Home, SafeHome: Smart Home Reliability with Visibility and Atomicity

Shegufta B. Ahsan, Rui Yang, Shadi A. Noghabi, and Indranil Gupta

Department of Computer Science, University of Illinois at Urbana-Champaign
Microsoft Research

** slides taken from authors and modified*

SafeHome

- A first step towards Smart Home OS
  - Reasons about atomicity and isolation

- Home Automation System that can
  - Support *long running* routines
  - Properly *isolate* concurrent routines (providing *serial equivalence*)
  - Ensure routine execution *atomicity*

- Key challenge: Actions are visible to users

- Methodology:
  - Four *Visibility Models* (Spectrum for user choices)
  - *Lock-based* mechanism with *leasing* design

Motivation (why it’s important?)

- Diversity & scale of smart devices
- Need for safe and smart home management systems
- Concurrency causes incongruent end-state in real world

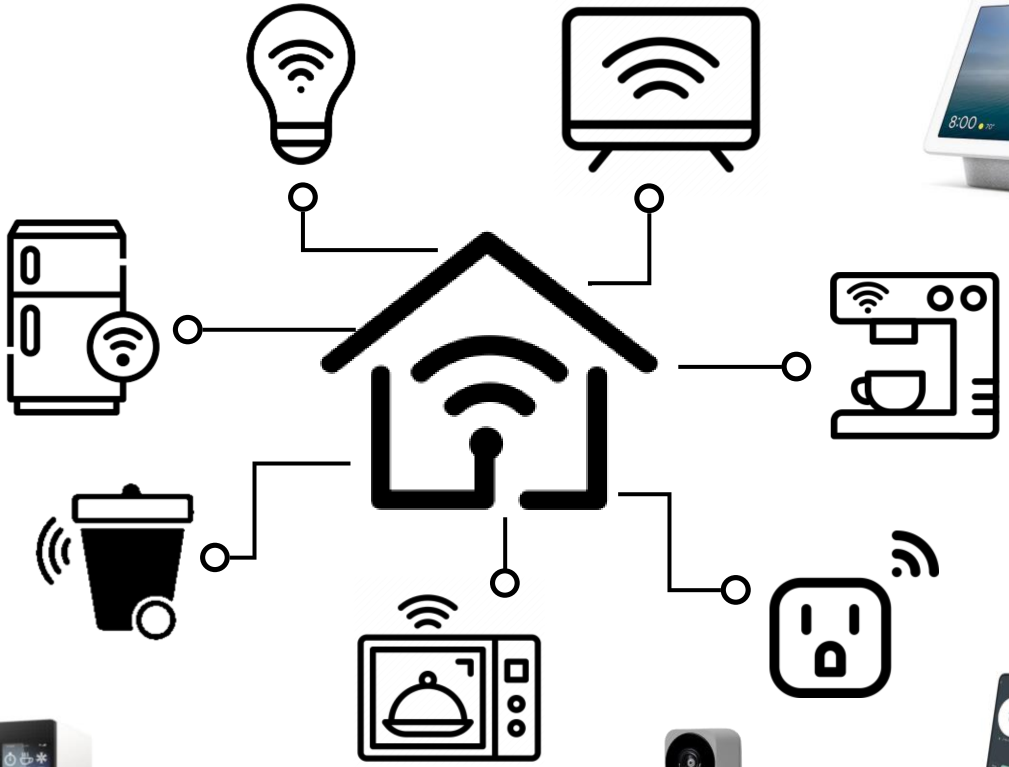
Diversity & scale of smart devices

Smart Device:

1) connected to other devices via wireless protocols
2) controlled by home automation systems

"Humans need to control their lives, not control devices."

-- Davidoff et al, UbiComp'06

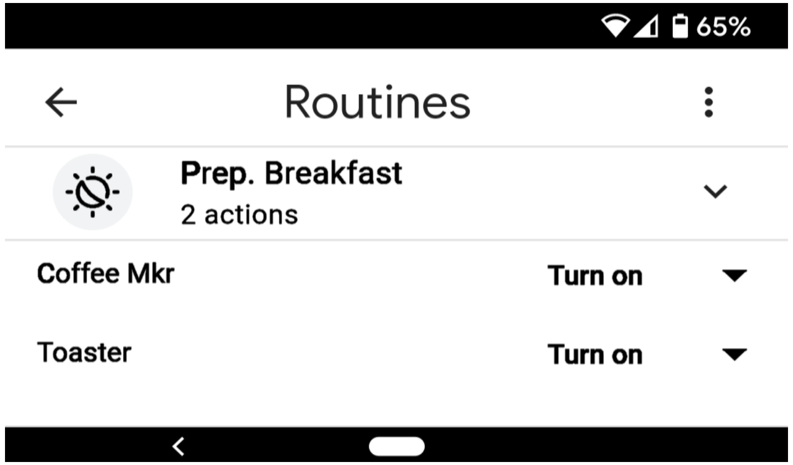


Need for safe management systems

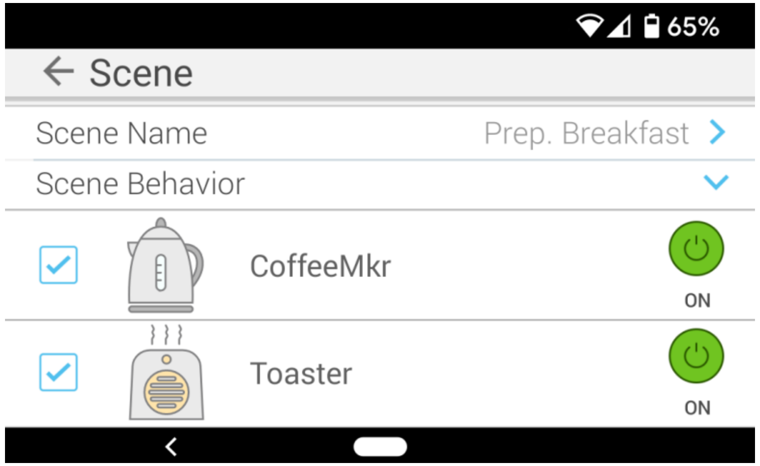
How people control smart home?

- by *Command*
  e.g. {Make an espresso}
- by *Routine*: *a sequence of commands*
  e.g. Prep. Breakfast = {Make an espresso; make a pancake}

Current systems execute Best-Effort!



