Recognition (accelerometer+ML)

MIDI Natural User Interface

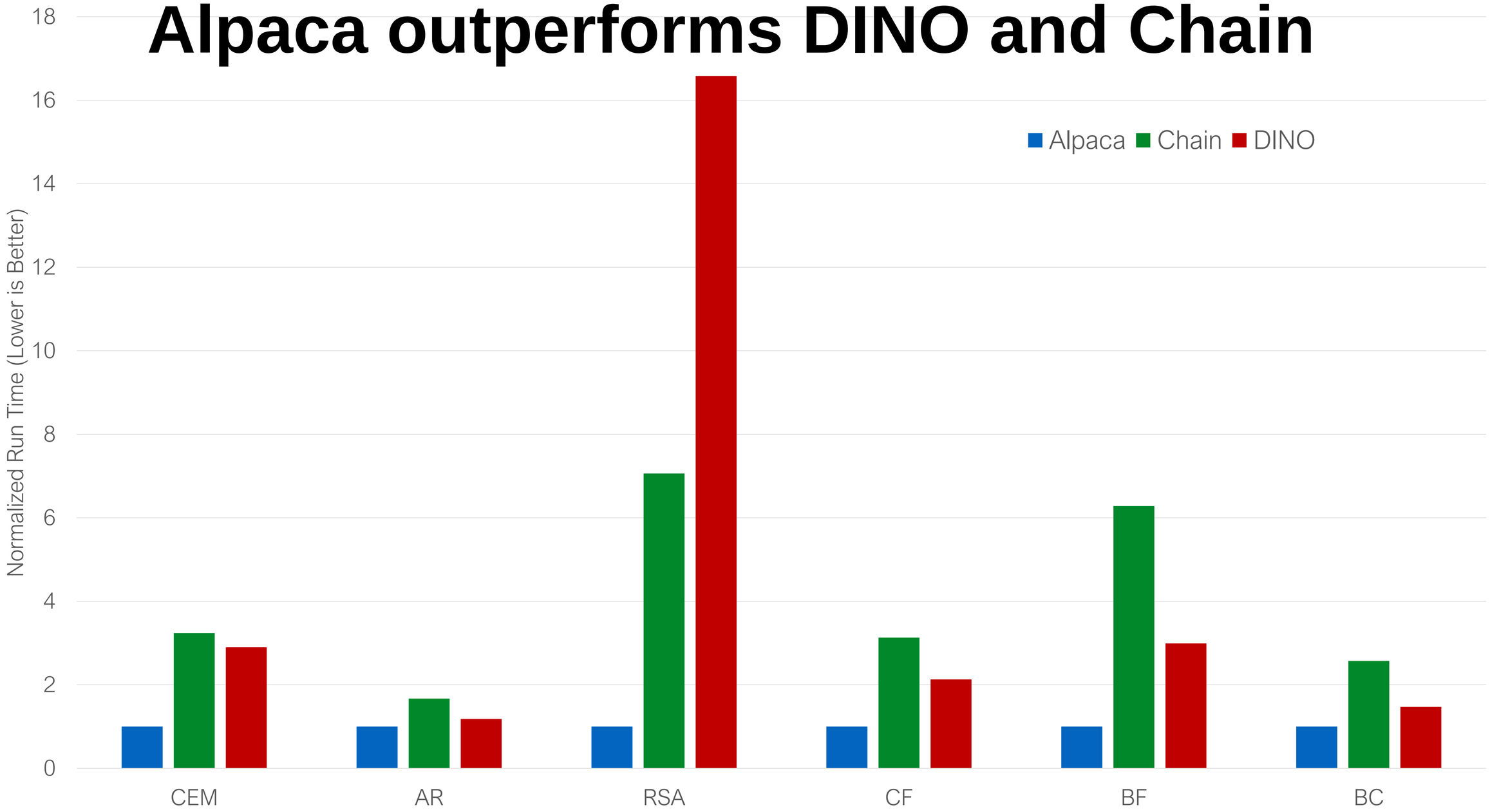
Cold-chain Equipment Monitor

Multi-granular Sensor Log

Key Result:

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

Alpaca outperforms DINO and Chain



Use Alpaca, Chain and DINO for your research!

