Mementos

System Support for Long-Running Computations on RFID-Scale Devices

Benjamin Ransford*, UMass Amherst
Jacob Sorber, Dartmouth College
Kevin Fu, UMass Amherst

http://spqr.cs.umass.edu/mementos

ASPLOS XVI — March 8, 2011

CNS-0627529, CNS-0845874, CNS-0923313, Grad Res. Fel.

Ubiquitous Computing

*"... the most powerful things are those that are effectively **invisible** in use."*

— Mark Weiser
(PARC, 1988)

Problem:

Batteryless **invisible** computer ⇒ constant reboots

Baby Steps Toward Ubicomp

1. Take Emerging Platform

Baby Steps Toward Ubicomp

1. Take Emerging Platform

2. Add Robustness Mechanism

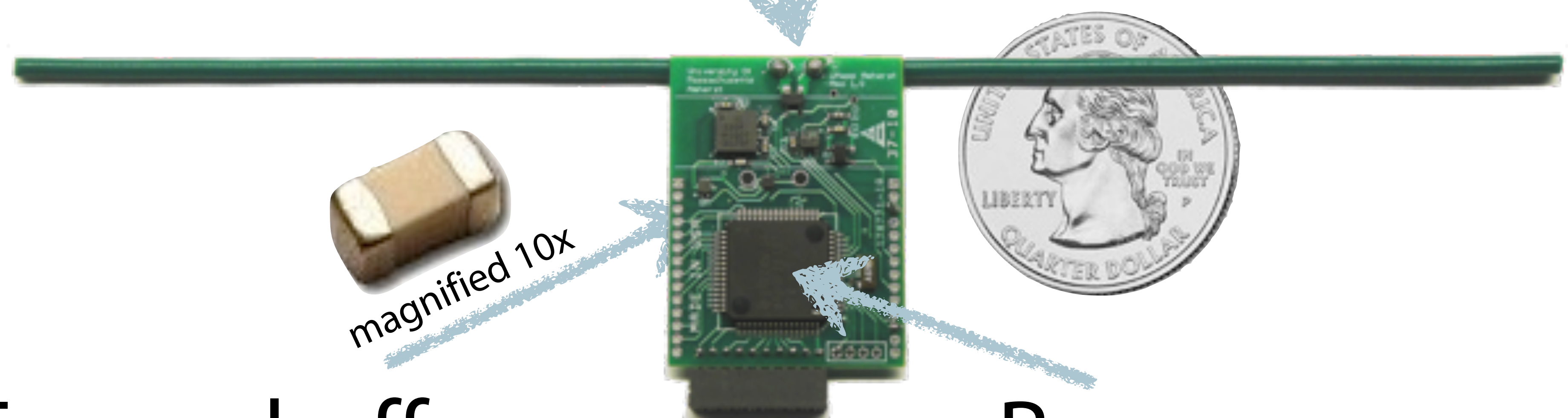
Baby Steps Toward Ubicomp

1. Take Emerging Platform
2. Add Robustness Mechanism
3. Provide Simulation Tools

RFID-Scale Devices

1. Emerging Platform

Radio (RF) harvester



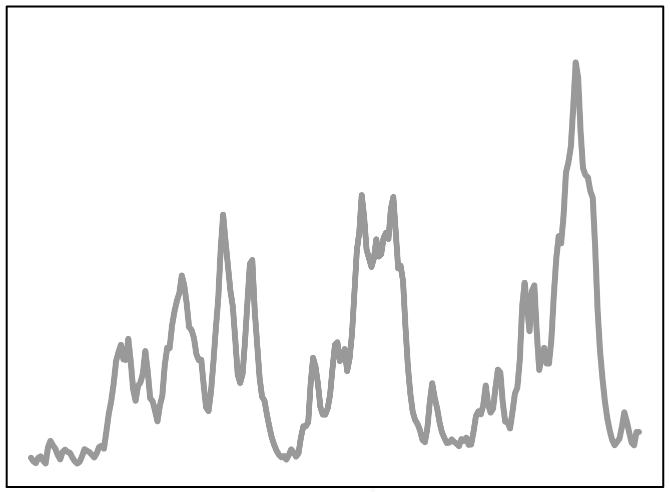
Energy buffer (capacitor)

Reprogrammable microcontroller (~1 MHz) w/ on-chip flash

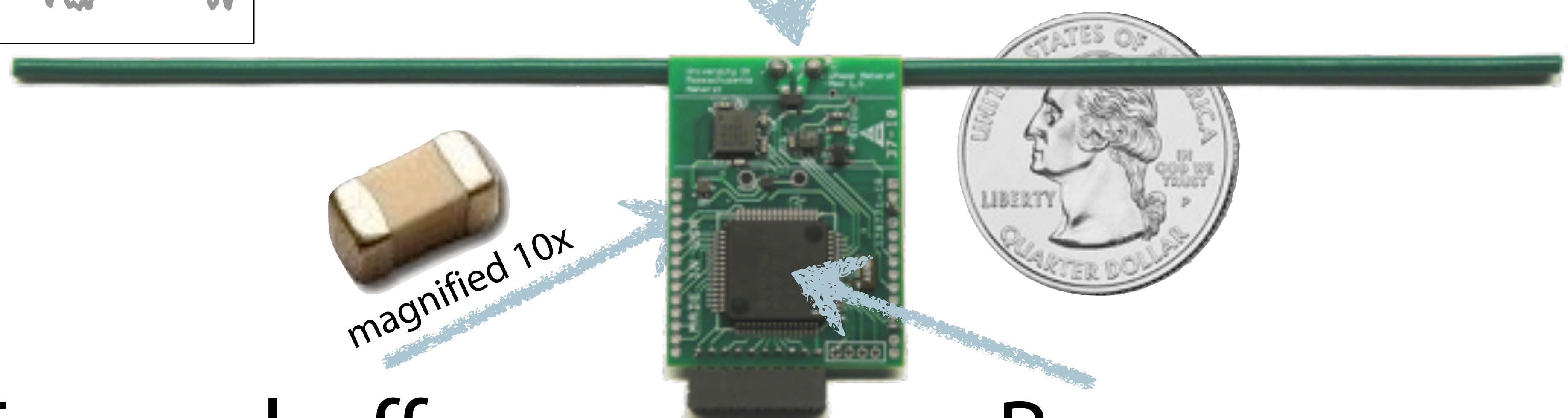
Moo WISP: Hong Zhang

RFID-Scale Devices

1. Emerging Platform



Radio (RF) harvester



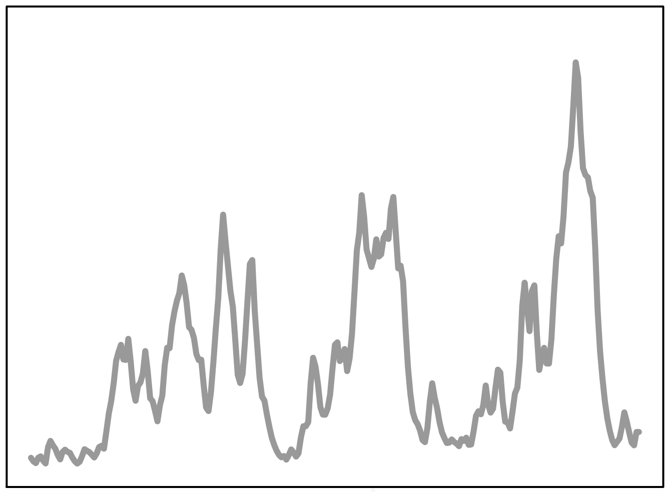
Energy buffer (capacitor)

Reprogrammable microcontroller (~1 MHz) w/ on-chip flash

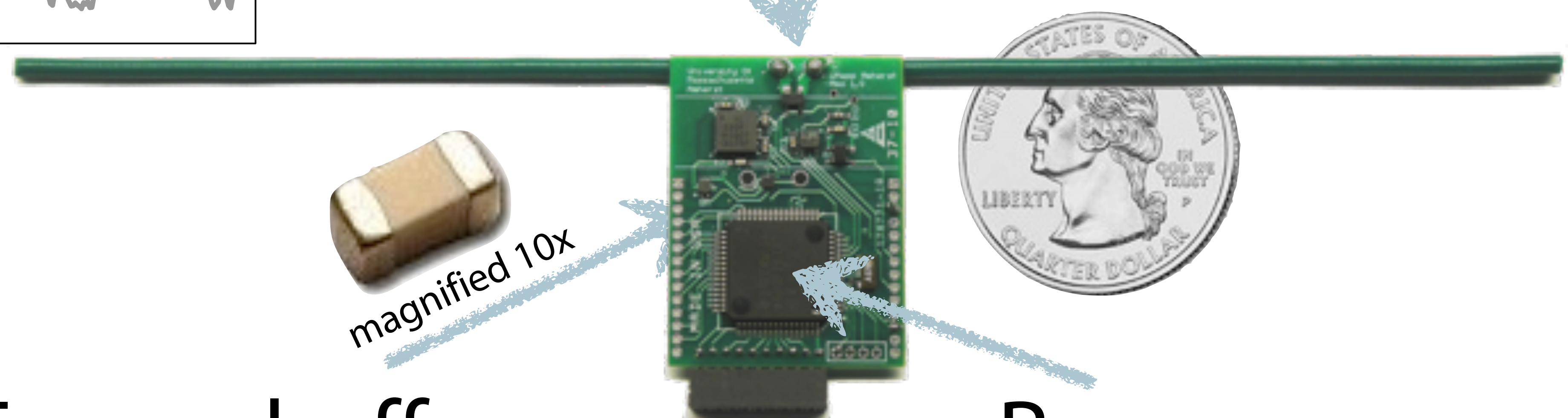
Moo WISP: Hong Zhang

RFID-Scale Devices

1. Emerging Platform



Radio (RF) harvester



Energy buffer (capacitor)

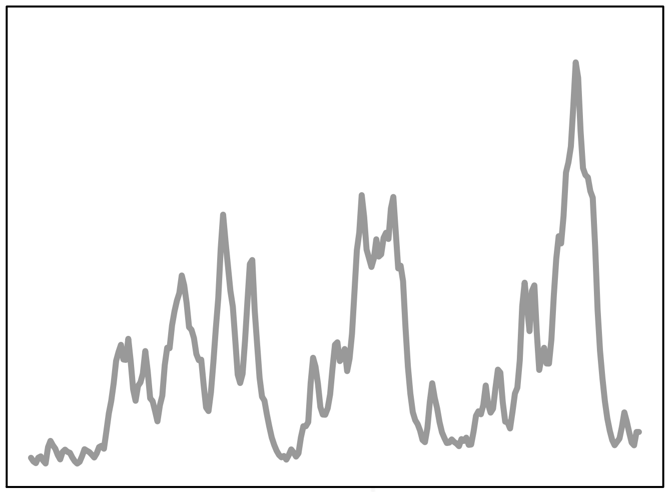
Reprogrammable microcontroller (~1 MHz) w/ on-chip flash

Fills quickly, low capacity

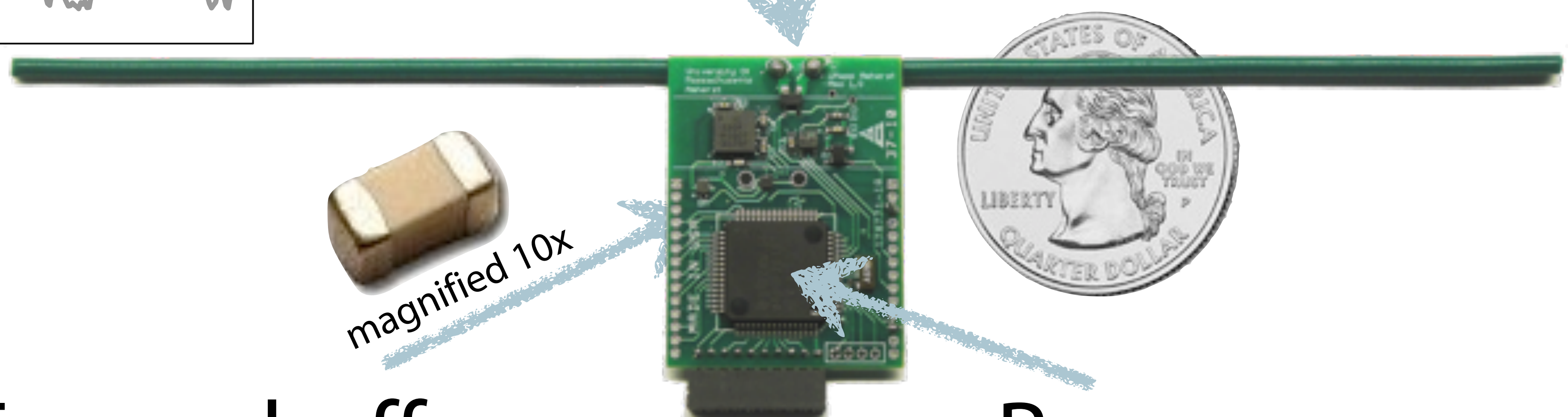
Moo WISP: Hong Zhang

RFID-Scale Devices

1. Emerging Platform



Radio (RF) harvester



magnified 10x

Energy buffer (capacitor)

Reprogrammable microcontroller (~1 MHz) w/ on-chip flash

Fills quickly, low capacity

Frequent reboots

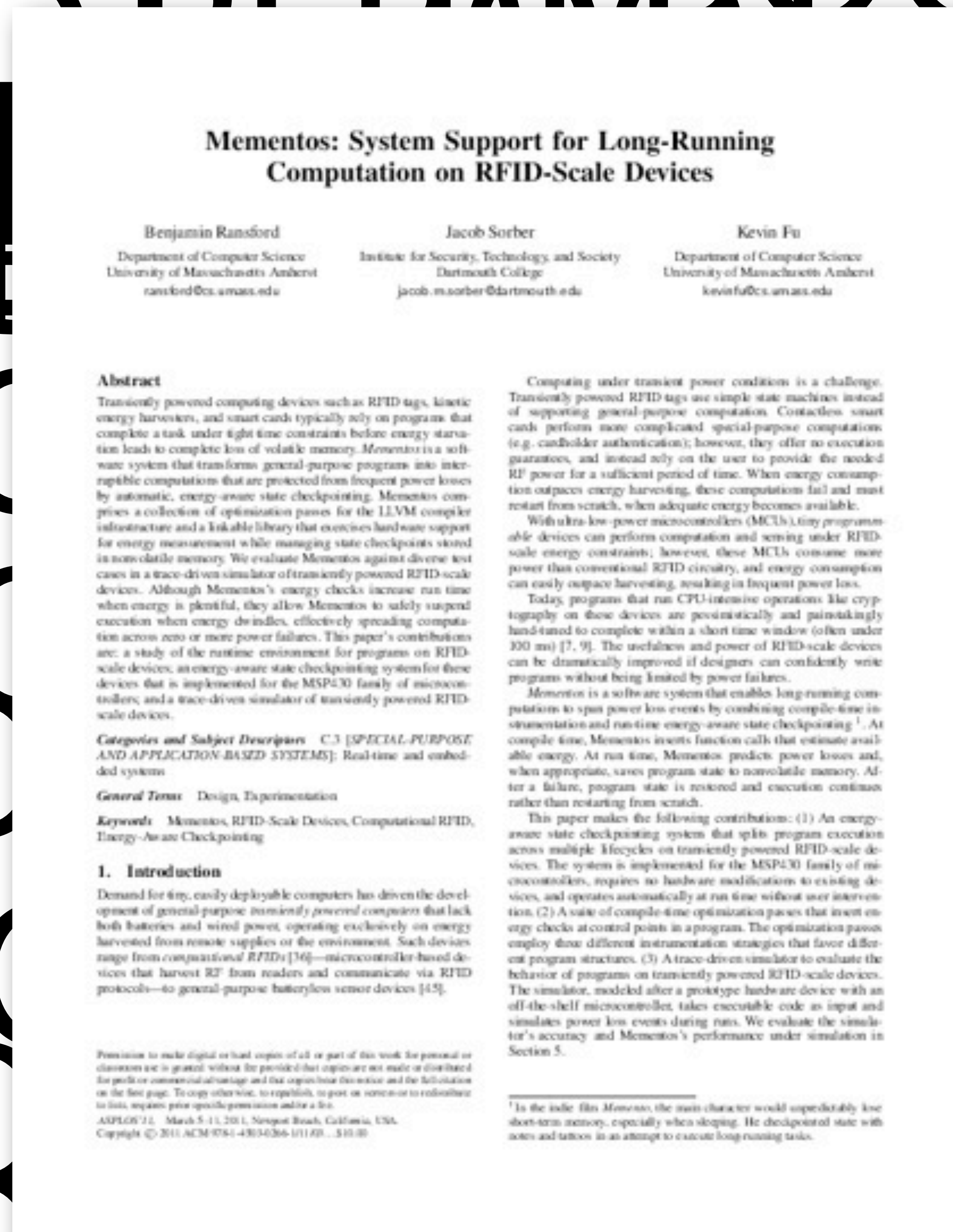
Moo WISP: Hong Zhang

ball: clipart.pierceinternet.com

REBOOT
REBOOT REBOOT
REBOOT REBOOT REBOOT
REBOOT REBOOT
REBOOT REBOOT
REBOOT REBOOT
REBOOT REBOOT REBOOT
REBOOT
REBOOT REBOOT
REBOOT
REBOOT REBOOT
REBOOT REBOOT
REBOOT REBOOT
REBOOT
REBOOT REBOOT
REBOOT REBOOT
REBOOT REBOOT

ball: clipart.pierceinternet.com

REBOOT REBOOT



Memento*s*: 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

Abstract

Transiently powered computing devices such as RFID tags, kinetic energy harvesters, and smart cards typically rely on programs that complete a task under tight time constraints before energy starvation leads to complete loss of volatile memory. *Mementos* is a software system that transforms general-purpose programs into interruptible computations that are protected from frequent power losses by automatic, energy-aware state checkpointing. Mementos comprises a collection of optimization passes for the LLVM compiler infrastructure and a linkable library that exercises hardware support for energy measurement while managing state checkpoints stored in nonvolatile memory. We evaluate Mementos against diverse test cases in a trace-driven simulator of transiently powered RFID-scale devices. Although Mementos's energy checks increase run time when energy is plentiful, they allow Mementos to safely suspend execution when energy dwindles, effectively spreading computation across zero or more power failures. This paper's contributions are: a study of the runtime environment for programs on RFID-scale devices; an energy-aware state checkpointing system for these devices that is implemented for the MSP430 family of microcontrollers; and a trace-driven simulator of transiently powered RFID-scale devices.

