# The Today and Future of Intermittent Computing for Sustainable Sensing

## Arm Research Summit 2020

Kasım Sinan Yıldırım
Assistant Professor
Department of Information Engineering and Computer Science,
University of Trento, Italy


UNIVERSITY
OF TRENTO - Italy

# Acknowledgements & Credits

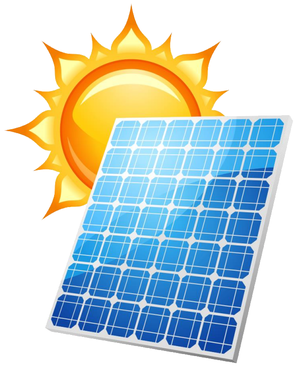

UNIVERSITY
OF TRENTO - Italy




Northwestern
University

# Transiently-Powered Computers

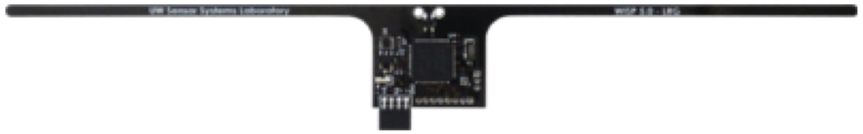
- Future sensing devices are tiny, sustainable and run forever!



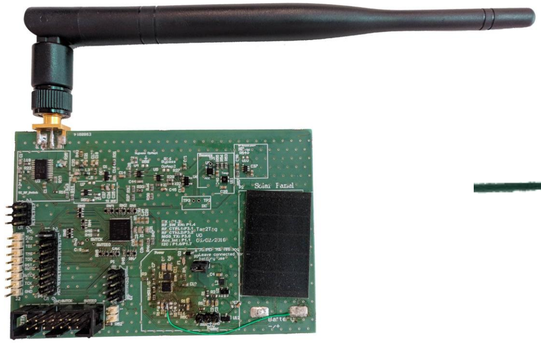
Solar

Radio Frequency

*Ambient Energy Sources*



WISP [Buettner et al., ACM SenSys'08]



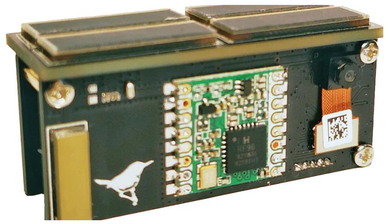
T2T [Majid et al., INFOCOM'18]

Flicker [Hester et al., ACM SenSys'17]

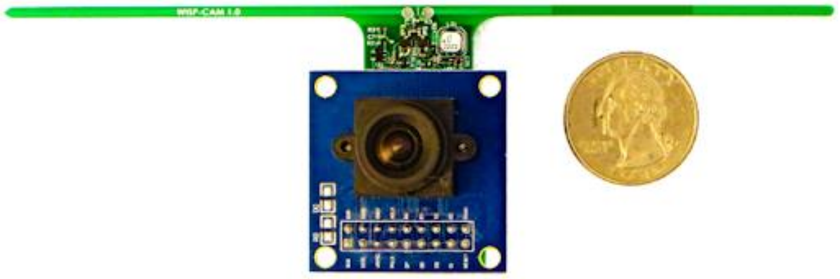
*Batteryless Devices*

# Prospective Applications

- Deploy and forget



Camaroptera
(Remote Image Sensor)
[Nardello et al., ACM EnsSys'19]



WISPCam
[Naderiparizi et al., EnsSys'16]

Tiny Batteryless Robot
[Yildirim et al., ACM SenSys'18]

**Infrastructure**
Pipelines,
Bridges
Roads

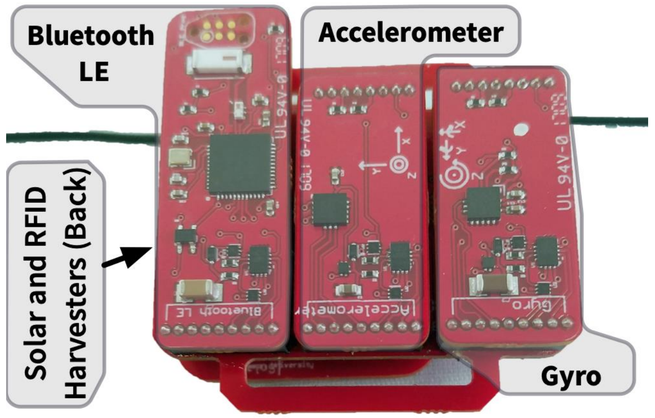
**Wearables**
Clothing
Jewellery
Implants

**Buildings**
Occupancy
Energy Monitor

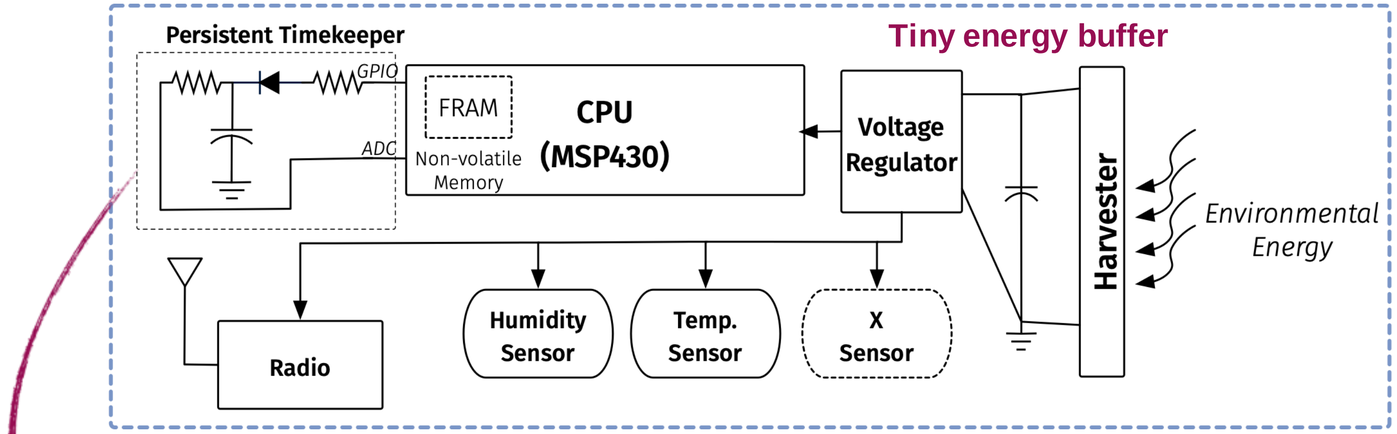
**Wildlife Tracking**
Small animal
Endangered

