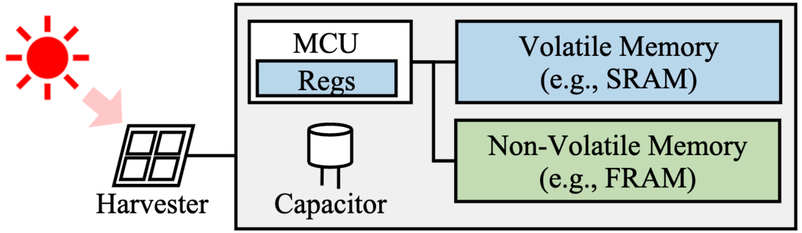
Advanced Operating System

IntOS: Persistent Embedded Operating System and Language Support for Multi-threaded Intermittent Computing

48-256445 山上航輝

Background: Intermittent Computing

- **Energy harvesting system**
  - Capture necessary energy from the environment (e.g., solar, RF)
  - No large battery
  - IoT, wearables, sensor networks, etc.
- **Frequent power interruptions**
  - Program execution is **intermittent**
  - Registers and SRAM states are lost (volatile)
  - Crash consistency is needed



Problems

- Embedded OSes are used in resource-constrained environments
  - Multi-threading, semaphores, events, timers, etc.
- **Existing embedded OSes** (e.g., FreeRTOS) **are not designed to be crash-consistent and do not support intermittent computing.**

Related Work

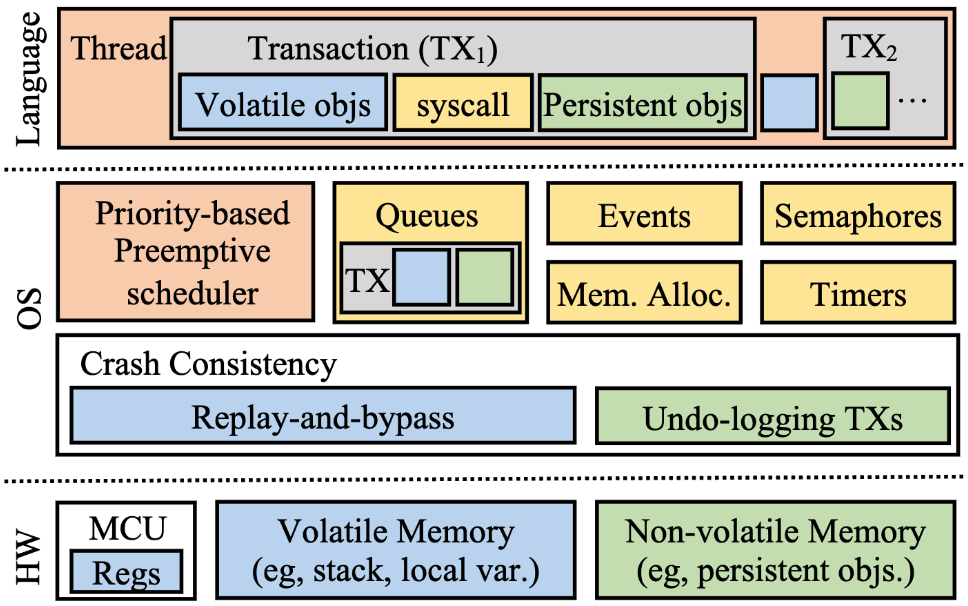
- **Manual task decomposition** (e.g., Alpaca [1])
  - Good performance
  - Requires manual efforts
- **Automatic checkpointing** (e.g., Ratchet [2] , ImmortalThreads [3])
  - Requires little or no annotations
  - NVM only, slow and less energy-efficient compared to SRAM

