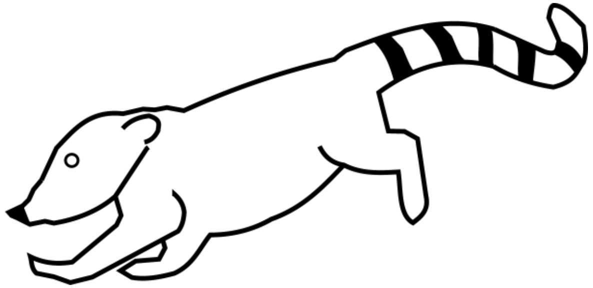
Transactional Concurrency Control for Intermittent, Energy-Harvesting Computing Systems

Emily Ruppel and Brandon Lucia

June 26, 2019

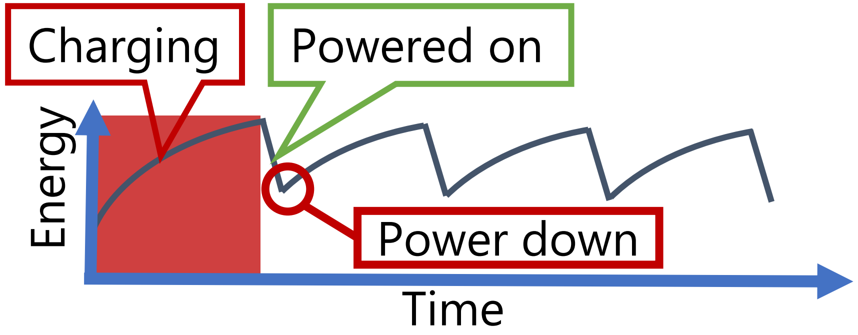
PLDI 2019 -- Systems II

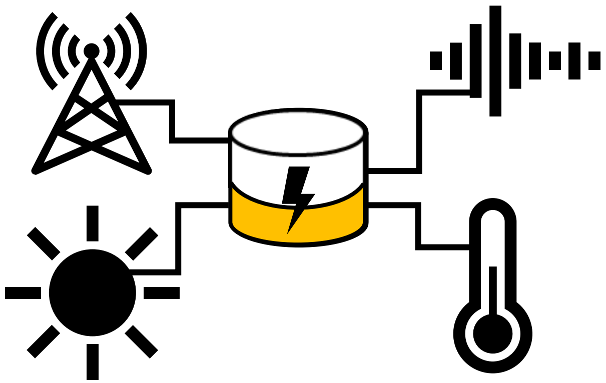
Carnegie Mellon University

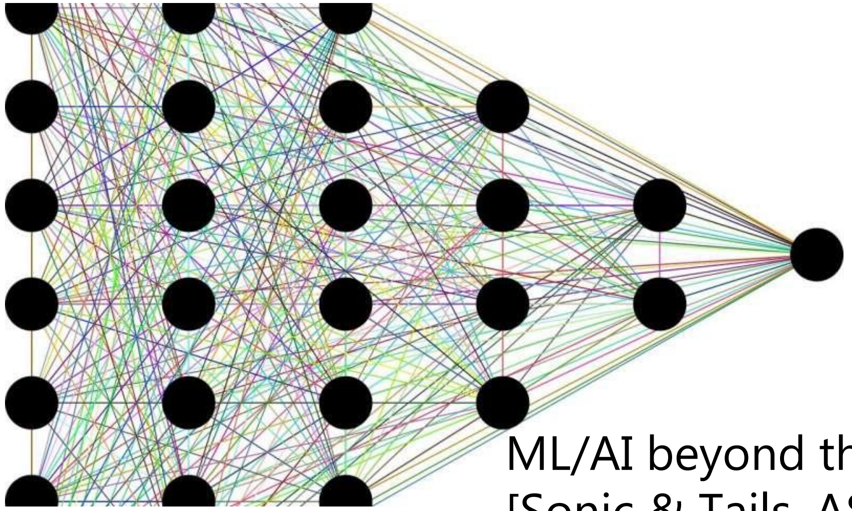


Electrical & Computer ENGINEERING

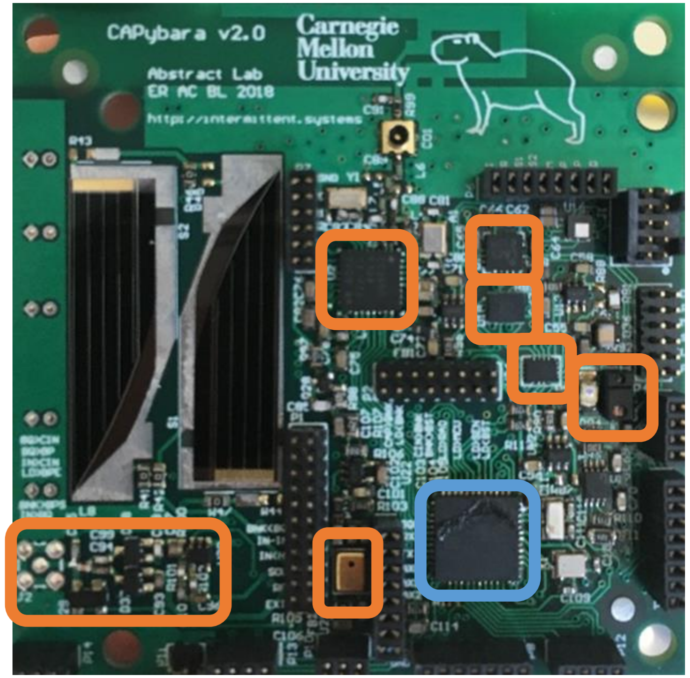
Intermittent Energy-Harvesting Computing Systems







ML/AI beyond the edge
[Sonic & Tails, ASPLOS 2019]

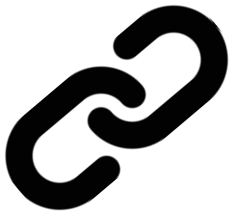


[Capybara, ASPLOS 2018]

