A Customizable Reinforcement Learning Environment for Semiconductor Fab Simulation

Benjamin Kovács, Pierre Tassel, Martin Gebser, Georg Seidel

Singapore, December 11-14 2022.

Contents

1. Introduction
2. Background
3. The Simulator
4. The Reinforcement Learning (RL) Framework
5. Integration with RL Algorithms
6. Conclusion

This work if a result of a collaborative project between the **University of Klagenfurt** and **Infineon Technologies Austria**.

Introduction

- Goal: optimise operations of semiconductor fabs

Dispatching

- Rules created by field experts
- Optimal scaling properties
- No information about solution quality

Simulation

- Fast, cost effective
- Helps to
  - analyse effects of
    - factory upgrades
    - policy changes
    - breakdowns
    - critical decisions
  - compare methods

Planning

- Modelling effort by field experts
- Optimal solution
- Scaling issues for large-scale instances

Background

- Open datasets available (e.g. SMT2020[1])
- Simulations in industry: commercial software
- Research: small-scale (*toy*) problem with custom simulators *or* commercial software

reproducibility not possible with closed-source software licenses, versions

no customisation opportunities for commercial tools — involvement of developers required

difficulties in comparing novel methods, measuring scientific progress

arbitrary reference implementations

Goal of our research project

Develop a universal simulator for fab scheduling research from prototyping to large-scale simulations for various dispatching and scheduling strategies.

Current paper

Introduction of our Reinforcement Learning Framework.

The SMT2020 Dataset

- 4 instances
  - high volume — low mix
  - low volume — high mix
  - + extensions with development lots

- Scale of datasets
  - 107 machine families
  - 1 300 + machines
  - 40 000 lots (for 2-year period)
  - 4 to 10 products
  - 300 to 600 steps / product

The Simulator

PySCFabSim[2]: open-source, scalable, customisable simulator in Python
https://github.com/prosysscience/PySCFabSim-release

open-source	custom integrations	reproducible
pre-defined interfaces	full data access	verified
scalable	super fast	multiple datasets
reentrant flow	batch machines	cascade machines
breakdowns	dedications	sequence-dependent setups

Validation & Performance

- Validation
  - Comparison to SMT2020 dataset reference results ✅
- Performance
  - example: 2 years of operation
    - 40 000 lots, average 500 steps / lot => 10 000 000 operations
    - simulated in 20 minutes
  - usable for machine learning methods with high sample complexity, parallelisation requirements
  - 4 seconds of startup time, 100-200 MB of memory usage / thread

Machine Learning for Dispatching

- Dispatching strategies
  - involves human expertise (engineering, experience)
  - optimality unknown
  - no automatic adaption to changing circumstances
- Dispatching with ML
  - higher upfront cross (engineering, training)
  - larger policy space
  - automatic adaption of policy to process changes

