A Difference World: High-performance, NVM-invariant, Software-only Intermittent Computation

Harrison Williams
hrwill@vt.edu

Saim Ahmad
saim19@vt.edu

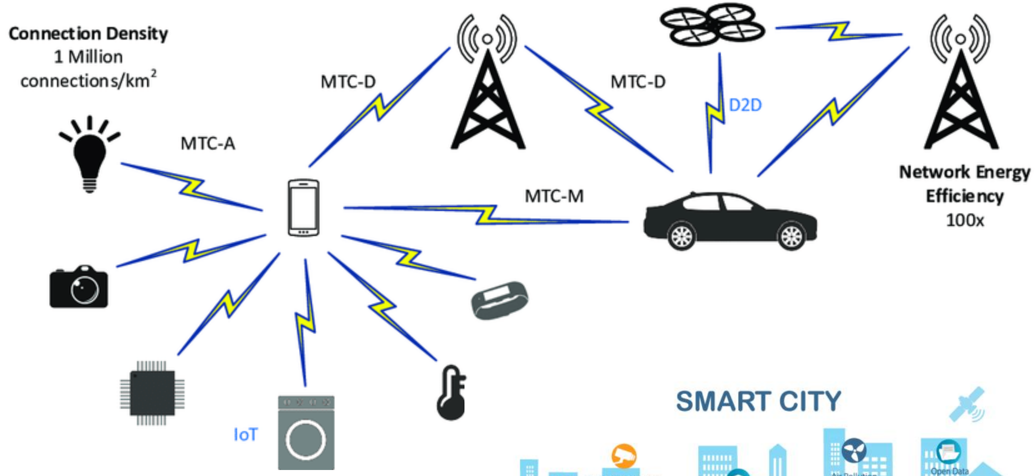
Matthew Hicks
mdhicks2@vt.edu

COMPUTER SCIENCE
VIRGINIA TECH

Mobile and IoT deployments are reaching massive scales

Billions of IoT devices

- Market estimate: over 25 billion devices by 2030
- Dominated by tiny, resource-limited sensor nodes

