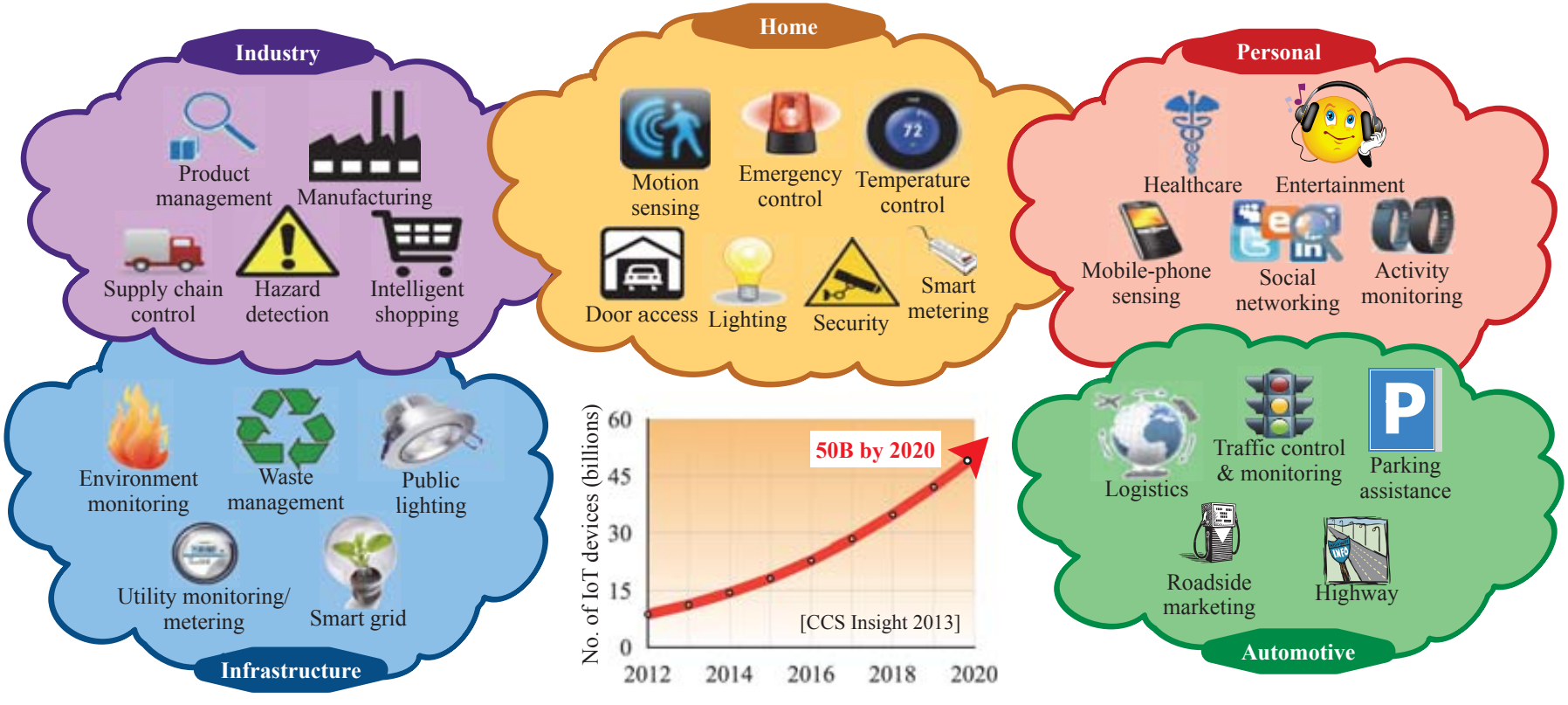
Peripheral State Persistence for Transiently Powered Systems

Tristan Delizy, Kevin Marquet, Guillaume Salagnac

LSC seminar, nov 2016

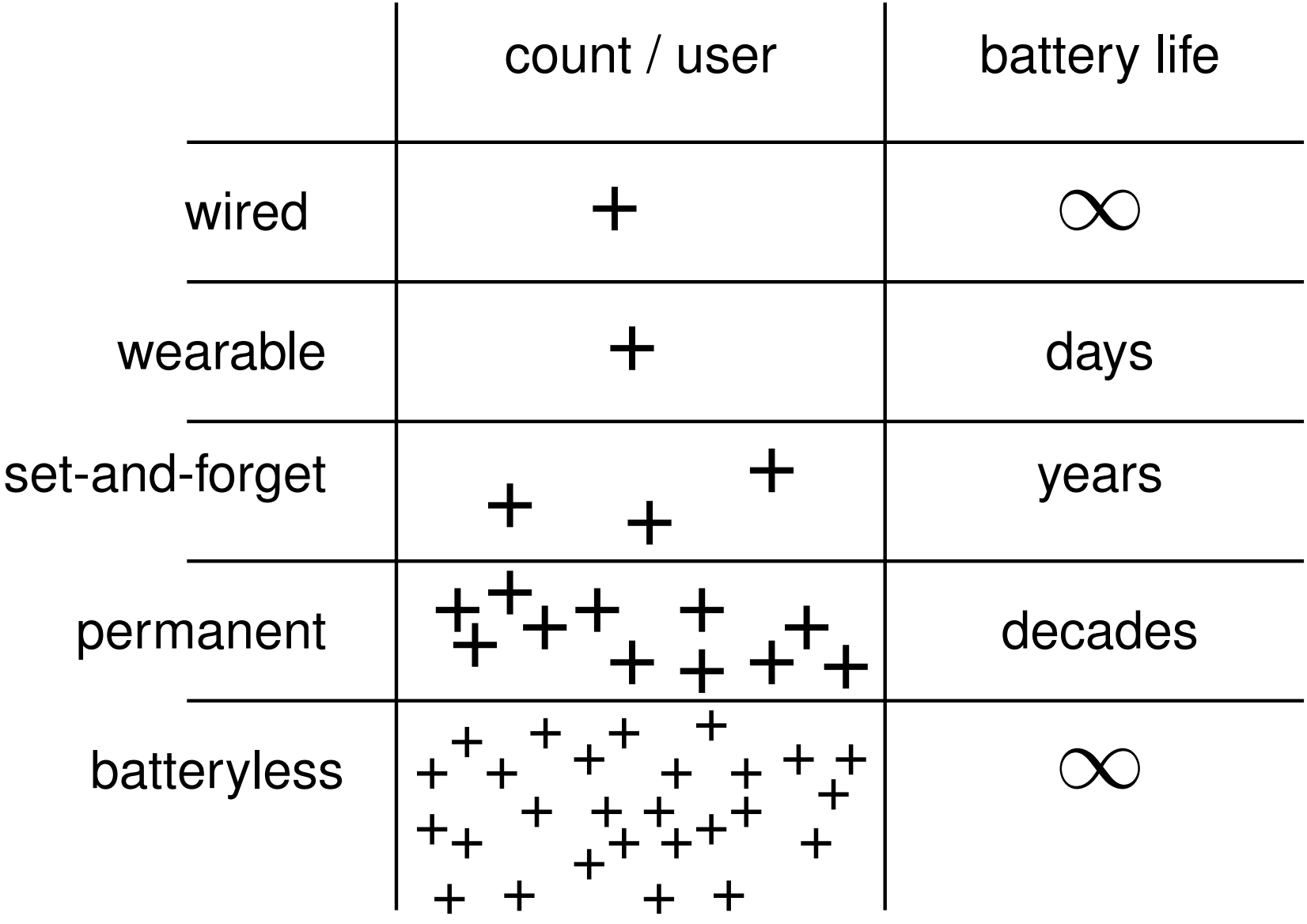
Context: the Internet of Things

IoT = Embedded Systems + Networking



Jayakumar, Lee, Lee, Raha, Kim, & Raghunathan. **Powering the Internet of Things**. In *ISPLED'14: International Symposium on Low Power Electronics and Design*, 2014