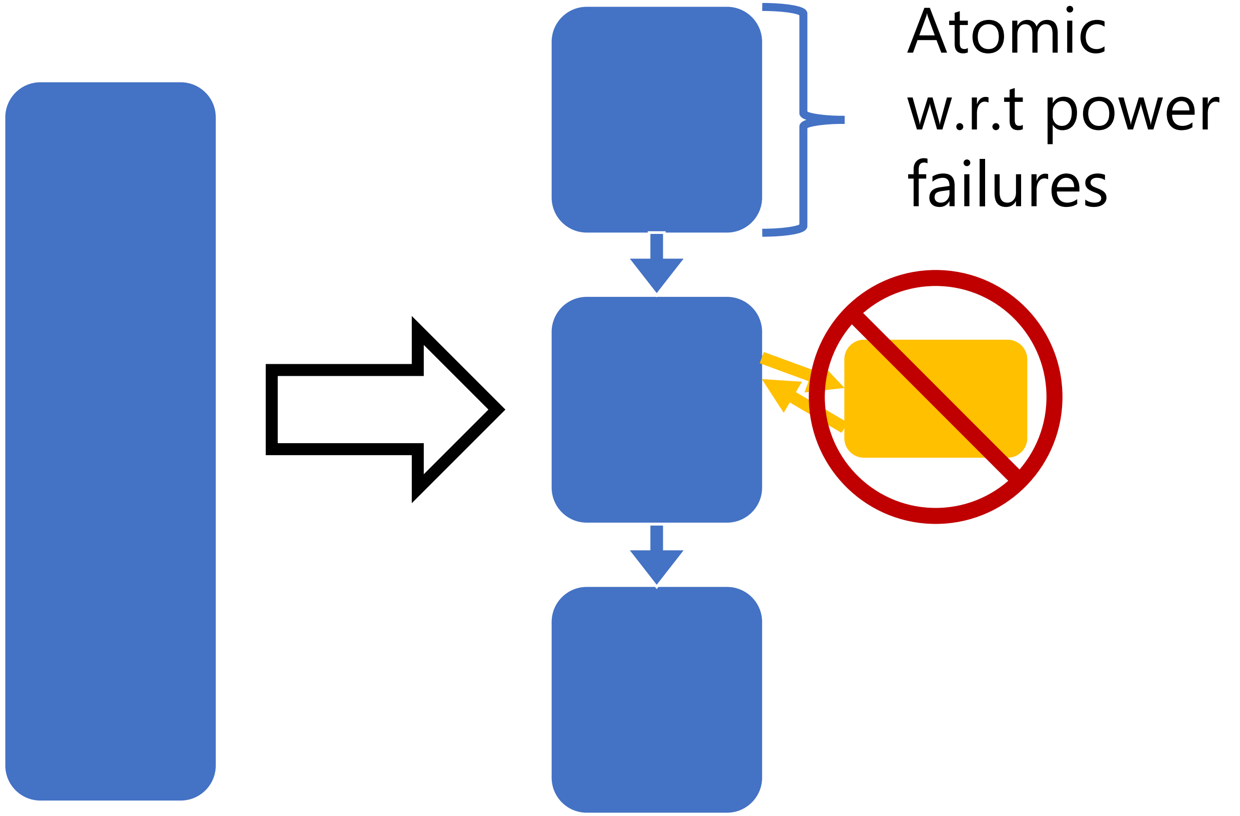
Intermittent devices need concurrent interrupts

Not intermittence safe

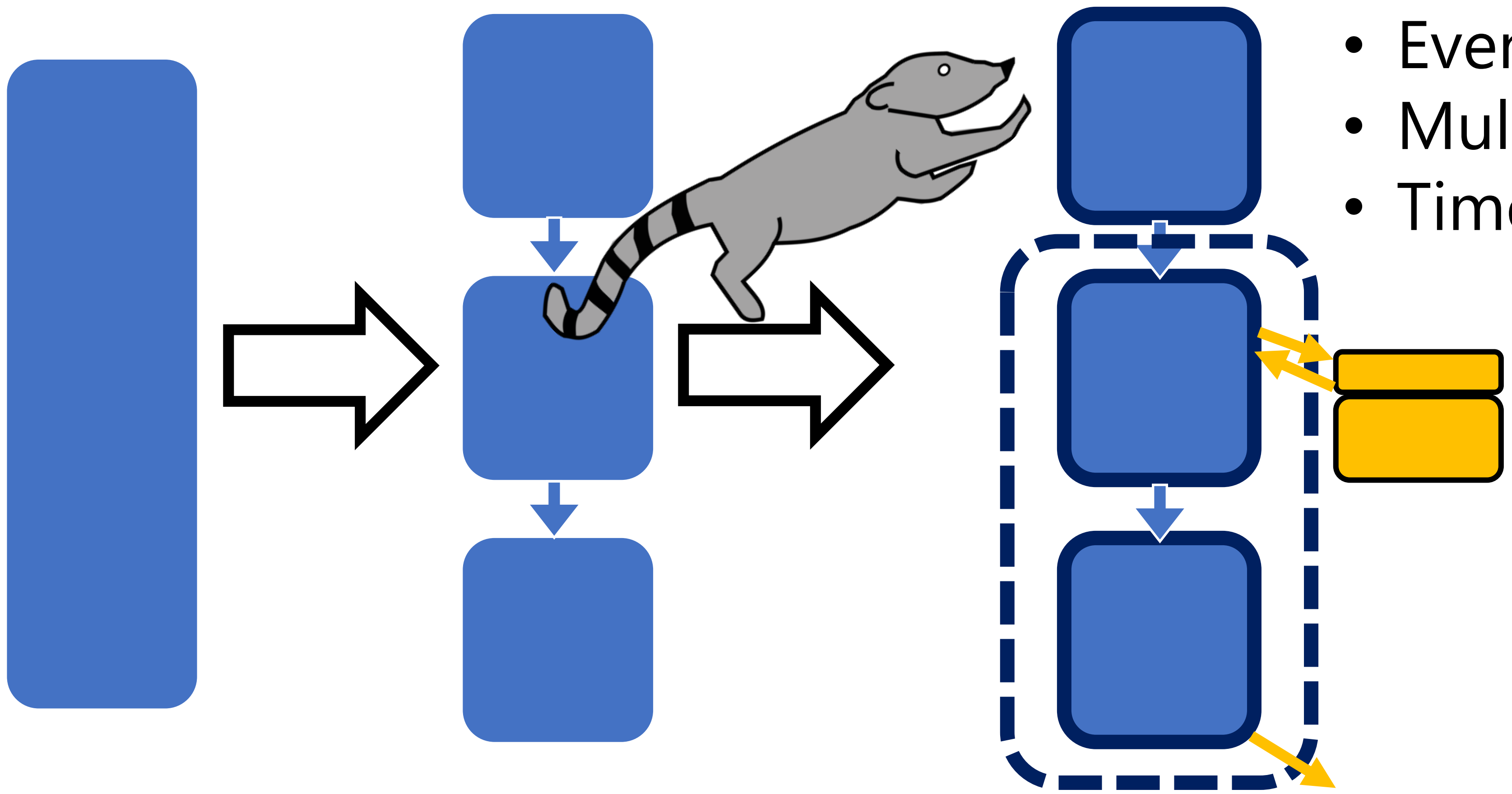
[Alpaca, OOPSLA 2017]



[Chain, OOPSLA 2016]



Intermittent devices need concurrent interrupts



- Event-atomic tasks
- Multi-task transactions
- Timely split-phase events

Coati Motivation

- Intermittent Programming Models & Interrupts
- Problem #1: Interrupt Interrupted
- Problem #2: False Flag

