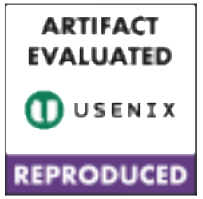



# If not RPCs for smart spaces, then what...?

Anna Karanika, Kai-Siang Wang, Han-Ting Liang,
Shalni Sundram, Indranil Gupta

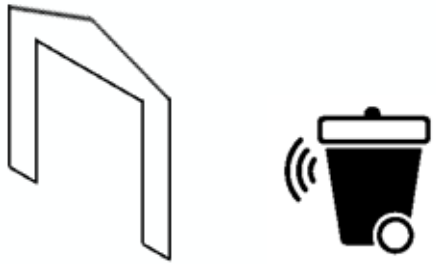


DISTRIBUTED PROTOCOLS RESEARCH GROUP: HTTPS://DPRG.CS.UIUC.EDU/

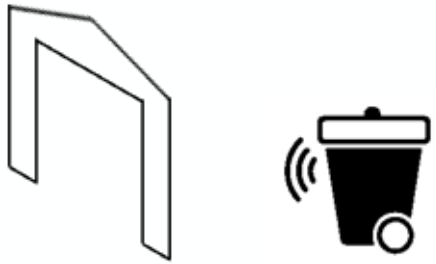
# A Routine Users Want: Is It Safe?

# A Routine Users Want: Is It Safe?

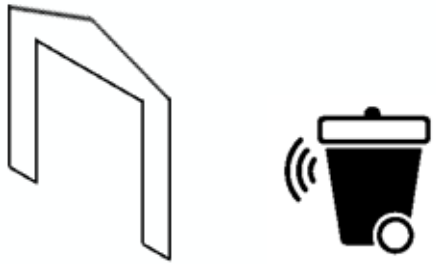
# A Routine Users Want: Is It Safe?



# A Routine Users Want: Is It Safe?

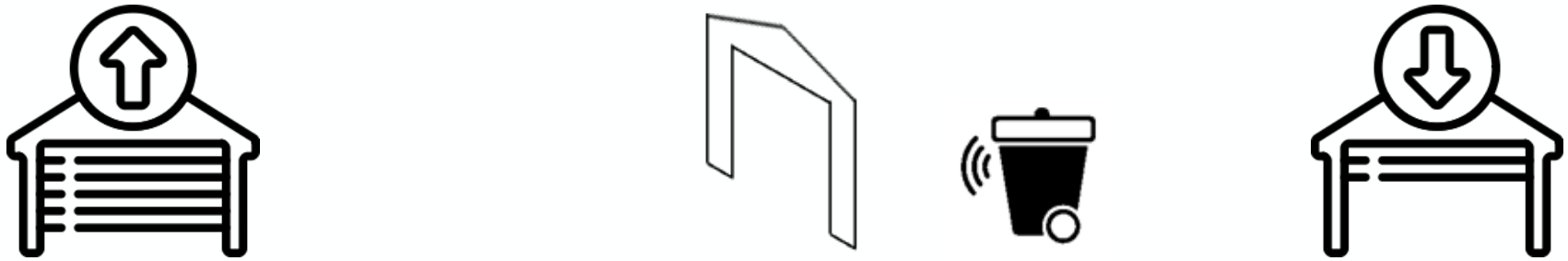


# A Routine Users Want: Is It Safe?



⚠ What if the trashcan gets stuck?

# A Routine Users Want: Is It Safe?

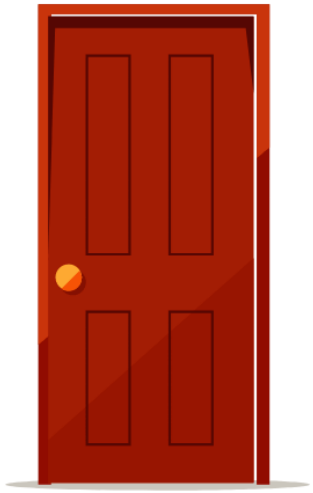


⚠ What if the trashcan gets stuck?