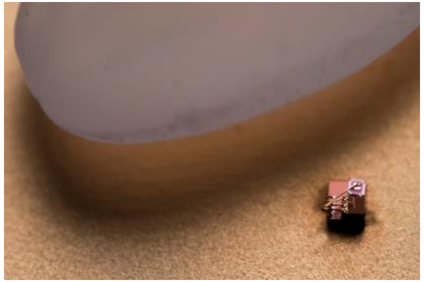


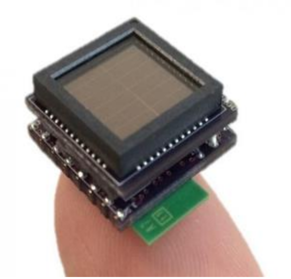
Massive-scale applications

- Industrial IoT
- Wearables
- Smart cities

Battery power is a **non-starter** at this scale

Batteryless systems enable new deployments





System-level Benefits

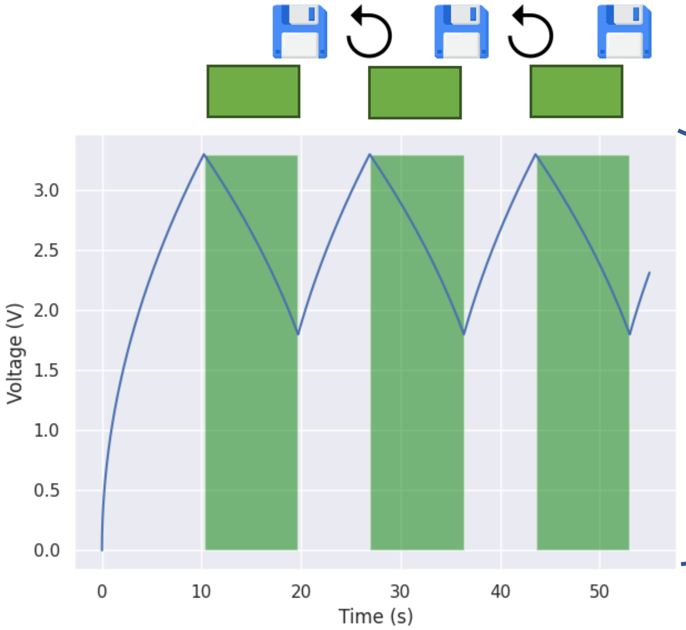
- Small
- Cheap
- Long-lived

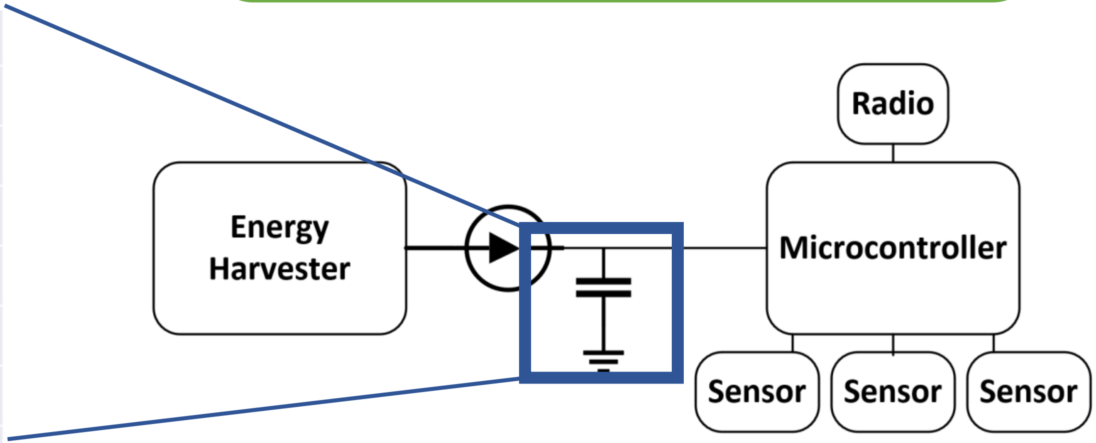
Device-level Challenges

- Power interruptions
- Energy constraints
- Software bugs

Intermittent software execution

Checkpoints sustain computation across power cycles

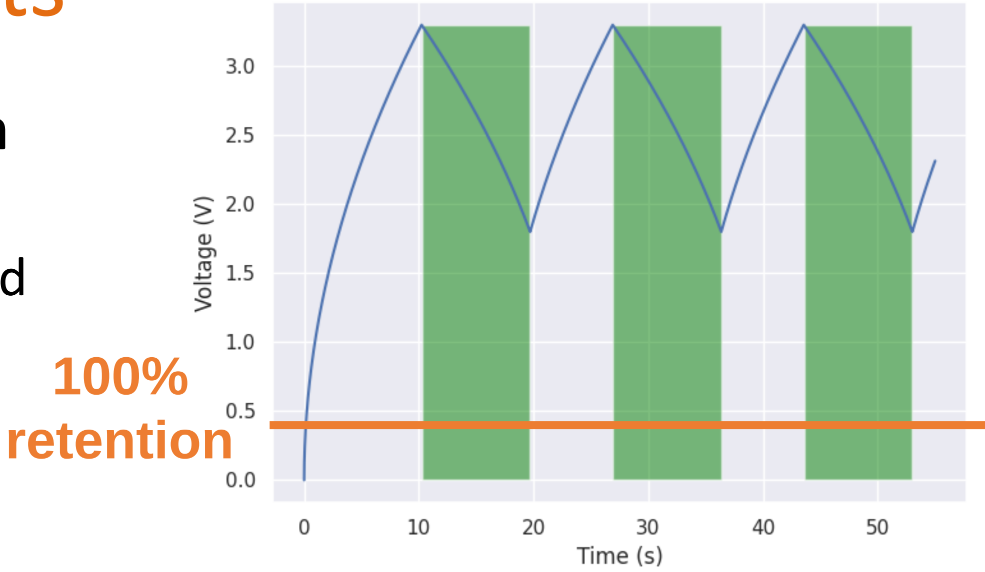




SRAM-based checkpoints

Typical checkpointing depends on performant NVM

- Flash: high-power, endurance limited
- FRAM/MRAM/ReRAM: limited adoption/availability