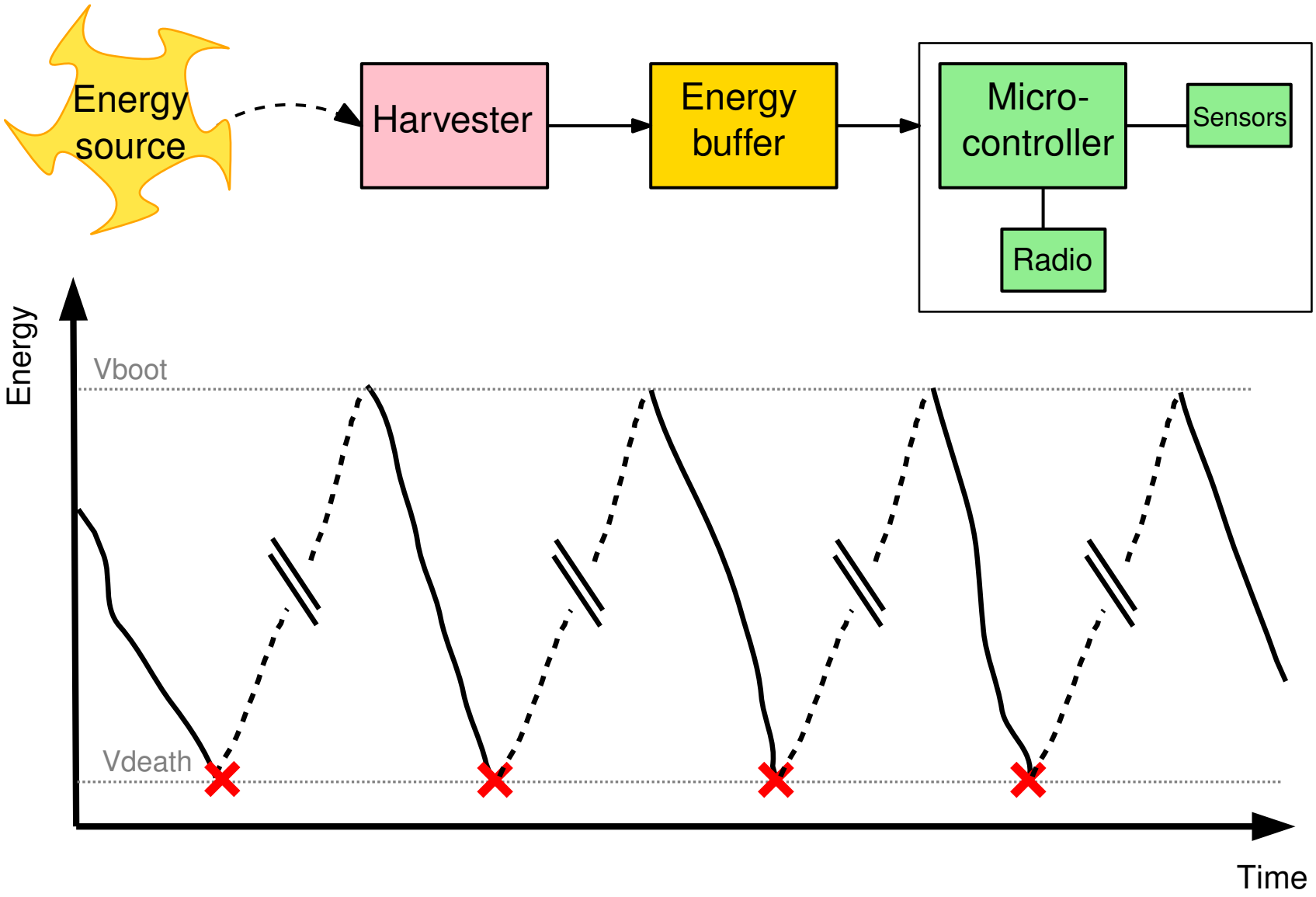
IoT: Energy constraints VS longevity requirements



Jayakumar, Lee, Lee, Raha, Kim, & Raghunathan. **Powering the Internet of Things**. In *ISPLED'14: International Symposium on Low Power Electronics and Design*, 2014

Outline

Transiently Powered Systems



Problem statement: how to run code despite constant reboots ?

Non-Volatile Random Access Memory

NVRAM aka Storage-Class Memory (SCM)

- retains data when not powered
- byte-addressable
- low latency / low power (vs Flash)

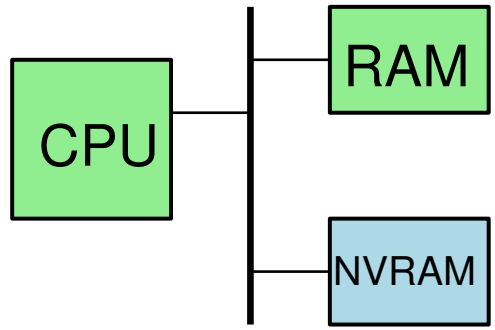
Many emerging technologies

- FeRAM, RRAM, MRAM, STT-RAM, CBRAM, PCM...

but no Universal Memory in sight (yet ?)

- problems: endurance, write latency / energy, bit errors...

- typical architecture = SRAM+NVRAM



The Broken Time Machine problem

Source program

```
NONVOLATILE int len = -1;
NONVOLATILE char buf[MAX];

void main(void){
  for(;;){
    append('a');
    ...
  }
}

void append(char c){
  register int reg = len;
  reg = reg + 1;
  len = reg;
  buf[len] = c;
}
```

Dynamic execution

```
main()
  append('a')
    reg = len;
    reg = reg + 1;
    len = reg;
    buf[len] = 'a';

main()
  append('a')
    reg = len;

main()
  append('a')
    reg = len;
    reg = reg + 1;
    len = reg;

...
```

Lucia & Ransford. **Nonvolatile Memory is a Broken Time Machine.** In *MSPC 2014: ACM SIGPLAN Workshop on Memory Systems Performance and Correctness*, 2014

The Broken Time Machine problem

Source program

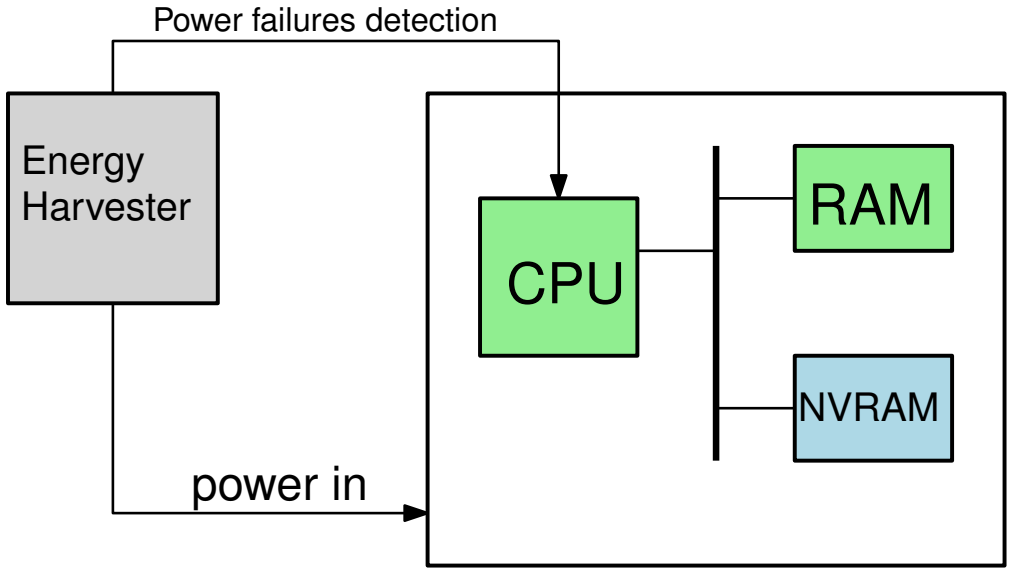
```
NONVOLATILE int len = -1;
NONVOLATILE char buf[MAX];

void main(void){
  for(;;){
    append('a');
    ...
  }
}

void append(char c){
  register int reg = len;
  reg = reg + 1;
  len = reg;
  buf[len] = c;
}
```

Dynamic execution

```
main()
  append('a')
    reg = len;
    reg = reg + 1;
    len = reg;
    buf[len] = 'a';
power failure + reboot
main()
  append('a')
    reg = len;
power failure + reboot
main()
  append('a')
    reg = len;
    reg = reg + 1;
    len = reg;
power failure + reboot
...
```

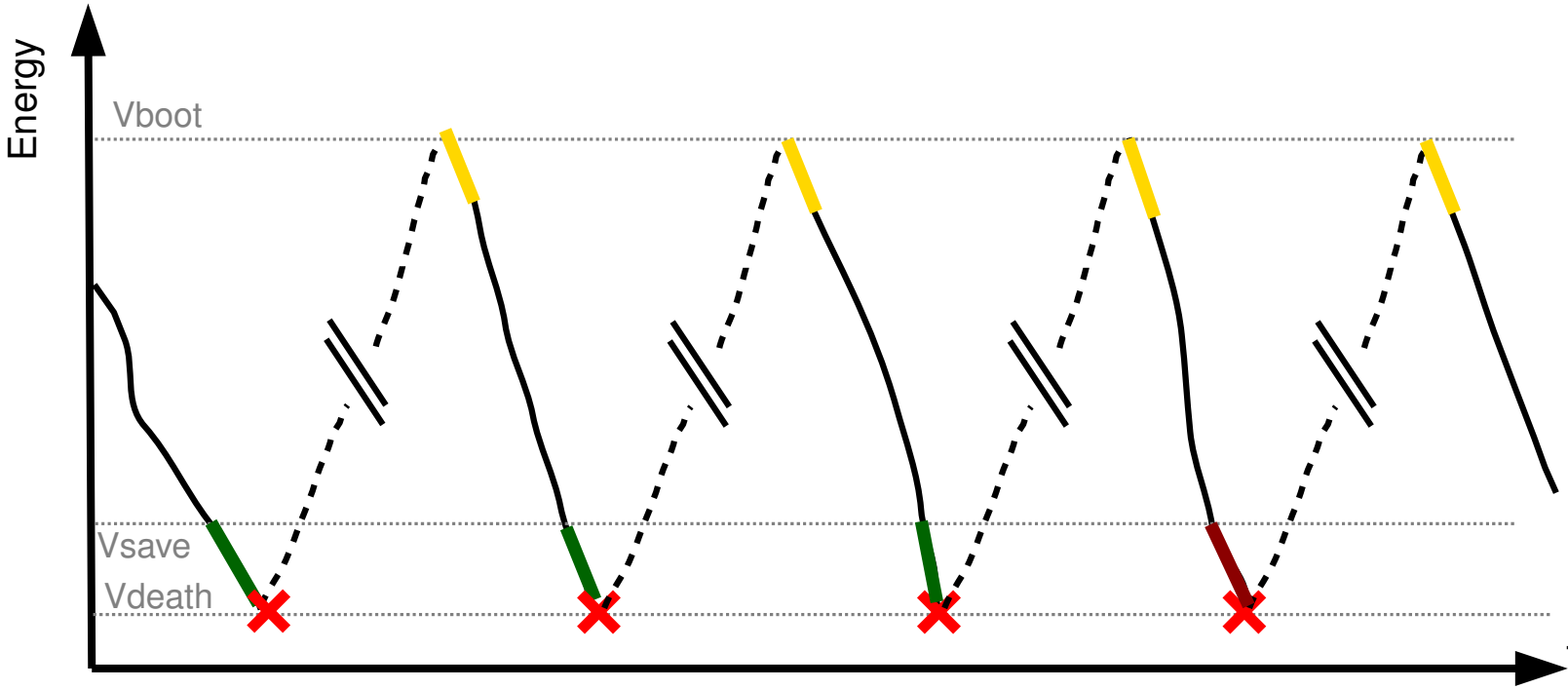
Lucia & Ransford. **Nonvolatile Memory is a Broken Time Machine.** In *MSPC 2014: ACM SIGPLAN Workshop on Memory Systems Performance and Correctness*, 2014