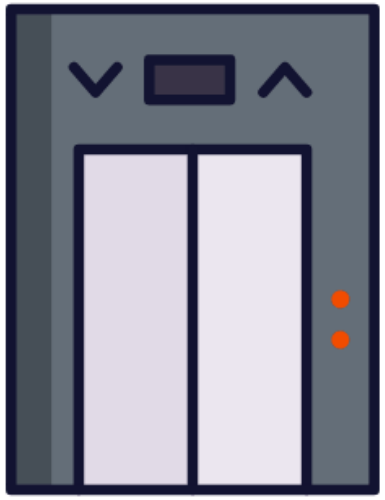
# IoT actions: long-running & physical



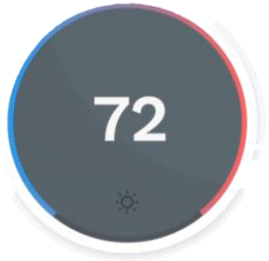
# IoT actions: long-running & physical



seconds

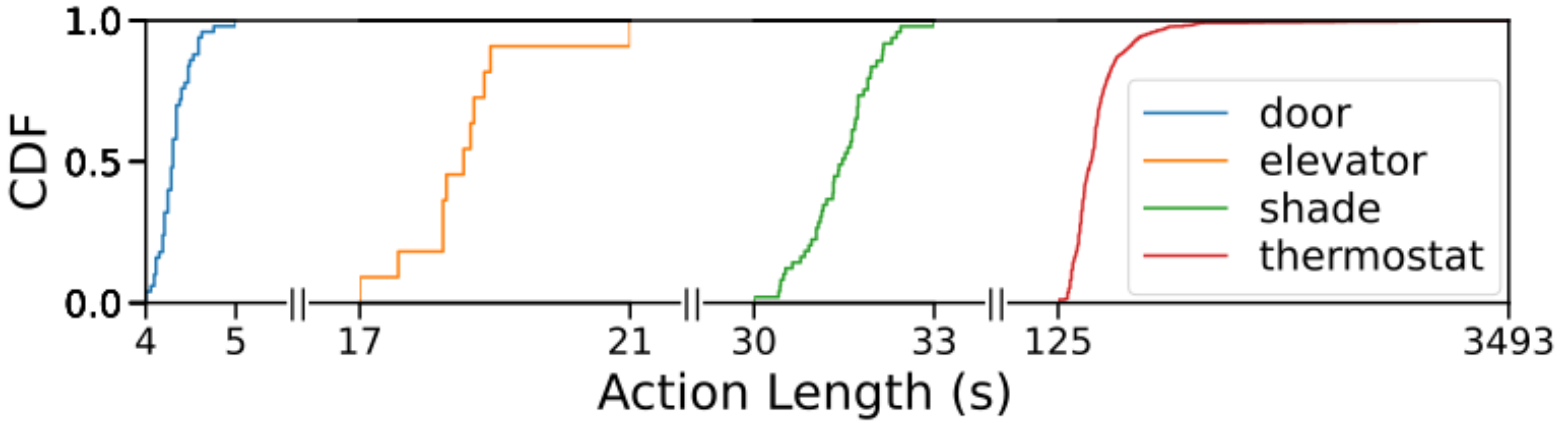


10s of seconds



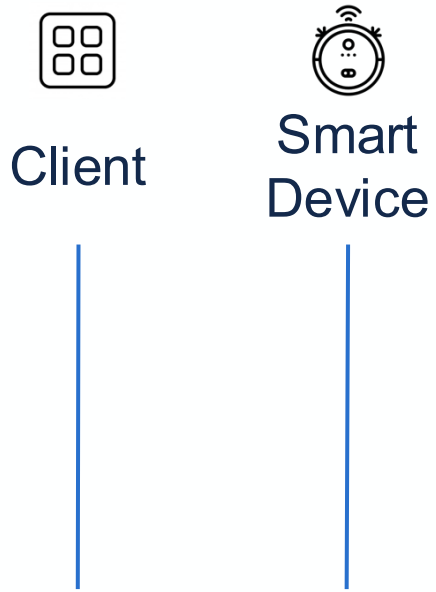

minutes to hours

Based on our office building's measurements and online traces

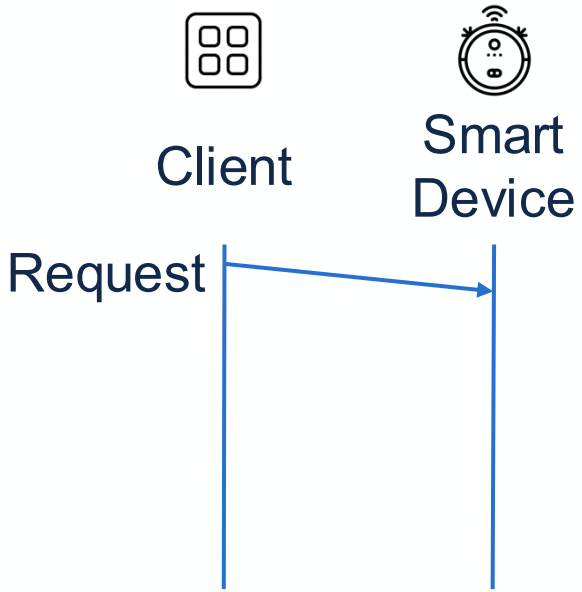


# The Core Mismatch with RPC

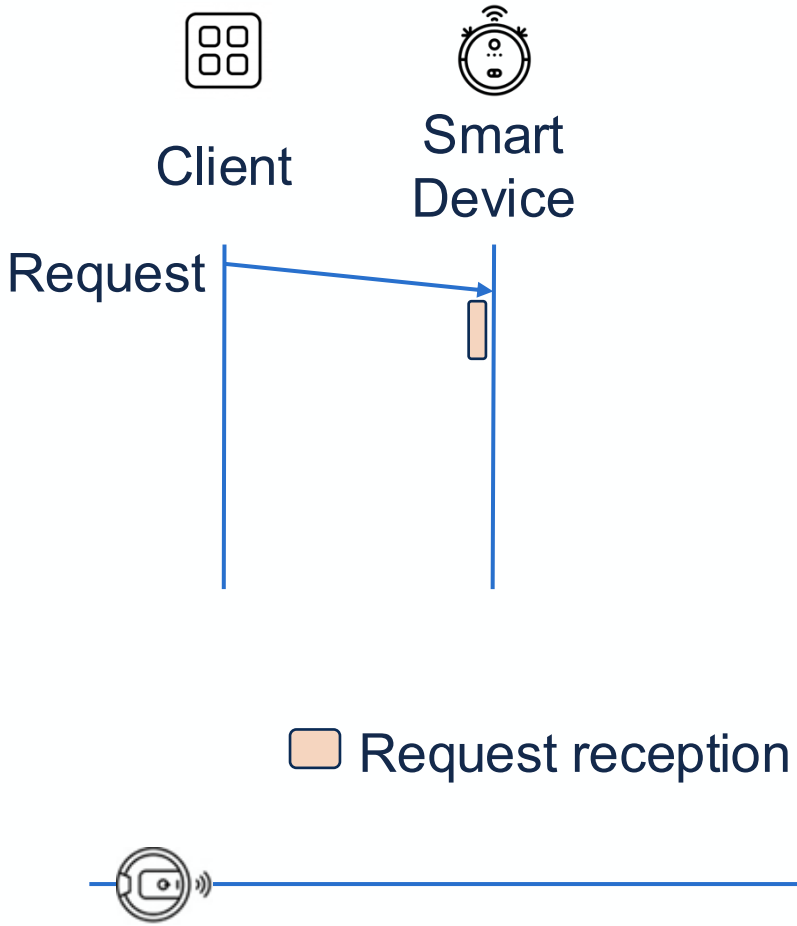
# The Core Mismatch with RPC



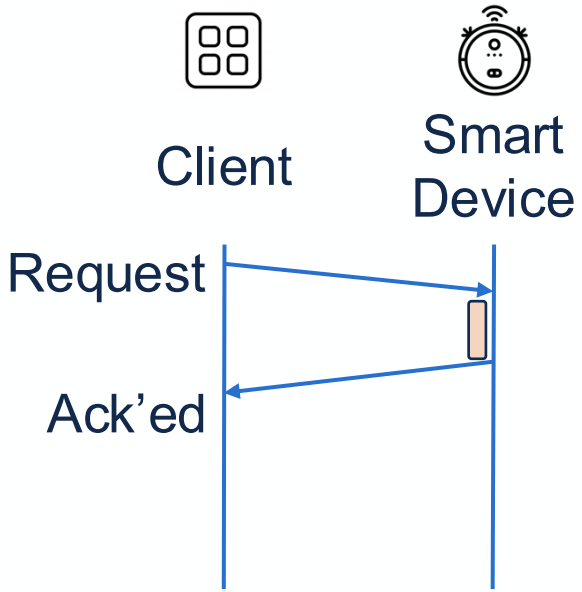

# The Core Mismatch with RPC




# The Core Mismatch with RPC

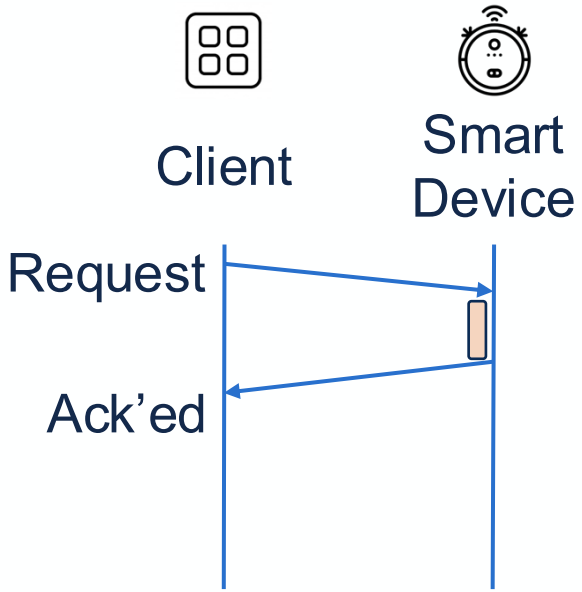


# The Core Mismatch with RPC




Request reception

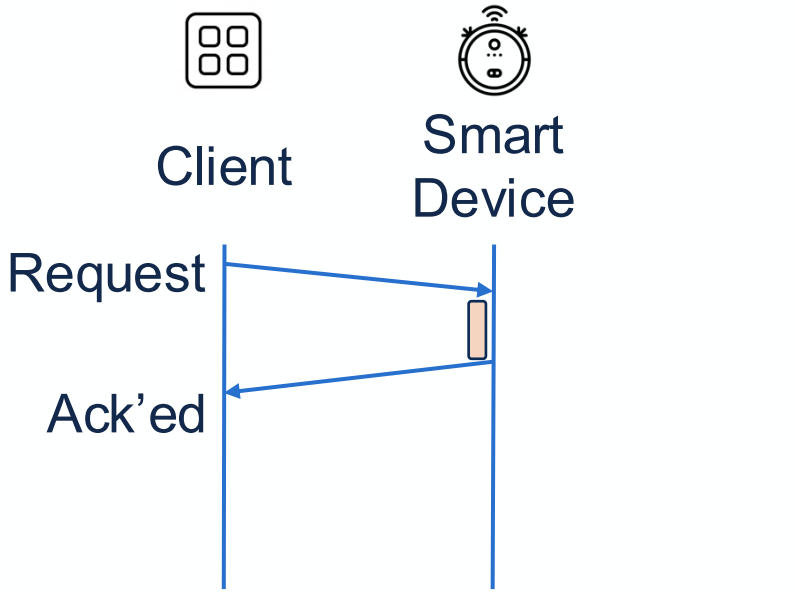
# The Core Mismatch with RPC




Request reception

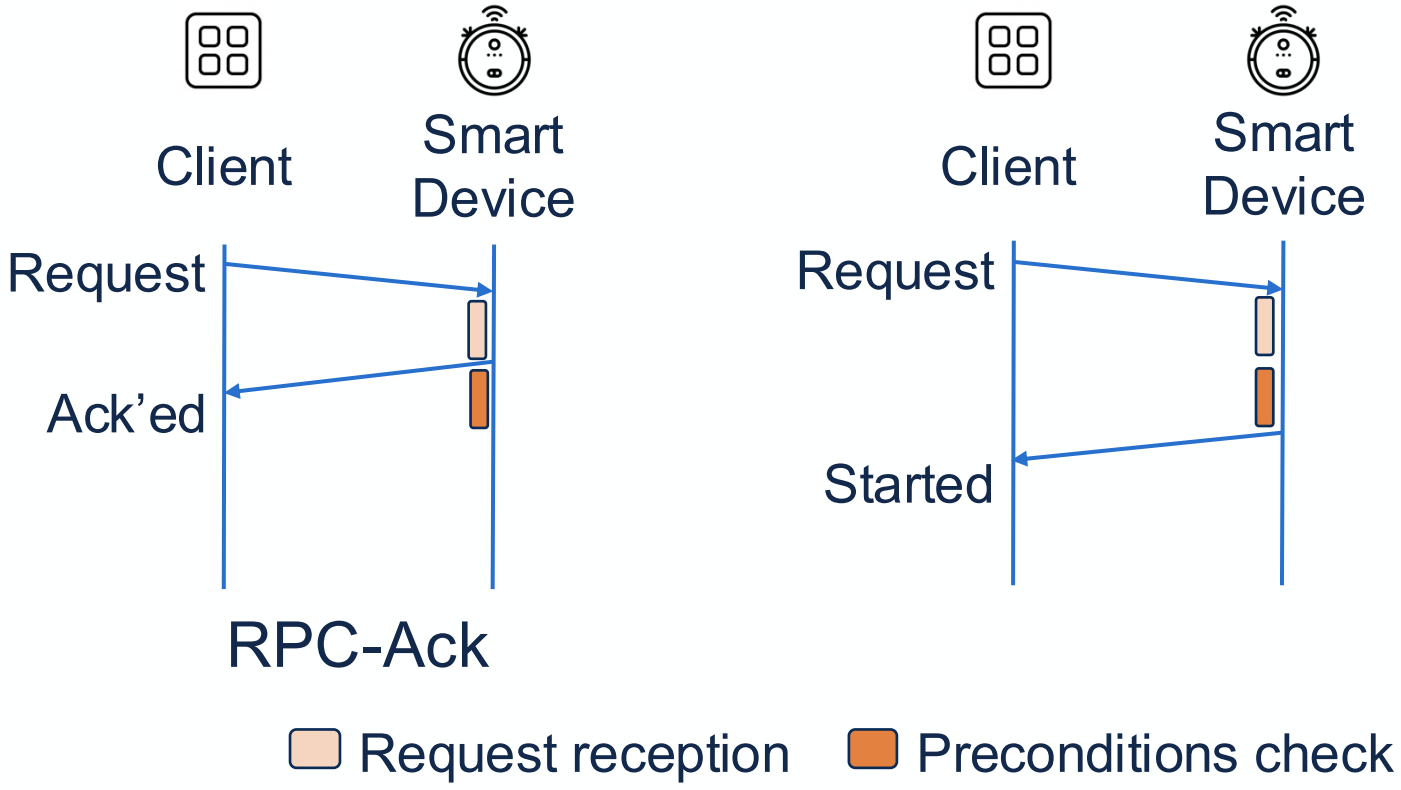
Vacuum **received request**

# The Core Mismatch with RPC

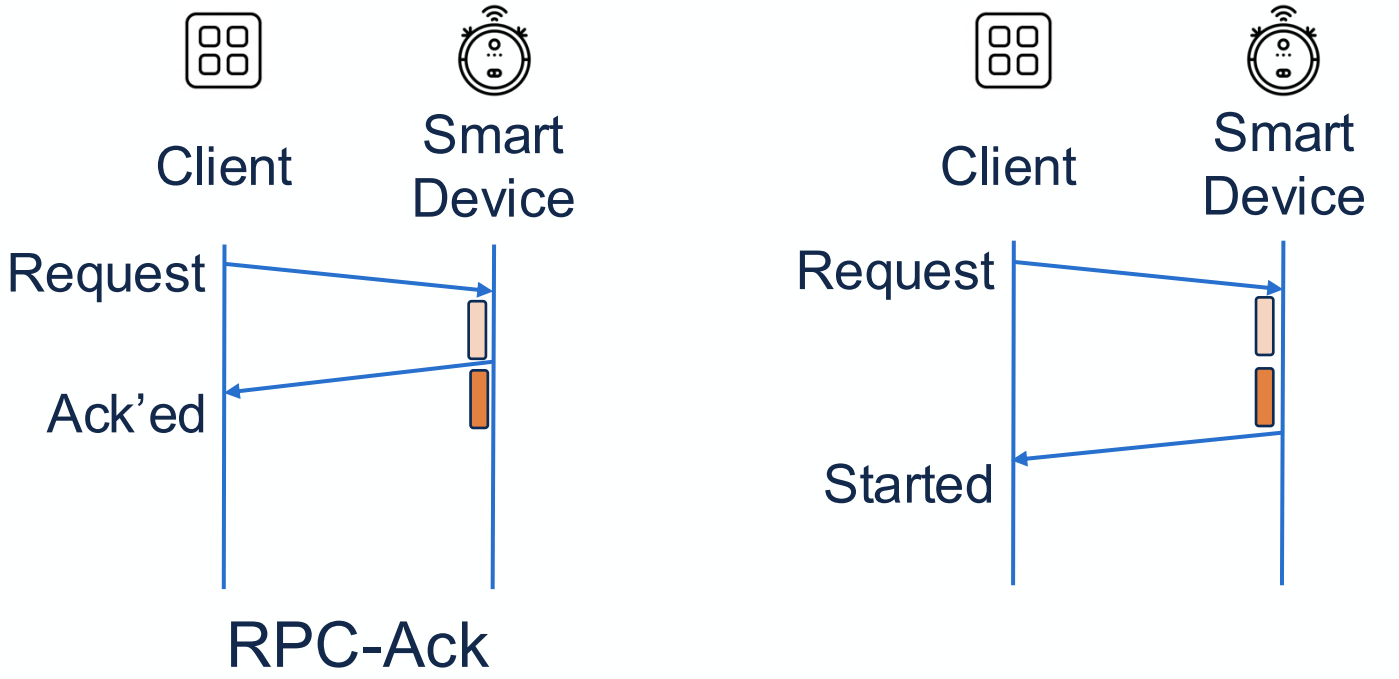



Request reception

# The Core Mismatch with RPC

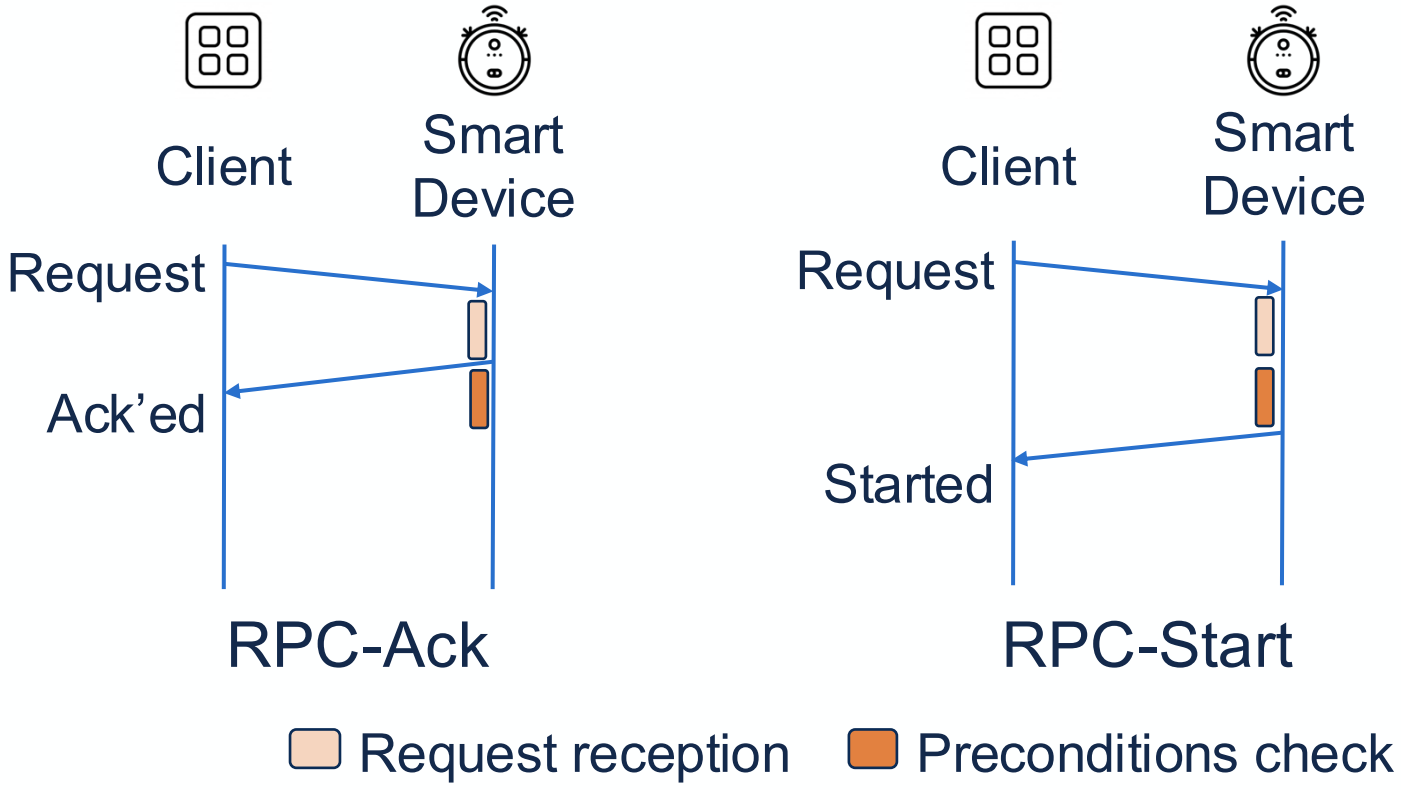



# The Core Mismatch with RPC

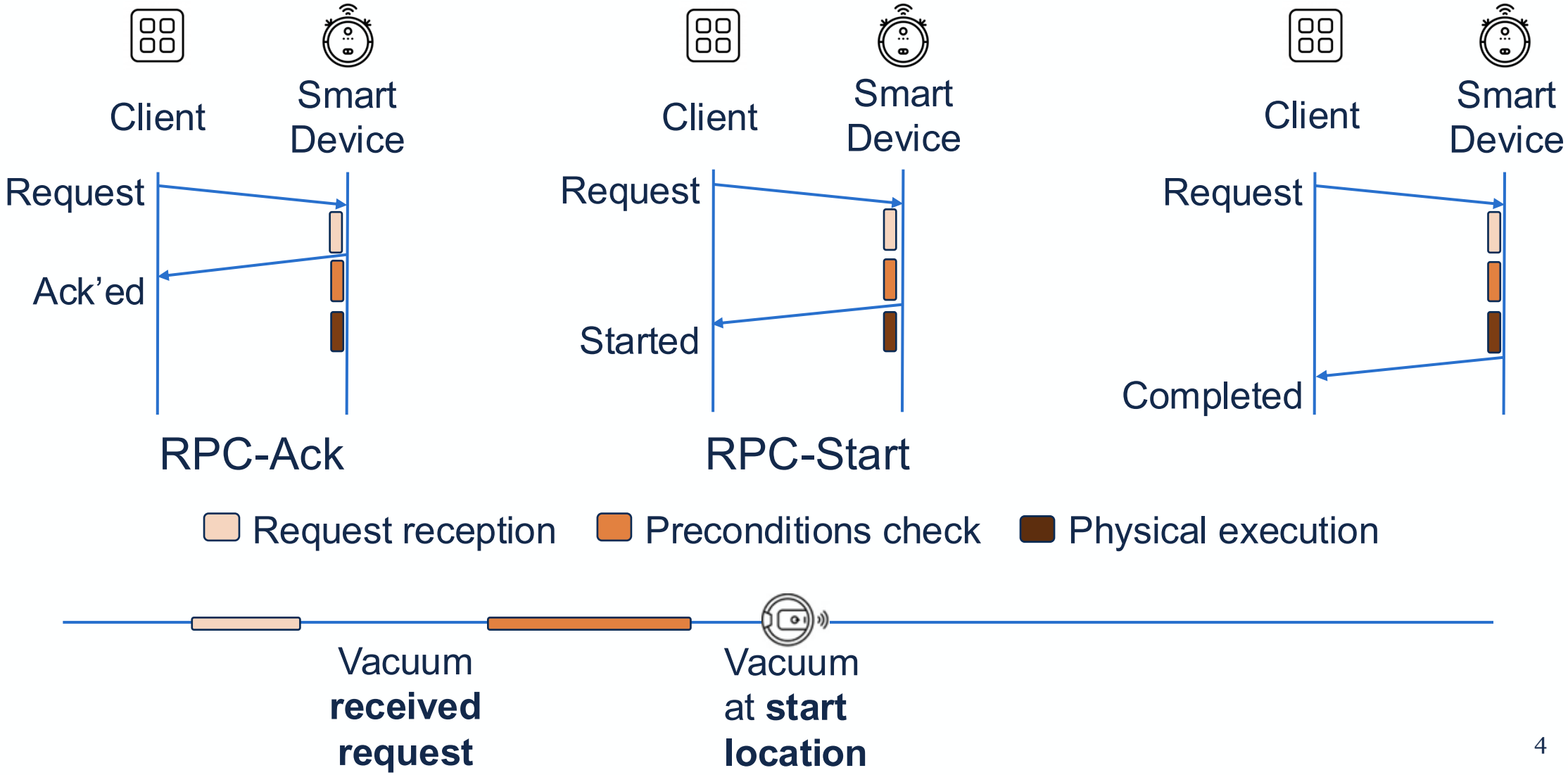



Request reception  Preconditions check

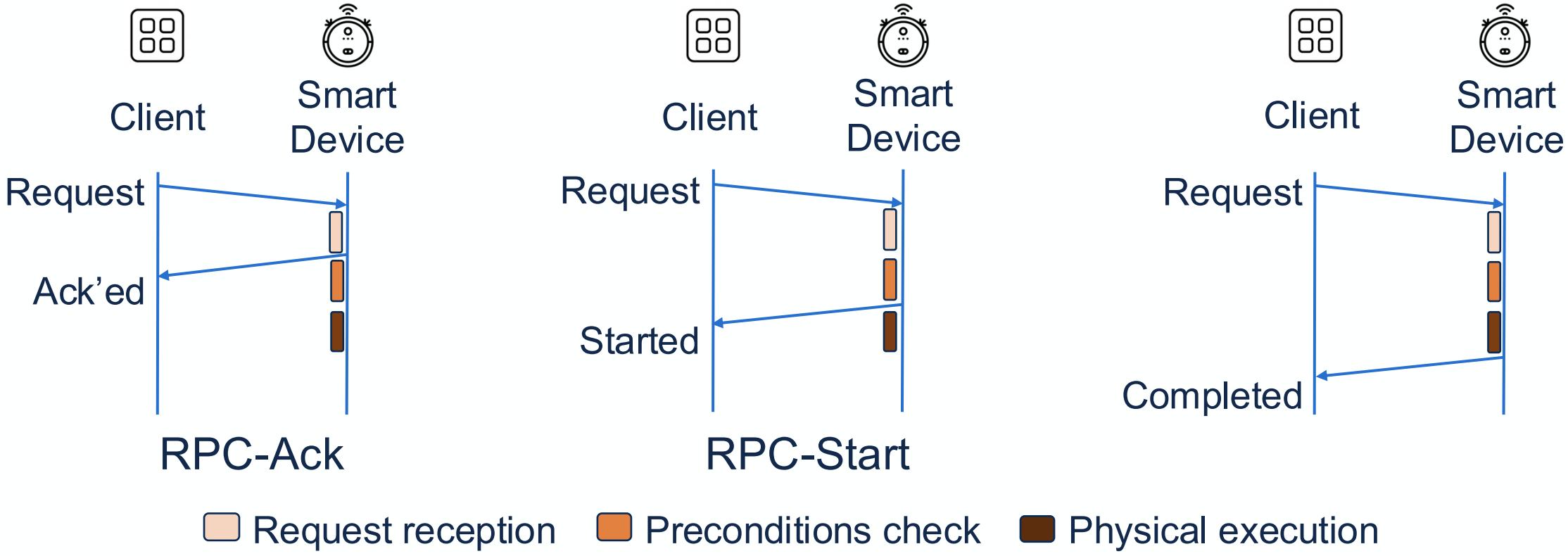
# The Core Mismatch with RPC



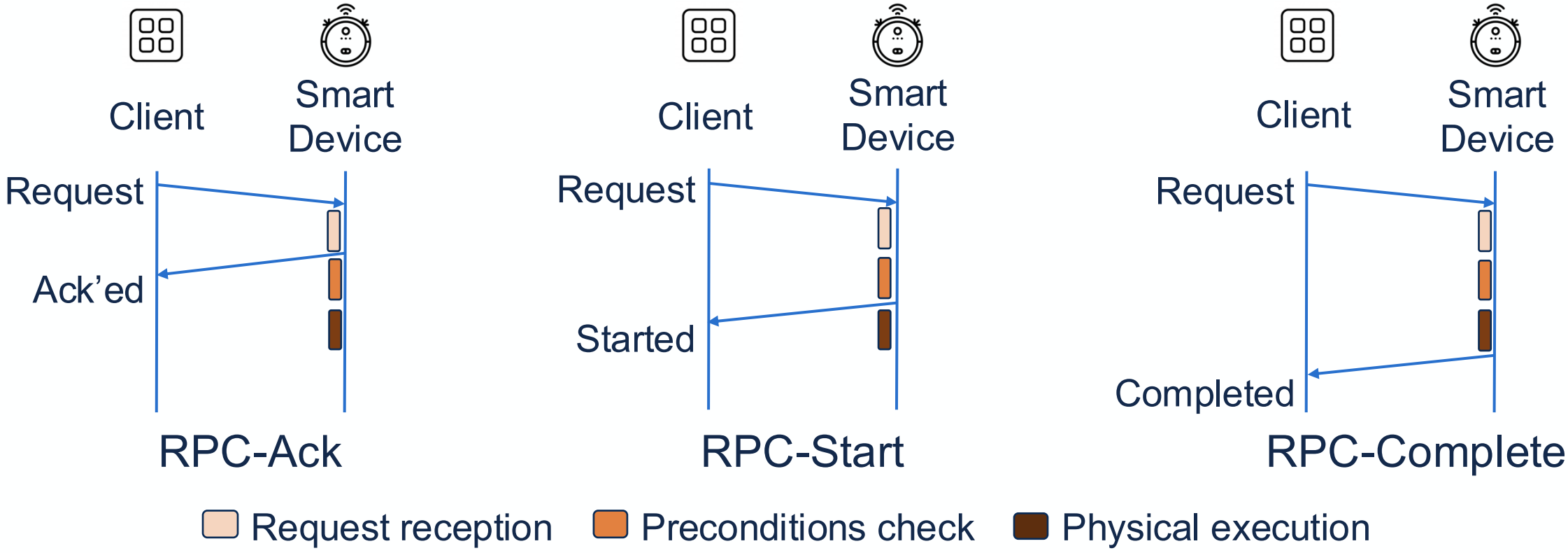
# The Core Mismatch with RPC



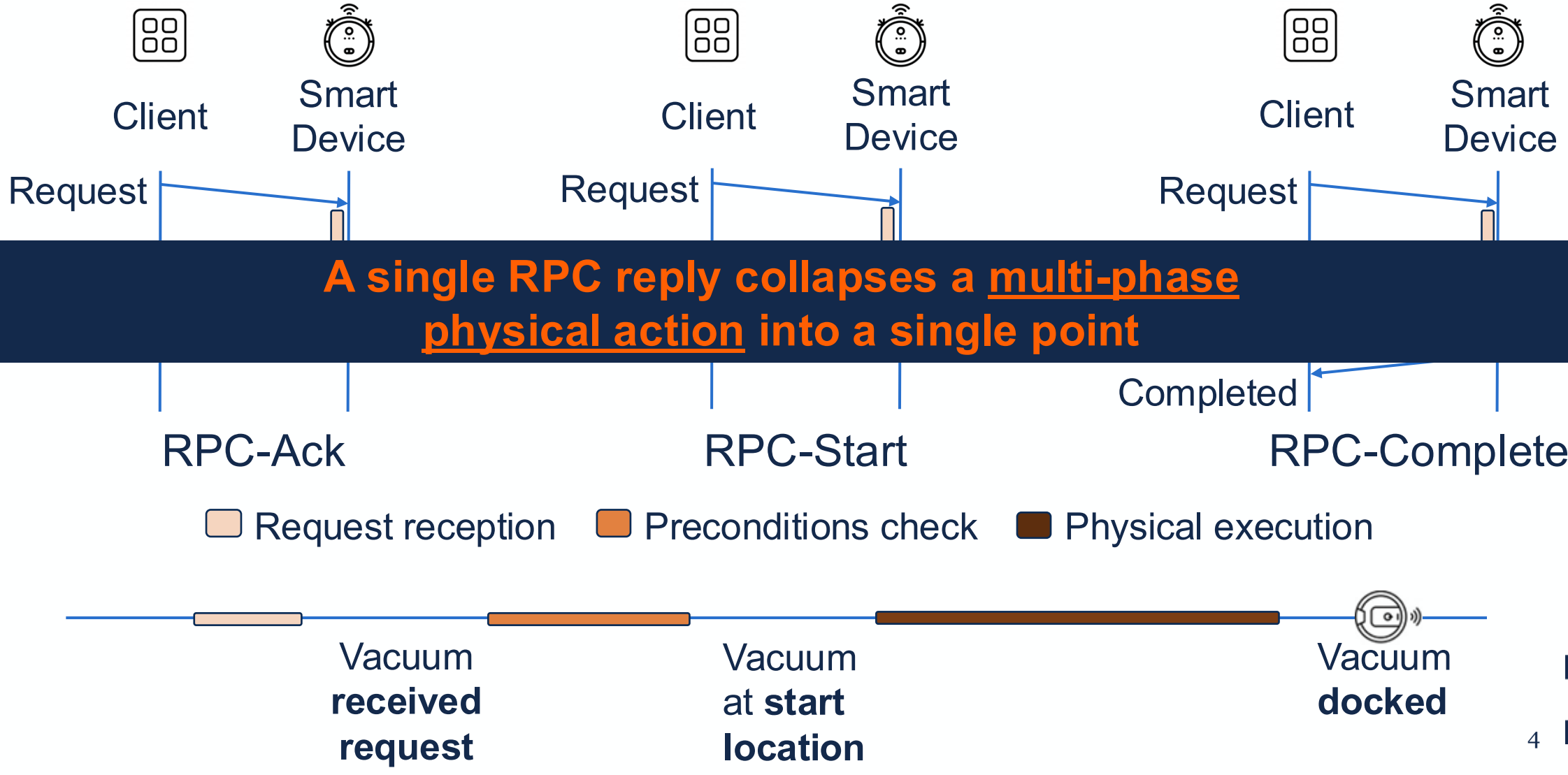
# The Core Mismatch with RPC



# The Core Mismatch with RPC



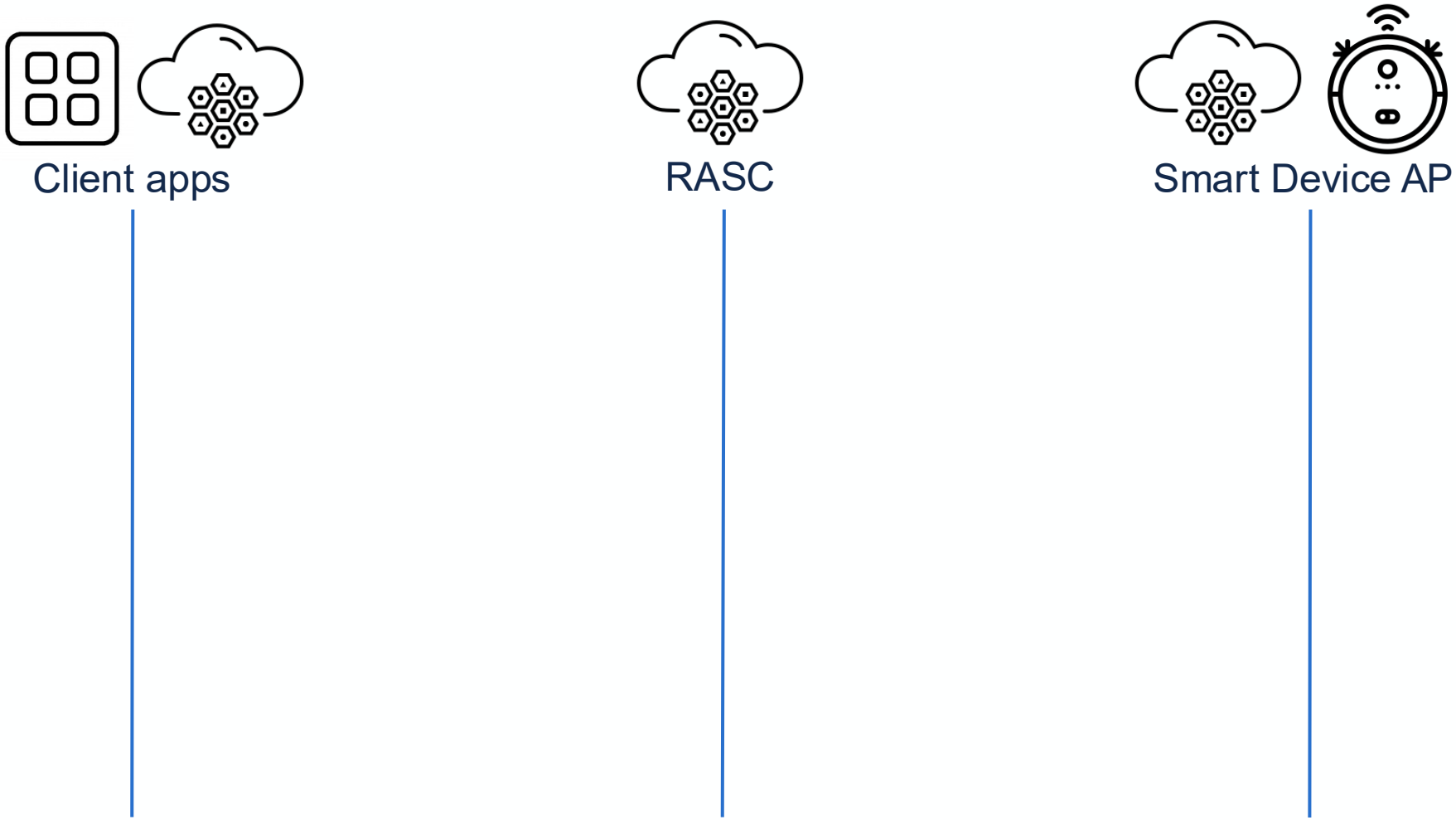
# The Core Mismatch with RPC



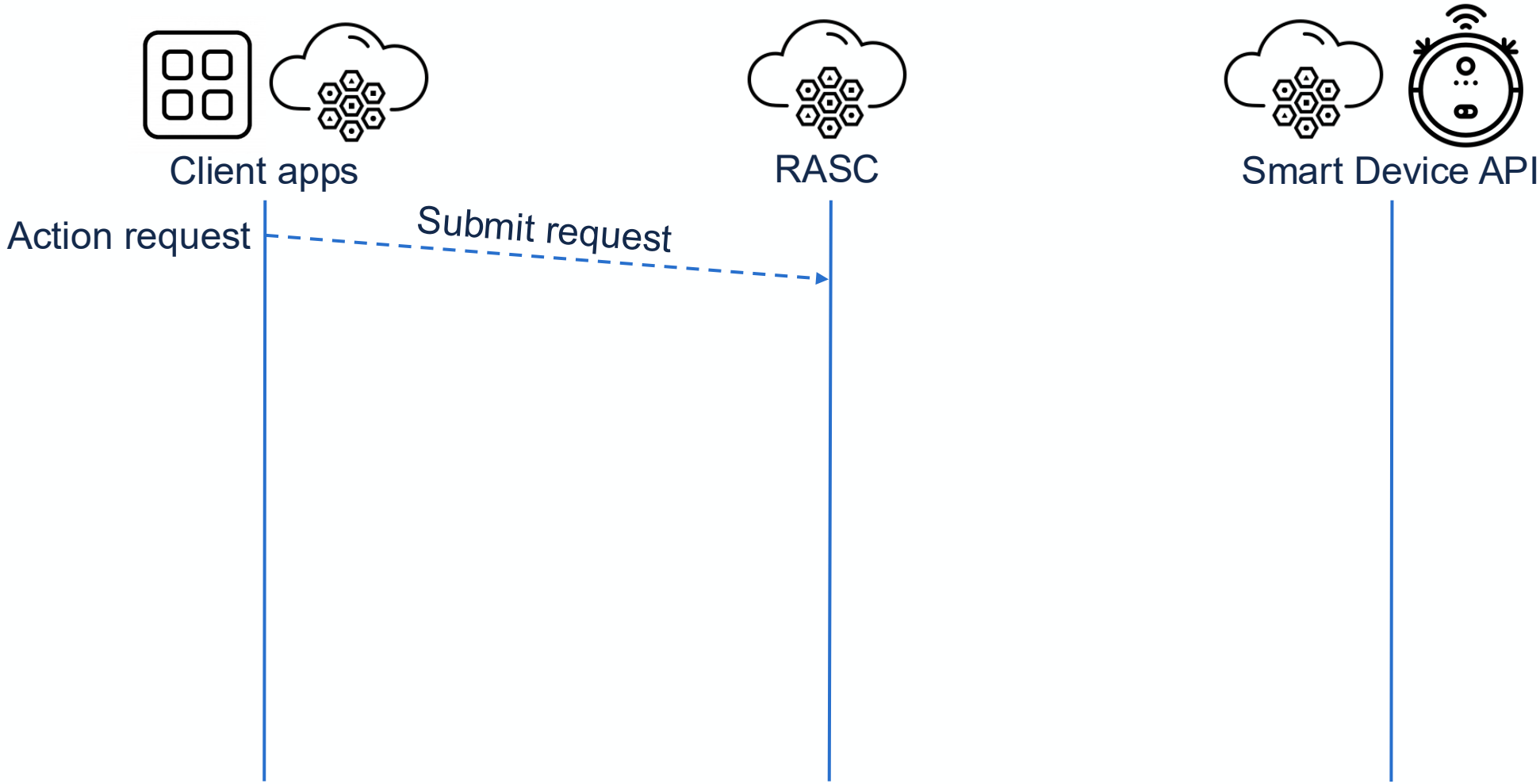
# Outline

- Introduction & Motivation
- RASC Abstraction
- Rascal Programmability: Building Atop RASC
- Rascal Observability: Implementing RASC
- Evaluation