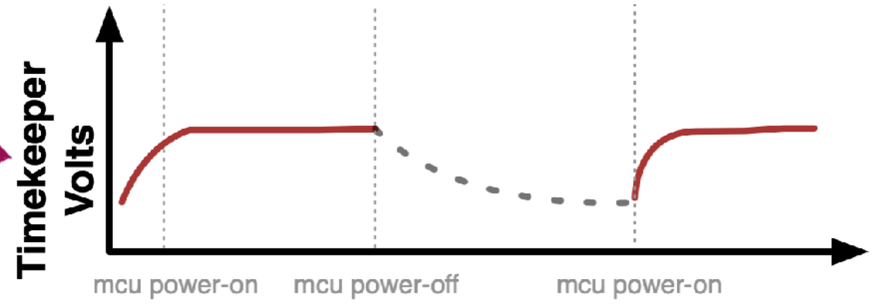
**Extreme Locations**
Deep Sea, High Altitude, Space

# A Typical Batteryless Hardware Architecture




Flicker [Hester et al., ACM Sensys'17]

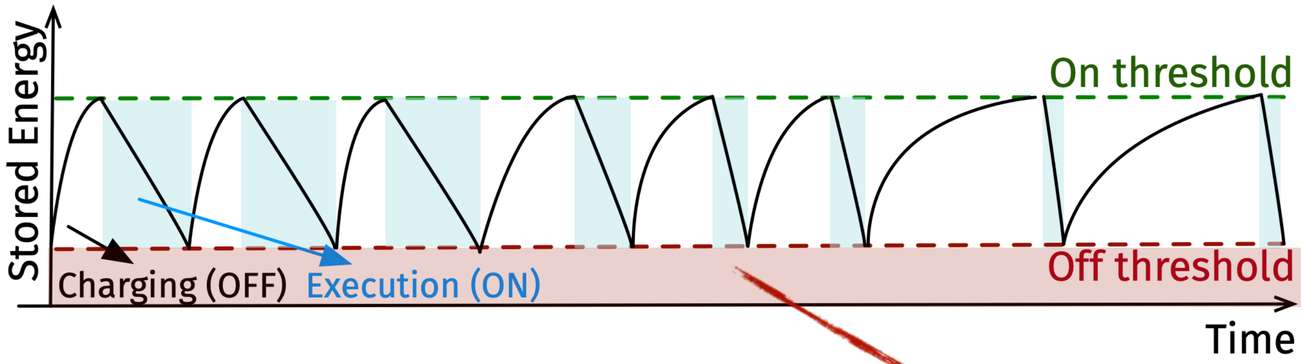


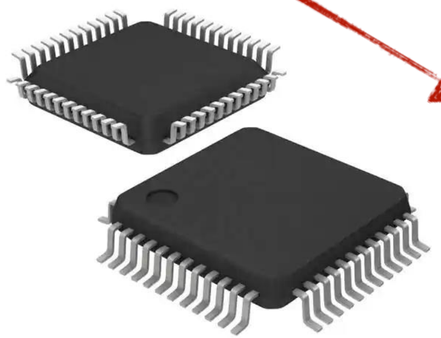




**Capacitor as Hourglass**

# Intermittent Operation

- Charge/Operate/Die cycle
  - Progress of computation?



**Volatile State is lost**

Registers
Memory (SRAM, DRAM, …)
Timers, I/O state…

```
int sz=0;
char buf[10];
main() {
 while(1)
  for(i=0..9)
   sz++
   buf[sz]=‘a’
    sleep()
}
```

```
main()
  while(1)
    for(i=0..9)
```

**Reboot**

```
main()
  while(1)
    for(i=0..9)
      sz++
```

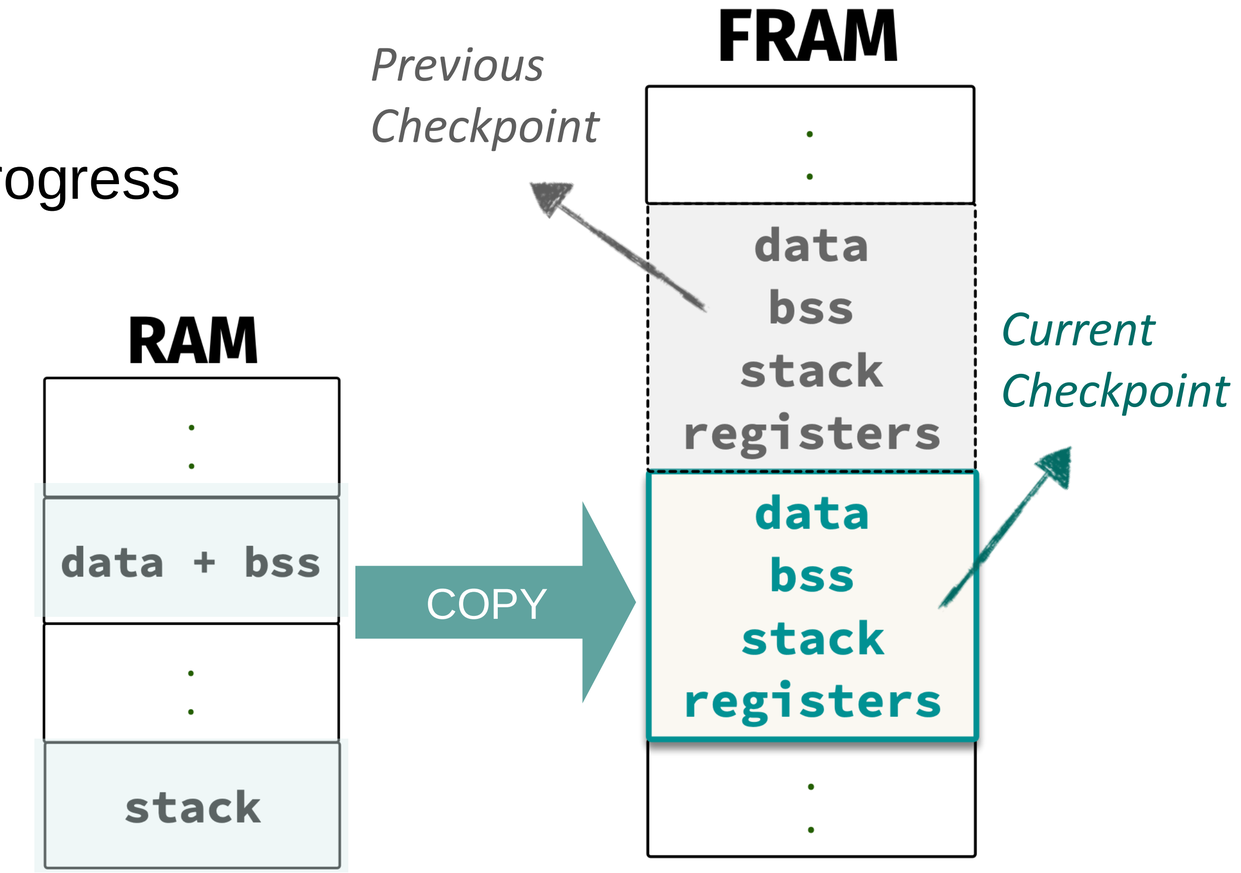
**Reboot**

```
main()
  while(1)
    for(i=0..9)
```

