RL4Sys: A Lightweight System-Driven RL Framework for Drop-in Integration in System Optimization

Jiaxin Dong[1]
jiddong@udel.edu

Md. Hasanur Rashid[1]
mrashid@udel.edu

Helen Xu[2]
hxu615@gatech.edu

Dong Dai[1]
dai@udel.edu

1 University of Delaware

2 Georgia Tech

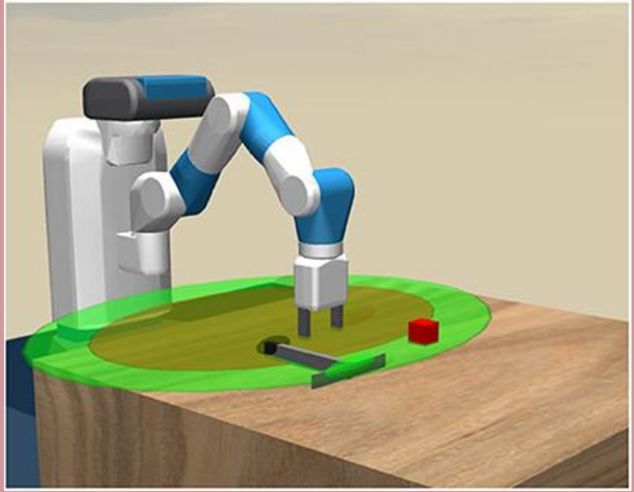
DIRLAB

DIRLAB

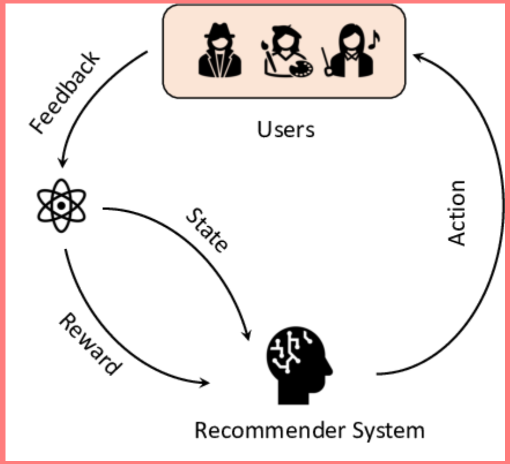
Reinforcement Learning in **Modern** Scenarios

Atari Game
(Mnih *et al.*, 2015)

Robotic
(Levine *et al.*, 2016)



Recommendation
(Zou *et al.*, 2019)



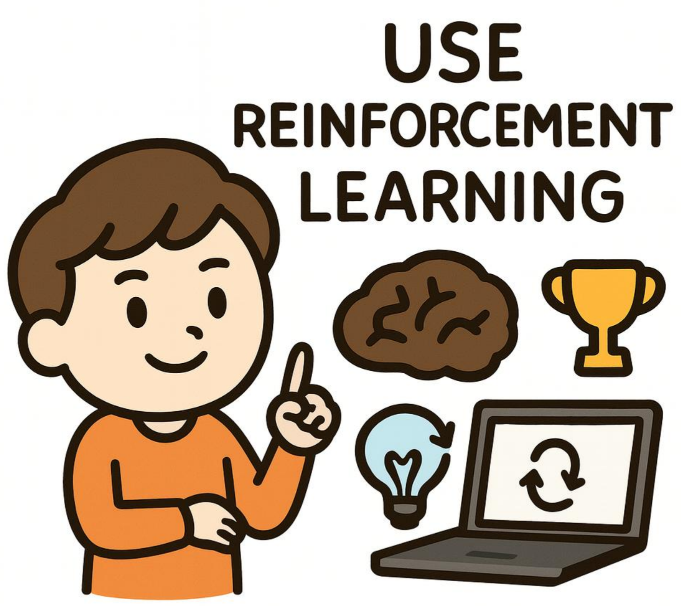
Mnih, V., Kavukcuoglu, K., Silver, D., *et al.* "Human-level control through deep reinforcement learning." *Nature* 518, 529–533, 2015.

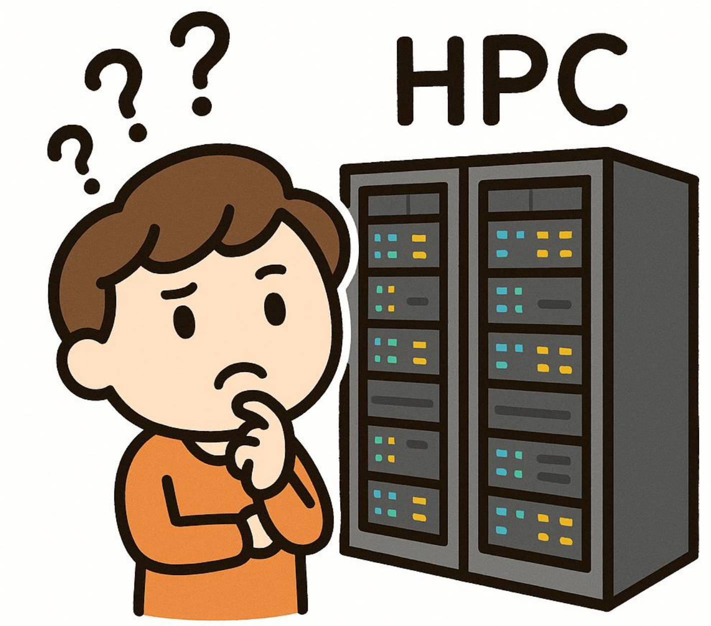
Levine, S., Finn, C., Darrell, T., & Abbeel, P. "End-to-End Training of Deep Visuomotor Policies." *Journal of Machine Learning Research* 17(39), 1–40, 2016

Zou, L., Xia, L., Ding, Z., Song, J., Liu, W., & Yin, D. "Reinforcement Learning to Optimize Long-term User Engagement in Recommender Systems." In *Proceedings of the 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19)*, 2810–2818, 2019.

DIRLAB

Reinforcement Learning in System Scenarios





DIRLAB

Why RL Is Hard to use in Real World Systems?

Traditional RL Frameworks

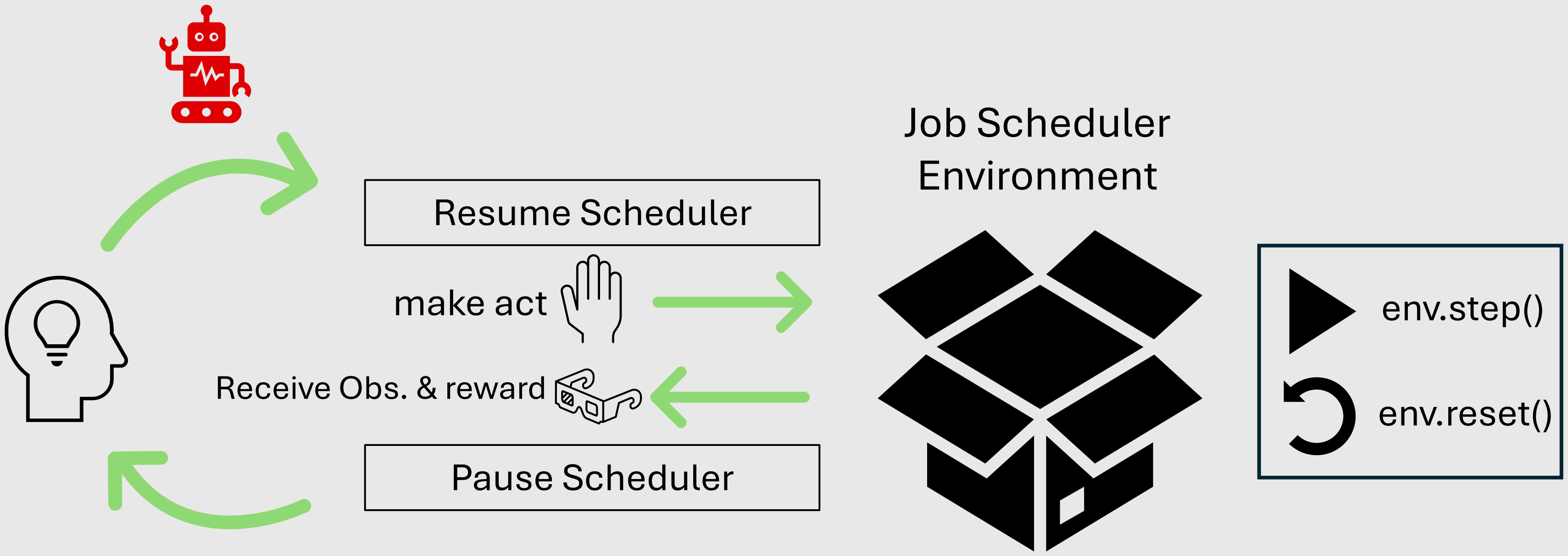
Stable Baseline 3

OpenAI Spinning Up

RLlib

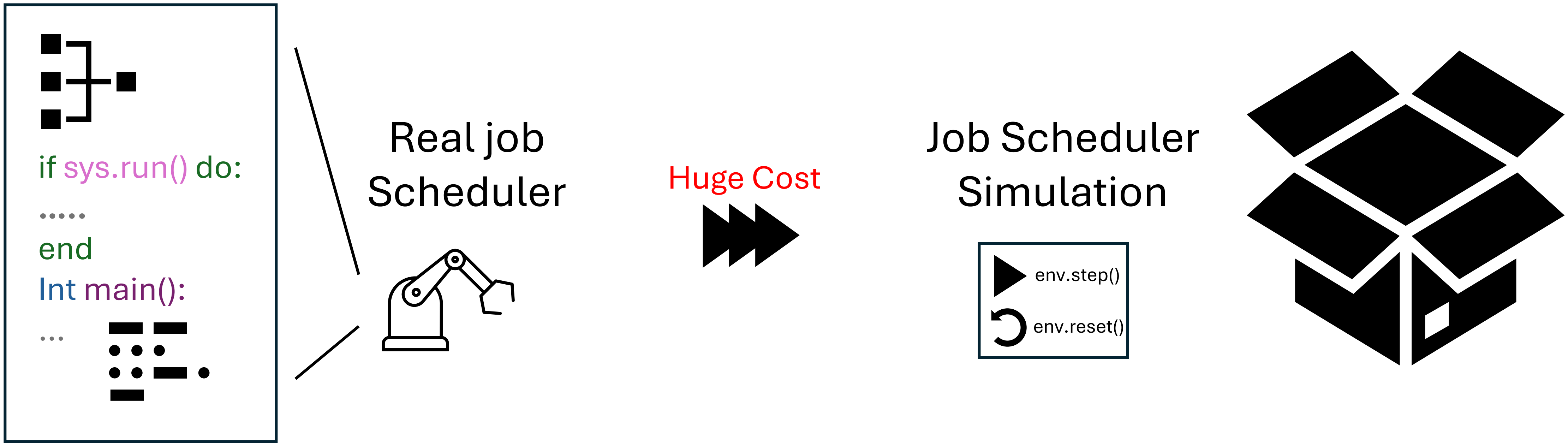
Traditional **Agent-Driven Paradigm**

For ex.
Job Scheduler



DIRLAB

Why RL Is Hard to use in Real World Systems?

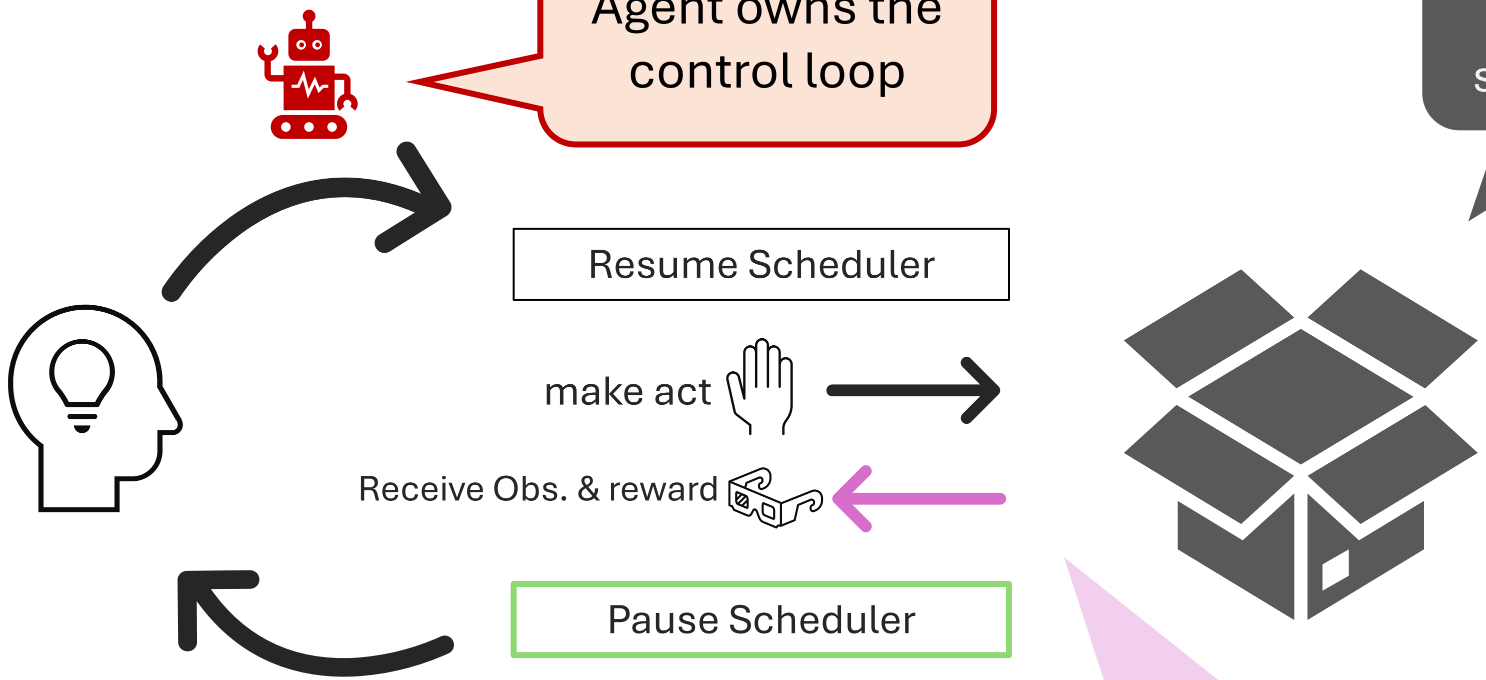


DIRLAB

Why RL Is Hard to use in Real World Systems?