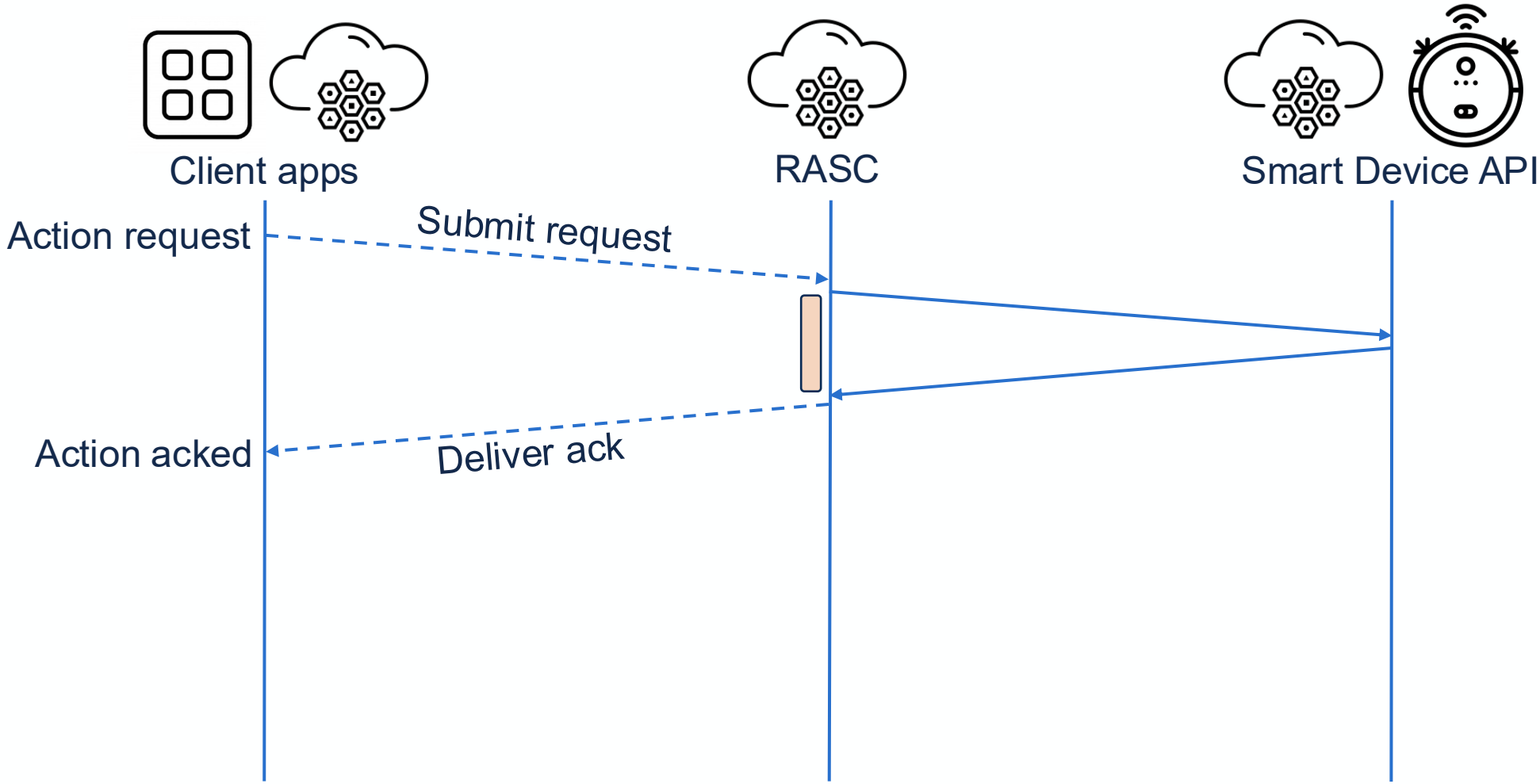
# RASC: Request-Ack-Start-Complete



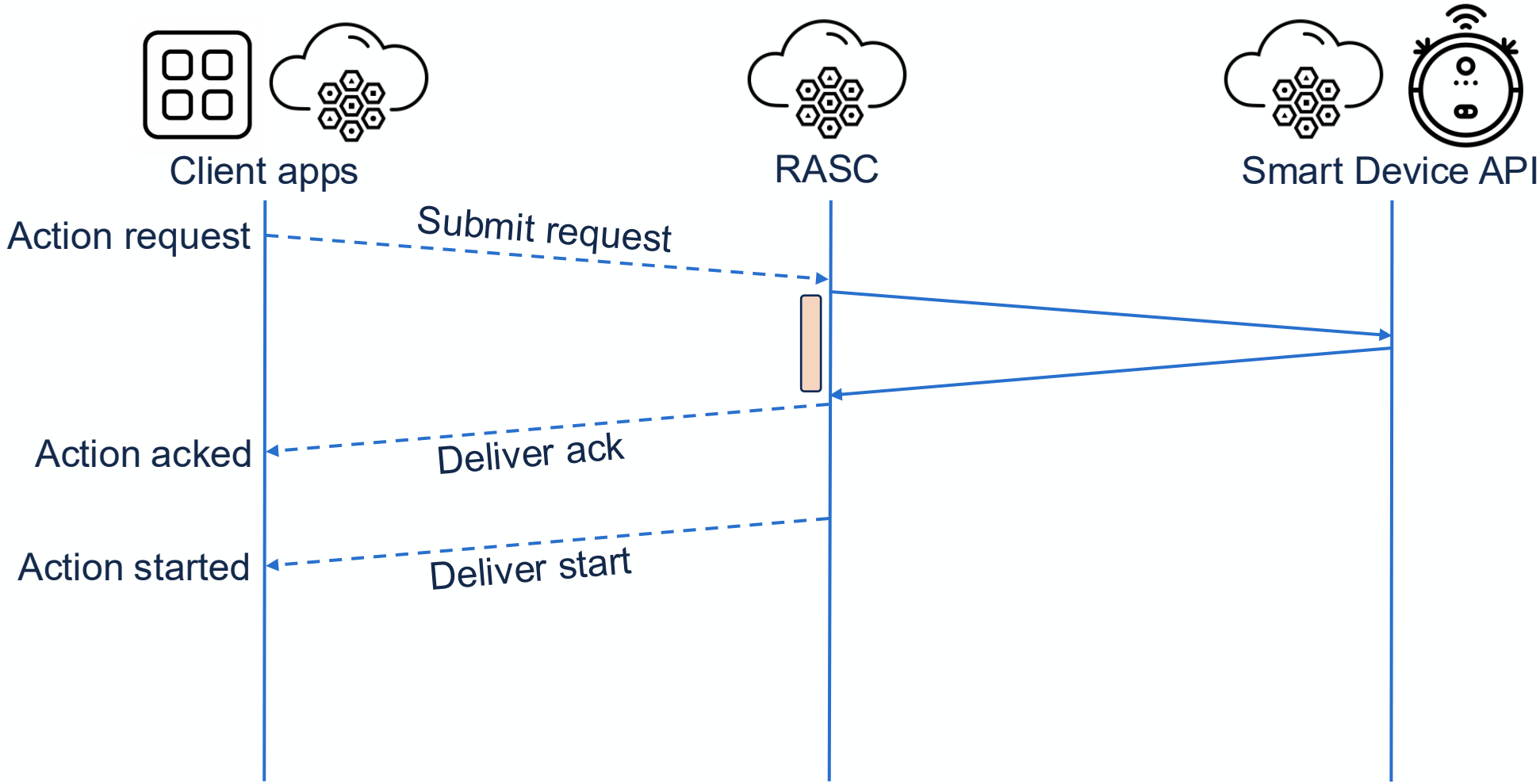
# RASC: Request-Ack-Start-Complete



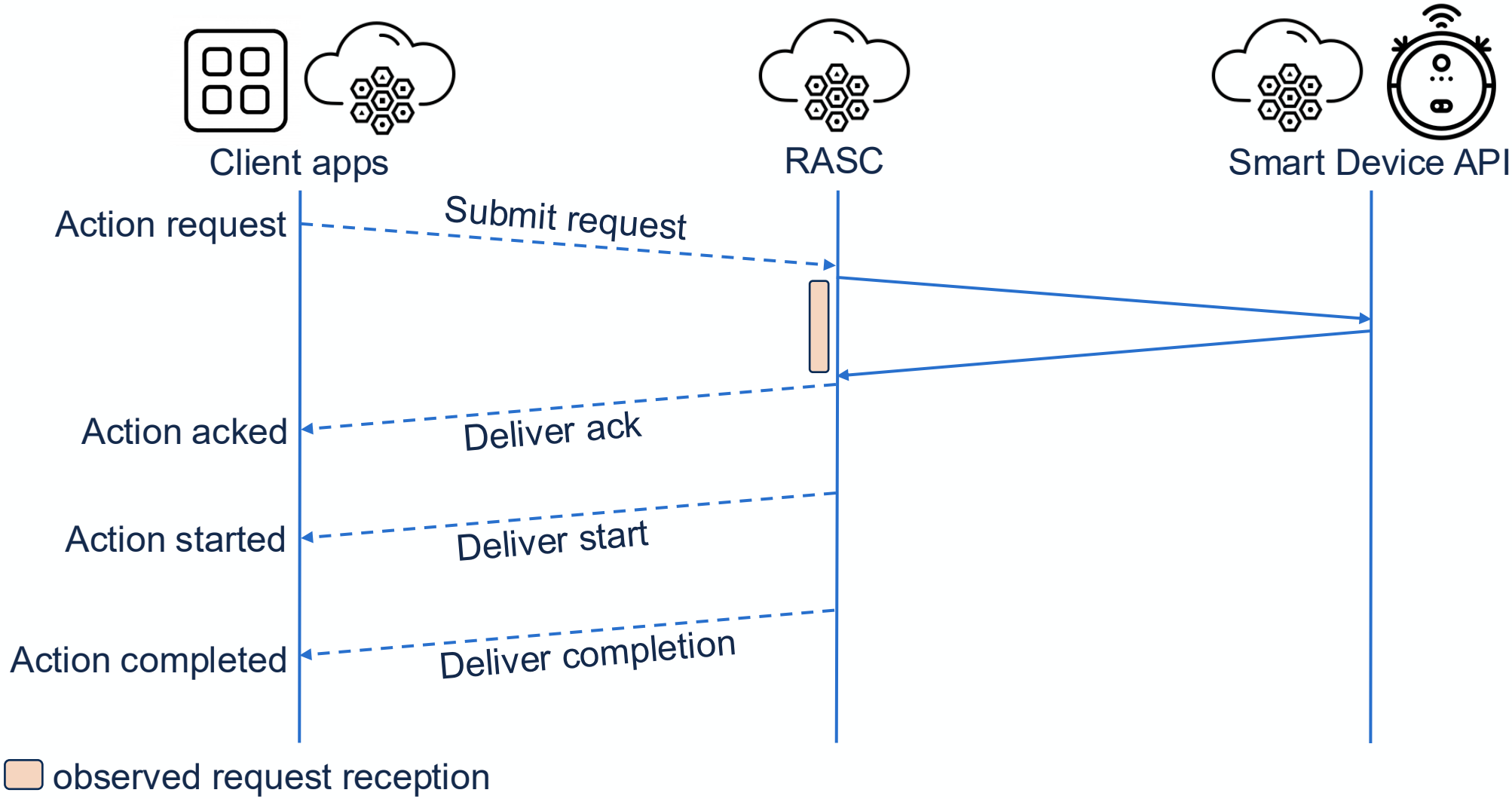
# RASC: Request-Ack-Start-Complete



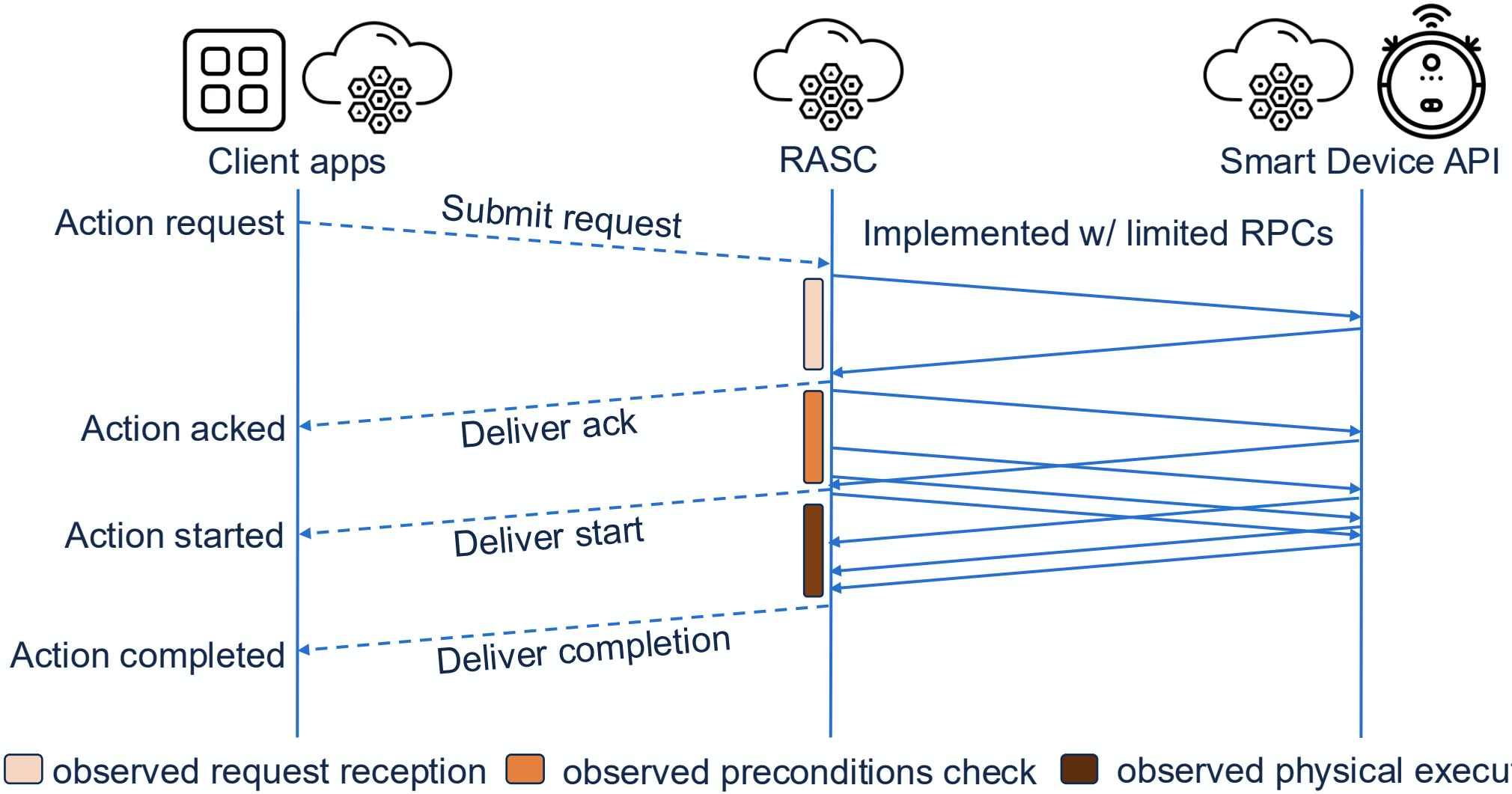
# RASC: Request-Ack-Start-Complete



# RASC: Request-Ack-Start-Complete



# RASC: Request-Ack-Start-Complete



# ↑ Observability → ↑ Programmability

A1: clean
living room

# ↑ Observability → ↑ Programmability

Goal: Vacuum cleans living room in an energy-efficient way

A1: clean living room

# ↑ Observability → ↑ Programmability

Goal: Vacuum cleans living room in an energy-efficient way

RPC-Ack

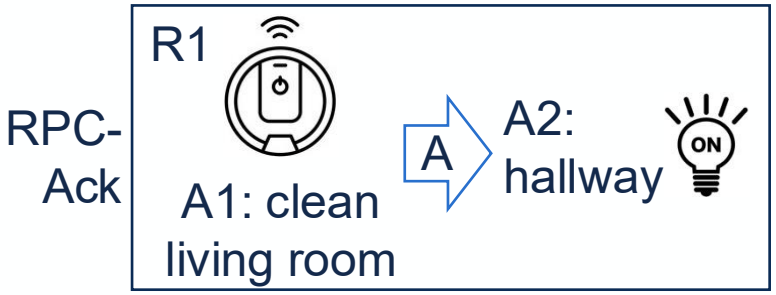
A1: clean living room

A

A2: hallway

# ↑ Observability → ↑ Programmability

Goal: Vacuum cleans living room in an energy-efficient way



# ↑ Observability → ↑ Programmability

Goal: Vacuum cleans living room in an energy-efficient way

RPC-Ack

R1

A1: clean living room

A

A2: hallway


R2

A3: living room


||

A4: hallway

# ↑ Observability → ↑ Programmability

Goal: Vacuum cleans living room in an energy-efficient way

RPC-Ack

R1

A1: clean living room

A

A2: hallway


R2

A3: living room


||

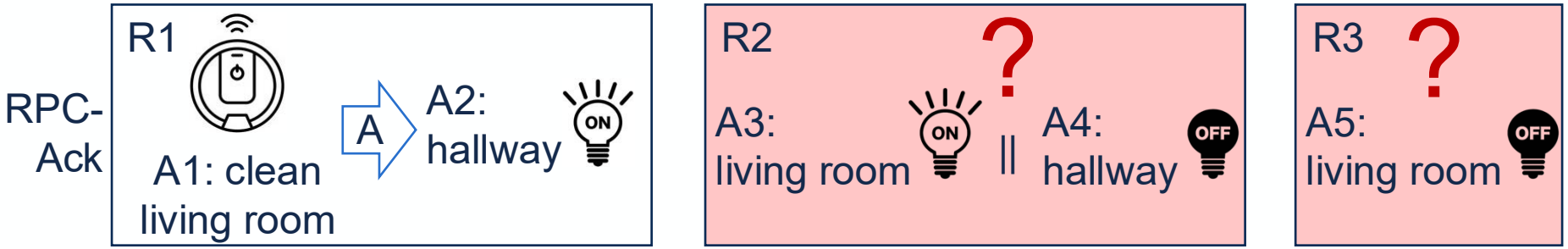
A4: hallway


R3

A5: living room

# ↑ Observability → ↑ Programmability

Goal: Vacuum cleans living room in an energy-efficient way

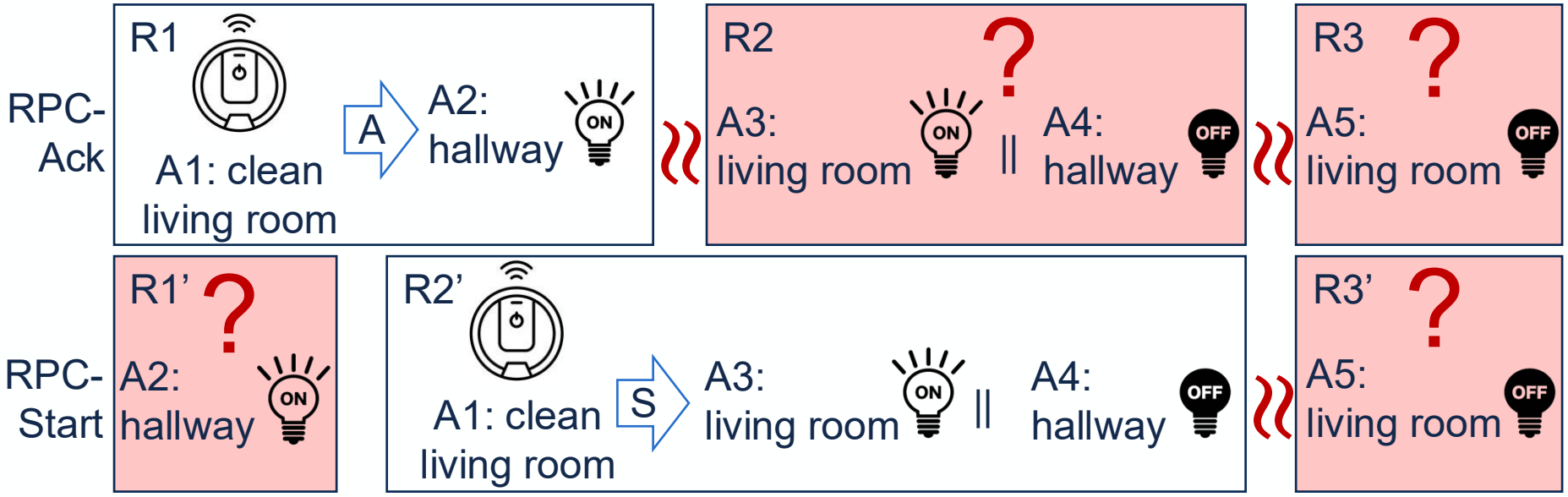


# ↑ Observability → ↑ Programmability

Goal: Vacuum cleans living room in an energy-efficient way

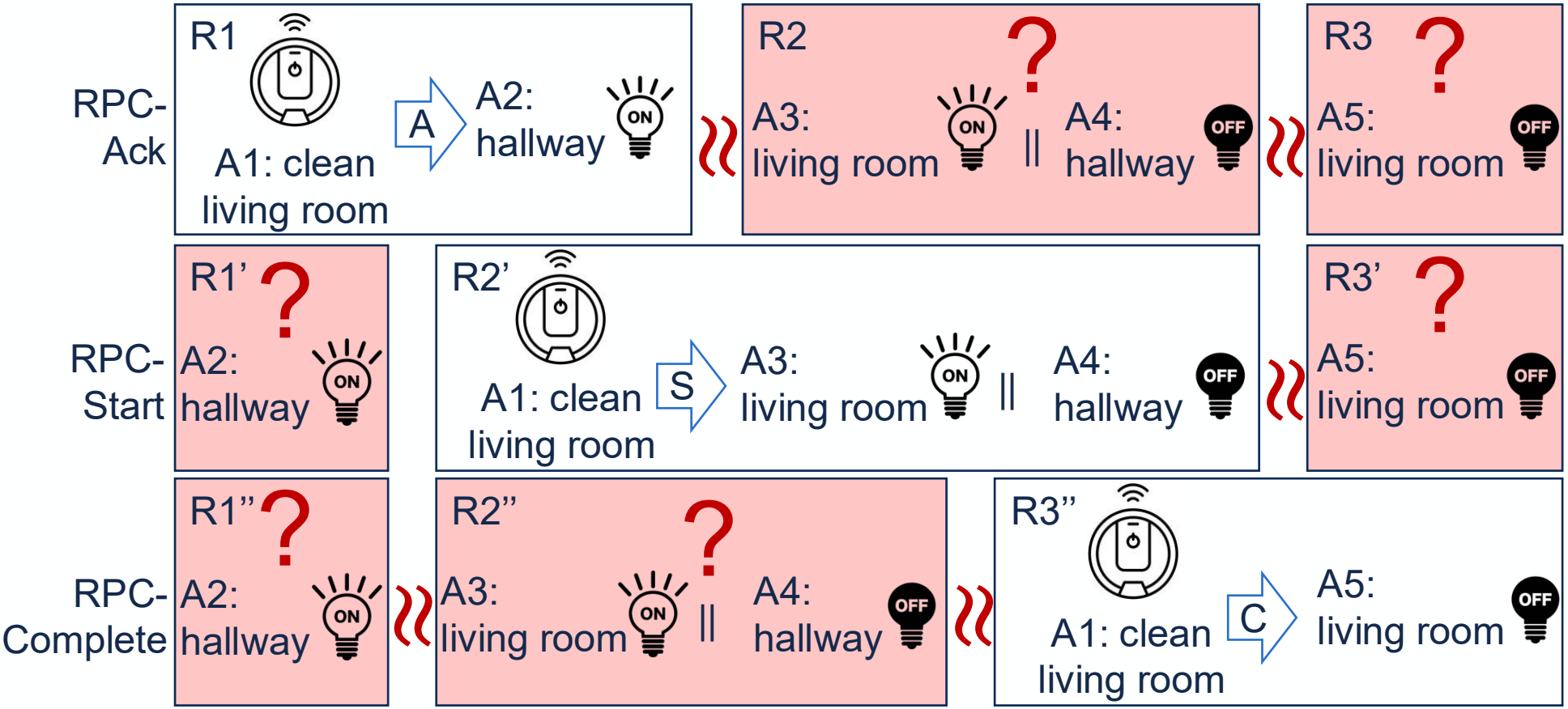
# ↑ Observability → ↑ Programmability

Goal: Vacuum cleans living room in an energy-efficient way



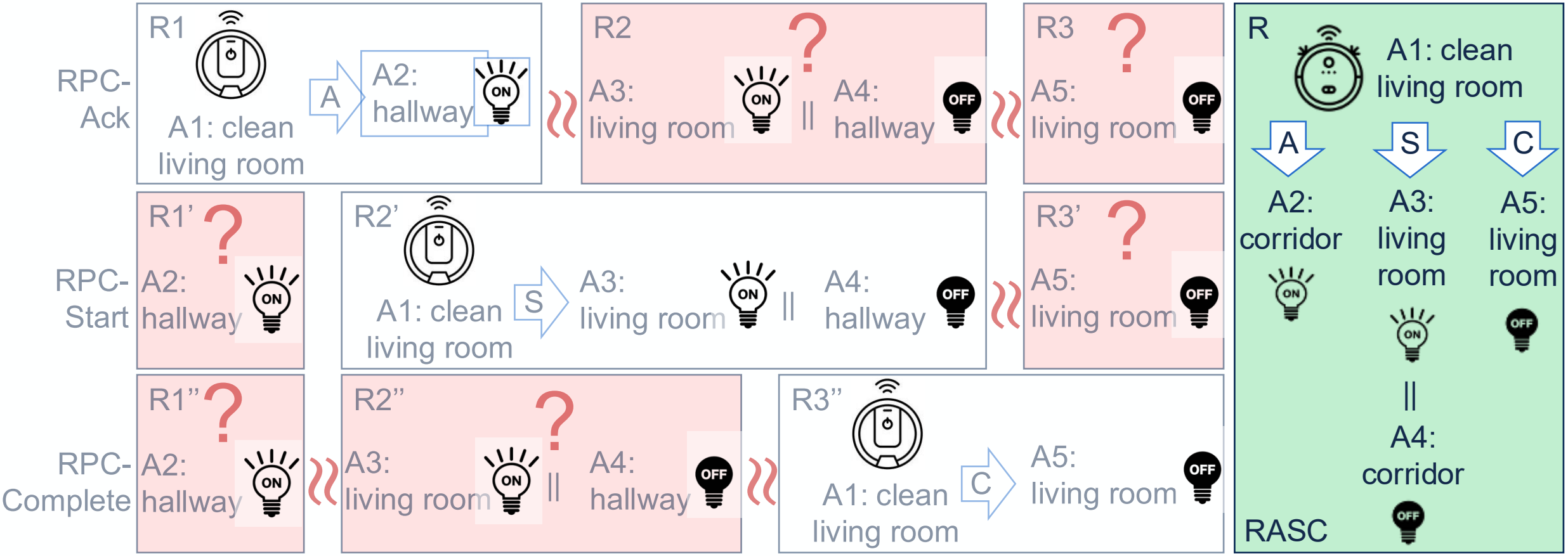
# ↑ Observability → ↑ Programmability

Goal: Vacuum cleans living room in an energy-efficient way



# ↑ Observability → ↑ Programmability

Goal: Vacuum cleans living room in an energy-efficient way



# Challenges and Opportunities

[1] Ahsan, S. B., Yang, R., Noghabi, S. A., & Gupta, I. (2021). Home, SafeHome: Smart Home Reliability with Visibility and Atomicity. In *EuroSys 21.*
[2] Karanika, A., Yang, R., Ma, X., Wang, J., Sundram, S., & Gupta, I. (2025). There is More Control in Egalitarian Edge IoT Meshes. IEEE TNSM 26.

# Challenges and Opportunities

- **Mutual Exclusion**
  - At most **one action per device at a time**
  - Must hold despite early starts and rescheduling

[1] Ahsan, S. B., Yang, R., Noghabi, S. A., & Gupta, I. (2021). Home, SafeHome: Smart Home Reliability with Visibility and Atomicity. In *EuroSys 21.*
[2] Karanika, A., Yang, R., Ma, X., Wang, J., Sundram, S., & Gupta, I. (2025). There is More Control in Egalitarian Edge IoT Meshes. IEEE TNSM 26.

# Challenges and Opportunities

- **Mutual Exclusion**
  - At most **one action per device at a time**
  - Must hold despite early starts and rescheduling
- **Serial Equivalence**
  - Concurrent routines execute equivalently to **some serial order**
  - Final device states are predictable and deterministic

[1] Ahsan, S. B., Yang, R., Noghabi, S. A., & Gupta, I. (2021). Home, SafeHome: Smart Home Reliability with Visibility and Atomicity. In *EuroSys 21.*
[2] Karanika, A., Yang, R., Ma, X., Wang, J., Sundram, S., & Gupta, I. (2025). There is More Control in Egalitarian Edge IoT Meshes. IEEE TNSM 26.

# Challenges and Opportunities

- **Mutual Exclusion**
  - At most **one action per device at a time**
  - Must hold despite early starts and rescheduling
- **Serial Equivalence**
  - Concurrent routines execute equivalently to **some serial order**
  - Final device states are predictable and deterministic
- **Polling Efficiency**
  - Progress detection without overwhelming devices
  - Meet detection deadlines under API limits

[1] Ahsan, S. B., Yang, R., Noghabi, S. A., & Gupta, I. (2021). Home, SafeHome: Smart Home Reliability with Visibility and Atomicity. In *EuroSys 21.*
[2] Karanika, A., Yang, R., Ma, X., Wang, J., Sundram, S., & Gupta, I. (2025). There is More Control in Egalitarian Edge IoT Meshes. IEEE TNSM 26.

# Outline

- Introduction & Motivation
- RASC Abstraction
- Rascal Programmability: Building Atop RASC
- Rascal Observability: Implementing RASC
- Evaluation

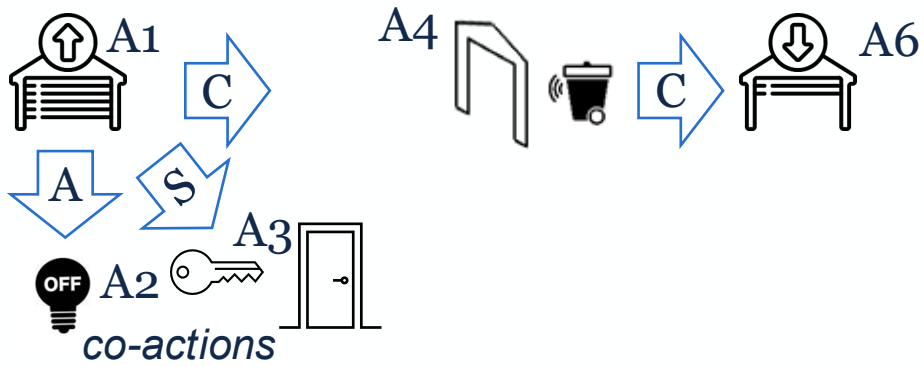
# RASC Enables Expressive Dependencies

# RASC Enables Expressive Dependencies

- Beyond sequential actions:
  - **Co-actions** (after ack / start)
  - **Fallbacks** (on failure)

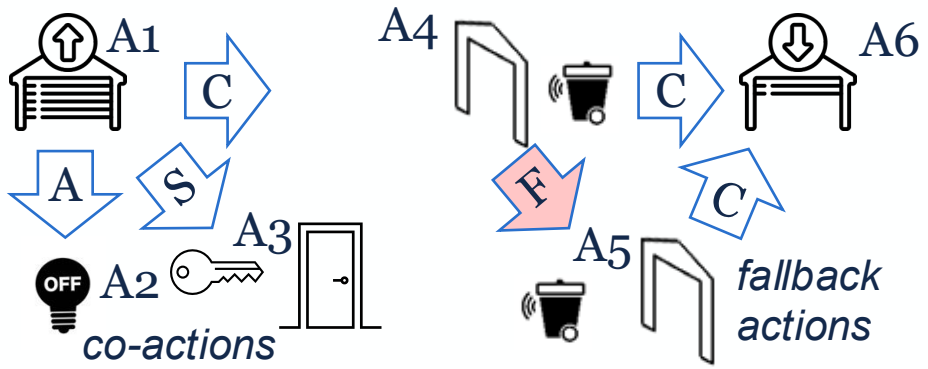
# RASC Enables Expressive Dependencies

- Beyond sequential actions:
  - **Co-actions** (after ack / start)
  - **Fallbacks** (on failure)



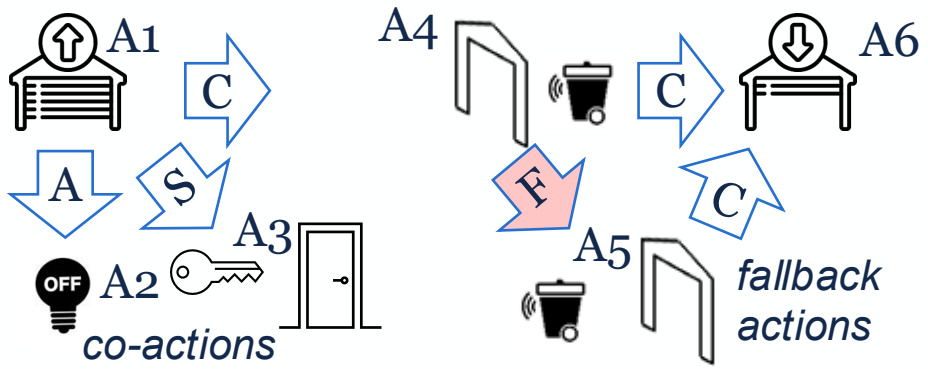
# RASC Enables Expressive Dependencies

- Beyond sequential actions:
  - **Co-actions** (after ack / start)
  - **Fallbacks** (on failure)



# RASC Enables Expressive Dependencies

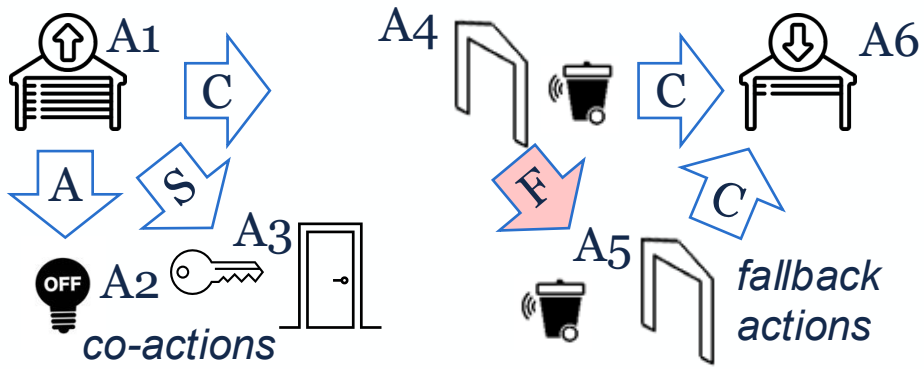
- Beyond sequential actions:
  - **Co-actions** (after ack / start)
  - **Fallbacks** (on failure)




- Routines are expressed as **DAGs**
  - With dependency types

# RASC Enables Expressive Dependencies

- Beyond sequential actions:
  - **Co-actions** (after ack / start)
  - **Fallbacks** (on failure)




- Routines are expressed as **DAGs**
  - With dependency types

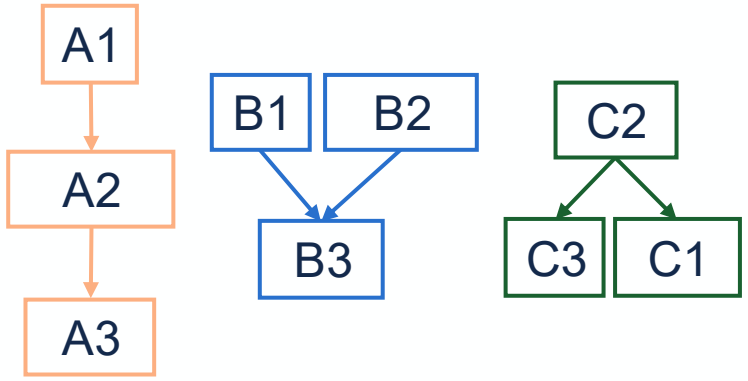
Abstraction **RASC** → System **Rascal**

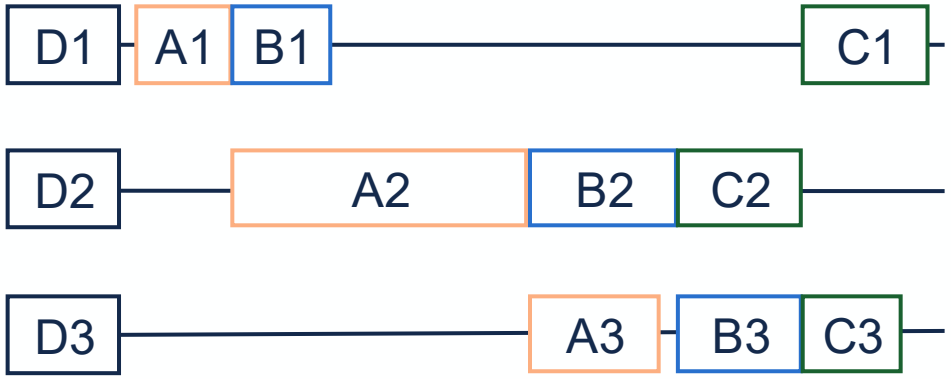
# RASC Enables Rescheduling

[1] Manimaran, G., Murthy, C. S. R., Vijay, M., & Ramamritham, K. (1997). New algorithms for resource reclaiming from precedence constrained tasks in multiprocessor real-time systems. Journal of Parallel and Distributed Computing, 44(2), 123-132.