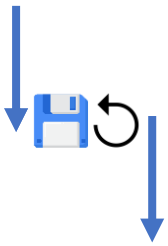


TotalRecall (ASPLOS `20): store checkpoints in SRAM

- Data retention well below MCU minimum
- Full retention for hours to days
- Verify integrity with checksum

"Checkpoint" is a checksum over all SRAM

Many operations require rollback



```
while(1){
  samples[count] = takeSensorReading();
  count++;
  if(count == 50){
    compressSamples();
    count = 0;
    txDataOverRadio();
  }
}
```

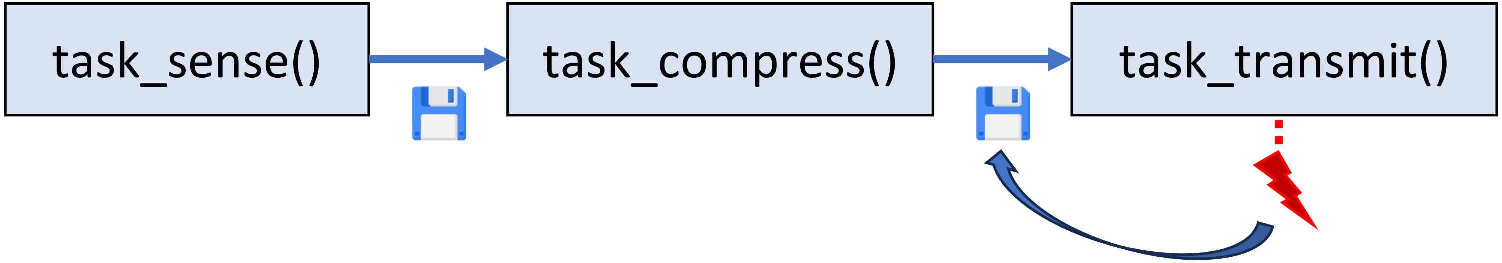
Can be split across power cycles

Must occur in **one** power cycle

Execution must roll back to beginning of atomic operation

Correctness, performance, programmability challenges

Task-based models make rollback tractable



Programmer decomposes code into tasks

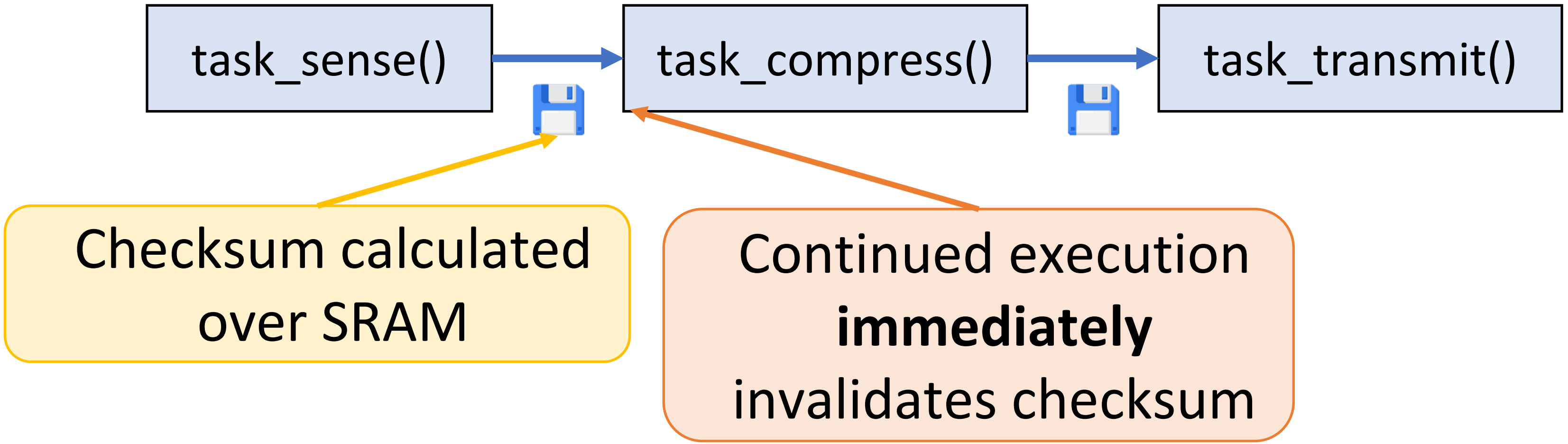
Compiler inserts code for checkpointing and rollback

Automatic rollback: no need for hardware energy monitor

Foundation: “known-good” state stored in NVM

Problem: incompatible with SRAM-based checkpointing

Task-based models make rollback tractable



Question: how can we apply in-place SRAM checkpoints to task-based intermittent systems?