[1] K. Maeng+, OOPSLA'17
[2] J. V. D. Woude+, OSDI'16
[3] E. Yıldız+, OSDI'22

Proposal

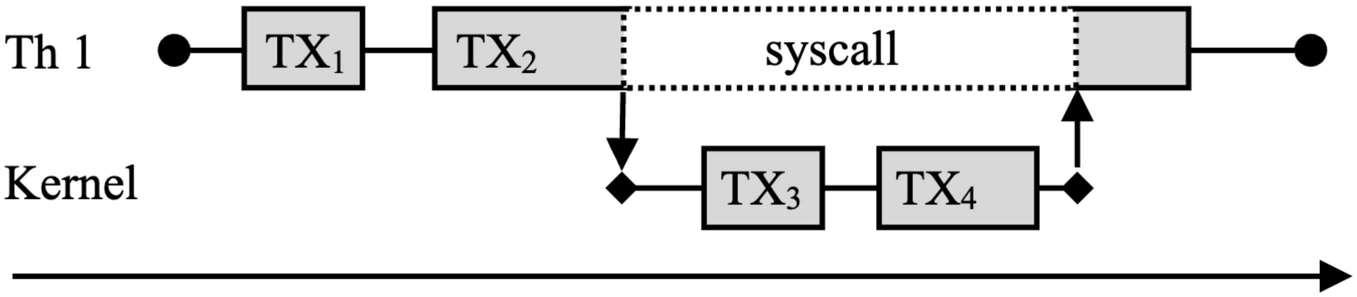
- **IntOS**
  - First embedded OS that supports multithreading and other core features, designed for intermittent computing
  - Combines **transactional programming with a replay-and-bypass mechanism**, utilizing both **volatile and non-volatile** memories
  - Rust-based programming model to ensure crash consistency

IntOS Design



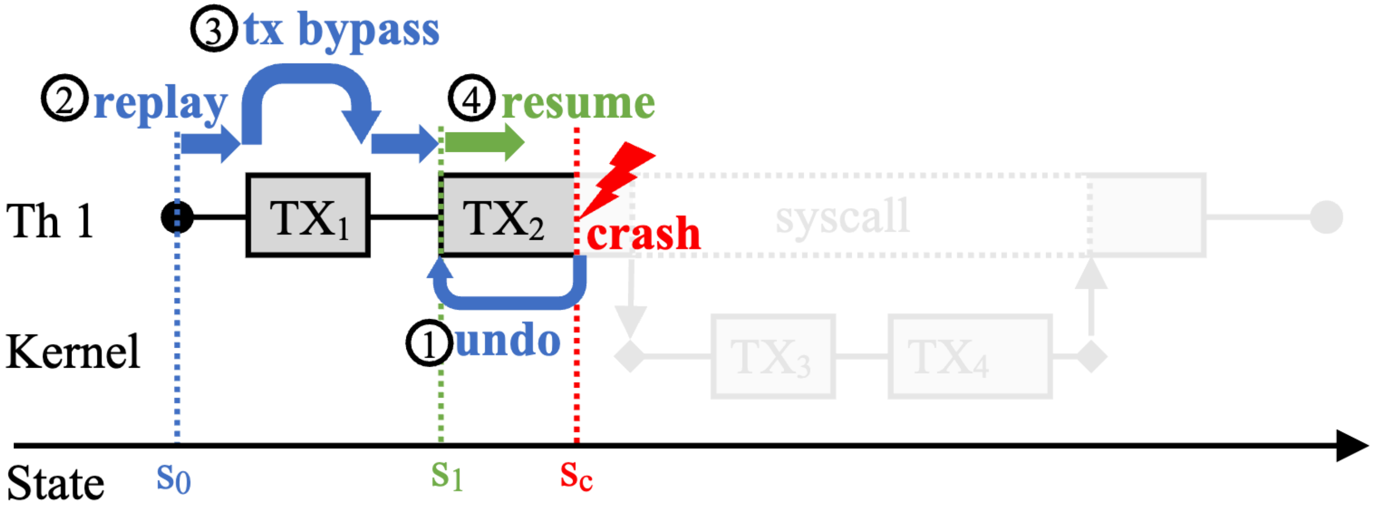
- **Supports multithreading**
  - Priority-based preemptive scheduling
- **Transactions**
  - Updates on persistent objects are automatically logged
- **Replay-and-bypass**
  - Restore volatile states after crash
  - Bypass committed transactions and syscalls to avoid re-execution