# RASC Enables Rescheduling

- Routine arrives
  - DAG-TL (DAG Timeline Scheduler)

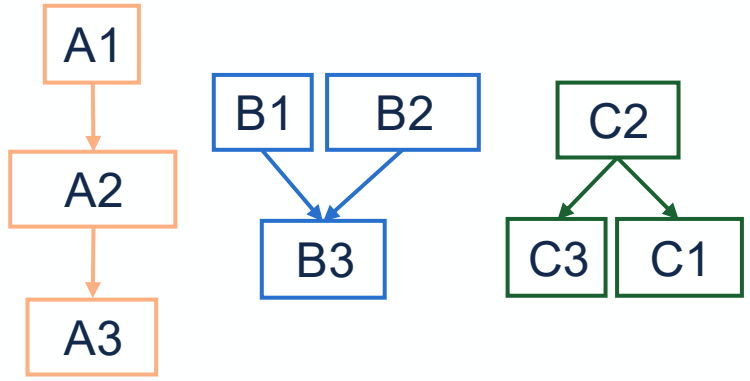


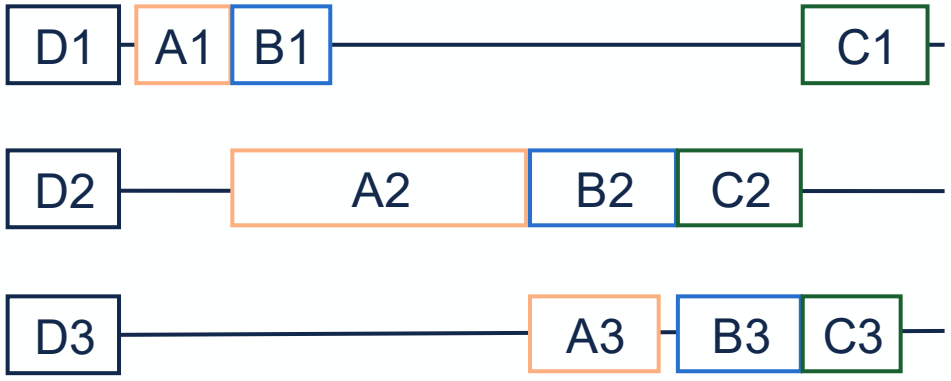




[1] Manimaran, G., Murthy, C. S. R., Vijay, M., & Ramamritham, K. (1997). New algorithms for resource reclaiming from precedence constrained tasks in multiprocessor real-time systems. Journal of Parallel and Distributed Computing, 44(2), 123-132.

# RASC Enables Rescheduling

- Routine arrives
  - DAG-TL (DAG Timeline Scheduler)
- When execution deviates

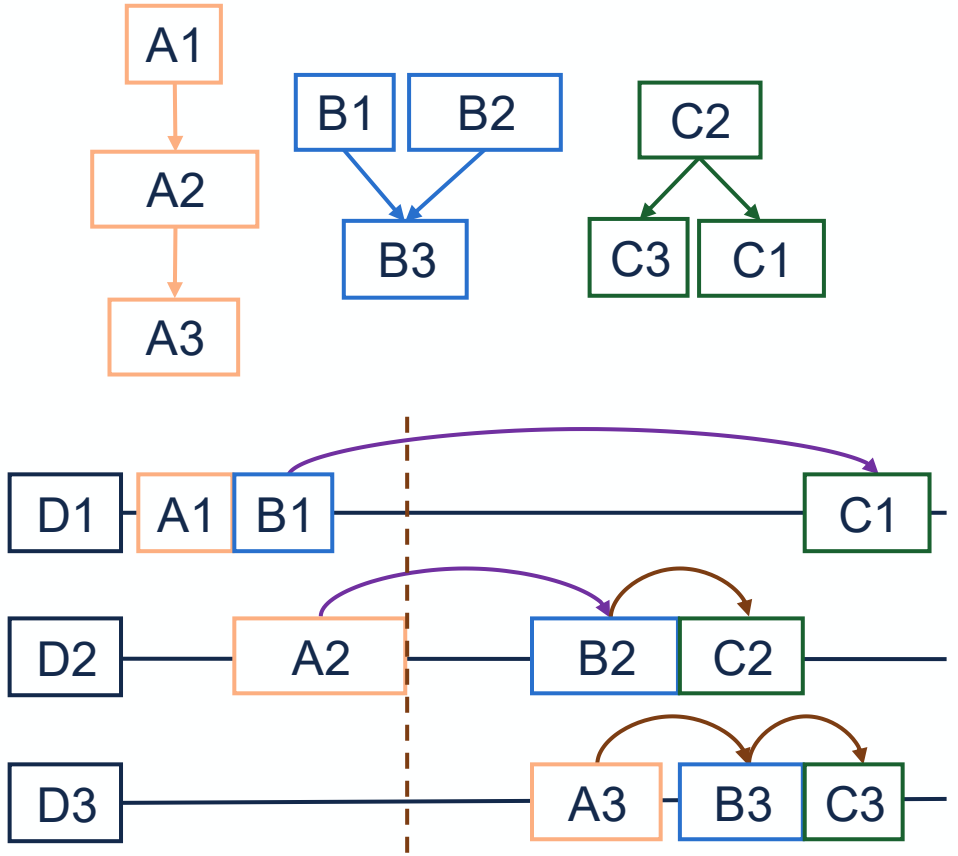



[1] Manimaran, G., Murthy, C. S. R., Vijay, M., & Ramamritham, K. (1997). New algorithms for resource reclaiming from precedence constrained tasks in multiprocessor real-time systems. Journal of Parallel and Distributed Computing, 44(2), 123-132.

# RASC Enables Rescheduling

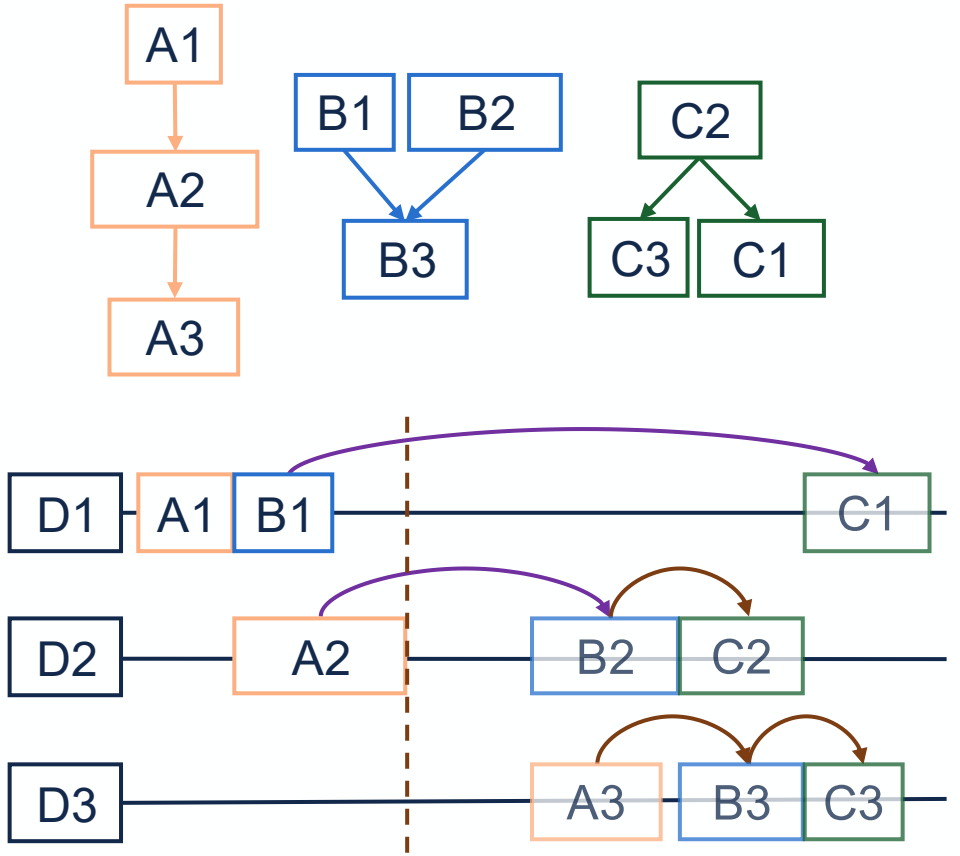
- Routine arrives
  - DAG-TL (DAG Timeline Scheduler)
- When execution deviates
  - Identify affected actions




[1] Manimaran, G., Murthy, C. S. R., Vijay, M., & Ramamritham, K. (1997). New algorithms for resource reclaiming from precedence constrained tasks in multiprocessor real-time systems. Journal of Parallel and Distributed Computing, 44(2), 123-132.

# RASC Enables Rescheduling

- Routine arrives
  - DAG-TL (DAG Timeline Scheduler)
- When execution deviates
  - Identify affected actions
  - Immutable serialization order

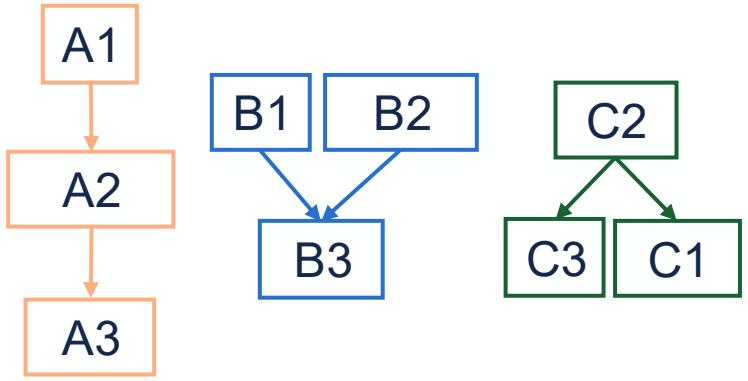


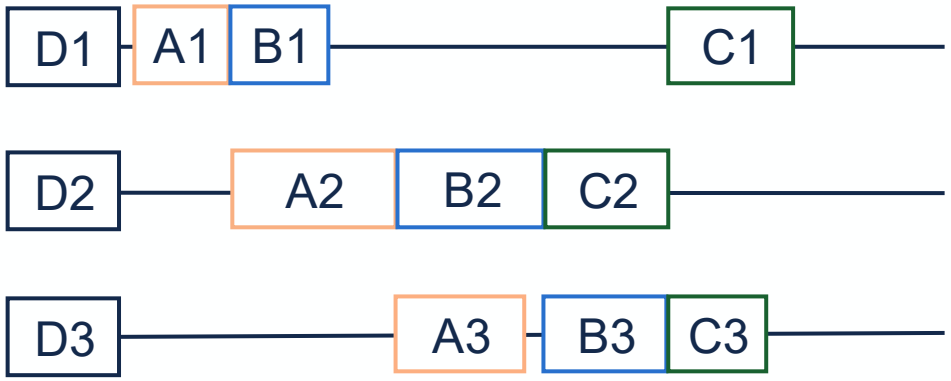

Immutable serialization order: $R_A \rightarrow R_B \rightarrow R_C$

[1] Manimaran, G., Murthy, C. S. R., Vijay, M., & Ramamritham, K. (1997). New algorithms for resource reclaiming from precedence constrained tasks in multiprocessor real-time systems. Journal of Parallel and Distributed Computing, 44(2), 123-132.

# RASC Enables Rescheduling

- Routine arrives
  - DAG-TL (DAG Timeline Scheduler)
- When execution deviates
  - Identify affected actions
  - Immutable serialization order
- Reschedule with policies adapting
  - STF (Shortest Task First)
  - RV (Restriction Vectors)



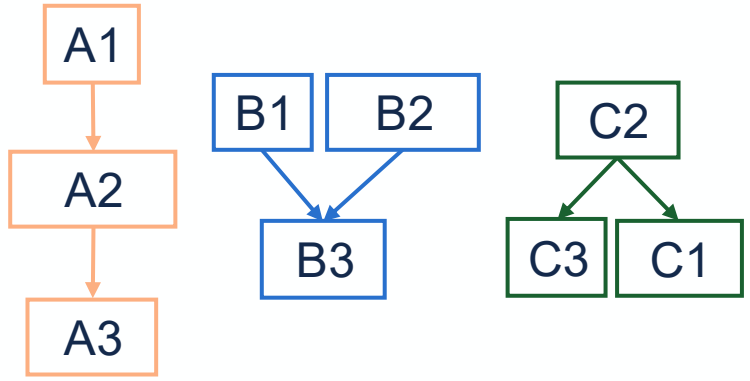



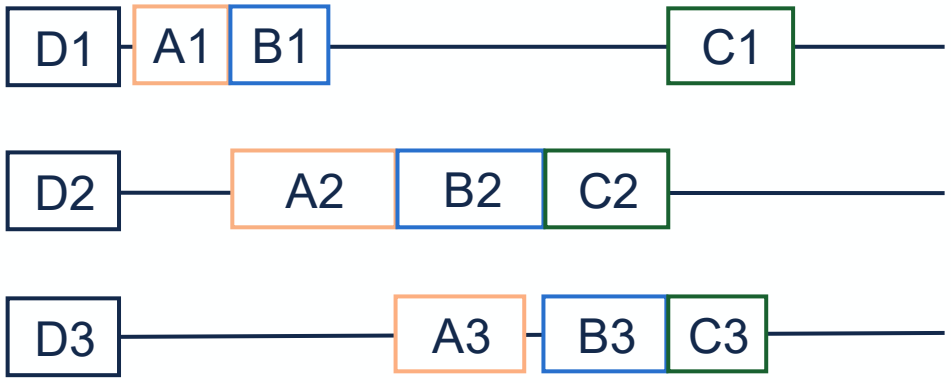

Immutable serialization order: $R_A \rightarrow R_B \rightarrow R_C$

[1] Manimaran, G., Murthy, C. S. R., Vijay, M., & Ramamritham, K. (1997). New algorithms for resource reclaiming from precedence constrained tasks in multiprocessor real-time systems. Journal of Parallel and Distributed Computing, 44(2), 123-132.

# RASC Enables Rescheduling

- Routine arrives
  - DAG-TL (DAG Timeline Scheduler)
- When execution deviates
  - Identify affected actions
  - Immutable serialization order
- Reschedule with policies adapting
  - STF (Shortest Task First)
  - RV (Restriction Vectors)
- Maintaining safety guarantees



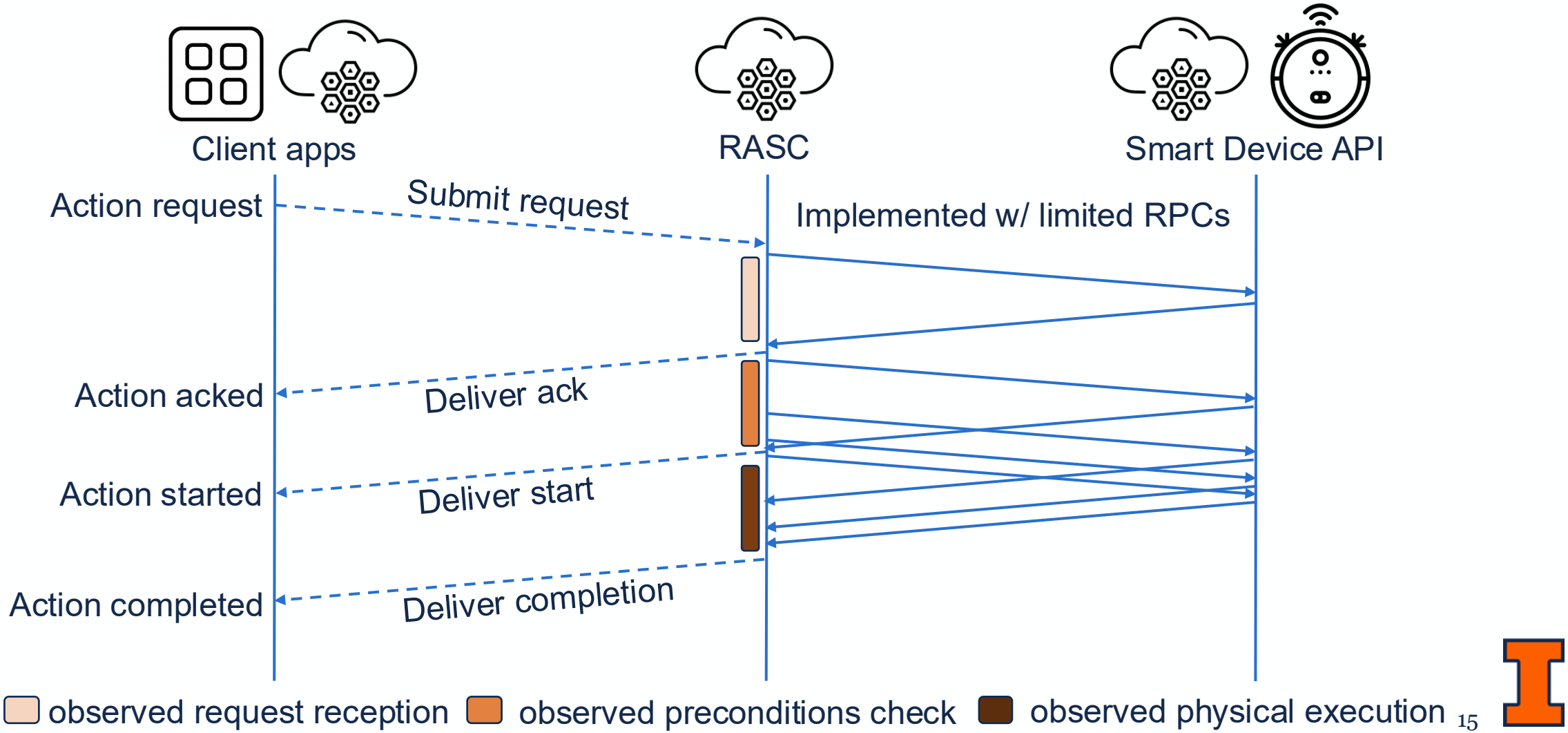

Immutable serialization order: $R_A \rightarrow R_B \rightarrow R_C$

[1] Manimaran, G., Murthy, C. S. R., Vijay, M., & Ramamritham, K. (1997). New algorithms for resource reclaiming from precedence constrained tasks in multiprocessor real-time systems. Journal of Parallel and Distributed Computing, 44(2), 123-132.

# Outline

- Introduction & Motivation
- RASC Abstraction
- Rascal Programmability: Building Atop RASC
- Rascal Observability: Implementing RASC
- Evaluation

# RASC: Request-Ack-Start-Complete



# Challenge: Polling Placement is Hard

# Challenge: Polling Placement is Hard

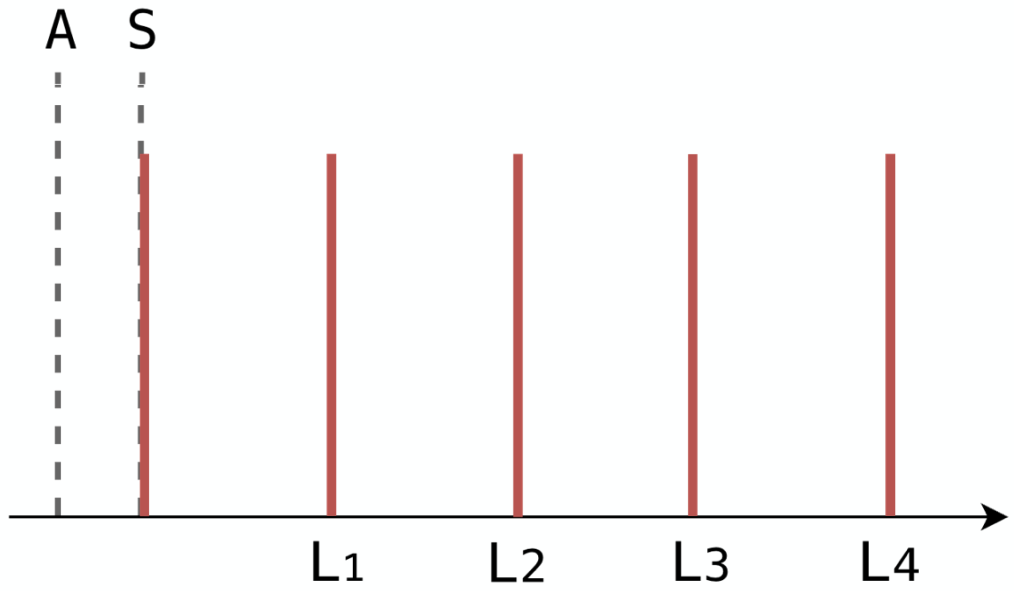
- **Constraint**: Large fraction of device vendors offer poll-only APIs



# Challenge: Polling Placement is Hard

- **Constraint**: Large fraction of device vendors offer poll-only APIs

- Naive periodic polling:
  - **Too fast** → overloads devices
  - **Too slow** → delayed detection

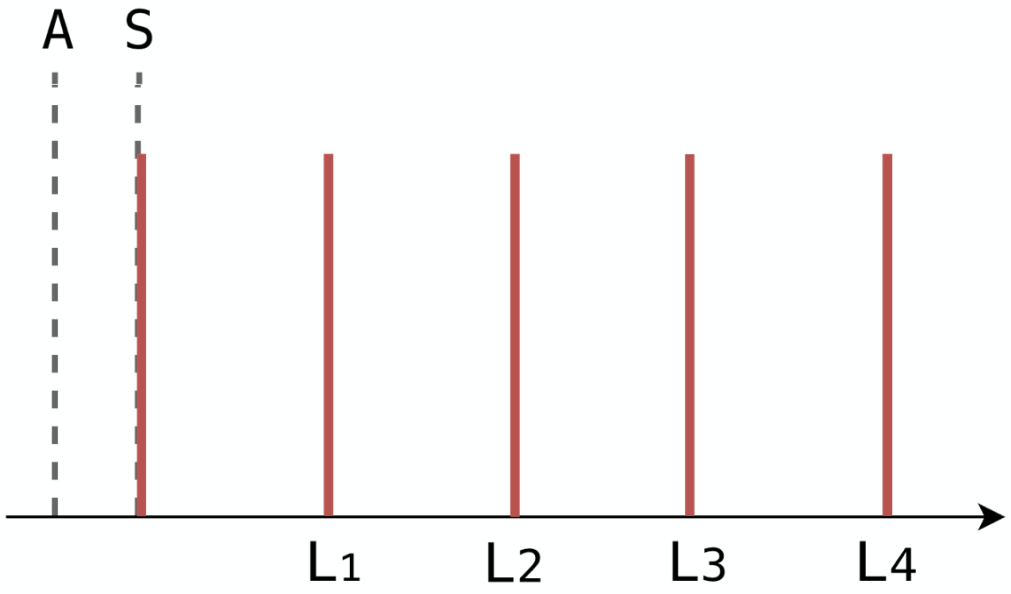


# Challenge: Polling Placement is Hard

- **Constraint**: Large fraction of device vendors offer poll-only APIs

- Naive periodic polling:
  - **Too fast** → overloads devices
  - **Too slow** → delayed detection

