Transactuations: Where Transactions Meet the Physical World*

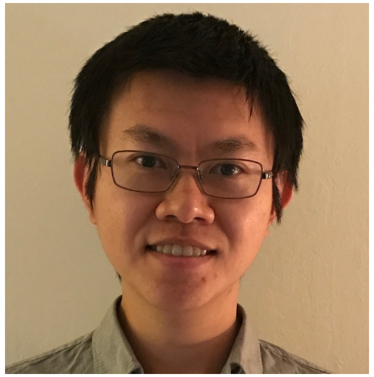
USENIX ATC '19

Aritra Sengupta
(Samsung Research)

Tanakorn Leesatapornwongsa
(Microsoft Research)

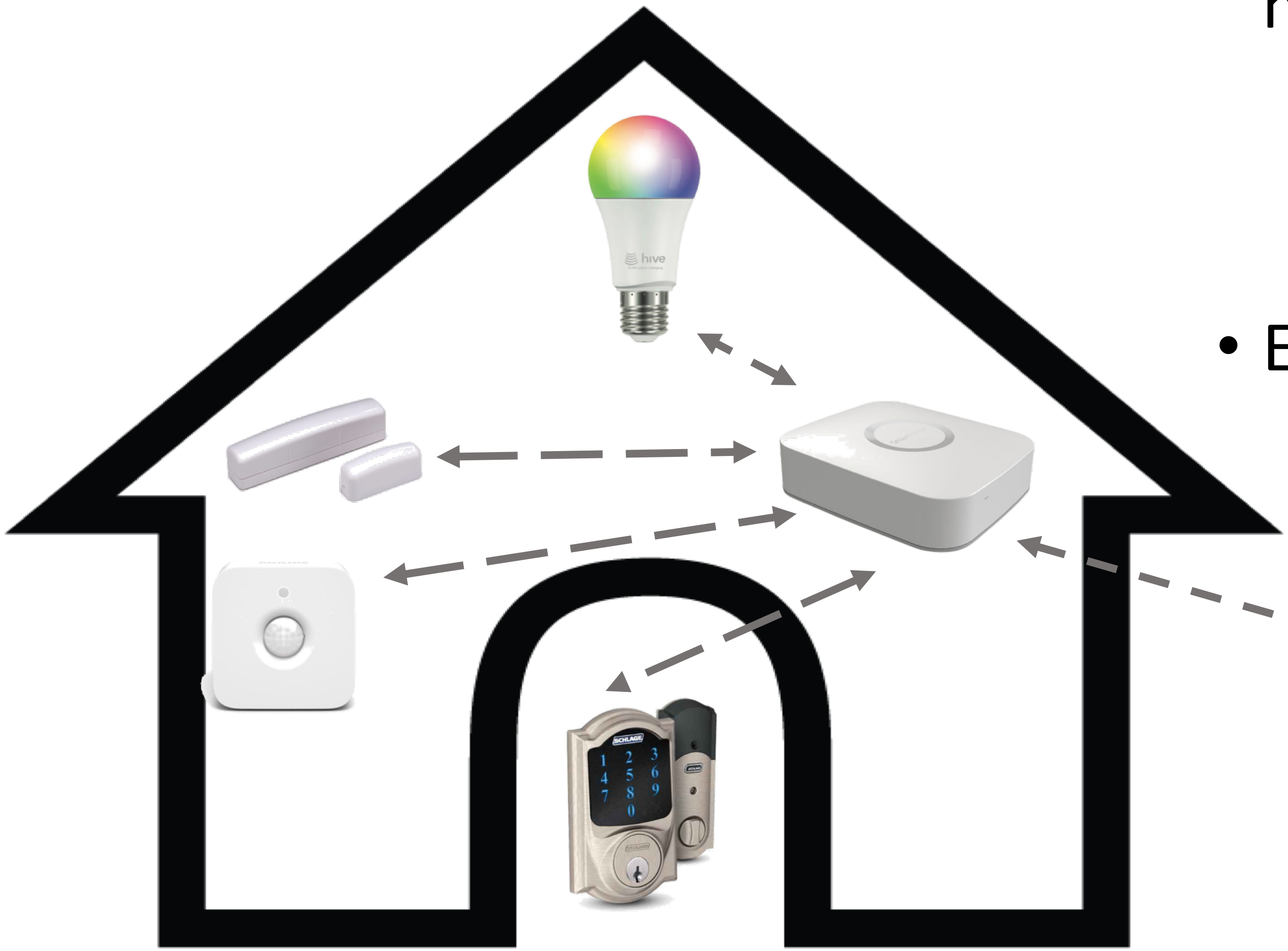
Masoud Saeida Ardekani
(Uber Technologies)

Cesar A. Stuardo
(University of Chicago)

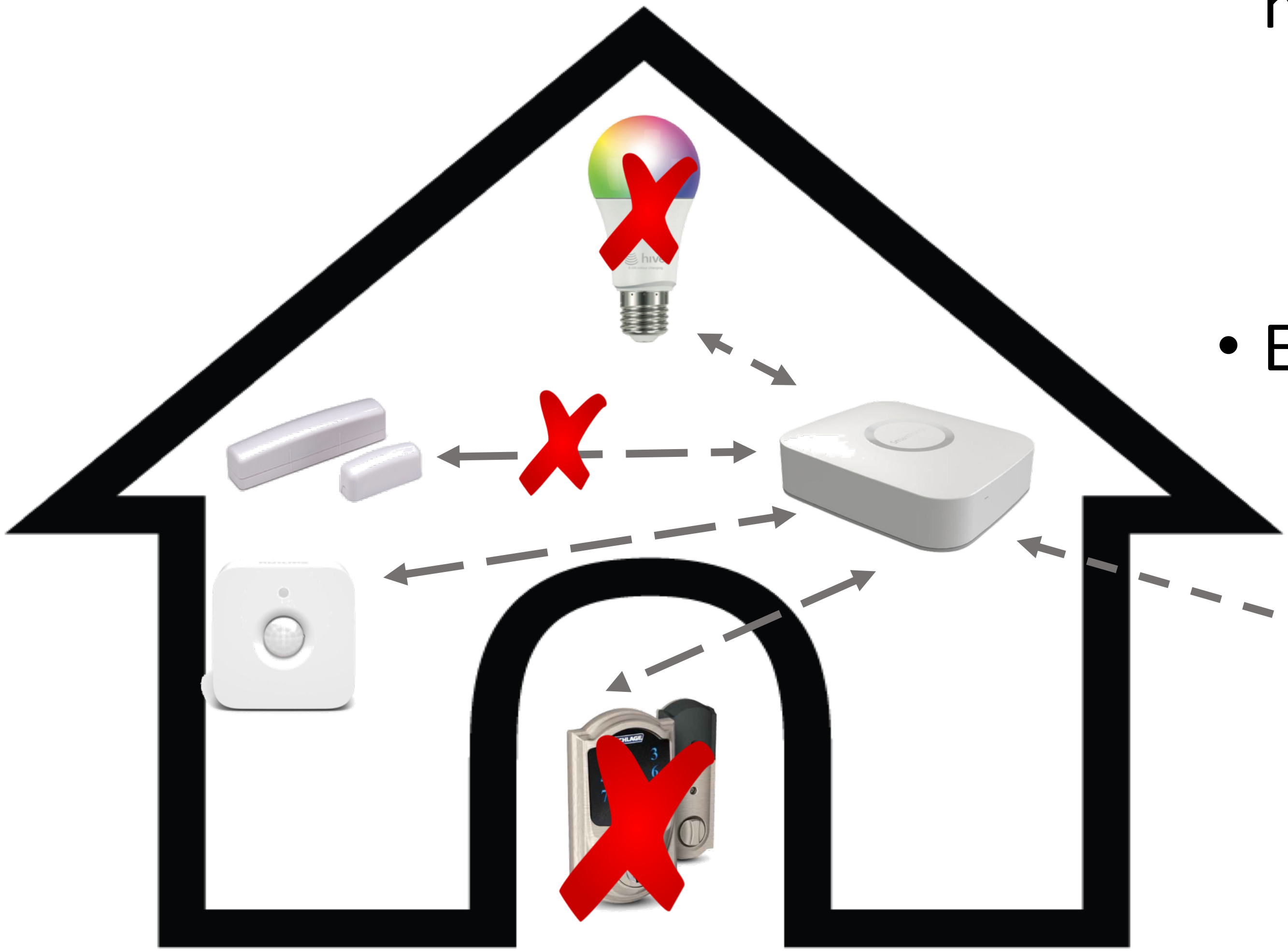



** Work done at Samsung Research America*

- IoT solutions are becoming ubiquitous
- Hundreds of applications for smart homes
    - Automation
    - Security
    - Safety
- Early stage and immature



- IoT solutions are becoming ubiquitous
- Hundreds of applications for smart homes
  - Automation
  - Security
  - Safety
- Early stage and immature



Failure implication goes beyond inconvenience!