The RL Framework

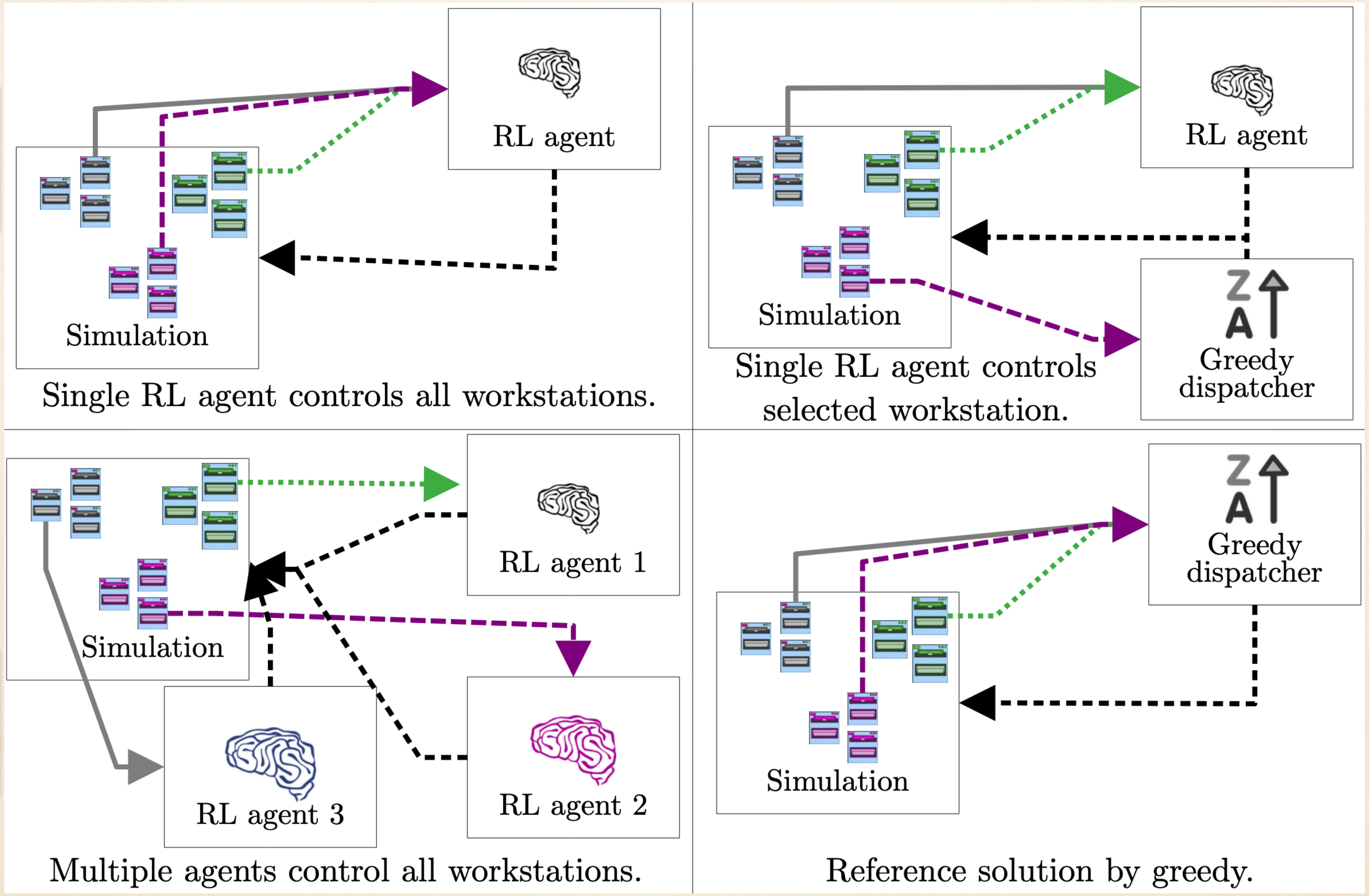
- (Why) Reinforcement Learning
  - learn from problem structure, instance characteristics
    - offline: pre-collected samples
    - online: live system
    - transfer: simulations
  - real-time dispatching

- Our RL toolbox
  - Customisable *gym* interface for the introduced simulator
    - Action
    - Observation space
    - Reward
  - „Plug-and-play“ environment
  - Single- and multi-agent configuration options
  - Partially observable