- We need to place **few polls** intelligently



# Idea: Adaptive Polling

# Idea: Adaptive Polling

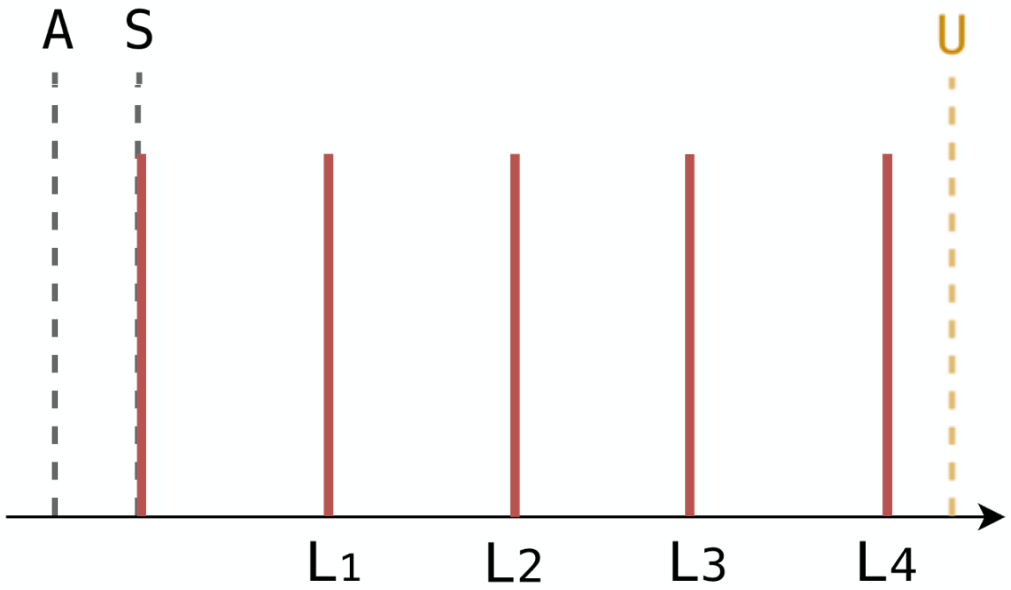
- Given:

# Idea: Adaptive Polling

- Given:
  - upper bound $U$

# Idea: Adaptive Polling

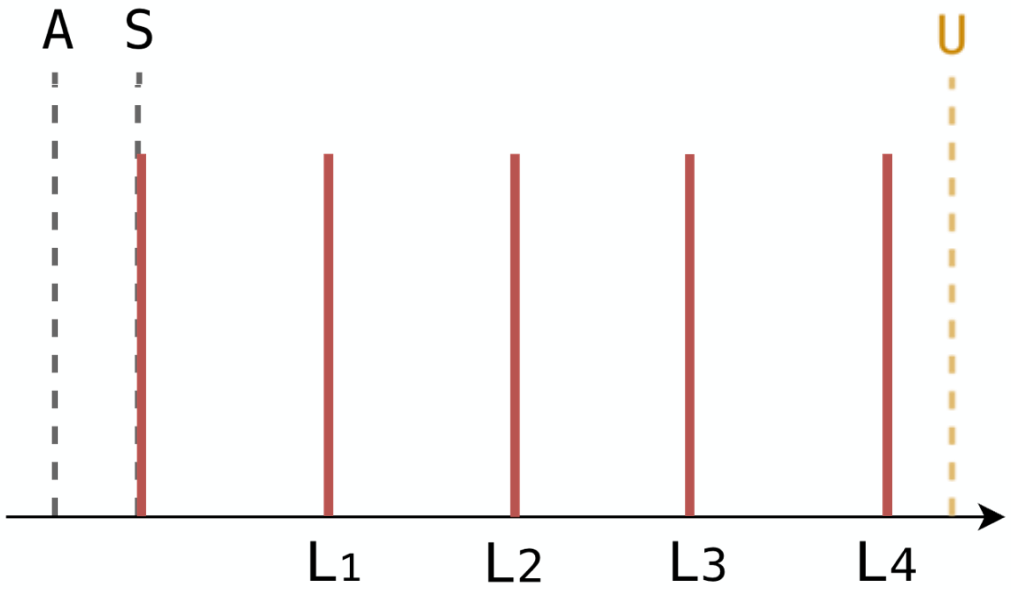
- Given:
  - upper bound $U$
  - poll budget



# Idea: Adaptive Polling

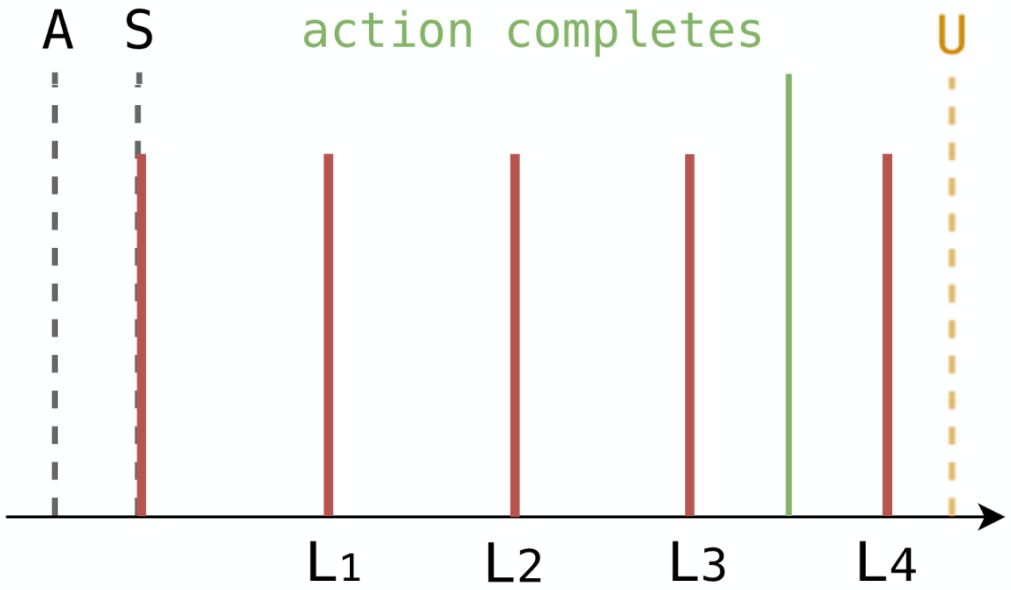
- Given:
  - upper bound $U$
  - poll budget

- Goal: Minimize detection time $Q$



# Idea: Adaptive Polling

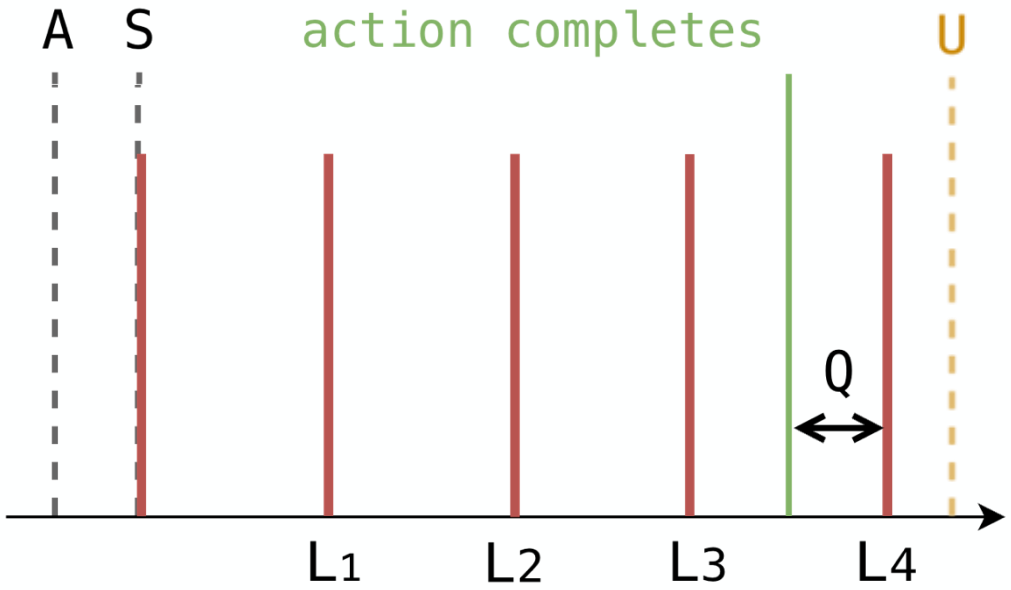
- Given:
  - upper bound $U$
  - poll budget
- Goal: Minimize detection time $Q$



# Idea: Adaptive Polling

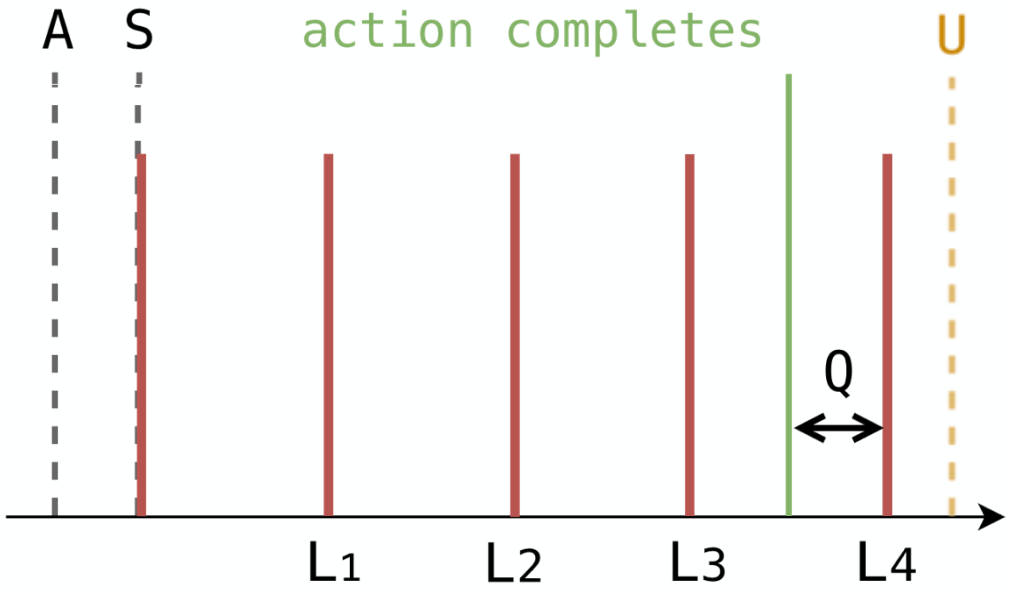
- Given:
  - upper bound $U$
  - poll budget

- Goal: Minimize detection time $Q$



# Idea: Adaptive Polling

- Given:
  - upper bound $U$
  - poll budget
- Goal: Minimize detection time $Q$
- Opportunity to leverage **historical data**

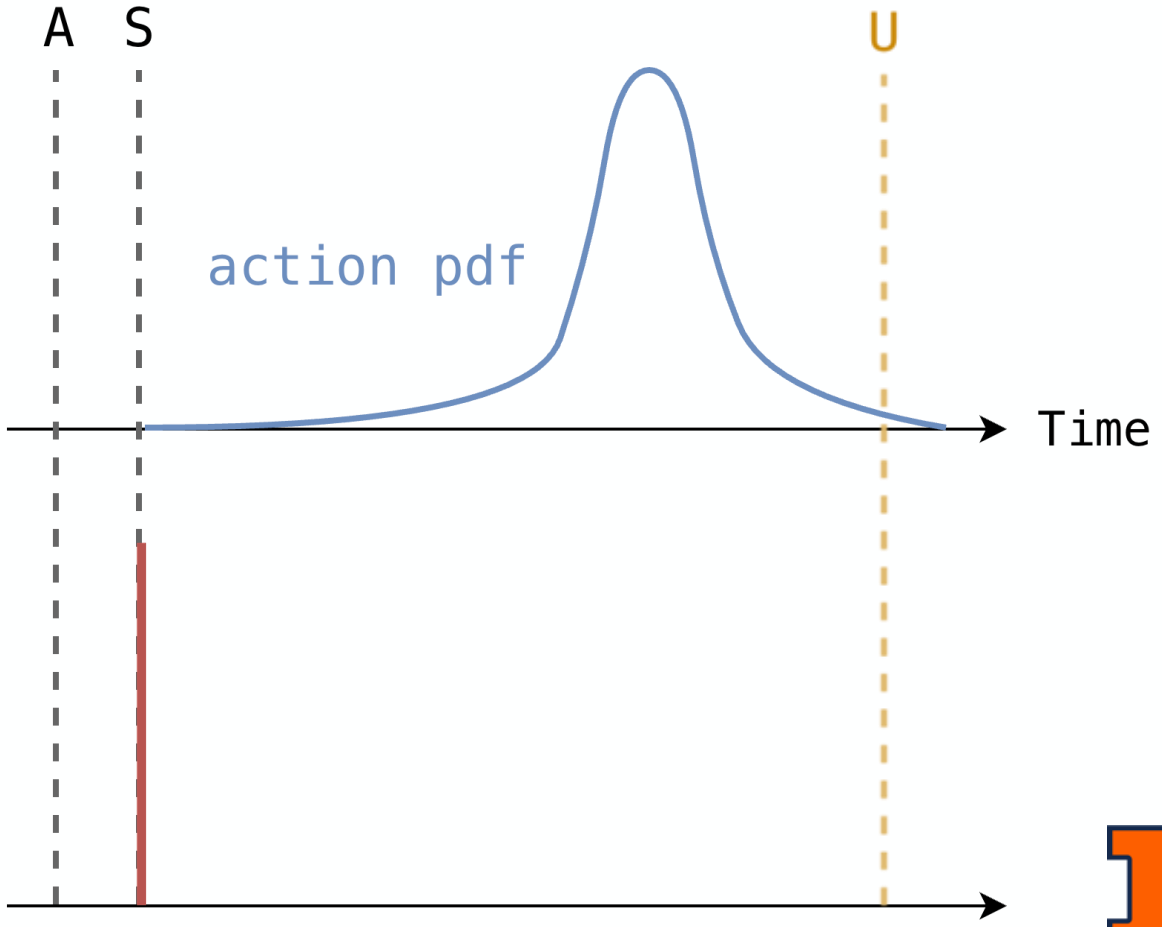


# Rascal: Adaptive Polling Strategy

Abstraction **RASC** → System **Rascal**

- Minimize Expected Detection Time
- $Q = \int_0^{L_1} (L_1 - t)p(t)dt + \int_{L_1}^{L_2} (L_2 - t)p(t)dt$

$$+ \cdots + \int_{L_{k-1}}^{L_k} (L_k - t)p(t)dt$$



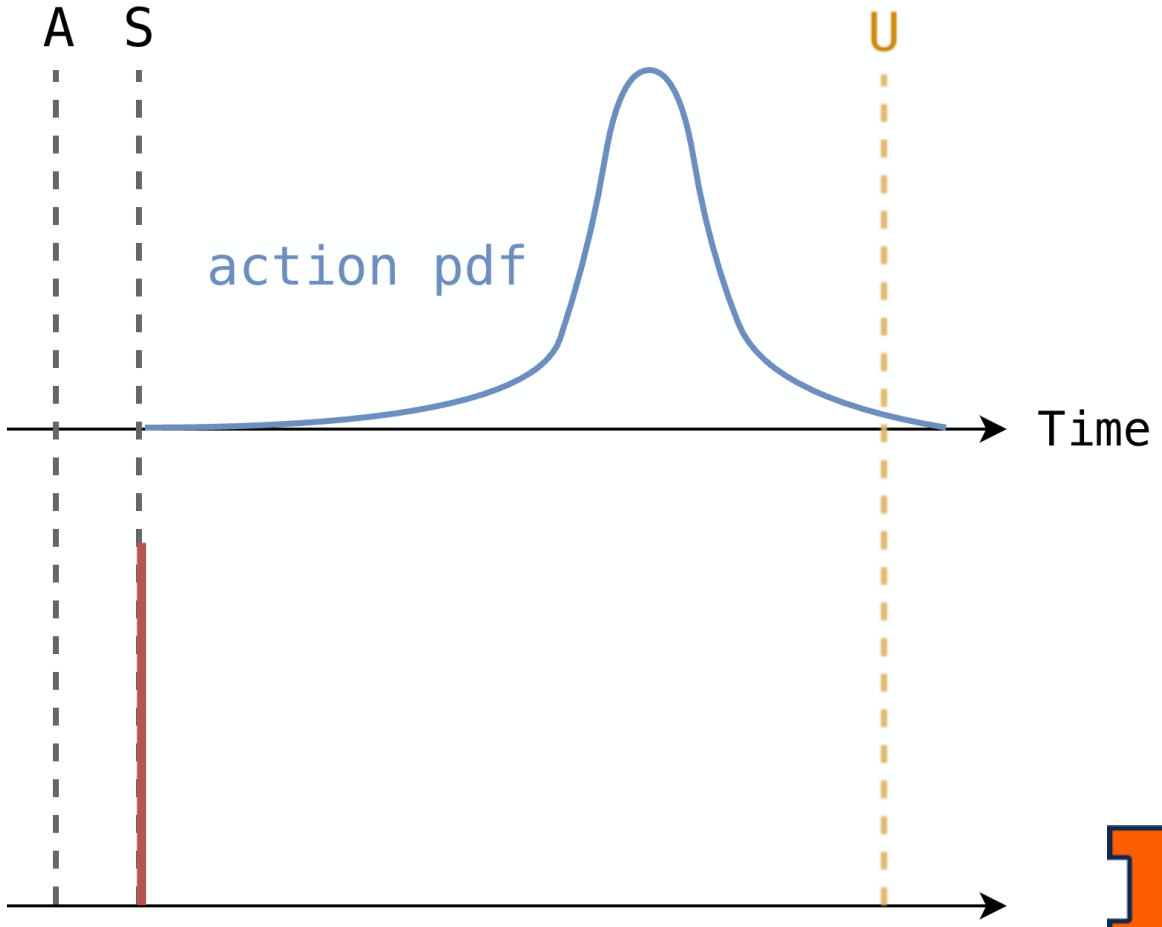
# Rascal: Adaptive Polling Strategy

Abstraction **RASC** → System **Rascal**

- Minimize Expected Detection Time
- $Q = \int_0^{L_1} (L_1 - t)p(t)dt + \int_{L_1}^{L_2} (L_2 - t)p(t)dt$

  first poll

  $+ \cdots + \int_{L_{k-1}}^{L_k} (L_k - t)p(t)dt$

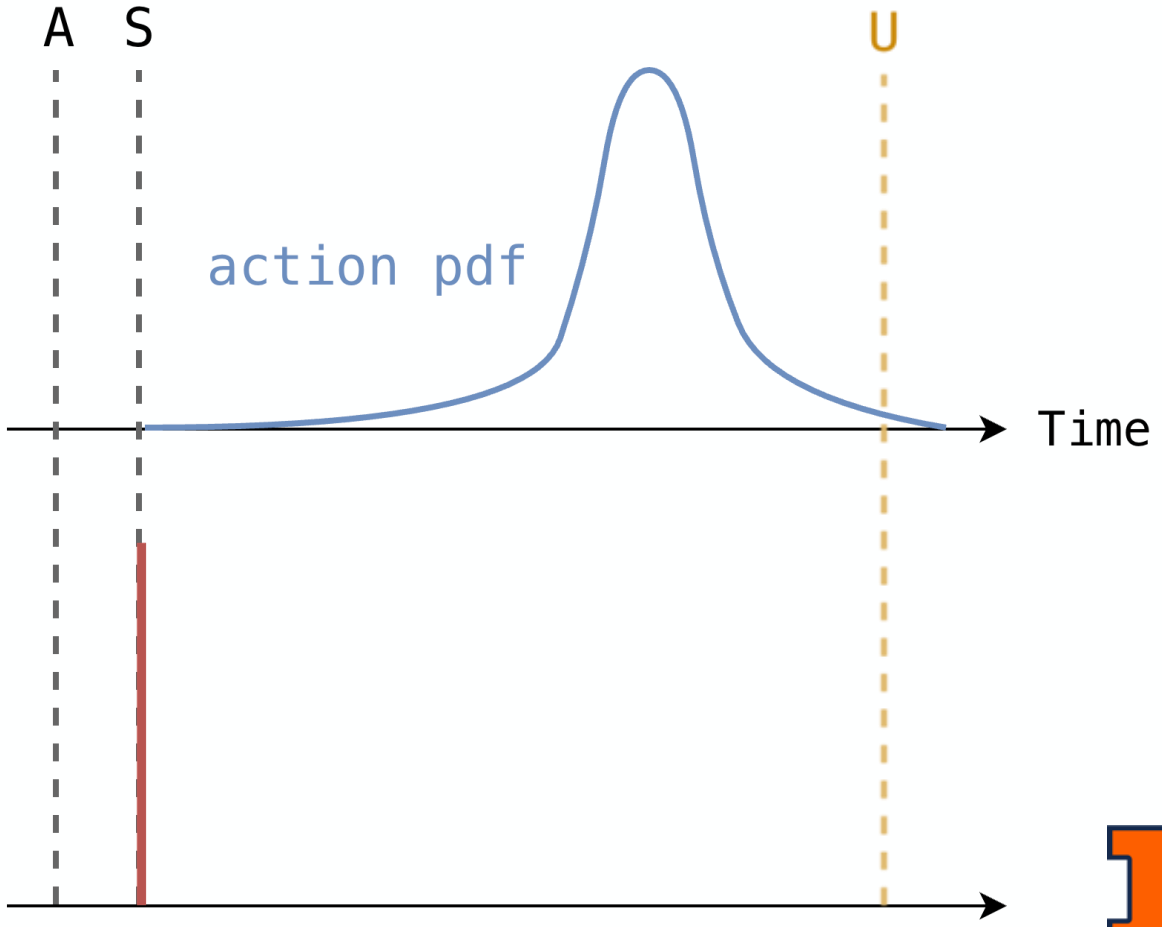


# Rascal: Adaptive Polling Strategy

Abstraction **RASC** → System **Rascal**

- Minimize Expected Detection Time
- $Q = \int_0^{L_1} (L_1 - t)p(t)dt + \int_{L_1}^{L_2} (L_2 - t)p(t)dt + \cdots + \int_{L_{k-1}}^{L_k} (L_k - t)p(t)dt$

first poll Start/completion time



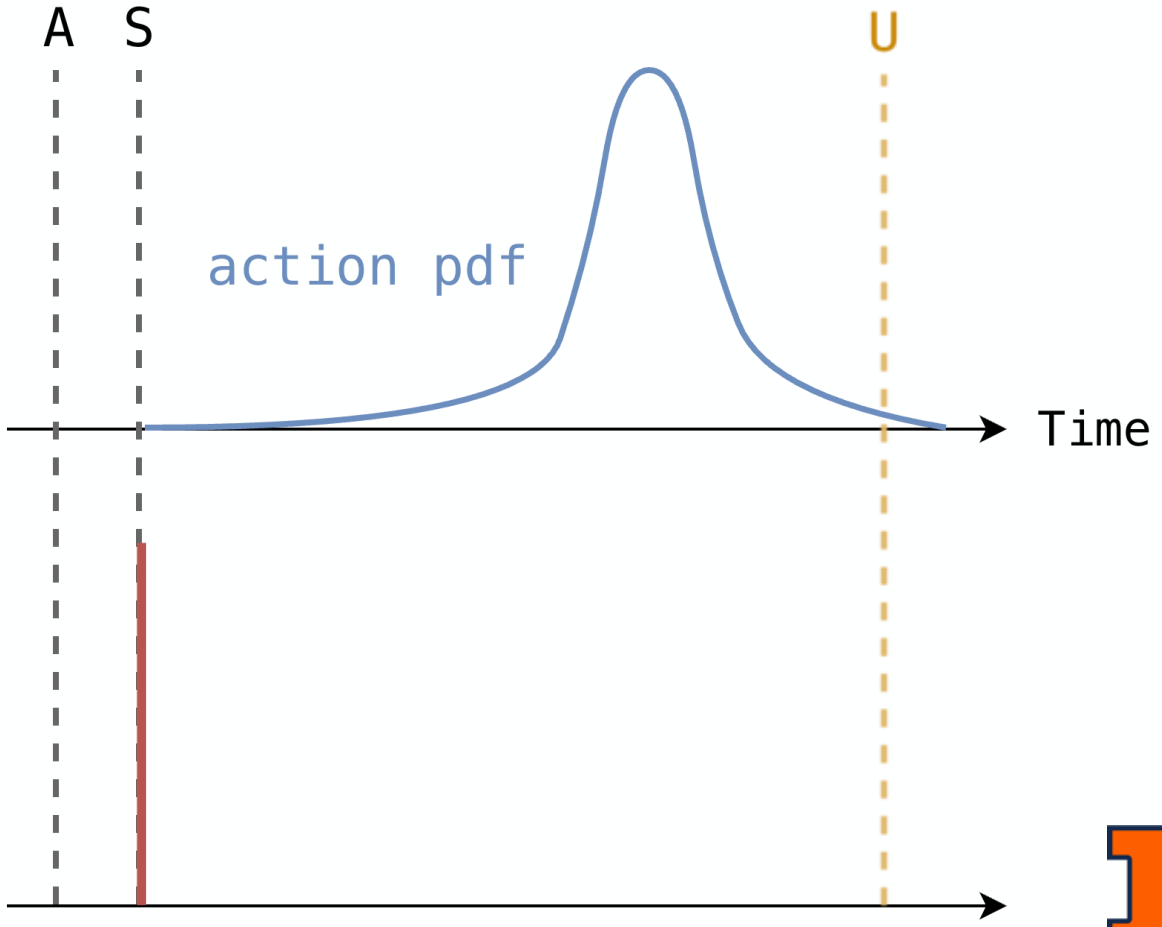
# Rascal: Adaptive Polling Strategy

Abstraction **RASC** → System **Rascal**

- Minimize Expected Detection Time
- $Q = \int_0^{L_1} (L_1 - t)p(t)dt + \int_{L_1}^{L_2} (L_2 - t)p(t)dt + \cdots + \int_{L_{k-1}}^{L_k} (L_k - t)p(t)dt$

first poll

Start/completion time



# Rascal: Adaptive Polling Strategy

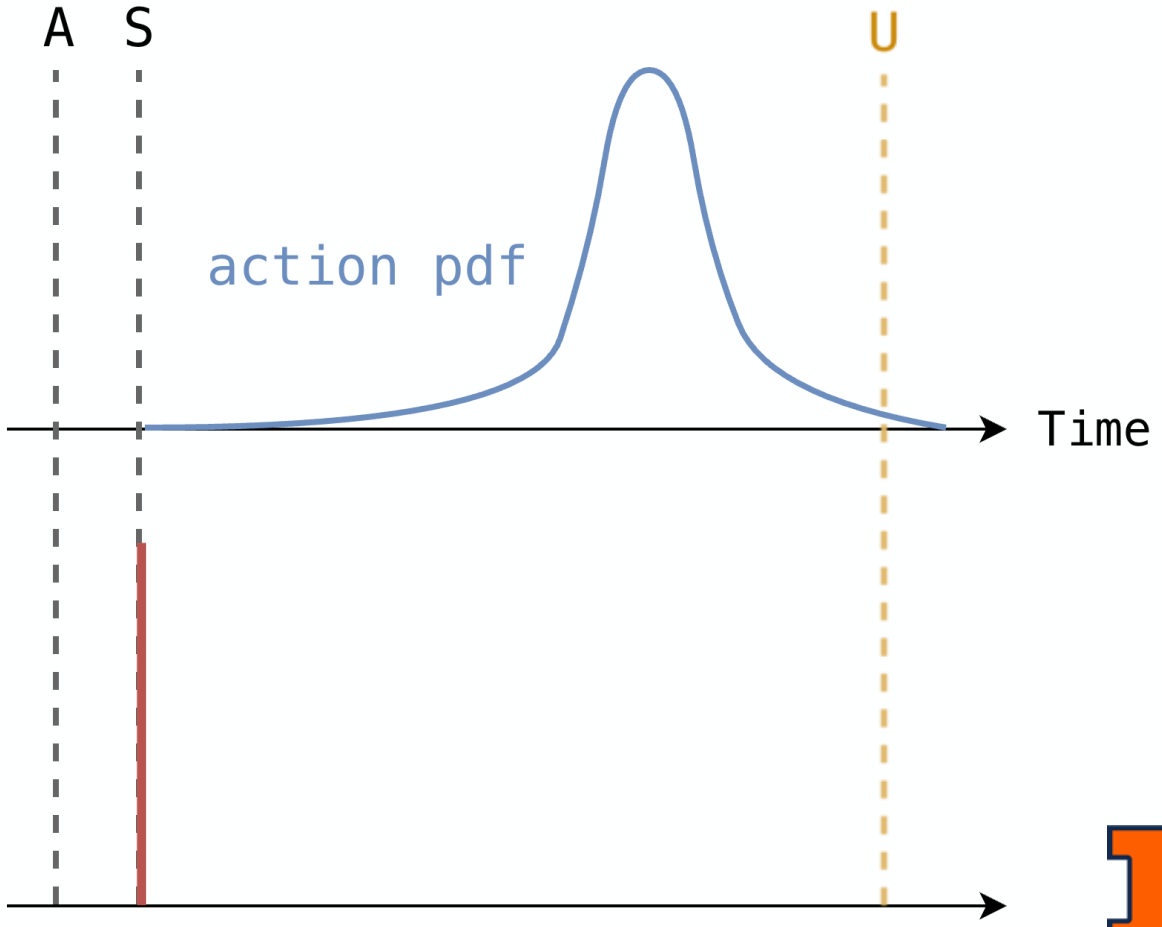
Abstraction **RASC** → System **Rascal**