Routine in Google Home

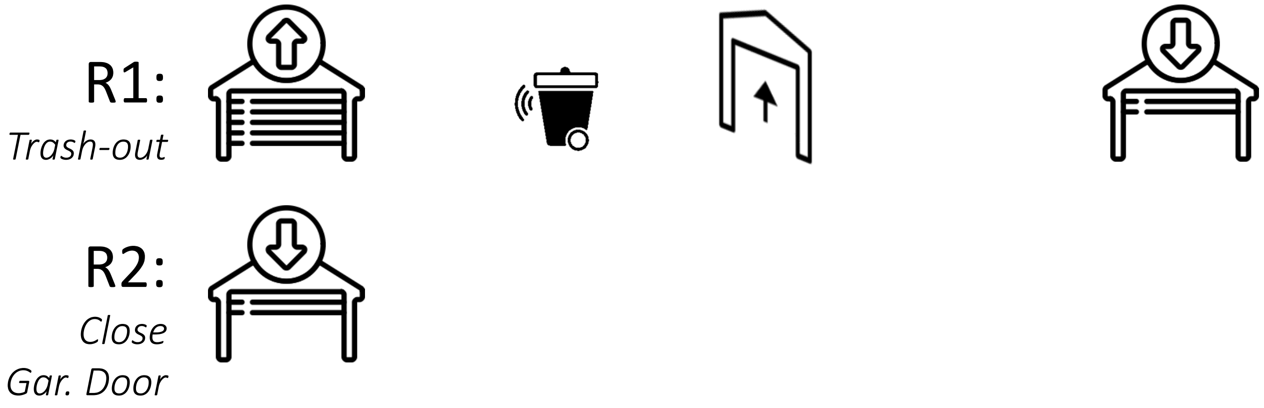


Routine in Kasa (TP-Link)

Concurrency causes incongruent end-state

- Execute everything in a routine – *Atomicity*
  - All commands in the routine need to finish successfully, or none do

- When conflicts happen, people hope routines to execute one after another – *Isolation / Serial Equivalence*



Poorly supported in current systems!

***Routines are common to be long running,** e.g. trash-out routine.*

SafeHome

- Home Automation System that can
  - Support *long running* routines
  - Properly *isolate* concurrent routines (providing *serial equivalence*)
  - Ensure routine execution *atomicity*

- Key challenge:
  - Actions are visible to users
  - Need to optimize for user-facing metrics
  - Device crashes/restarts and long-running routines are common

- Methodology:
  - Four *Visibility Models* (Spectrum for user choices)
  - *Lock-based* mechanism with *leasing* design

How it builds upon previous works?

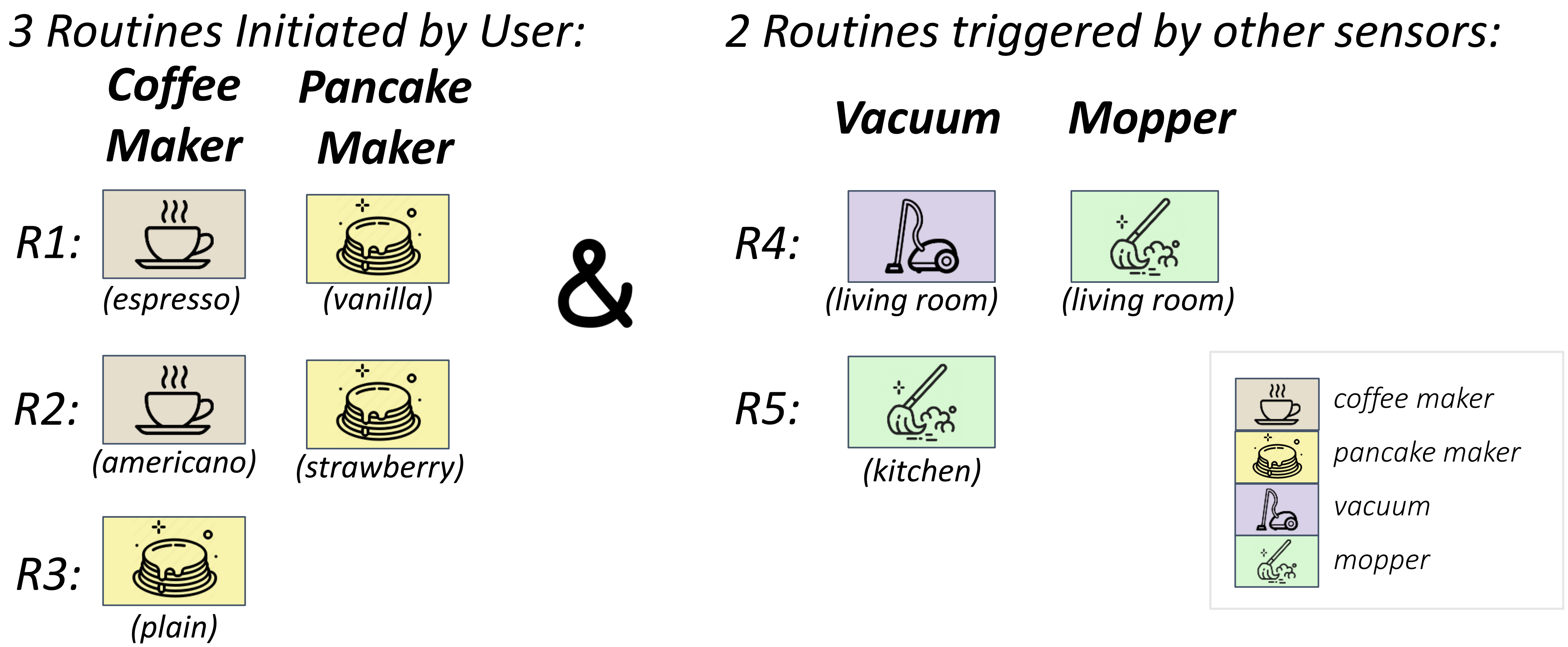
- Visibility models are counterpart to weak consistency models explored previously
- Some works use priority-based techniques to address concurrency
- Transactuations and APEX papers discuss atomicity and isolation for routine dependencies
- Many parallels b/w SafeHome and ACID properties but:
    - Optimize latency vs. throughput
    - Device failures (data is replicated but devices are not)
    - Long-running routines (starvation)