Agent owns the control loop

Use fixed HPC trace or synthesized trace



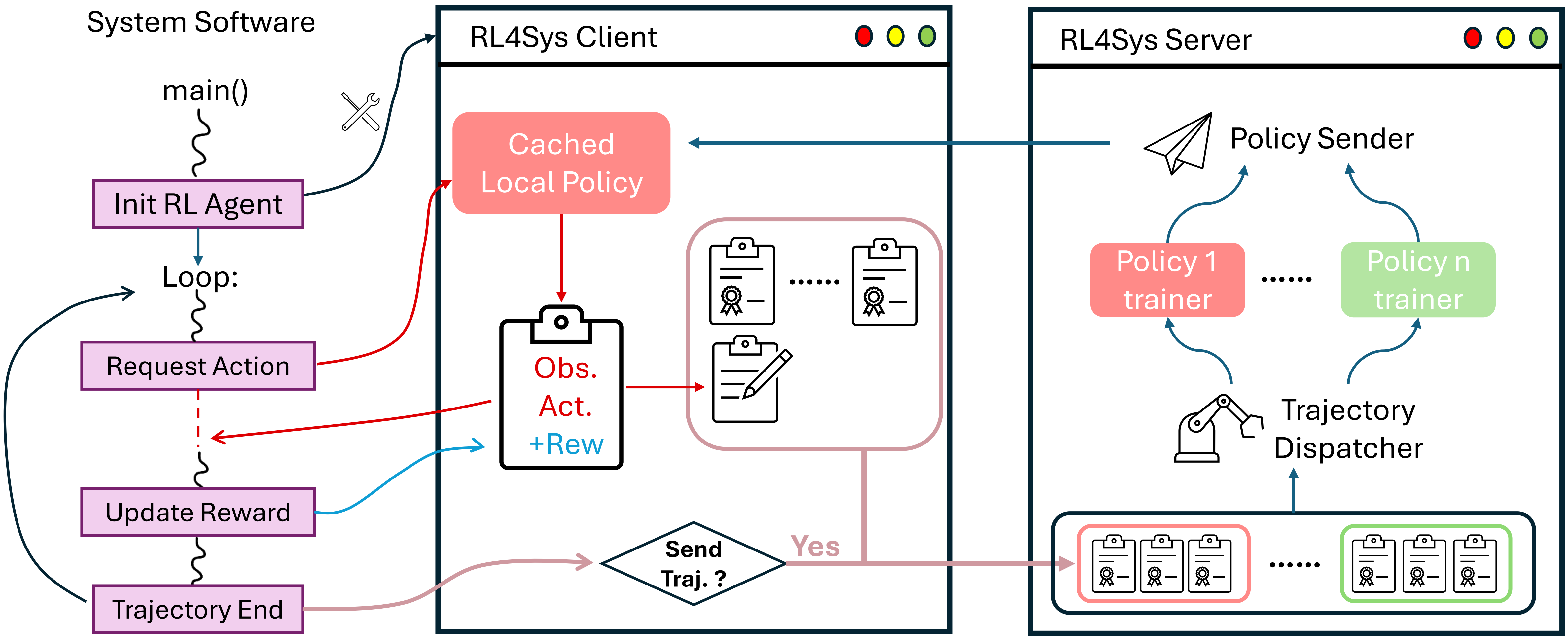
Job Scheduler Simulation

Synchronize/blocking Scheduler simulation running

Assuming next observation and rewards **are immediately available** after each action

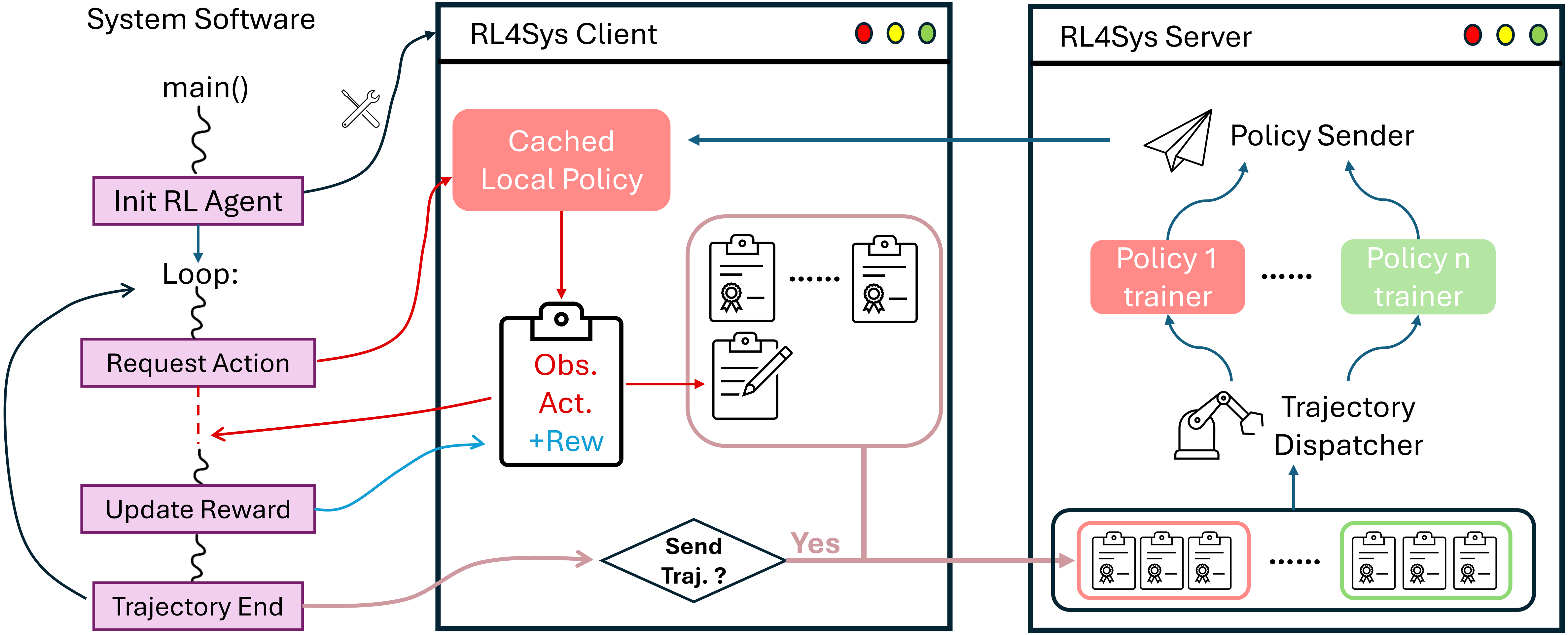
DIRLAB

RL4Sys: A Lightweight System-Driven RL Framework



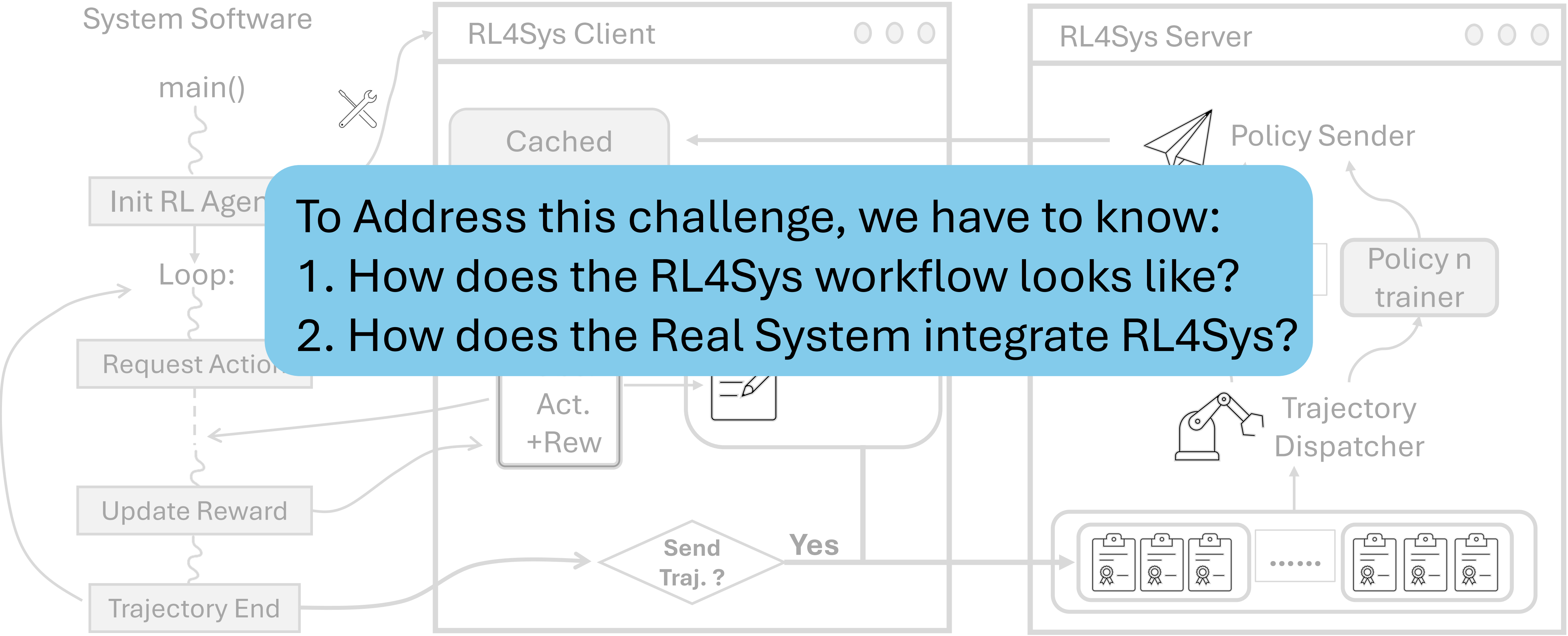
DIRLAB

Challenge 1: **how to define system friendly interface**



DIRLAB

Challenge 1: **how to define system friendly interface**



To Address this challenge, we have to know:
1. How does the RL4Sys workflow looks like?
2. How does the Real System integrate RL4Sys?

DIRLAB

Challenge 1: **how to define system friendly interface**

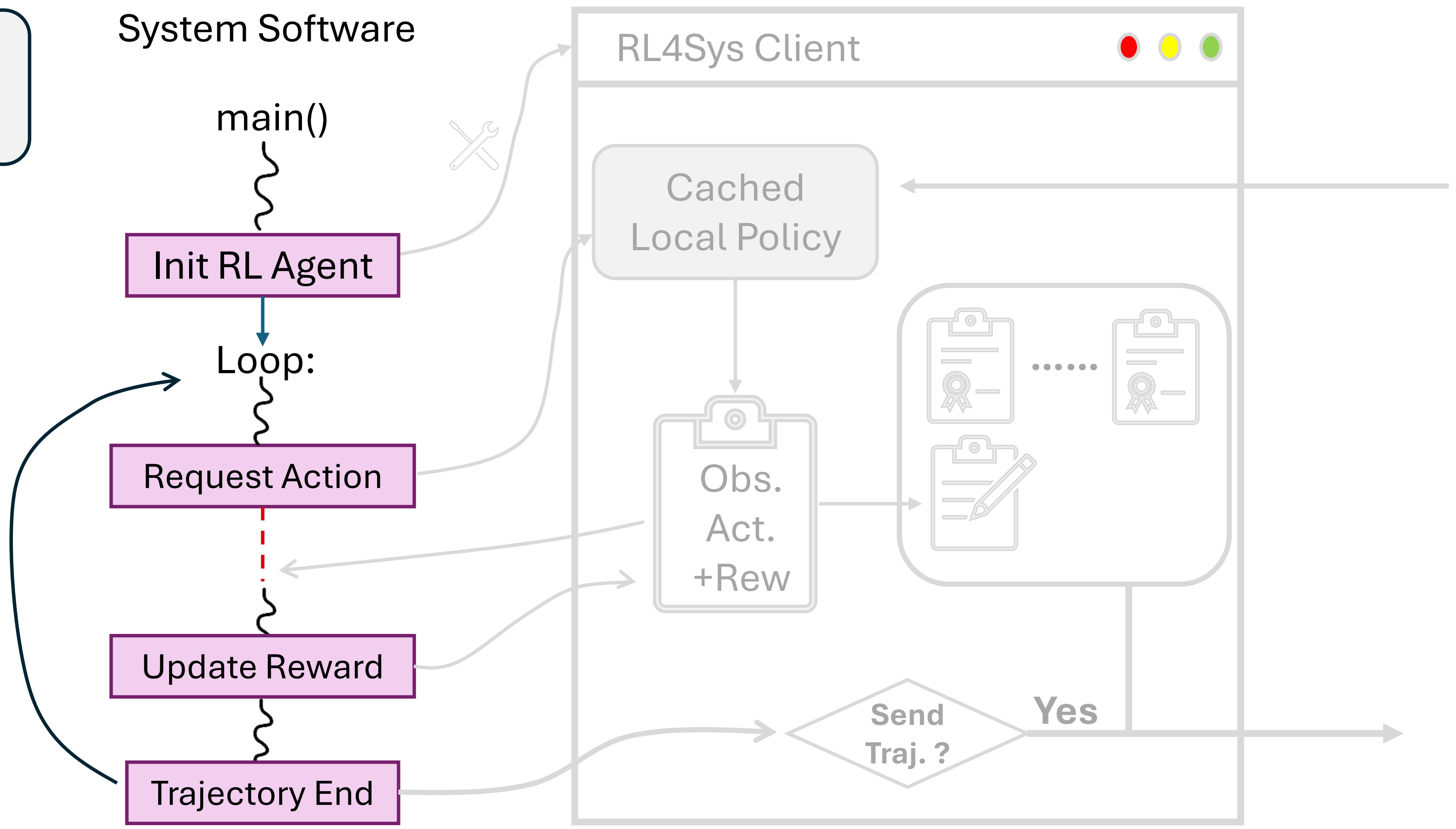
1. RL4Sys only have maximum 5 APIs

RLAgent = RL4sys.init()

act, traj = rl4sys.request_action(obs)
traj.add(act)

act.update_reward(value)

traj.mark_end_of_trajectory()