Classical solution: program checkpointing



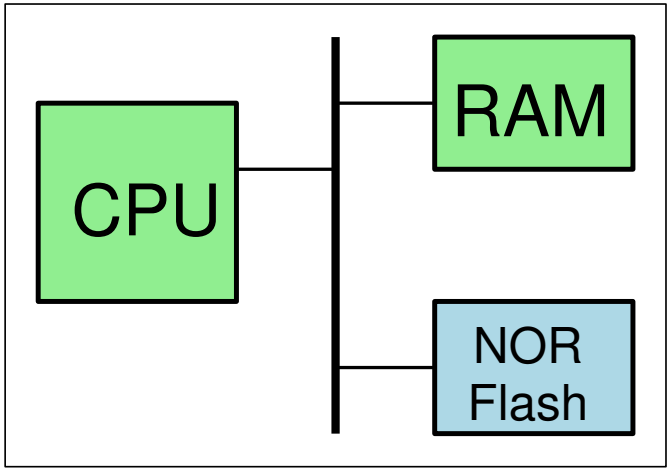
Principles:

- Keep execution volatile
- Anticipate power failures
- Save state to non-volatile memory
- Restore state at next boot



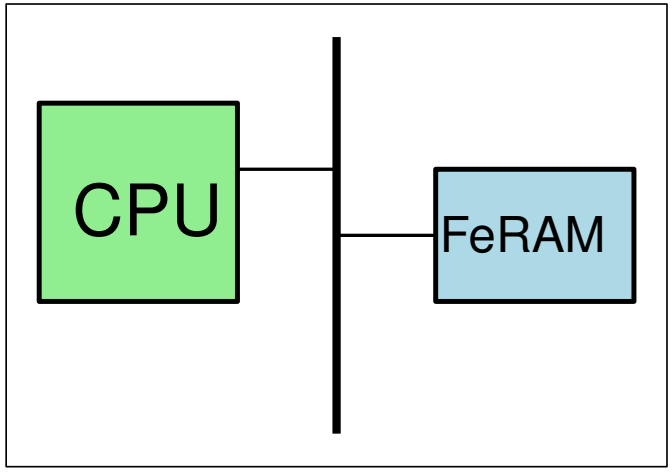
Program checkpointing for TPS

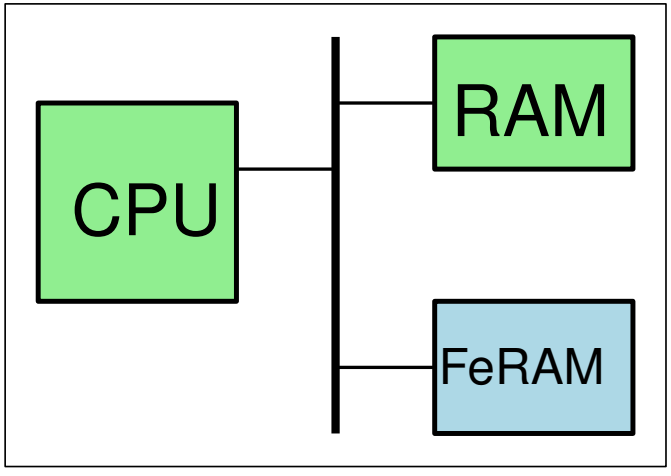
[Ransford *et al* ’13]



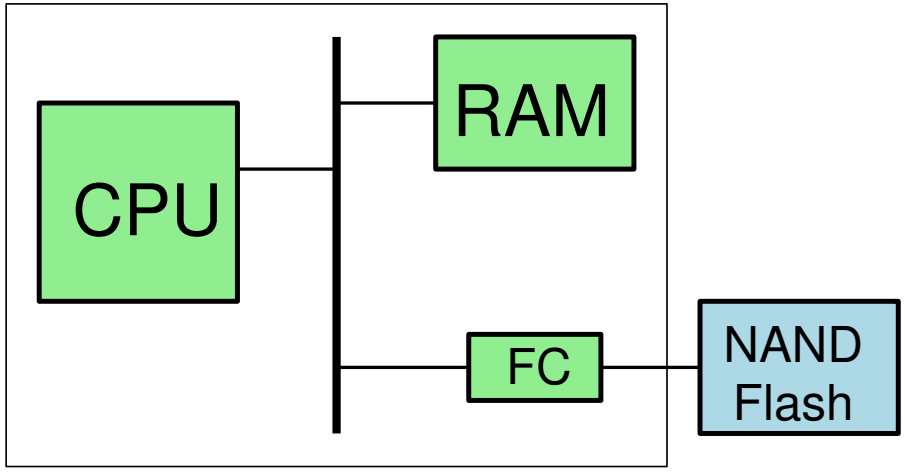
[Jayakumar *et al* ’14]

[Lucia & Ransford ’15]





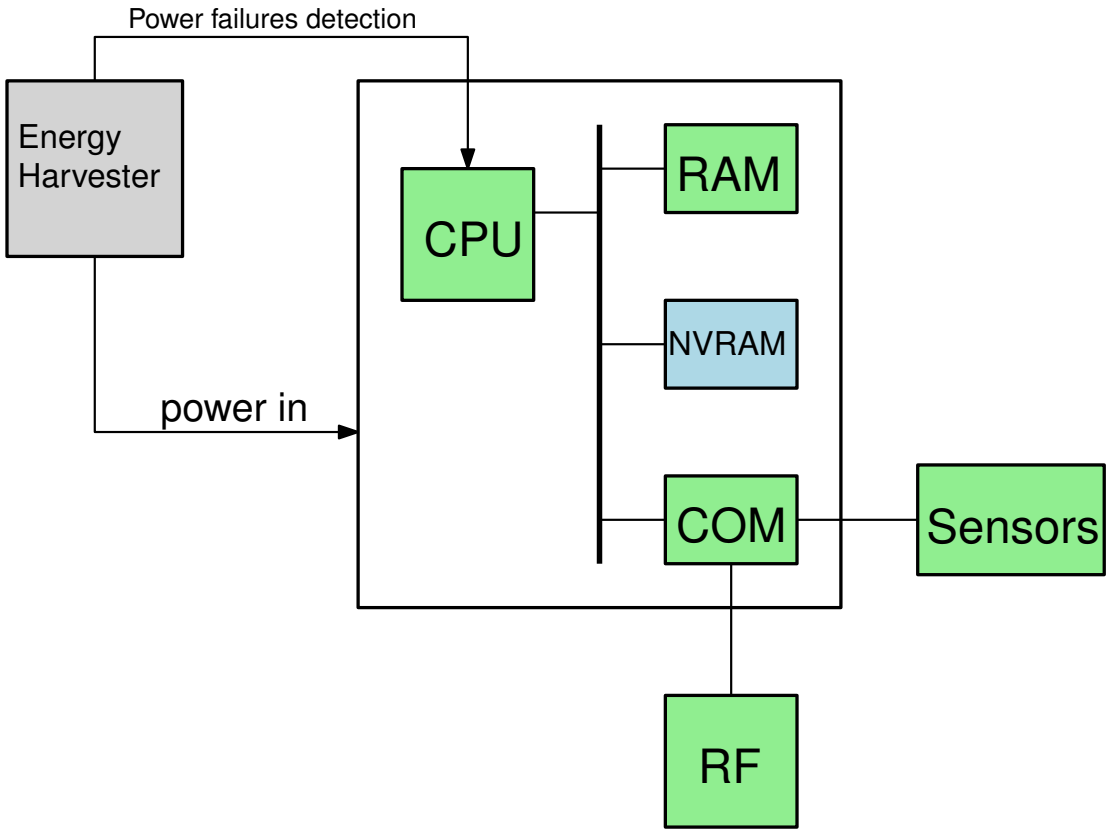
[Ait Aoudia *et al* ’14]



[Bhatti & Mottola ’16]

Outline

Making peripherals persistent too ?