When Smart Home Is Not Smart

Inconsistent Behavior [1]

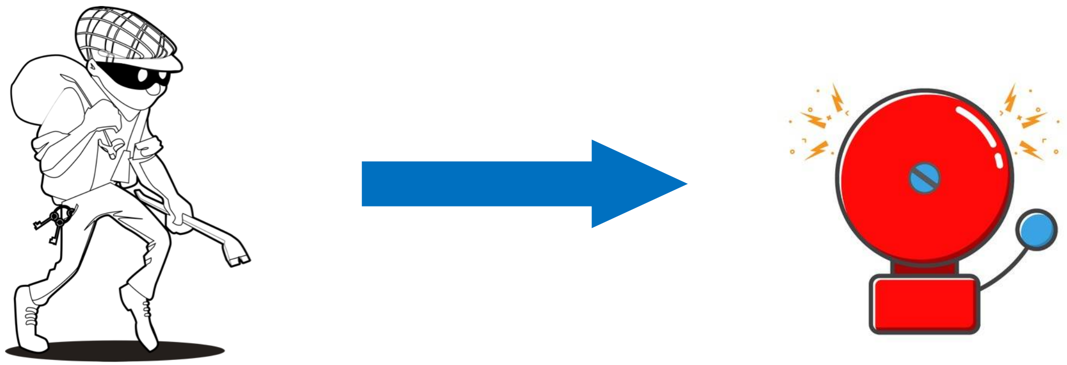
Upset Customer A

*"... More importantly, **we were robbed** when we were out on vacation. I had it set to armed away. The **logs show the motion of the robbers, but it never sounded the alarm** ... I no longer trust it to do what it is supposed to do when it is supposed to do ... "*

[1] SmartThings Community: https://community.smartthings.com

Intrusion Detection Application

```
function handleMotion(evt) {
//isIntruder reads other sensors
//and determines intrusion
  if (isIntruder(evt)
    && !state.alarmActive) {
    alarm.strobe();
    state.alarmActive = true;
  }
}
```



&

!state.alarmActive

state.alarmActive = true

(for avoiding redundant actions)

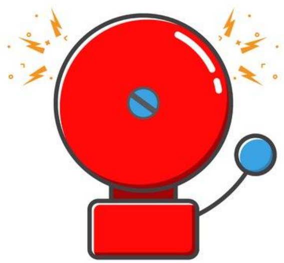
Intrusion Detection Application

```
function handleMotion(evt) {
//isIntruder reads other sensors
//and determines intrusion
  if (isIntruder(evt)
    && !state.alarmActive) {
    alarm.strobe();
    state.alarmActive = true;
  }
}
```

Read sensor and app state

&

`!state.alarmActive`



`state.alarmActive = true`

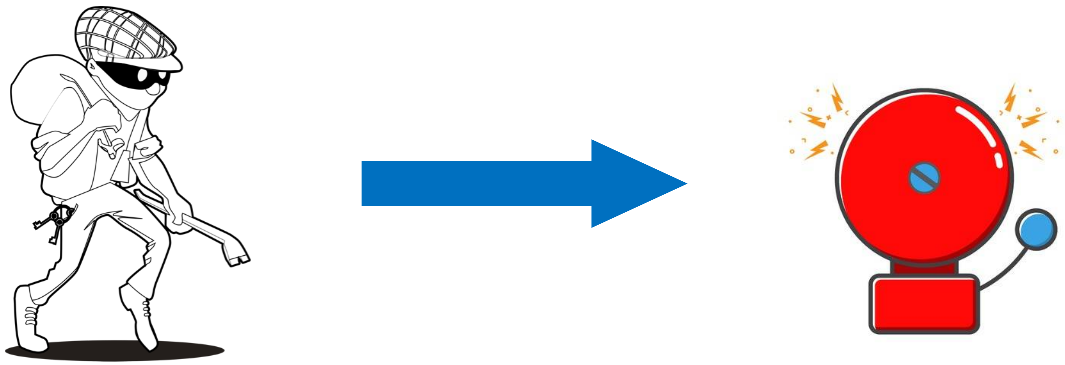
(for avoiding redundant actions)

Intrusion Detection Application

```
function handleMotion(evt) {
//isIntruder reads other sensors
//and determines intrusion
  if (isIntruder(evt)
    && !state.alarmActive) {
    alarm.strobe();
    state.alarmActive = true;
  }
}
```

Read sensor and app state

Actuating a device



`&`
`!state.alarmActive`

`state.alarmActive = true`
(for avoiding redundant actions)

Intrusion Detection Application

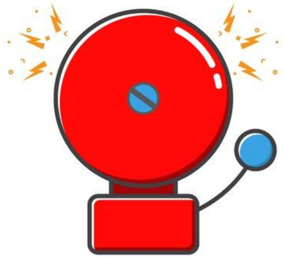
```
function handleMotion(evt) {
//isIntruder reads other sensors
//and determines intrusion
  if (isIntruder(evt)
    && !state.alarmActive) {
    alarm.strobe();
    state.alarmActive = true;
  }
}
```

Read sensor and app state

Actuating a device

Writing app state

& !state.alarmActive



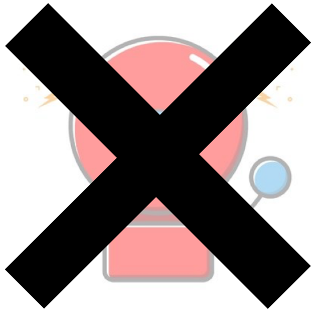
state.alarmActive = true

(for avoiding redundant actions)

Failure Example

&

`!state.alarmActive`

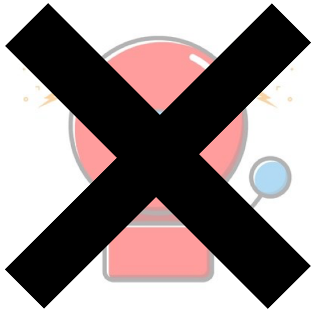


`state.alarmActive = true`

(for avoiding redundant actions)

What if actuation command is lost or a glitch in the alarm?

Failure Example



Inconsistency

&

`!state.alarmActive`

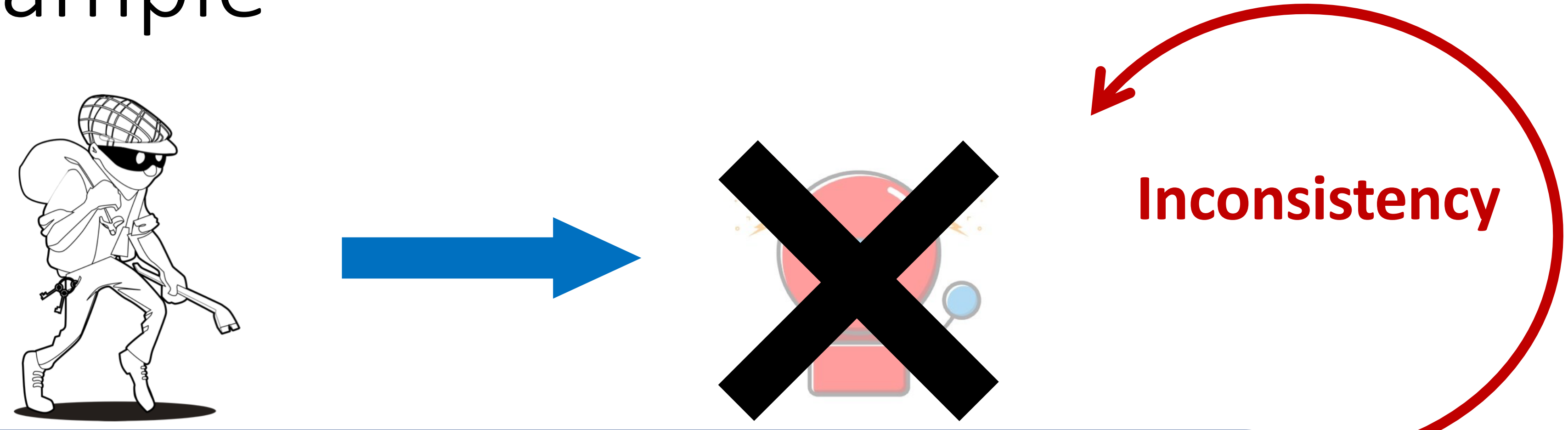
`state.alarmActive = true`

(for avoiding redundant actions)

What if actuation command is lost or a glitch in the alarm?

Physical state != Application state

Failure Example



WARNING

The alarm is based on wireless transmissions … can be subject to <u>RF interference</u>, … cause the alarm to <u>not operate as intended</u> …

?

Failure makes application and device states **inconsistent**

Failure makes application and device states **inconsistent**

Inherent concurrency in applications also leads to inconsistencies

How often can inconsistencies happen?

- Identified 3 classes of dependencies in application logic
- Dependencies capture semantic relationship between app and device
- These 3 dependencies are <u>vulnerable to failures</u>

How often can inconsistencies happen?

- Identified 3 classes of dependencies in application logic
- Dependencies capture semantic relationship between app and device
- These 3 dependencies are <u>vulnerable to failures</u>

By statically analyzing applications for dependencies, we can identify potential inconsistencies in smart applications

Dependency

1. Sensing → actuating

```
c = co2.value()
if (c > threshold){
  fans.on()
}
```

Reading sensor

Actuating based on sensor read

2. Sensing → app state update

```
t = thermo.value()
if (t > 90){
  setMode("HOT")
}
```

3. Actuating → app state update

```
alarm.strobe()

active = "TRUE"
```

Dependency

1. Sensing → actuating

```
c = co2.value()
if (c > threshold){
   fans.on()
}
```

Reading sensor

Actuating based on sensor read

2. Sensing → app state update

```
t = thermo.value()
if (t > 90){
   setMode("HOT")
}
```

Reading sensor

Updating app state based on sensor

3. Actuating → app state update

```
alarm.strobe()

active = "TRUE"
```

Dependency

1. Sensing → actuating

```
c = co2.value()
if (c > threshold){
   fans.on()
}
```

Reading sensor

Actuating based on sensor read

2. Sensing → app state update

```
t = thermo.value()
if (t > 90){
   setMode("HOT")
}
```

Reading sensor

Updating app state based on sensor

3. Actuating → app state update

```
alarm.strobe()

active = "TRUE"
```

Actuating device

Updating app state tied to device

Can Transactions address the problem?