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

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

Alpaca

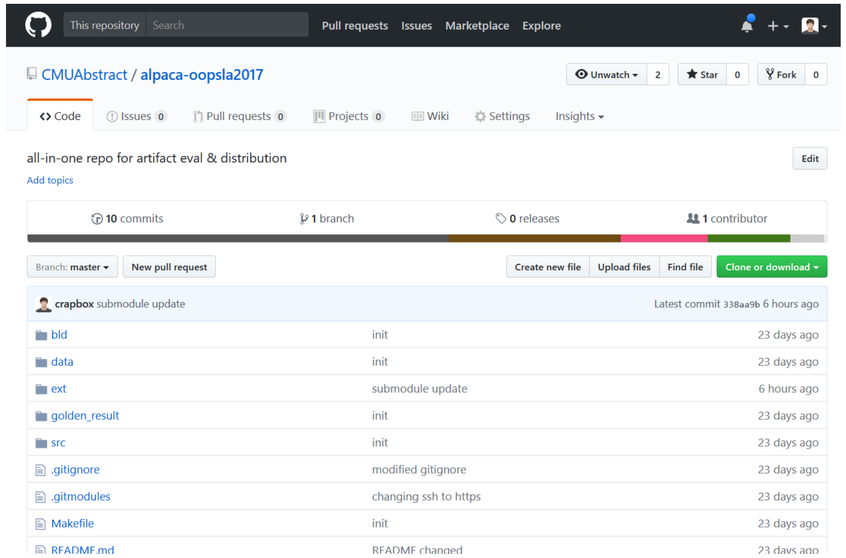
A programming language for writing reliable software for intermittent, energy-harvesting computer systems

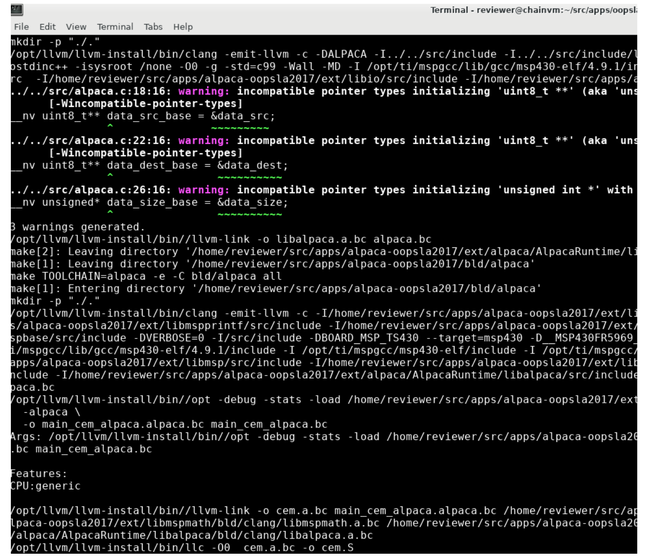
Download VM (10.67GB)

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

Pull source code from Github

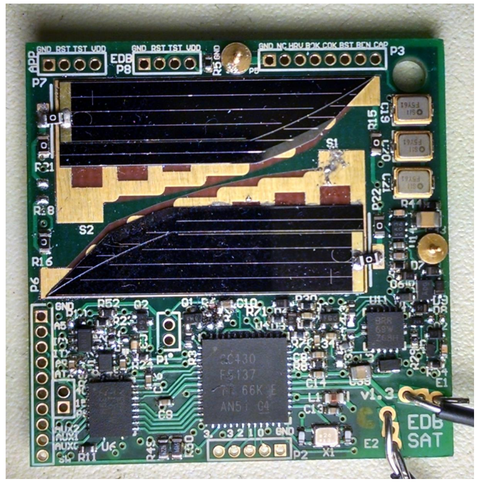
https://github.com/CMUAbstract/alpaca-oopsla2017