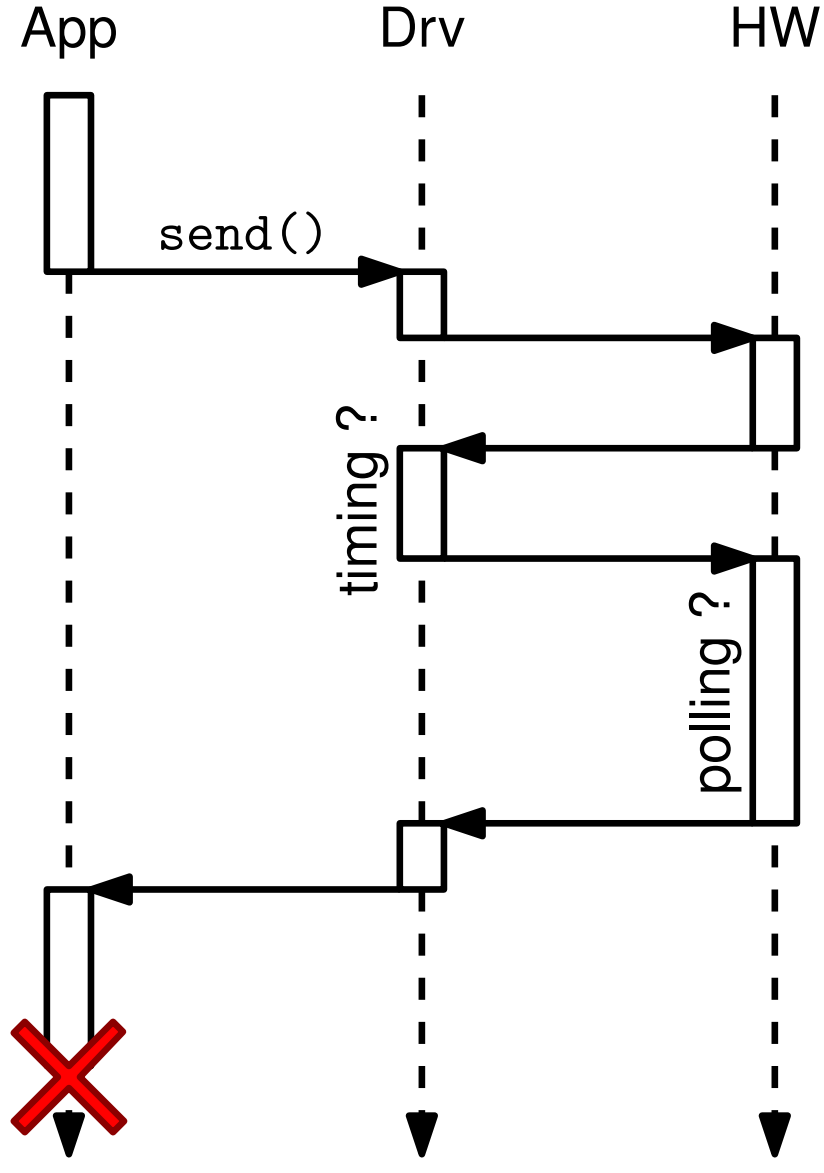
Program checkpointing is not enough:

- indirect access
- hardware initialization procedures
- atomicity of each access

The Peripheral State Volatility Problem

Source program

```
void main(void){
   sensor_init();
   radio_init(myconfig);

   for(;;){
     v = sensor_read();
     radio_send(v);
     ...
  }
}
```

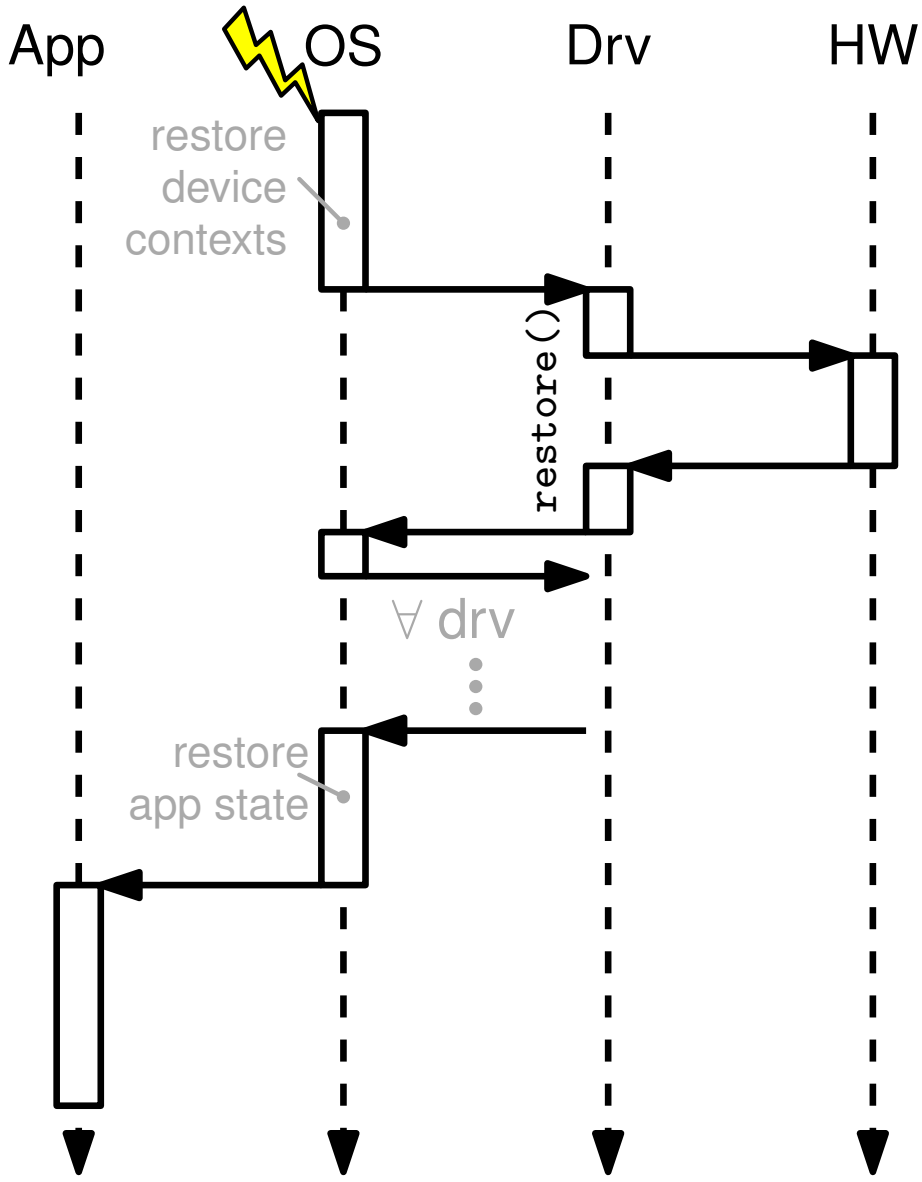


Problem: reloading memory will not restore device state

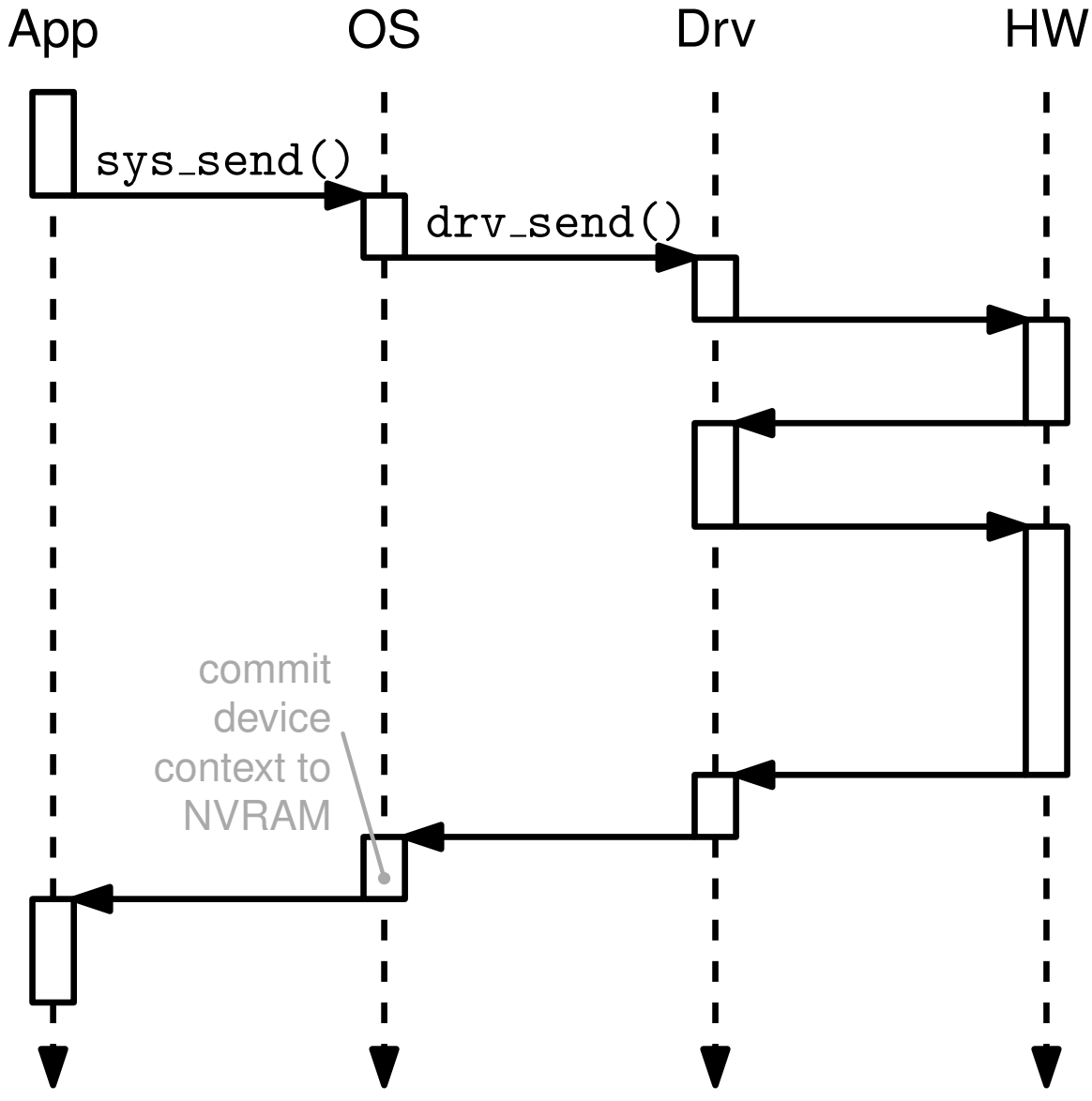
Our approach (some refactoring required)