NO

Can Transactions address the problem?

NO

- IoT devices cannot be locked
  - Users can observe intermediate value

Can Transactions address the problem?

NO

- IoT devices cannot be locked
  - Users can observe intermediate value
- Rolling back IoT devices have consequences
  - A user observes a door locks then rolls back to unlocked
  - Not a good user experience!

Can Transactions address the problem?

NO

- IoT devices cannot be locked
  - Users can observe intermediate value
- Rolling back IoT devices have consequences
  - A user observes a door locks then rolls back to unlocked
  - Not a good user experience!
- Some actuations cannot be rolled back
  - Undoing a water dispenser

Transactuation

- High level abstraction and programming model
  - Allows a developer to read/write from/to devices
  - Failure-aware association of application and device states

Transactuation

- High level abstraction and programming model
  - Allows a developer to read/write from/to devices
  - Failure-aware association of application and device states
- Atomic durability for application states
  - Actuations never roll back

Transactuation

- High level abstraction and programming model
  - Allows a developer to read/write from/to devices
  - Failure-aware association of application and device states
- Atomic durability for application states
  - Actuations never roll back
- (Internal) atomic visibility among transactuations
  - External atomic visibility cannot be guaranteed for end users!
  - Disallows several concurrency related bugs

Transactuation

- High level abstraction and programming model
  - Allows a developer to read/write from/to devices
  - Failure-aware association of application and device states
- Atomic durability for application states
  - Actuations never roll back
- (Internal) atomic visibility among transactuations
  - External atomic visibility cannot be guaranteed for end users!
  - Disallows several concurrency related bugs
- Guarantees two invariants

Transactuation

- High level abstraction and programming model
  - Allows a developer to read/write from/to devices
  - Failure-aware association of application and device states
- Atomic durability for application states
  - Actuations never roll back
- (Internal) atomic visibility among transactuations
  - External atomic visibility cannot be guaranteed for end users!
  - Disallows several concurrency related bugs
- Guarantees two invariants

Sensing Invariant	**Actuating Invariant**
Governs executing a transactuation	Governs committing a transactuation

Sensing Invariant
Transactuation executes only when staleness of its sensor reads is bounded, as per specified sensing policy

Sensing policy
How much staleness is acceptable
How many failed sensors is acceptable

Example of sensing policy
at least one co2 sensor can be read within last 5 mins

Actuating Invariant

When a transactuation commits its app states, sufficient number of actuations have succeeded as per specified actuation policy

Actuation policy

How many failed actuation is acceptable

Example of actuation policy

At least one alarm should successfully turn on

Simplified Example

```
(sensors) => {
  let active = read('active');
  if (sensors['co2'] > threshold && !read('active')) {
    actuate('fans', 'on');
    write('active', true);
  }
  ...
}
```

Application logic

Simplified Example

```
let tx = new Transactuation();



tx.perform(
   (sensors) => {
     let active = read('active');
     if (sensors['co2'] > threshold && !read('active')) {
       actuate('fans', 'on');
       write('active', true);
     }
     ...
   }
);
```

Application logic

Simplified Example

Sensing policy

Application logic

```
let tx = new Transactuation();

tx.perform(['co2'], 5m, 'sense_all'
  (sensors) => {
    let active = read('active');
    if (sensors['co2'] > threshold && !read('active')) {
      actuate('fans', 'on');
      write('active', true);
    }
    ...
  }
);
```

Simplified Example

```
let tx = new Transactuation();
```

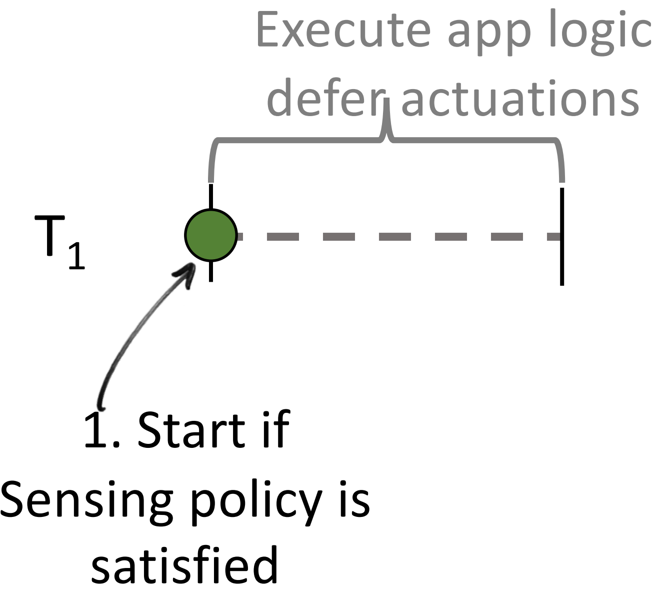
Sensing policy

Actuating policy

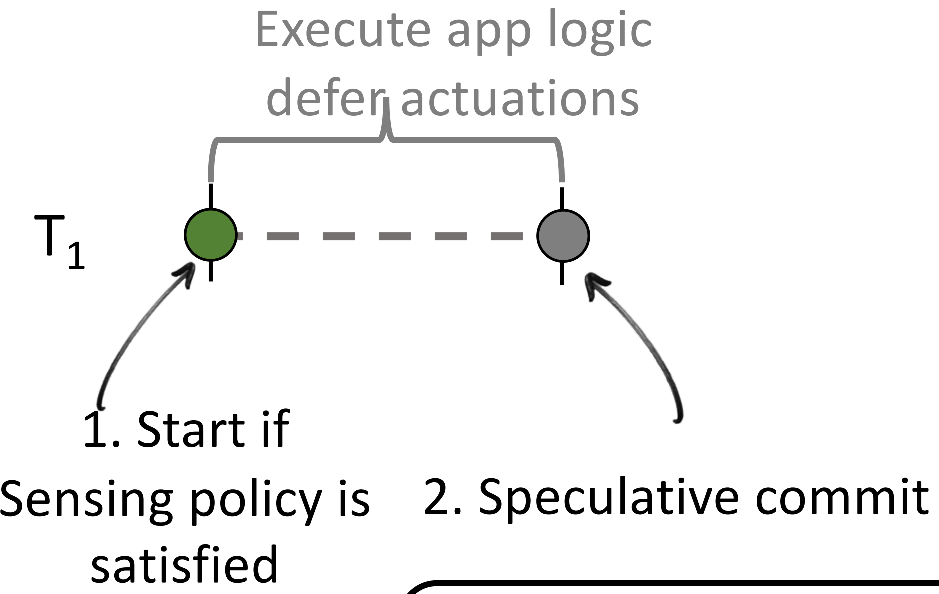
```
tx.perform(['co2'], 5m, 'sense_all', 'act_all',
    (sensors) => {
      let active = read('active');
      if (sensors['co2'] > threshold && !read('active')) {
        actuate('fans', 'on');
        write('active', true);
      }
      ...
    }
);
```