Categories and Subject Descriptors C.3 [*SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS*]: Real-time and embedded systems

General Terms Design, Experimentation

Keywords Mementos, RFID-Scale Devices, Computational RFID, Energy-Aware Checkpointing

1. Introduction

Demand for tiny, easily deployable computers has driven the development of general-purpose *transiently powered computers* that lack both batteries and wired power, operating exclusively on energy harvested from remote supplies or the environment. Such devices range from *computational RFIDs* [36]—microcontroller-based devices that harvest RF from readers and communicate via RFID protocols—to general-purpose batteryless sensor devices [45].

Computing under transient power conditions is a challenge. Transiently powered RFID tags use simple state machines instead of supporting general-purpose computation. Contactless smart cards perform more complicated special-purpose computations (e.g. cardholder authentication); however, they offer no execution guarantees, and instead rely on the user to provide the needed RF power for a sufficient period of time. When energy consumption outpaces energy harvesting, these computations fail and must restart from scratch, when adequate energy becomes available.

With ultra-low-power microcontrollers (MCUs), *tiny programmable* devices can perform computation and sensing under RFID-scale energy constraints; however, these MCUs consume more power than conventional RFID circuitry, and energy consumption can easily outpace harvesting, resulting in frequent power loss.

Today, programs that run CPU-intensive operations like cryptography on these devices are pessimistically and painstakingly hand-tuned to complete within a short time window (often under 100 ms) [7, 9]. The usefulness and power of RFID-scale devices can be dramatically improved if designers can confidently write programs without being limited by power failures.

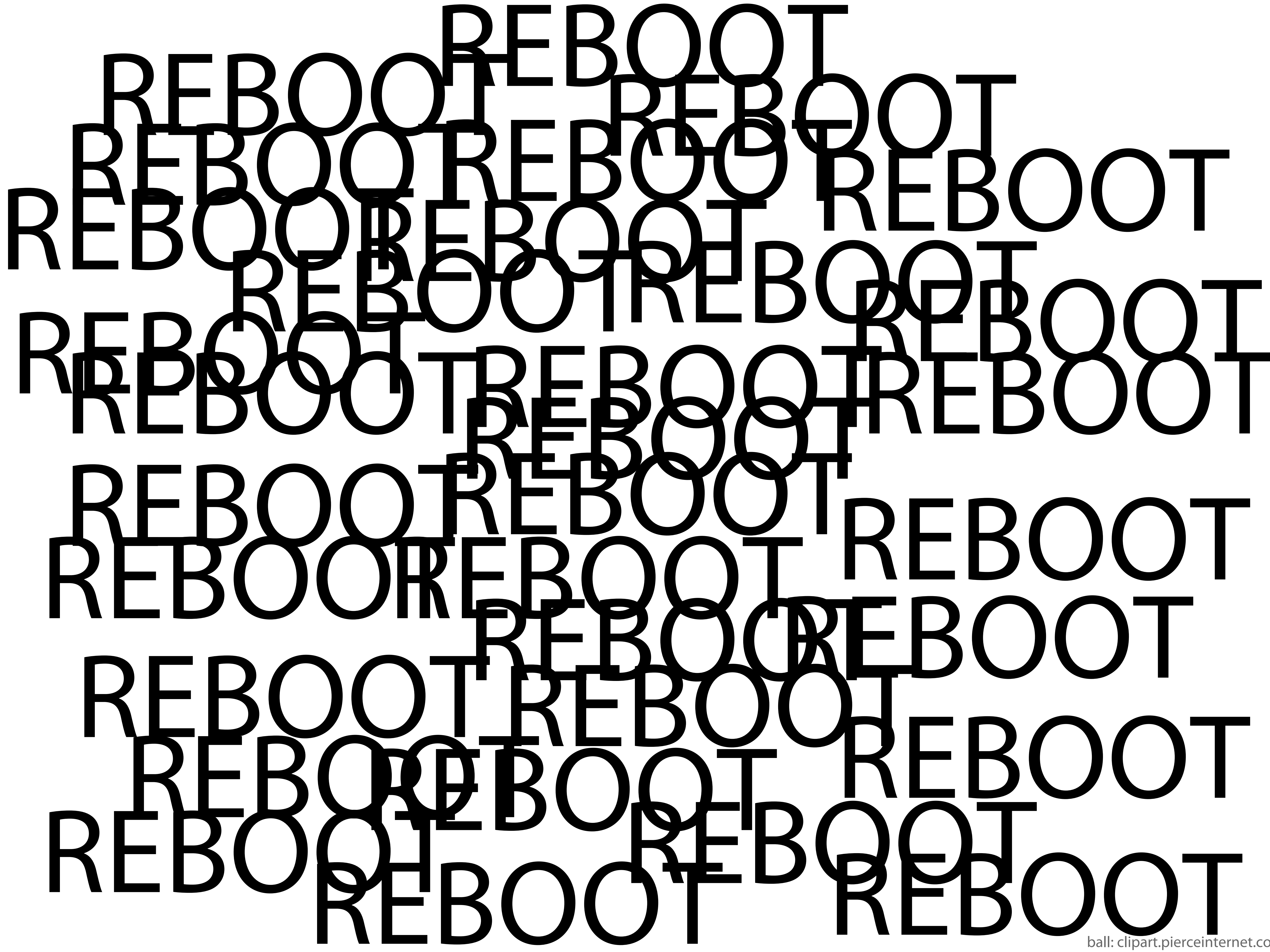
Mementos is a software system that enables long-running computations to span power loss events by combining compile-time instrumentation and run-time energy-aware state checkpointing [1]. At compile time, Mementos inserts function calls that estimate available energy. At run time, Mementos predicts power losses and, when appropriate, saves program state to nonvolatile memory. After a failure, program state is restored and execution continues rather than restarting from scratch.

This paper makes the following contributions: (1) An energy-aware state checkpointing system that splits program execution across multiple lifecycles on transiently powered RFID-scale devices. The system is implemented for the MSP430 family of microcontrollers, requires no hardware modifications to existing devices, and operates automatically at run time without user intervention. (2) A suite of compile-time optimization passes that insert energy checks at control points in a program. The optimization passes employ three different instrumentation strategies that favor different program structures. (3) A trace-driven simulator to evaluate the behavior of programs on transiently powered RFID-scale devices. The simulator, modeled after a prototype hardware device with an off-the-shelf microcontroller, takes executable code as input and simulates power loss events during runs. We evaluate the simulator's accuracy and Mementos's performance under simulation in Section 5.

ASPLOS'11, March 5–11, 2011, Newport Beach, California, USA.

[1] In the indie film *Memento*, the main character would unpredictably lose short-term memory, especially when sleeping. He checkpointed state with notes and tattoos in an attempt to execute long-running tasks.

ball: clipart.pierceinternet.com



ball: clipart.pierceinternet.com

REBOOT
REBOOT REBOOT
REBOOT REBOOT REBOOT
REBOOT REBOOT
REBOOT REBOOT REBOOT
REBOOT REBOOT REBOOT REBOOT
REBOOT REBOOT REBOOT
REBOOT REBOOT REBOOT
REBOOT REBOOT
REBOOT REBOOT REBOOT
REBOOT REBOOT
REBOOT REBOOT REBOOT REBOOT
REBOOT REBOOT REBOOT
REBOOT REBOOT

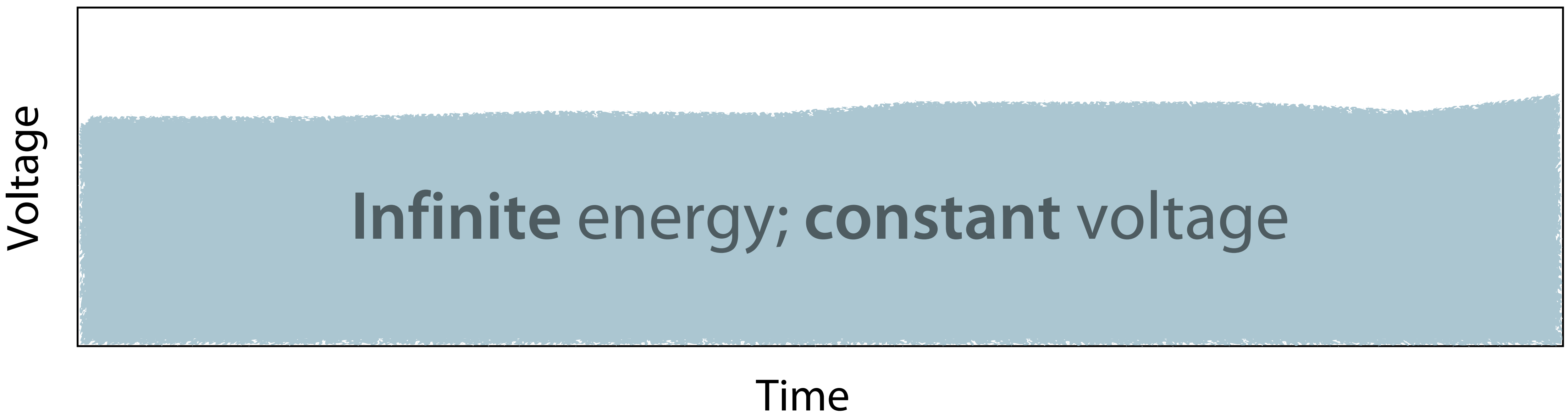
ball: clipart.pierceinternet.com

Robustness Under RF Harvesting

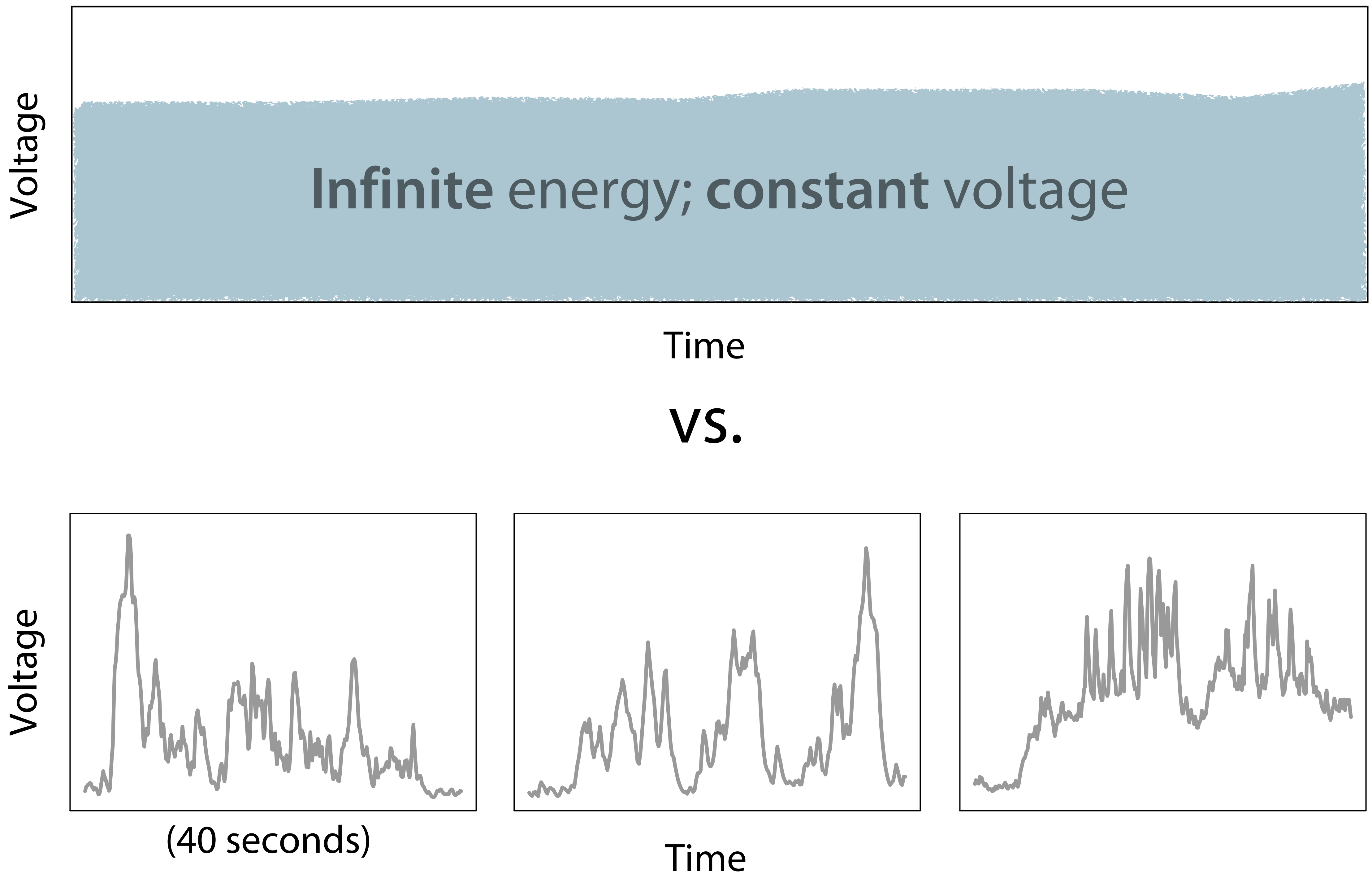
300 ms

- Typical approach: constrain the problem
- **Mementos:** relax constraints to make general-purpose computation feasible