In each driver:

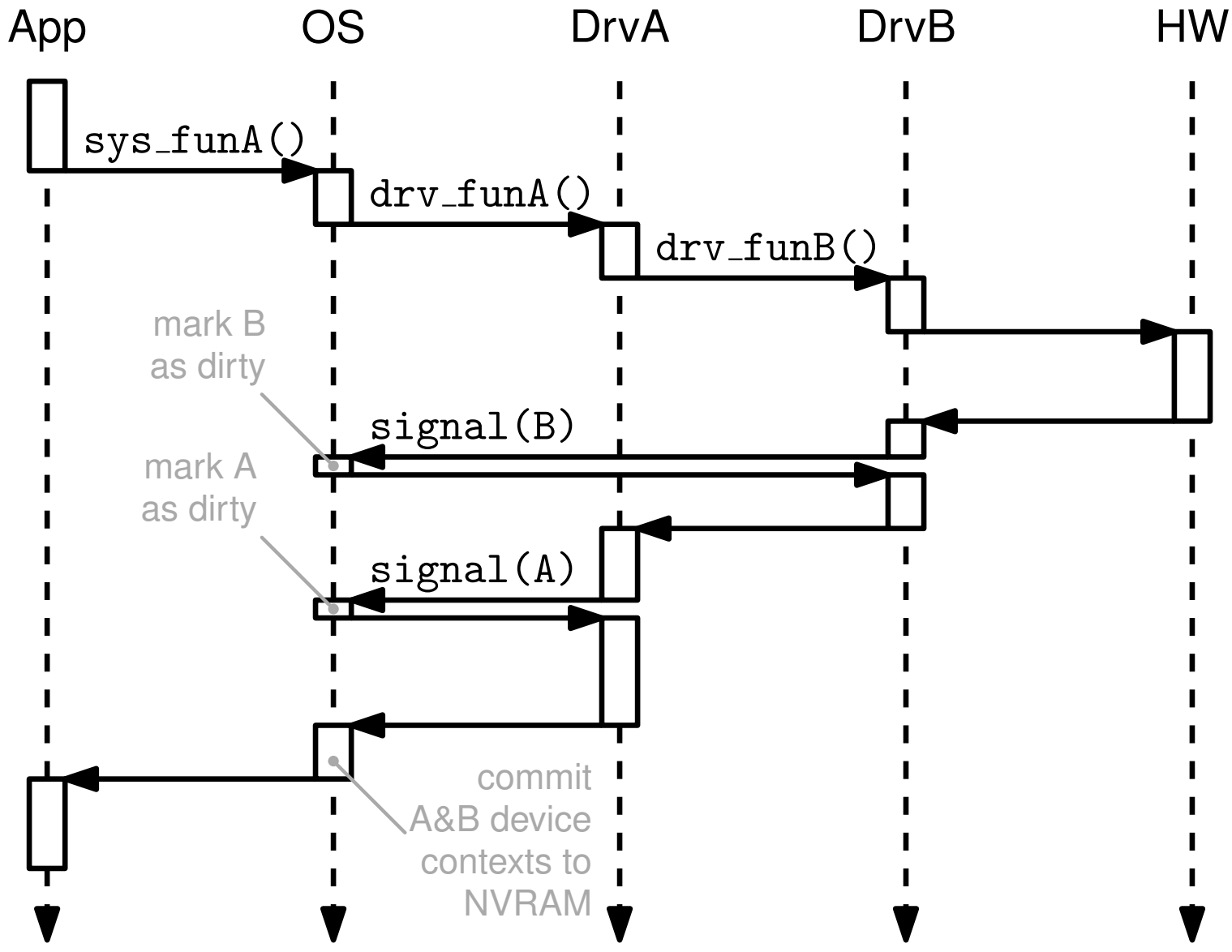
- add a `restore()` method
  - similar to `init()` + maybe a switch/case
  - makes use of already existing functions
- maintain a device context
  - describing some “restore()-able” state
  - will be included in checkpoint image



When should device contexts be saved to NVRAM ?



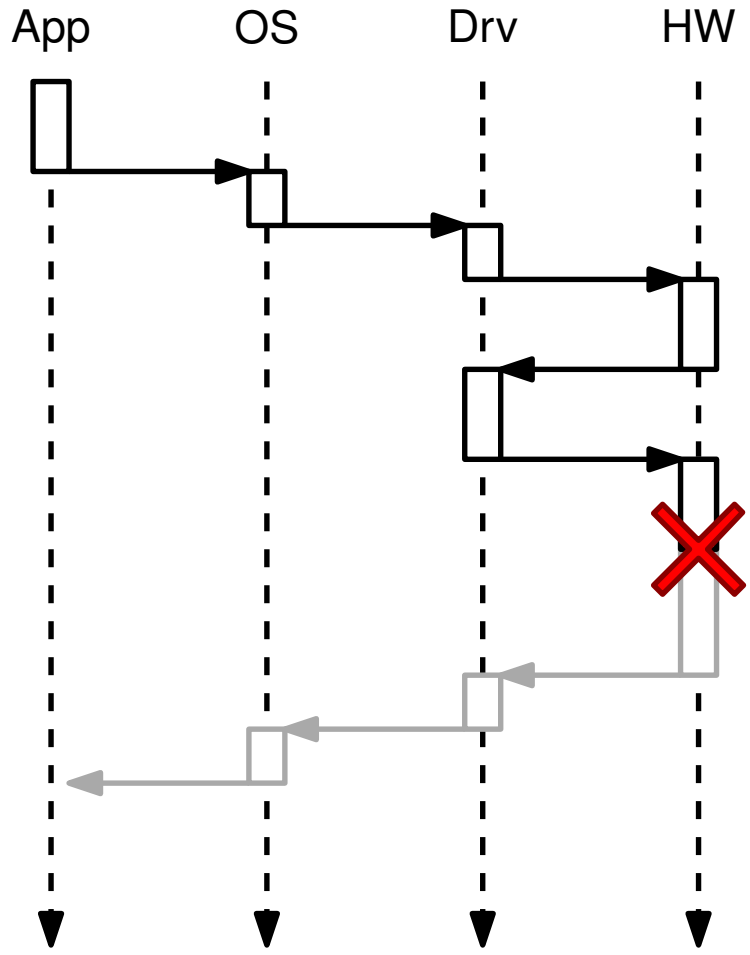
Capturing state of nested drivers



The Peripheral Access Atomicity Problem

Source program

```
void main(void){
   sensor_init();
   radio_init(myconfig);

   for(;;){
     v = sensor_read();
     radio_send(v);
     ...
  }
}
```



Problem: resuming execution in the middle of a hardware access does not always make sense

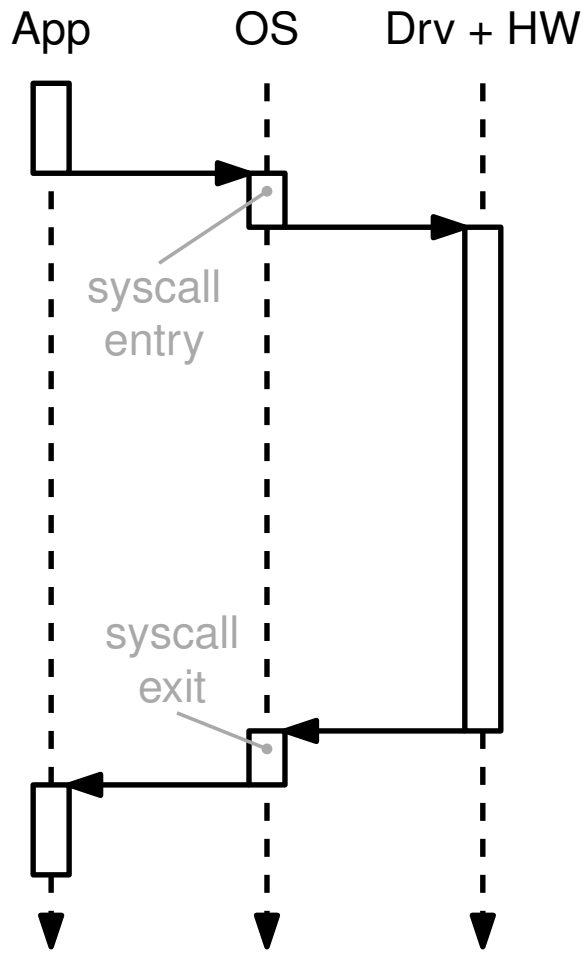
Our approach: make "syscalls" atomic

On syscall entry:

- backup arguments + syscall id
- switch to auxiliary volatile stack

On syscall exit:

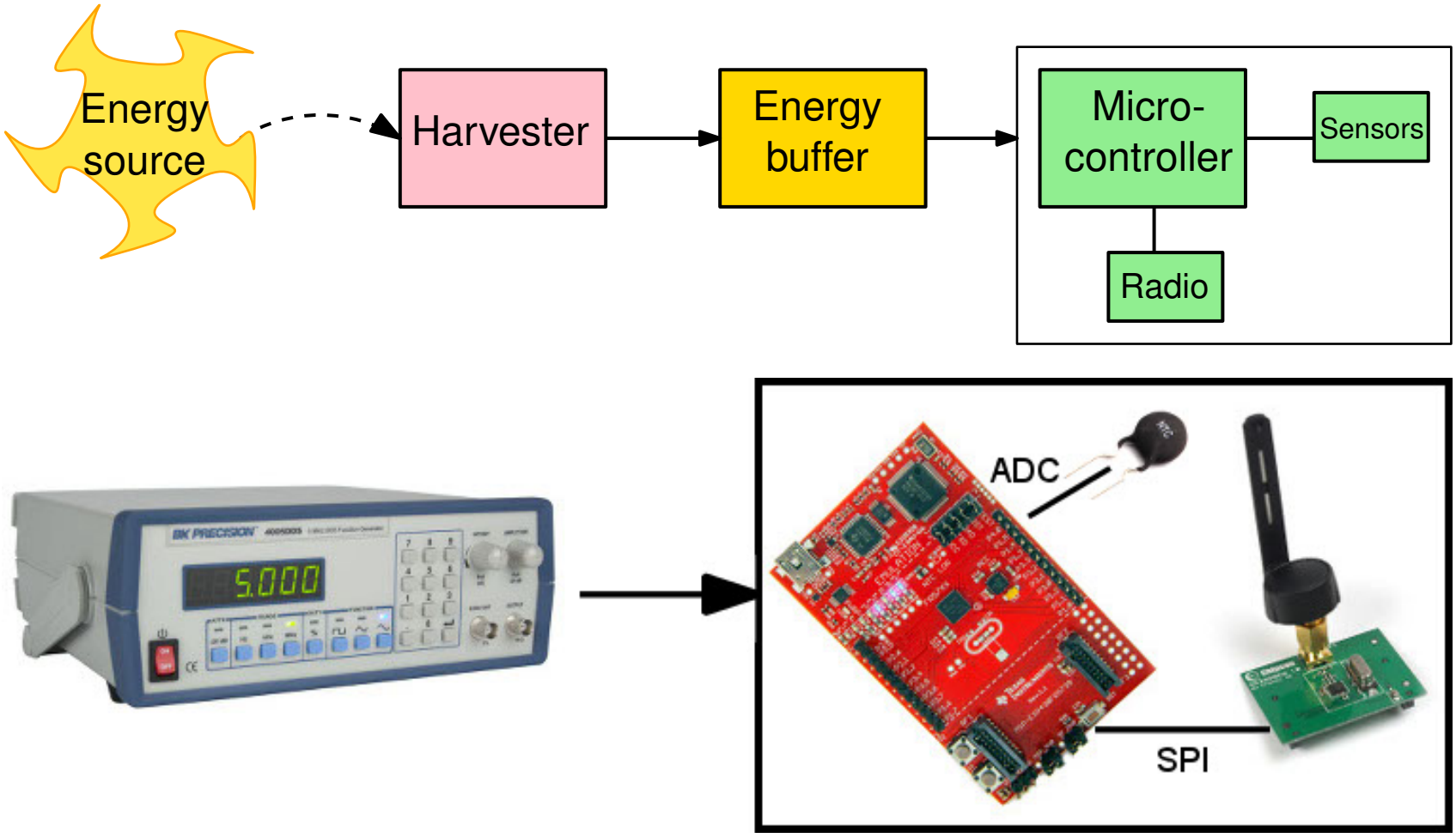
- clear arguments + syscall id
- commit device contexts
- switch back to main stack



- Interrupted syscalls get retried and not just resumed.

Outline

Prototype Implementation



- MSP430RF5739: 16-bit CPU 24MHz, 1kB SRAM, 15kB FRAM 8MHz
- CC2500: 2.4 GHz transciever, 64B packets