Application logic

Execution Model



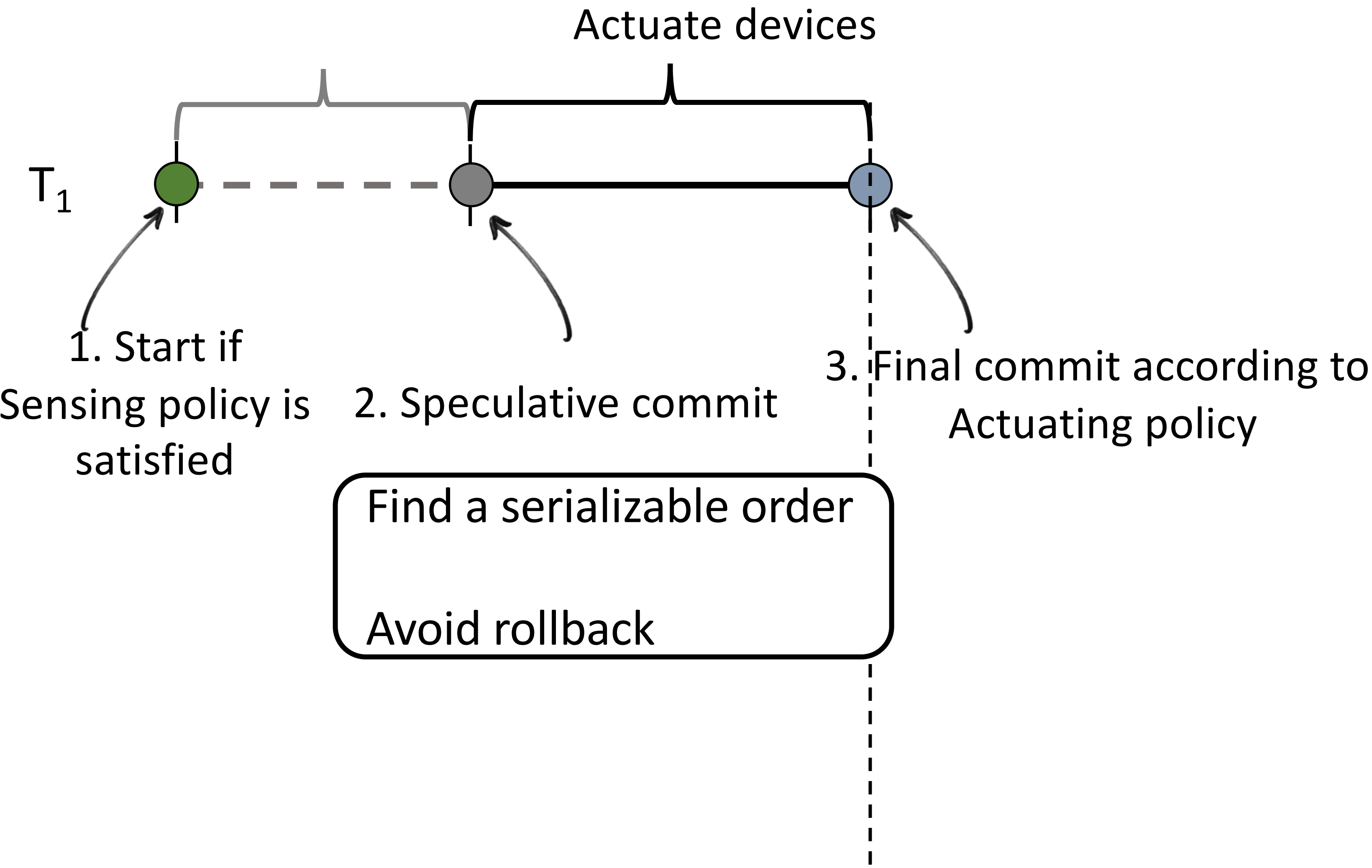
Execution Model



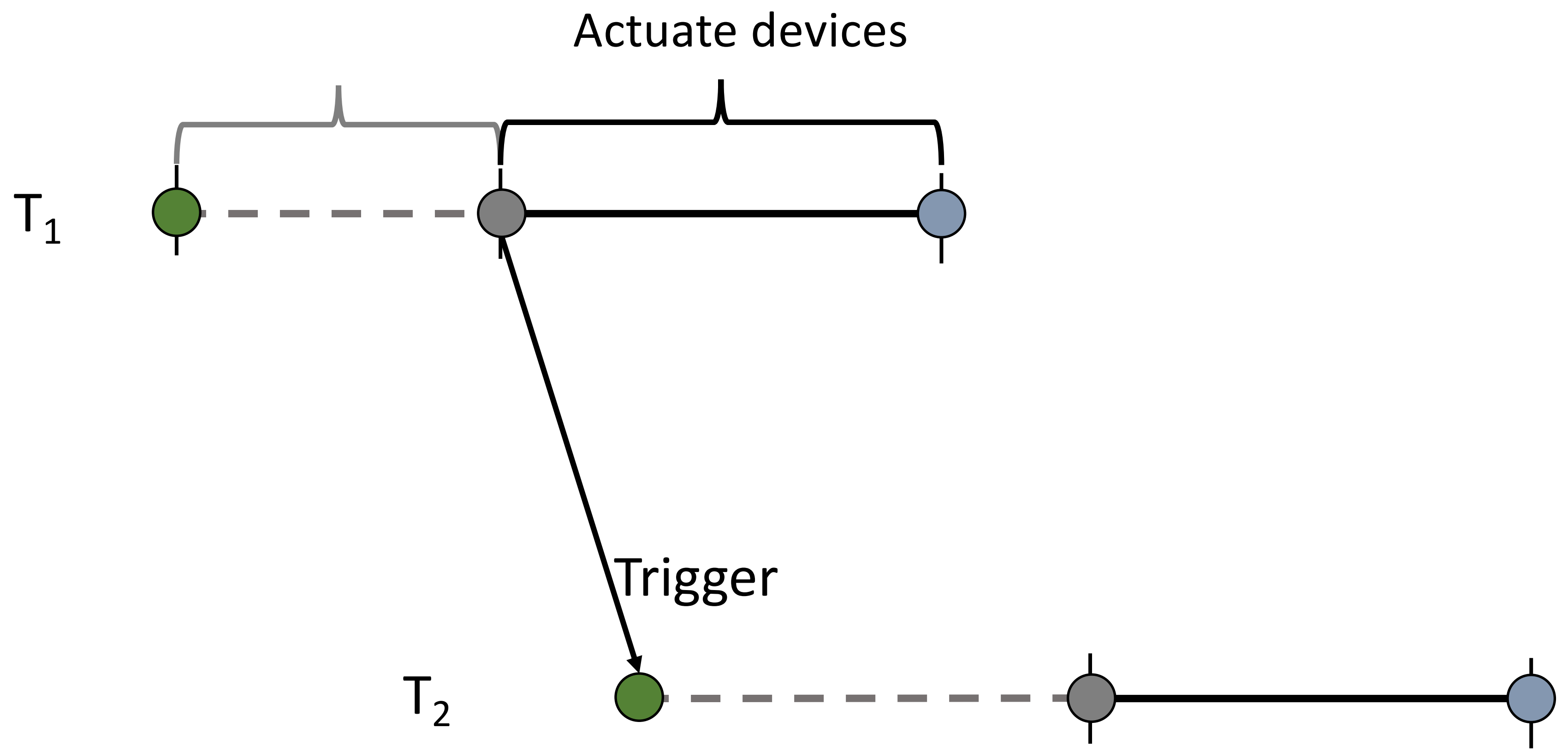
Find a serializable order

Avoid rollback

Execution Model

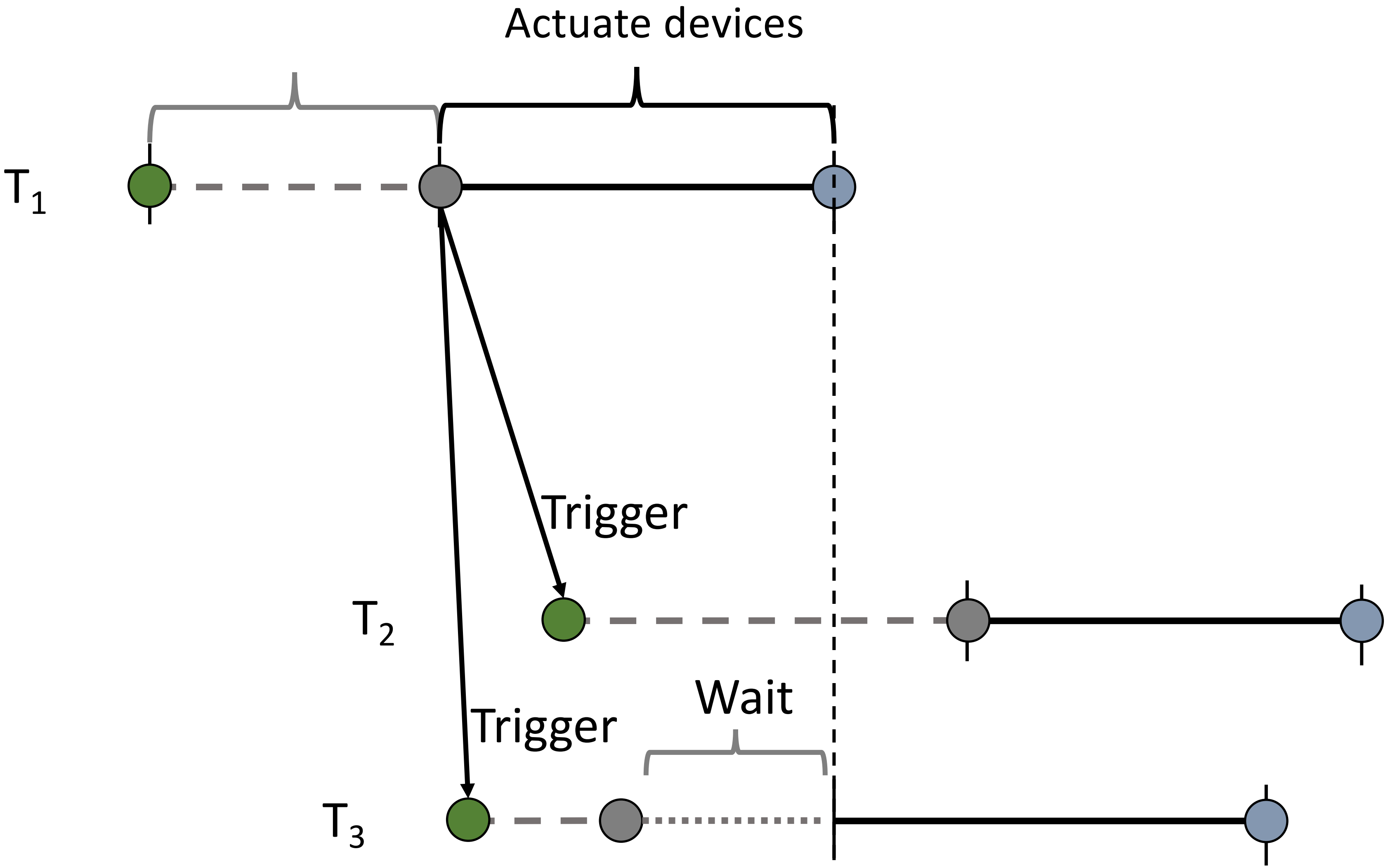


Execution Model



Overlapping computation and actuation

Execution Model



Overlapping computation and actuation

Implementation: Relacs

- Runtime called Relacs is built on Azure technology
  - Azure Functions (serverless functions)
  - Cosmos DB (Relacs store)