# Checkpoints

- Use NV memory to capture **volatile state**.
  - e.g., MementOS [Ransford et al., ASPLOS'11]
  - Ensures memory consistency/forward progress

```
1   int sz=0;
2   char buf[10];
3   main() {
4     while(1){
5       for(i=0..9){   Checkpoint
6           sz++
7           buf[sz]='a'
8       }
9       sleep()
10    }
11  }
```



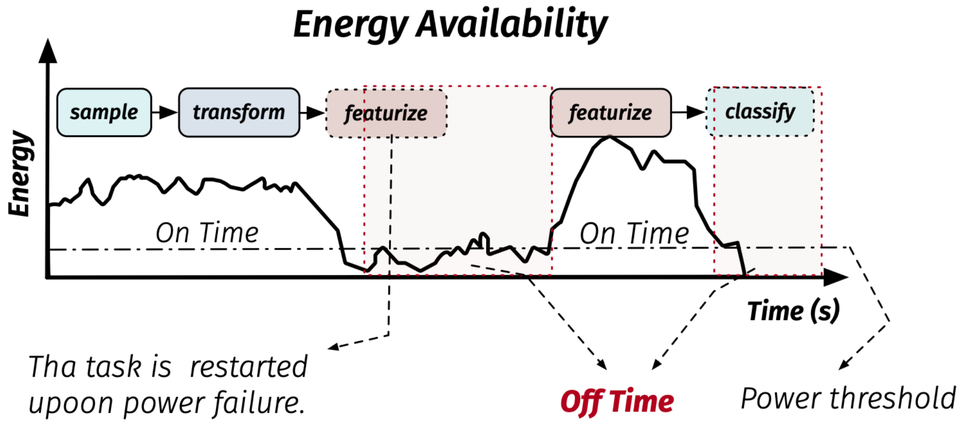
# Task-based Programming Model

- e.g., Chain [Colin et al., OOPSLA 2016]
- Idempotent Tasks
  - Fit into capacitor (forward progress)
  - Atomic execution
    - memory consistency
    - Separate inputs from outputs; e.g. via non-volatile channels
  - Explicit control flow

*Task-shared variables*

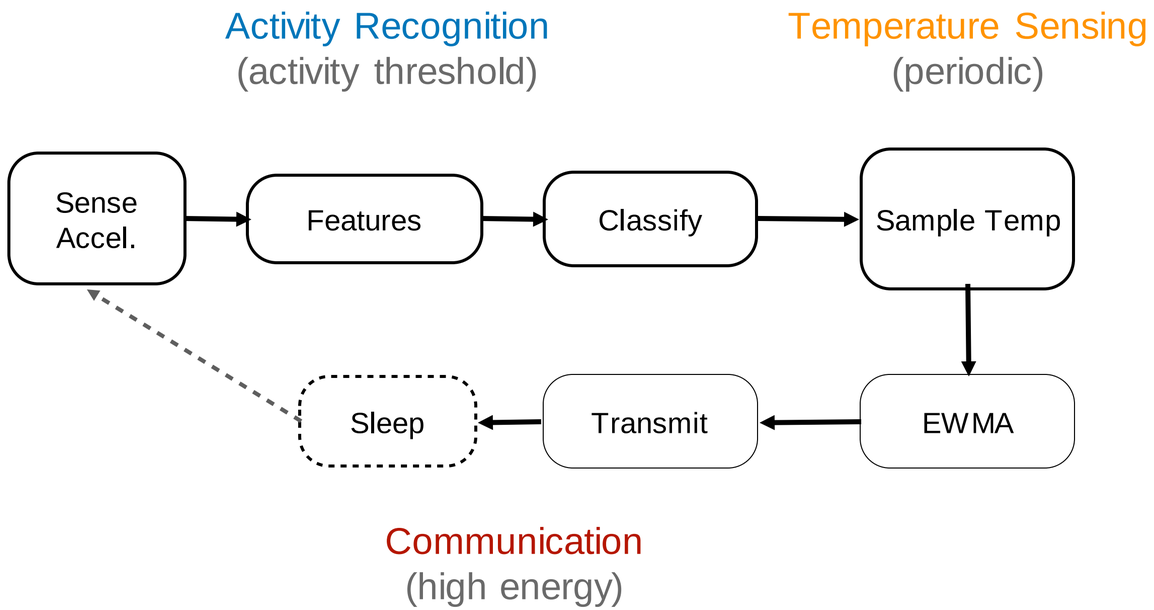
```
int data[N];
float mean;
float avg;
```

```
TASK(sample){
  for(i=0; i<N; i++){
    data[i] = _read();
  }
  NEXT(transform)
}

TASK(transform){
  filter(data);
  NEXT(featurize)
}

TASK(featurize){
  variance = var(data);
  mean = avg(data);
  NEXT(classify)
}

TASK(classify){
  if(mean ==){
    ...
  }
  NEXT(sample)
}
```