DIRLAB

Challenge 1: **how to define system friendly interface**

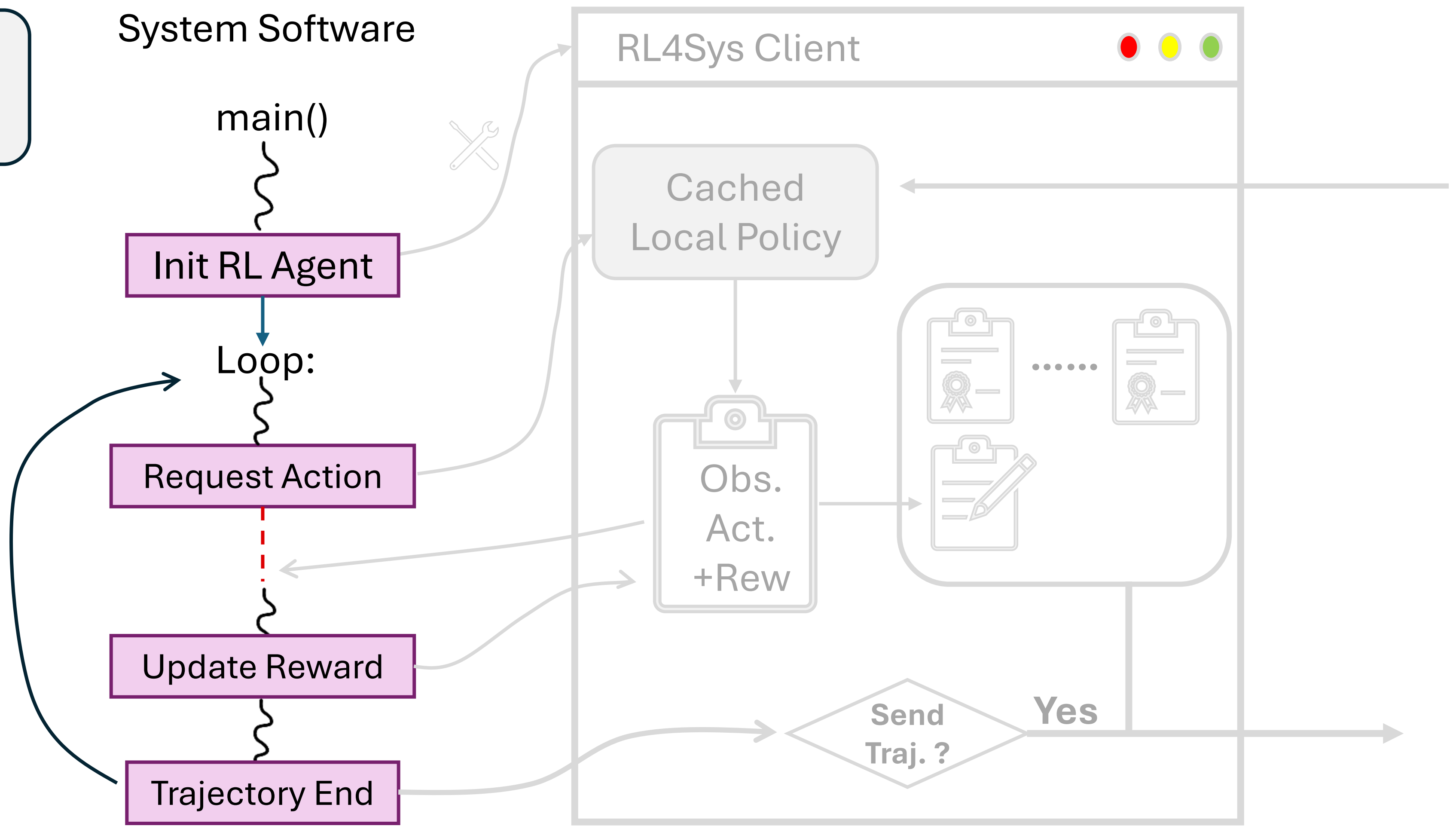
1. RL4Sys only have maximum 5 APIs

RLAgent = RL4sys.init()

act, traj = rl4sys.request_action(obs)
traj.add(act)

act.update_reward(value)

traj.mark_end_of_trajectory()



DIRLAB

Challenge 1: **how to define system friendly interface**

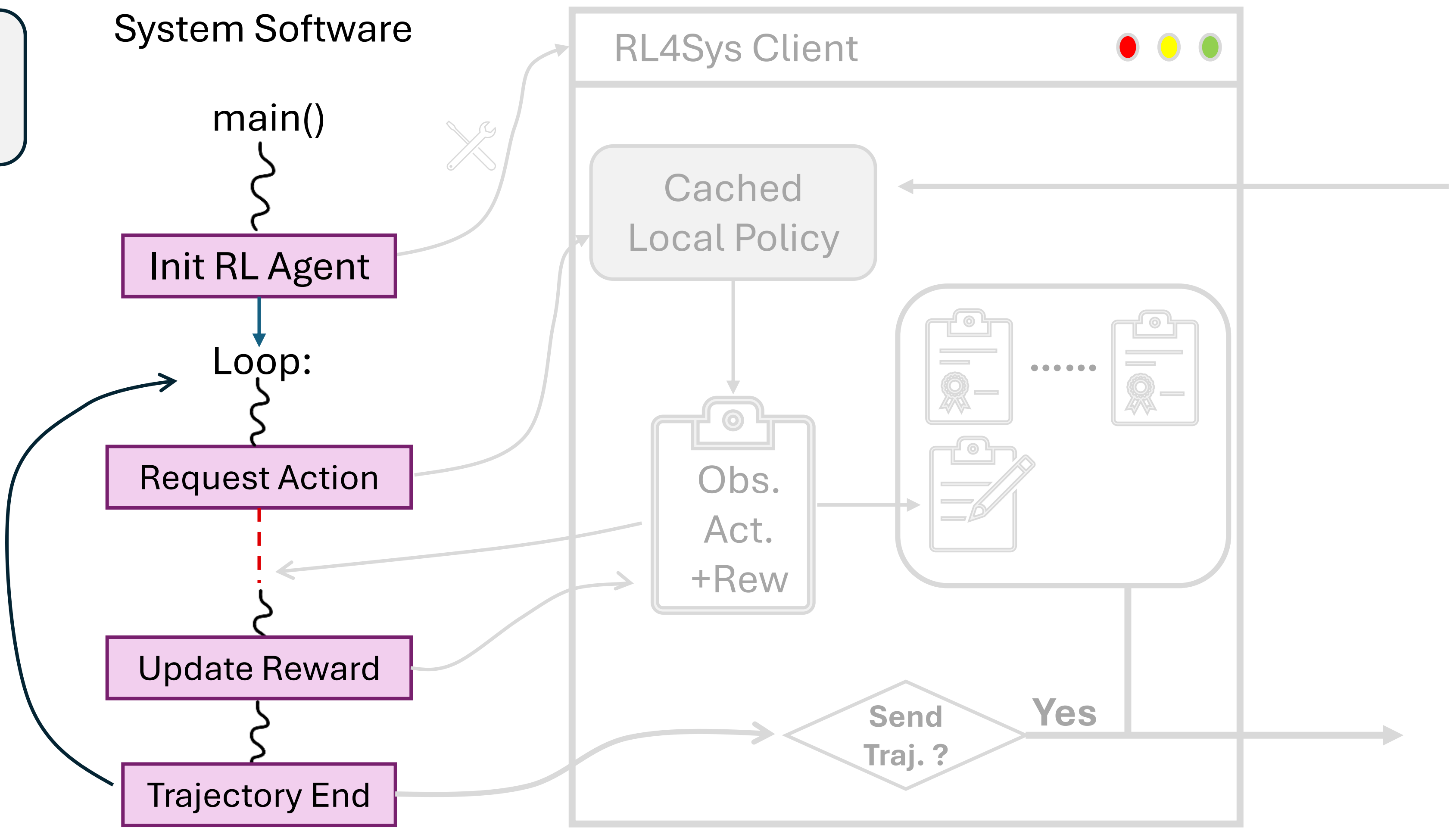
1. RL4Sys only have maximum 5 APIs

RLAgent = RL4sys.init()

act, traj = rl4sys.request_action(obs)
traj.add(act)

act.update_reward(value)

traj.mark_end_of_trajectory()



DIRLAB

Challenge 1: **how to define system friendly interface**

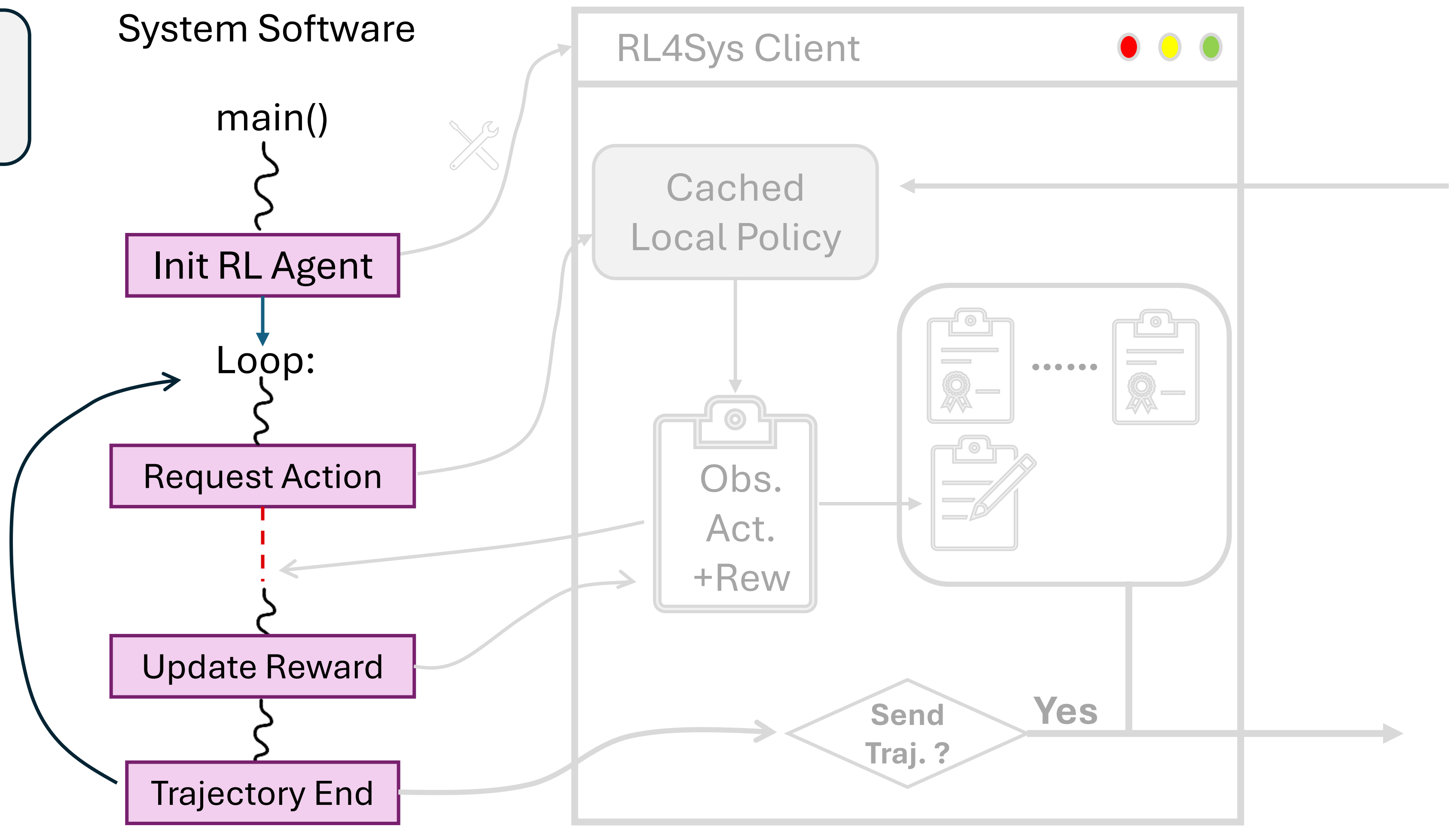
1. RL4Sys only have maximum 5 APIs

RLAgent = RL4sys.init()

act, traj = rl4sys.request_action(obs)
traj.add(act)

act.update_reward(value)

traj.mark_end_of_trajectory()



Challenge 1: **how to define system friendly interface**

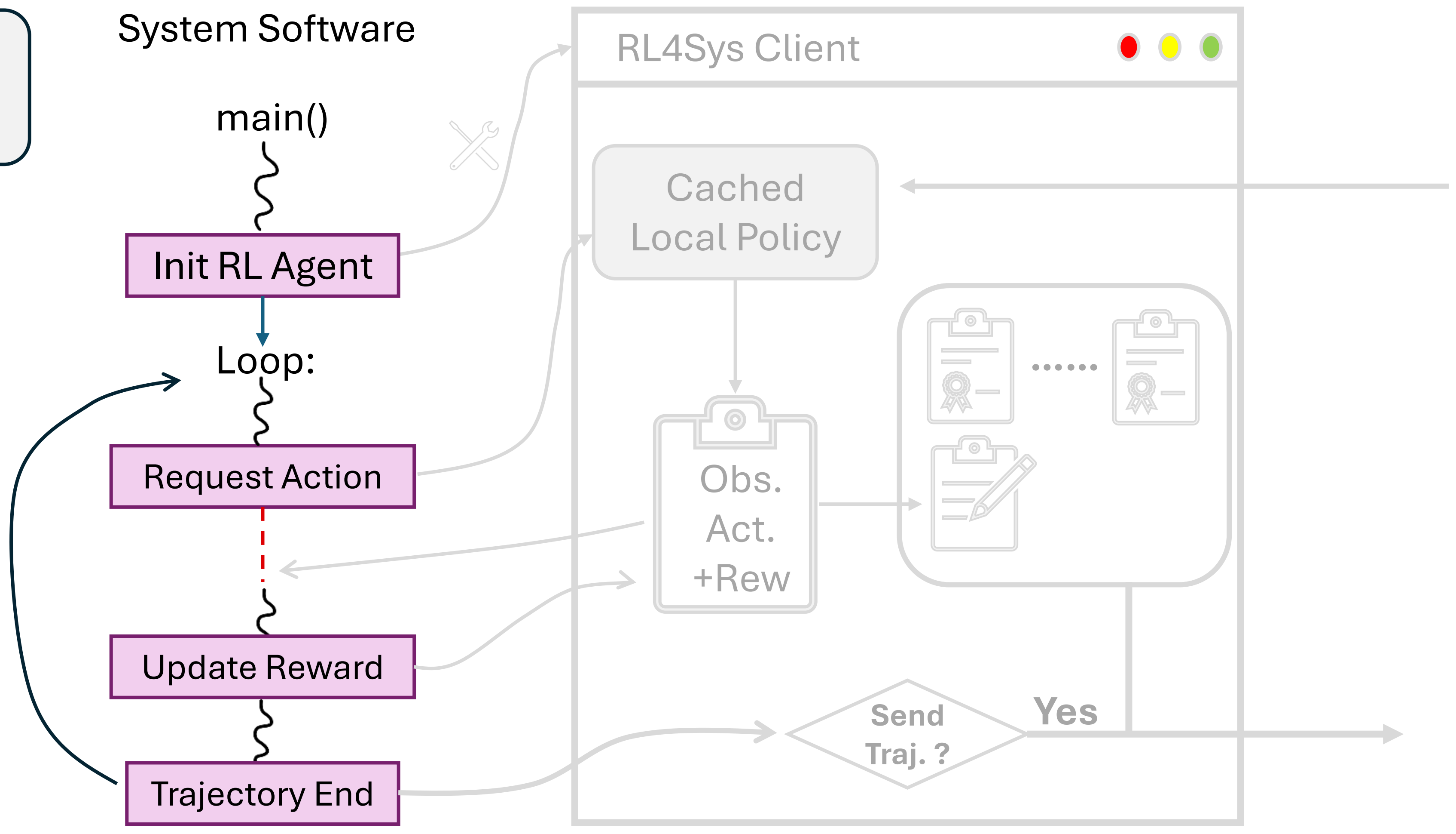
1. RL4Sys only have maximum 5 APIs

RLAgent = RL4sys.init()

act, traj = rl4sys.request_action(obs)
traj.add(act)

act.update_reward(value)

traj.mark_end_of_trajectory()