Transactions



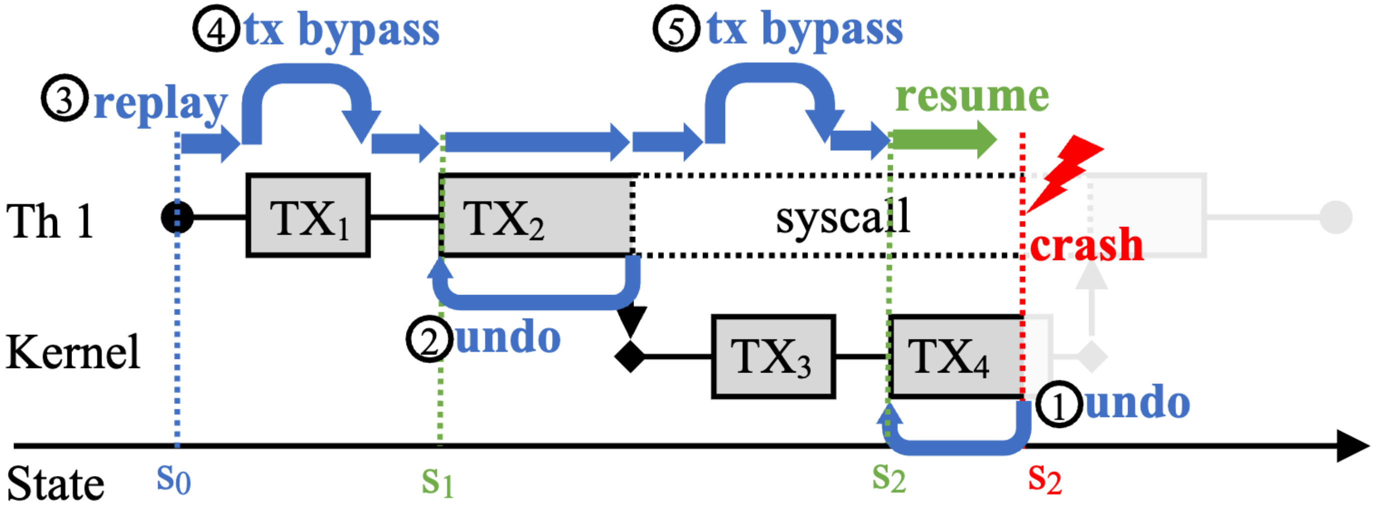
- A thread includes transactions (TX1 and TX2) for persistent objects
- A syscall contains transactions (TX3 and TX4) in the kernel