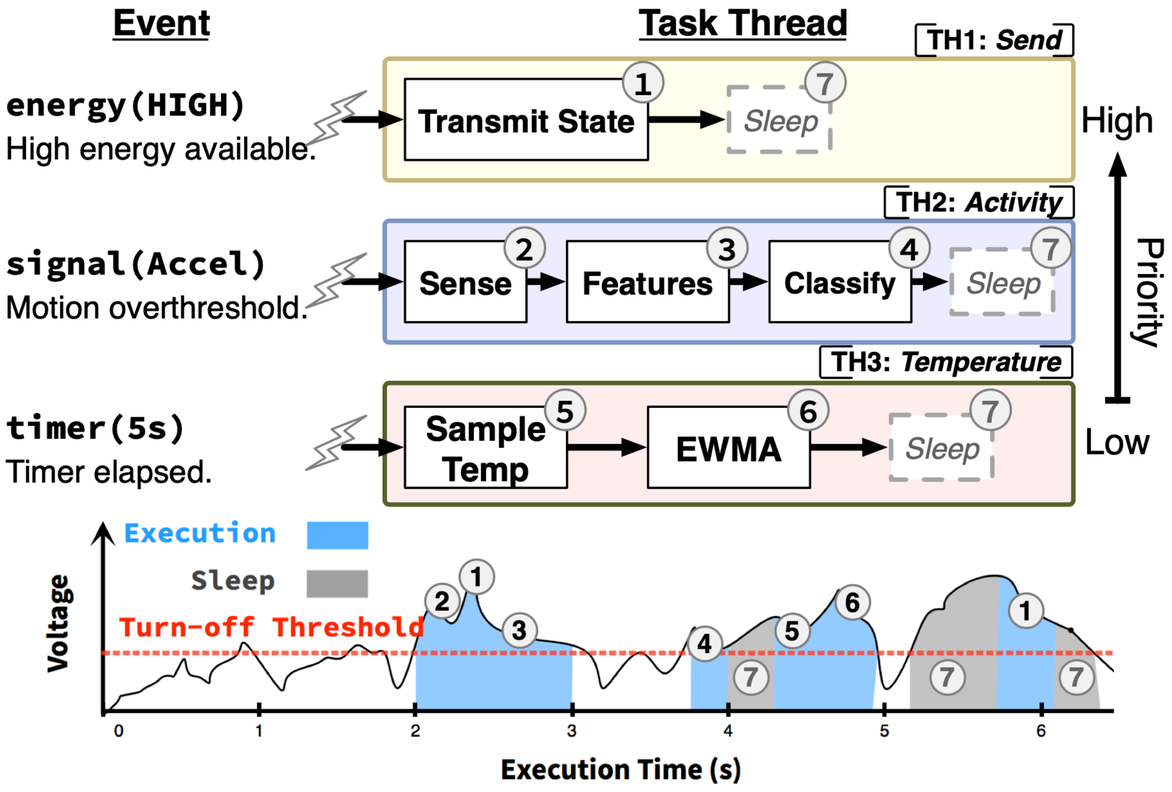
# Task-based Intermittent Computing

- Existing  task-based models are inherently **non-reactive** and **static**.
  - **Crucial Problem:** cannot change control-flow at run-time to respond to events.



# InK - Intermittent Kernel [ACM SenSys'18]

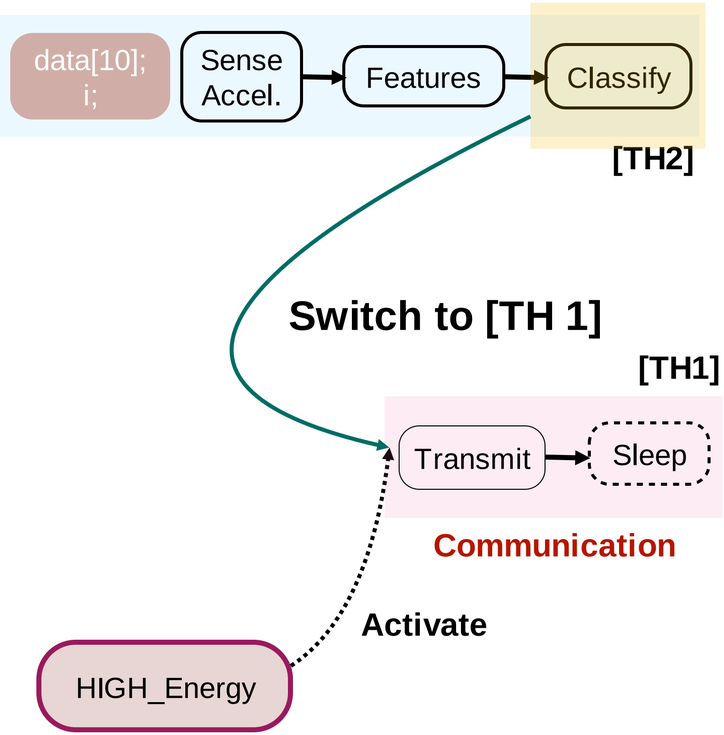
- Reactive, timely, and event-driven batteryless applications.
- **Task-threads**
  - encapsulates successive tasks
  - stack-less, single entry, unique priority
- Static priority-based scheduling
  - task-thread preemptive