Operation Modes: Single- or Multi-Agent

Full RL control

Partial RL control



Full RL control

Heuristic control

Architecture of the RL-Framework

Main program

1. Create RL environment
2. Instantiate simulation
3. Start simulation

Simulator

4. Simulate events
5. Stop at decision point

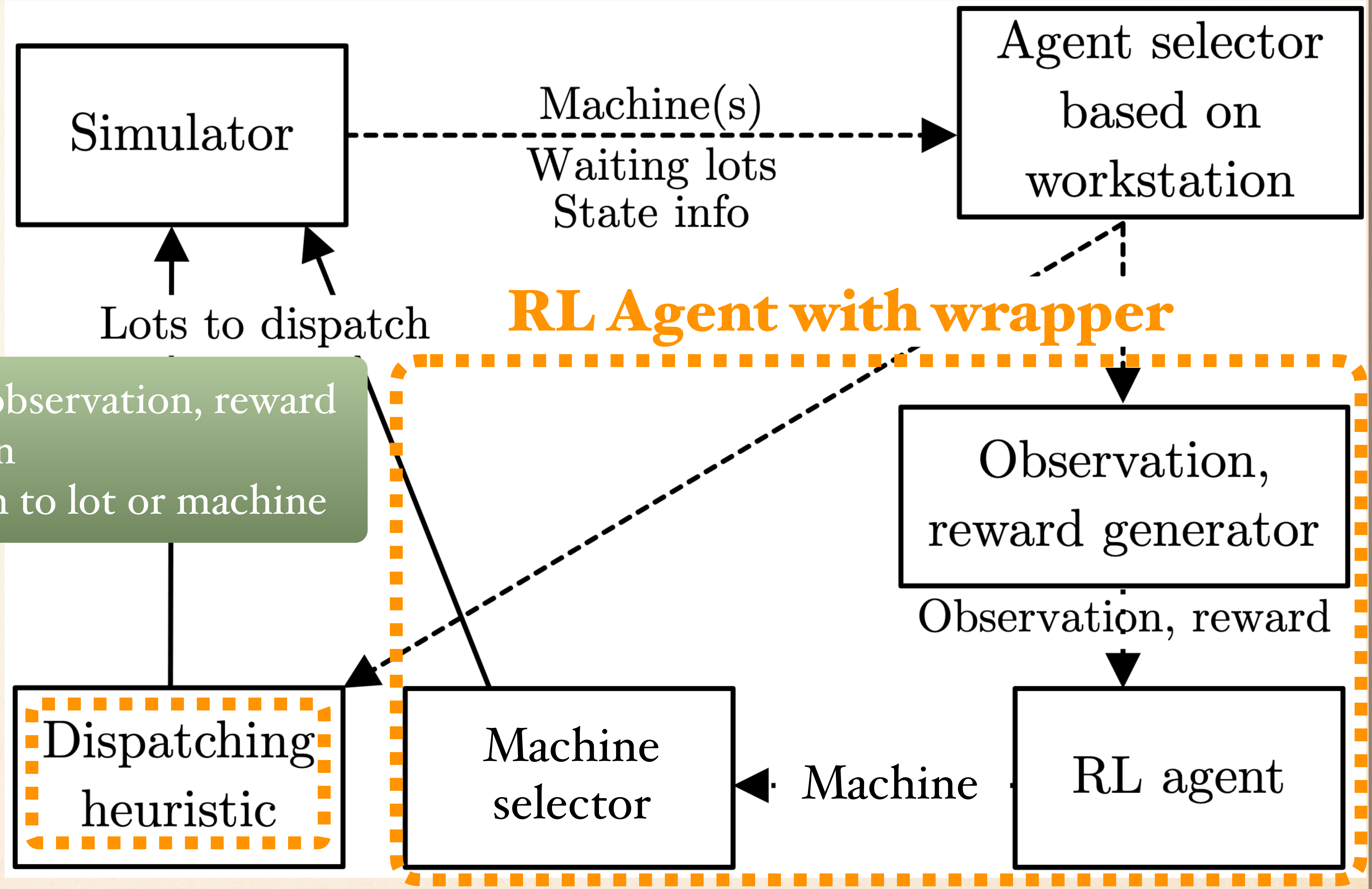
RL framework

6. Select agent (RL / heuristic)
7. Find lot or machine based on agent

Sim

8. Execute selection

1. Generate observation, reward
2. Pick Action
3. Map action to lot or machine



Action spaces

- 2 decision points
  - lot available => assign to machine
  - machine available => find lot
- 4 available action spaces
  - direct lot / machine selection
  - queue creation
  - heuristic selection