Unpredictable Energy Morass

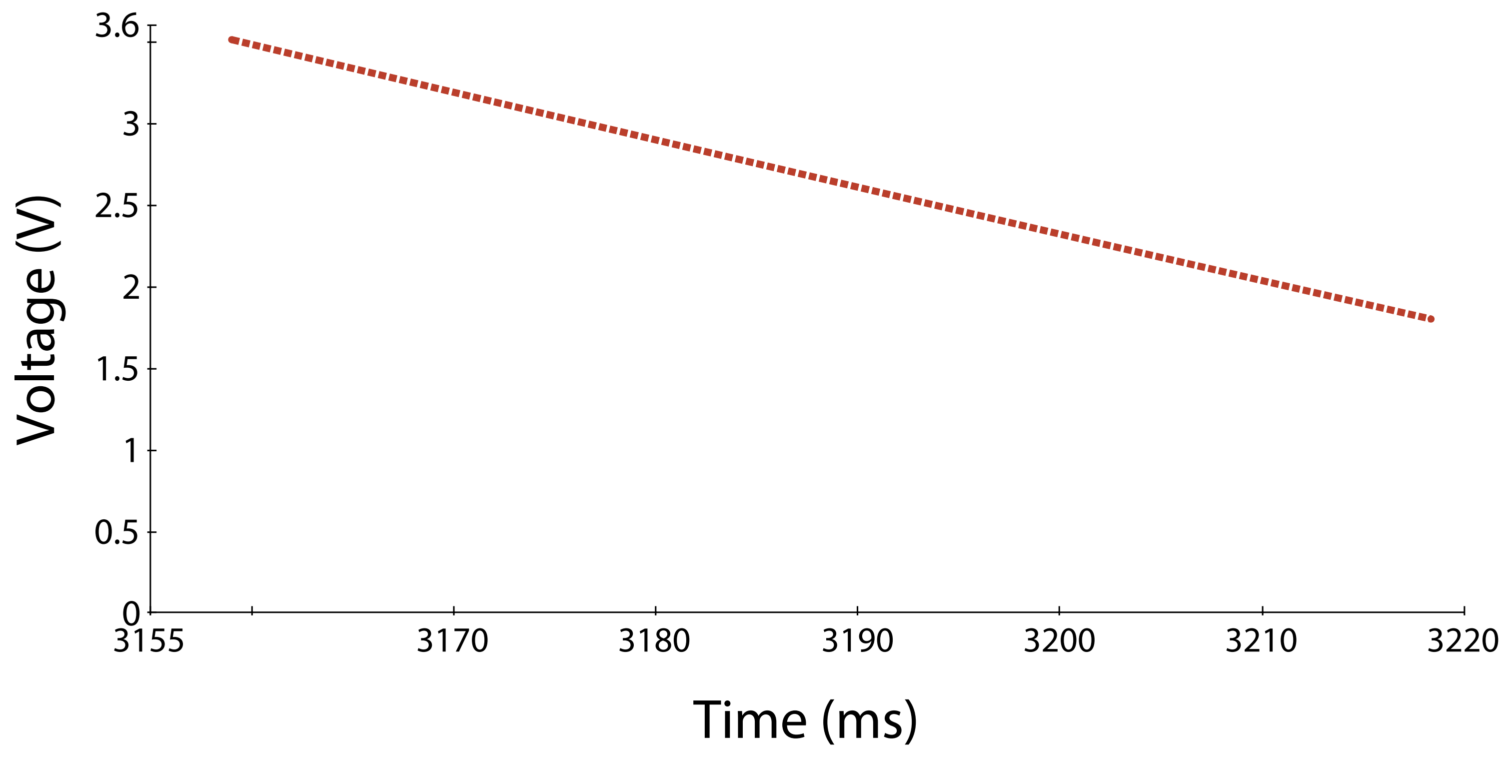


Unpredictable Energy Morass



Mementos Approach

2. Robustness Mechanism

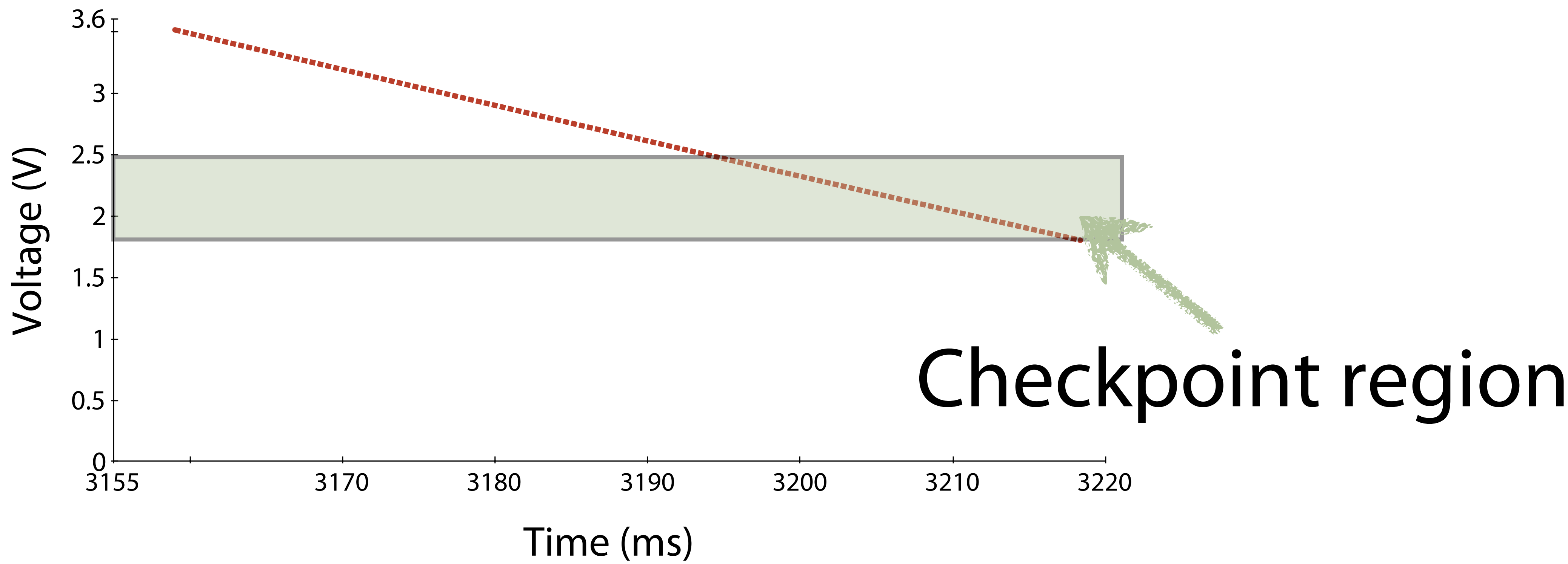


- Checkpoint when failure appears imminent
- Spread computation across reboots

Movie poster: publispain.com

Mementos Approach

2. Robustness Mechanism

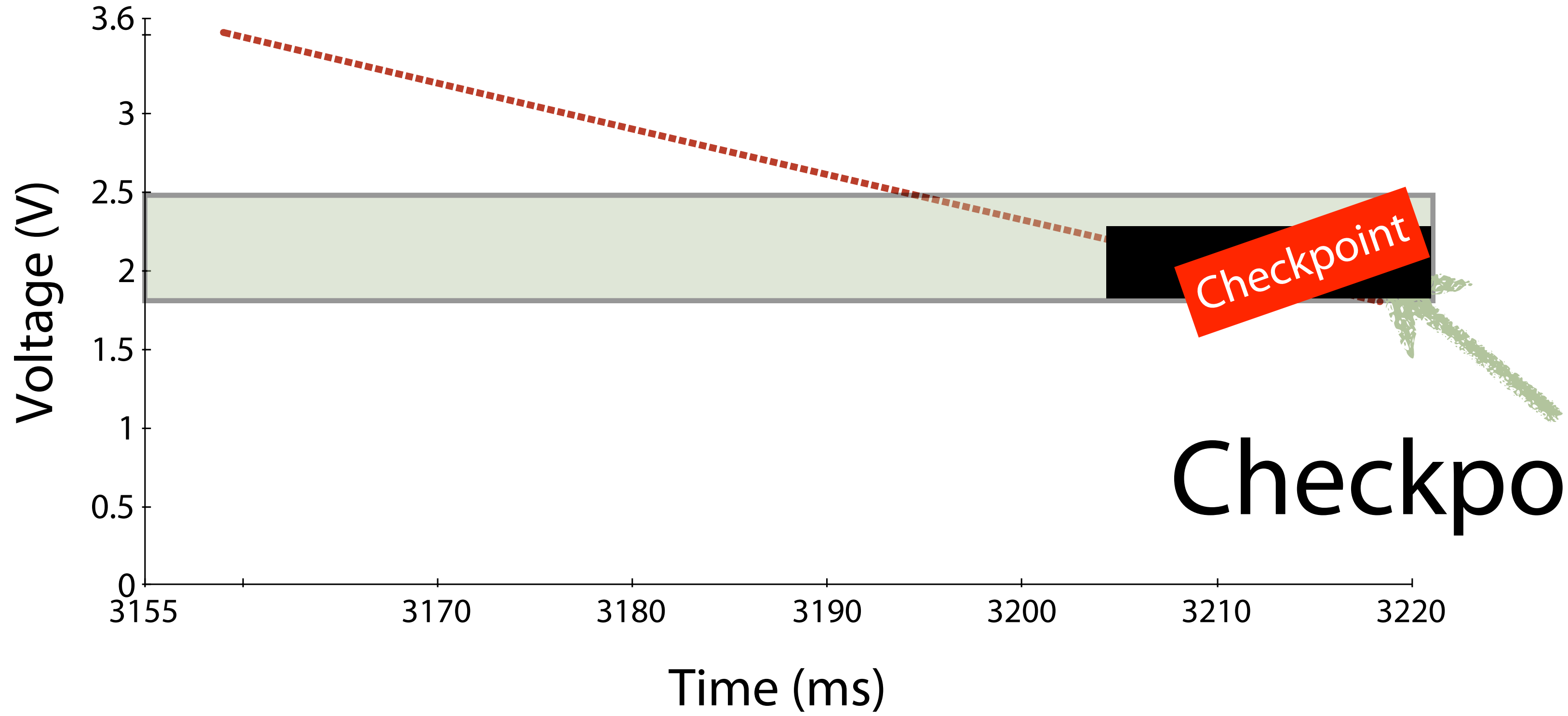


- Checkpoint when failure appears imminent
- Spread computation across reboots

Movie poster: publispain.com

2. Robustness Mechanism

Mementos Approach

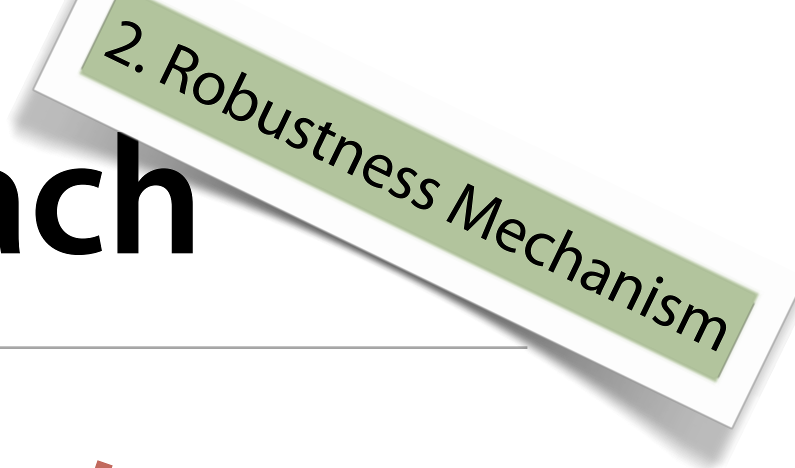


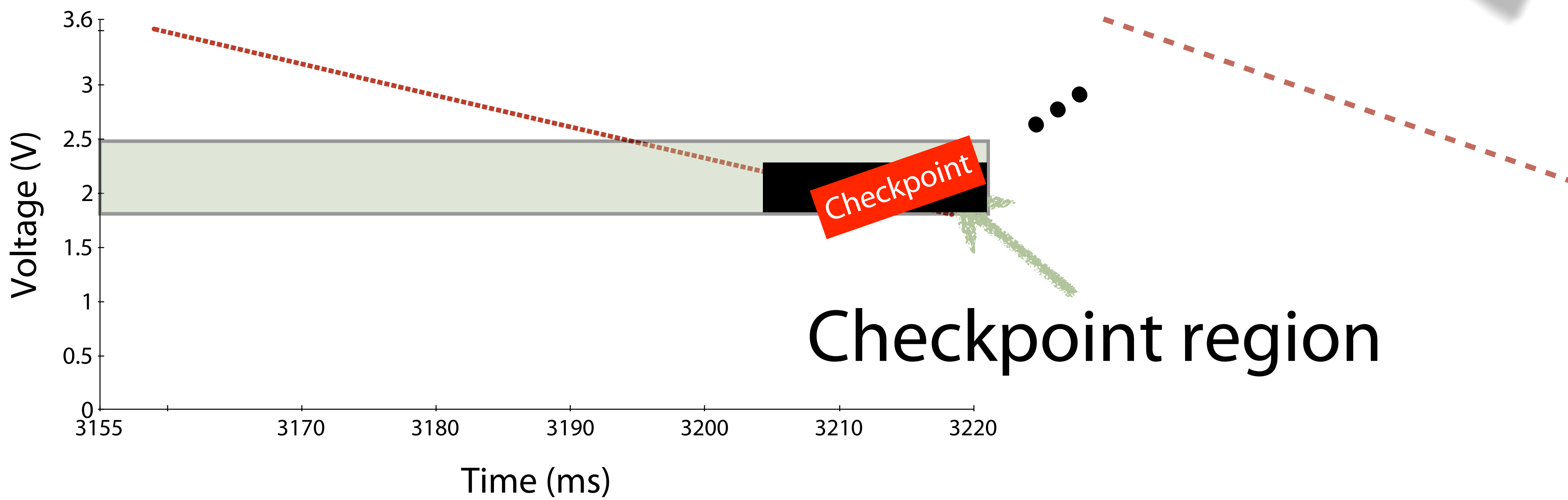
Checkpoint region

- Checkpoint when failure appears imminent
- Spread computation across reboots

Movie poster: publispain.com

Mementos Approach

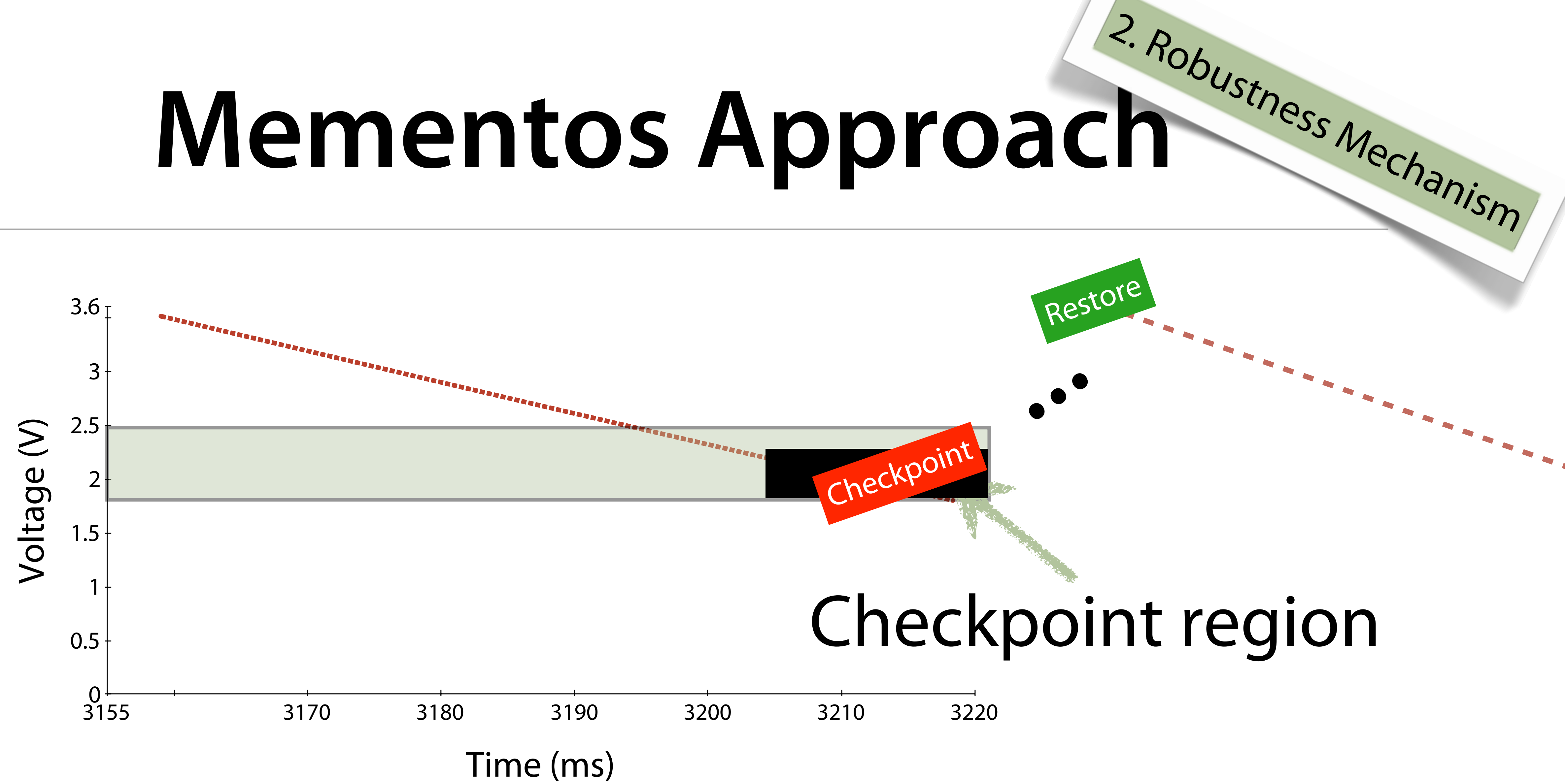




- Checkpoint when failure appears imminent
- Spread computation across reboots

Movie poster: publispain.com

Mementos Approach



- Checkpoint when failure appears imminent
- Spread computation across reboots

Movie poster: publispain.com

Running Example: CRC

- Compute CRC16-CCITT checksum over 2 KB data
- Tight nested loops
- 575,000 CPU cycles ~ 575 ms

crc-vanilla.c (/opt/mementos/src/mementos/samples) - VIM

```c
unsigned short crc16_ccitt(volatile unsigned char *data, unsigned short n) {
    register unsigned short i, j;
    unsigned short crc_16;

    crc_16 = 0xFFFFu; // Equivalent Preset to 0x1D0F
    for (i=0; i<n; i++) {
        crc_16^=data[i] << 8;
        for (j = 0; j < 8; ++j) {
            if (crc_16 & 0x8000) {
                crc_16 <<= 1;
                crc_16 ^= 0x1021; // (CCITT) x16 + x12 + x5 + 1
            } else {
                crc_16 <<= 1;
            }
        }
    }
    return(crc_16^0xFFFFu);
}
```

/opt/mementos/src/mementos/samples/crc-vanilla.c [c] [#1] 12/42

Running Example: CRC

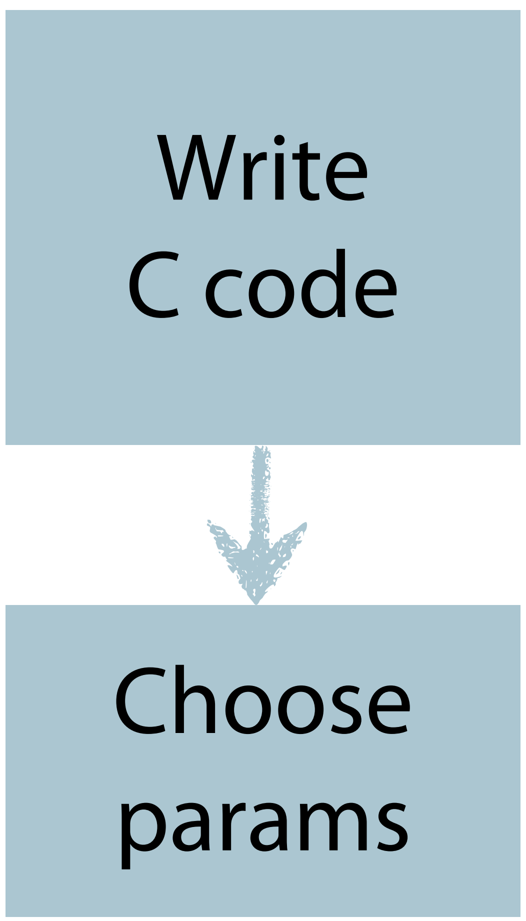
- Compute CRC16-CCITT checksum over 2 KB data
- Tight nested loops Reboots every O(100) ms!
- 575,000 CPU cycles ~ 575 ms

crc-vanilla.c (/opt/mementos/src/mementos/samples) – VIM

```
unsigned short crc16_ccitt(volatile unsigned char *data, unsigned short n) {
    register unsigned short i, j;
    unsigned short crc_16;

    crc_16 = 0xFFFFu; // Equivalent Preset to 0x1D0F
    for (i=0; i<n; i++) {
        crc_16^=data[i] << 8;
        for (j = 0; j < 8; ++j) {
            if (crc_16 & 0x8000) {
                crc_16 <<= 1;
                crc_16 ^= 0x1021; // (CCITT) x16 + x12 + x5 + 1
            } else {
                crc_16 <<= 1;
            }
        }
    }
    return(crc_16^0xFFFFu);
}
/opt/mementos/src/mementos/samples/crc-vanilla.c [c] [#1]                12/42
```