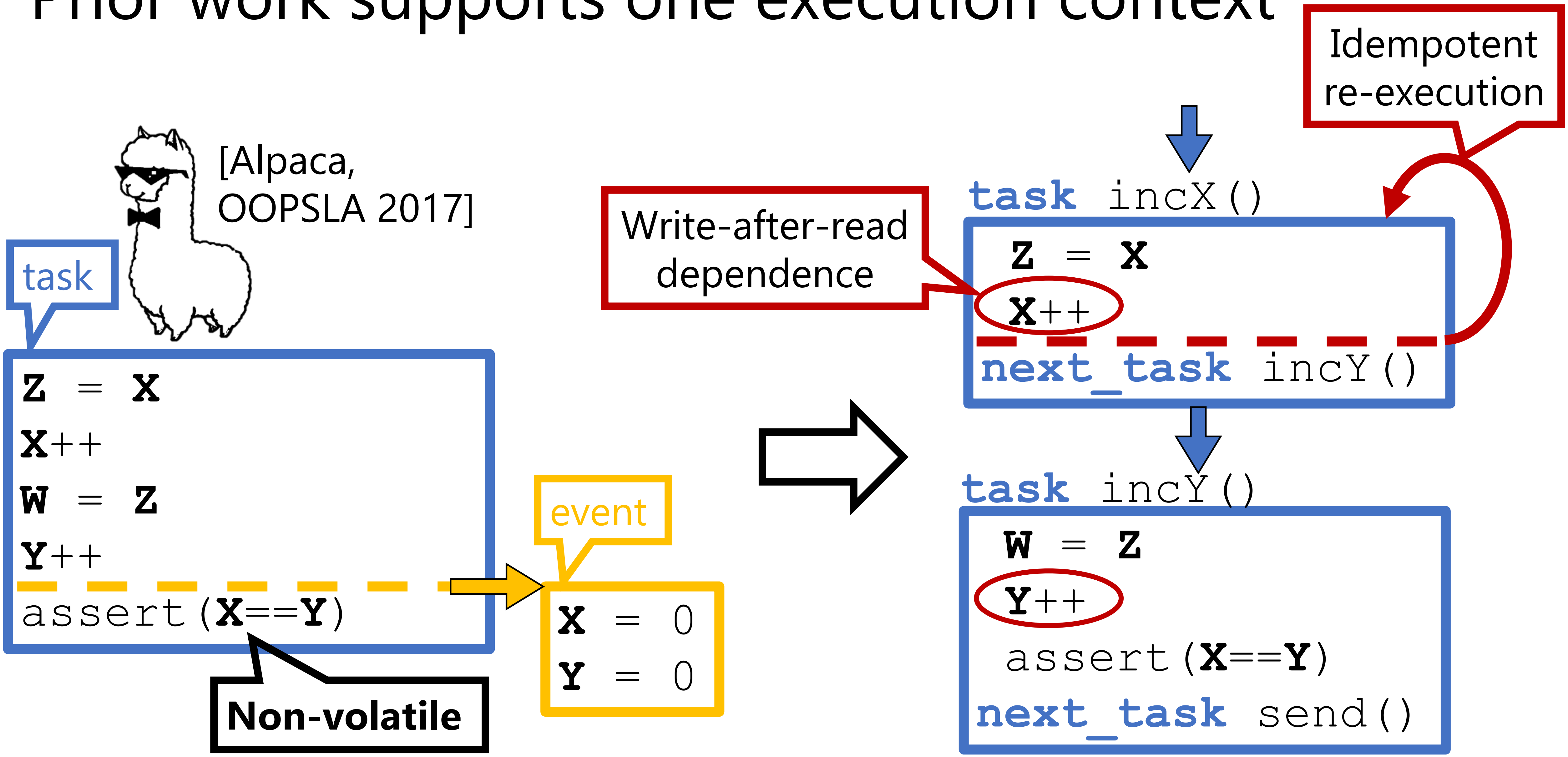
Coati System Design

- Coati Overview
- Task & Event Interaction
- Multi-Task Transactions

Coati Evaluation

- Correctness
- Programming Effort
- Runtime Overhead

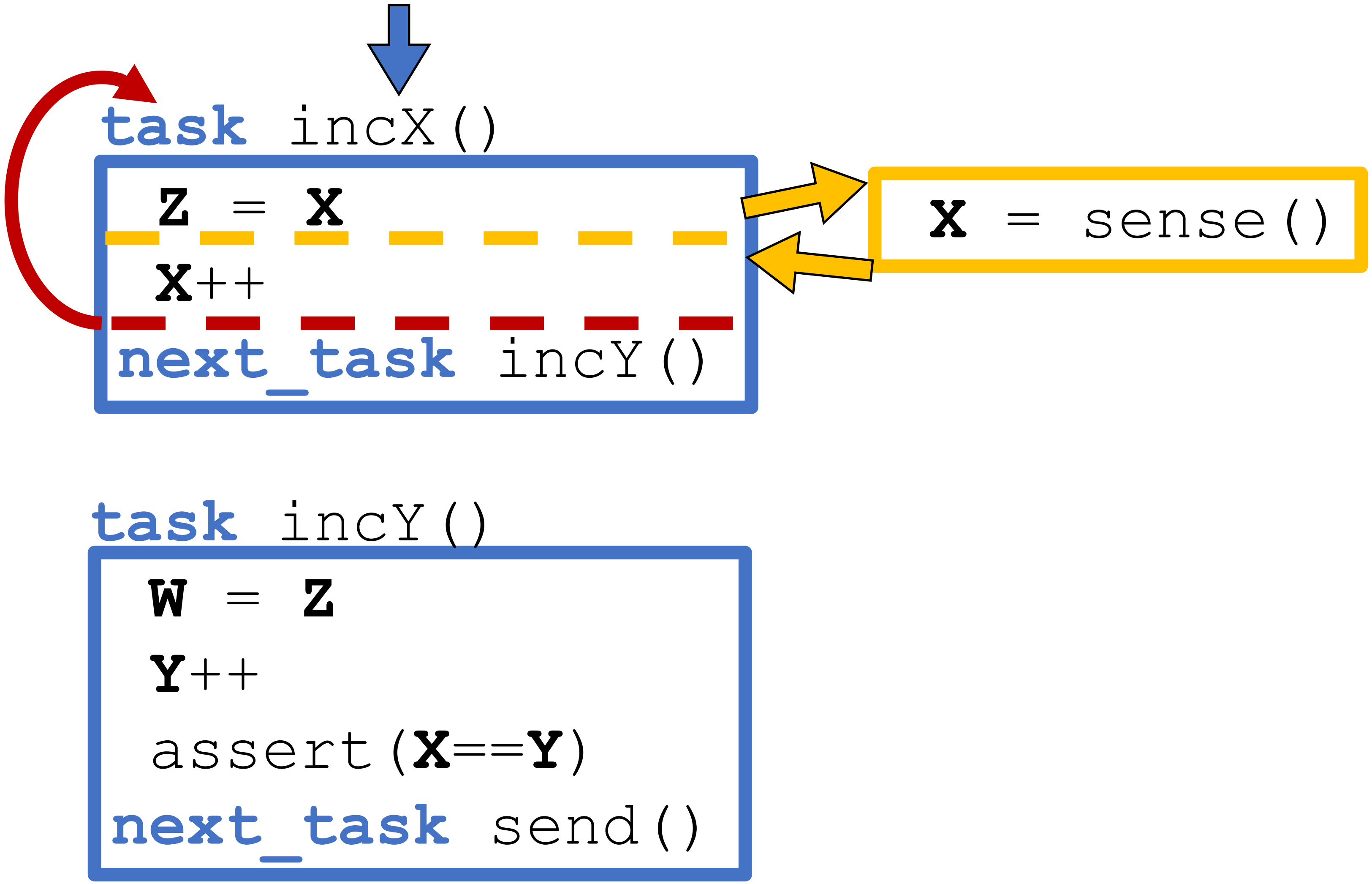
Prior work supports one execution context