Replay-and-bypass Mechanism — Case 1



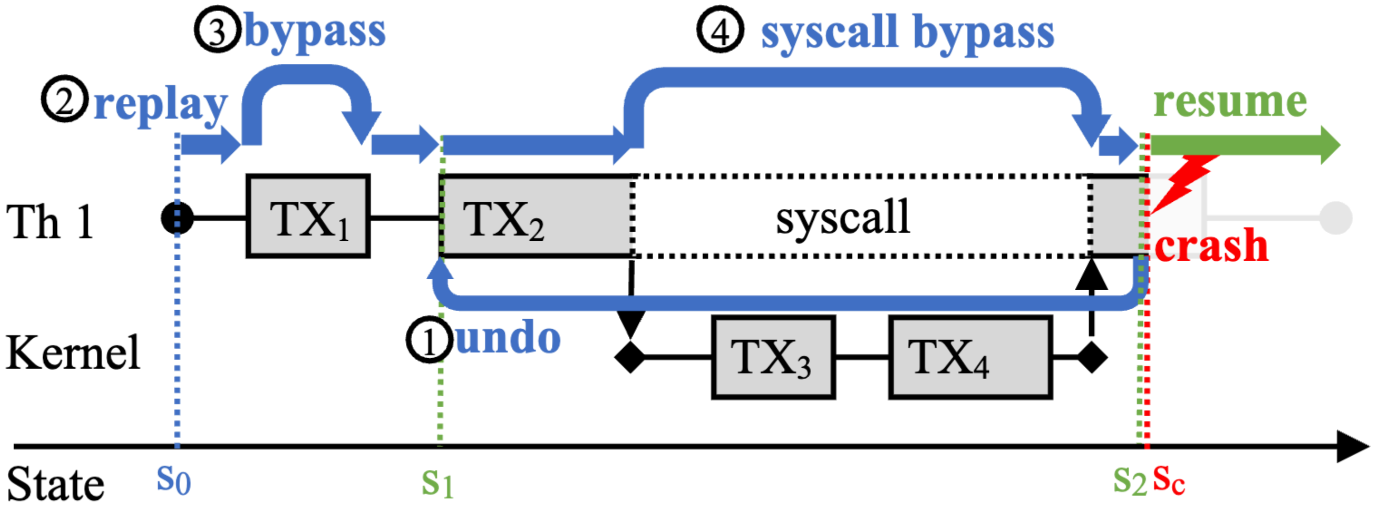
- **Undo**: Rollback the uncommitted non-volatile states
- **Replay**: Re-execute the thread to restore the volatile states
- **Bypass**: Skip the committed transactions and syscalls

Replay-and-bypass Mechanism — Case 2



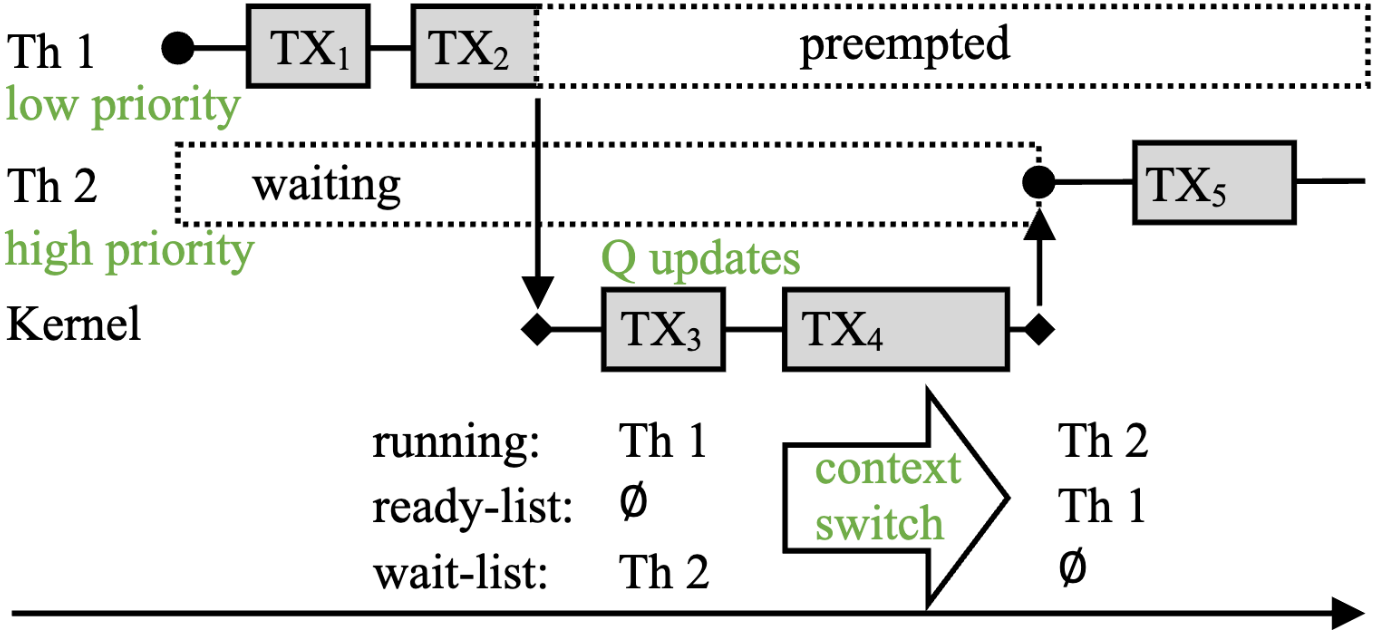
- **Undo**: Rollback the uncommitted non-volatile states
- **Replay**: Re-execute the thread to restore the volatile states
- **Bypass**: Skip the committed transactions and syscalls