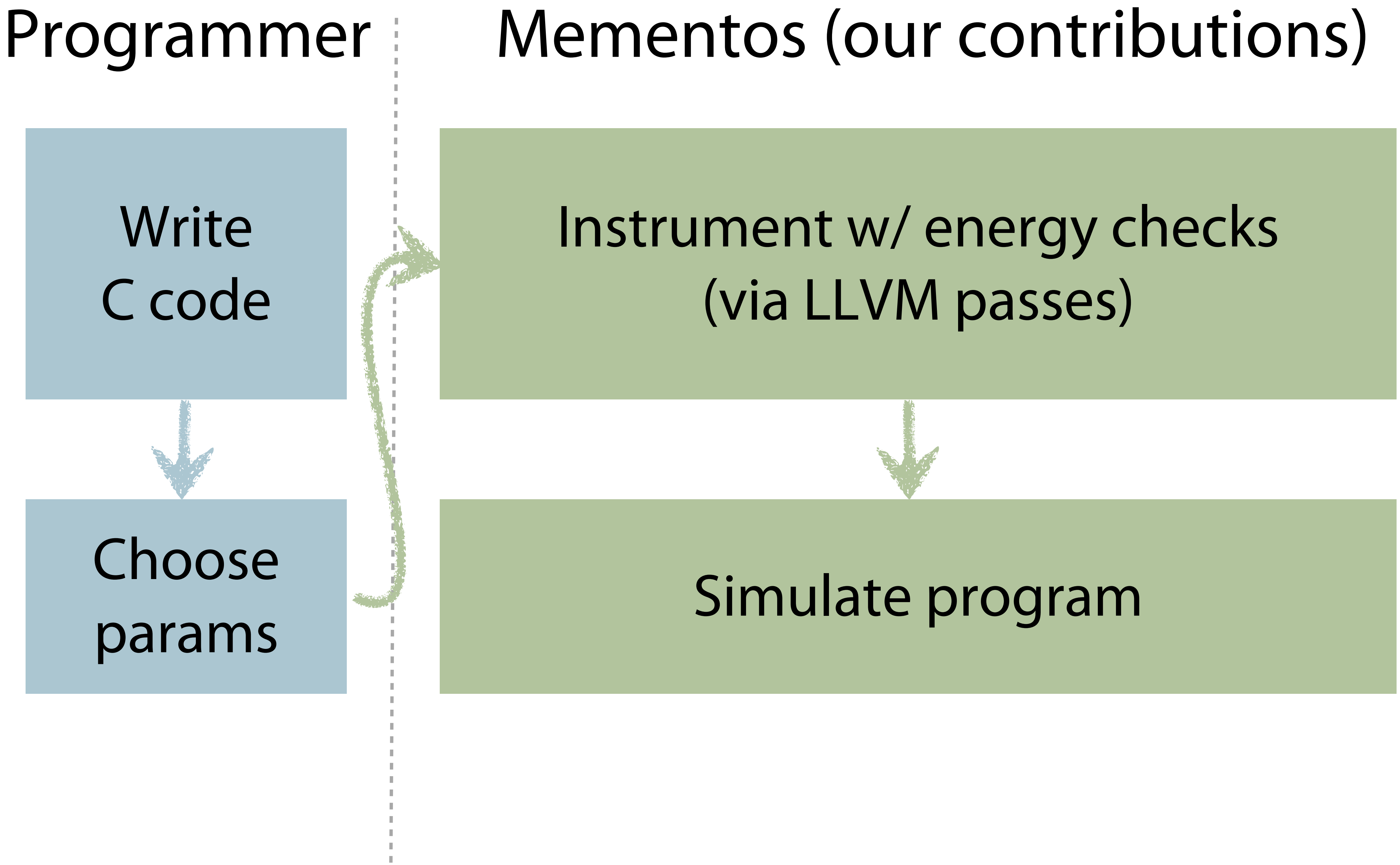
How to Use Mementos

Programmer

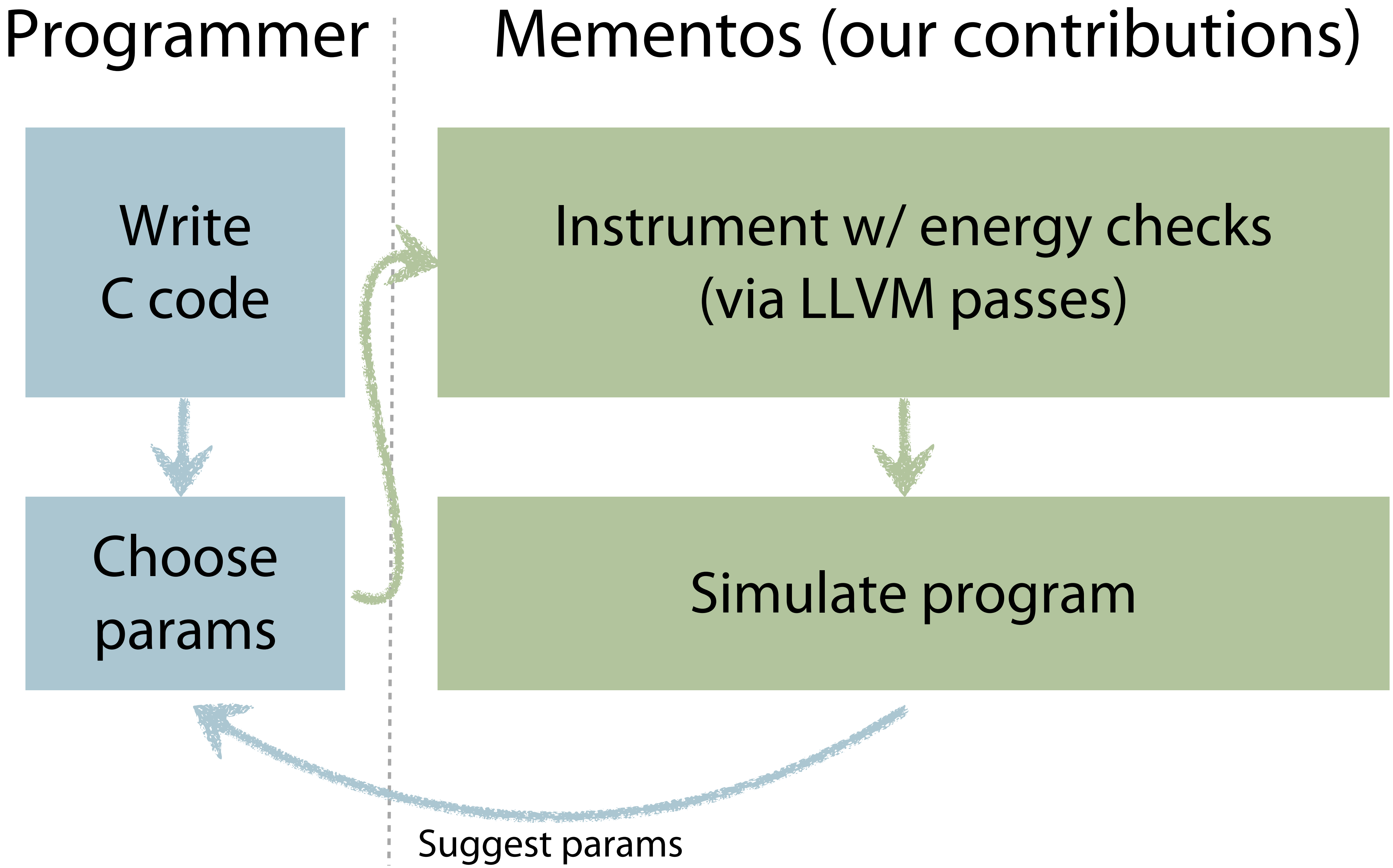


Mementos (our contributions)

How to Use Mementos



How to Use Mementos



How to Use Mementos

Programmer

Write C code

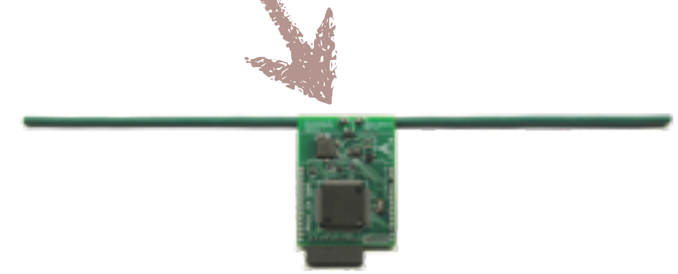
Choose params

Mementos (our contributions)

Instrument w/ energy checks (via LLVM passes)

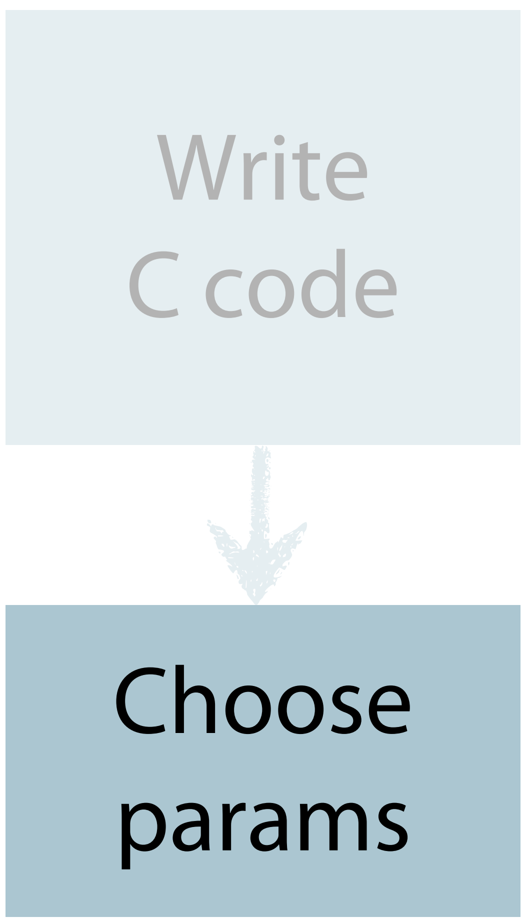
Simulate program

Suggest params



Choosing Parameters (1/2)

Programmer



1) Instrumentation strategy

Choosing Parameters (1/2)

Programmer

Write C code

↓

Choose params

1) Instrumentation strategy

crc-vanilla.c (/opt/mementos/src/mementos/samples) - VIM

```
unsigned short crc16_ccitt(volatile unsigned char *data, unsigned sho
    register unsigned short i, j;
    unsigned short crc_16;

    crc_16 = 0xFFFFu; // Equivalent Preset to 0x1D0F
    for (i=0; i<n; i++) {
        crc_16^=data[i] << 8;
        for (j = 0; j < 8; ++j) {
            if (crc_16 & 0x8000) {
                crc_16 <<= 1;
                crc_16 ^= 0x1021; // (CCITT) x16 + x12 + x5 + 1
            } else {
                crc_16 <<= 1;
            }
        }
    }
    return(crc_16^0xFFFFu);
}
/opt/mementos/src/mementos/samples/crc-vanilla.c [c] [#1]
```

Choosing Parameters (1/2)

Programmer

Write C code

↓

Choose params

1) Instrumentation strategy

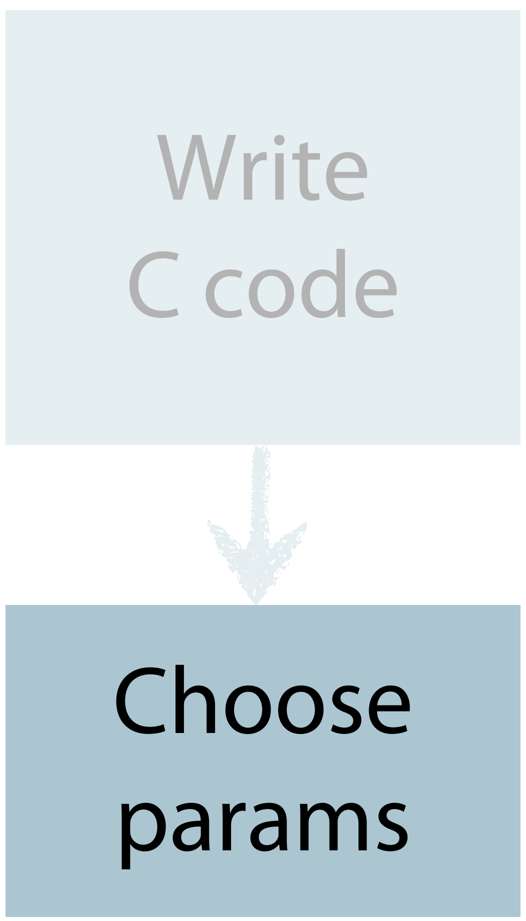
crc-vanilla.c (/opt/mementos/src/mementos/samples) – VIM

```
unsigned short crc16_ccitt(volatile unsigned char *data, unsigned sho
    register unsigned short i, j;
    unsigned short crc_16;

    crc_16 = 0xFFFFu; // Equivalent Preset to 0x1D0F
    for (i=0; i<n; i++) {
        crc_16^=data[i] << 8;
        for (j = 0; j < 8; ++j) {
            if (crc_16 & 0x8000) {
                crc_16 <<= 1;
                crc_16 ^= 0x1021; // (CCITT) x16 + x12 + x5 + 1
            } else {
                crc_16 <<= 1;
            }
        }
    }
    return(crc_16^0xFFFFu);
}
/opt/mementos/src/mementos/samples/crc-vanilla.c [c] [#1]
```

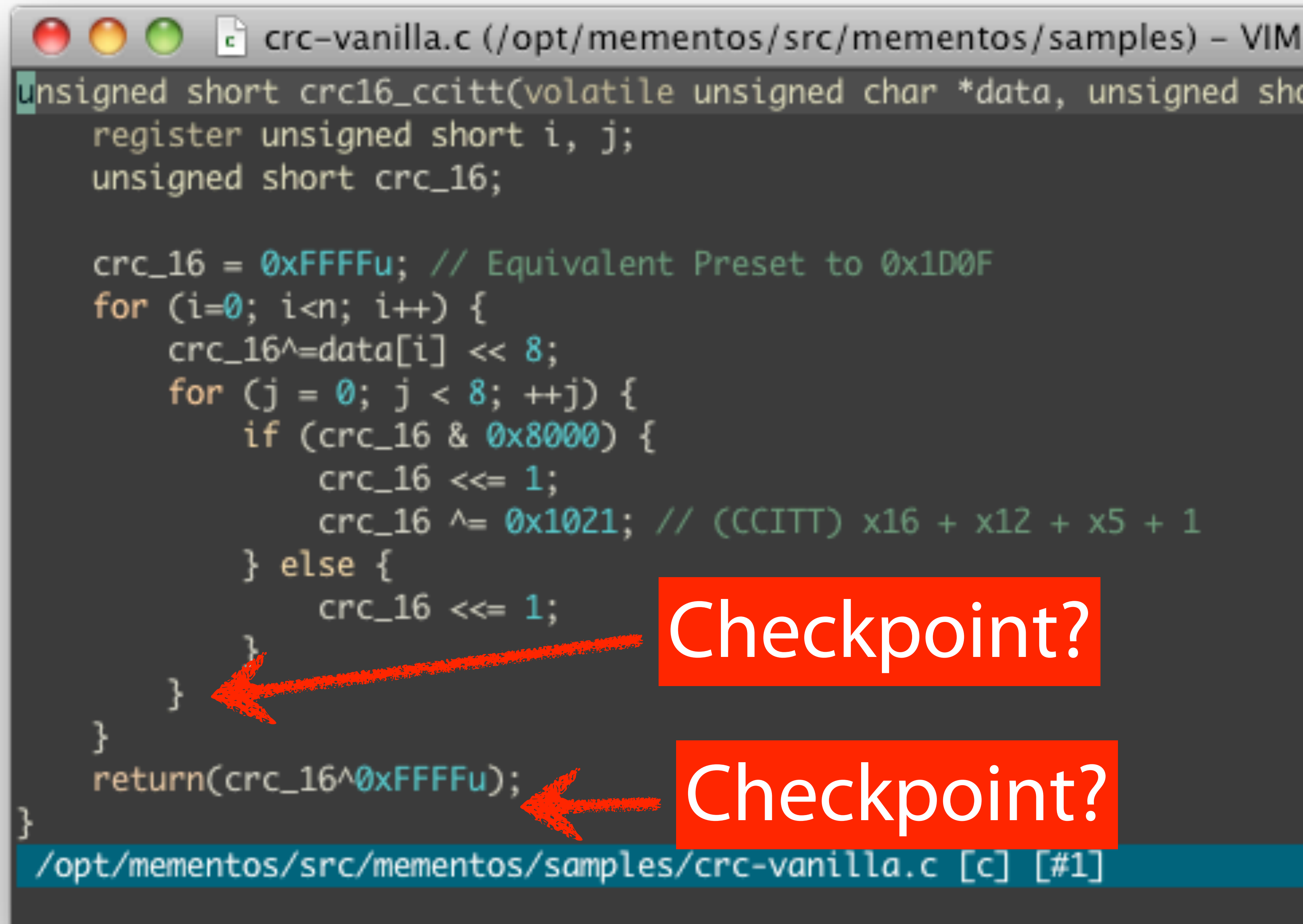
Checkpoint?