- Integrated to Samsung SmartThings IoT platform

Evaluation

- <u>Programmability</u>
- Correctness
- <u>Runtime overhead without failures</u>
- Runtime overhead with failures

Programmability

Lines of Codes

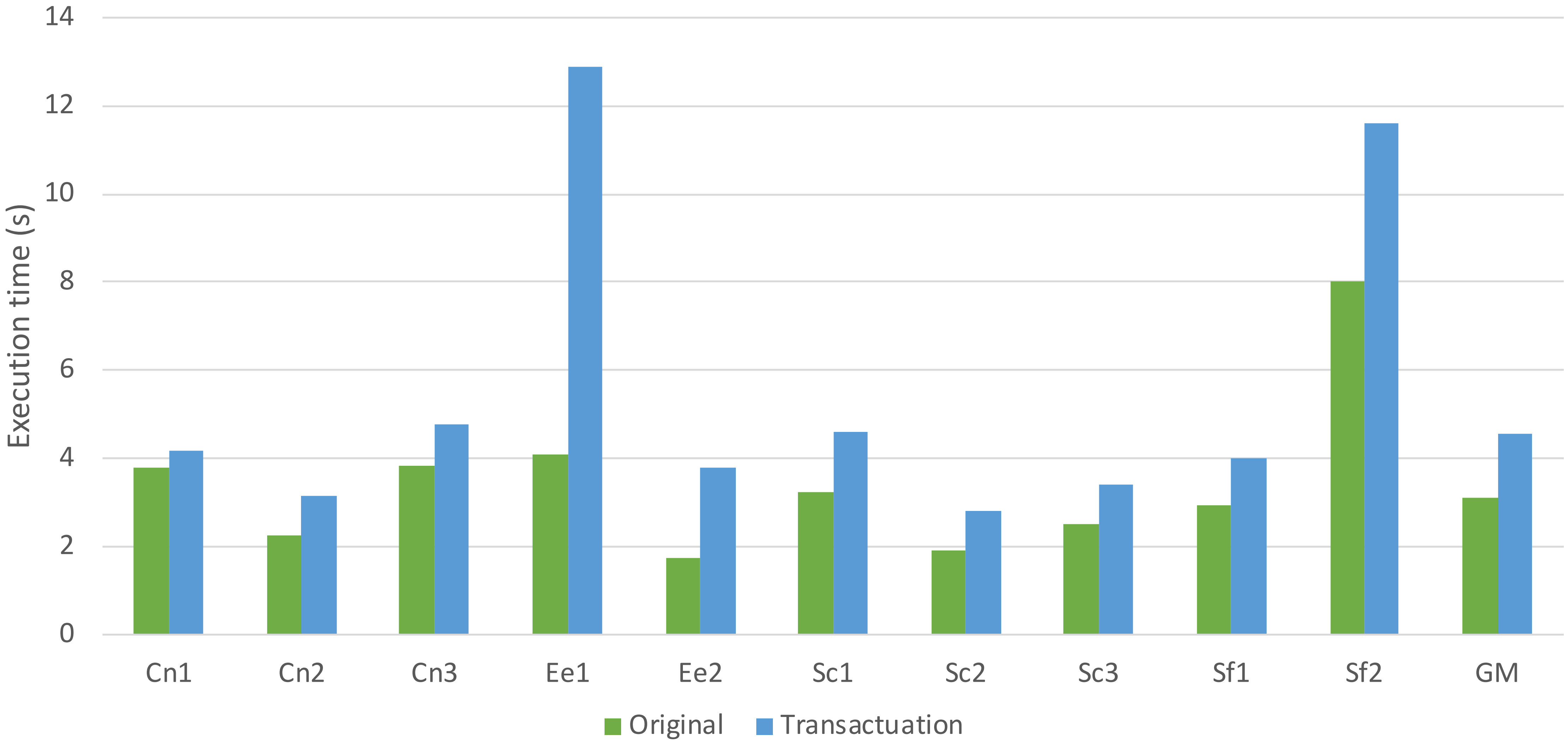
Application	Original App	Original App + Consistency	Transactuation
Rise and Shine (Cn1)	72	195	68
Whole House Fan (Cn2)	29	176	26
Thermostat Auto Off (Cn3)	70	198	68
Auto Humidity Vent (Ee1)	49	170	100
Lights Off With No Motion (Ee2)	56	161	67
Cameras On When Away (Sc1)	31	149	88
Nobody Home (Sc2)	65	175	62
Smart Security (Sc3)	144	323	144
Co2 Vent (Sf1)	29	152	26
Lock It When I Leave (Sf2)	51	180	54

Programmability

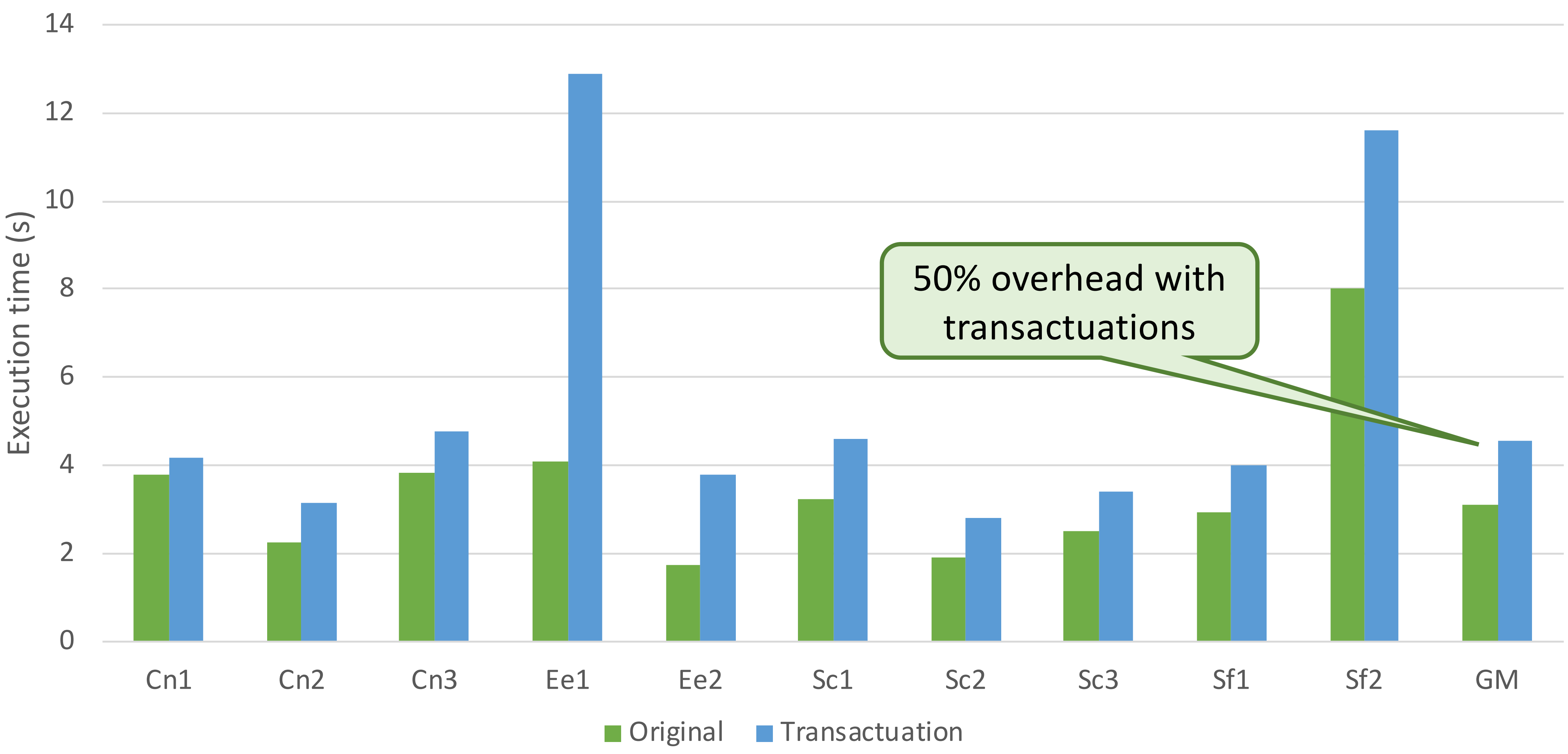
Lines of Codes

Application	Original App	Original App + Consistency	Transactuation
Rise and Shine (Cn1)	72	195	68
Whole House Fan (Cn2)	29	176	26
Thermostat Auto Off (Cn3)	70	198	68
Auto Humidity Vent (Ee1)	49	170	100
Lights Off With No Motion (Ee2)	56	161	67
Cameras On When Away (Sc1)	31	149	88
Nobody Home (Sc2)	65	175	62
Smart Security (Sc3)	144	323	144
Co2 Vent (Sf1)	29	152	26
Lock It When I Leave (Sf2)	51	180	54