- Minimize Expected Detection Time
- $Q = \int_0^{L_1} (L_1 - t)p(t)dt + \int_{L_1}^{L_2} (L_2 - t)p(t)dt + \cdots + \int_{L_{k-1}}^{L_k} (L_k - t)p(t)dt$

first poll

Start/completion time

Start/completion distribution



# Rascal: Adaptive Polling Strategy

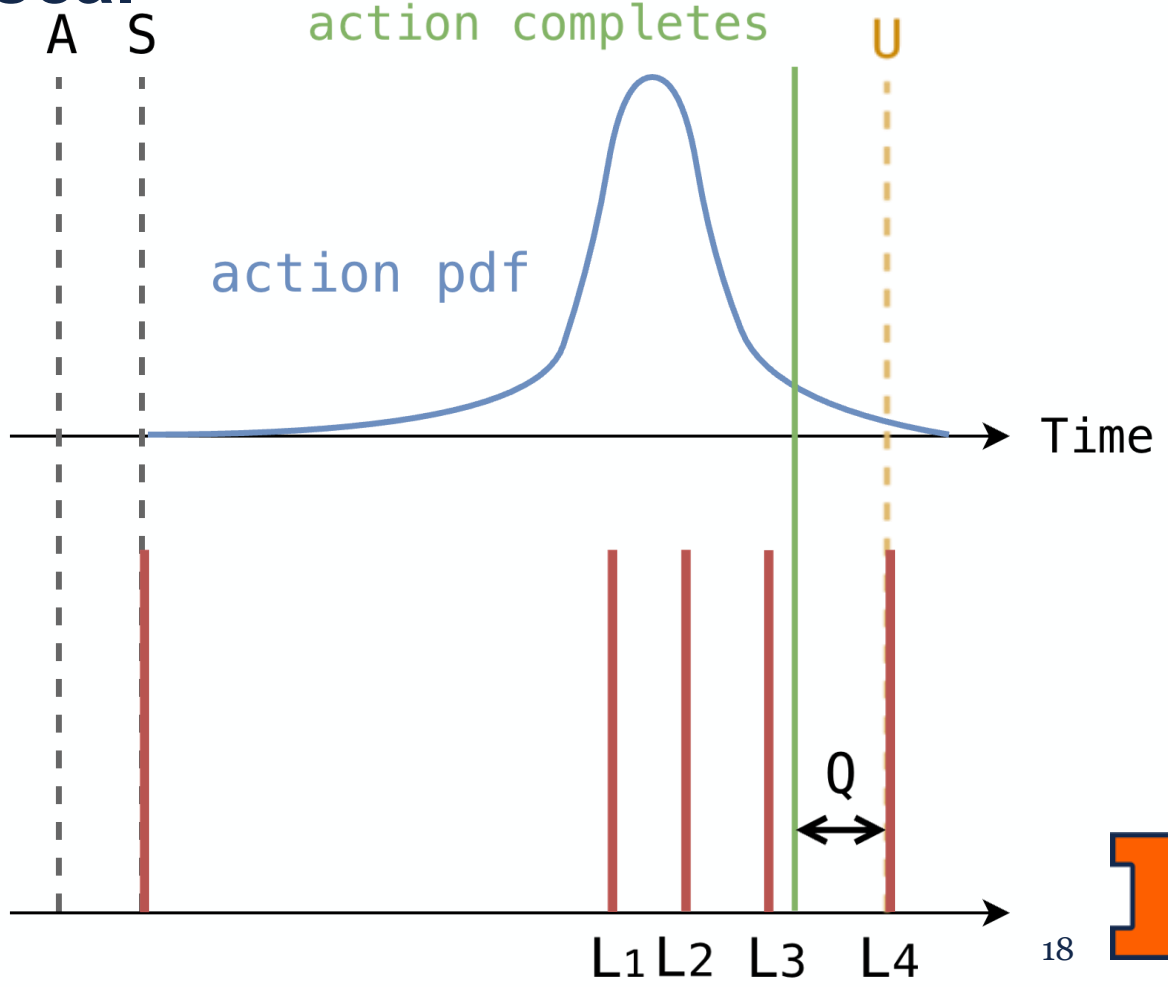
Abstraction **RASC** → System **Rascal**

- Minimize Expected Detection Time
- $Q = \int_0^{L_1} (L_1 - t)p(t)dt + \int_{L_1}^{L_2} (L_2 - t)p(t)dt + \cdots + \int_{L_{k-1}}^{L_k} (L_k - t)p(t)dt$

first poll

Start/completion time

Start/completion distribution



# Detection beyond upper bound $U$

# Detection beyond upper bound $U$

- Partial progress—next poll: $\min\{estimate, Q_w\}$

# Detection beyond upper bound $U$

- Partial progress—next poll: $\min\{estimate, Q_w\}$

**Expected time $t = 5s$** for **distance $d = 5m$**

# Detection beyond upper bound $U$

- Partial progress—next poll: $\min\{estimate, Q_w\}$

**Expected time $t = 5s$** for **distance $d = 5m$**

**Observed time $t_1 = 6s \geq U$** for **distance $d_1 = 4m < d$**
**Observed rate $r$** for **distance $d_1$**

# Detection beyond upper bound $U$

- Partial progress—next poll: $\min\{estimate, Q_w\}$



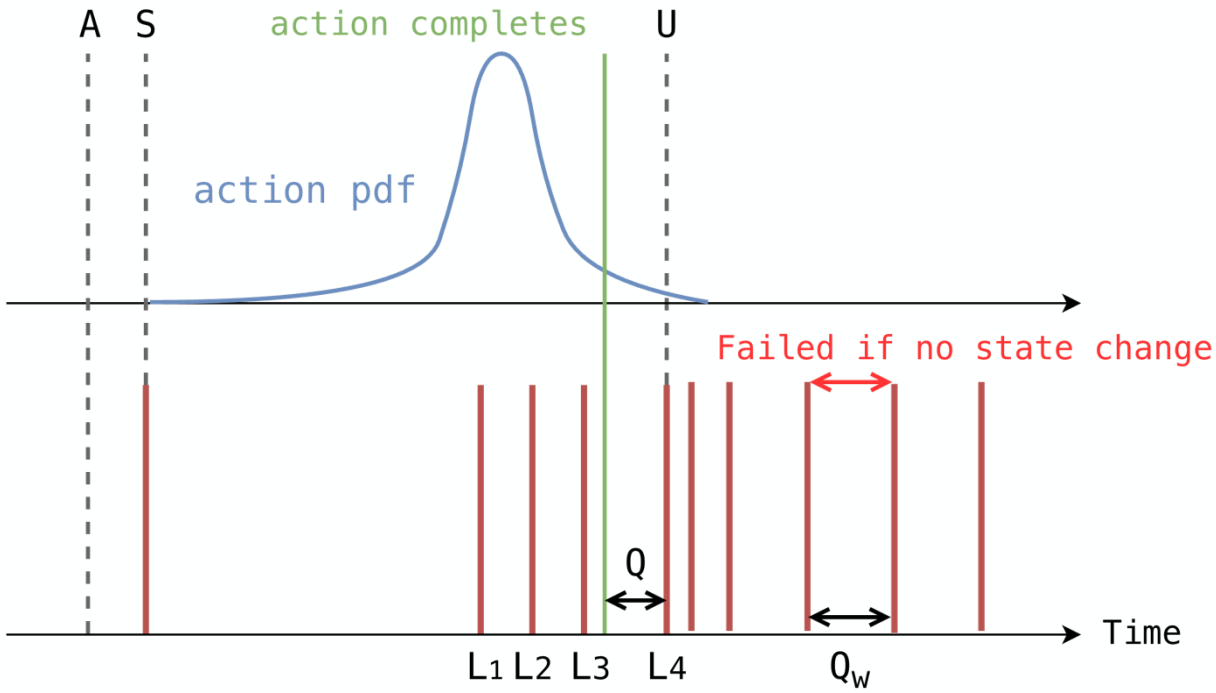
**Expected time $t = 5s$** for **distance $d = 5m$**

**Observed time $t_1 = 6s \geq U$** for **distance $d_1 = 4m < d$**
**Observed rate $r$** for **distance $d_1$**

**New estimated time $t_2 = 7.5s > t_1$** for **original distance $d$**

# Detection beyond upper bound $U$

- Partial progress—next poll: $\min\{estimate, Q_w\}$
- No progress: exponential backoff up to $Q_w$



# Outline

- Introduction & Motivation
- RASC Abstraction
- Rascal Programmability: Building Atop RASC
- Rascal Observability: Implementing RASC
- Evaluation

# Trace-driven Eval w/ HomeAssistant

# Trace-driven Eval w/ HomeAssistant

- Adaptive Polling
  - Manually measured action lengths of doors, shades, elevators
  - Used existing thermostat trace dataset

[1] Fraunhofer Center for Sustainable Energy Systems. Multifamily programmable thermostat data. https://catalog.data.gov/dataset/multifamily-programmable-thermostat-data, 2022.

# Trace-driven Eval w/ HomeAssistant

- Adaptive Polling
  - Manually measured action lengths of doors, shades, elevators
  - Used existing thermostat trace dataset

- Adapted routine traces from IoTBench, SmartThings repos

[1] Fraunhofer Center for Sustainable Energy Systems. Multifamily programmable thermostat data. https://catalog.data.gov/dataset/multifamily-programmable-thermostat-data, 2022.
[2] IoTBench test-suite. https://github.com/IoTBench/IoTBench-test-suite/tree/master/openHAB.
[3] SmartThings open-source DeviceTypeHandlers and SmartApps code. https://github.com/SmartThingsCommunity/SmartThingsPublic/tree/master/smartapps.

# Trace-driven Eval w/ HomeAssistant

- Adaptive Polling
  - Manually measured action lengths of doors, shades, elevators
  - Used existing thermostat trace dataset

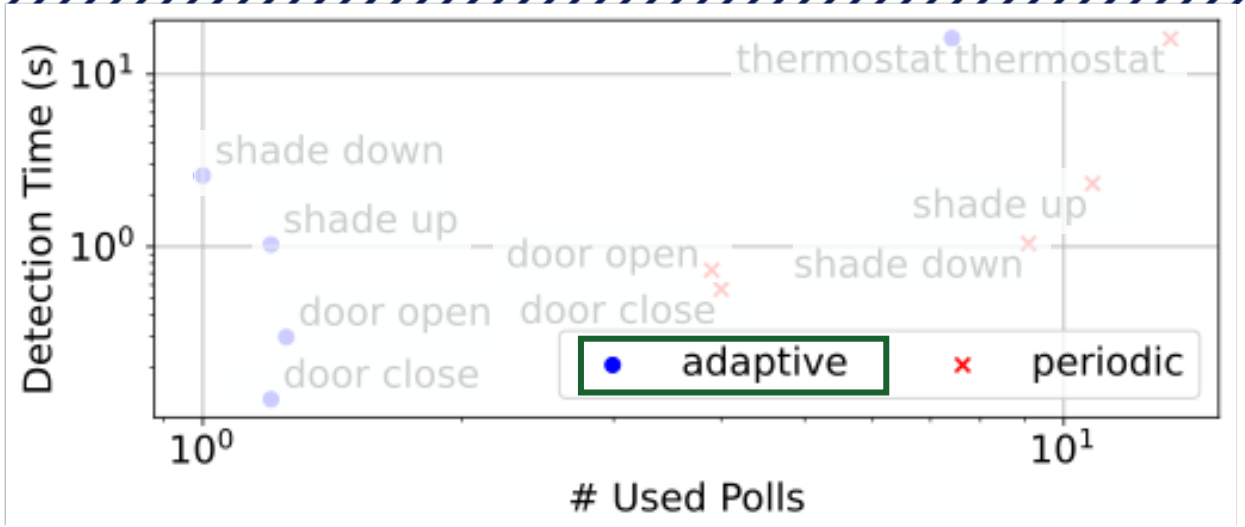
- Adapted routine traces from IoTBench, SmartThings repos
- Generated routine arrival workloads
  - random
  - random + bursty

[1] Fraunhofer Center for Sustainable Energy Systems. Multifamily programmable thermostat data. https://catalog.data.gov/dataset/multifamily-programmable-thermostat-data, 2022.
[2] IoTBench test-suite. https://github.com/IoTBench/IoTBench-test-suite/tree/master/openHAB.
[3] SmartThings open-source DeviceTypeHandlers and SmartApps code. https://github.com/SmartThingsCommunity/SmartThingsPublic/tree/master/smartapps.

# How Efficient is Rascal's Polling?

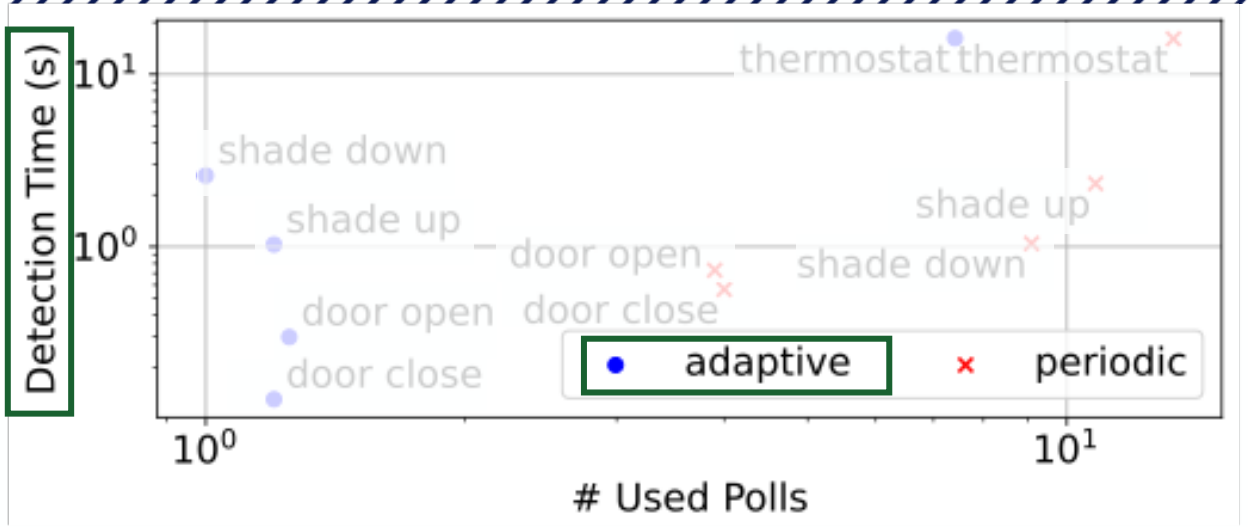
# How Efficient is Rascal's Polling?

- Rascal detects action state change with minimal polls within detection deadlines



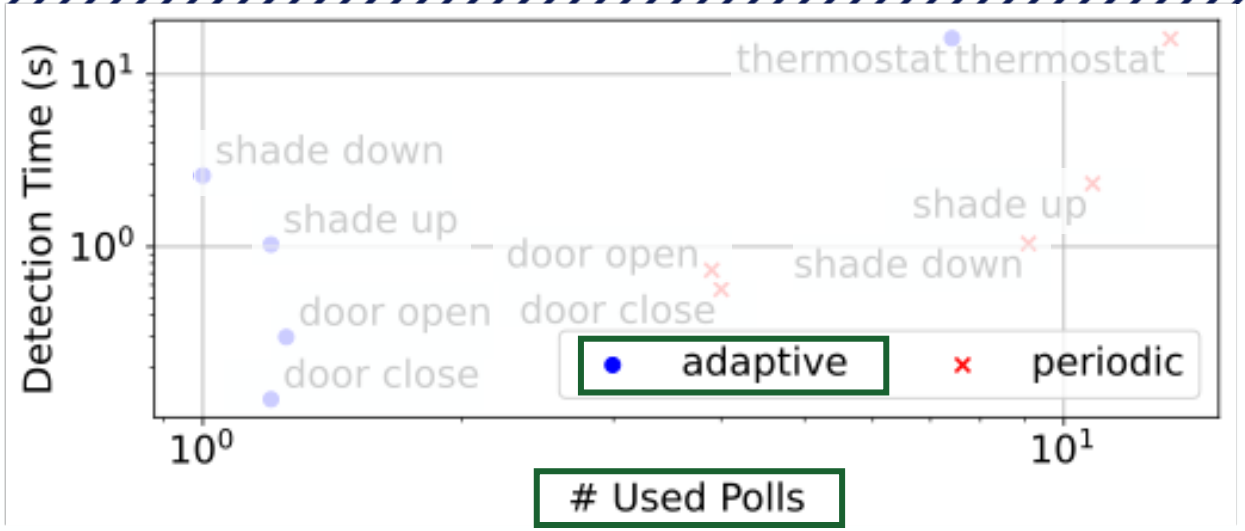
# How Efficient is Rascal's Polling?

- Rascal detects action state change with minimal polls within detection deadlines



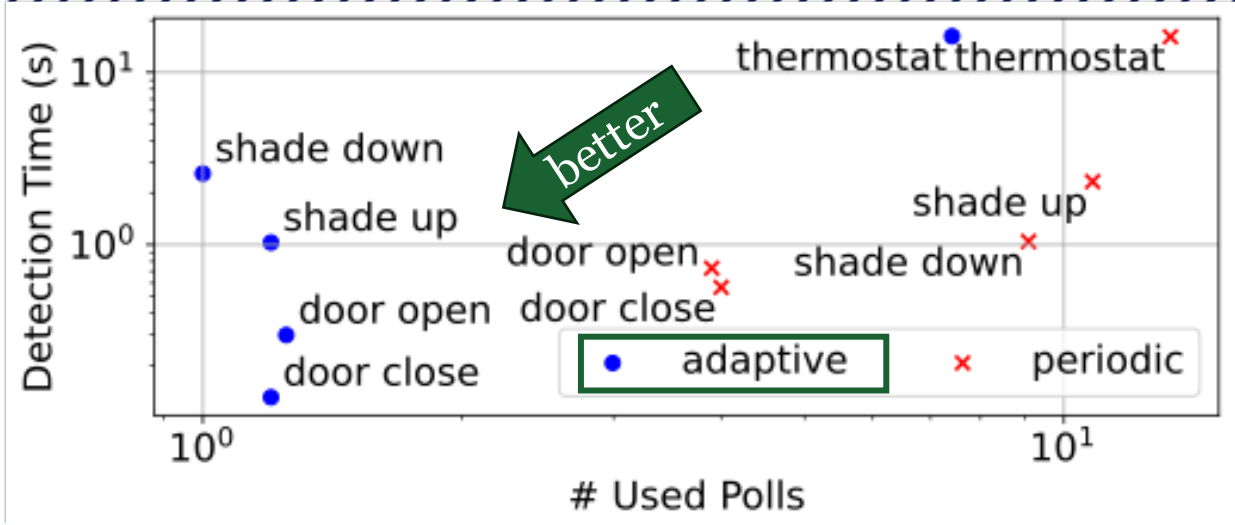
# How Efficient is Rascal's Polling?

- Rascal detects action state change with minimal polls within detection deadlines



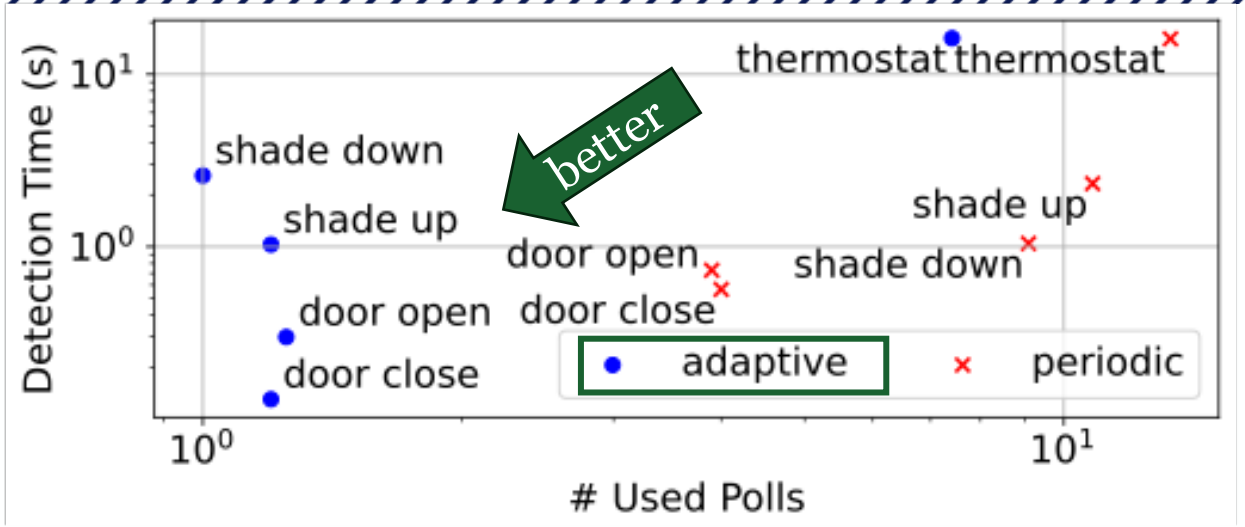
# How Efficient is Rascal's Polling?

- Rascal detects action state change with minimal polls within detection deadlines



# How Efficient is Rascal's Polling?

- Rascal detects action state change with minimal polls within detection deadlines




| Dataset | Polling Strategy | CPU (%) | | | Memory (MB) | | |
|---|---|---|---|---|---|---|---|
| | | avg | q50 | q99 | avg | q50 | q99 |
| random | adaptive | 8.79 | 1.8 | 19.8 | 4.12 | 4.73 | 8.33 |
| | periodic | 9.6 | 8.6 | 24.06 | **3.36** | **3.33** | **6.22** |
| | none | **2.41** | **1.1** | **10.1** | 17.72 | 19.19 | 20.28 |
| random + bursty | adaptive | 3.55 | 1.75 | **9.68** | **4.1** | **4.36** | **5.9** |
| | periodic | 4.6 | 2.9 | 19.26 | 4.54 | 4.61 | 6.58 |
| | none | **2.72** | **1.3** | 22.39 | 17.87 | 18.21 | 19.07 |

- With low overheads for random and bursty routine arrival workloads

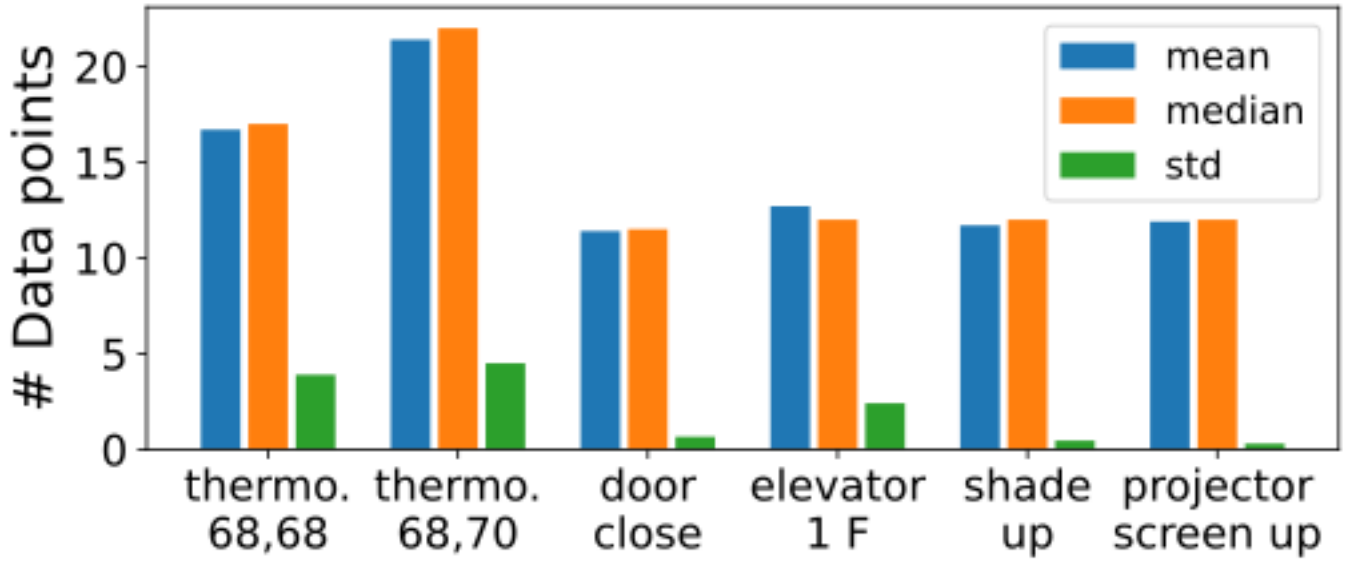
# Does Rascal's Estimation Converge?