Interrupts violate correctness assumptions

Assumptions:

- Tasks always start from the same point
- Tasks re-execute idempotently



[Alpaca, OOPSLA 2017]

Problem #1: Interrupt Interrupted

Power failures can happen during interrupts

X	Y
0	1

task increment()

```
…
t1 = X + 1
…
t2 = Y + 1
assert(t1==t2)
next_task send()
```

event clear()

```
X = 0
Y = 0
return
```

Problem #2: False Flag

Conventional synchronization fails

T1	T2	X	Y	flag
2	1	0	0	1

```
task incX()
  …
  flag = 1
  T1 = X + 1
next_task incY()
```

```
task incY()
  …
  T2 = Y + 1
  assert(T1==T2)
  flag = 0
next_task send()
```

```
event clear()
if(flag) return
  X = 0
  Y = 0
  return
```

Coati Motivation

- Intermittent Programming Models & Interrupts
- Problem #1: Interrupt Interrupted
- Problem #2: False Flag

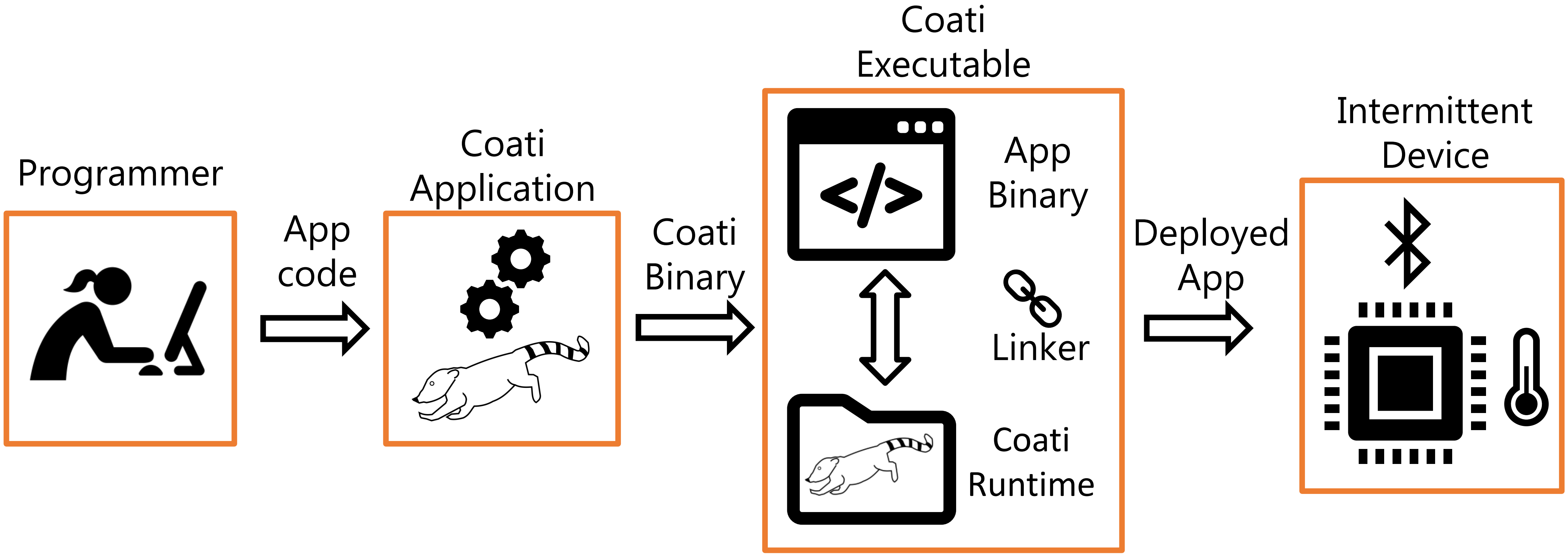
Coati System Design

- Coati Overview
- Task & Event Interaction
- Multi-Task Transactions

Coati Evaluation

- Correctness
- Programming Effort
- Runtime Overhead

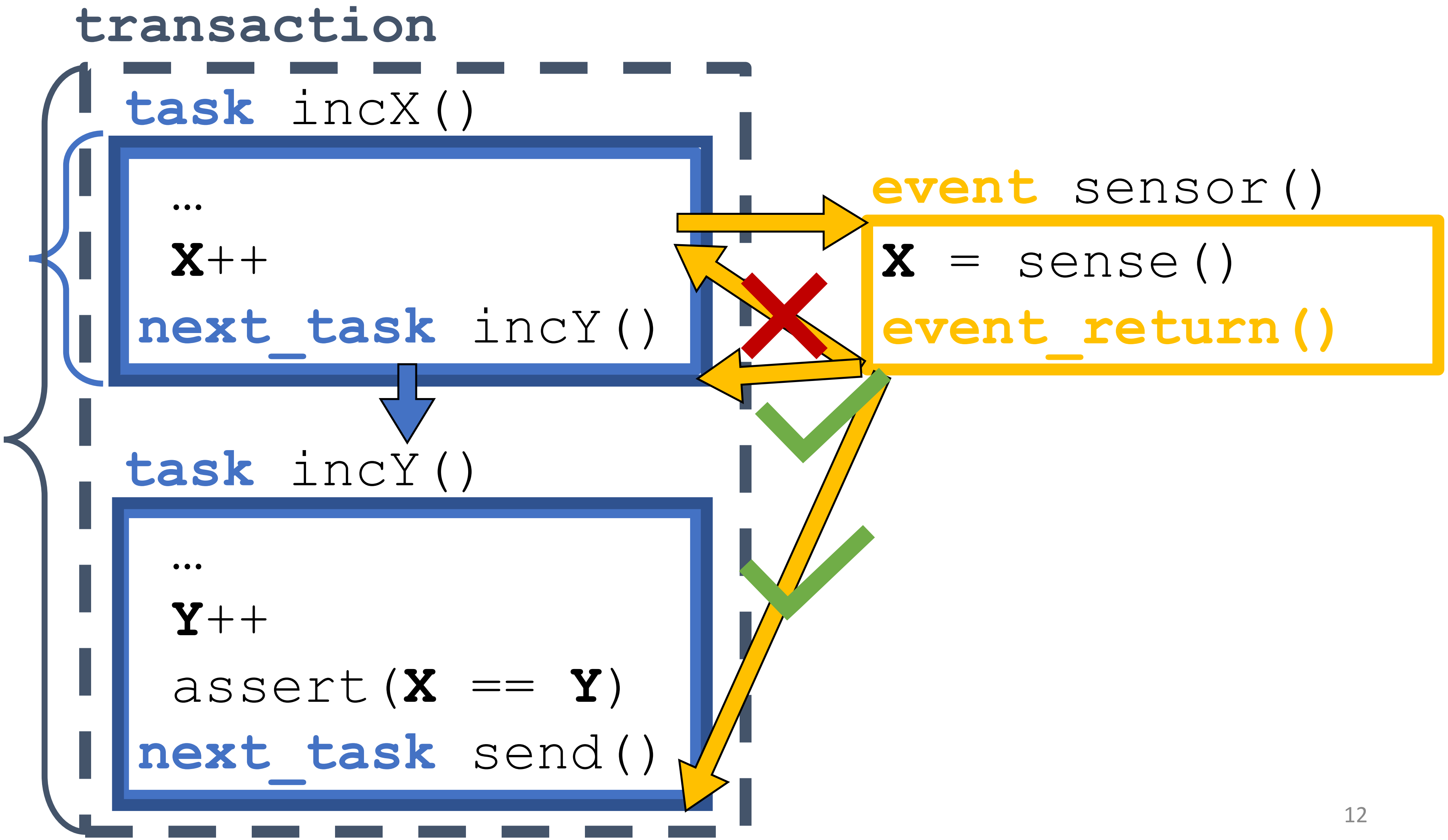
Coati produces correct code with interrupts for intermittent systems