Camel: mixed-volatility SRAM worlds

SRAM

Volatile	"Non-Volatile" (checksum-backed)

Store working data in volatile SRAM

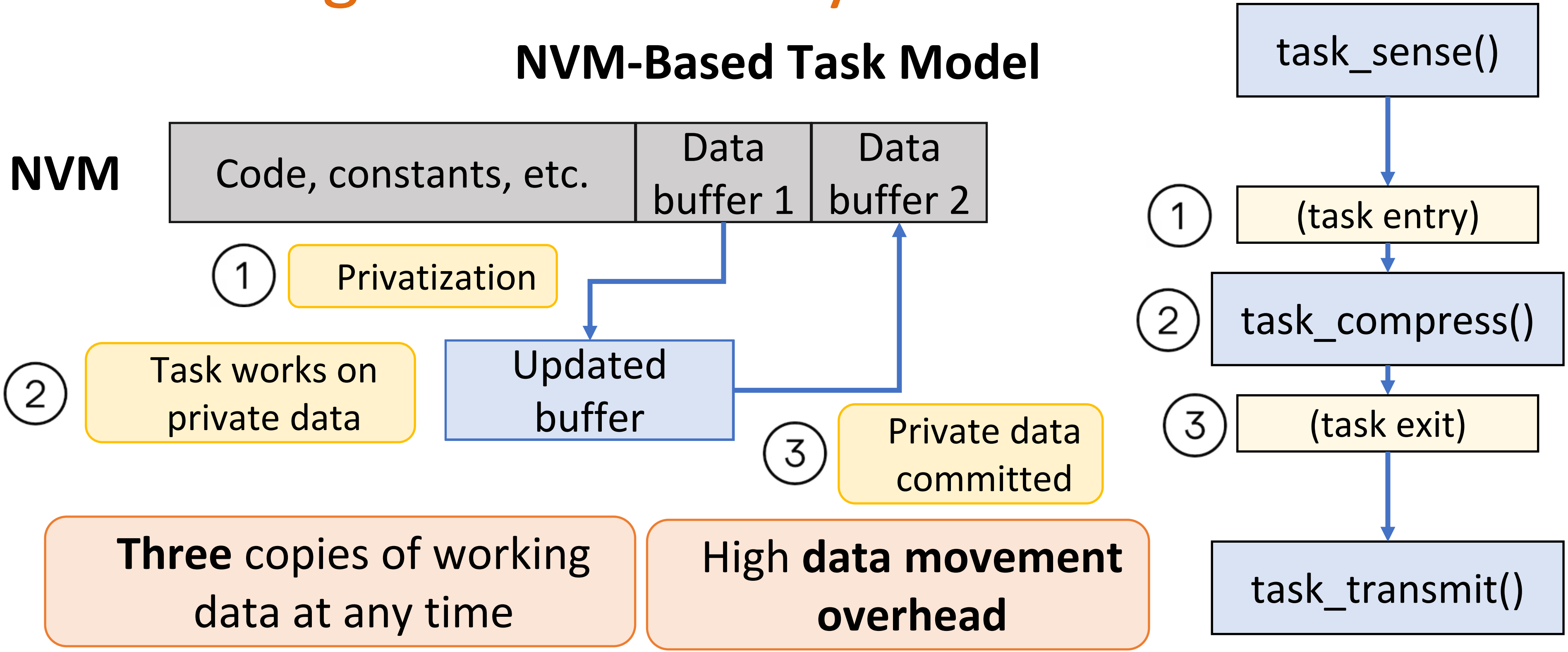
Store known-good state in checksum-backed region of SRAM

Main design considerations:

SRAM is scarce → minimize memory overhead

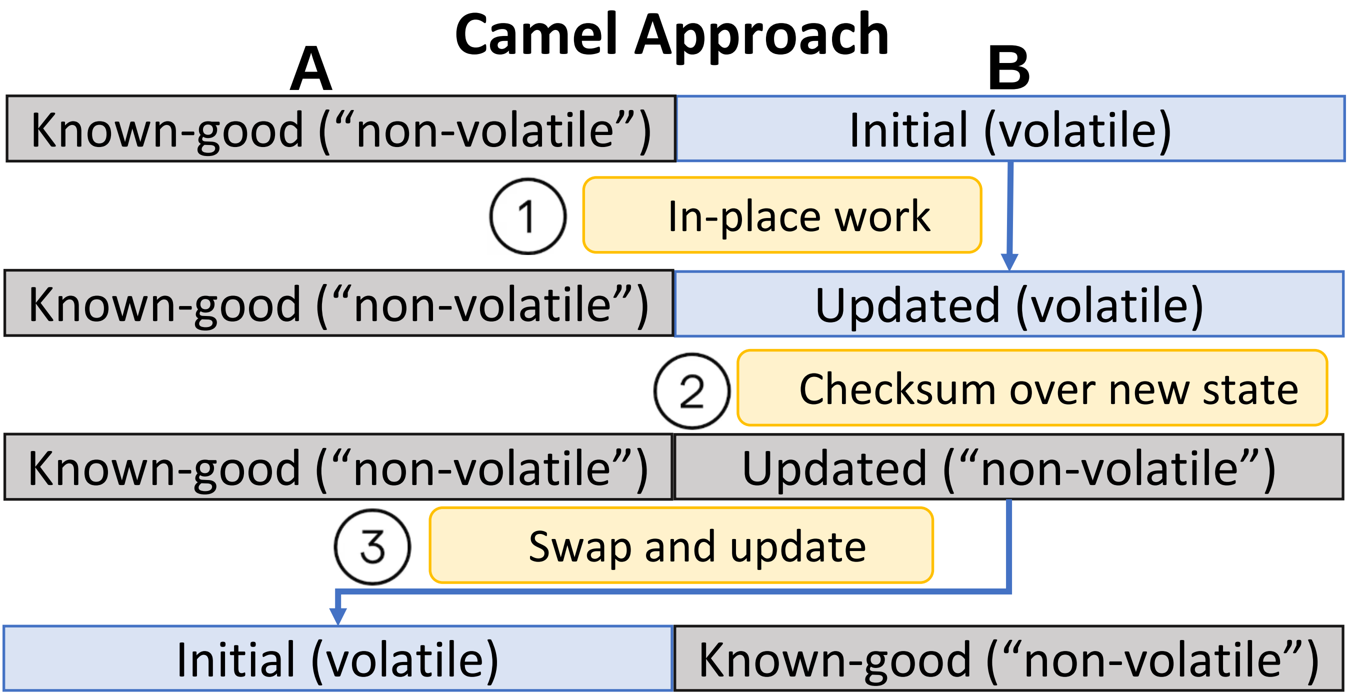
Checksum is expensive → minimize writes to NV world

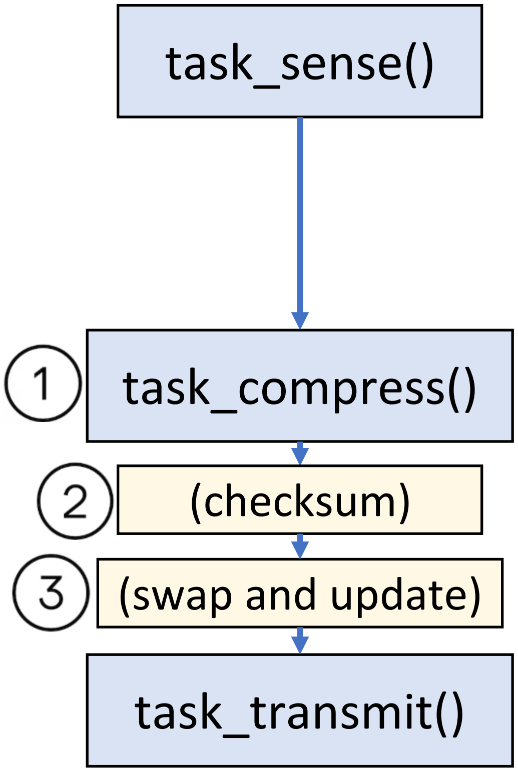
Alternating world volatility



COMPUTER SCIENCE
VIRGINIA TECH.