DIRLAB

Challenge 1: **how to define system friendly interface**

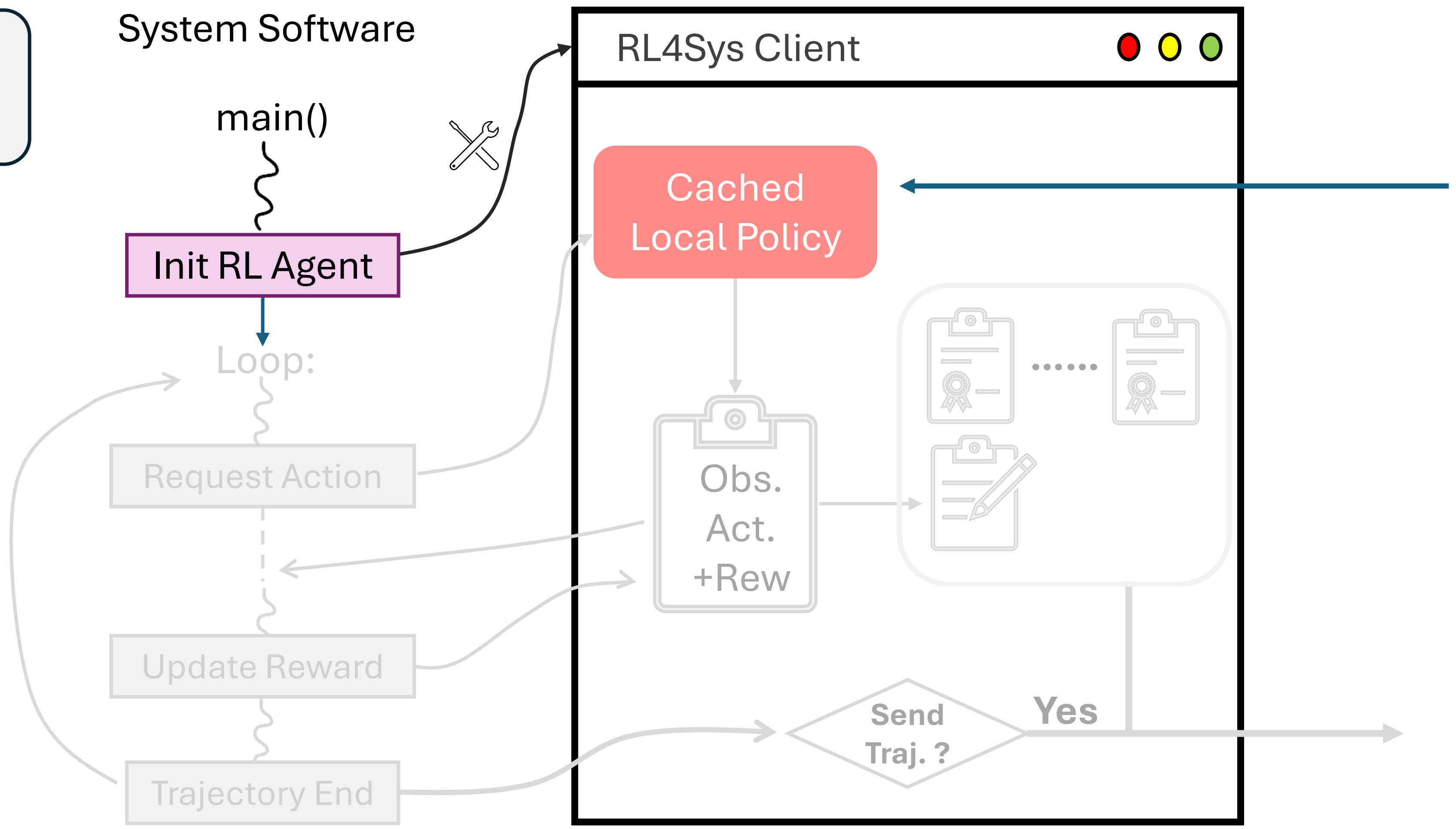
1. Always retrieve best policy from Server

RLAgent = RL4sys.init()

act, traj = rl4sys.request_action(obs)
traj.add(act)

act.update_reward(value)

traj.mark_end_of_trajectory()



DIRLAB

Challenge 1: **how to define system friendly interface**

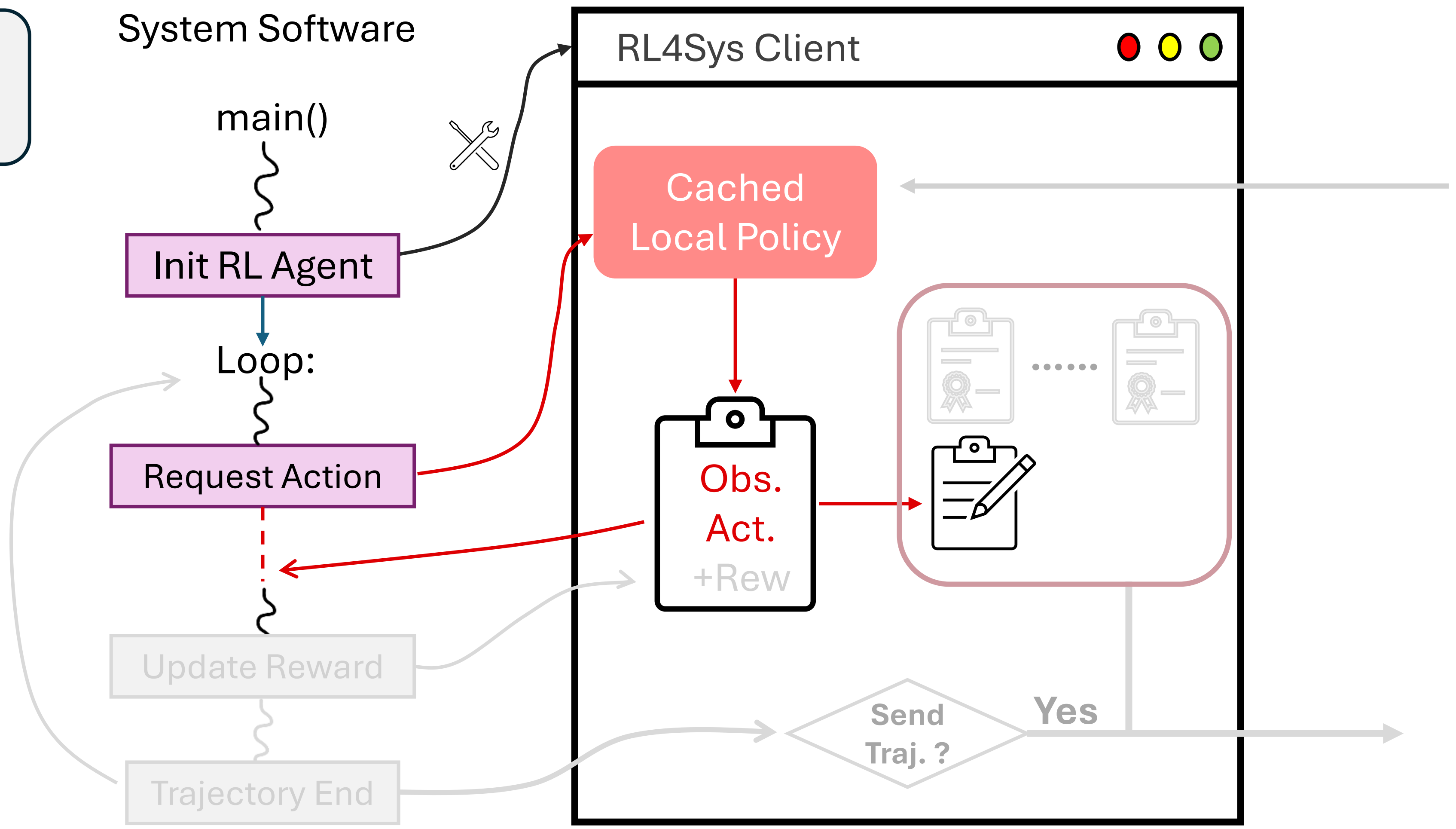
2. System call Agent for dominant control

RLAgent = RL4sys.init()

act, traj = rl4sys.request_action(obs)
traj.add(act)

act.update_reward(value)

traj.mark_end_of_trajectory()



DIRLAB

Challenge 1: **how to define system friendly interface**

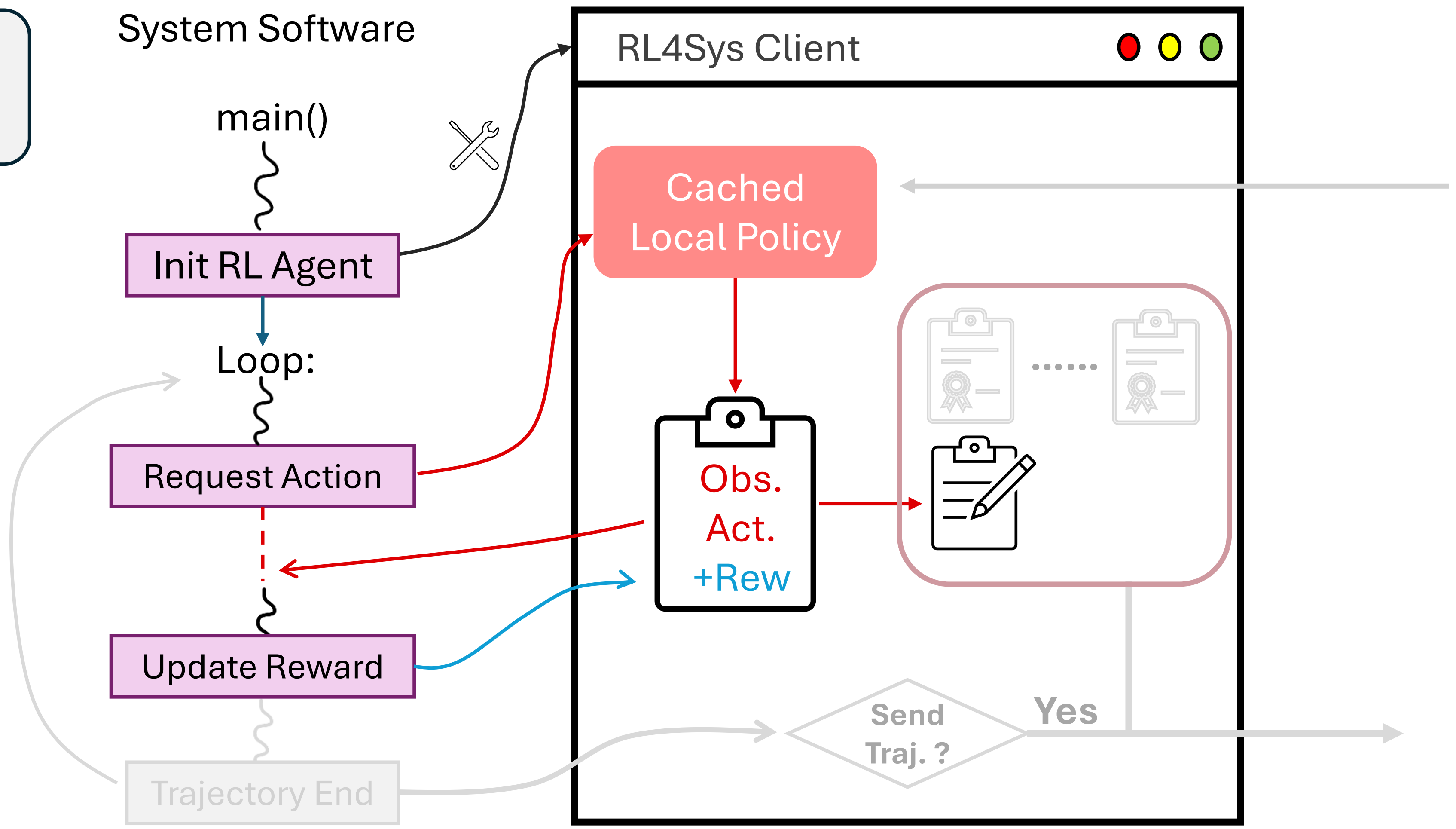
3. Delay reward updating strategy best fit for system

RLAgent = RL4sys.init()

act, traj = rl4sys.request_action(obs)
traj.add(act)

act.update_reward(value)

traj.mark_end_of_trajectory()