Coati serializes interrupts after scheduled tasks

Atomic w.r.t power failures **& events**

Atomic w.r.t events **not power failures**



Split-phase events provide timely reaction

```
event sensor()
  X = senseX()
  Y = senseY()
  AvgX = movingAvg(X)
  AvgY = movingAvg(Y)
evt_return()
```

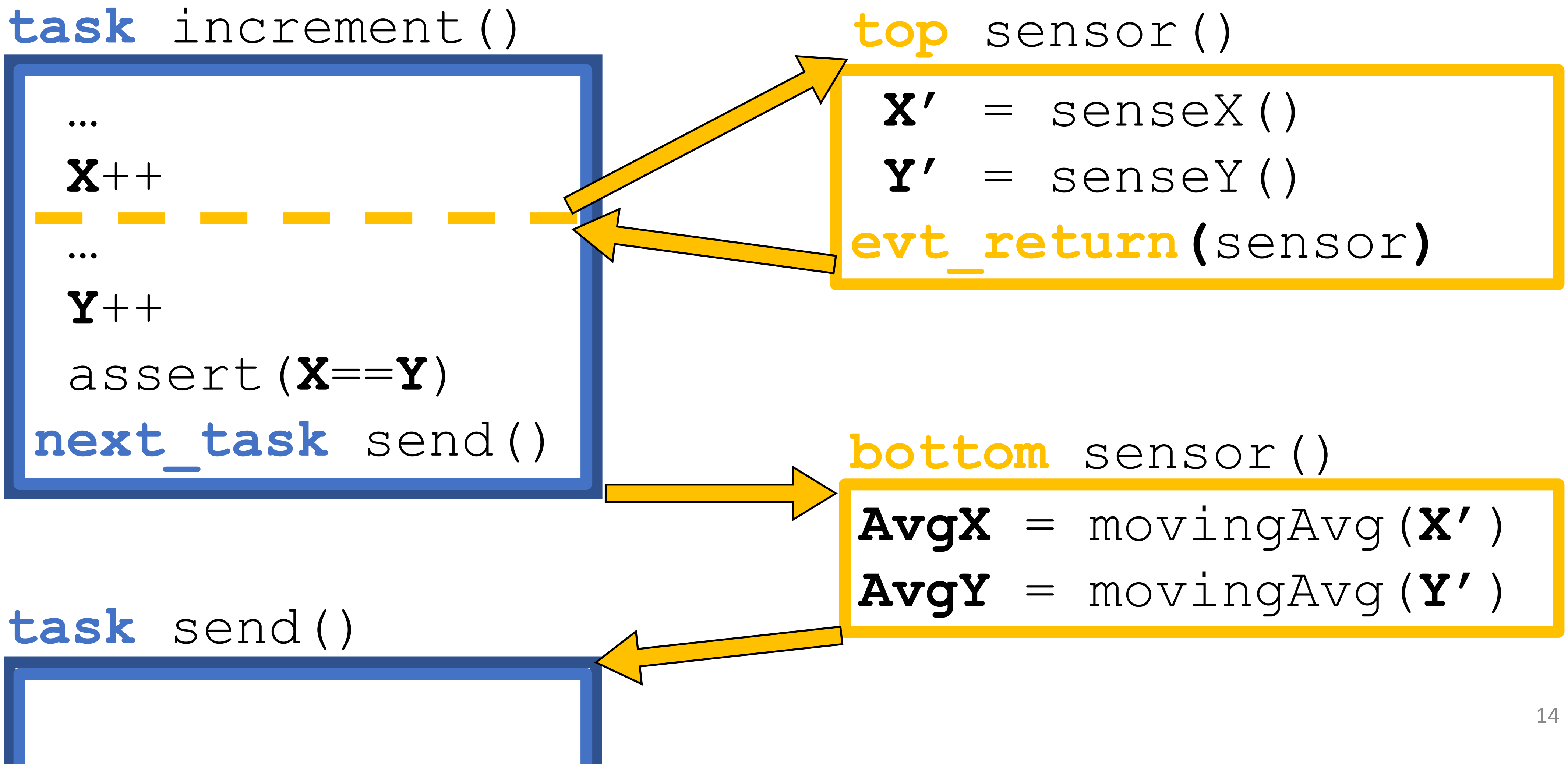
Private Variable

Schedules bottom half

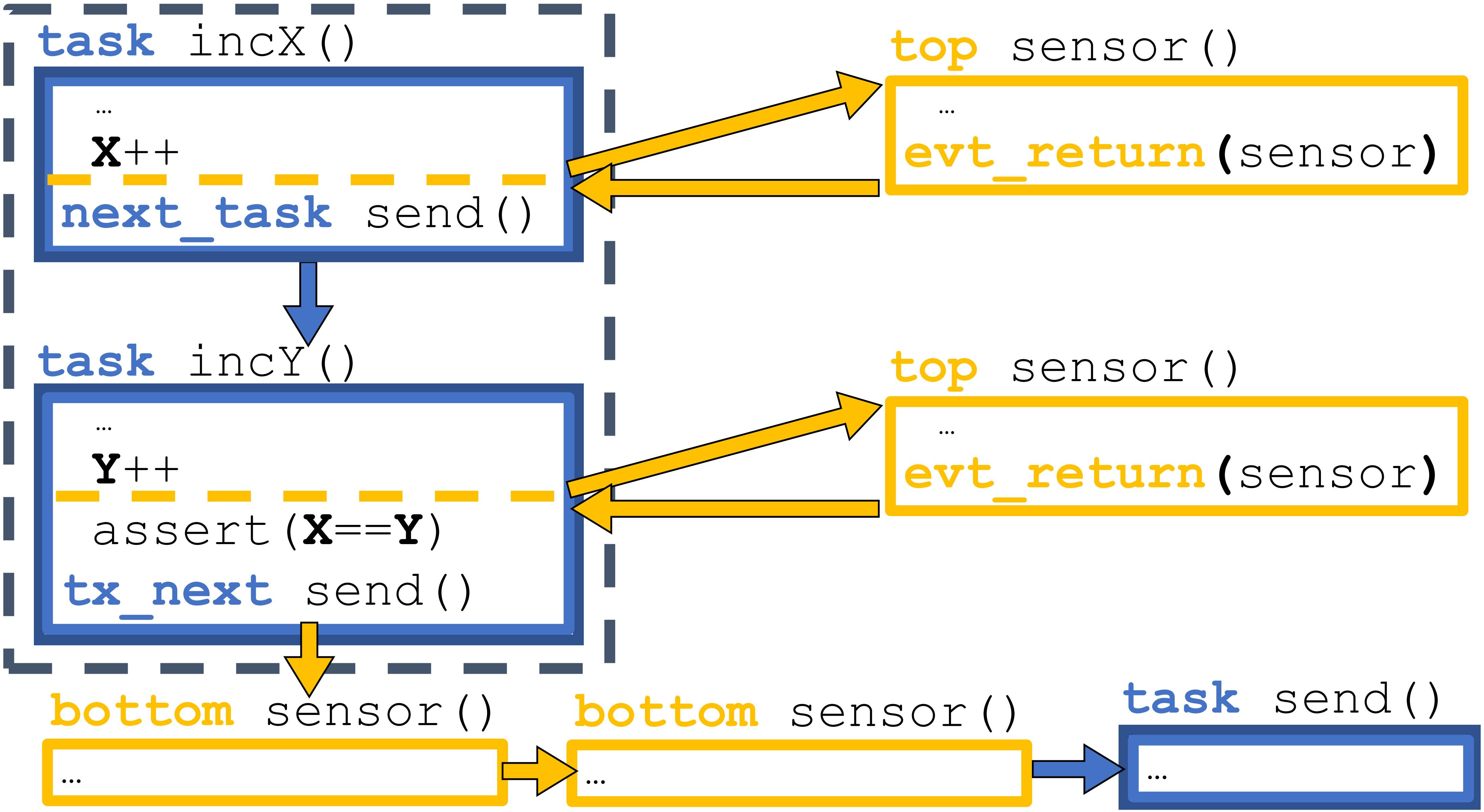
```
top sensor()
  X' = senseX()
  Y' = senseY()
top_return(sensor)
```

```
bottom sensor()
  AvgX = movingAvg(X')
  AvgY = movingAvg(Y')
```

Split-phase serialization orders tasks & events



Coati serializes multiple interrupts in FIFO order