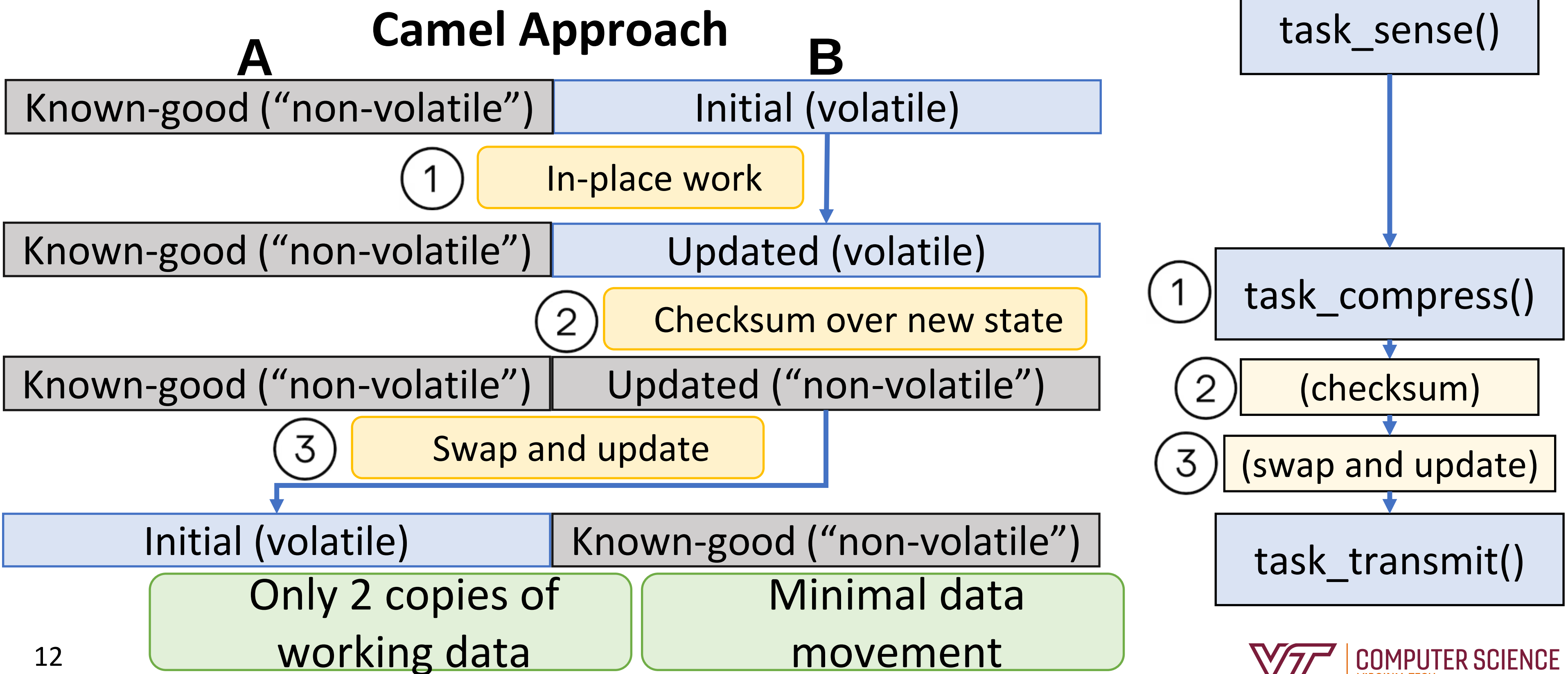
Alternating world volatility





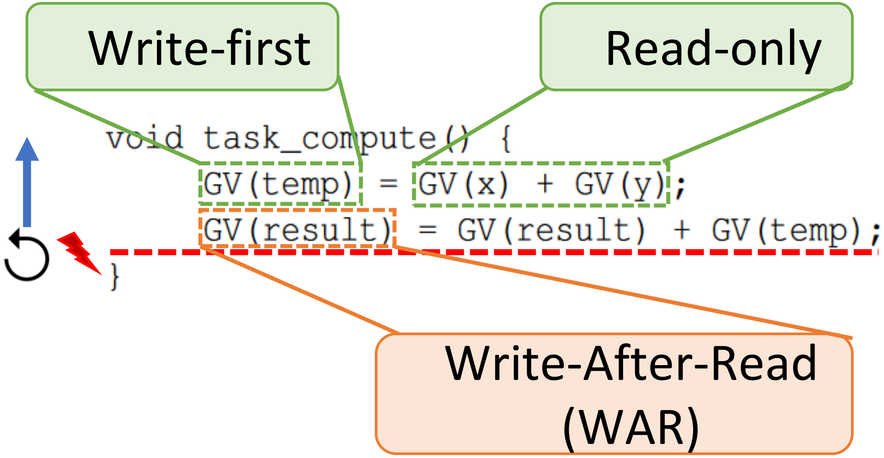
Camel **eliminates data privatization**

Alternating world volatility



Efficient state rollback after power failures

Variable	temp	x	y	result
Initial	0	1	2	4
Execution 1	3	1	2	7
Execution 2	3	1	2	10



Efficient state rollback after power failures

Variable	temp	x	y	result
Initial	0	1	2	4
Execution 1	3	1	2	7
Rollback	3	1	2	4
Execution 2	3	1	2	7

Initial state does not affect output

Rolled back to enforce idempotency

Write-first

Read-only

```
void task_compute() {
    GV(temp) = GV(x) + GV(y);