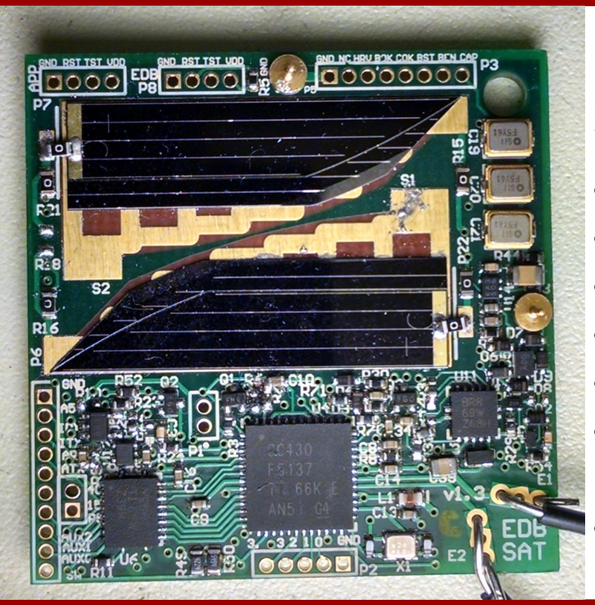


Intermittent Computing Platforms

Hardware, Programming Models, System Software

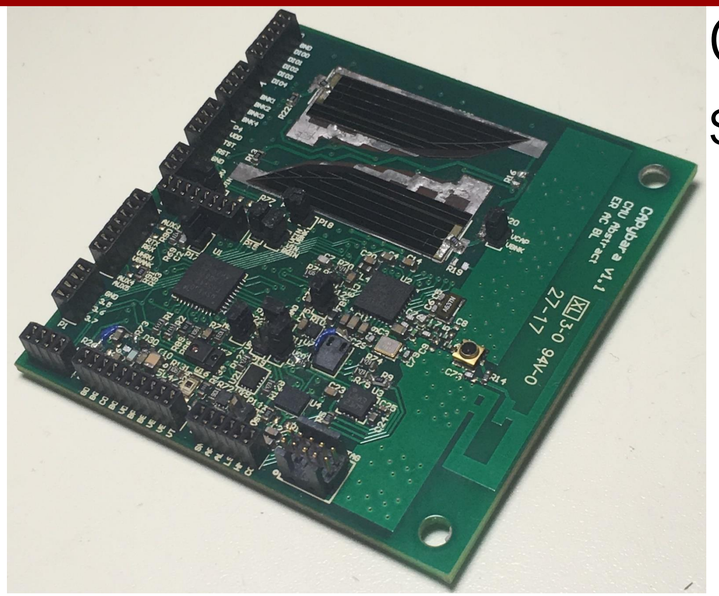


Intermittence-safe HW/SW Systems [ASPLOS 2018]



EDBSat Chip-scale satellite

- 3cmx3cmx4mm, solar power
- Magnetometer + gyro sensors
- Low-power radio (to earth)
- Custom “burst” power system
- Integrated HW/SW diagnostics
- Programmed in intermittence-safe Chain programming lang.
- Launch partner: KickSat/NASA



Capybara multi-sensor platform

- 5cmx5cm, solar/RF power
- 10 sensor array
- Bluetooth Low-energy Radio
- Software-switchable variable-capacity “burst” power system
- Integrated HW/SW diagnostics
- Supports Alpaca & Chain intermittent programming langs.

Key Insight: Co-design PL, OS, Arch, and power system for intermittent operation

The Intermittent Execution Model

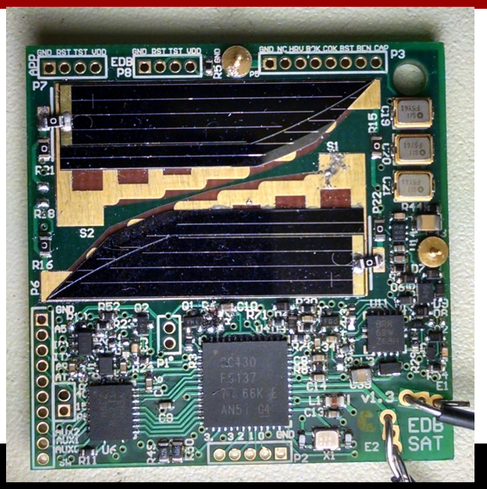
Reasoning About Intermittently-powered Devices

Designing for Intermittence

System Support for Reliable Intermittence

Intermittent Computing Platforms

Hardware, Programming Models, System Software



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)