Visibility Models

Four Visibility Models:

- *Weak, Eventual, Partitioned Strict, Global Strict*

Example Scenarios: 5 routines are initiated *simultaneously* on 4 devices

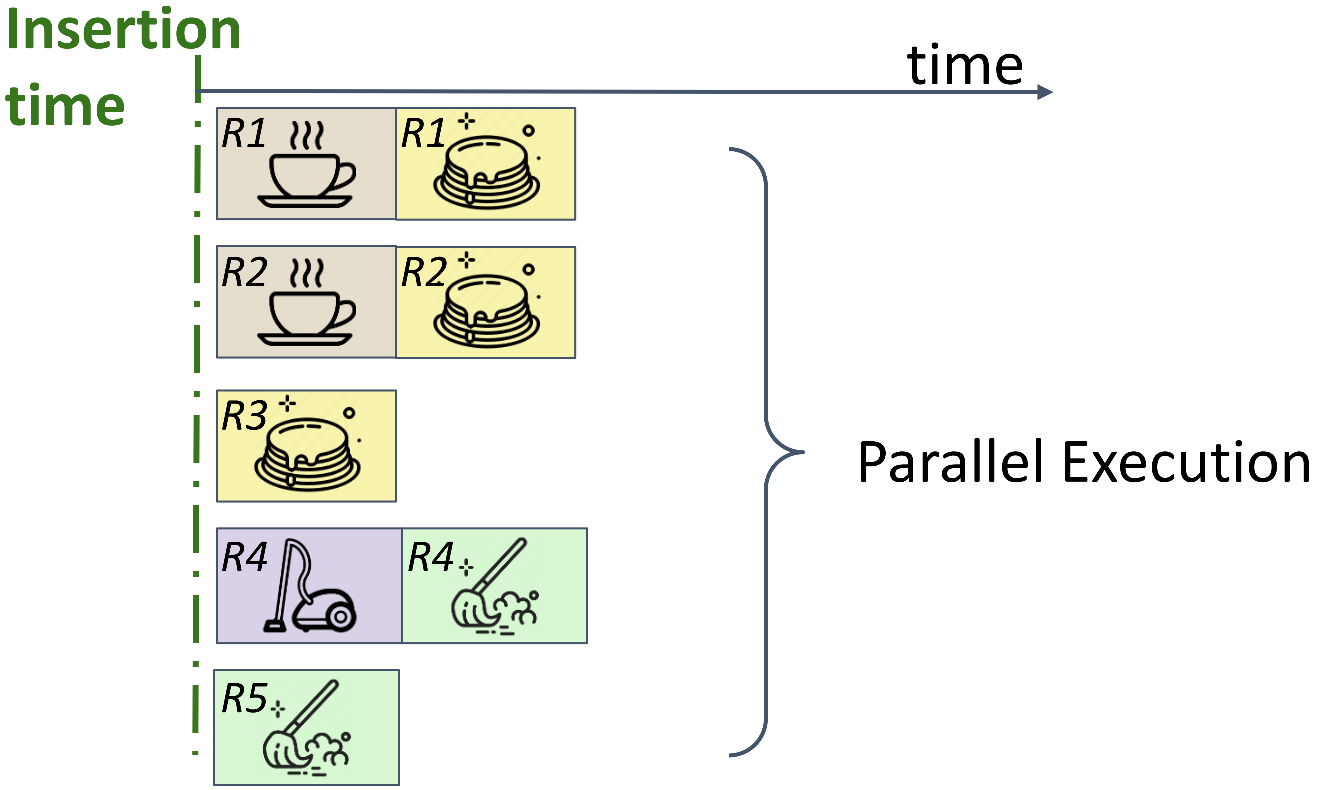


Weak Visibility (WV) Model -- Status Quo

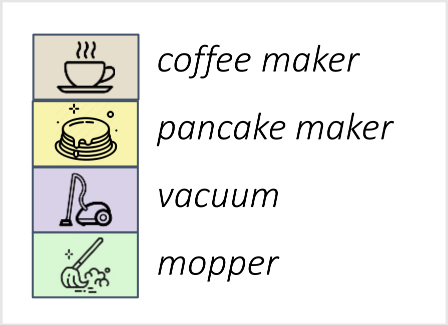
Strategy:

- Execute routine immediately when triggered

Finish in 2 time units



Two commands send simultaneously to one device may cause errors.

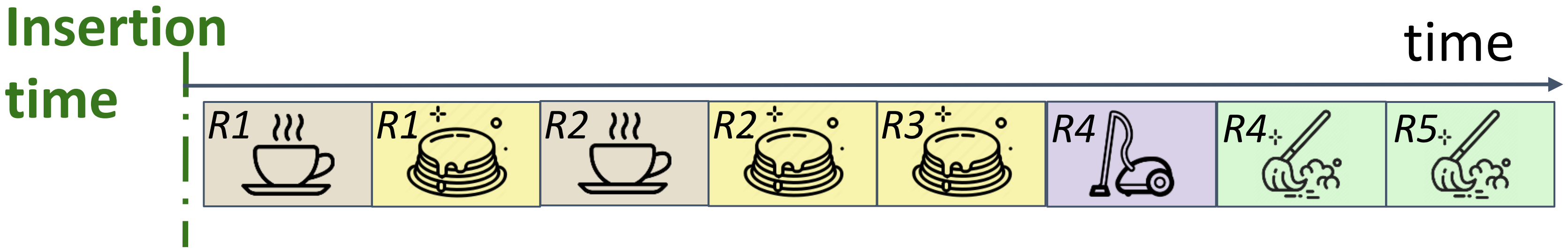


Global Strict Visibility (GSV) Model

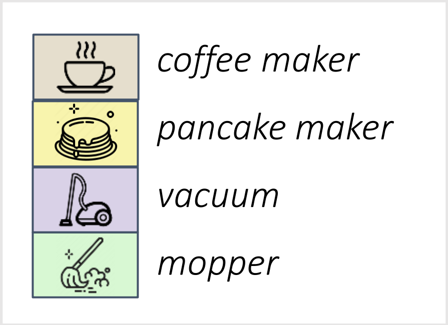
Strategy:

- Execute at most one routine at a time

Finish in 8 time units



- Strongest Visibility Model
- Example Usage: resource constrained environment:
  - e.g. 1000-watt max supply < coffee maker 600W + pancake maker: 600W

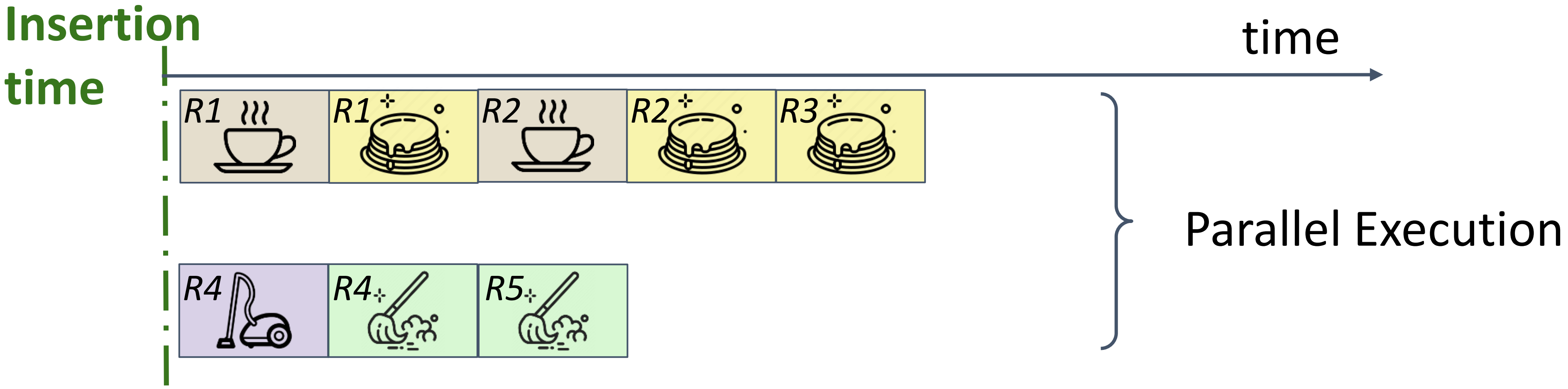


Partitioned Strict Visibility (PSV) Model

Finish in 5 time units

Strategy:

- Routines touching disjoint devices do not block each other



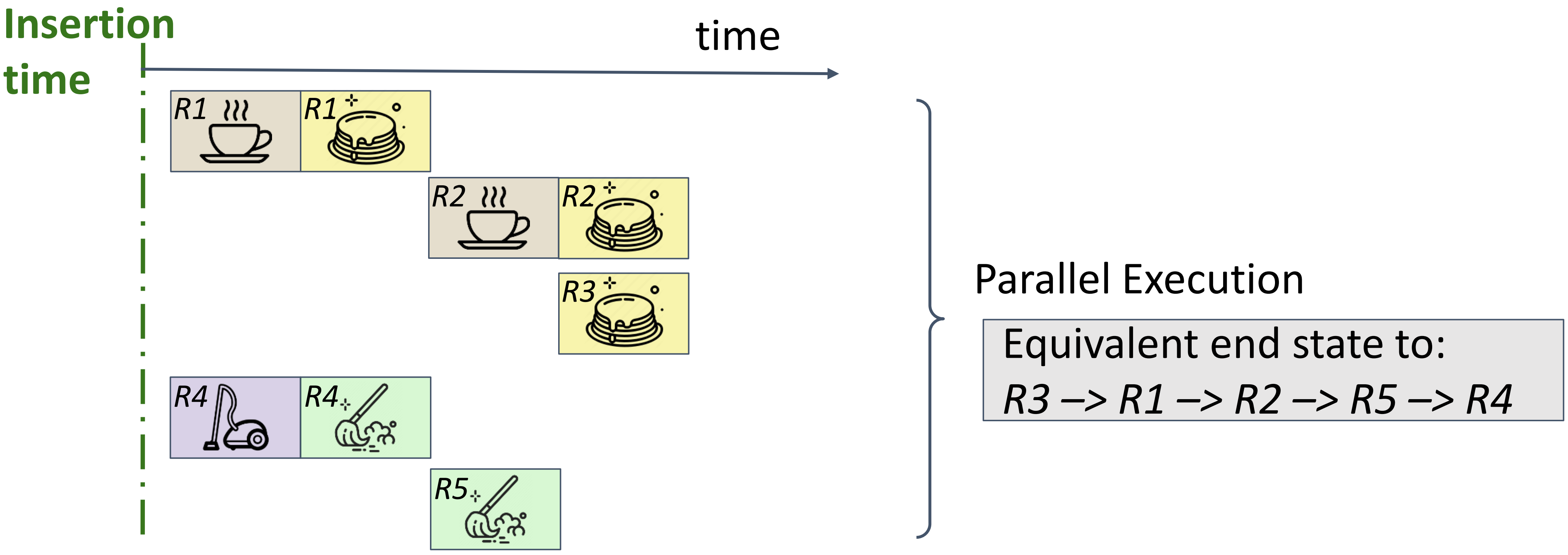
- Useful when routines need to execute without interference through duration.
- Might still takes long with long running routines.

Eventual Visibility (EV) Model

Finish in 3 time units

Strategy:

- Routines can concurrently execute *without violating some serial order*.