# InK - Intermittent Kernel [ACM SenSys'18]

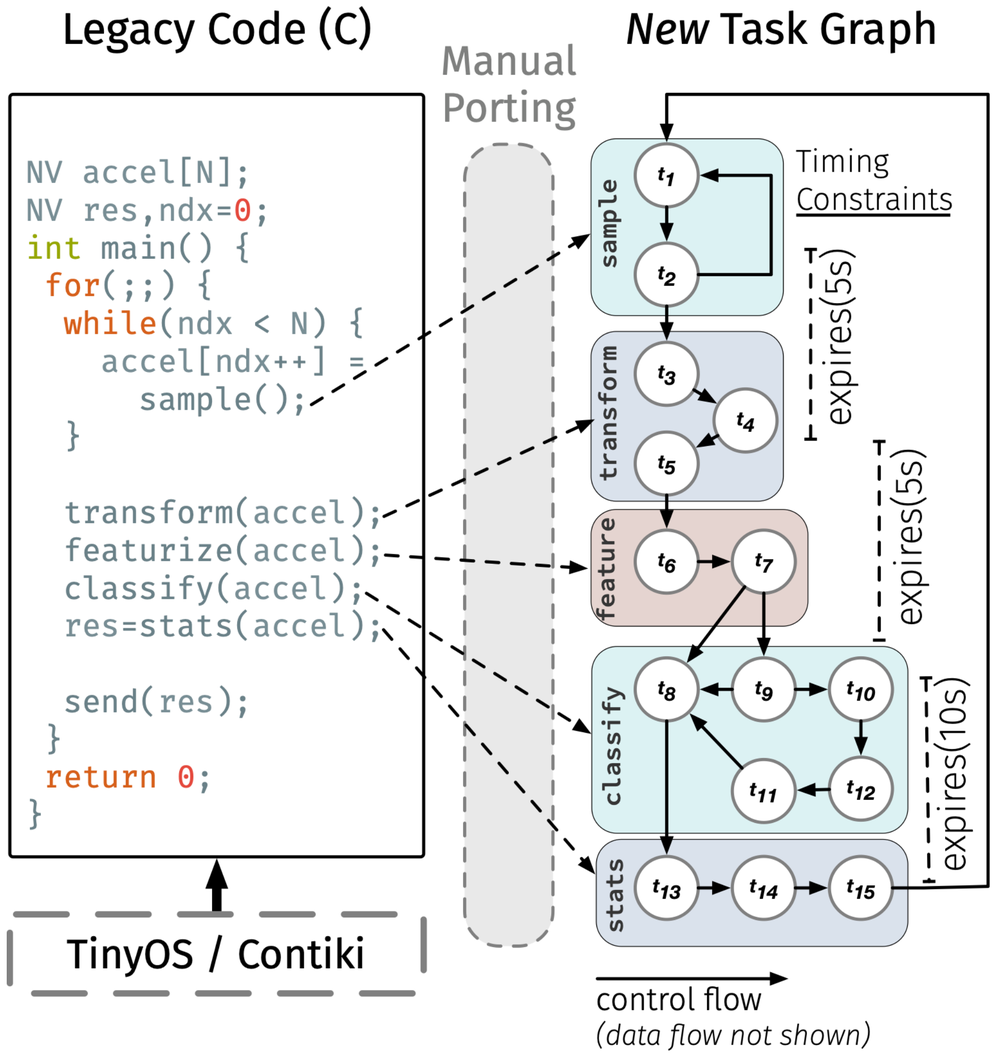
- Tasks and ISRs can activate and deactivate other task-threads
  - change control flow dynamically.

```
__shared (int data [10]; int i;);

ENTRY(Sense_Accel){
  int read = __sample_acc();
  __SET (data[__GET(i)],read);
   ...
   NEXT (Features);
}
...
TASK (Classify){
   ...
   NEXT (...);
}

_interrupt(HIGH_Energy){
   ...
   event . data = dataptr ;
   event . size = datasize ;
   event . timestamp = __getTime();
   __SIGNAL_EVENT (TH1,&_event);
   ...
}
```



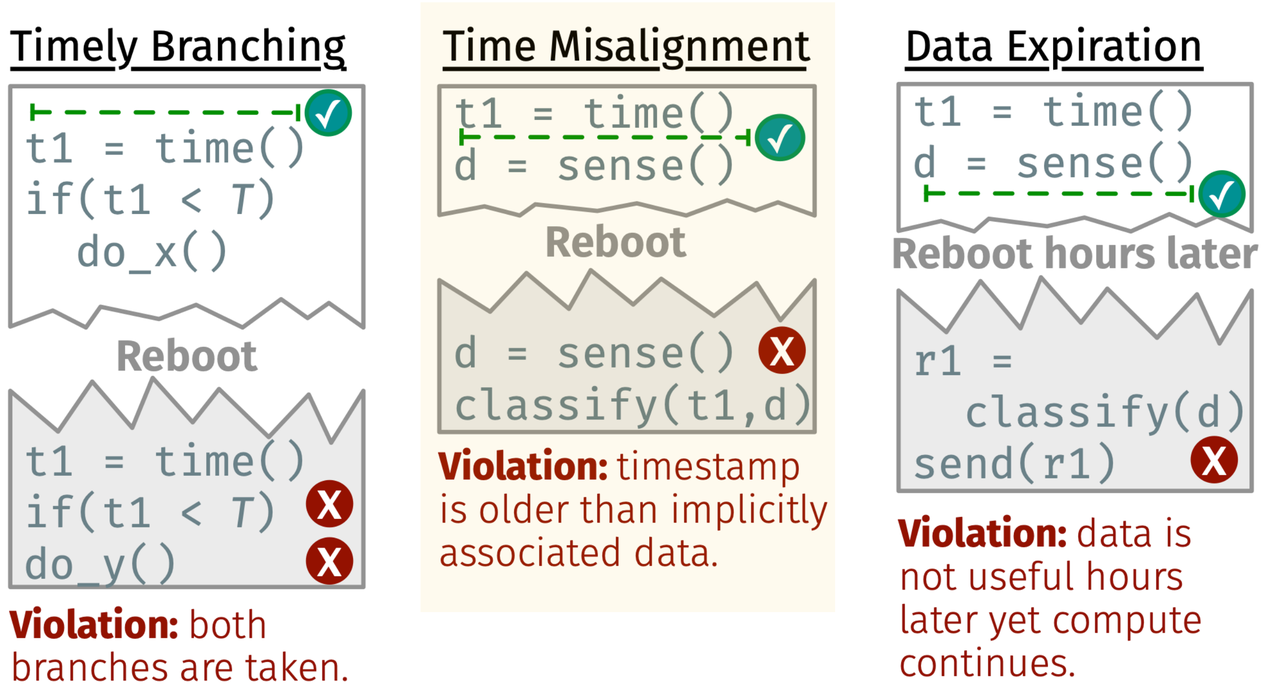
# TICS - Time Sensitive Intermittent Computing [ASPLOS'20]

- The programmer burden of task-based systems is huge!
  - Identification of the tasks
  - Control flow
  - Data expiration
- Checkpoint systems
  - Scalability issues (Pointers)
  - Time inconsistencies