DIRLAB

Challenge 1: **how to define system friendly interface**

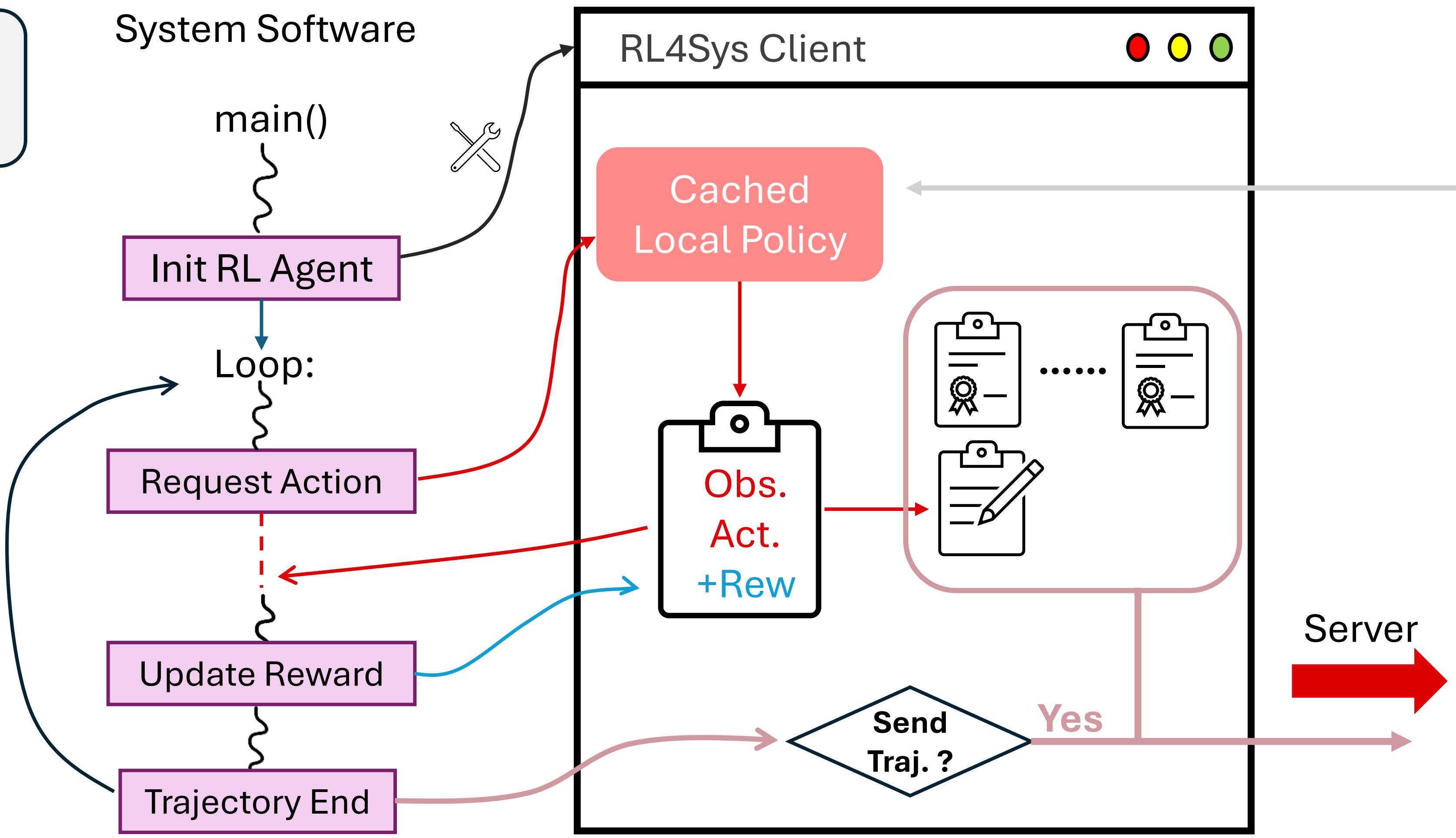
4. Send Trajectory to server at the backend

RLAgent = RL4sys.init()

act, traj = rl4sys.request_action(obs)
traj.add(act)

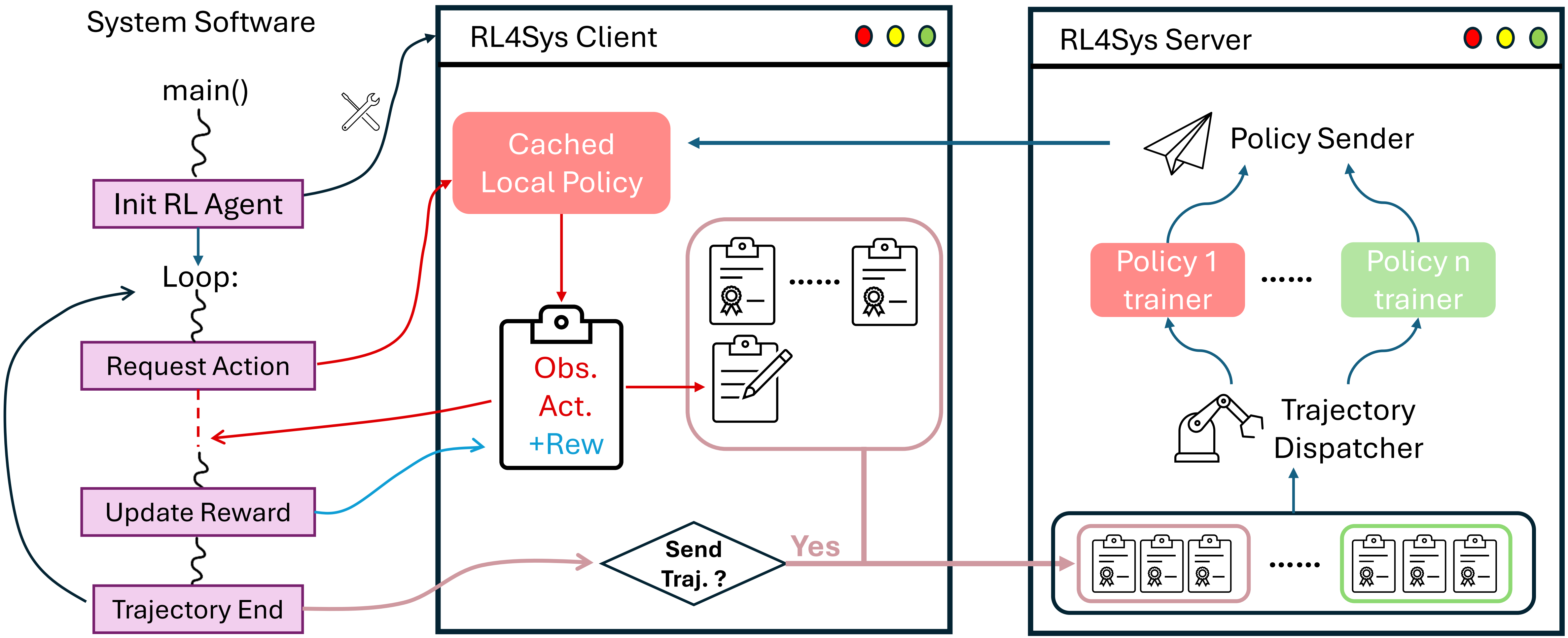
act.update_reward(value)

traj.mark_end_of_trajectory()



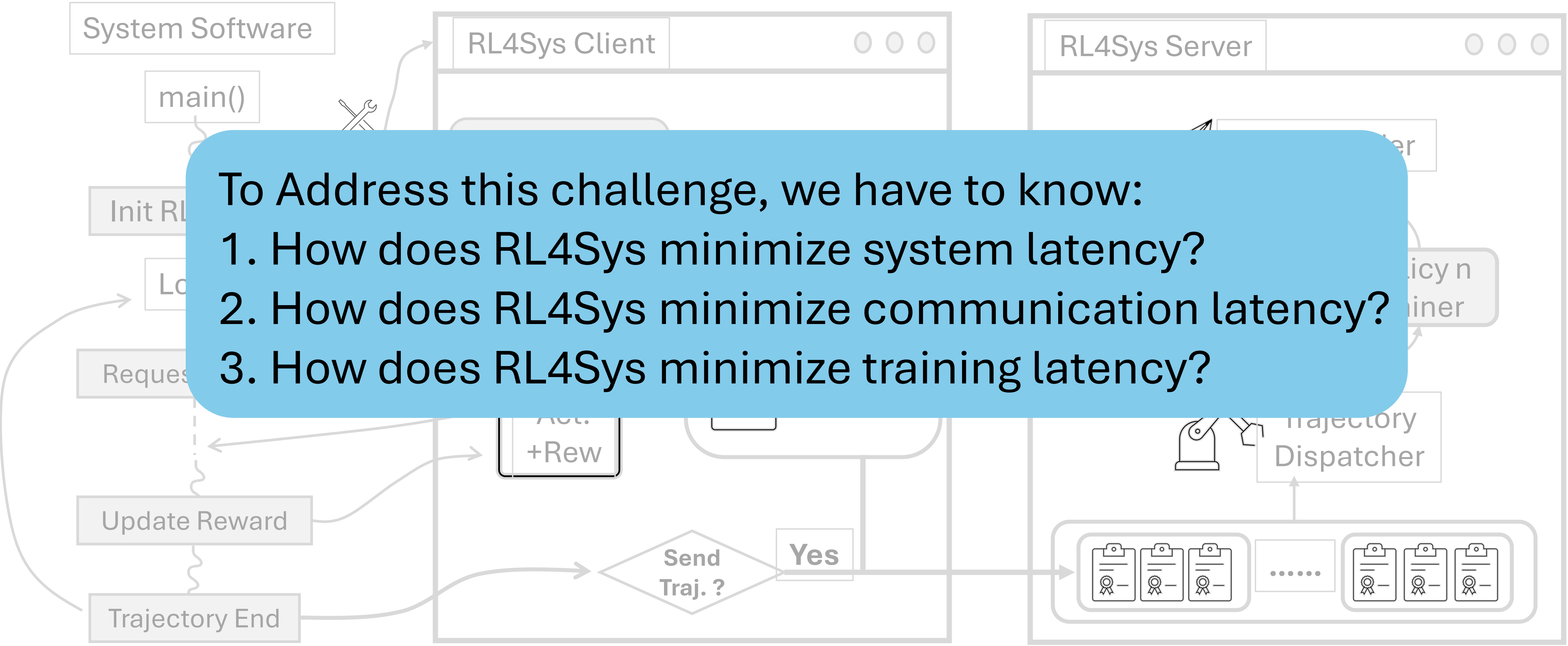
DIRLAB

Challenge 2: **Prevent System Stalling & Overhead**



DIRLAB

Challenge 2: **Prevent System Stalling & Overhead**

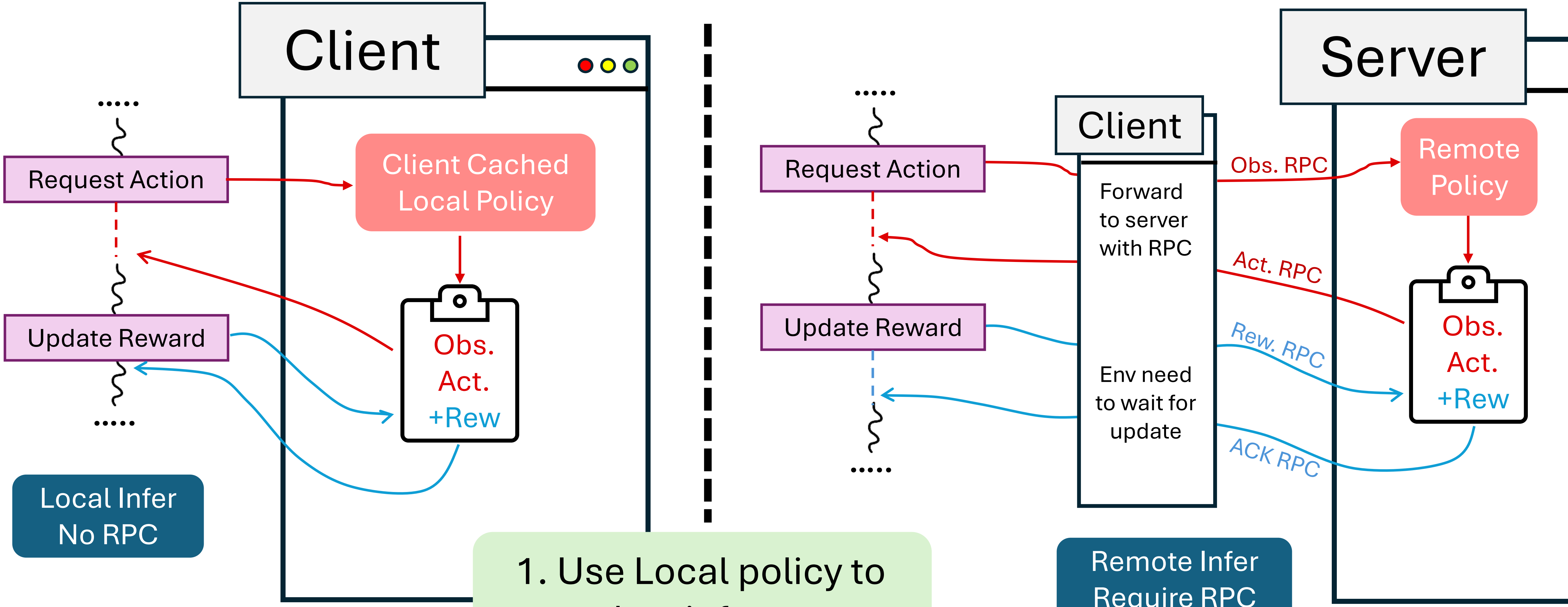


To Address this challenge, we have to know:

1. How does RL4Sys minimize system latency?
2. How does RL4Sys minimize communication latency?
3. How does RL4Sys minimize training latency?