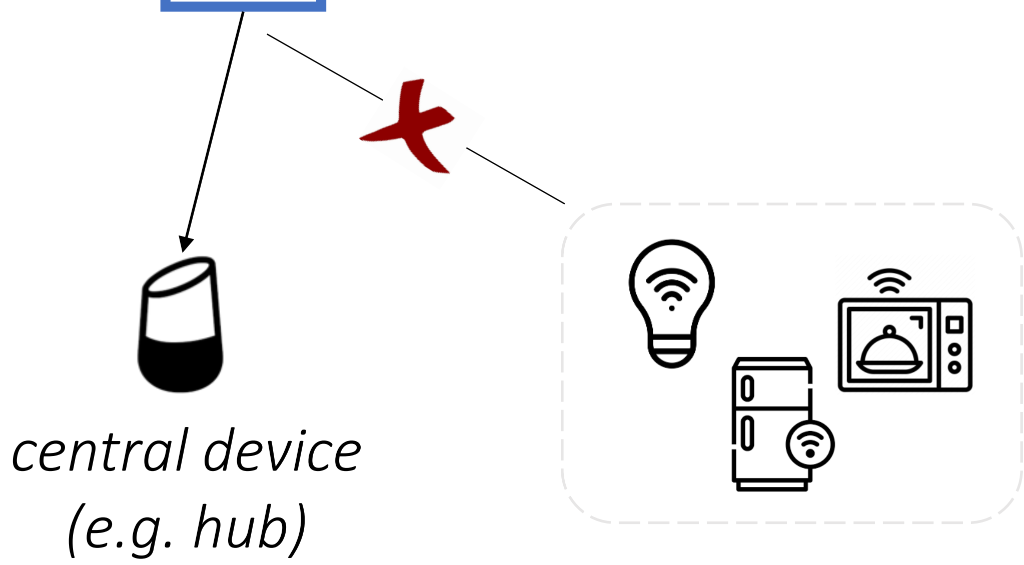
Eventual Visibility (EV) Model

Strategy:

- Routines can concurrently execute *without violating some serial order*.
- Each routine holds the *locks* for devices it touches (but can lease the lock).

Post-lease

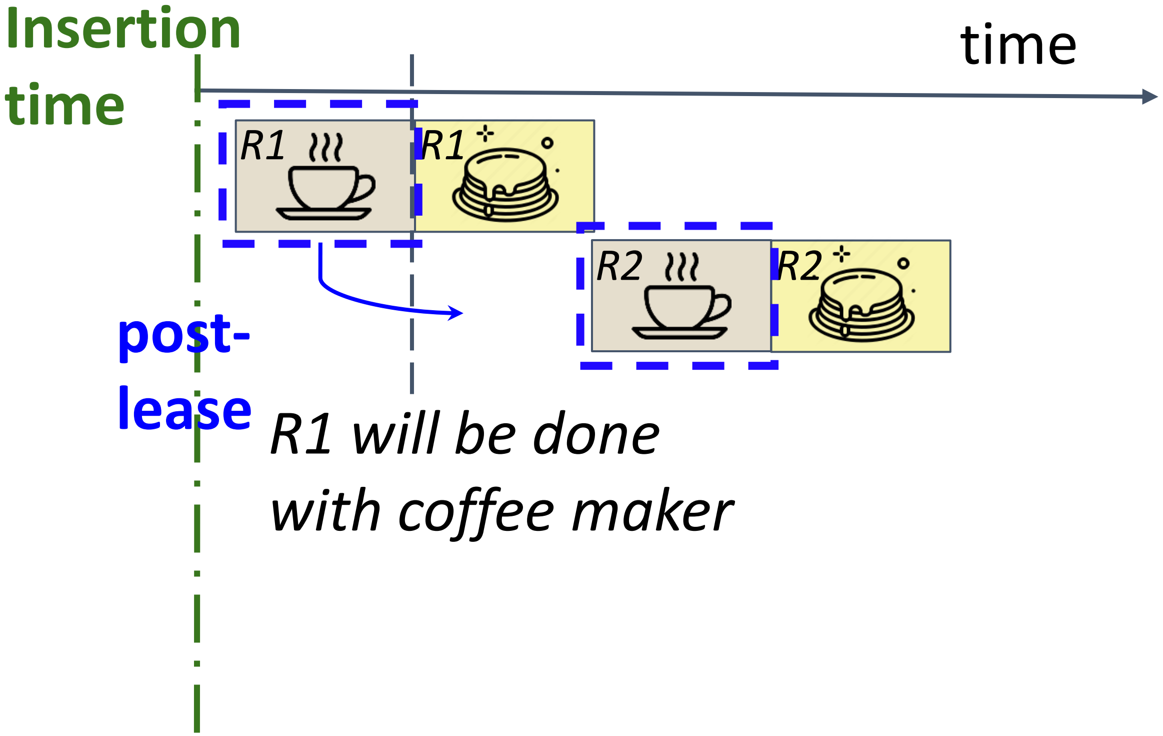
Pre-lease



Eventual Visibility (EV) - Post-Lease

Post-lease:

- If a routine is done with a device *D*, it can post-lease *D*'s lock to another routine.

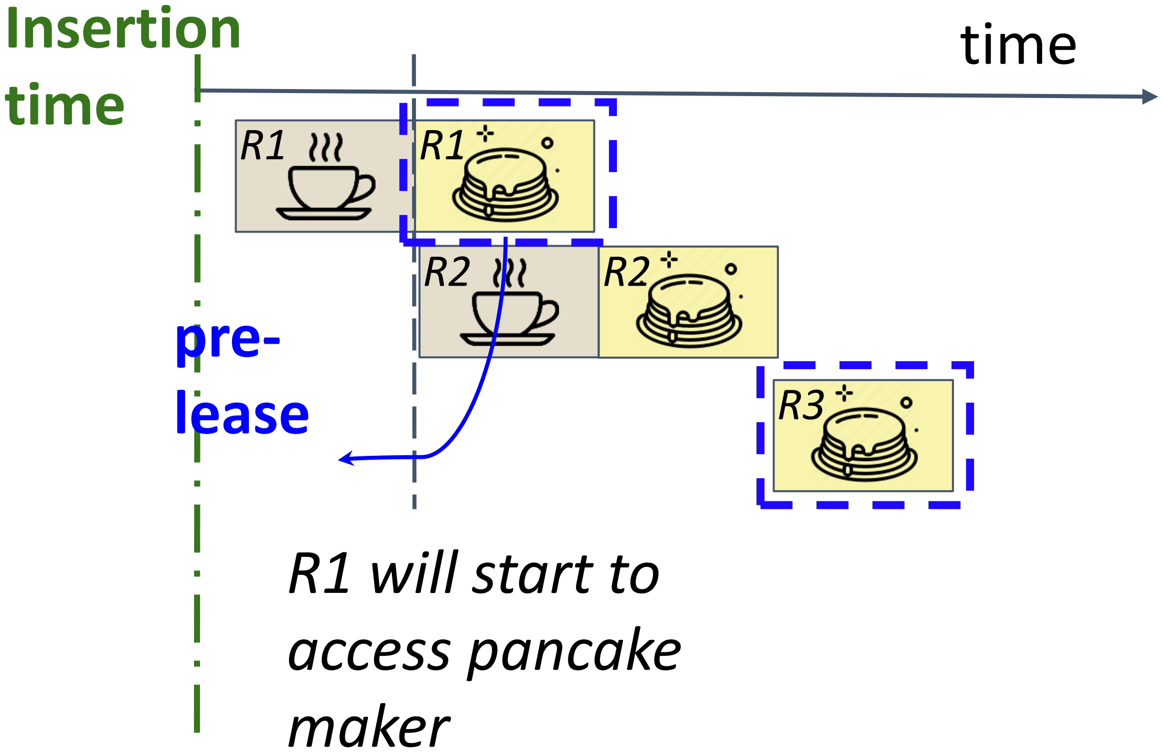


Serial order:
lessor –> lessee
(*R1* –> *R2*)