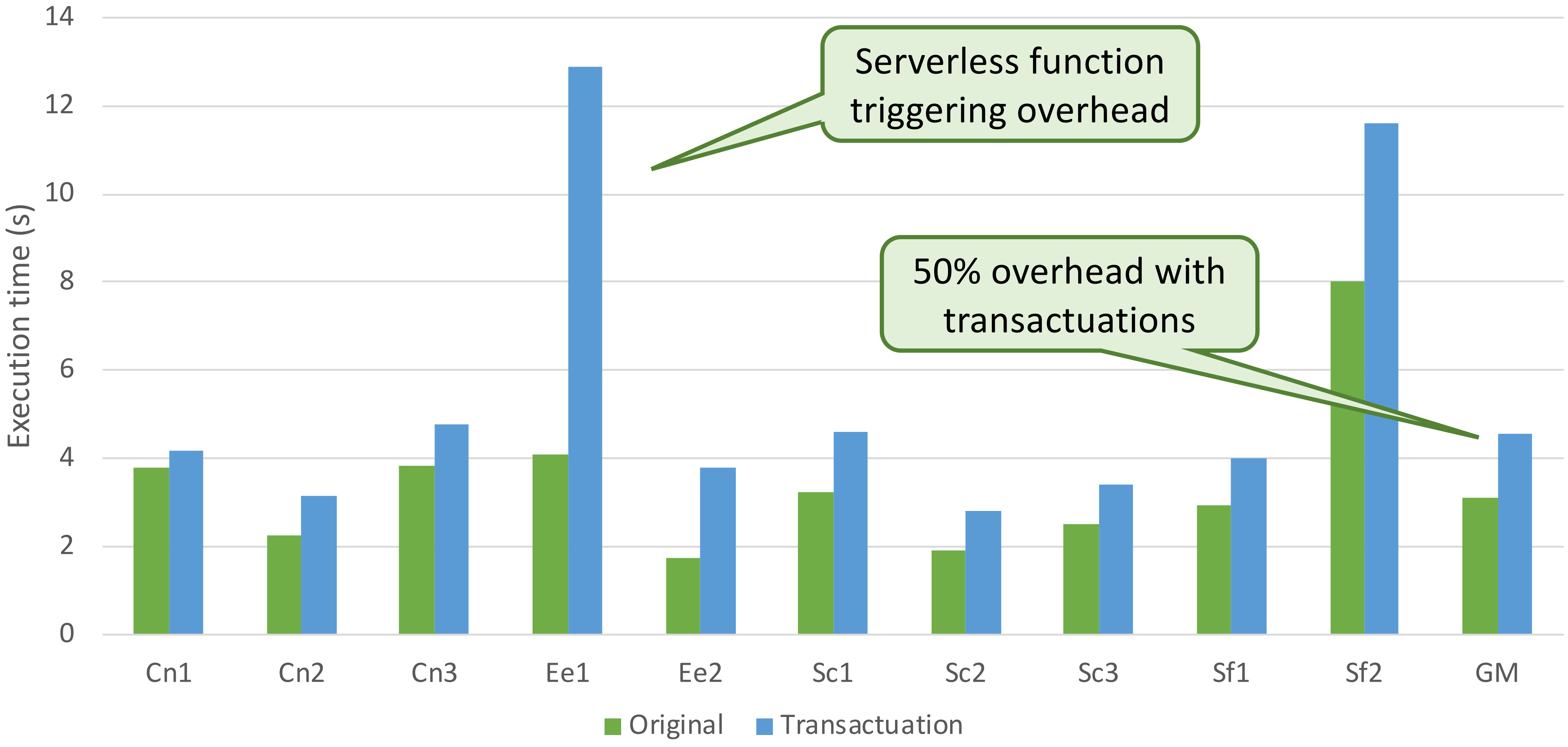
Runtime Overhead without Failures



Runtime Overhead without Failures



Runtime Overhead without Failures

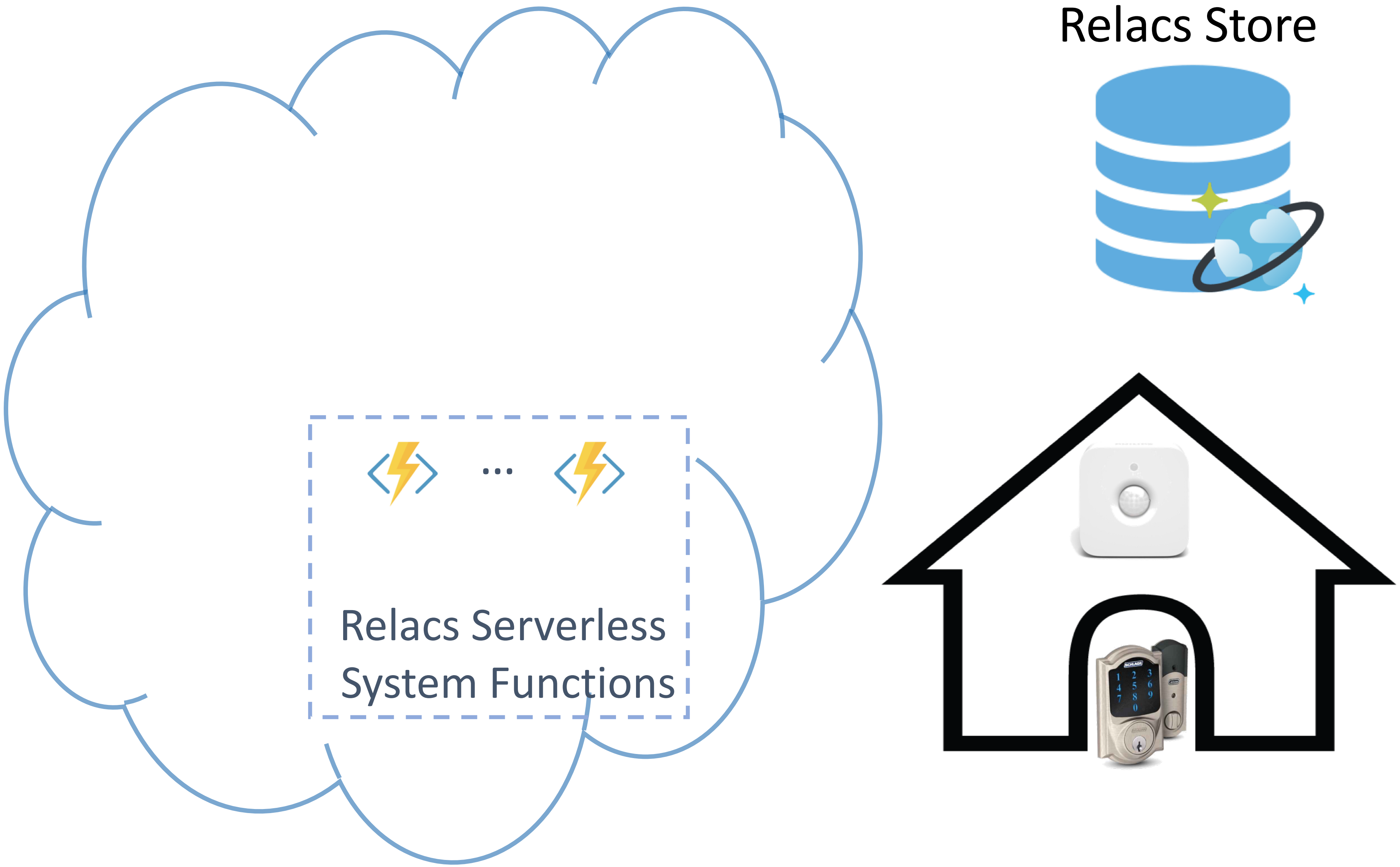


Conclusion

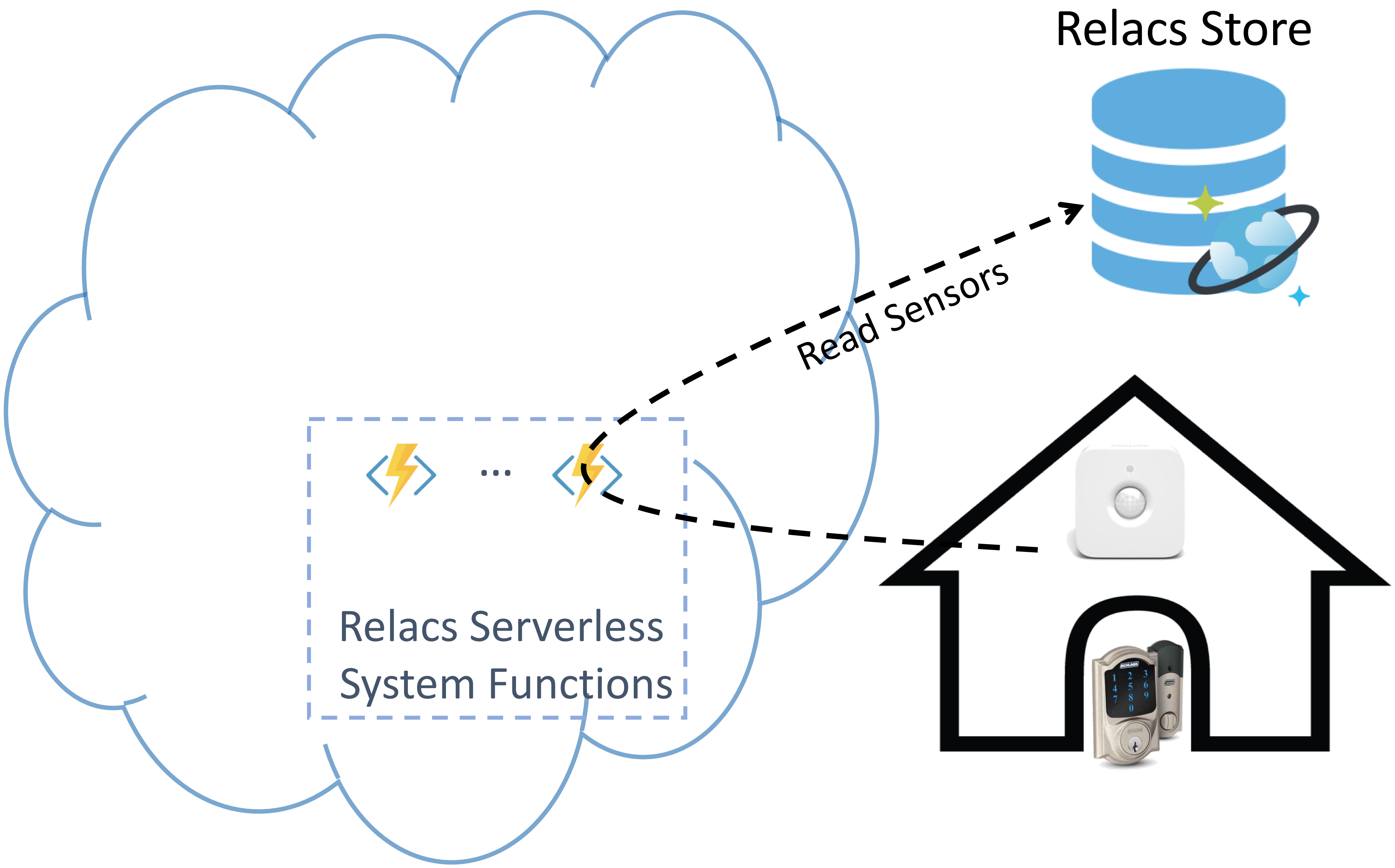
- Established a critical reliability issue due to inconsistencies