Coati Motivation

- Intermittent Programming Models & Interrupts
- Problem #1: Interrupt Interrupted
- Problem #2: False Flag

Coati System Design

- Coati Overview
- Task & Event Interaction
- Multi-Task Transactions

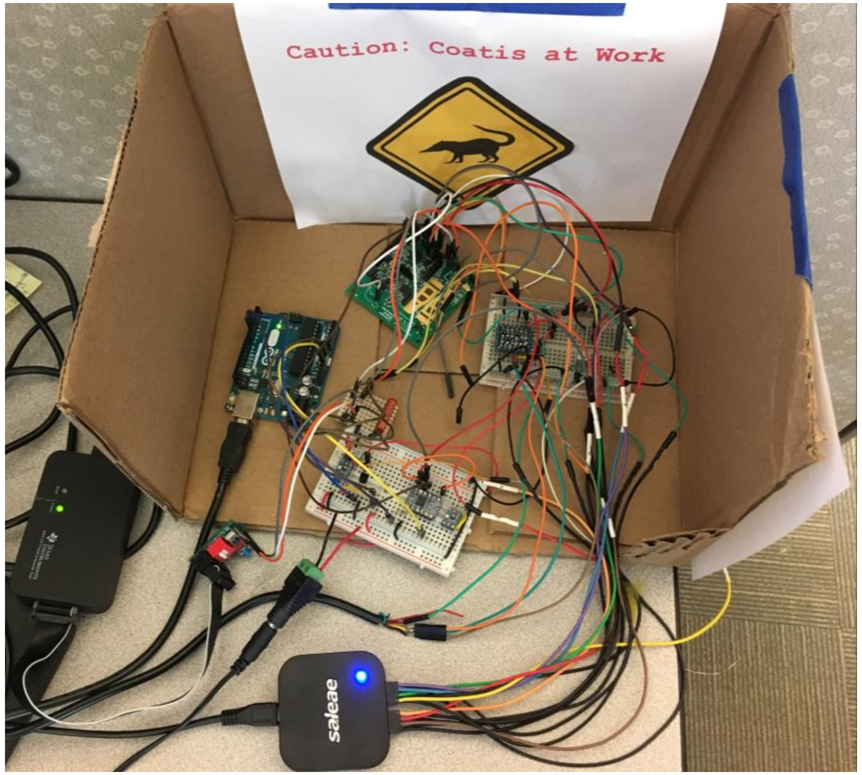
Coati Evaluation

- Correctness
- Programming Effort
- Runtime Overhead

Coati Evaluation

- Correctness
- Programming Effort
- Runtime Overhead

We evaluated Coati on real hardware

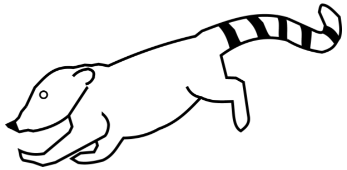


Systems
Coati
Alpaca
Atomic
Hand-Optimized

Apps	
Bitcount	BC
Activity Recognition	AR
RSA Encryption	RSA
Cold-Chain Equipment Monitor	CEM
Cuckoo Filter	CF
Blowfish Encryption	BF