Replay-and-bypass Mechanism — Case 3



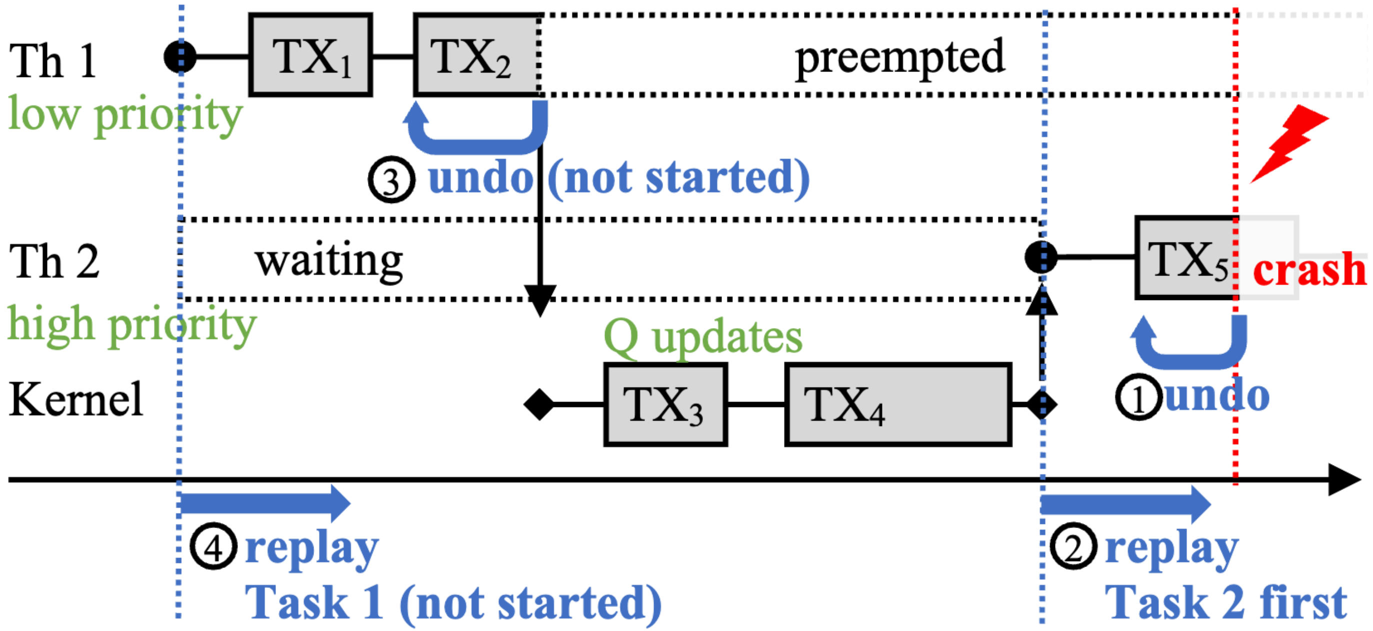
- **Undo**: Rollback the uncommitted non-volatile states
- **Replay**: Re-execute the thread to restore the volatile states
- **Bypass**: Skip the committed transactions and syscalls

Multi-thread Crash Consistency



- Kernel maintains multiple ready-lists and wait-lists

Multi-thread Crash Consistency



- Recovers the ready task with the highest priority first
- Other tasks will be recovered when they are scheduled later

Programming Model (enforced by Rust)

- Persistent objects should not be accessed outside the transaction
- References to persistent objects should not be returned from the transaction
- Persistent objects should not contain references to volatile objects
- System calls should only be made within transactions
- Locks should not be used inside transactions