- Transactuation allows a developer to program in a failure-aware way
- Demonstrated transactuation’s programmability, performance, and effectiveness

Additional Slides

Relacs

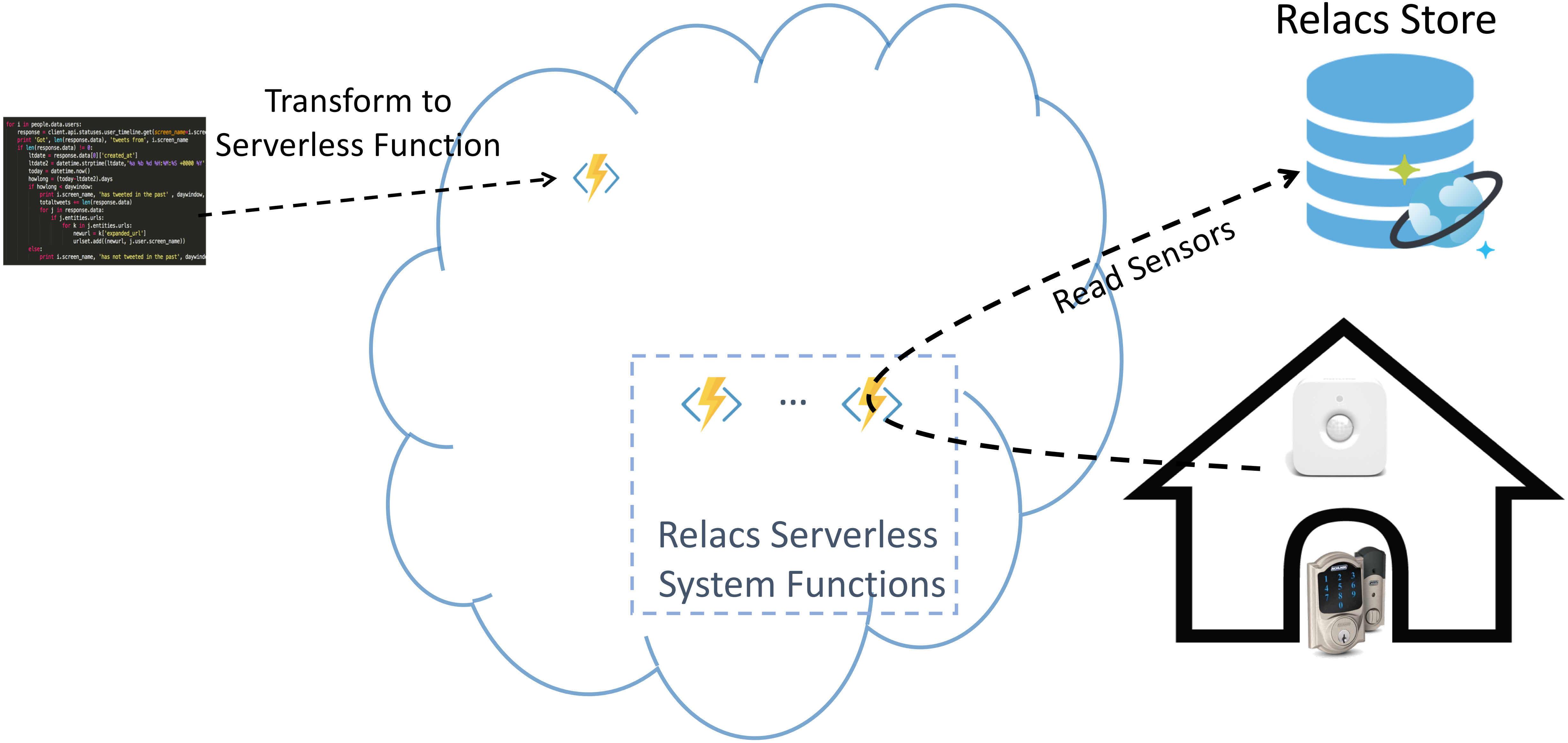


Relacs



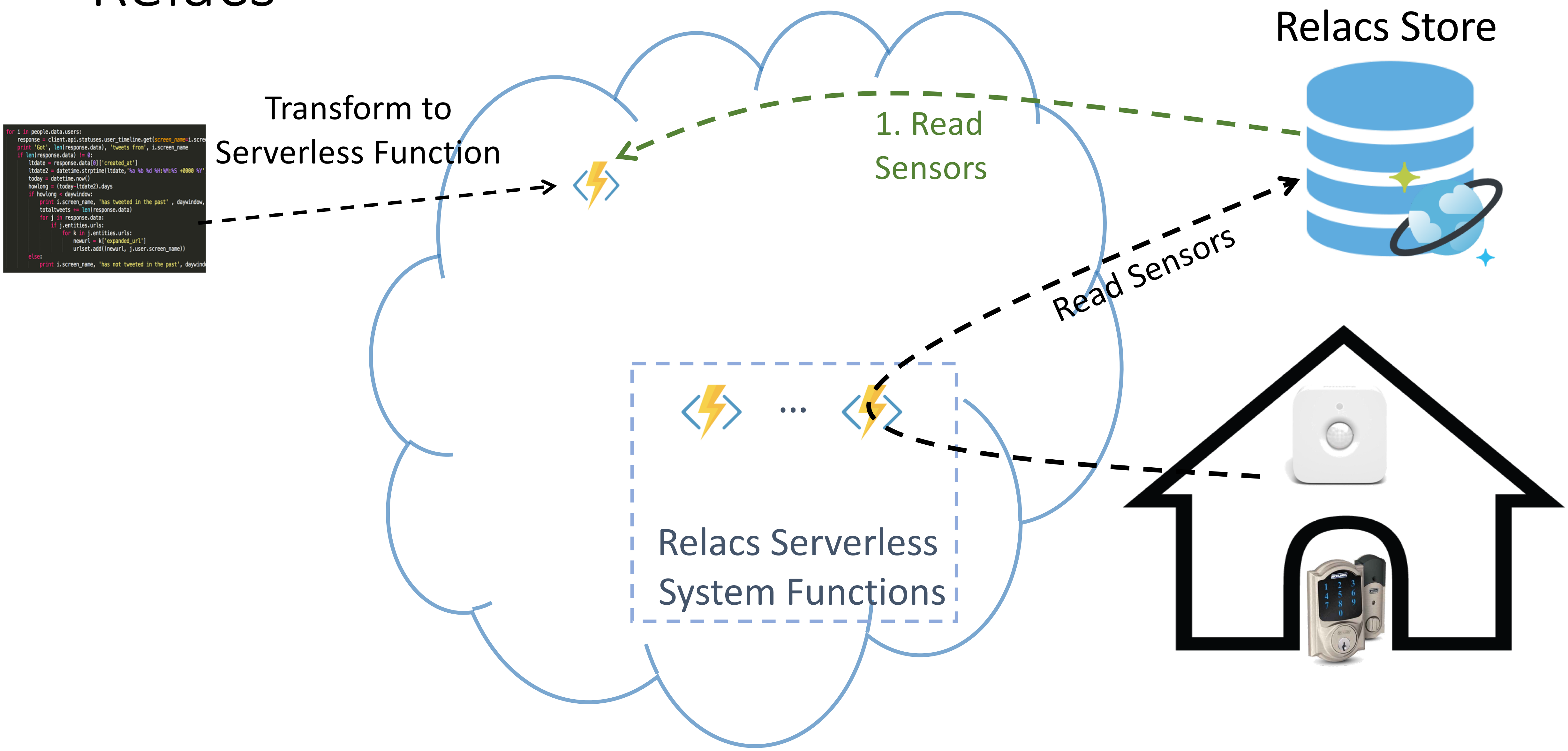
Relacs Store
Read Sensors
...
Relacs Serverless
System Functions