DIRLAB

Challenge 2: Prevent System Stalling & Overhead

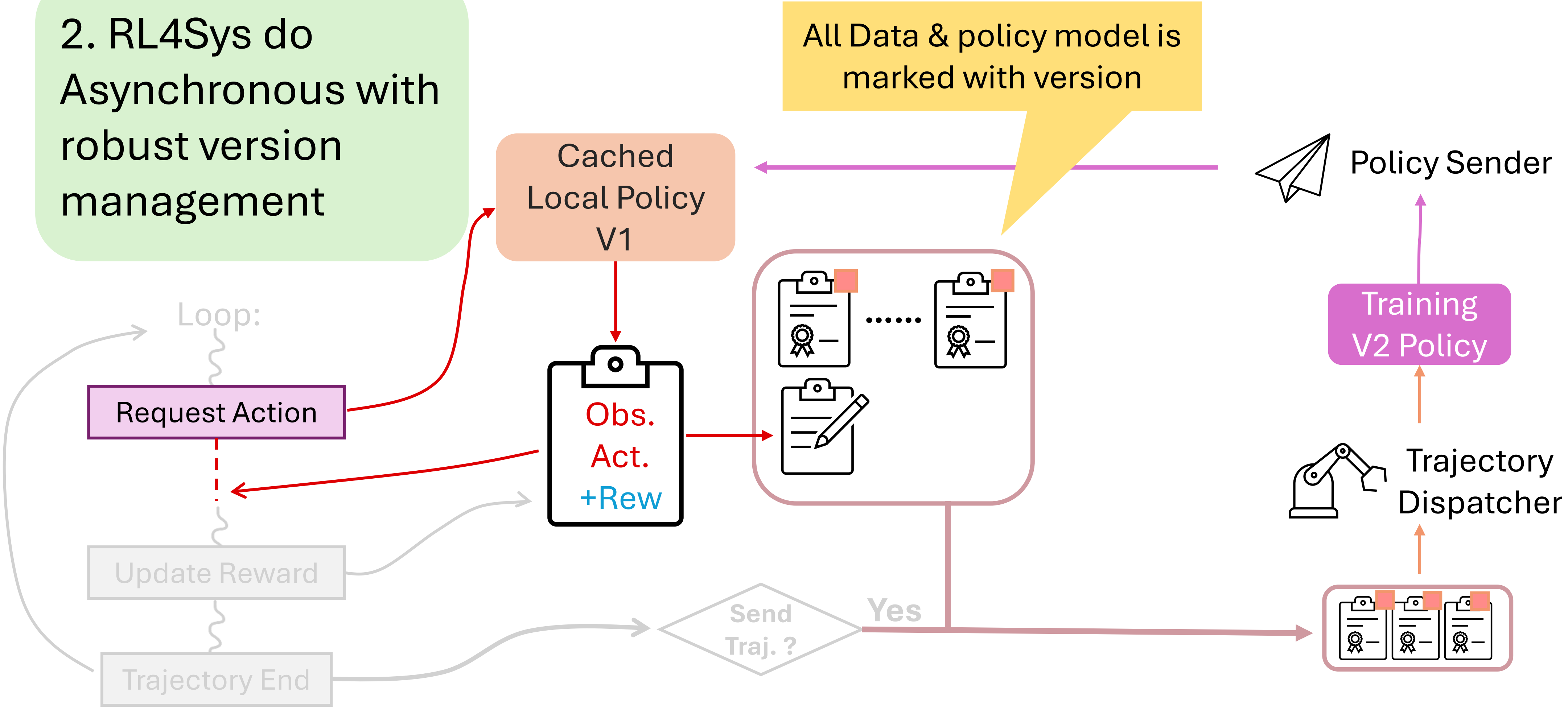


1. Use Local policy to speed up inference

DIRLAB

Challenge 2: Prevent System Stalling & Overhead

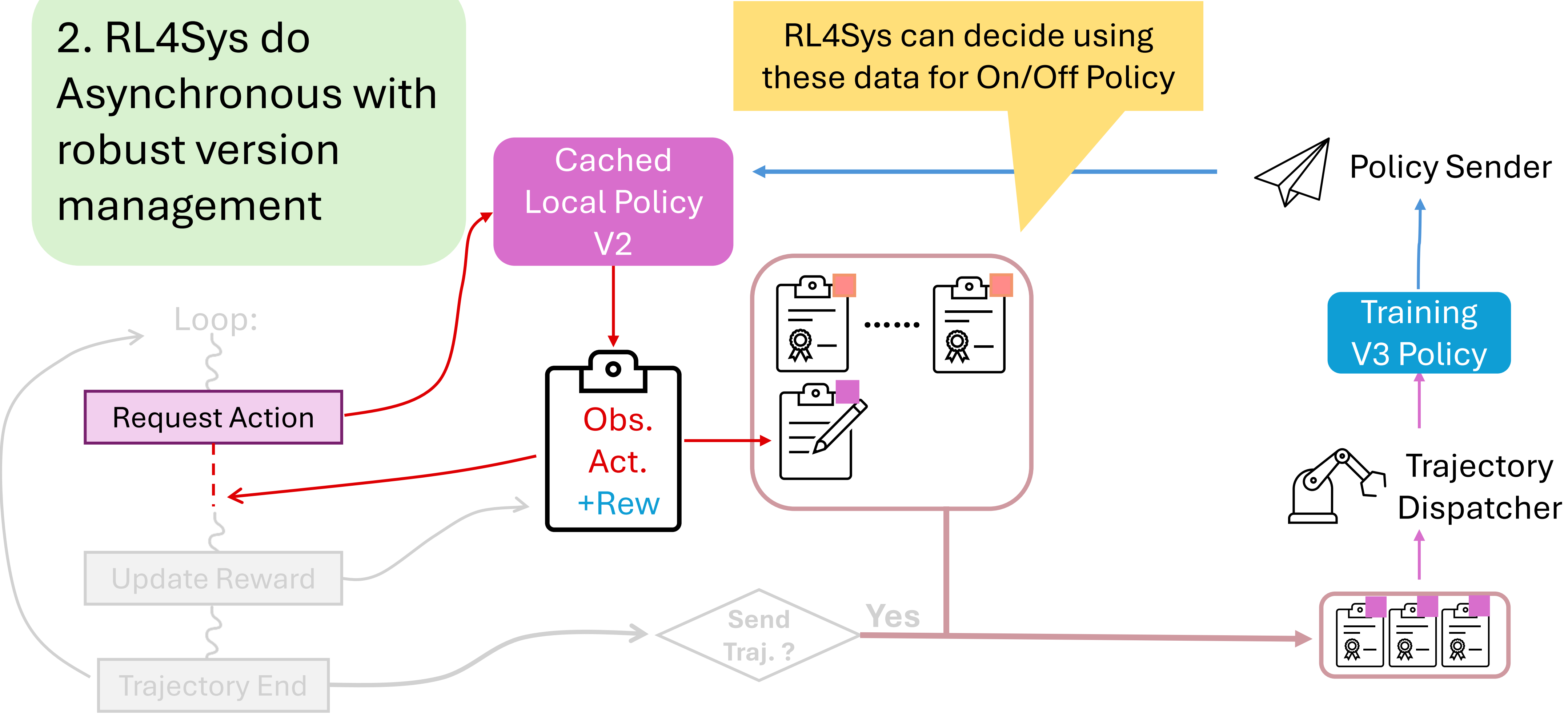
2. RL4Sys do Asynchronous with robust version management



DIRLAB

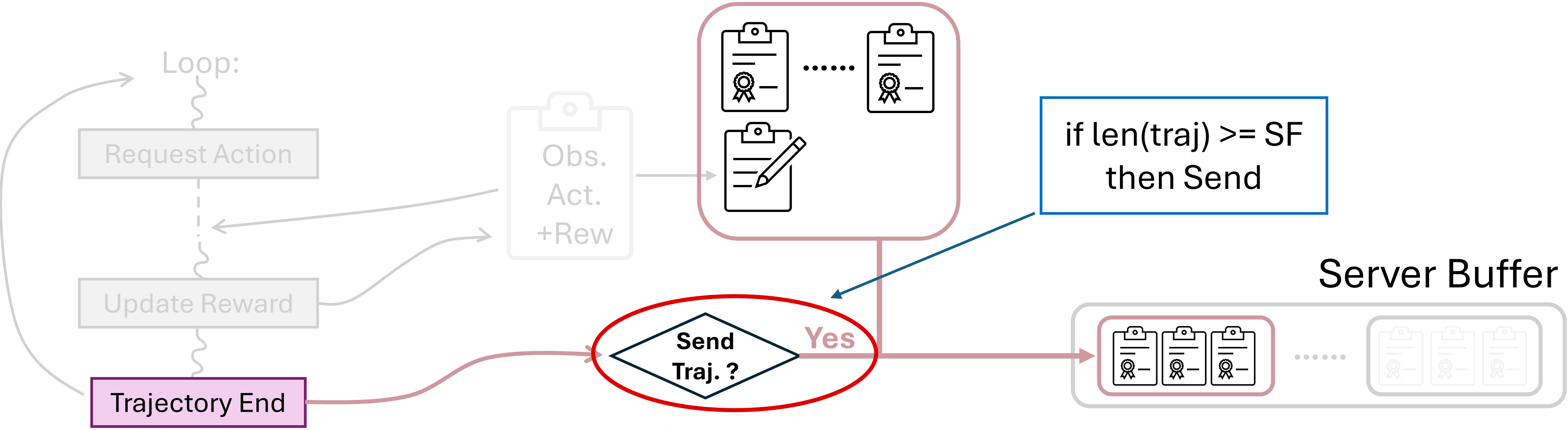
Challenge 2: Prevent System Stalling & Overhead

2. RL4Sys do Asynchronous with robust version management



DIRLAB

Challenge 2: Prevent System Stalling & Overhead

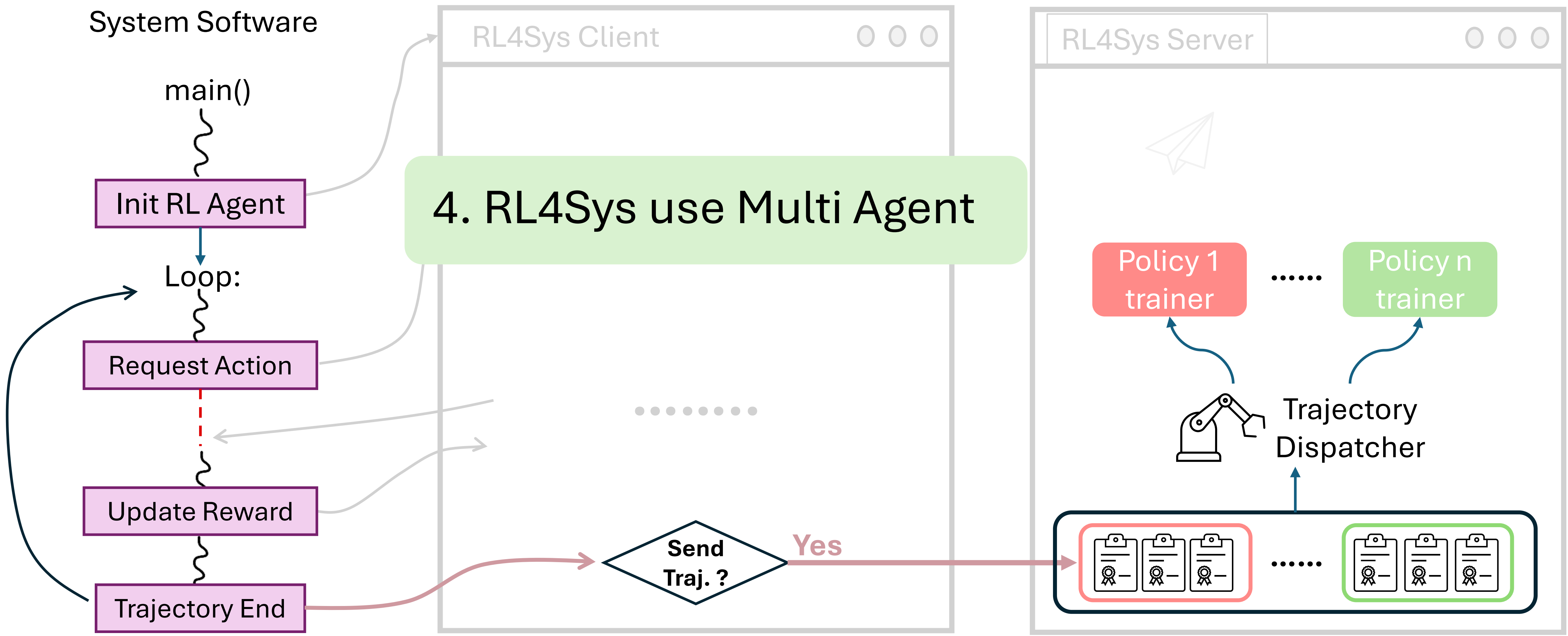


Sending condition depend on a parameter called **send frequency (SF)**

3. RL4Sys use gRPC & tunable RPC parameters

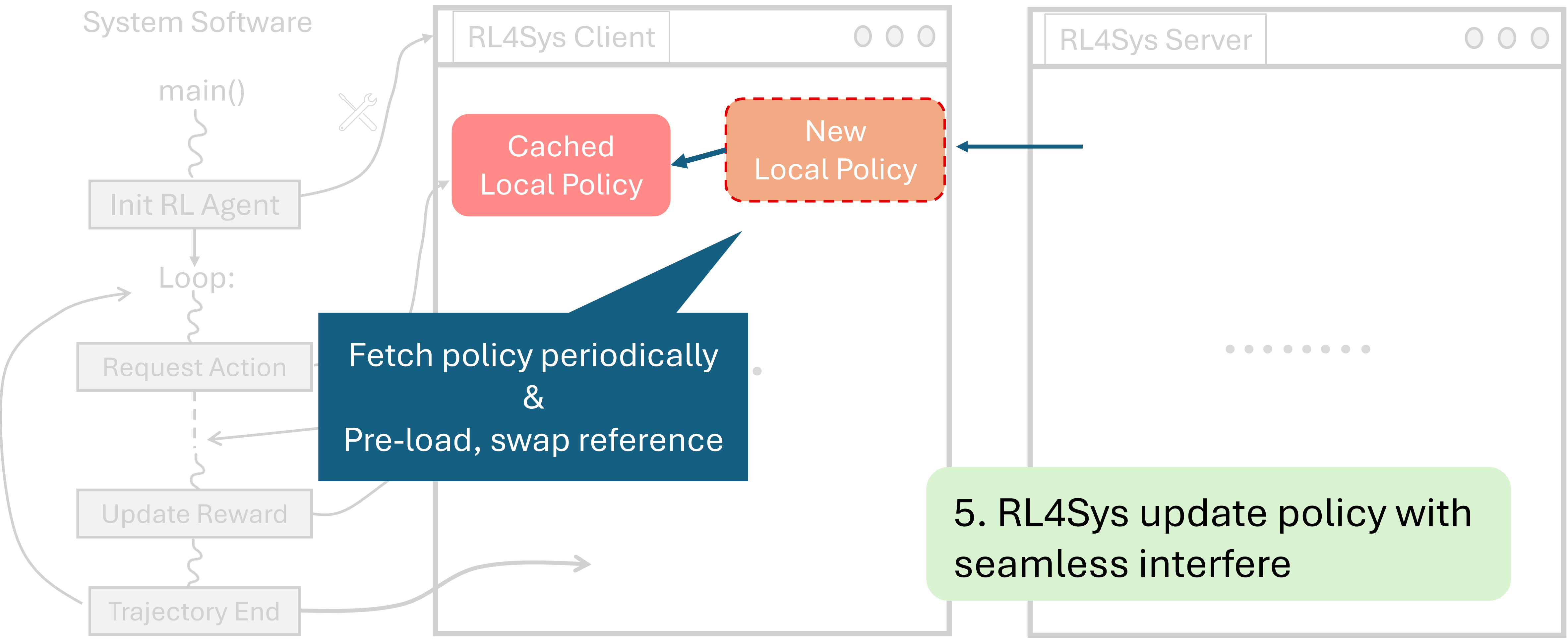
DIRLAB

Challenge 2: Prevent System Stalling & Overhead



DIRLAB

Challenge 2: Prevent System Stalling & Overhead



5. RL4Sys update policy with seamless interfere

DIRLAB

Evaluation of RL4Sys

- We use Real world use case examples as evidence of **RL4Sys' correctness and performance** and the **feasibility on other System scenarios**.