Evaluation

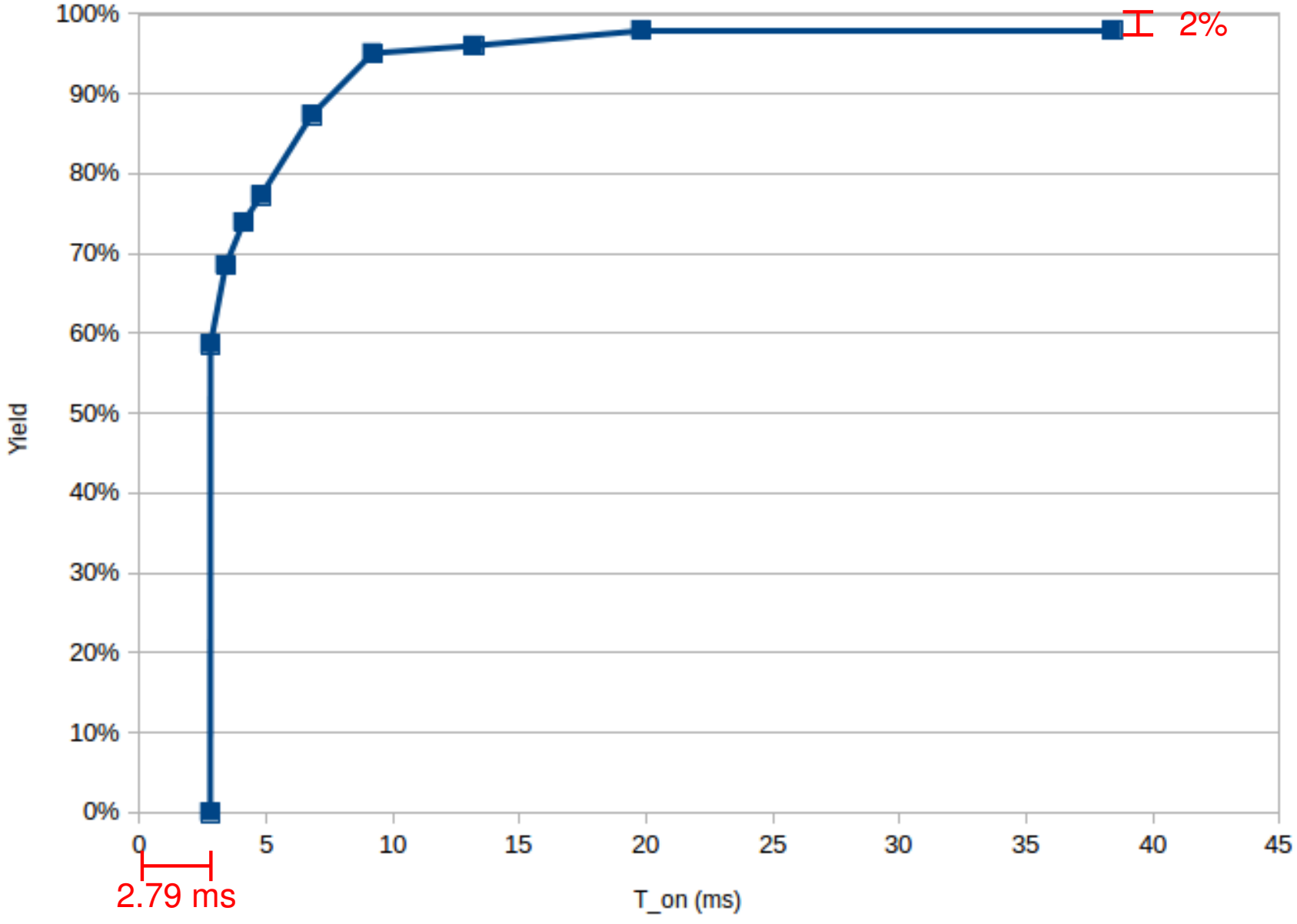
Benchmark programs

- RSA encryption
- Diode counter
- Sense and aggregate
- Sense and send

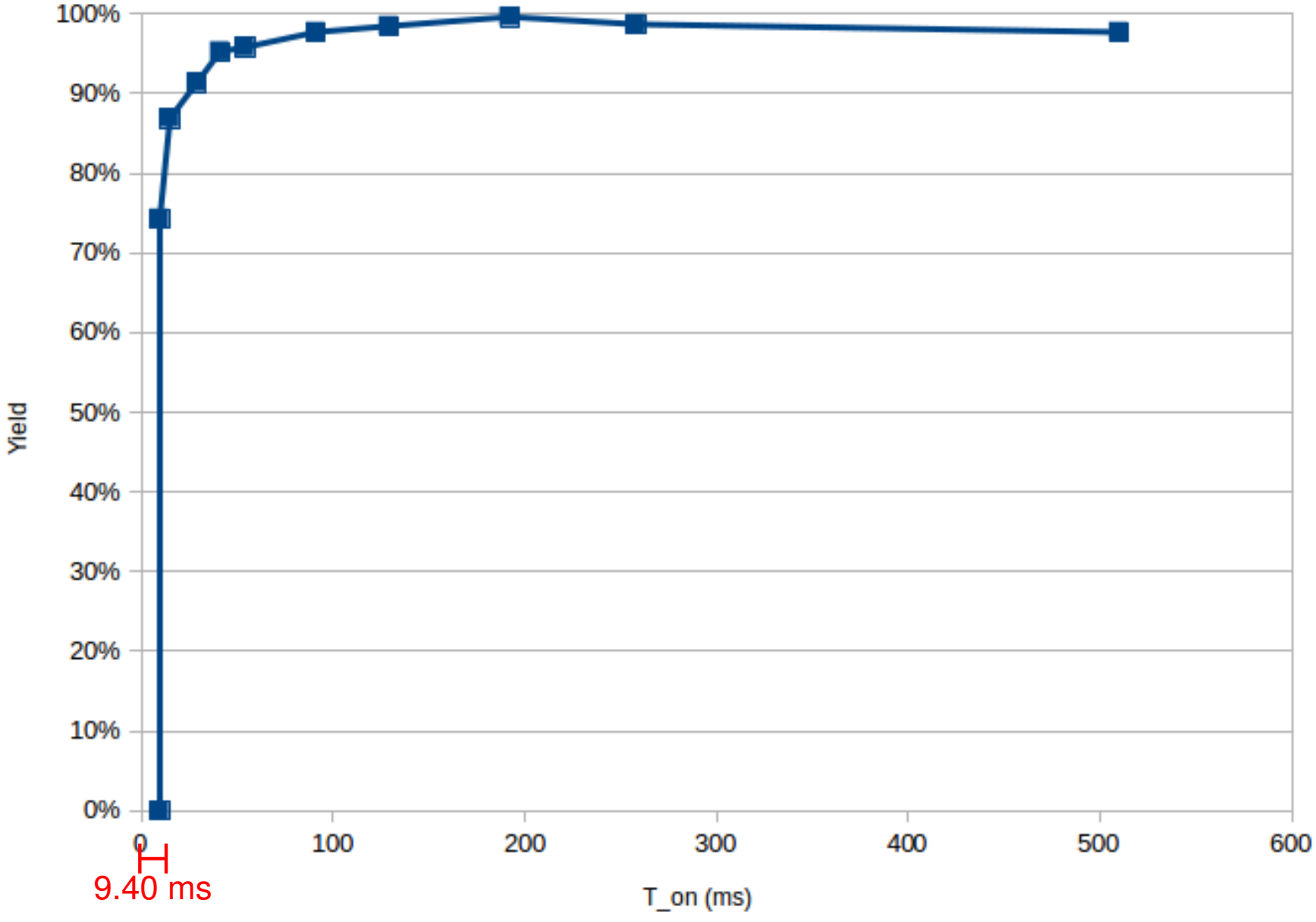
Evaluation metrics

- Duration of shortest usable lifecycle
- Execution overhead

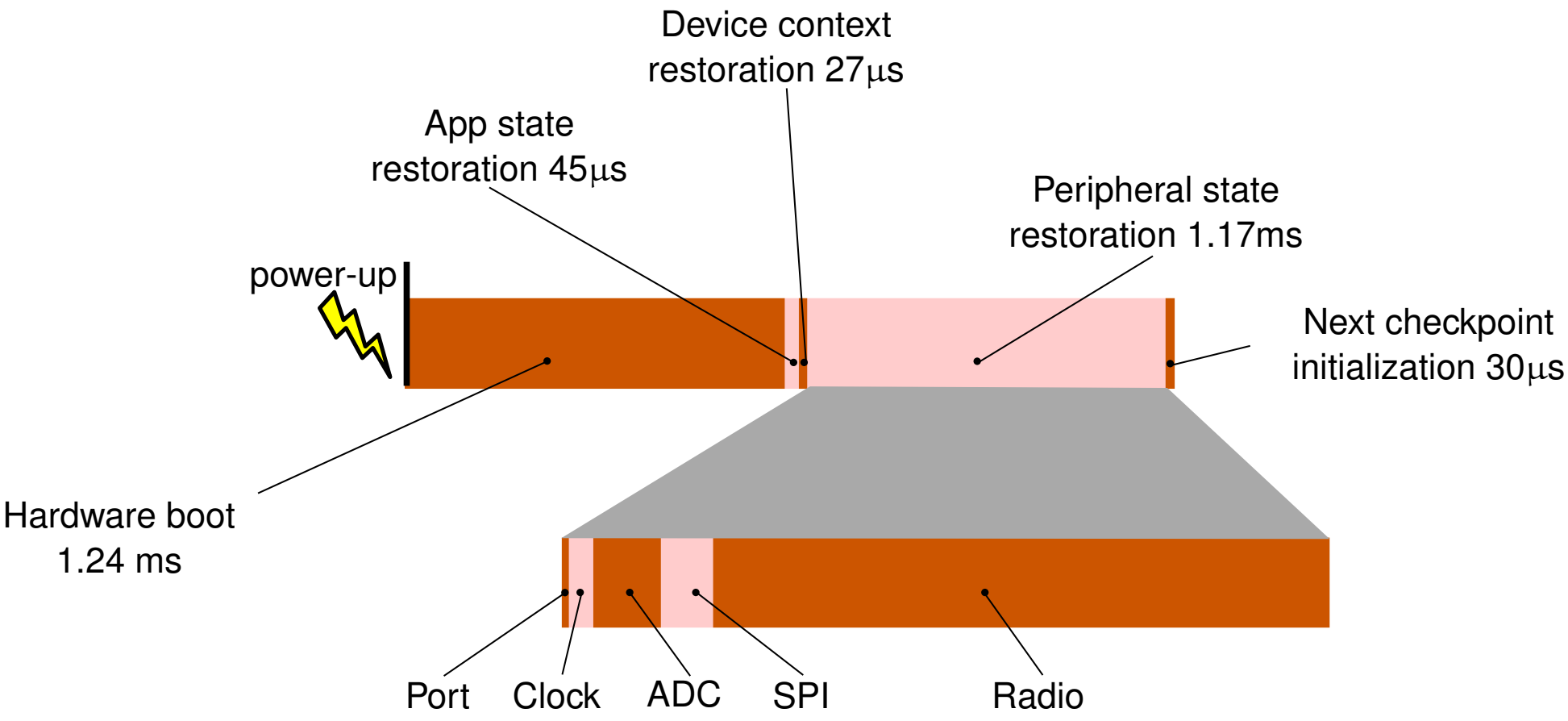
Results: RSA Encryption



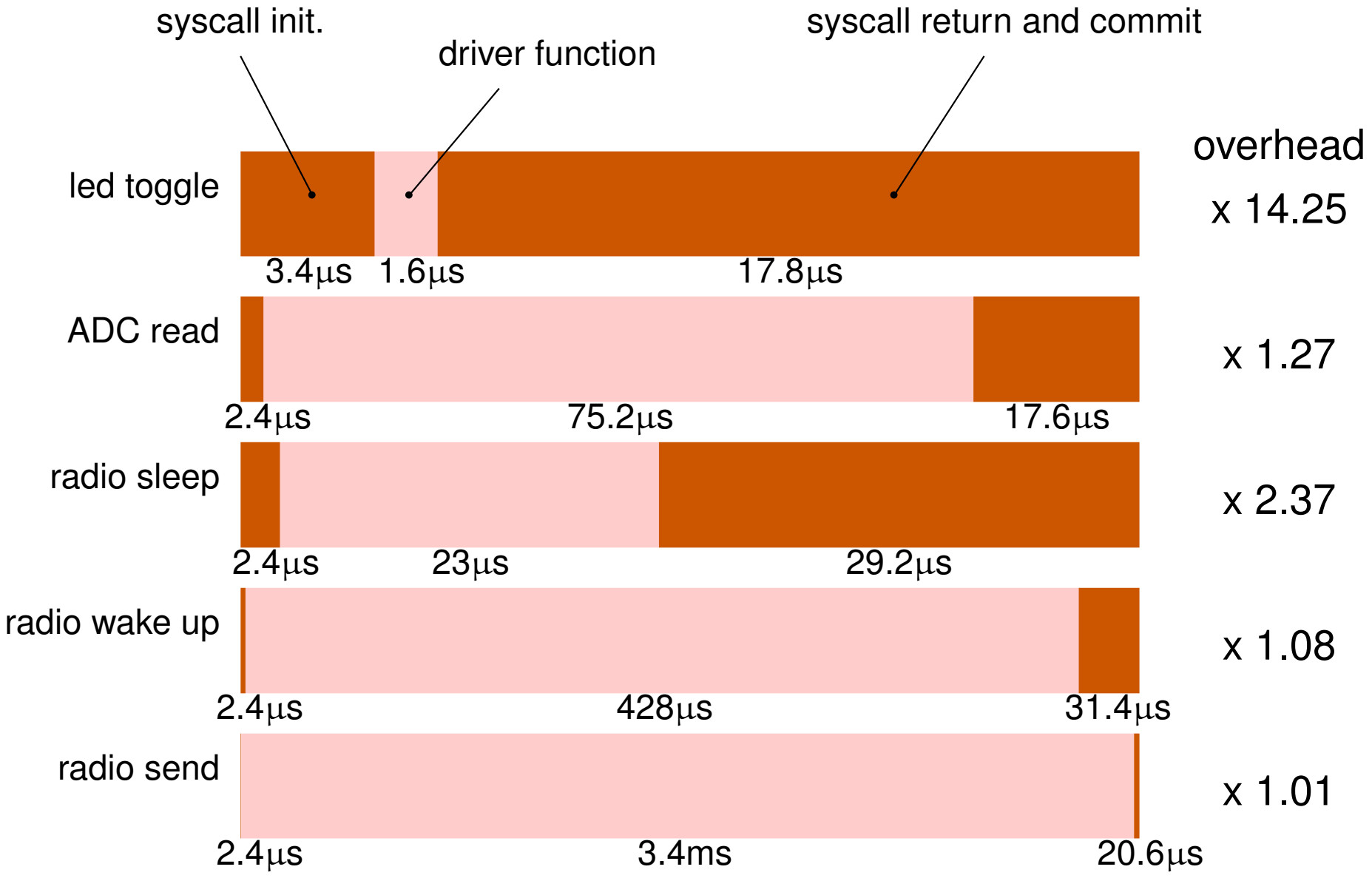
Results: Sense and Send



Results: detail of the boot sequence



Results: detail of syscall overhead



Outline

Conclusion and Perspectives

Transiently Powered Systems with NVRAM

- Broken Time Machine problem ▶ use checkpointing
- but checkpointing doesn't work for peripherals

Peripheral State Persistence

(with some help from the driver)

- Volatility: device contexts + `restore()` methods
- Atomicity of "syscalls": retry VS resume

Future Work

- programming abstractions for transient power ?
  - passing of time ; networking ; interrupts
- look at other combinations of hypotheses
  - e.g. battery+NVRAM
  - e.g. managed runtime

Merci de votre attention

Questions ?