Choosing Parameters (1/2)

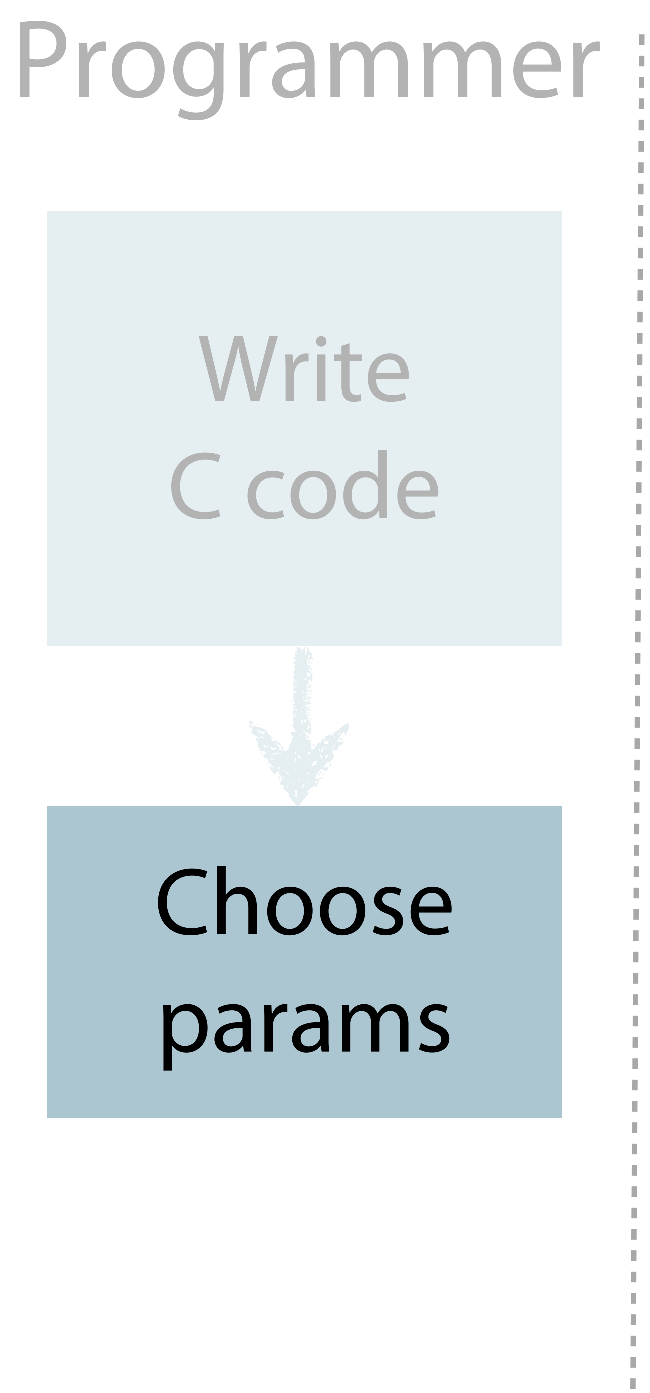
Programmer



1) Instrumentation strategy



Choosing Parameters (2/2)



2) Checkpoint threshold V_{thresh}

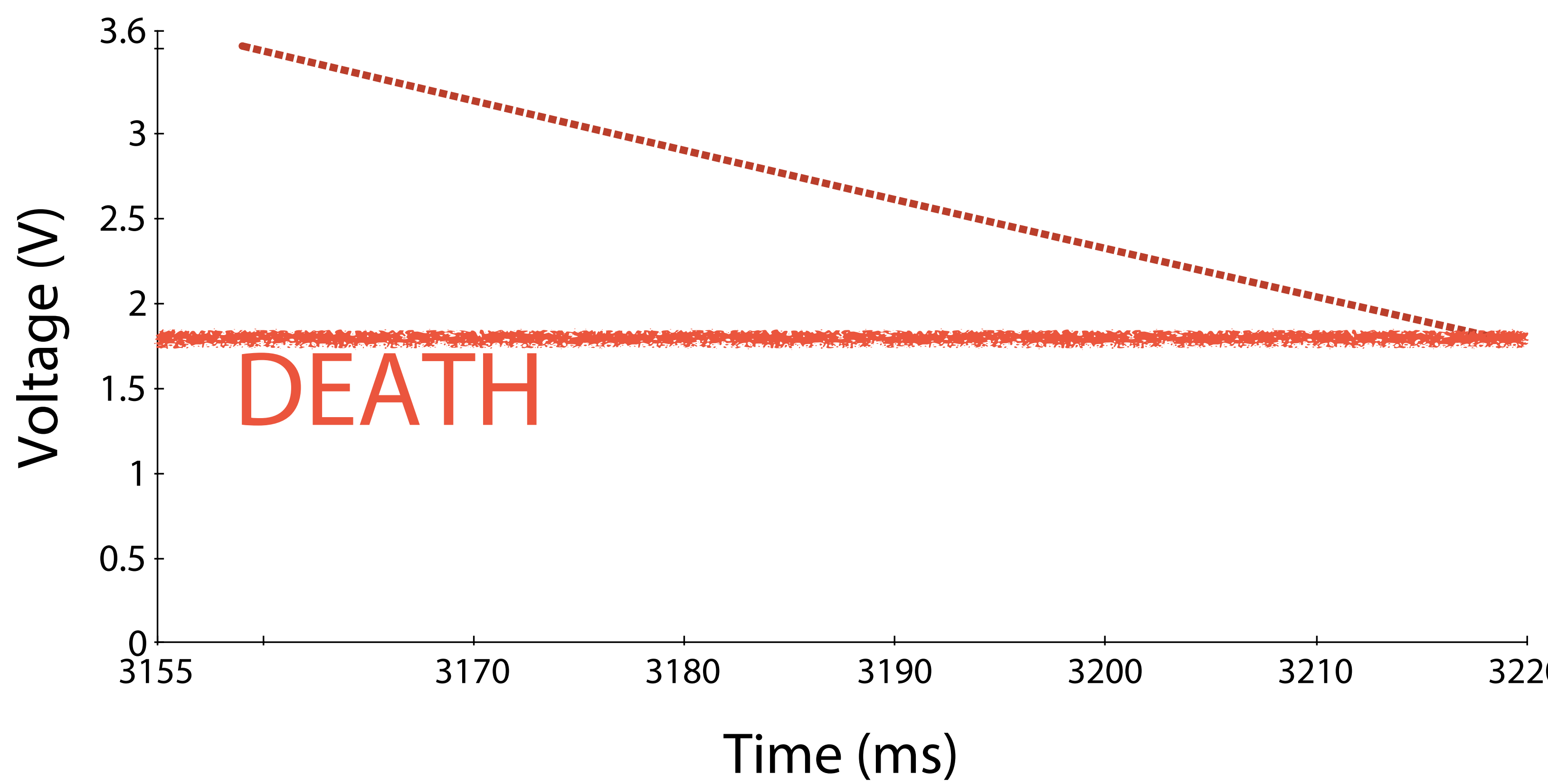
Choosing Parameters (2/2)

Programmer

Write C code

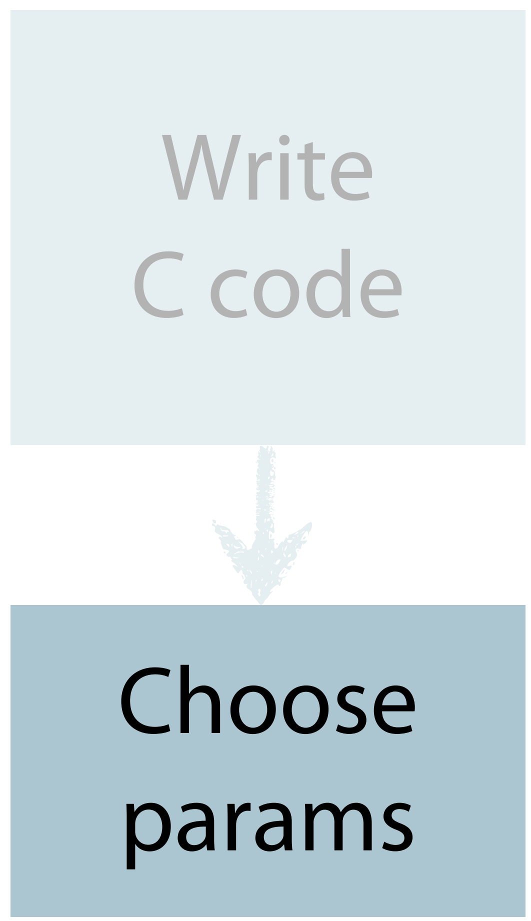
Choose params

2) Checkpoint threshold V_{thresh}

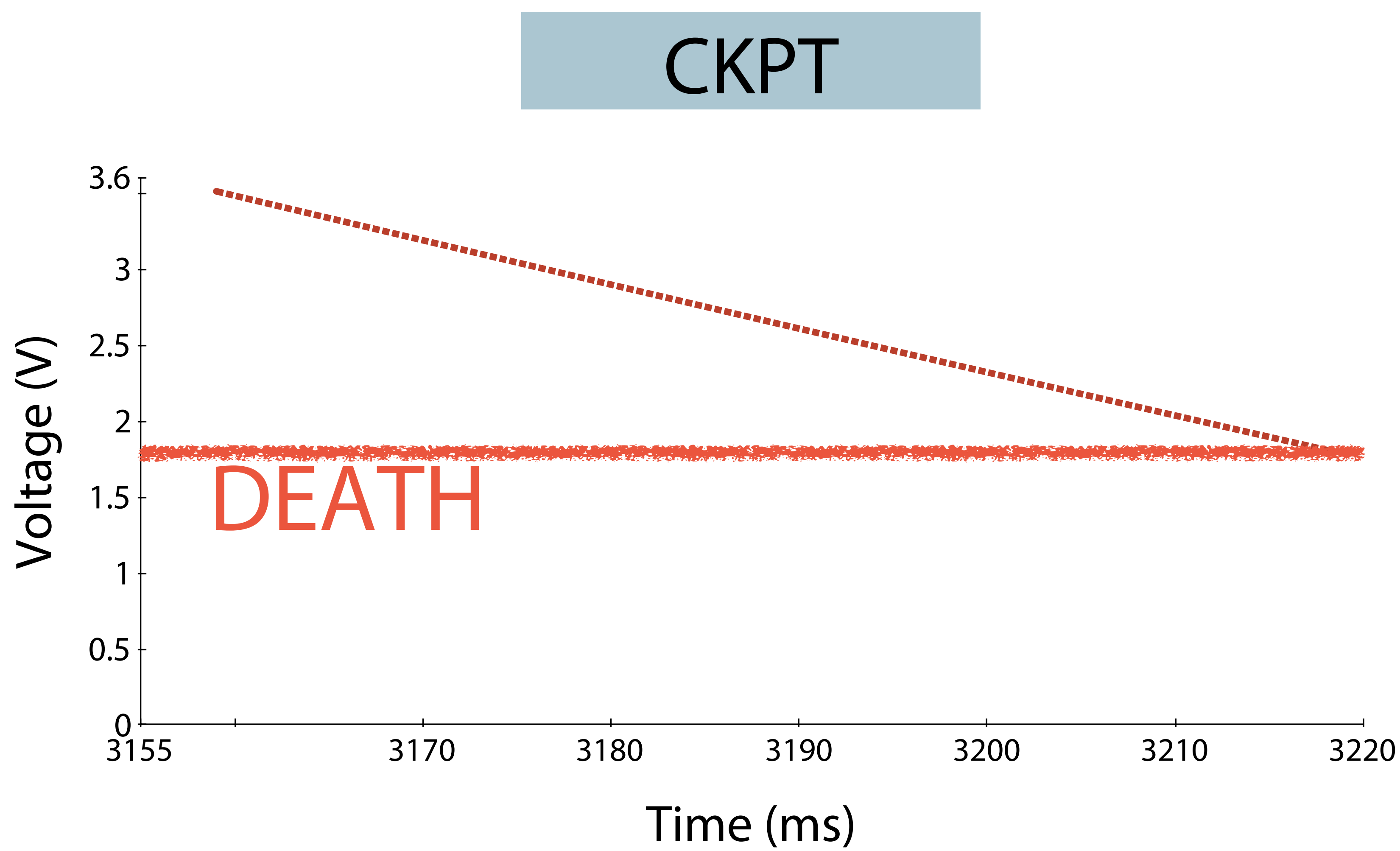


Choosing Parameters (2/2)

Programmer

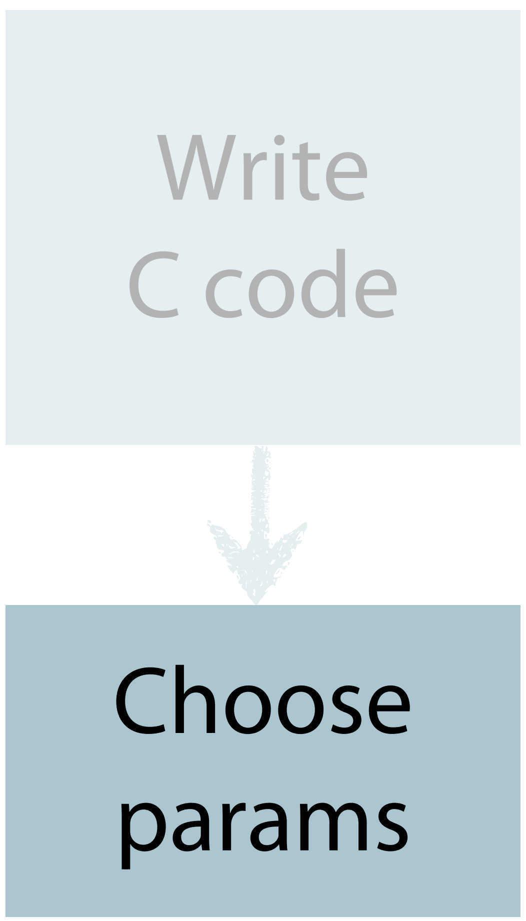


2) Checkpoint threshold V_{thresh}

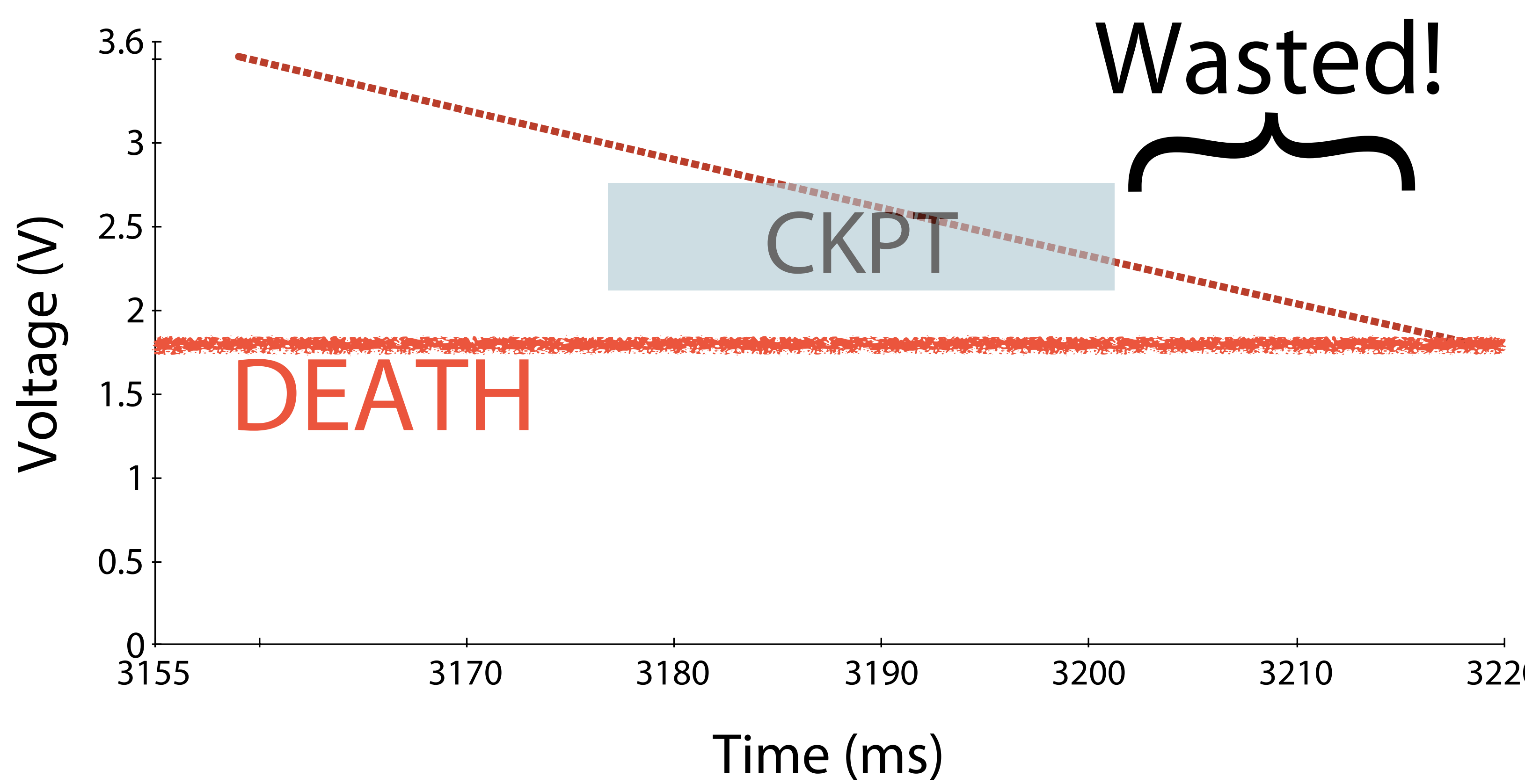


Choosing Parameters (2/2)