# Does Rascal's Estimation Converge?

- Initially slow and inefficient

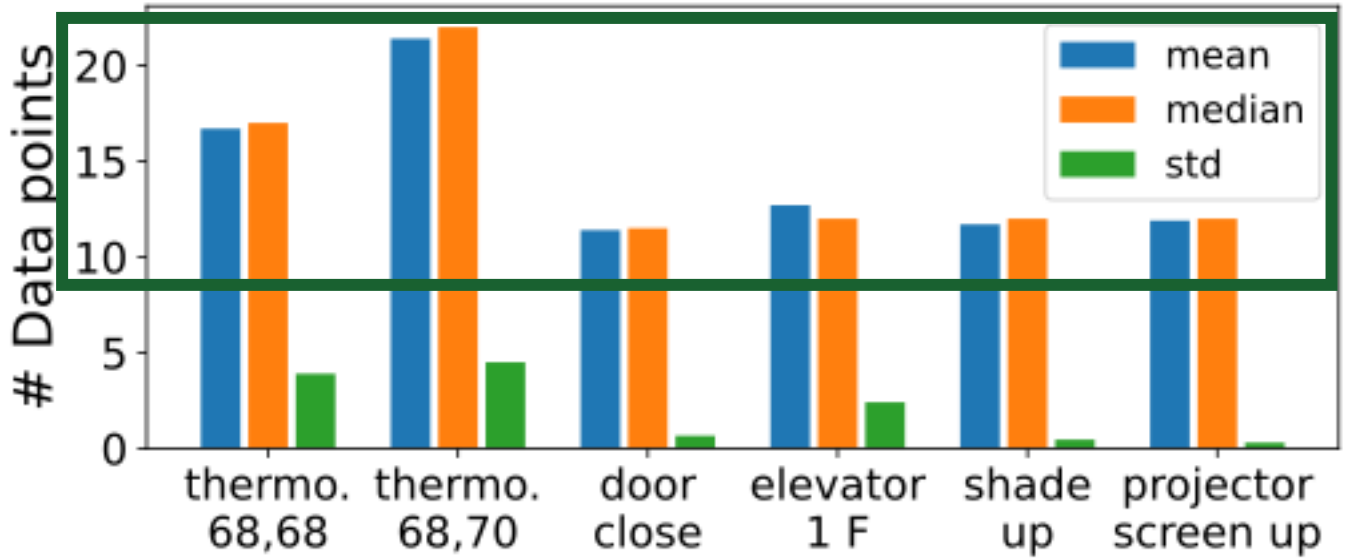
# Does Rascal's Estimation Converge?

- Initially slow and inefficient



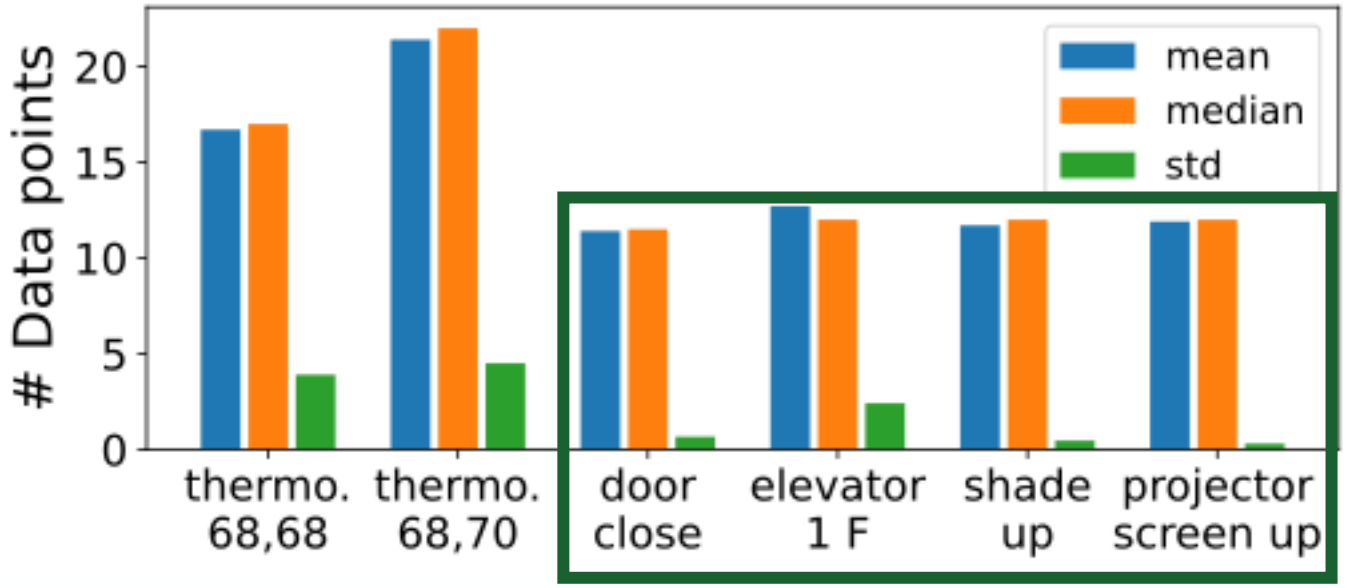
# Does Rascal's Estimation Converge?

- Initially slow and inefficient
- Rascal converges at 10-20 samples



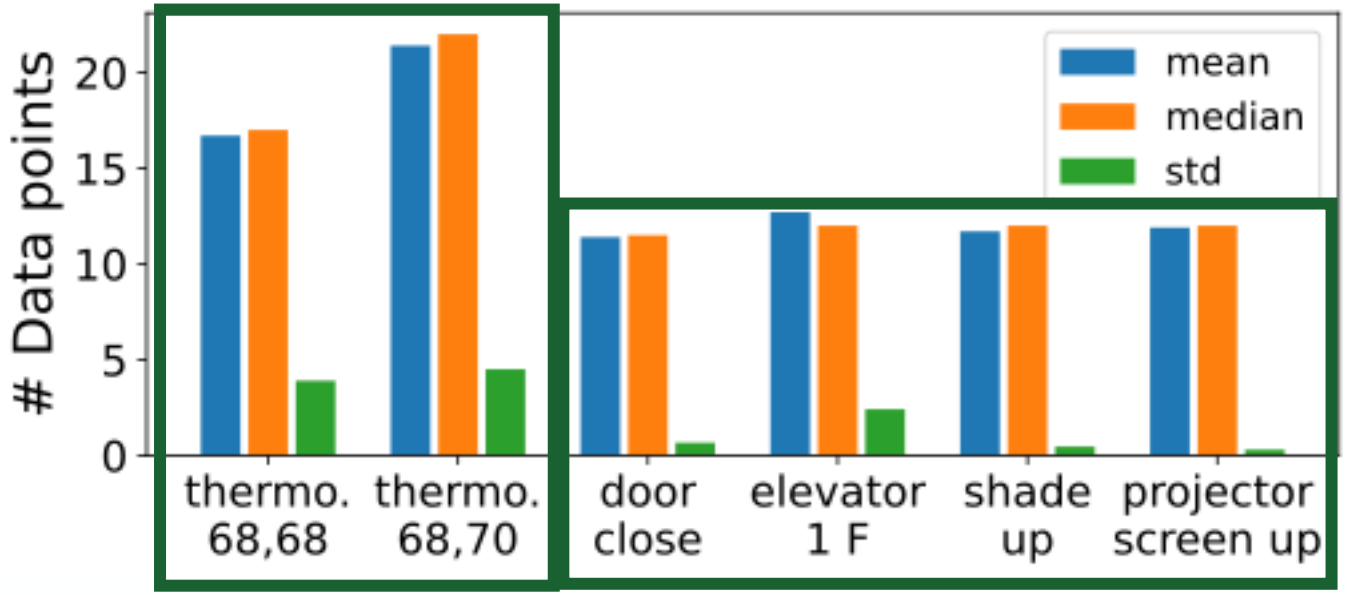
# Does Rascal's Estimation Converge?

- Initially slow and inefficient
- Rascal converges at 10-20 samples



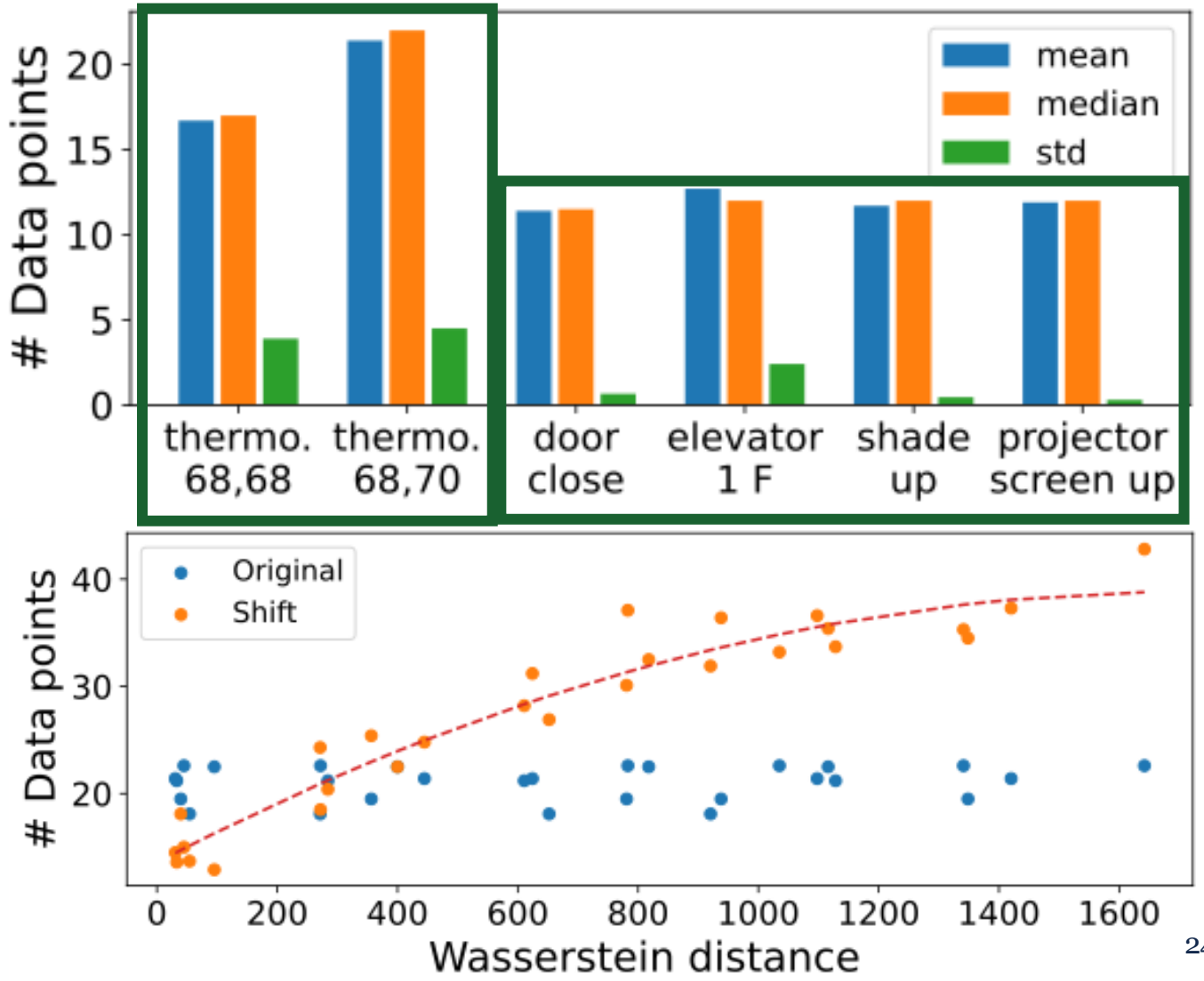
# Does Rascal's Estimation Converge?

- Initially slow and inefficient
- Rascal converges at 10-20 samples



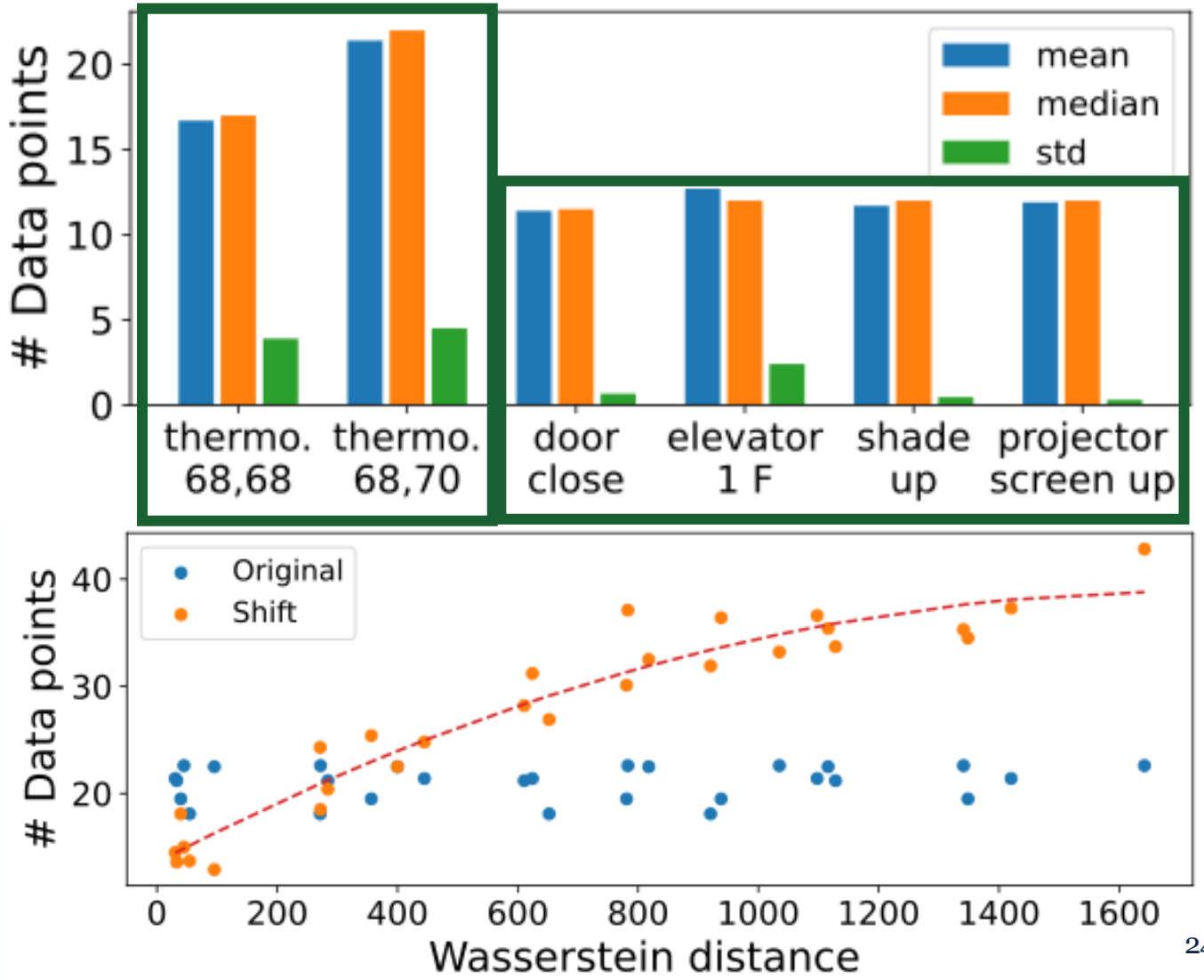
# Does Rascal's Estimation Converge?

- Initially slow and inefficient
- Rascal converges at 10-20 samples



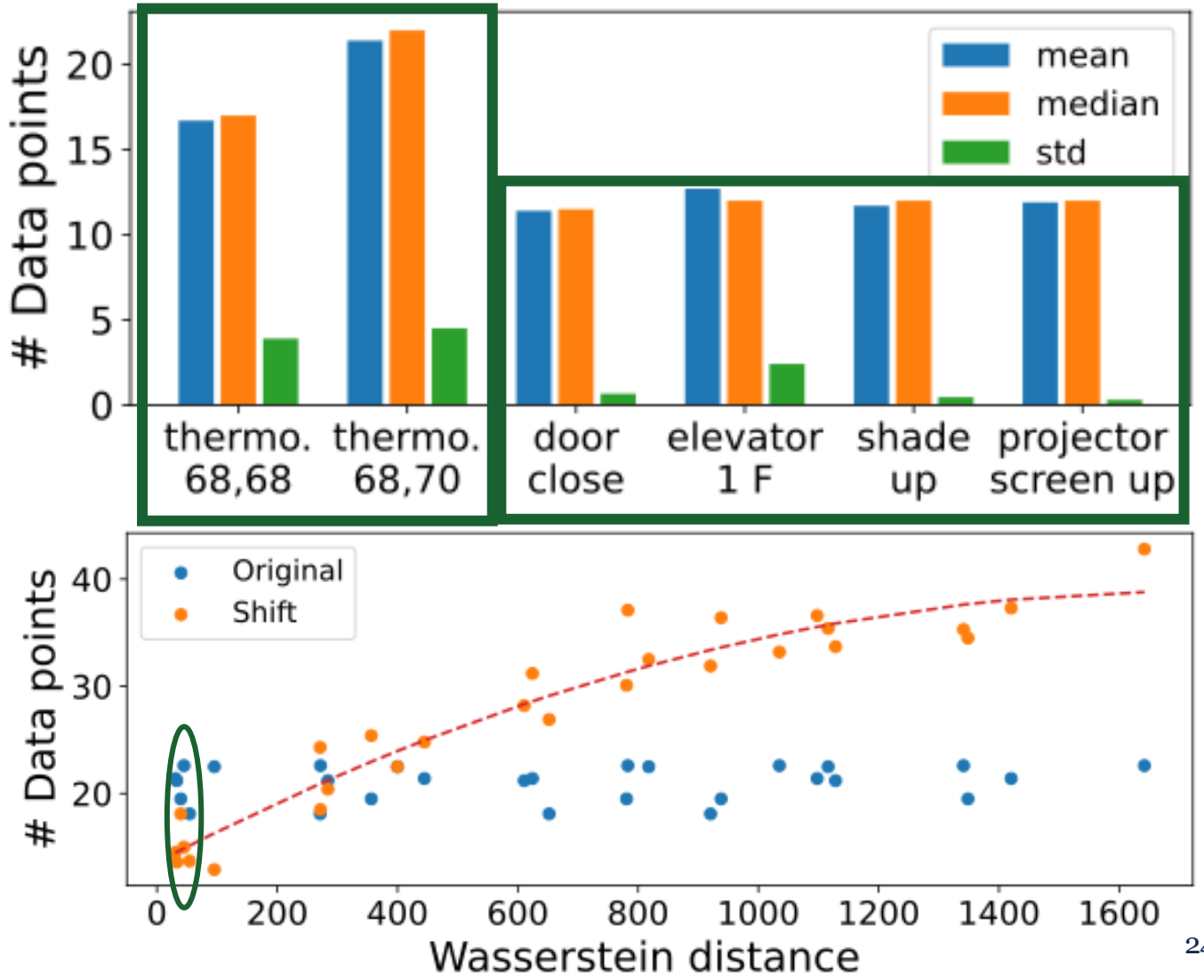
# Does Rascal's Estimation Converge?

- Initially slow and inefficient
- Rascal converges at 10-20 samples
- Rascal reconverges within 40 samples as the behavior drifts



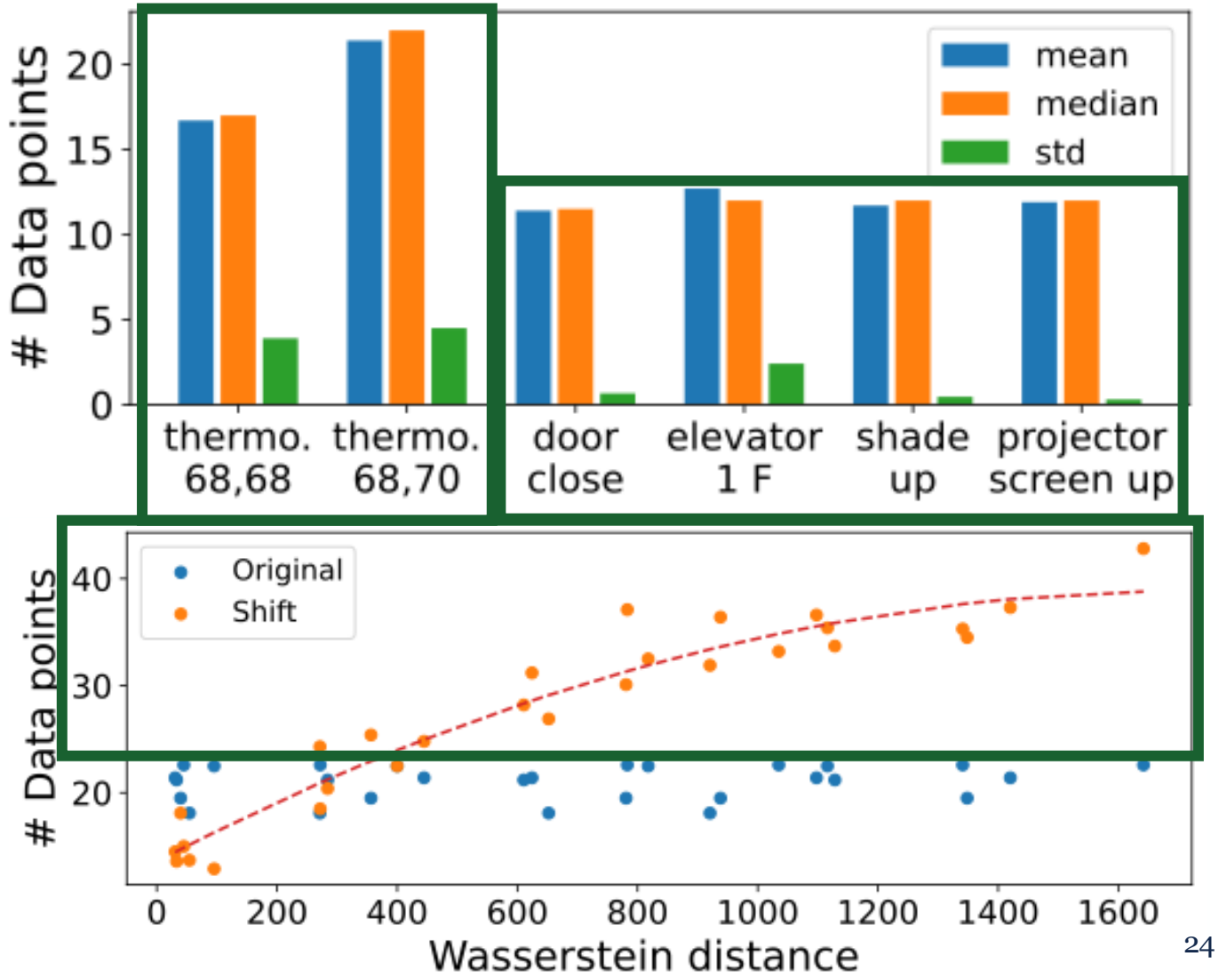
# Does Rascal's Estimation Converge?

- Initially slow and inefficient
- Rascal converges at 10-20 samples
- Rascal reconverges within 40 samples as the behavior drifts



# Does Rascal's Estimation Converge?

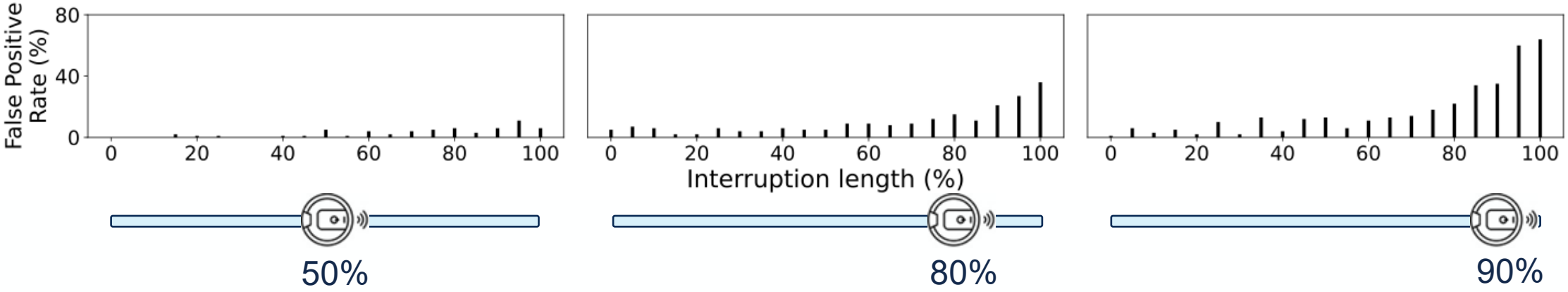
- Initially slow and inefficient
- Rascal converges at 10-20 samples
- Rascal reconverges within 40 samples as the behavior drifts



# Can Rascal Detect Interruptions?

# Can Rascal Detect Interruptions?

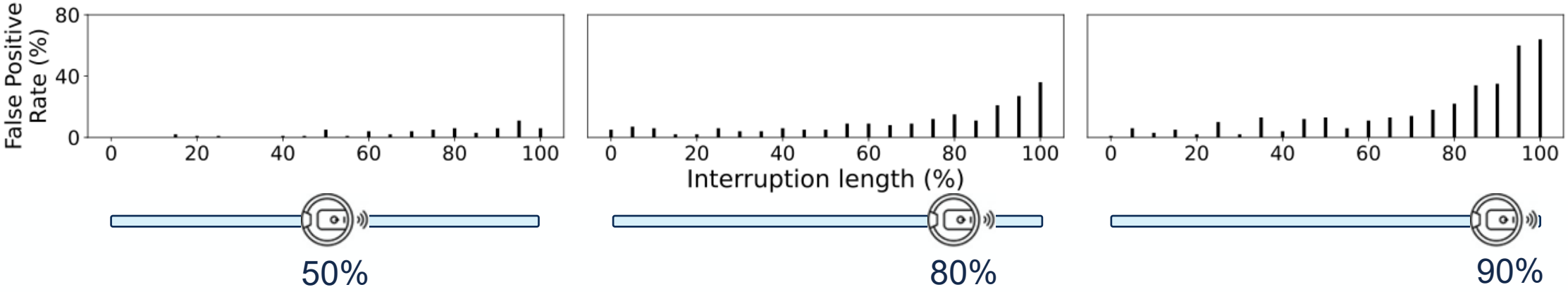
- Low false positives
  - Most interruptions ≠ failures
  - Errors mainly for **long, late interruptions**




Interruption start at the x% point of action execution

# Can Rascal Detect Interruptions?

- Low false positives
  - Most interruptions ≠ failures
  - Errors mainly for **long, late interruptions**