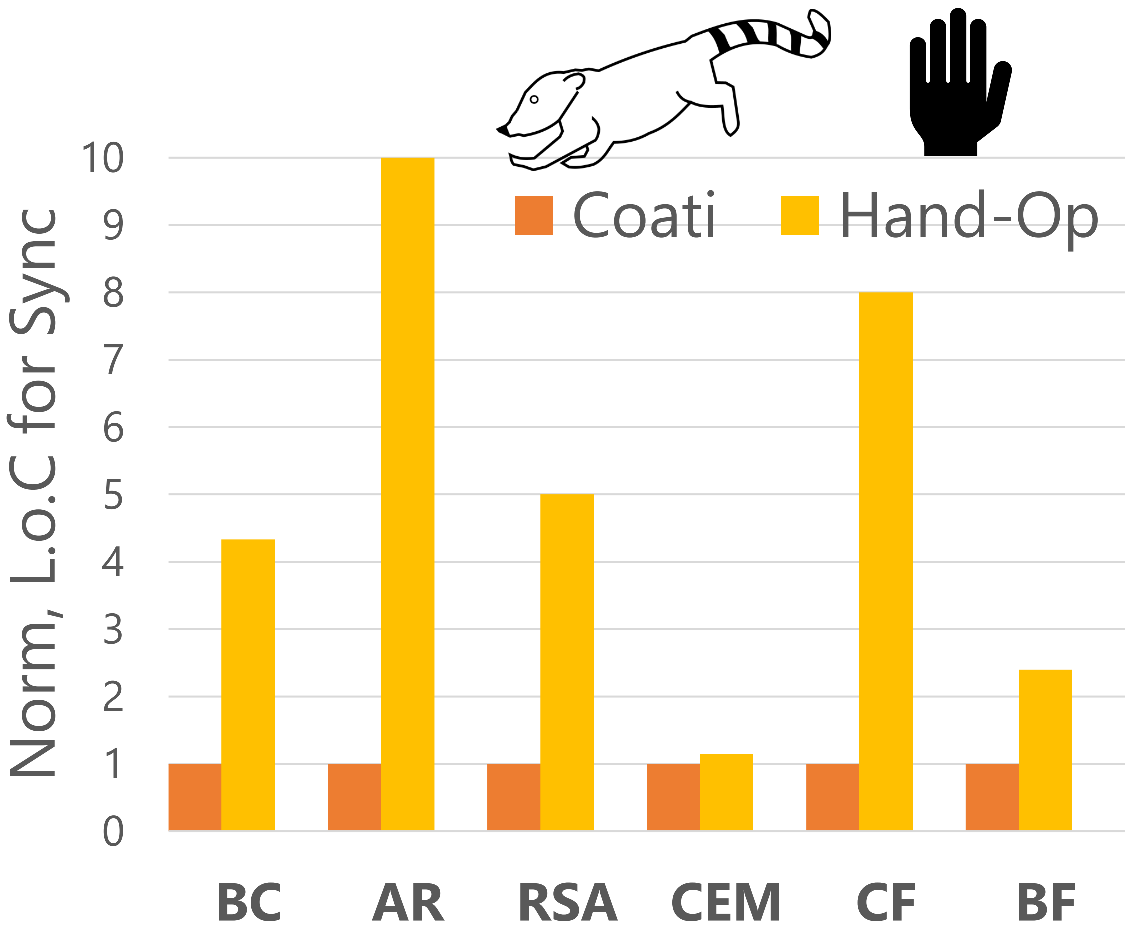
Coati ensures correct execution with interrupts

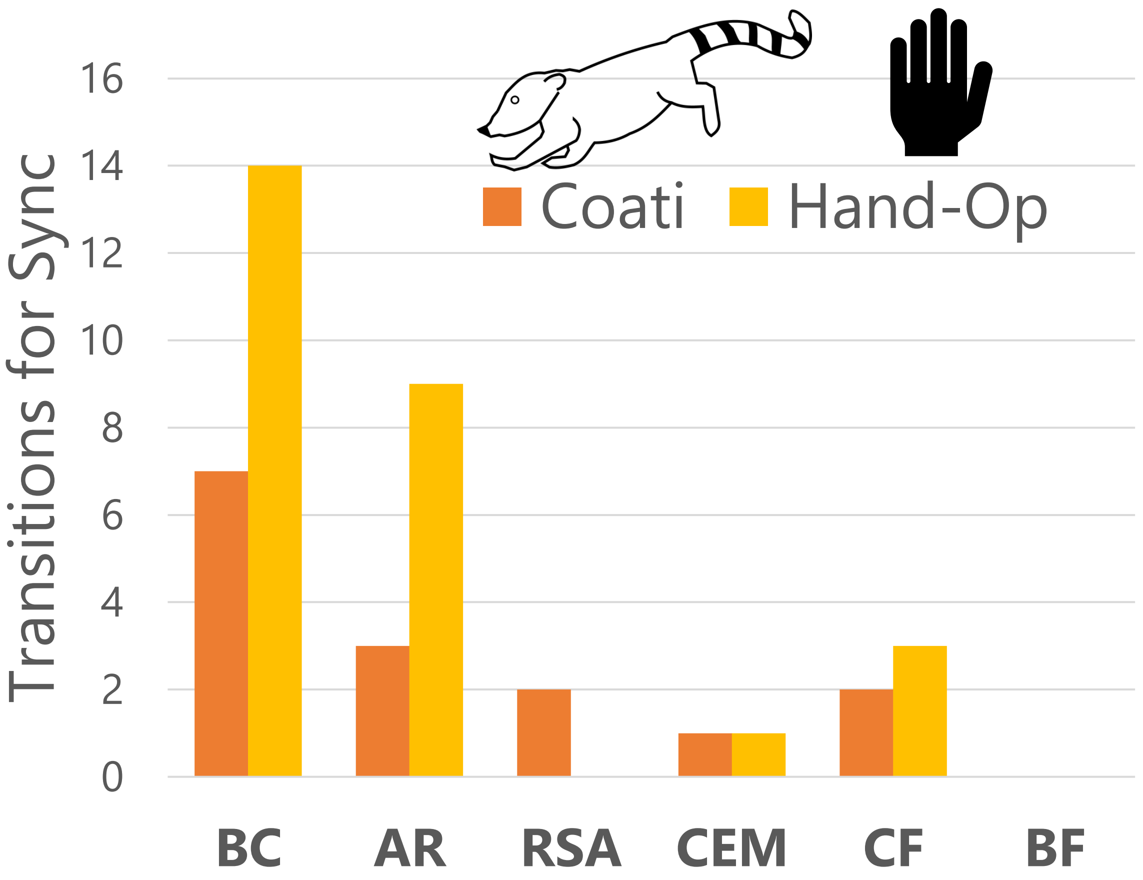
- Synchronization was carefully added to the Alpaca code
- Both systems used the same task decomposition
- Data in Alpaca still become inconsistent