Programmer

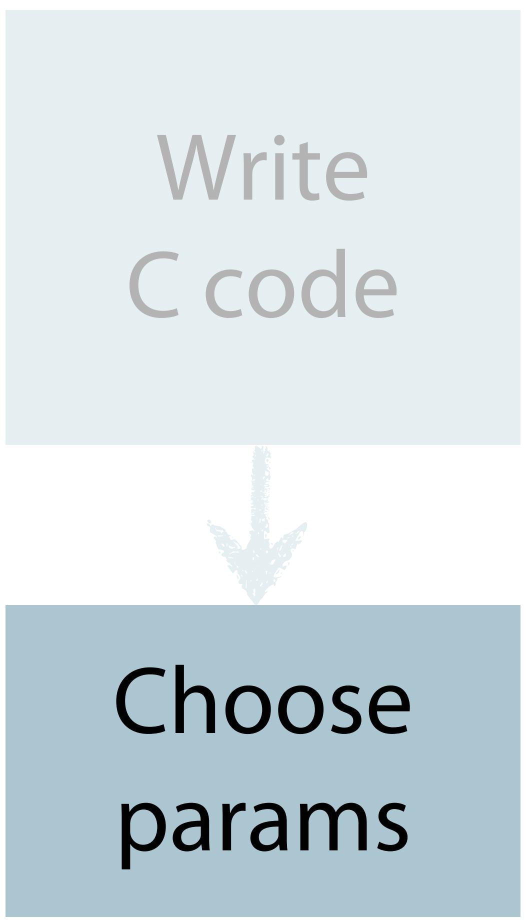


2) Checkpoint threshold V_{thresh}

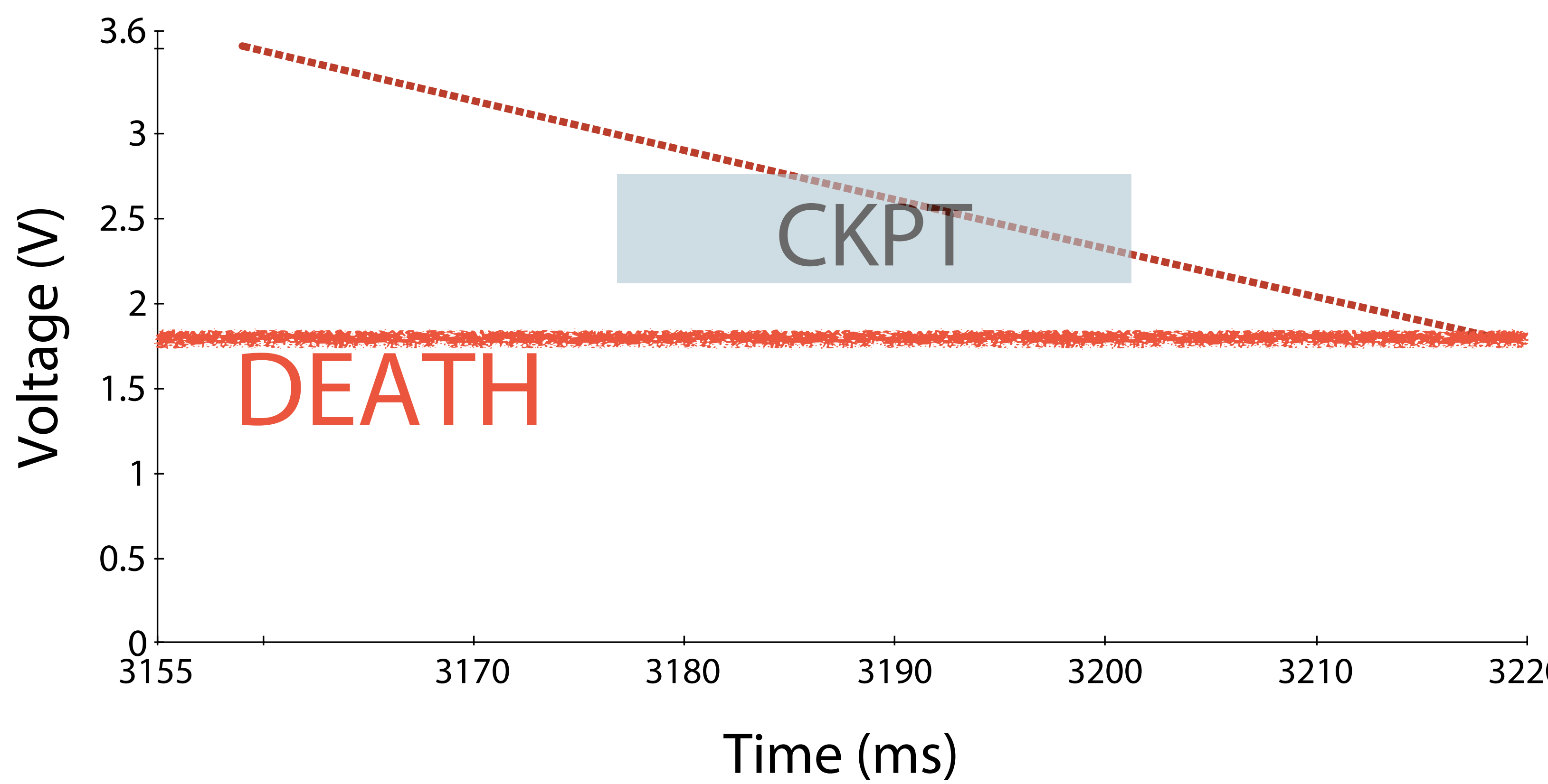


Choosing Parameters (2/2)

Programmer



2) Checkpoint threshold V_{thresh}



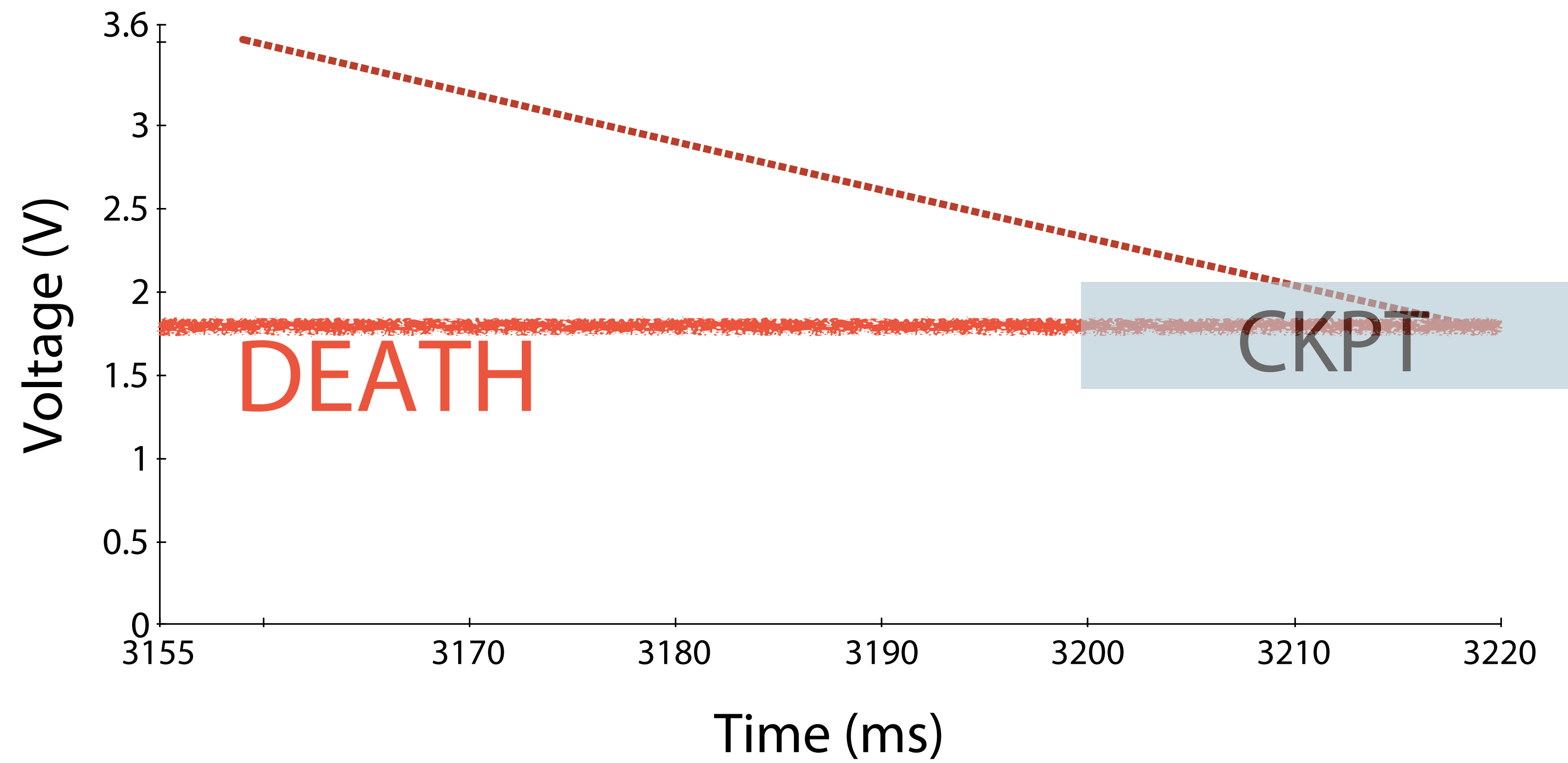
Choosing Parameters (2/2)

Programmer

Write C code

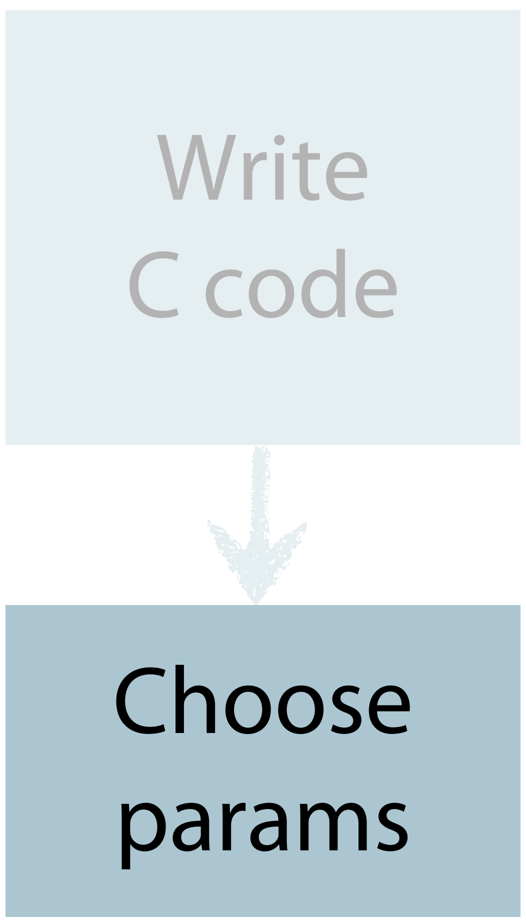
Choose params

2) Checkpoint threshold V_{thresh}

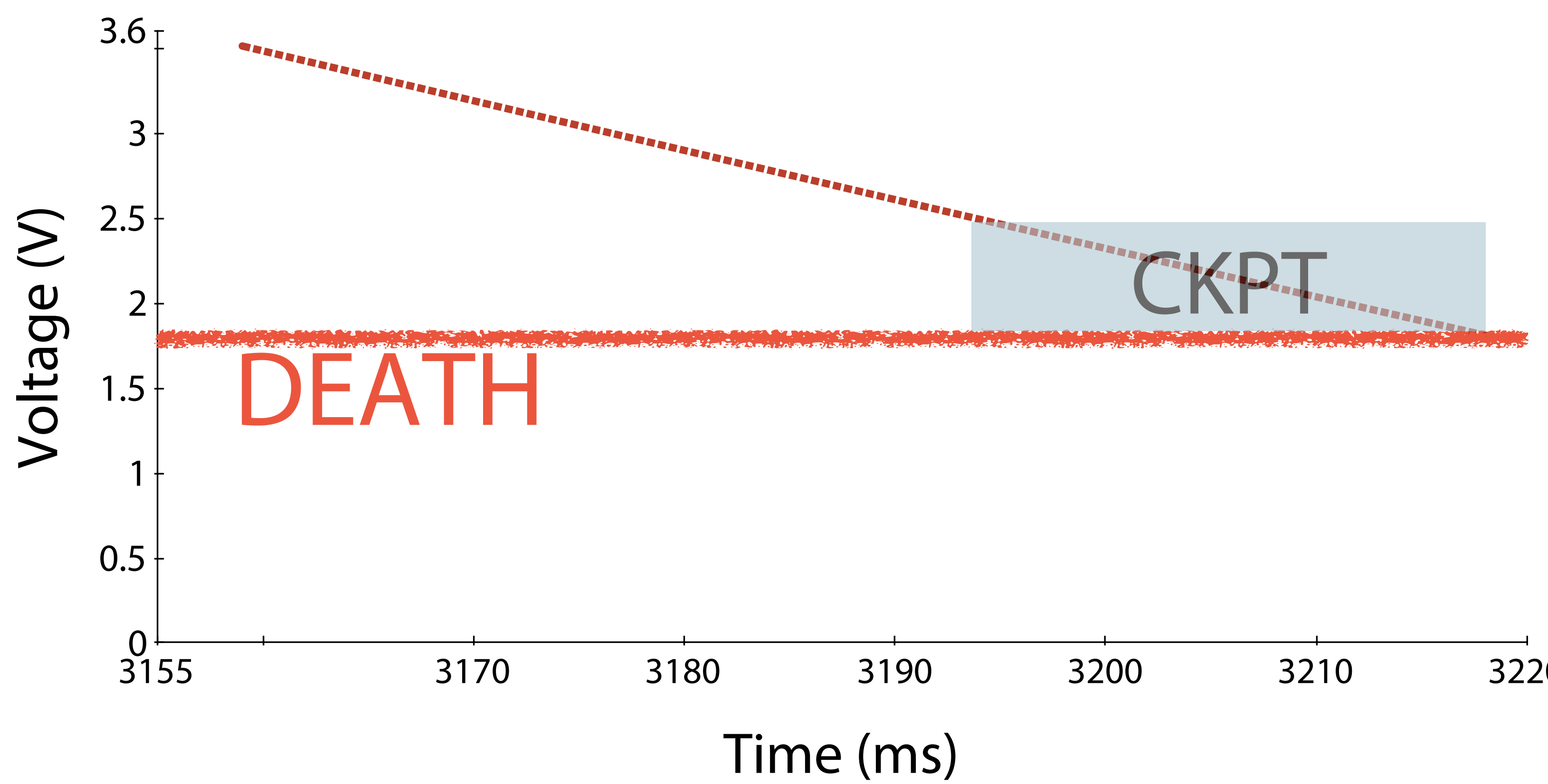


Choosing Parameters (2/2)

Programmer



2) Checkpoint threshold V_{thresh}



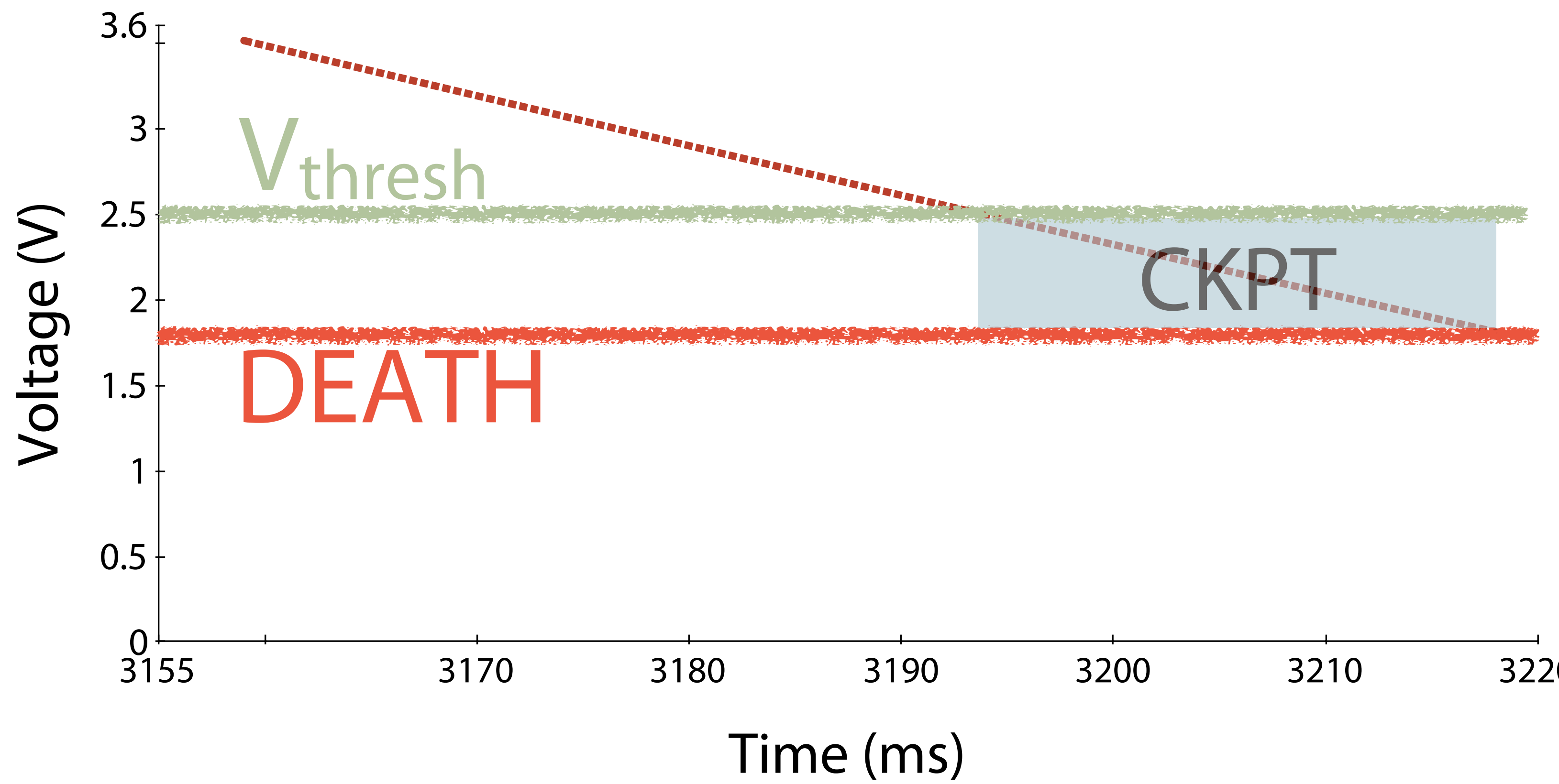
Choosing Parameters (2/2)

Programmer

Write C code

Choose params

2) Checkpoint threshold V_{thresh}



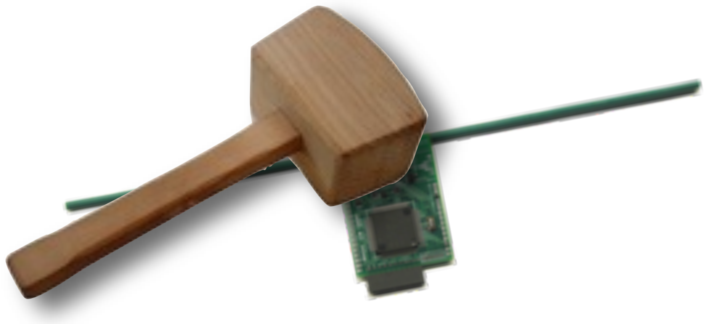
Assorted Challenges

Checkpointing isn't trivial in this context

- No FTL; manage flash ourselves
- Can't overwrite arbitrary bit patterns in flash memory ➔ tricky checkpoint maintenance