Eventual Visibility (EV) - Pre-Lease

Pre-lease:

- If a routine has acquired the lock but not accessed a device *D*, it can pre-lease *D*'s lock to another routine.

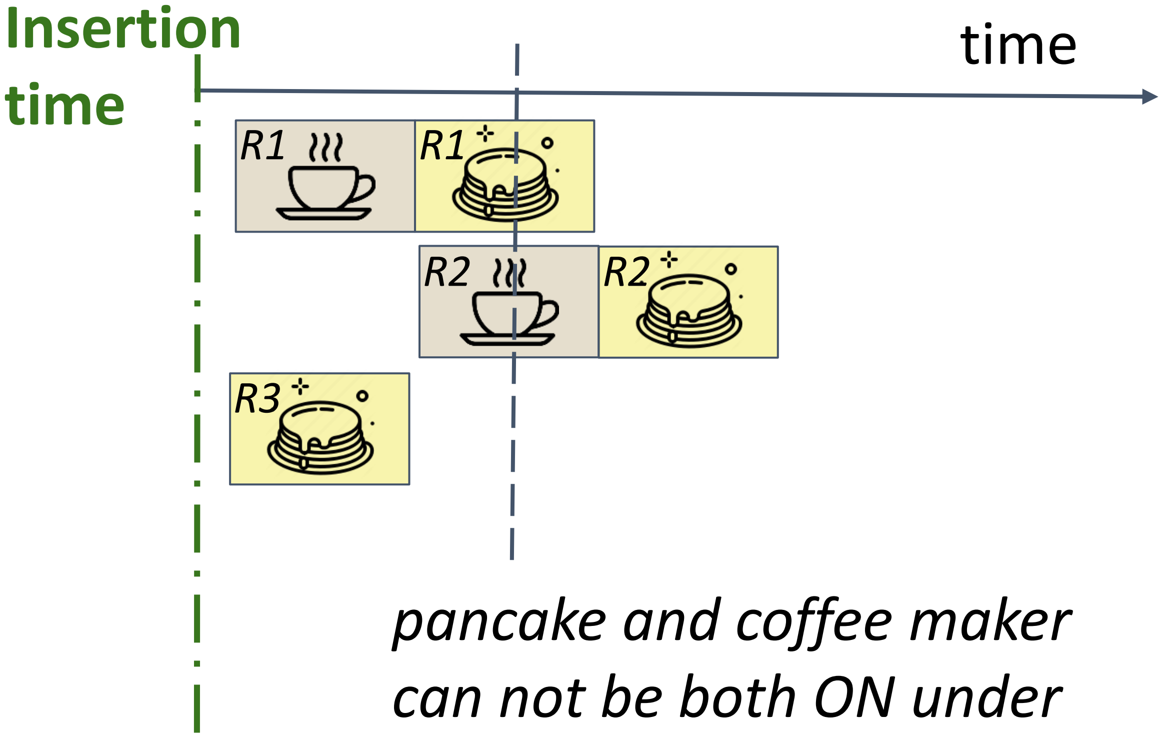


Serial order:
lessee –> lessor
(*R3* –> *R1*)

Eventual Visibility (EV)

Finish in 3 time units

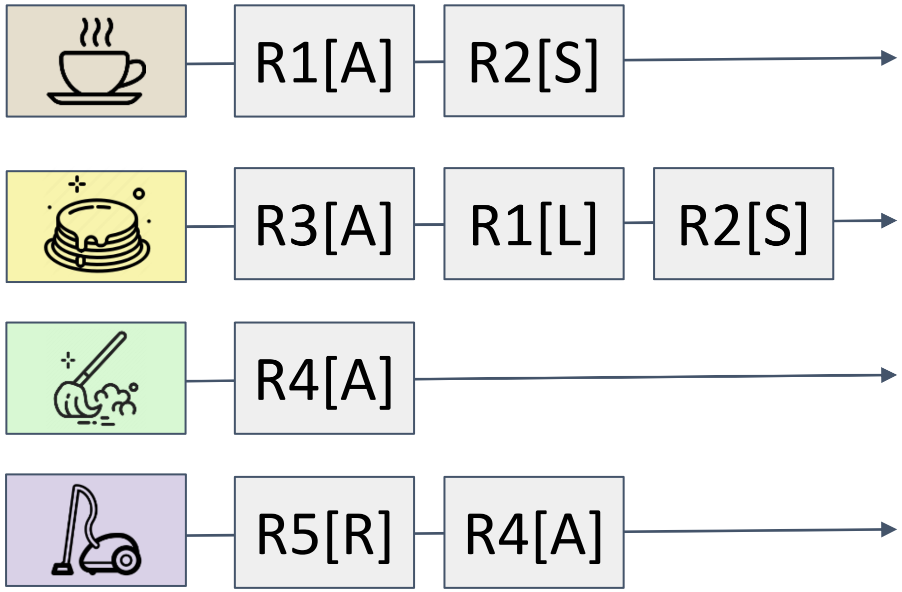
EV finishes routine
- with *short wait* and provides *serial equivalence*
- with higher *temporary incongruence: intermediate state is not serially equivalent*



pancake and coffee maker can not be both ON under any serial order

Eventual Visibility (EV) - Lineage Table

Lineage Table: SafeHome's plan of which routine will access which device.