    GV(result) = GV(result) + GV(temp);
}
```

Write-After-Read (WAR)

Camel compiler identifies the minimum set of variables to roll back for correctness

Evaluation scenarios and benchmarks

Two target platforms

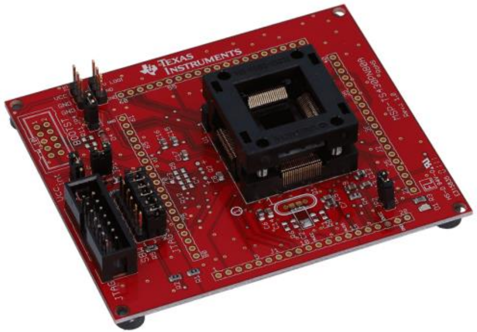
- MSP430G2955 (Flash)
- MSP430FR6989 (FRAM)

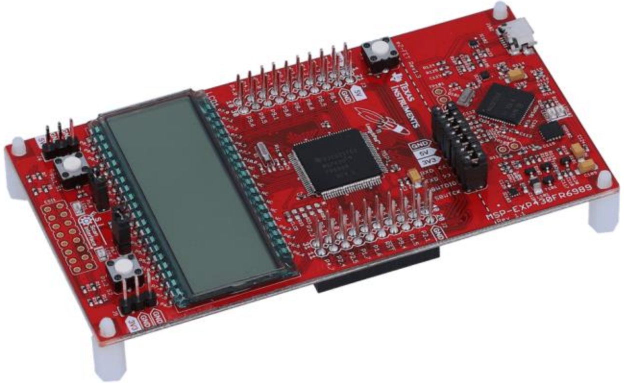
Hardware and simulation

- Hardware: RF energy harvester
- Simulation: measure CPU cycles, deep program instrumentation

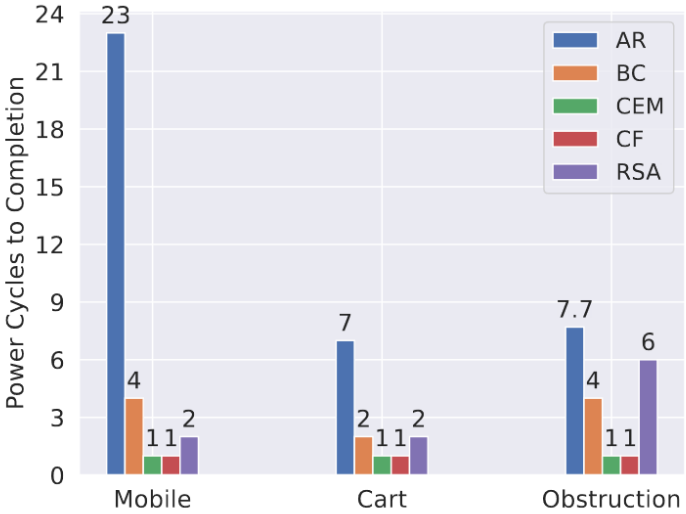
Baselines + benchmarks

- TotalRecall and prior task-based systems
- 8 benchmarks for correctness and performance