Working on these devices is painful

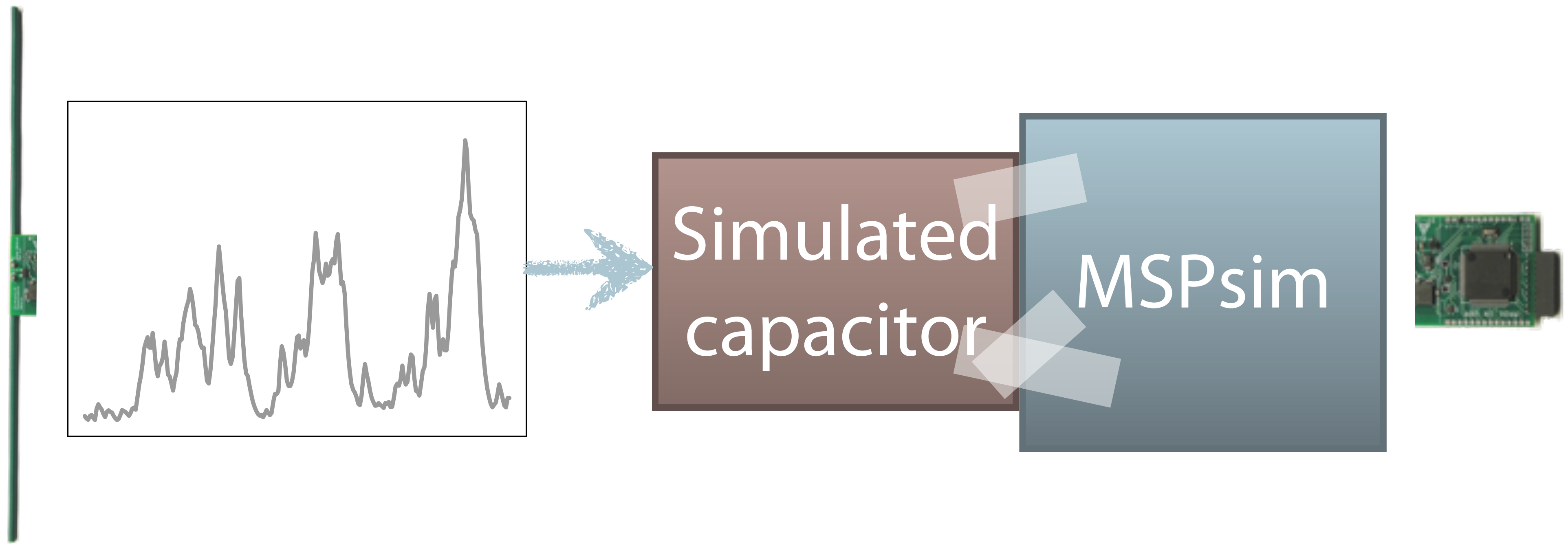
- Fickle harvesting ➔ runs not reproducible
- Limited visibility into running hardware



Trace-Driven Simulator

- Based on MSPsim — cycle-accurate, open-source MSP430 simulator [EWSN '07]
- We augmented MSPsim with notions of energy (harvester, capacitor, power loss)

Trace-Driven Simulator



- Based on MSPsim — cycle-accurate, open-source MSP430 simulator [EWSN '07]
- We augmented MSPsim with notions of energy (harvester, capacitor, power loss)

Accurate Energy Simulation

- Simulated capacitor obeys capacitor equations to buffer incoming energy

- Validated with microbenchmarks (all chip modes, all instruction classes)
  - Measured MSP430 current down to μA
  - Details in paper

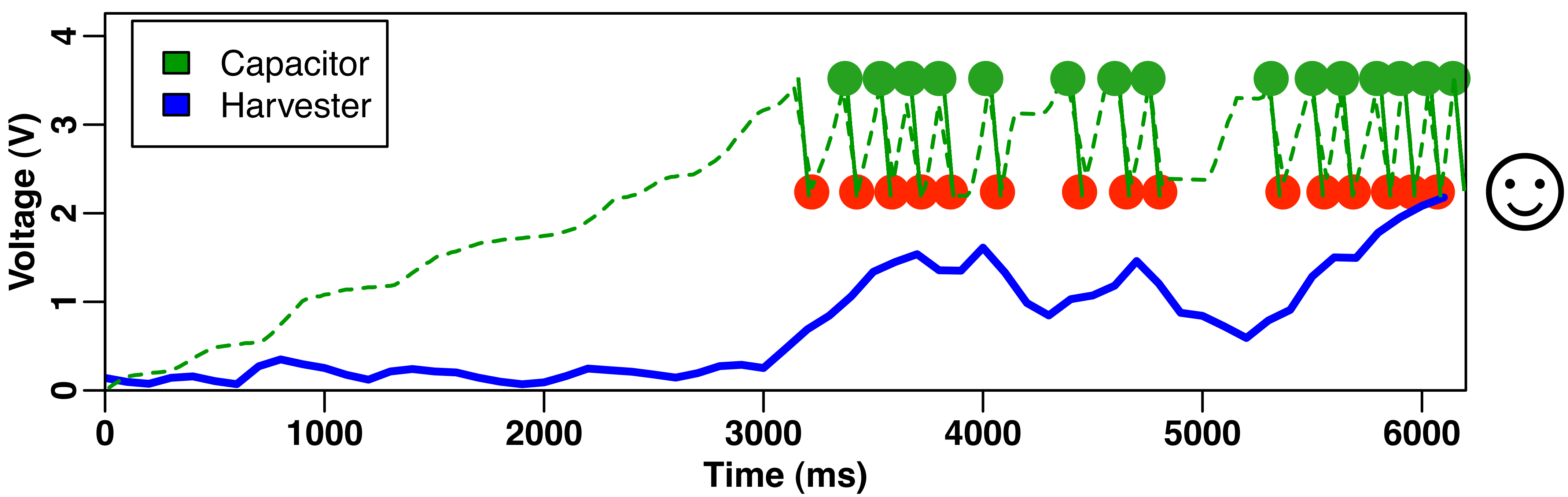
Moo WISP: Hong Zhang; Scalpel: H-G UK; EE 101: UC Santa Cruz

Straightforward Simulation

- Simulator input: <*executable, voltage trace*>
- Output: <*# reboots to completion, # CPU cycles, total time, execution trace*>

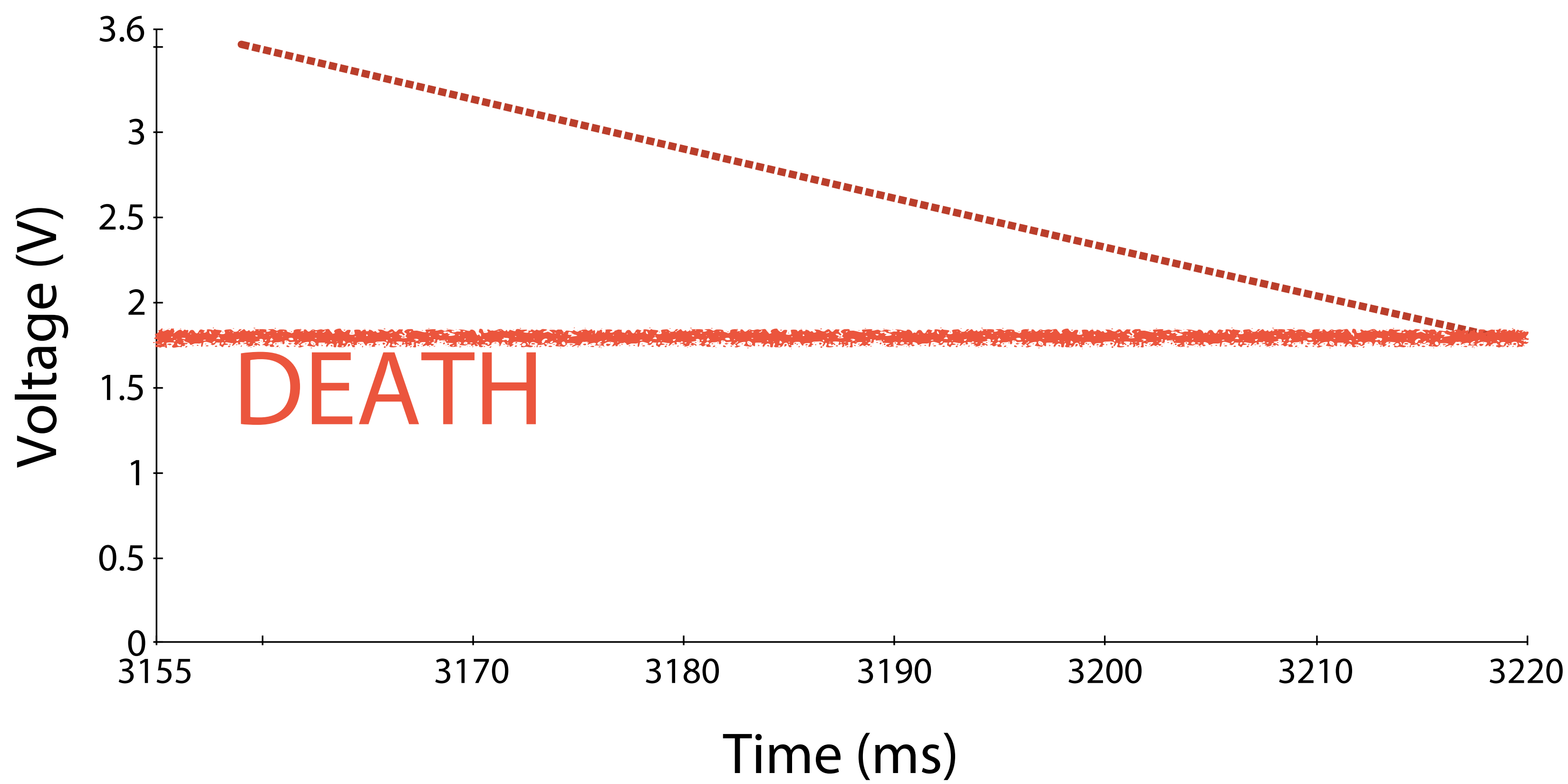
Straightforward Simulation

- Simulator input: <*executable, voltage trace*>
- Output: <*# reboots to completion, # CPU cycles, total time, execution trace*>



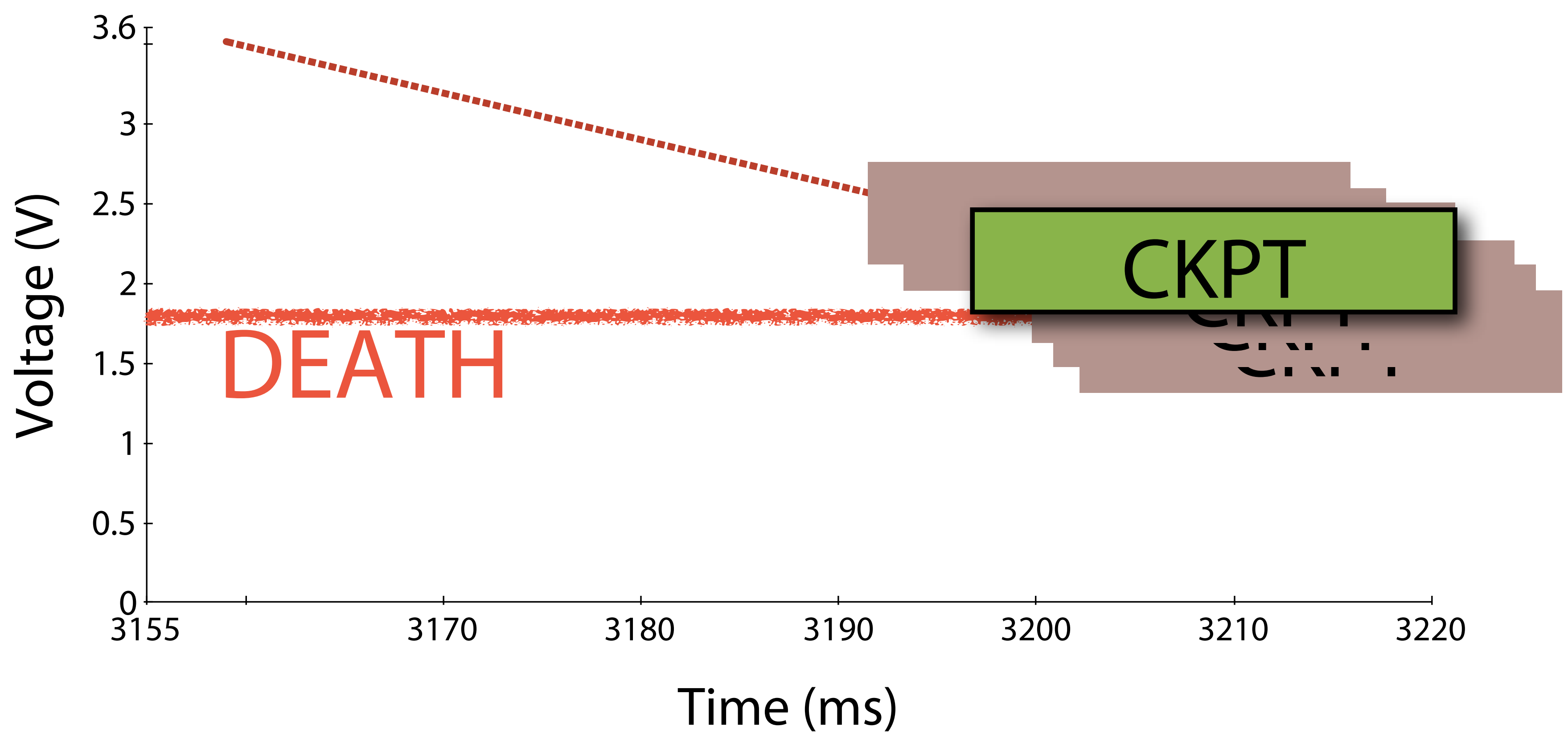
Simulation with an Oracle

- Checkpoint oracle finds last practicable opportunity by binary search on V_{thresh}
  - Uninstrumented code ➔ best-case estimate
  - Final report: *lower bound* for V_{thresh}



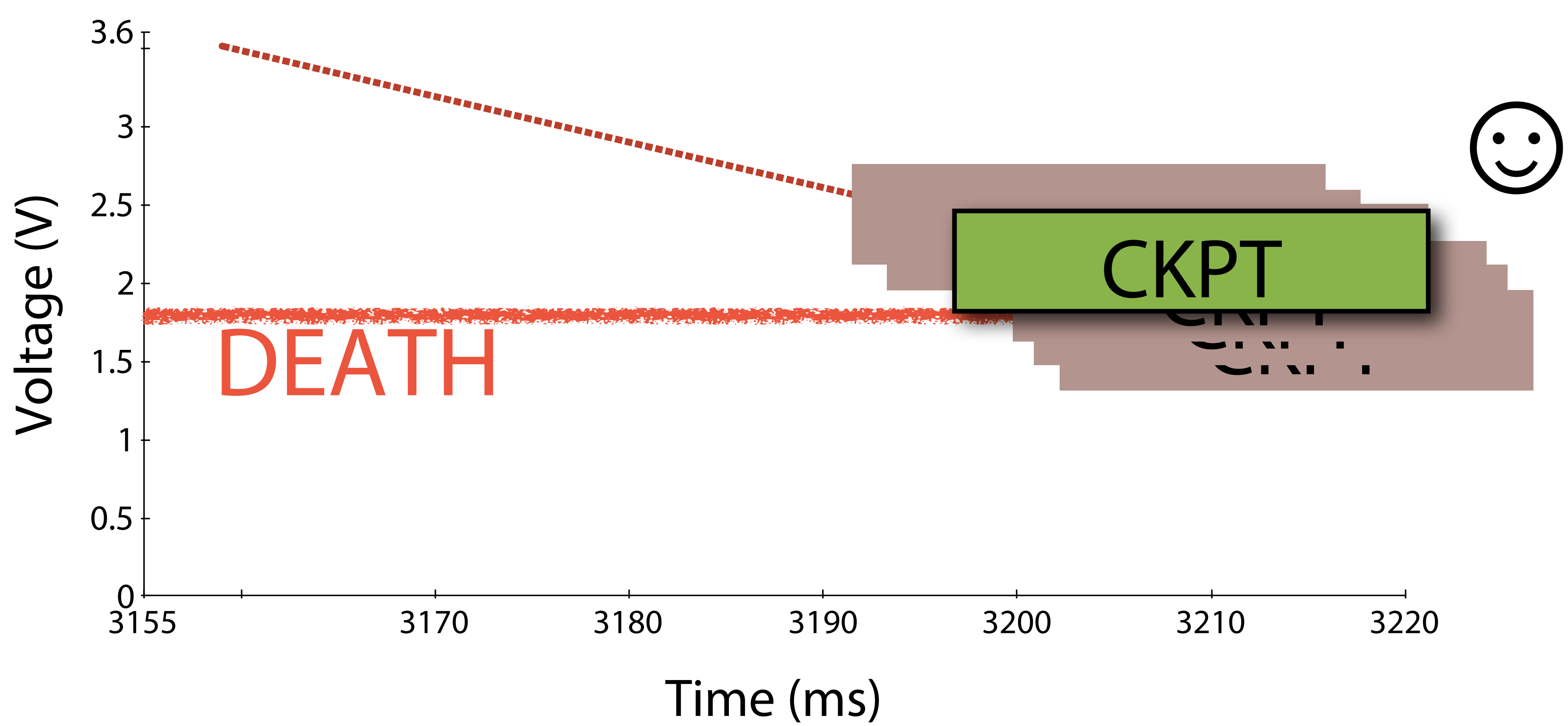
Simulation with an Oracle

- Checkpoint oracle finds last practicable opportunity by binary search on V_{thresh}
  - Uninstrumented code ➔ best-case estimate
  - Final report: *lower bound* for V_{thresh}



Simulation with an Oracle

- Checkpoint oracle finds last practicable opportunity by binary search on V_{thresh}
  - Uninstrumented code ➔ best-case estimate
  - Final report: *lower bound* for V_{thresh}



Evaluation

- **High-level:** Mementos splits execution in simulation and on hardware
- Focus on CRC example test case
- Baselines:
  - Execution without Mementos
  - Execution against checkpoint oracle

Constant Part of Overhead

- Impact on code memory (NVRAM):
  - 2.4 KB for Mementos library
  - 1 KB reserved checkpoint storage
- Impact on run time:
  - ~0.1 ms per energy check (mostly ADC read)
  - CRC (46 bytes): checkpoint 4 ms, restore 2 ms

Constant Part of Overhead

- Impact on code memory (NVRAM):
  - 2.4 KB for Mementos library
  - 1 KB reserved checkpoint storage
- Impact on run time:
  - ~0.1 ms per energy check (mostly ADC read)
  - CRC (46 bytes): checkpoint 4 ms, restore 2 ms