[A]: *Get lock Access*
[S]: *Routine Scheduled*
[L]: *Lock Leased out*
[R]: *Lock Released*

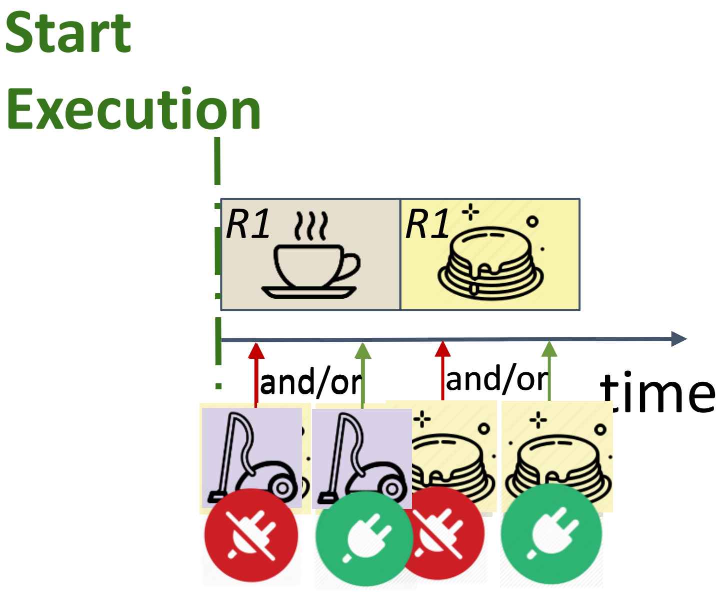
Scheduling plan placement:

- Placed when routine is triggered
- Use *backtracking* for valid placement
- Explore two other policies (FCFS, JiT)

Failure Serialization and Rollback

Device might *fail*:

- *Rollback*? Try to *serialize* the failure/restart event!
- If the failed device is not touched by the routine:
  - *Arbitrary* Serial Equivalence order
- If device fails/restarts after the last touch:
  - *Routine –> Fail/Restart* Serial Equivalence order
- If device fails/restarts before the first touch:
  - *Fail/Restart –> Routine* Serial Equivalence order
- If device fails/restarts during the touch:
  - *Rollback* routine



Failure –> Restart –> R1

SafeHome Implementation

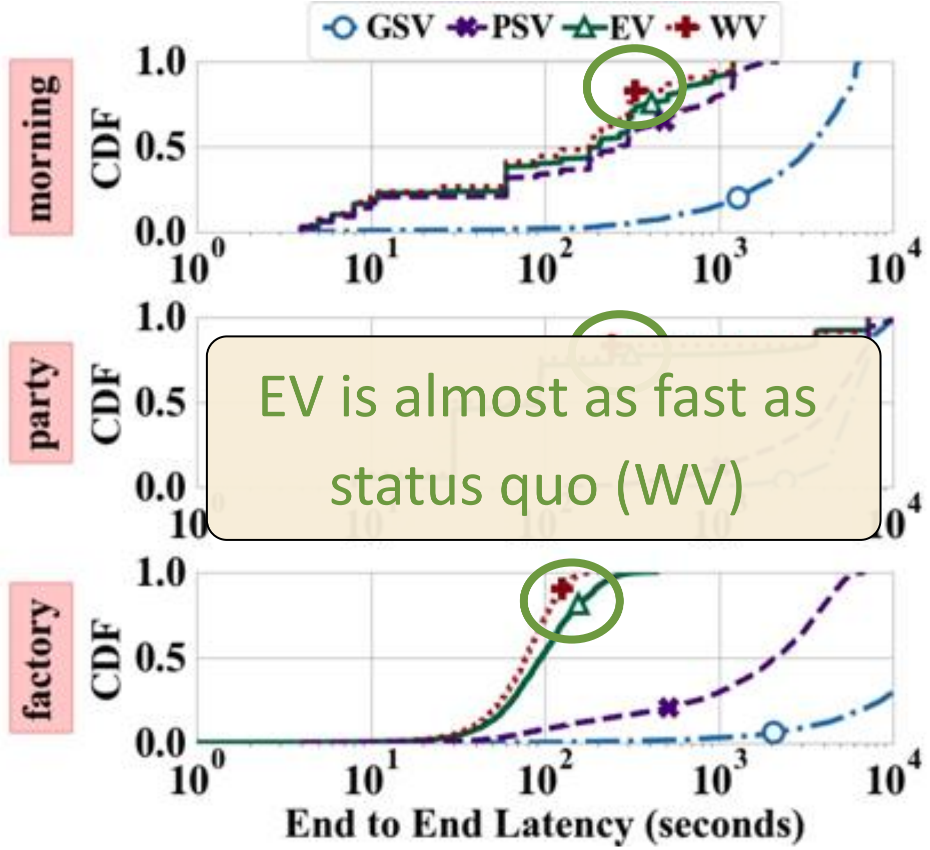
Implementation

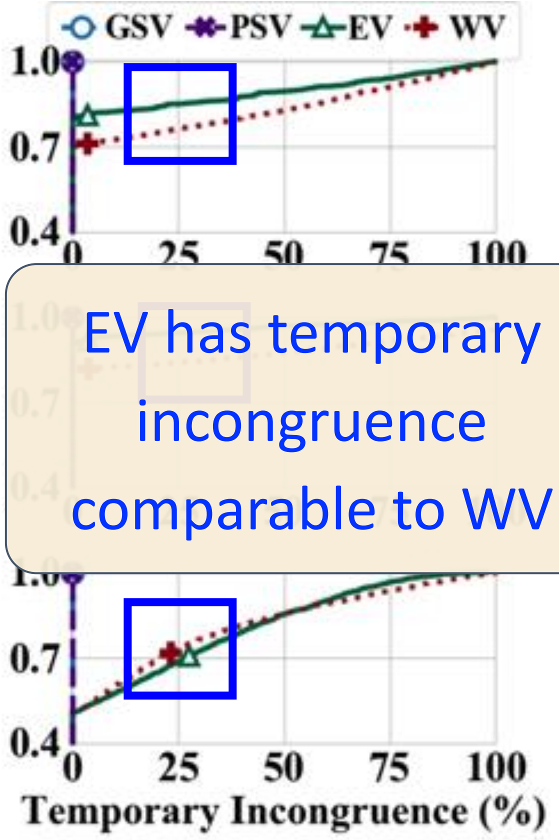
- ~2k line of *Java* code
- Support *long running routine* expression (JSON)
- Popular Smart Device *integration* (TP-link, Wemo)