Relacs



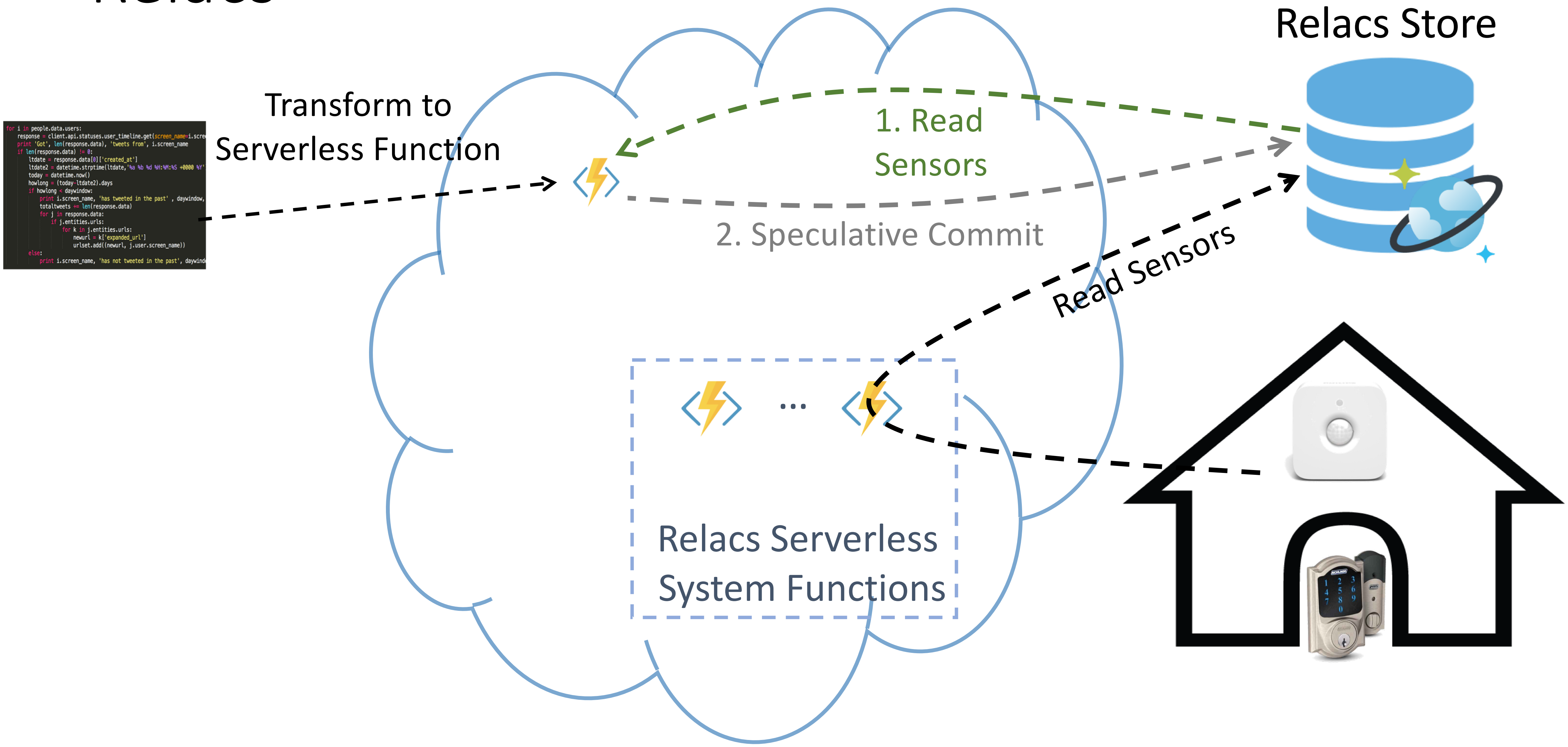
Relacs Store
Transform to
Serverless Function
Read Sensors
...
Relacs Serverless
System Functions

Relacs

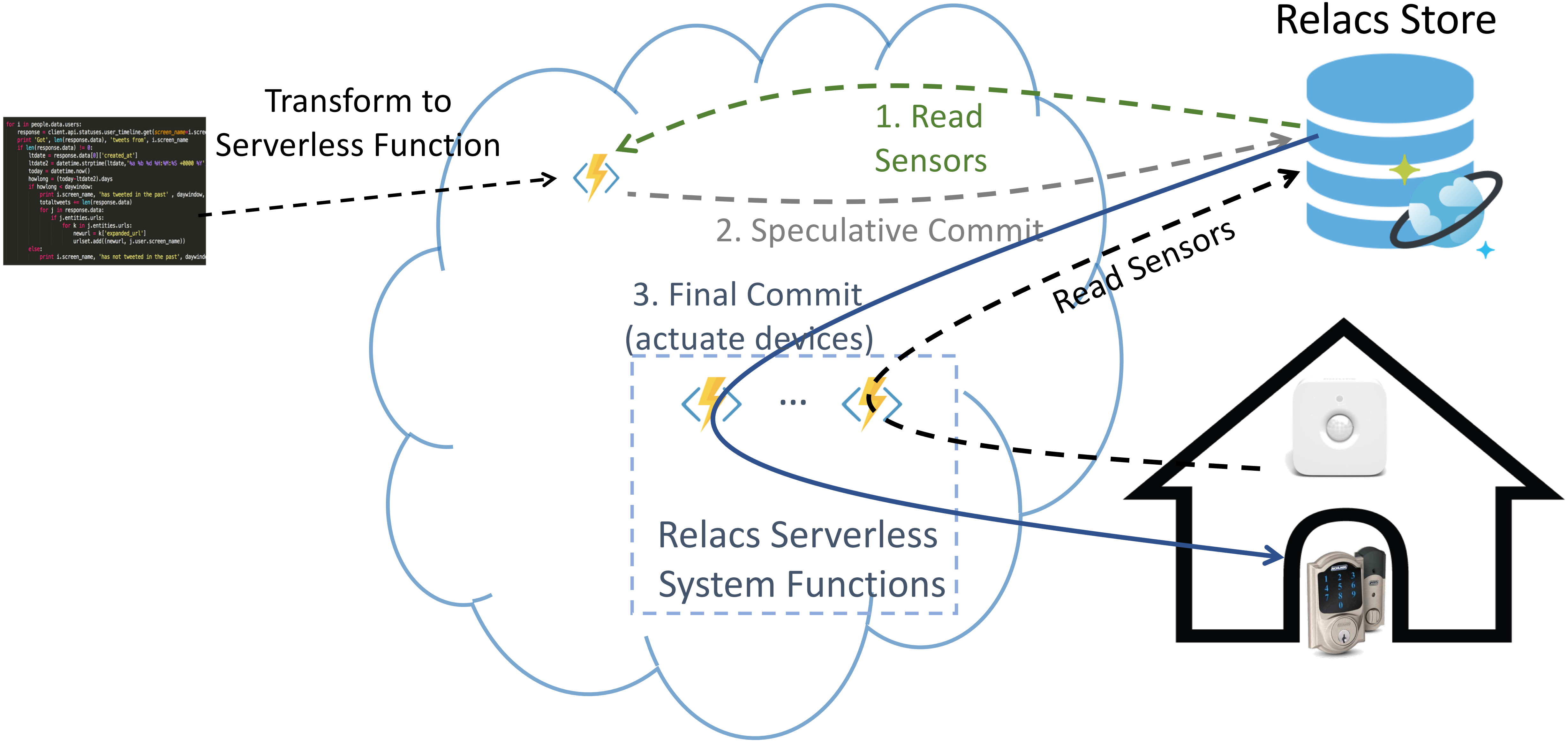


for i in people.data.users:
response = client.api.statuses.user_timeline.get(screen_name=i.scre
print 'Got', len(response.data), 'tweets from', i.screen_name
if len(response.data) != 0:
ltdate = response.data[0]['created_at']
ltdate2 = datetime.strptime(ltdate,'%a %b %d %H:%M:%S +0000 %Y'
today = datetime.now()
howlong = (today-ltdate2).days
if howlong < daywindow:
print i.screen_name, 'has tweeted in the past' , daywindow,
totaltweets += len(response.data)
for j in response.data:
if j.entities.urls:
for k in j.entities.urls:
newurl = k['expanded_url']
urlset.add((newurl, j.user.screen_name))
else:
print i.screen_name, 'has not tweeted in the past', daywind
Transform to
Serverless Function
1. Read
Sensors
Relacs Store
Read Sensors
...
Relacs Serverless
System Functions

Relacs


Relacs Store
Transform to Serverless Function
1. Read Sensors
2. Speculative Commit
Read Sensors
...
Relacs Serverless System Functions

Relacs


Relacs Store
Transform to
Serverless Function
1. Read
Sensors
2. Speculative Commit
Read Sensors
3. Final Commit
(actuate devices)
...
Relacs Serverless
System Functions