2 ms boot vs. TinyOS ≥100 ms

CRC Test Case

crc-vanilla.c (/opt/mementos/src/mementos/samples) - VIM

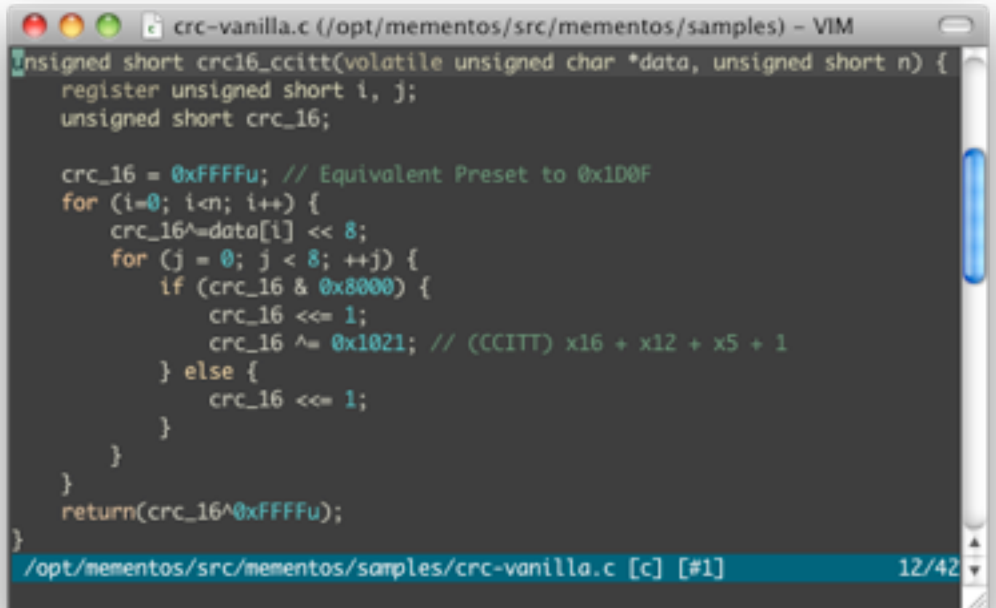
```c
unsigned short crc16_ccitt(volatile unsigned char *data, unsigned short n) {
    register unsigned short i, j;
    unsigned short crc_16;

    crc_16 = 0xFFFFu; // Equivalent Preset to 0x1D0F
    for (i=0; i<n; i++) {
        crc_16^=data[i] << 8;
        for (j = 0; j < 8; ++j) {
            if (crc_16 & 0x8000) {
                crc_16 <<= 1;
                crc_16 ^= 0x1021; // (CCITT) x16 + x12 + x5 + 1
            } else {
                crc_16 <<= 1;
            }
        }
    }
    return(crc_16^0xFFFFu);
}
```

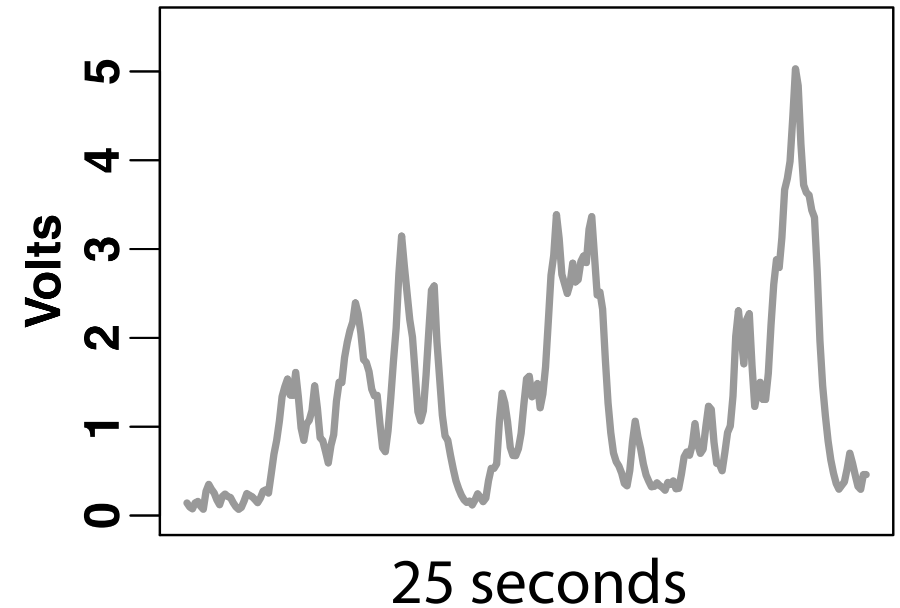
/opt/mementos/src/mementos/samples/crc-vanilla.c [c] [#1] 12/42

Uninstrumented, unlimited energy:

575,315 cycles
575 ms

CRC Test Case





Uninstrumented, unlimited energy:	**Oracle:**	**Best execution:**
575,315 cycles	685,608 cycles	761,983 cycles
575 ms	~4,000 ms	6,145 ms
	14 reboots	16 reboots
	$V_{thresh} \geq 2.35$ V	$V_{thresh} = 2.6$ V

CRC Test Case

crc-vanilla.c (/opt/mementos/src/mementos/samples) - VIM

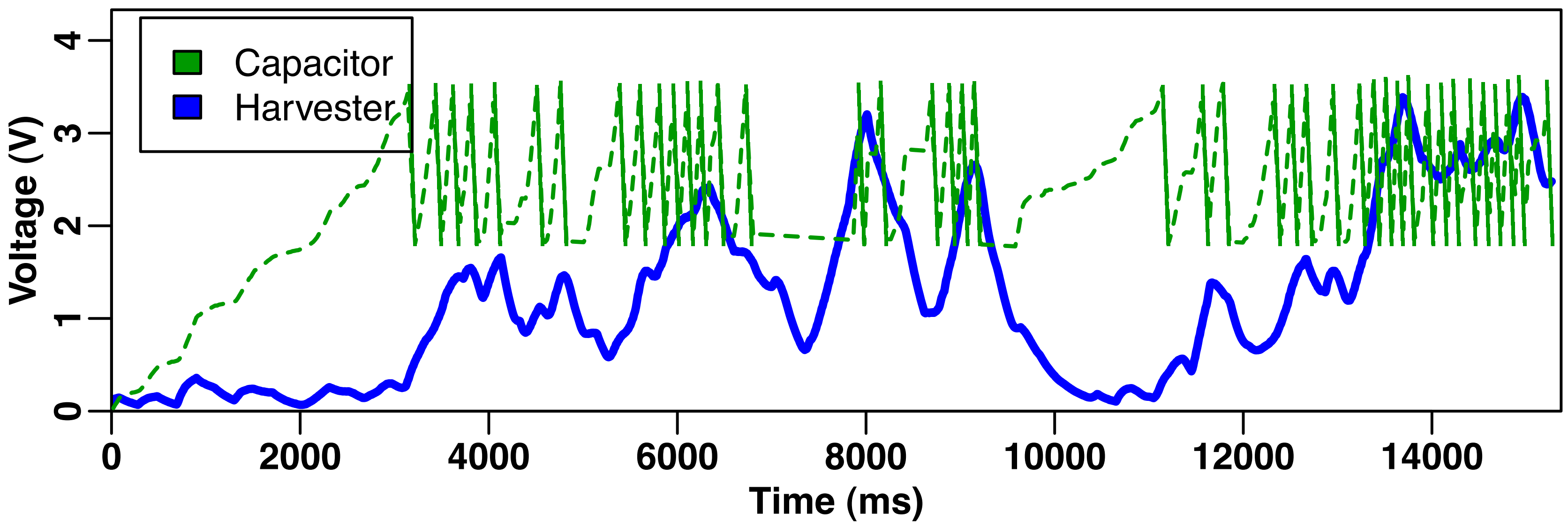
```c
unsigned short crc16_ccitt(volatile unsigned char *data, unsigned short n) {
    register unsigned short i, j;
    unsigned short crc_16;

    crc_16 = 0xFFFFu; // Equivalent Preset to 0x1D0F
    for (i=0; i<n; i++) {
        crc_16^=data[i] << 8;
        for (j = 0; j < 8; ++j) {
            if (crc_16 & 0x8000) {
                crc_16 <<= 1;
                crc_16 ^= 0x1021; // (CCITT) x16 + x12 + x5 + 1
            } else {
                crc_16 <<= 1;
            }
        }
    }
    return(crc_16^0xFFFFu);
}
```

/opt/mementos/src/mementos/samples/crc-vanilla.c [c] [#1] 12/42

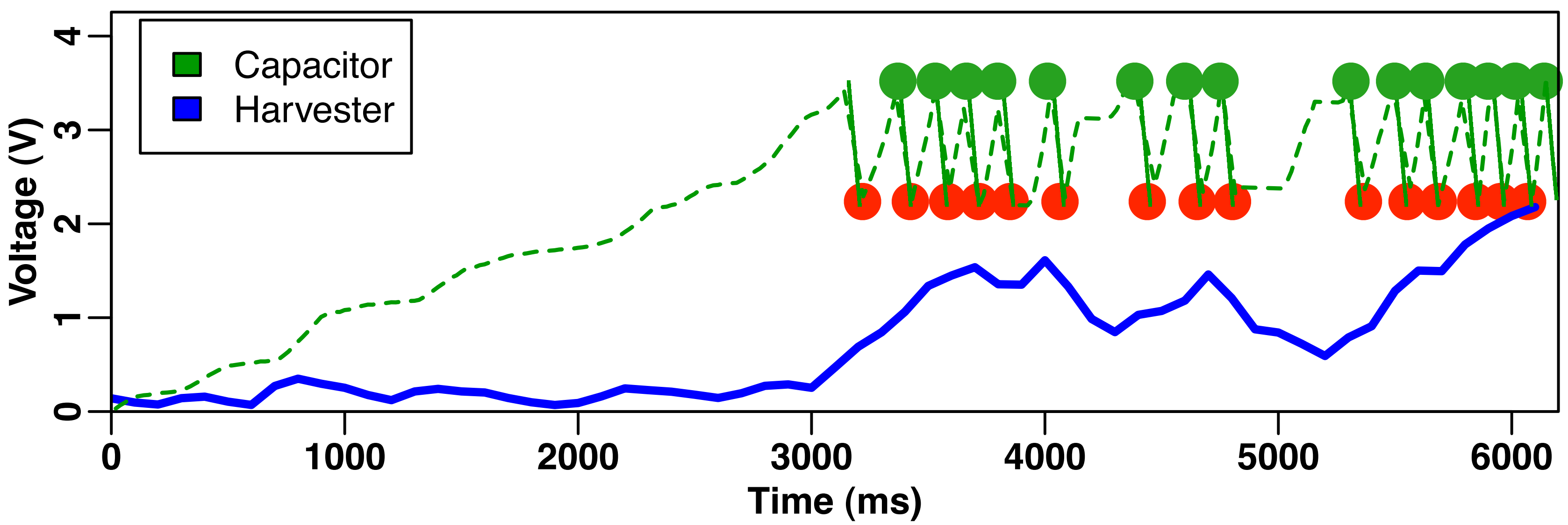
10x blowup is better than never finishing at all!

Uninstrumented, unlimited energy:	Oracle:	Best execution:
575,315 cycles	685,608 cycles	761,983 cycles
575 ms	~4,000 ms	6,145 ms
	14 reboots	16 reboots
	$V_{thresh} \geq 2.35$ V	$V_{thresh} = 2.6$ V

With and Without Mementos



CRC
w/o Mementos:
never finishes

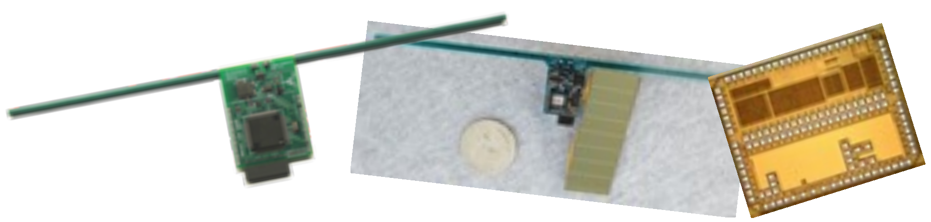


CRC
☺ w/ Mementos:
16 reboots

Oracle: 14 reboots

Related Work

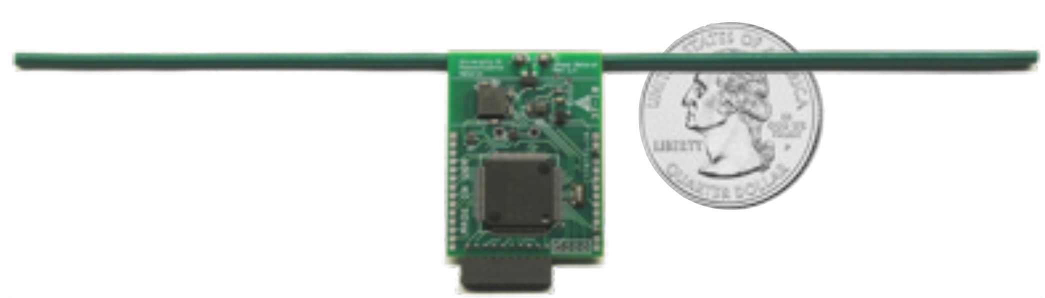
- **RFID-scale devices**
  - Mementos workshop paper [HotPower ’08]
  - Dewdrop scheduler for RFID-scale devices [NSDI 2011]
  - WISP [IEEE TIM ’08] and friends



- **Checkpointing**
  - Sensornet checkpointing [EWSN ’09]
  - Checkpointing for process migration (Condor [ICDCS ’88], Porch [IEEE Micro ’98])

Extensions

- Dynamic or randomized V_{thresh} adaptation
- NVRAM technology (PCM? FeRAM?)
- Smarter checkpointing (incremental, LVA...)
- Integrate with asynchronous communications on upcoming RFID-scale prototype (June ’11)



Mementos Conclusion

- Energy-aware checkpoints for computation on batteryless RFID-scale devices
- Tools available today; built on LLVM and MSPsim
- Applications: implantable devices, insect-scale tracking, infrastructure monitoring...

Acknowledgements

Your Homework

Get **Mementos**, simulator, hardware:
http://spqr.cs.umass.edu/mementos

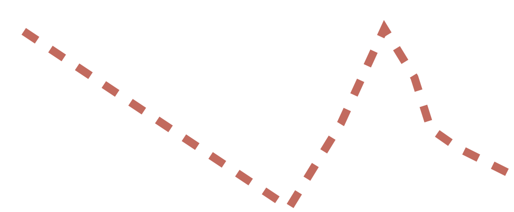
HW
- What should be moved off-chip to save energy?
- Right combo of RAM & NVRAM? Tiny off-chip NVRAM?
- HW/SW interface for detecting an impending failure?
- Conditional branches predicated on available energy?

OS
PL
- Task scheduling when failure is common case?
- Should we write a SuperTinyOS that expects failure?
- Compile-time optimizations for expected failure?
- How to not repeat non-idempotent actions?

Contingency Slides

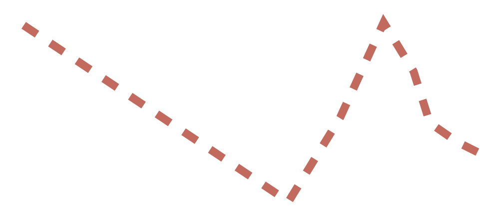
V_{thresh} Subtleties

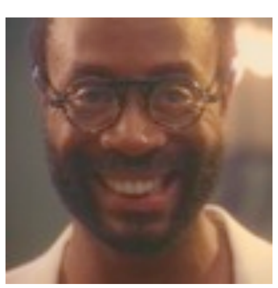
- Size, duration of CKPT are application dependent
- V_{thresh} is a conservative estimate
  - More energy might arrive
  - Choose according to risk tolerance



V_{thresh} Subtleties

- Size, duration of CKPT are application dependent
- V_{thresh} is a conservative estimate
  - More energy might arrive
  - Choose according to risk tolerance



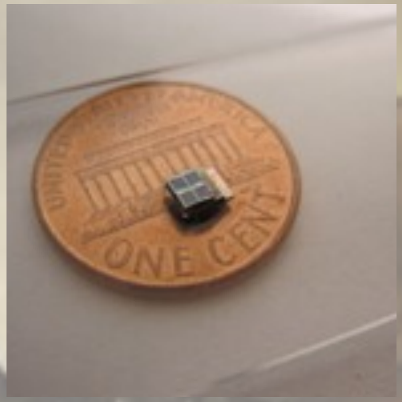


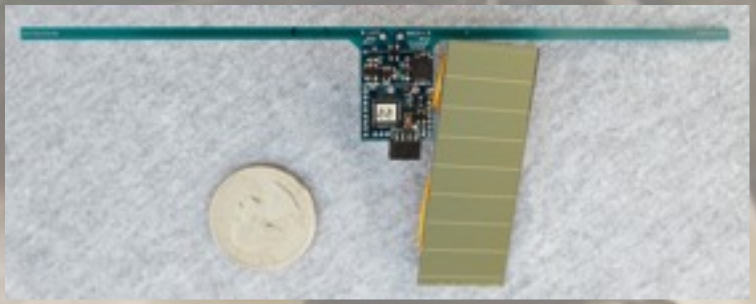
Spoiler: We help the programmer choose V_{thresh}

Low-Power Modes

- MSP430 has a variety of low-power modes (~1 μA in LPM3/LPM4) that retain RAM
- Sleeping when energy is low is an optimistic strategy
- We don't know when or whether energy will return — should Mementos guess?

Applications





Daeyeon Kim; Gyouho Kim; (SolarWISP) Shane Clark; Medtronic

Thanks for asking about TinyOS

- Condor, Porch, libckpt — all depend on OS-level or out-of-band facilities
- Sensor OSes (e.g., TinyOS) designed to boot *infrequently* and sip from batteries
  - TinyOS boot: $\geq$ 100 ms (too slow)

Thanks for asking about TinyOS

- Condor, Porch, libckpt — all depend on OS-level or out-of-band facilities
- Sensor OSes (e.g., TinyOS) designed to boot *infrequently* and sip from batteries
  - TinyOS boot: ≥ 100 ms (too slow)

Existing lightweight OSes *still* too heavy

CRC Example: Overhead

How much CPU overhead for checks?

Consistently high voltage ($V > V_{thresh}$):

Instrumentation	CPU Cycles	Mementos
Uninstrumented	575,315	0%
Loop latches	619,450	6.9%
Function returns	577,702	0.2%
Timer + latches	598,171	3.4%

Why Not Just Add...

- Thin-film battery?
- Deeper charge pump (higher voltage)?
- Tiny dedicated NVRAM?
- Hardware “low energy” interrupt support?

Other Use Cases

Programmer can:

- Disable instrumentation at a function level
- Manually call Mementos routines