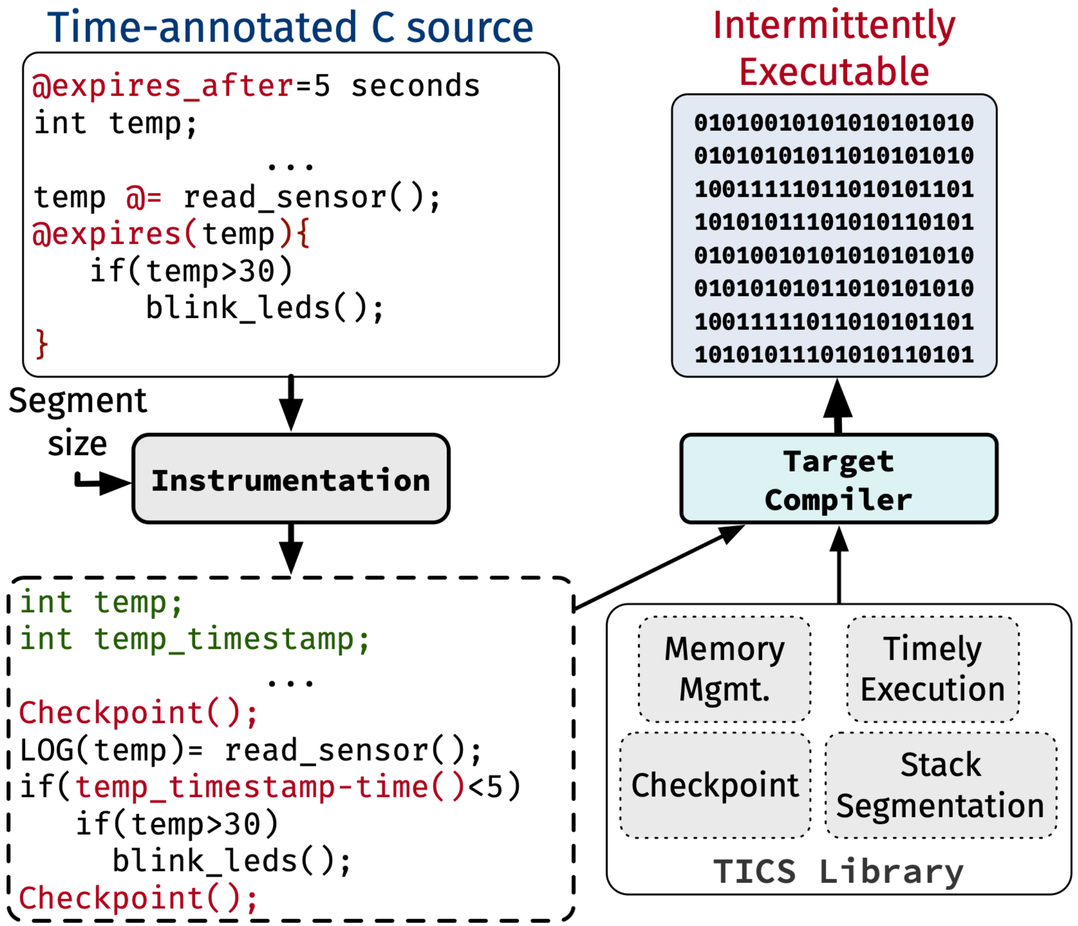
# TICS - Time Sensitive Intermittent Computing [ASPLOS'20]

- Time consistency violations.
  - Data expiration
  - Time misalignment
  - Timely branching



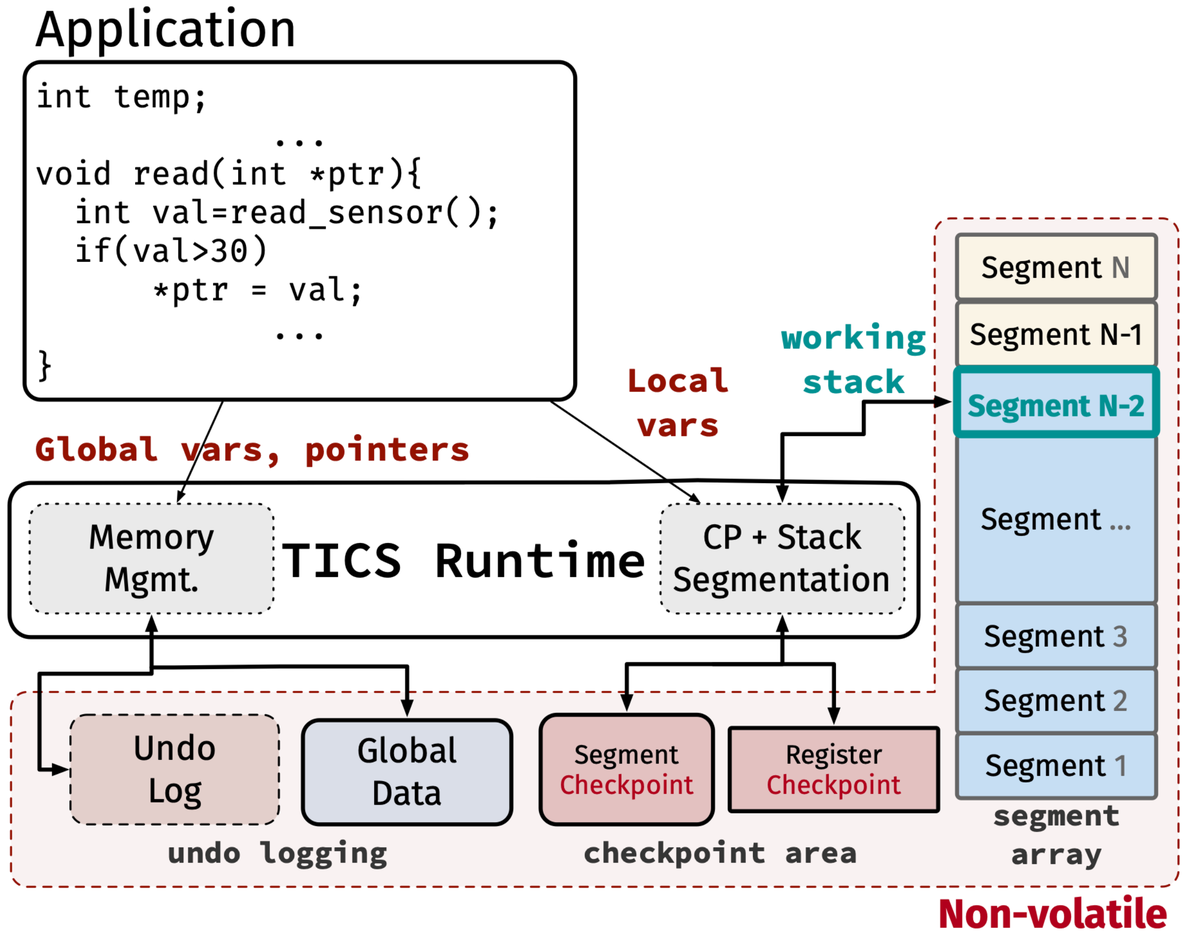
# TICS - Time Sensitive Intermittent Computing [ASPLOS'20]

- C language support
- Time consistency handling
  - Timely execution
- No programmer burden