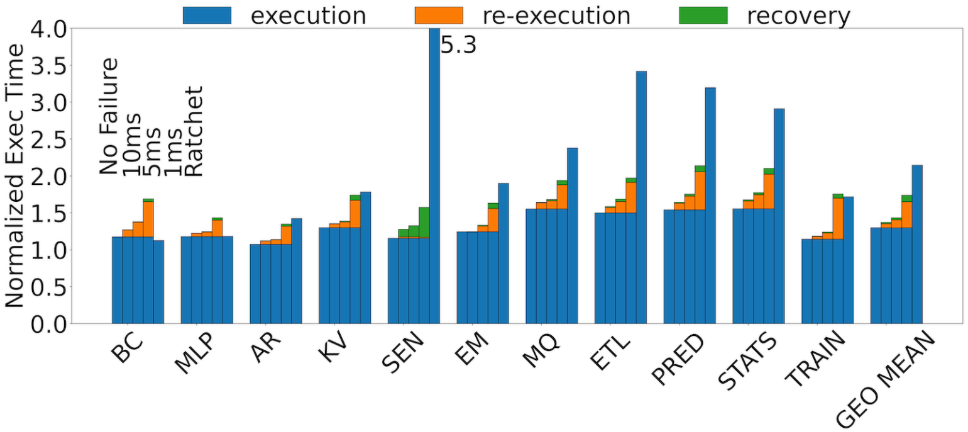
Implementation

- IntOS is implemented in Rust, based on the FreeRTOS kernel
- Three per-thread replay tables that cache the return values
  - User-level transactions
  - Kernel-level transactions
  - Syscalls
- Three performance optimizations
  - Loop optimization
    - Use non-volatile iteration counter to avoid excessive replay window size
  - Linked list optimization
  - Undo-logging optimization

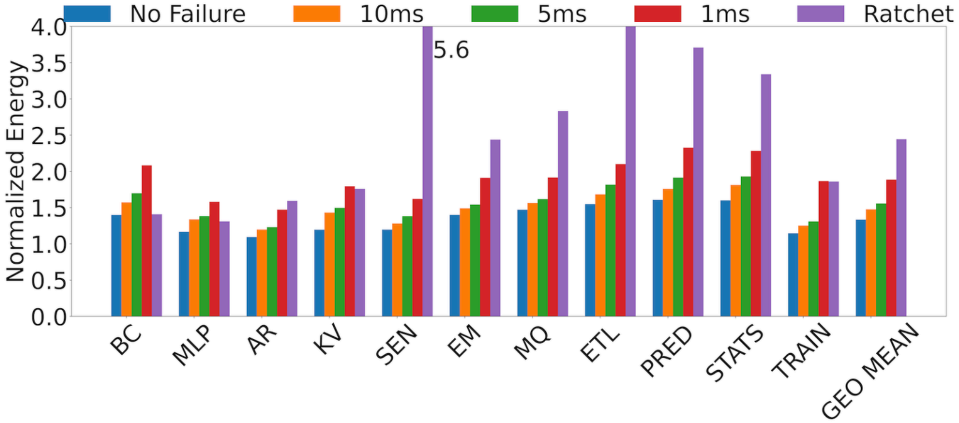
Evaluation

- Seven micro-benchmarks (Activity recognition, Sensing, MLP, etc.) and four macro-benchmarks
- MSP430 and Apollo 4 as the testbeds

Performance overhead with power failure



Latency overhead w/ power failure



Energy overhead w/ power failure

- Lower latency and energy overheads compared to prior work (Ratchet)
- Can make progress under frequent power failures

Summary

- Introduced IntOS, an embedded OS designed for multi-threaded intermittent computing on a battery-less energy-harvesting system
- IntOS utilizes both volatile and non-volatile memories, ensuring crash consistency through transactions and a replay-and-bypass mechanism
- IntOS can make progress under frequent power failures at lower runtime and energy overheads than prior works
- IntOS ensures whole system consistency using Rust-based type system