- High throughput
  - **6%** overhead than baseline, **2.2x** speedup than the SOTA solution, RLlib
- Low resource consumption
  - CPU Usage/Core: **3%** overhead than baseline, **5x** optimized than RLlib
  - Memory Usage: **3-7%** overhead than baseline compare with **20%** overhead for RLlib

Evaluation – Job Scheduling (Setup & Baselines)

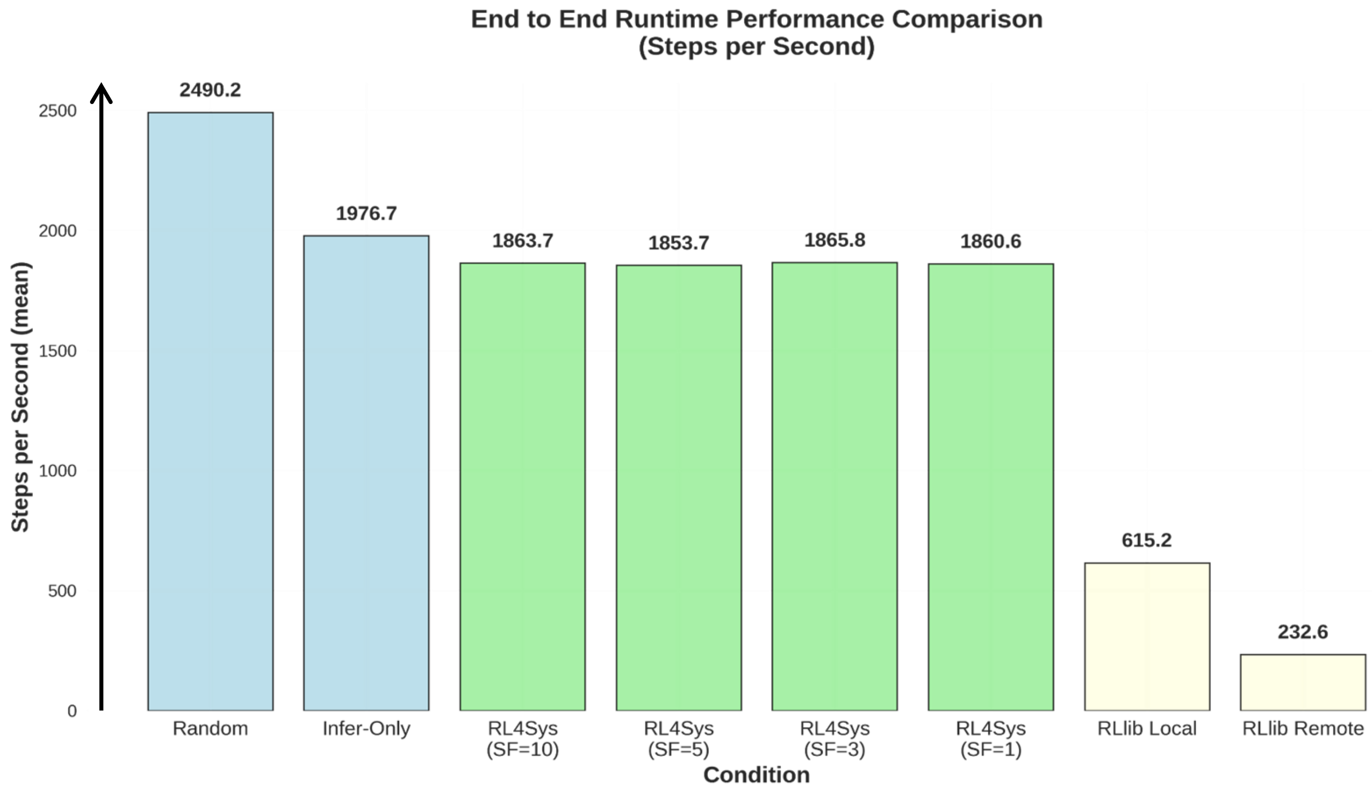
- We evaluate on two HPC scheduling workloads: either from a real cluster **SDSC SP2** or a synthesized trace **Lublin256**.

4 Baselines:

- **Random Scheduling:** scheduler with random actions. Simulate normal Scheduler behavior.
- **Static NN Policy:** A policy network integrated in scheduler without RL update. Simulate necessary latency using RL
- **Conventional RLlib:** An RL-based Scheduler using RLlib (Ray's framework) in both local and remote modes.
  - **Local RLlib:** same client-server architecture as RL4Sys
  - **Remote RLlib:** Both policy and trainer is on server; Client must use heavy communication and may block when fetching actions.

Evaluation – Job Scheduling (Throughput)

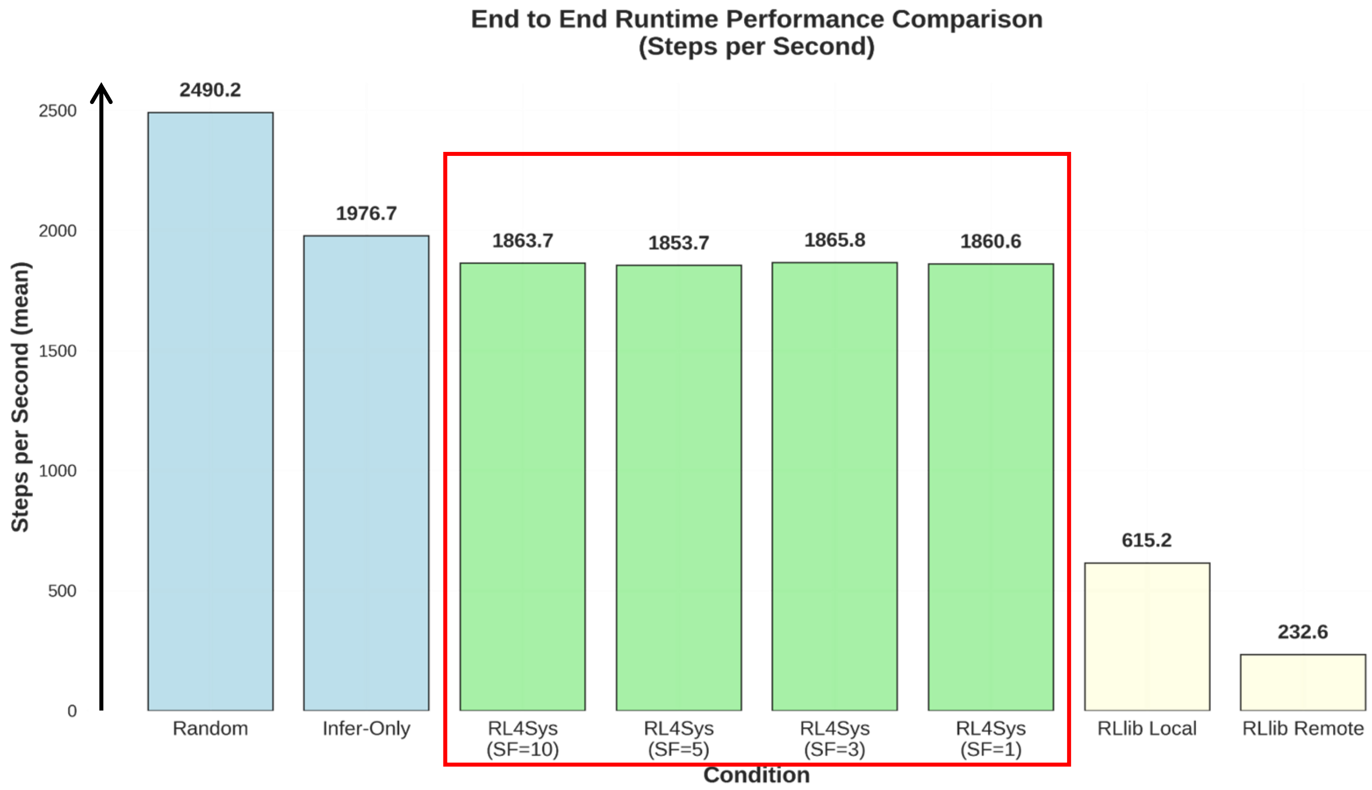
- **Higher Throughput:** RL4Sys achieves up to **~2.2×** the throughput of the **RLlib-based schedulers**
- The total runtime overhead stays under 6% compare with **Static NN Policy**



DIRLAB

Evaluation – Job Scheduling (Throughput)

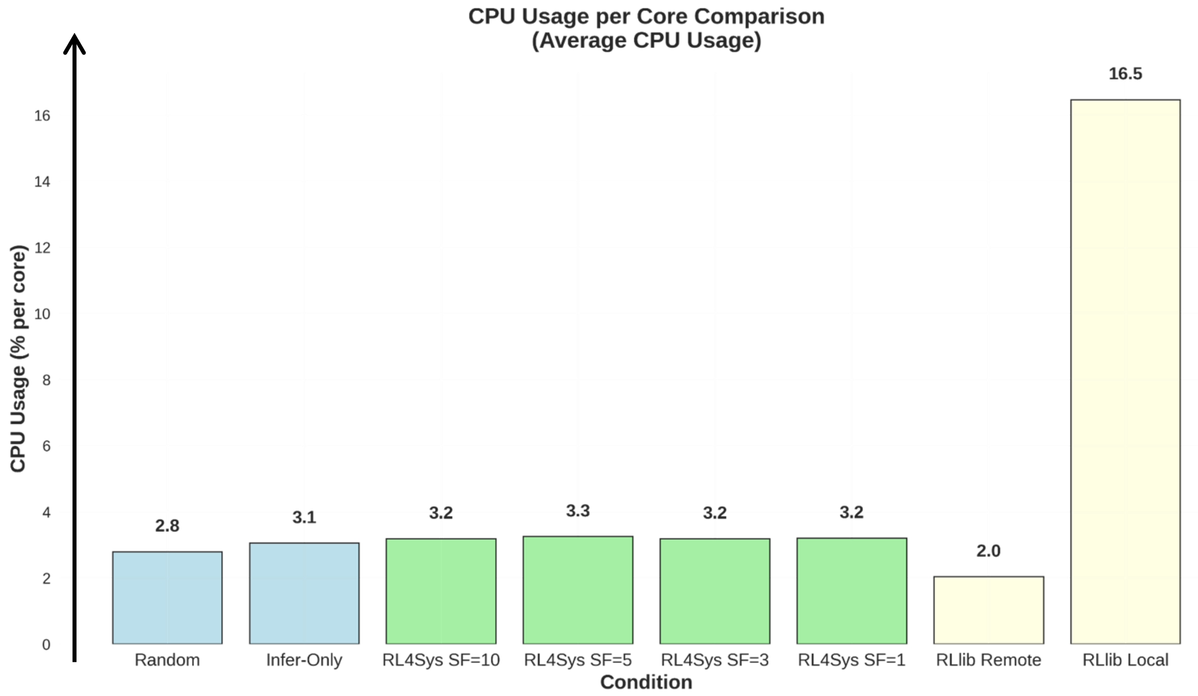
- **Higher Throughput:** RL4Sys achieves up to **~2.2×** the throughput of the **RLlib-based schedulers**
- The total runtime overhead stays under **6%** compare with **Static NN Policy**

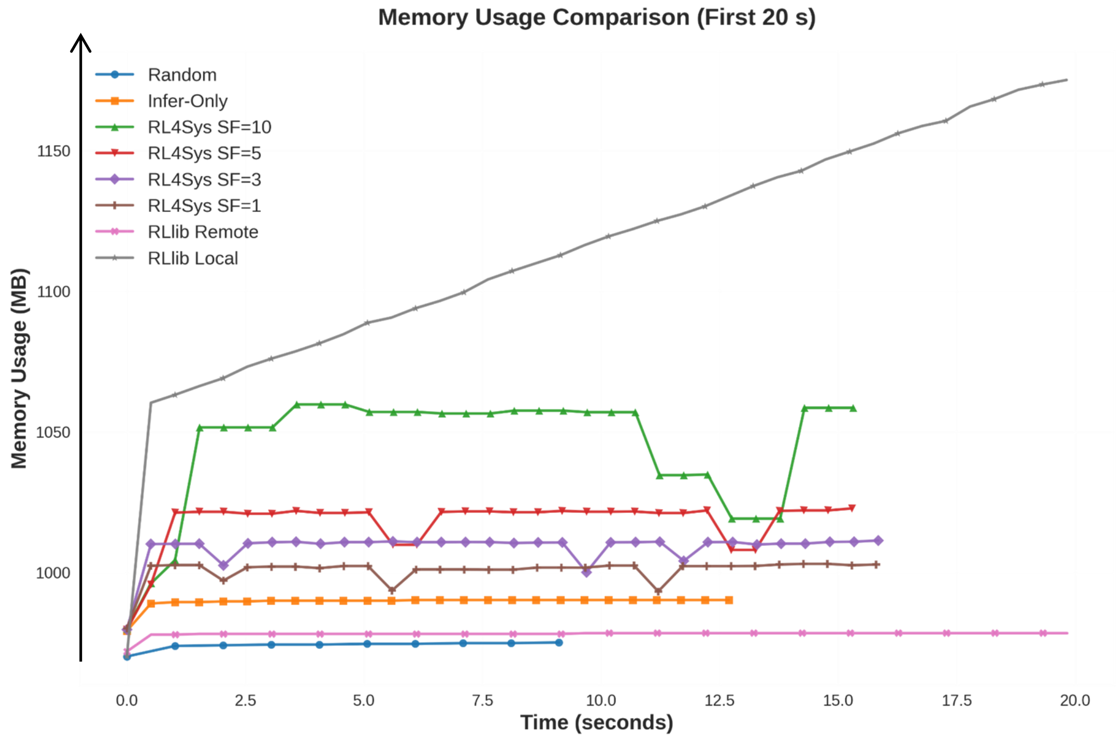


DIRLAB

Evaluation – Job Scheduling (Resource)

- RL4Sys CPU usage is around **3.2–3.3% usage per core** which is very close to Random/Static NN policy and about **5× lower** than RLlib Local's **16.5%**
- RL4Sys RAM usage **2–7% above** the baseline (depend on send frequency) Whereas RLlib usage **20+% higher** memory consumption for Local mode.