Backup Slides

Embedded Systems

ST23ZL48 microcontroller

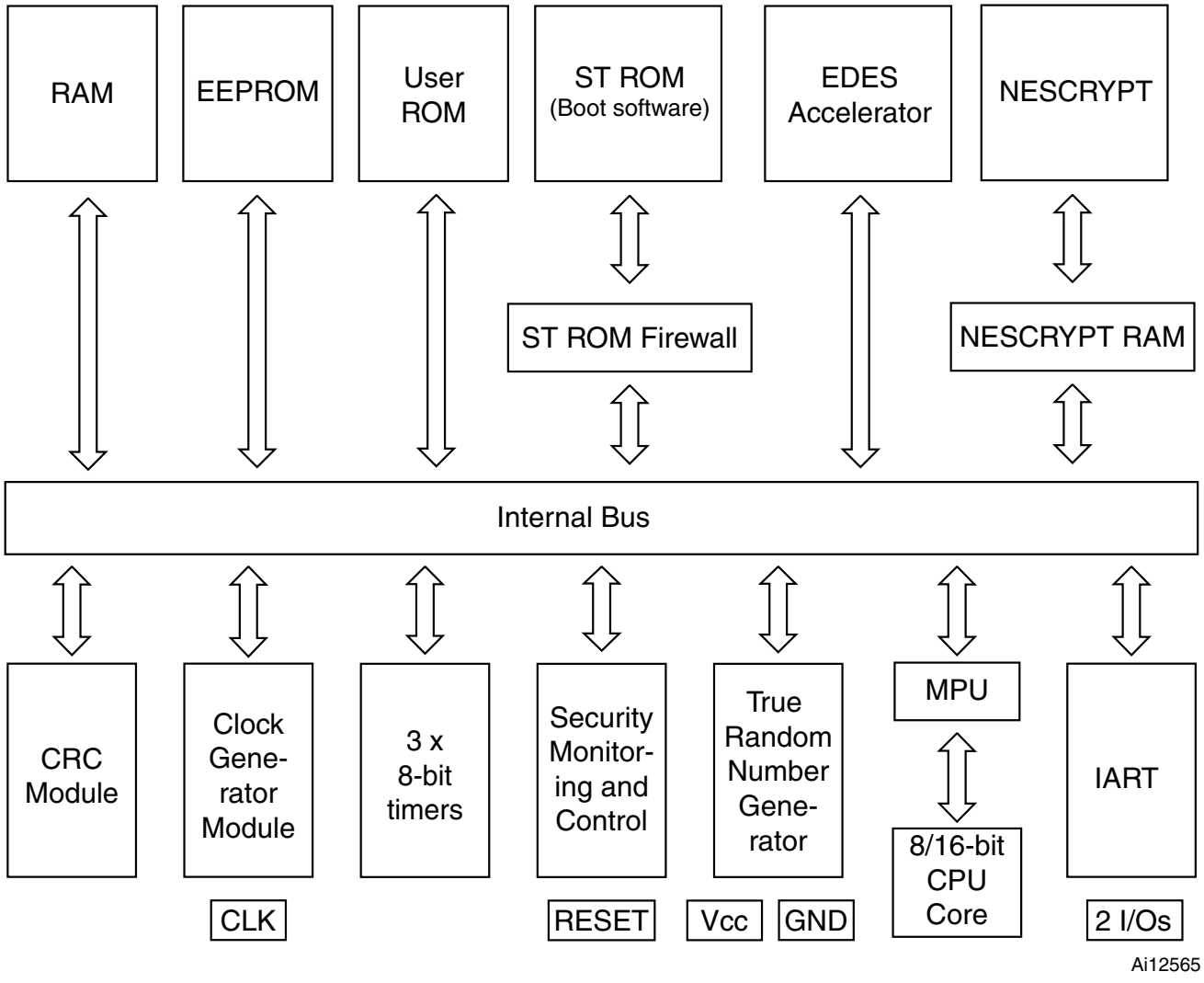
Energy Harvesting

Typical power ranges of ambient sources

Non-Volatile Memory

Comparison of NVRAM technologies

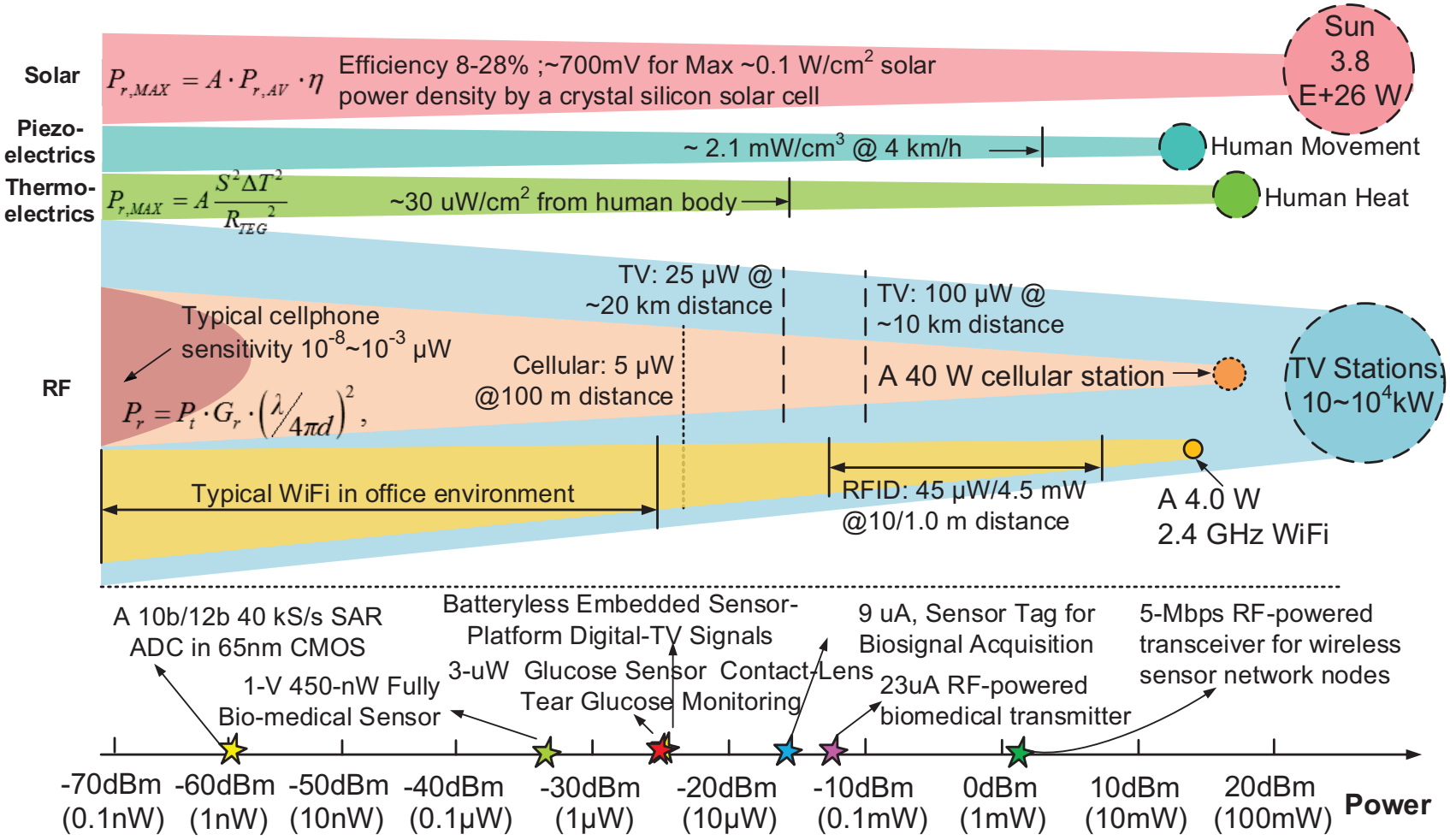
ST23ZL48 microcontroller



- 16-bits CPU (27MHz)
- 8kB RAM
- 300kB ROM
- 48kB EEPROM

http://www.st.com/en/secure-mcus/st23zl48.html

Typical power ranges of ambient sources



Ma, Zheng, Li, Swaminathan, Li, Liu, Sampson, Xie, & Narayanan. **Architecture exploration for ambient energy harvesting nonvolatile processors**. In *HPCA'15: High Performance Computer Architecture*, 2015

Comparison of NVRAM technologies

	SRAM	FLASH	PCM	MRAM	RRAM	FRAM
Maturity	++	++	+	–	– –	++
Density	– –	++	++	–	+	n/a
Scalability	++	+	++	+ / –	+	– –
Endurance	++	– –	–	+ / –	– –	++
Leakage power	– –	++	++	++	++	++
Read latency	++	–	+	+	+	–
Read energy	++	– –	++	++	++	–
Write latency	++	– –	– –	+ / –	+	–
Write energy	++	– –	–	–	–	–