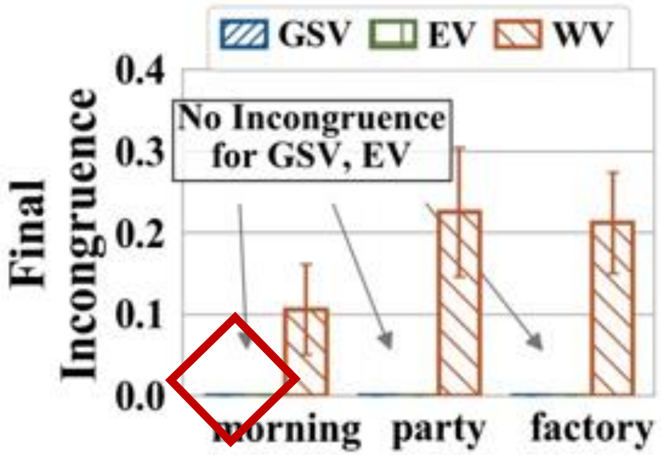
Experiment Setup

- Deployment & Simulation
- Real-world Benchmark
  - Derived from *IoTBench Test Suite*
  - *Morning, Party, Factory* Scenario
- Workload-Driven
  - Average of *500k* runs

Real-World Benchmark







EV is almost as fast as status quo (WV)

EV has temporary incongruence comparable to WV

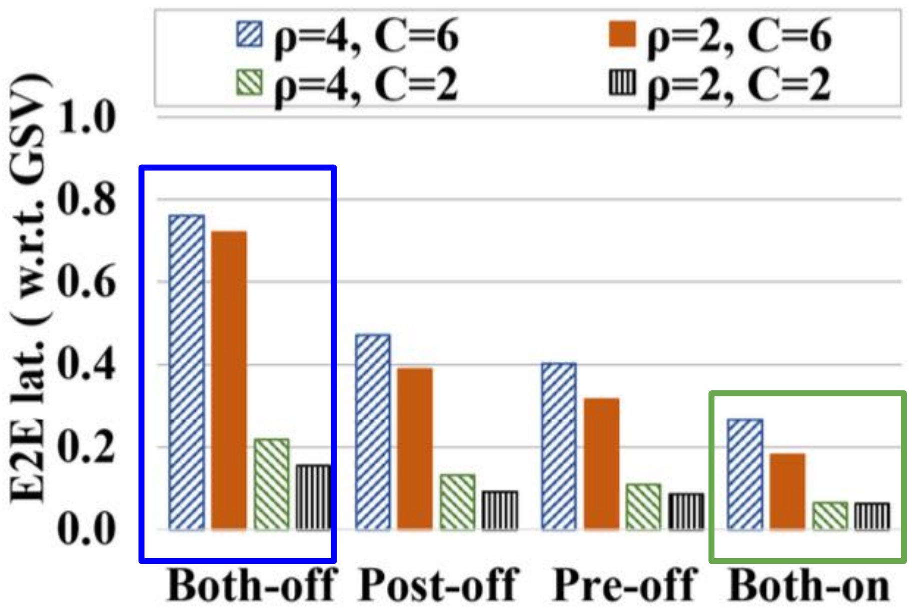
EV is serially equivalent, but WV not

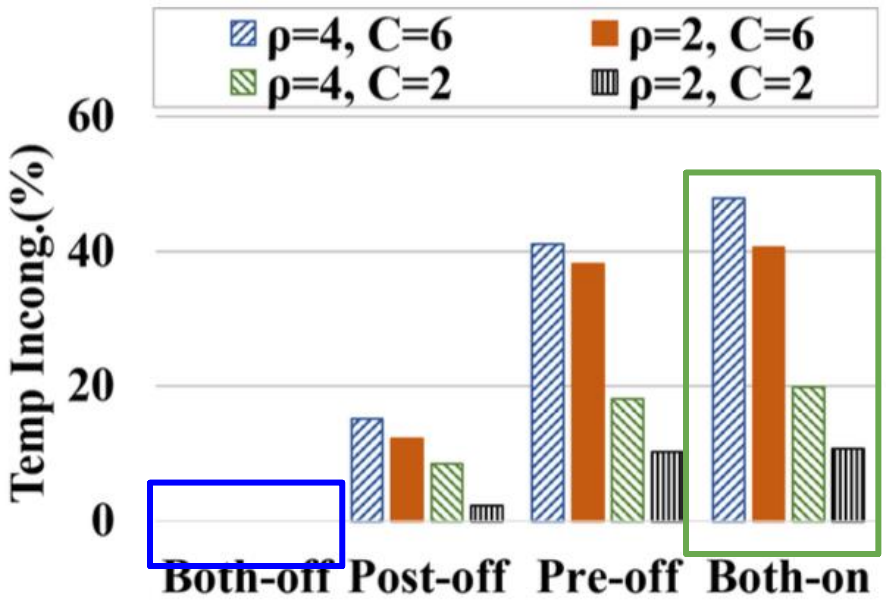
Temporary Incongruence: *the ratio of time when* ***intermediate*** *state is not serially equivalent.*
Final Incongruence: *the ratio of runs that* ***end up*** *in an incongruent state.*

Workload Evaluation -- Pre/Post-Lease

High Latency, Zero Temporary Incongruence

Low Latency, High Temporary Incongruence





Pre/Post leases reduce the E2E latency (user-facing metrics) with the cost of Temporary Incongruence

Takeaways

- Safehome is a first step to provide *reliability* from routine level exection
- SafeHome provides four *Visibility Models* (WV, EV, PSV, and GSV)
- *Eventual Visibility* (EV) model provides the best of both worlds, with:
  - Good user-facing *responsiveness* (0 - 23.1%)
  - Strongest *end state congruence* equivalent guarantee (as GSV)
- Lock-leasing *improves latency* by 1.5X - 4X

Trade-off b/w incongruence vs. latency while guaranteeing serial-equivalence

Discussion & Questions

- Think of a simpler scheme than early lock acquisition and lease?
- What happens when SafeHome fails?
- Paper discuss fail-stop failures
  - Can we reason about byzantine failures? Why or why not?
- The paper discussed reliability but what about availability?
  - Wait for next paper → Rivulet