	Coati	Alpaca
BC	✓	✓
AR	✓	✗
RSA	✓	✗
CEM	✓	✗
CF	✓	✗
BF	✓	✗

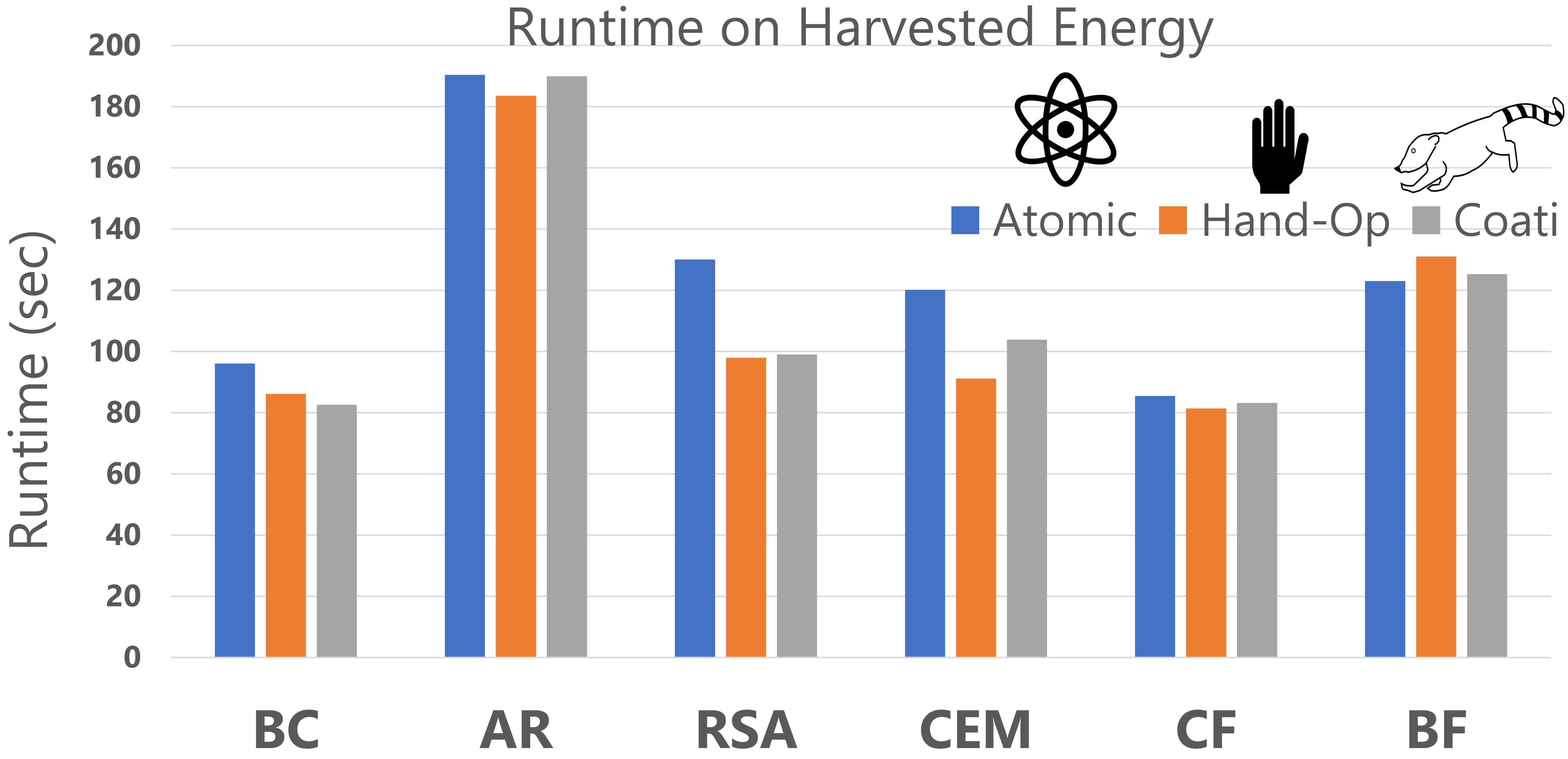
Coati transactions simplify concurrent code





- Lines of synchronization code reduced by 67% on average
- On average, fewer, simpler transitions are required

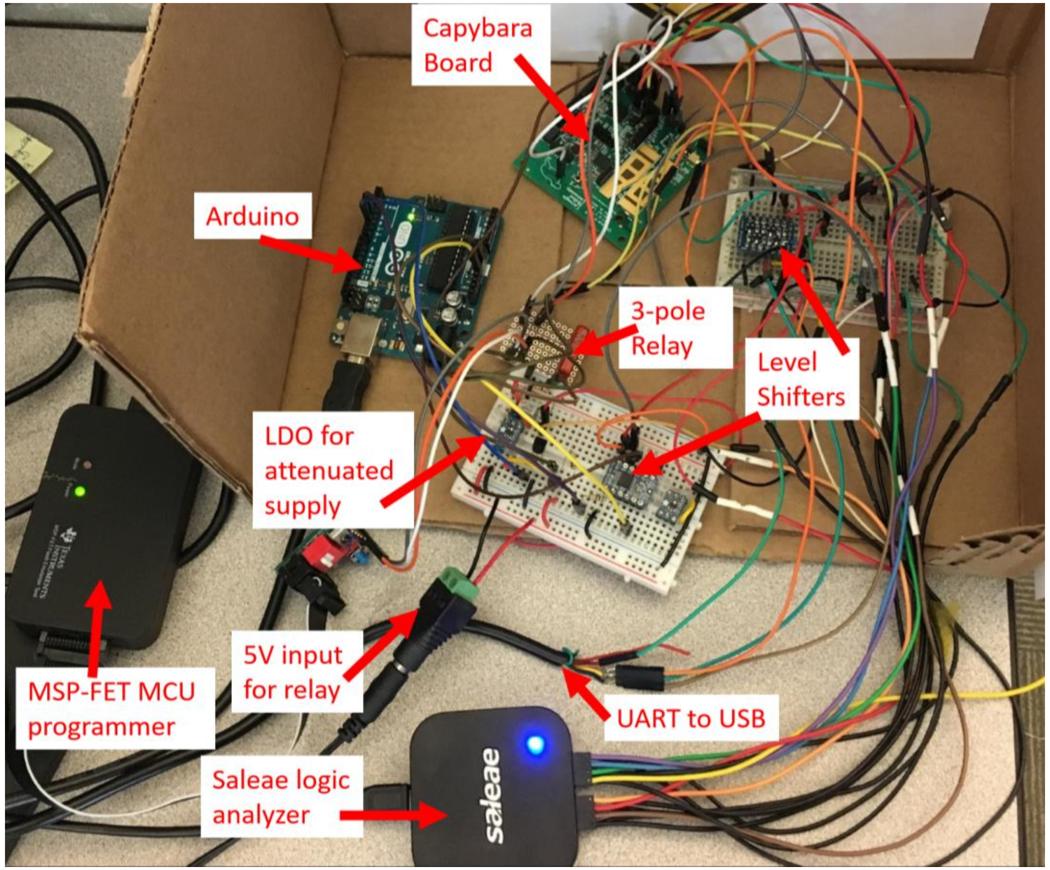
Coati's overhead is similar to other strategies without their drawbacks



Coati provides simple, correct semantics for concurrency control in an intermittent execution

For more:

- Read the paper
- Experiment with our code
- Stick around!



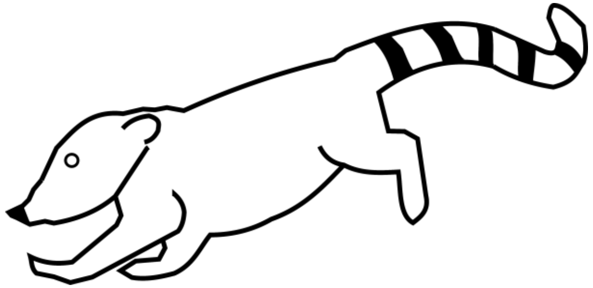
github.com/CMUAbstract/coati_pldi19.git

Transactional Concurrency Control for Intermittent, Energy-Harvesting Computing Systems

Emily Ruppel and Brandon Lucia

June 26, 2019

PLDI 2019 -- Systems II



Carnegie Mellon University

Electrical & Computer ENGINEERING