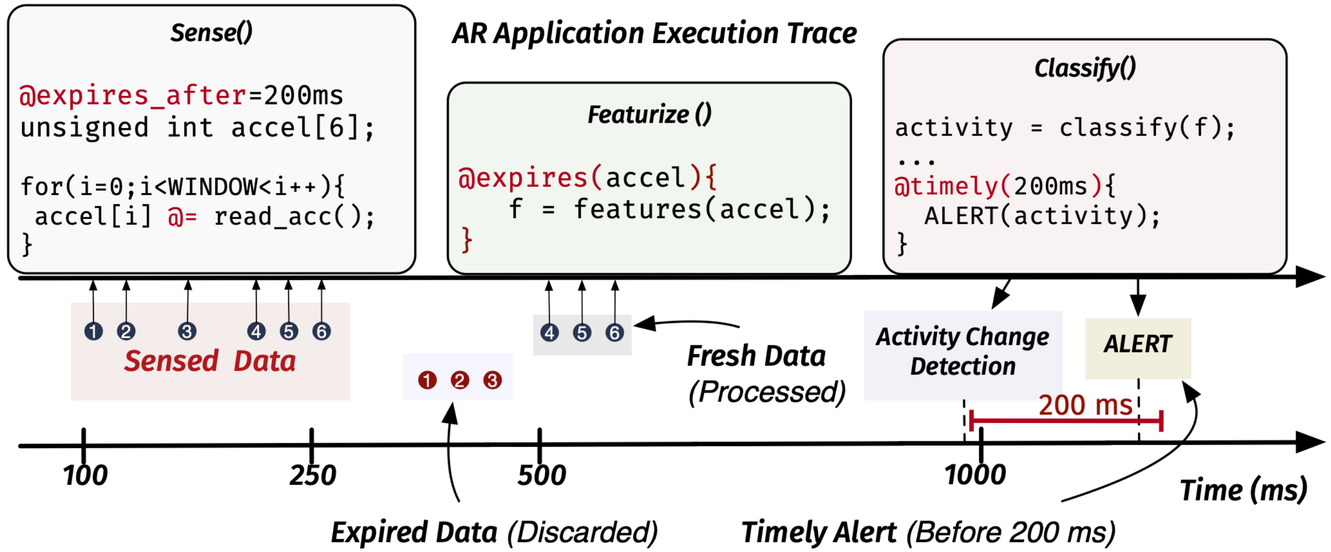
# TICS - Time Sensitive Intermittent Computing [ASPLOS'20]

- C language support
  - Memory management for pointers
- Constant worst-case checkpoint size
  - Stack segmentation

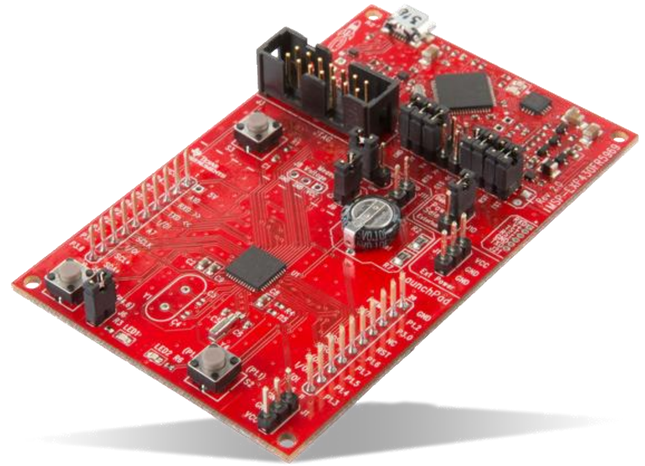


# TICS - Time Sensitive Intermittent Computing [ASPLOS'20]

- An intermittent activity recognition application with timing constraints.



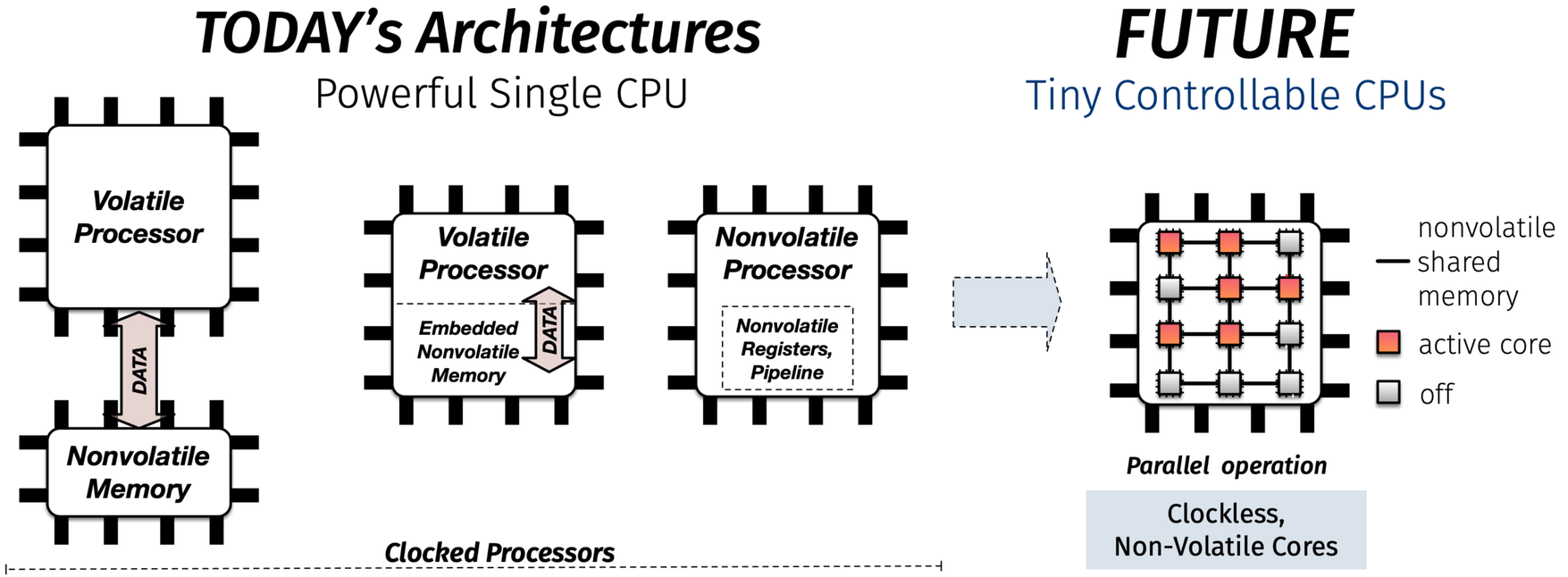

| Time Consistency Violation | Potential Count (during experiment) | Observed Violations w/o TICS | Observed Violations w/ TICS |
|---|---|---|---|
| *Timely Branch* | 256 | 32 ✗ | 0 ✓ |
| *Time Misalignment* | 870 | 78 ✗ | 0 ✓ |
| *Data Expiration* | 870 | 173 ✗ | 0 ✓ |