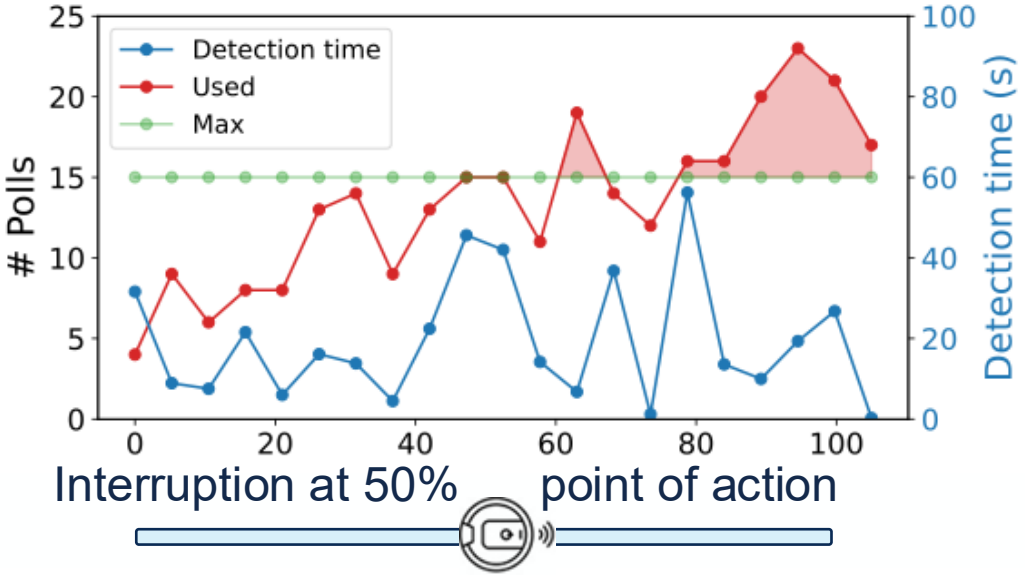


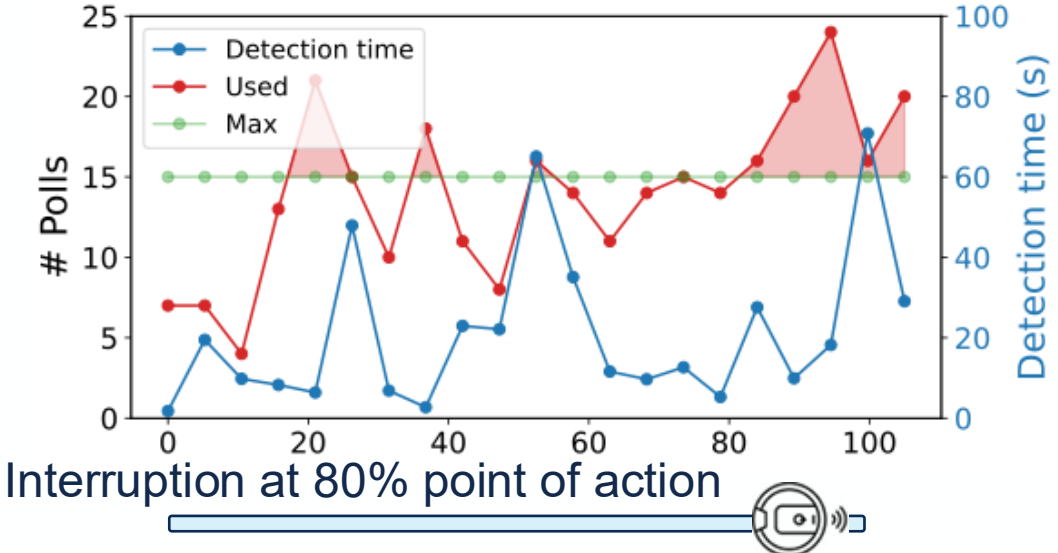

Interruption start at the x% point of action execution

More false positives

# Can Rascal Detect Interruptions?

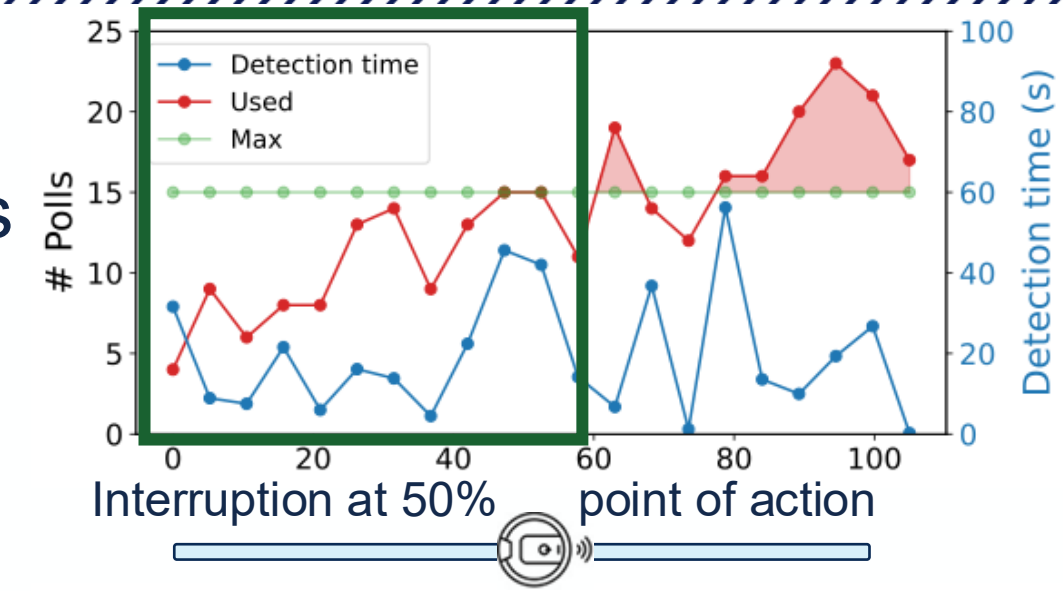
- Low false positives
  - Most interruptions ≠ failures
  - Errors mainly for **long, late interruptions**
- Polls extra only when needed
  - **Detection time low**

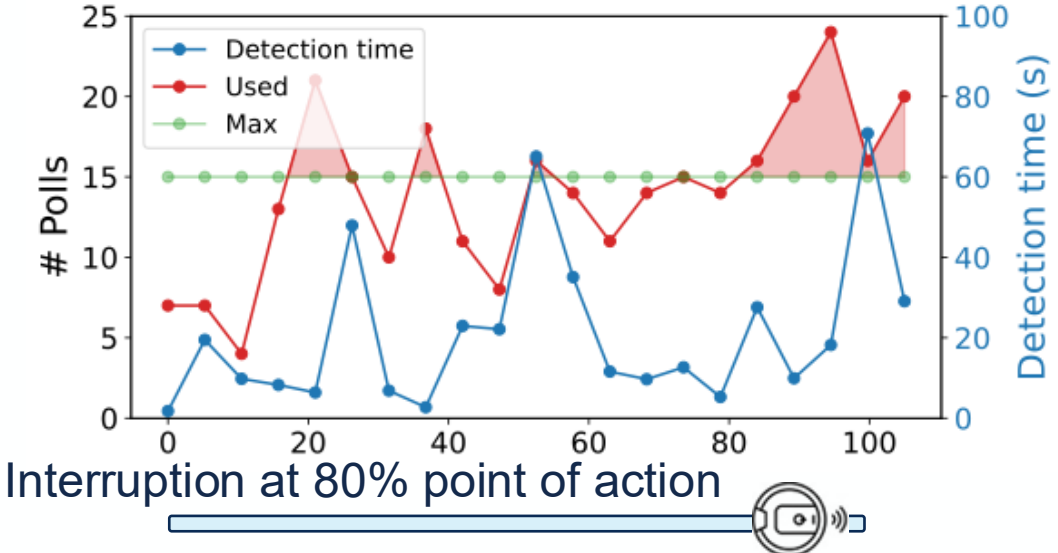


# Can Rascal Detect Interruptions?

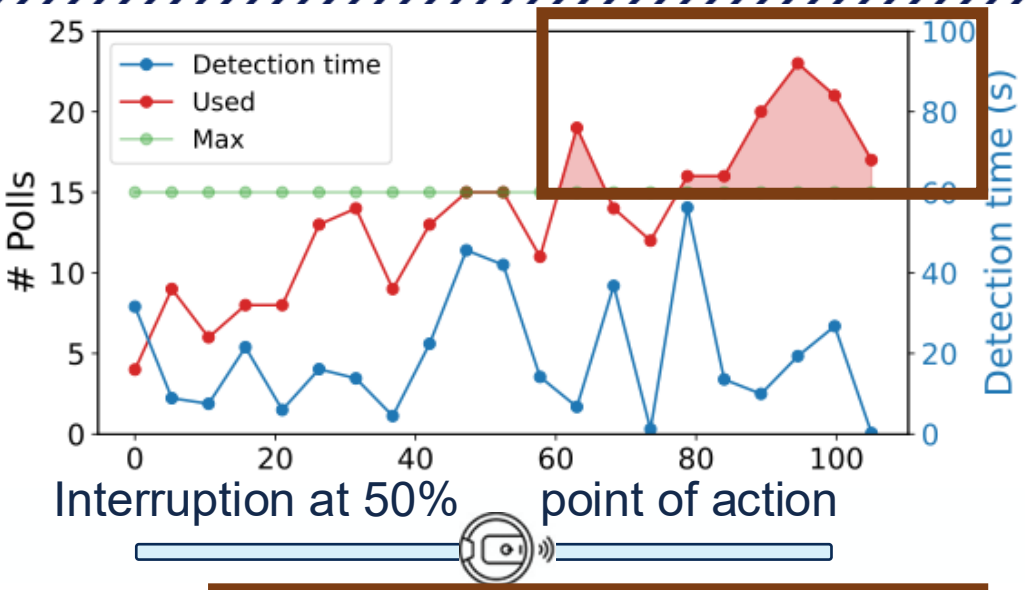
- Low false positives
  - Most interruptions ≠ failures
  - Errors mainly for **long, late interruptions**
- Polls extra only when needed
  - **Detection time low**

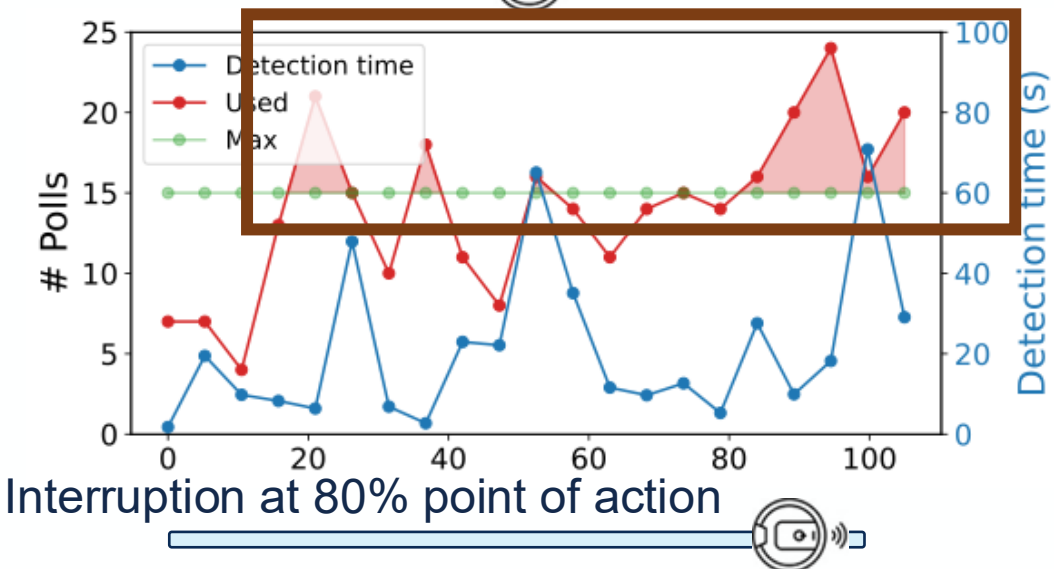


# Can Rascal Detect Interruptions?

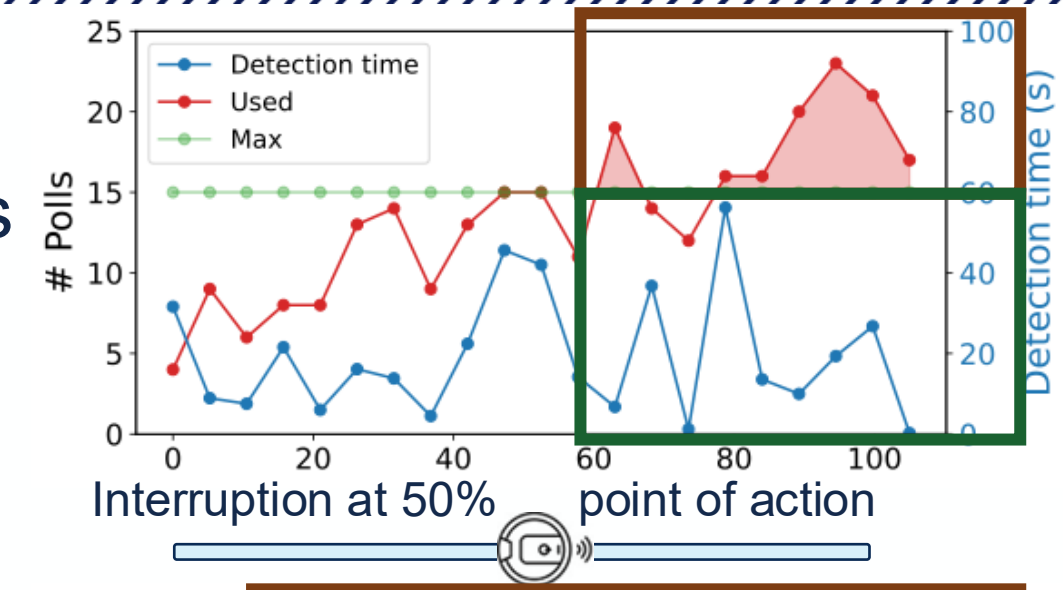
- Low false positives
  - Most interruptions ≠ failures
  - Errors mainly for **long, late interruptions**
- Polls extra only when needed
  - **Detection time low**

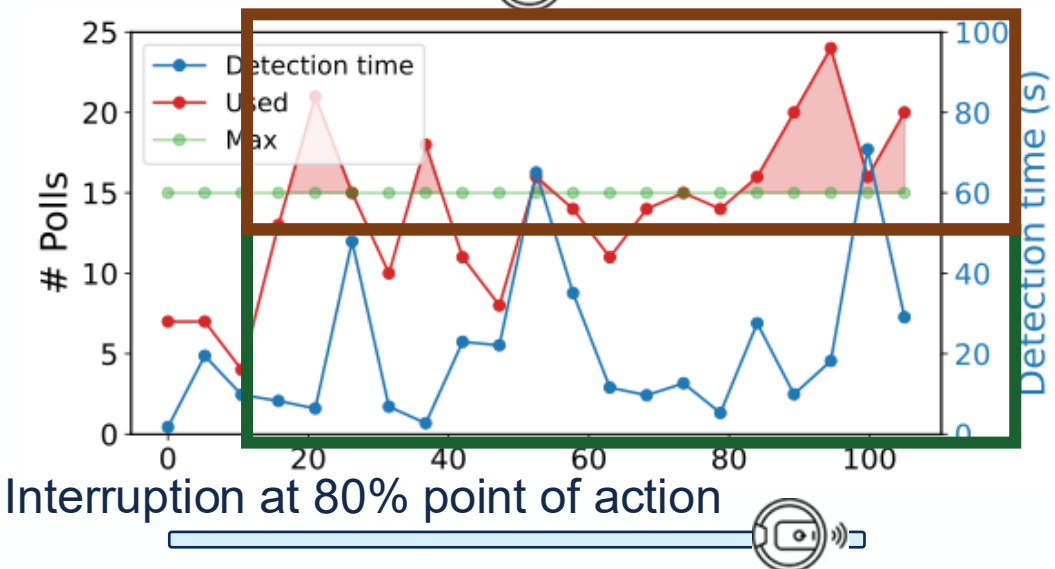


# Can Rascal Detect Interruptions?

- Low false positives
  - Most interruptions ≠ failures
  - Errors mainly for **long, late interruptions**
- Polls extra only when needed
  - **Detection time low**

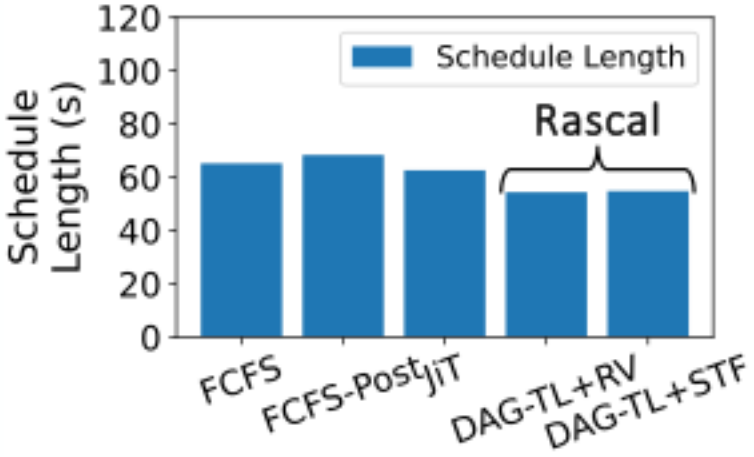


# How Efficient is Rascal's Scheduling?

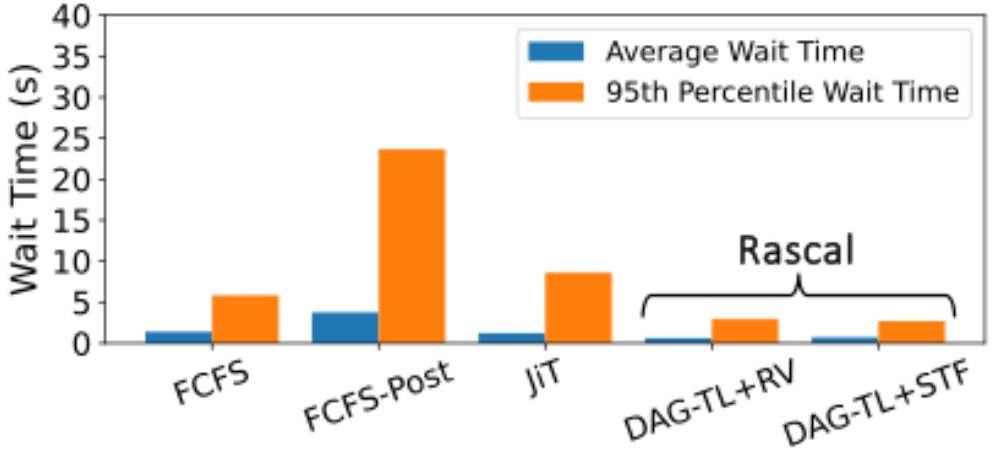
# How Efficient is Rascal's Scheduling?

- **Faster** routine **schedules**
  - Up to 11% vs best baseline



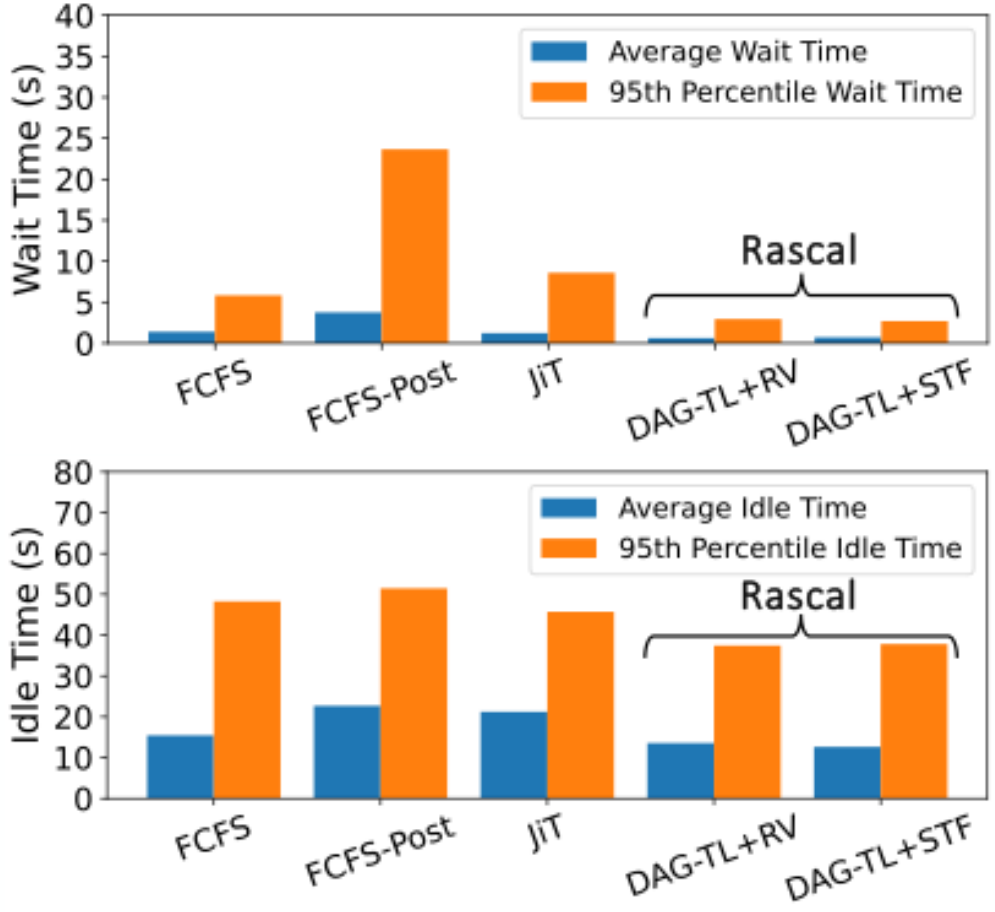
# How Efficient is Rascal's Scheduling?

- **Faster** routine **schedules**
  - Up to 11% vs best baseline
- **Lower wait & idle time**
  - Wait ↓ 33% (mean), 50% (p95)



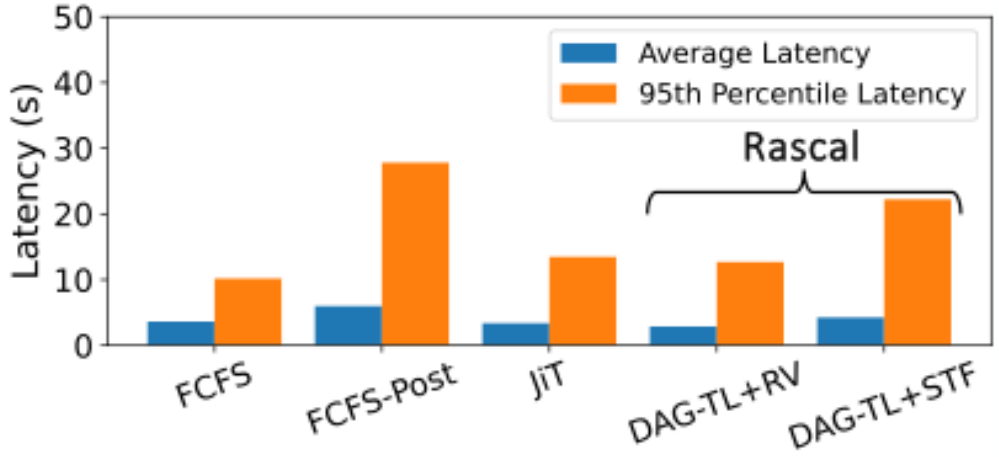
# How Efficient is Rascal's Scheduling?

- **Faster** routine **schedules**
  - Up to 11% vs best baseline
- **Lower wait & idle time**
  - Wait ↓ 33% (mean), 50% (p95)
  - Idle time ↓ 18–27%



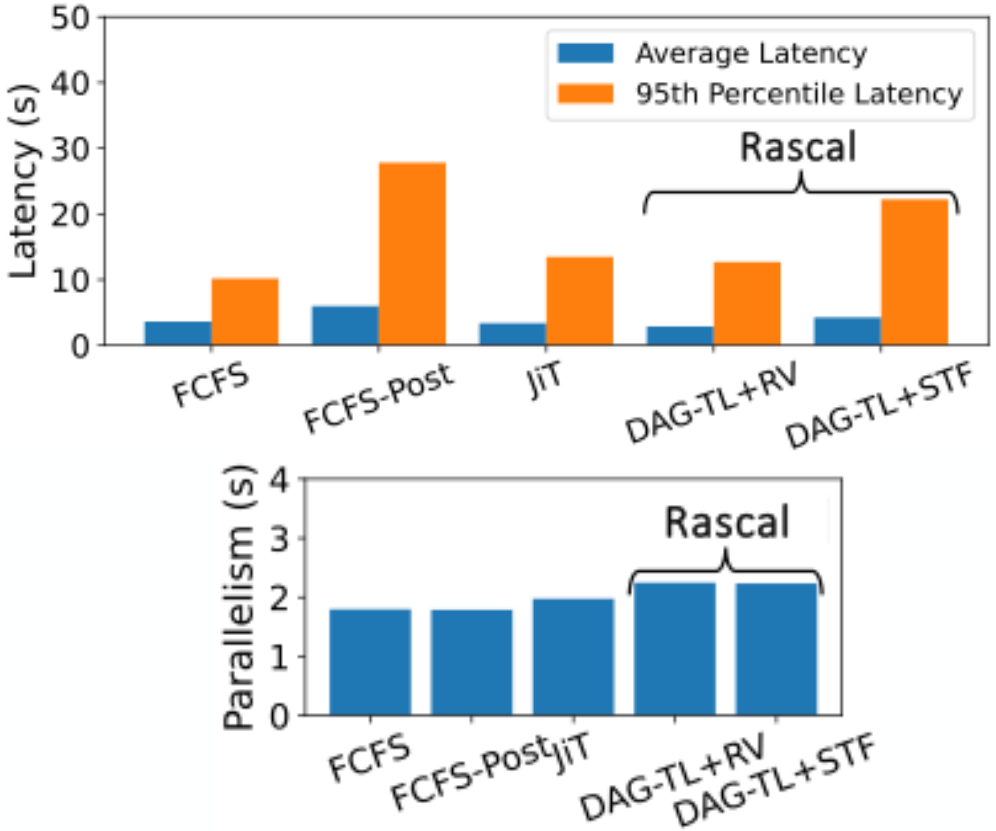
# How Efficient is Rascal's Scheduling?

- **Faster** routine **schedules**
  - Up to 11% vs best baseline
- **Lower wait & idle time**
  - Wait ↓ 33% (mean), 50% (p95)
  - Idle time ↓ 18–27%
- **Higher parallelism**
  - ~10% more than baselines



# How Efficient is Rascal's Scheduling?

- **Faster** routine **schedules**
  - Up to 11% vs best baseline
- **Lower wait & idle time**
  - Wait ↓ 33% (mean), 50% (p95)
  - Idle time ↓ 18–27%
- **Higher parallelism**
  - ~10% more than baselines



# RASC: Enhancing Observability & Programmability in Smart Spaces

# RASC: Enhancing Observability & Programmability in Smart Spaces

- **RASC: new abstraction** atop RPC

# RASC: Enhancing Observability & Programmability in Smart Spaces

- **RASC: new abstraction** atop RPC
- Implemented by our system **Rascal**, extending open-source Home Assistant, with minimal configuration extension

# RASC: Enhancing Observability & Programmability in Smart Spaces

- **RASC: new abstraction** atop RPC
- Implemented by our system **Rascal**, extending open-source Home Assistant, with minimal configuration extension
- **Adaptively polls** device states to detect action state change
- **Dynamically reschedules** automations to maintain **serializability & mutual exclusion**



# RASC: Enhancing Observability & Programmability in Smart Spaces

- **RASC: new abstraction** atop RPC
- Implemented by our system **Rascal**, extending open-source Home Assistant, with minimal configuration extension
- **Adaptively polls** device states to detect action state change
- **Dynamically reschedules** automations to maintain **serializability & mutual exclusion**
- **Human-facing systems → new opportunities**

# RASC: Enhancing Observability & Programmability in Smart Spaces

- **RASC: new abstraction** atop RPC
- Implemented by our system **Rascal**, extending open-source Home Assistant, with minimal configuration extension
- **Adaptively polls** device states to detect action state change
- **Dynamically reschedules** automations to maintain **serializability & mutual exclusion**
- **Human-facing systems → new opportunities**

- Access the paper here!


- Questions?
- **Anna Karanika** (annak8@illinois.edu)

nsdi '26