Efficient, correct SRAM-based intermittent execution



Benchmark	TotalRecall	Camel
Transmit	Fails	✓
Actuate	Fails	✓
Sense	Hangs	✓

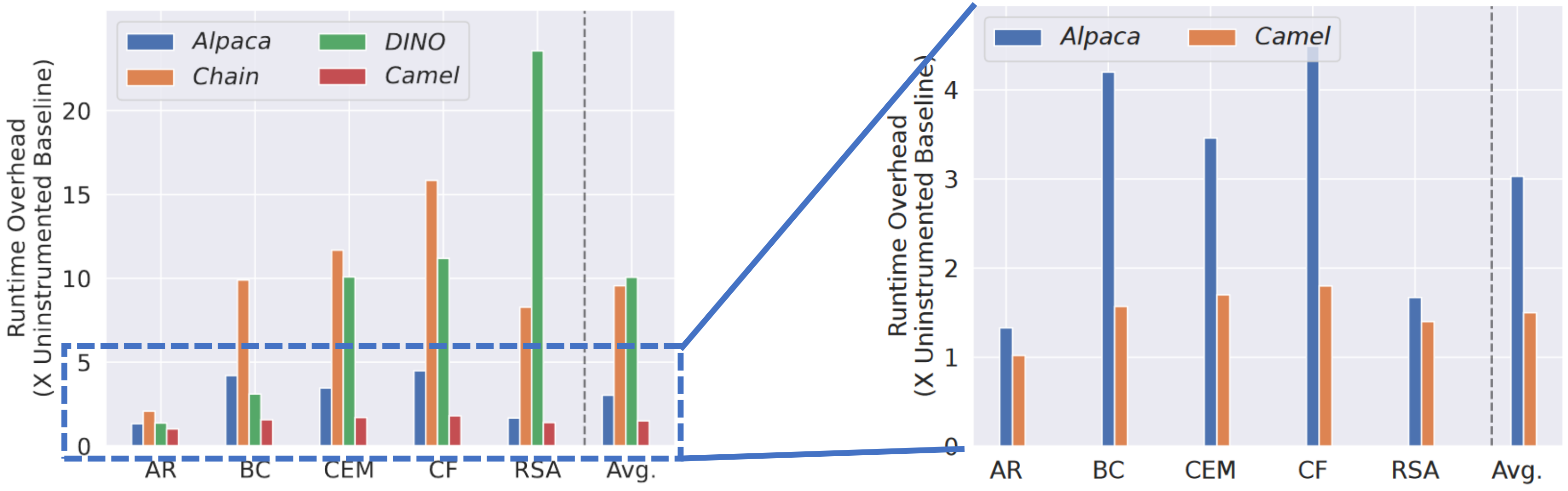
Camel eliminates the need for on-chip voltage monitoring

3-5x performance improvement over TotalRecall

Camel correctly executes peripheral-centric software

COMPUTER SCIENCE
VIRGINIA TECH.

Differential buffer design cuts software overhead



Camel's buffer design outperforms next-best task-based system by 2x

Differential buffer approach improves *all* intermittent systems

	AR	BC	CEM	CF	RSA	avg.
DINO [25]	1136	717	259	324	1830	788
Chain [5]	2008	717	231	452	315	744
Alpaca [28]	2008	717	225	452	315	743
CAMEL	1999	709	114	385	254	692

Commit count

High-performance, NVM-invariant intermittent computation

Camel brings efficient, correct intermittent computation to the largest class of devices today

Camel's differential buffer design substantially improves task-based systems on *any* intermittent platform

See the paper for more: memory consumption, checkpoint cycle overhead, integrity check methods, etc.

Group: forte-research.com
Me: harriswms.github.io