MSP430FR5969

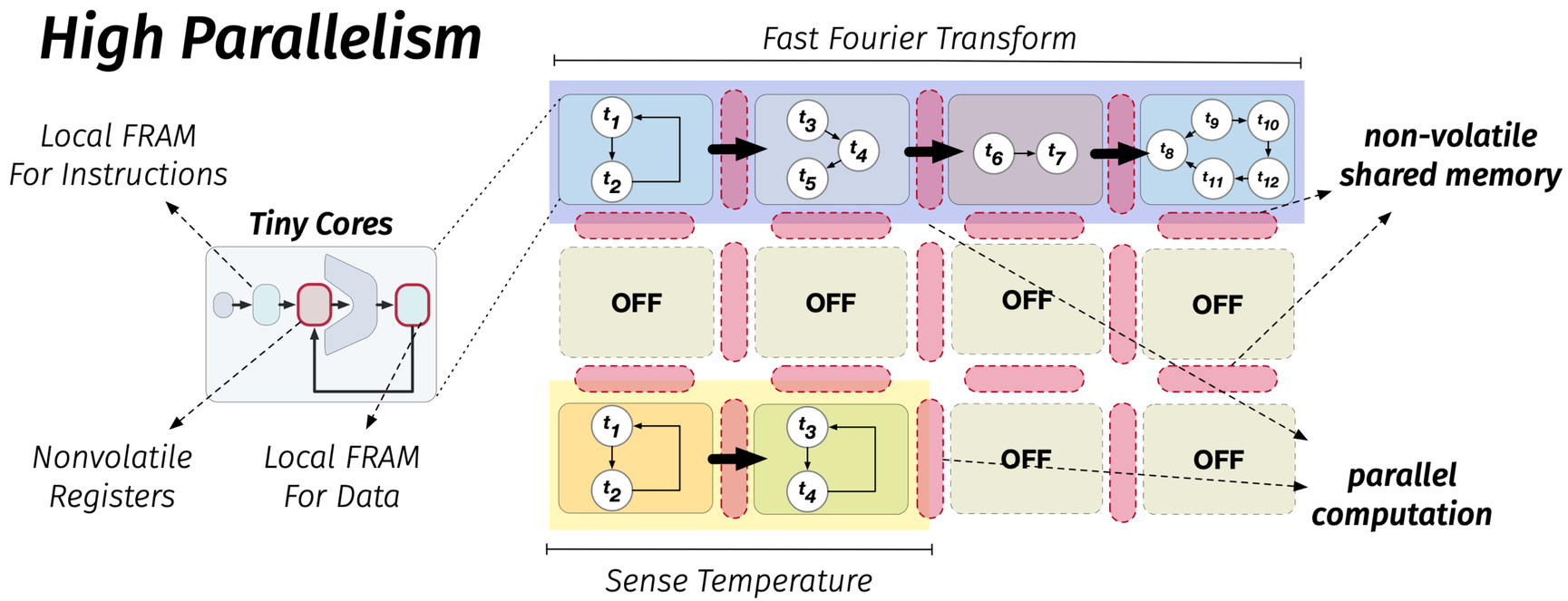
# Future Proposal: Energy Scaling Processor Architecture

- Today's batteryless sensors are based on conventional hardware architectures
  - the software can control the energy consumption only to some extent



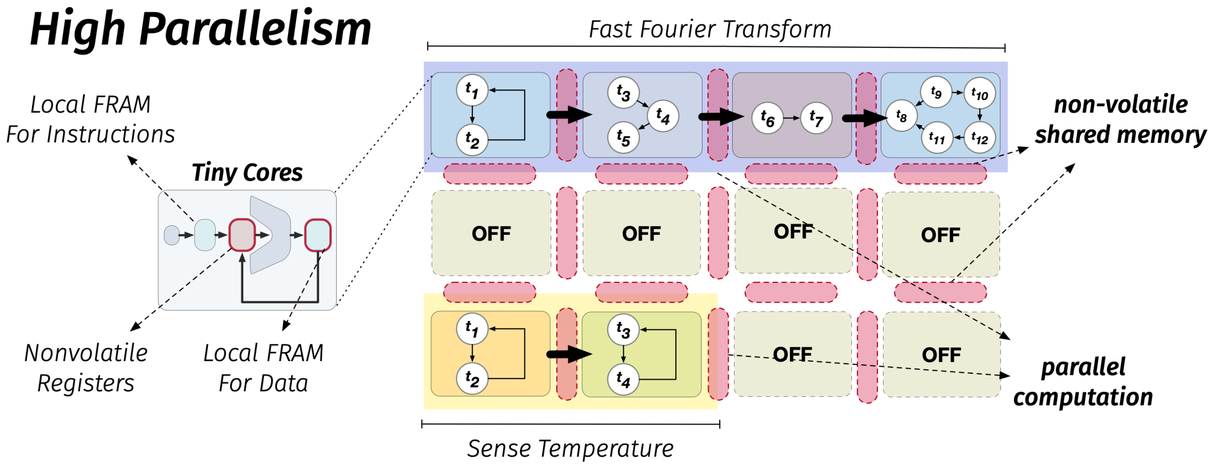
# Future Proposal: Energy Scaling Processor Architecture

- Tiny cores that are energy-input responsive
  - Fine-grained control of energy consumption



# Future Proposal: Energy Scaling Processor Architecture

- Challenges
  - Multi-core Design and Implementation
    - volatile/non-volatile
    - synch/async
  - Developing a Programming Model (Runtime)
    - Language constructs
    - Adaptation/Approximate Computing



# Conclusions

- Our runtimes **InK** and **TICS** for intermittent computing
  - InK [Yildirim et al., SenSys'18]
    - Reactive task-based execution
  - TICS [Kortbeek et al., ASPLOS'20]
    - Removes cognitive burden and enables timely execution
- Future: Energy-responsive software and hardware
  - Energy-responsive and dynamic adaptation at run-time via many cores