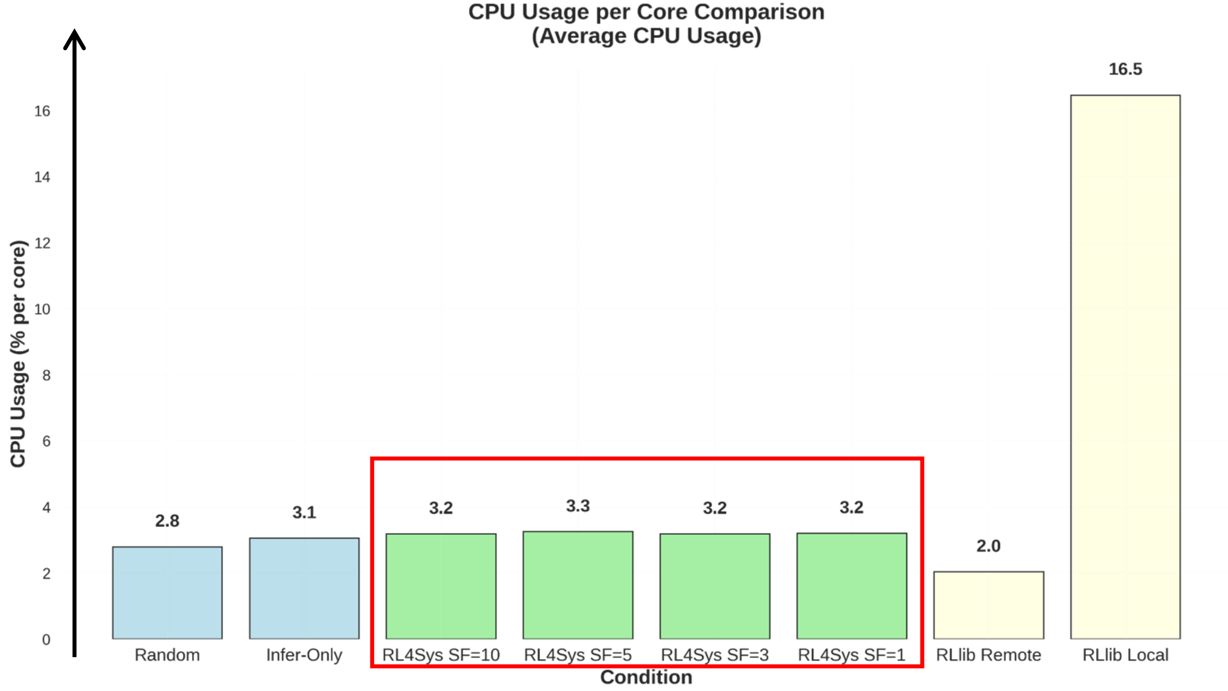


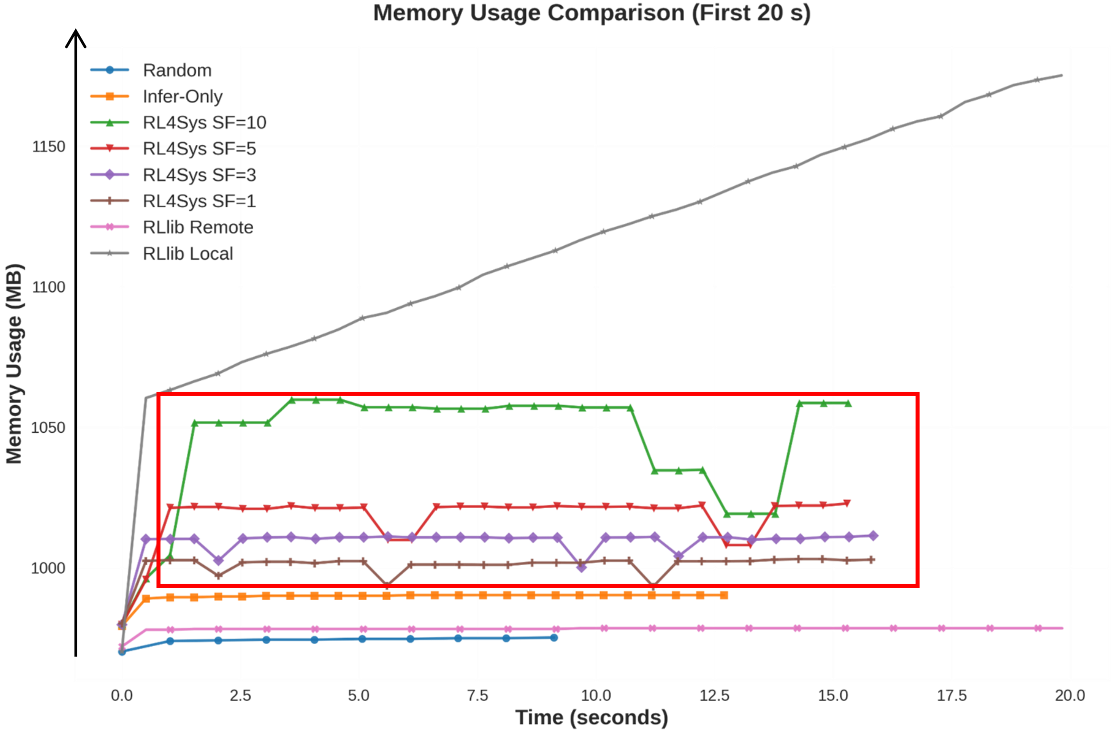


DIRLAB

Evaluation – Job Scheduling (Resource)

- RL4Sys CPU usage is around **3.2–3.3% per core** which is very close to Random/Infer-Only and about **5× lower** than RLlib Local’s **16.5%**
- RL4Sys RAM usage **2–7% above** the baseline (depend on send frequency) Whereas RLlib usage **20+% higher** memory consumption for RLlib Local.

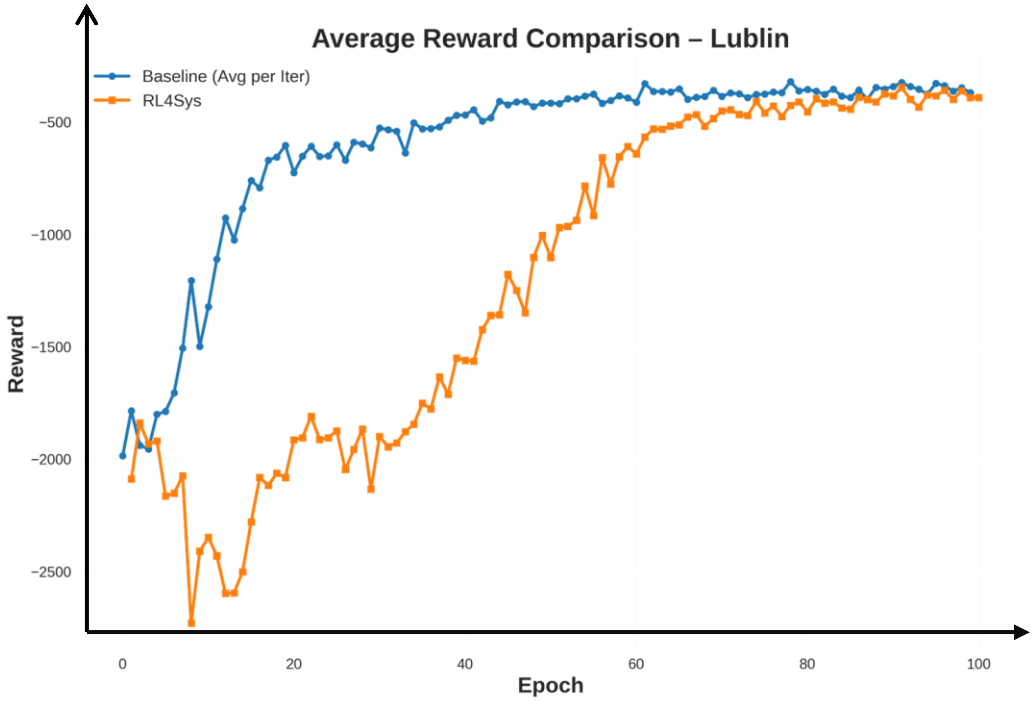


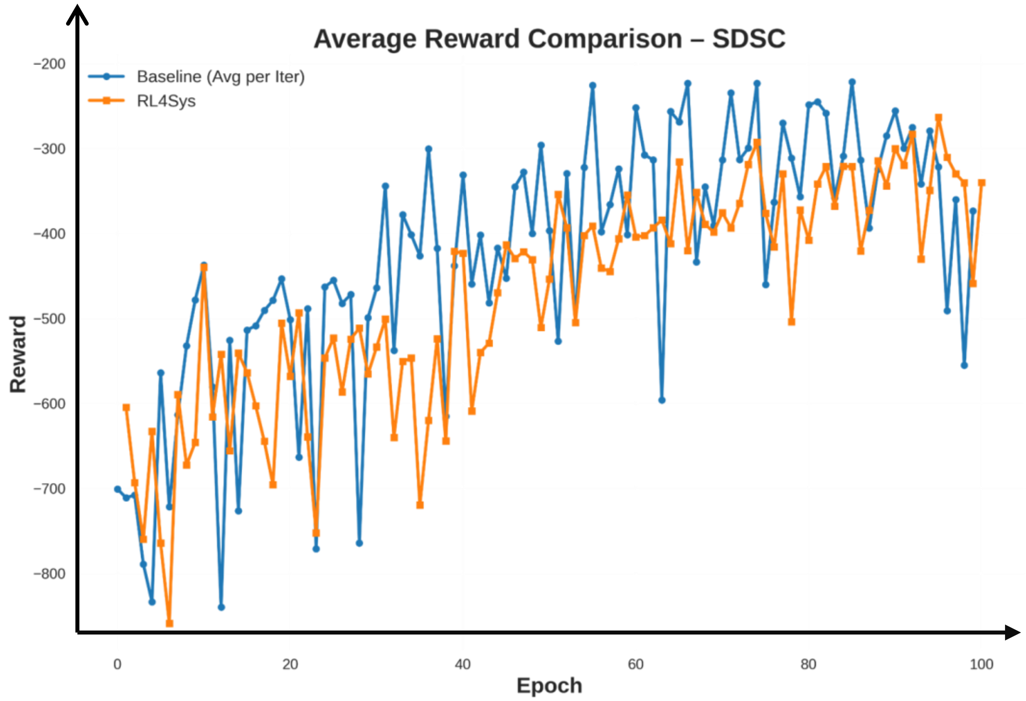


DIRLAB

Evaluation – Job Scheduling (Correctness)

We use RL4Sys in Job Scheduler on SDSC SP2 and Lublin256.
The reward curve is compared with the baseline solution proposed in Zhang *et al.*, 2020





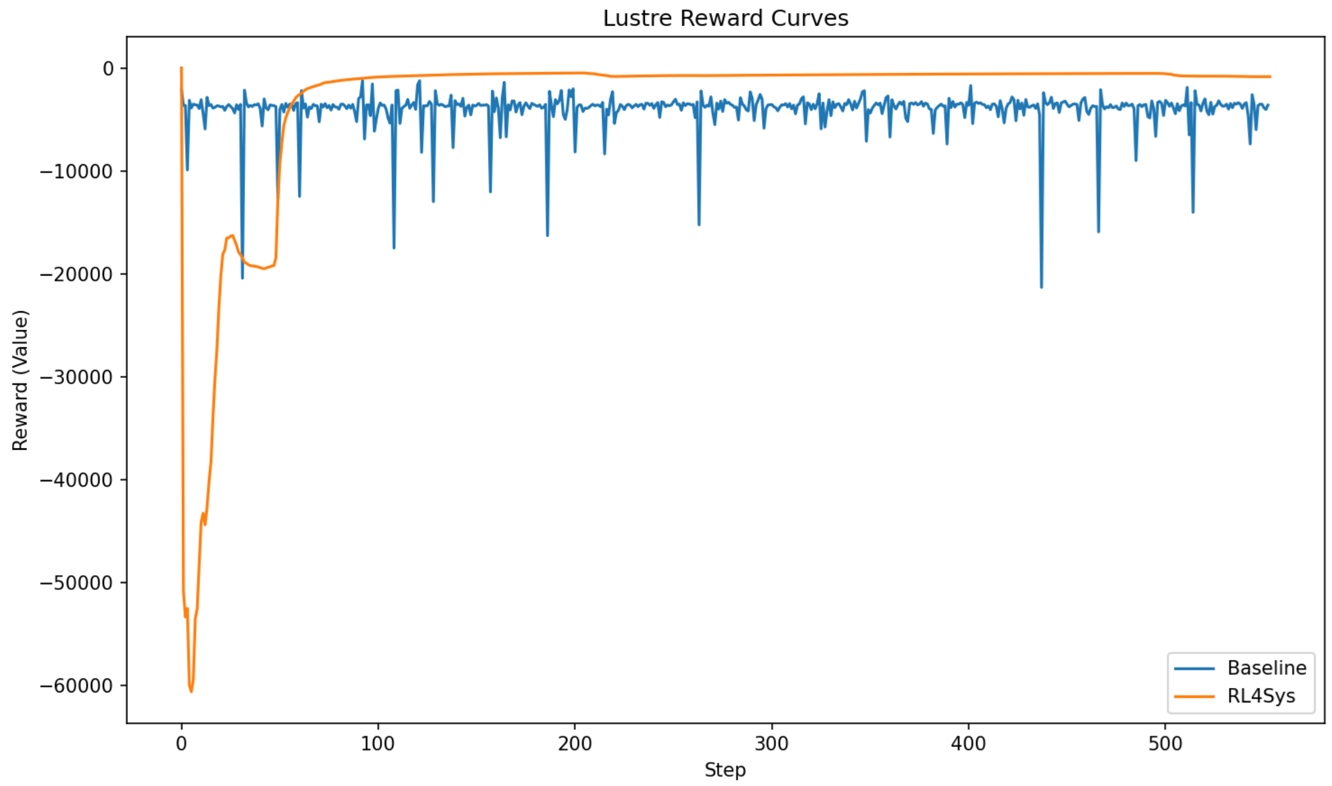
Zhang, D., Dai, D., He, Y., Bao, F. S., & Xie, B. (2020). **RLScheduler: an automated HPC batch job scheduler using reinforcement learning.** In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 1–15). IEEE.

DIRLAB

Evaluation – Lustre Optimization (Correctness)

- **Parameters Tuned:** We select **max payload size per RPC**, and **max parallel RPC number**

- **Baselines:** We compare default tuning with 2 on each parameters against RL4Sys dynamic tuning.



DIRLAB

Conclusion

- We presented **RL4Sys**, an **easy and practical** RL framework for real system.
- Our design *exceed* state-of-the-art RLlib frameworks performance on **System Driven Paradigm**.
- We proved RL4Sys delivered a **correct output with a large performance gains with two examples**

https://github.com/DIR-LAB/RL4Sys.git

Questions

DIRLAB