Actions

Pick operation for available machine[1]
Assign lot to machine's queue[2]
Pick lot from for available machine[3]
Select heuristic for available machine[4]

Observation Space

- Construct observation space from pre-defined features
    - single / multi features
    - some limited to specific action space
- Define own features based on custom data collection (plugins)

Observation space components

Machine	Global	Operation type / Lot
Next maintenance[1234]	No of WIP lots[1234]	Available lot count for op.[1234]
Setup/processing ratio[1234]	Average utilization[1234]	Maximum batch size[1234]
Idle/utilized ratio[1234]	No of lot types / routes[1234]	Batch utilization[1234]
Class (batch/cascade/...)[1234]		Steps remaining after op.[1234]
Cost of current setup[1234]		Lot timeliness[1234]
Bottleneck factor[1234]		Lot idle time[1234]
Queue length[2]		Lot priority[1234]
Machine family size[1234]		Processing time[1234]
Count of possible setups[1234]		Fit before maintenance[1234]
Time of setups[1234]		Setup time[1234]
Count of op. types[1234]		Rel. machine performance[1234]
		Remaining coupling time[1234]

Reward Function

- Define scalar function using pre-defined reward components
    - Sparse and dense functions
- Add custom components based on data collection plugins

Reward components

Dense	Sparse
Timeliness of lots	Lot completion
No of WIP lots	Coupling violation
Utilization	

Summary: Observation & Action Spaces, Reward

Actions

Pick operation for available machine[1]
Assign lot to machine's queue[2]
Pick lot from for available machine[3]
Select heuristic for available machine[4]

Reward components

Dense

Timeliness of lots
No of WIP lots
Utilization

Sparse

Lot completion
Coupling violation

Observation space components

Machine

Next maintenance[1234]
Setup/processing ratio[1234]
Idle/utilized ratio[1234]
Class (batch/cascade/...)[1234]
Cost of current setup[1234]
Bottleneck factor[1234]
Queue length[2]
Machine family size[1234]
Count of possible setups[1234]
Time of setups[1234]
Count of op. types[1234]

Global

No of WIP lots[1234]
Average utilization[1234]
No of lot types / routes[1234]

Operation type / Lot

Available lot count for op.[1234]
Maximum batch size[1234]
Batch utilization[1234]
Steps remaining after op.[1234]
Lot timeliness[1234]
Lot idle time[1234]
Lot priority[1234]
Processing time[1234]
Fit before maintenance[1234]
Setup time[1234]
Rel. machine performance[1234]
Remaining coupling time[1234]

Instantiation of Environments

```python
from rl.env.action_choose_rule_for_machine import ChooseRuleForMachine
from rl.env.actions import SingleObservationFeature
from rl.env.agent import RLAgent, GreedyAgent
from rl.env.environment import DynamicSCFabSimEnv
from rl.env.reward import Reward
from simulation.plugins.wandb_plugin import WandBPlugin

R = Reward
O2 = ChooseRuleForMachine.Observation
P = GreedyAgent.Policy
DEMO_ENV_2 = lambda max_steps=100000000, max_days=730: DynamicSCFabSimEnv(
    action=ChooseRuleForMachine(
        alternatives=[P.CriticalRatio, P.FIFODeadline, P.FIFOWaiting, P.AvoidSetup, P.HotLotFirst, P.CombinedFIFO, ],
        observation_space=[O2.Machine.cascading, O2.Machine.bottleneck_factor, O2.Machine.setup_last_at,
                           O2.Machine.setup_last_cost, O2.Machine.idle_processing_ratio, O2.Machine.next_maintenance,
                           SingleObservationFeature(lambda instance, **kwargs: len(instance.done_lots), True), ], ),
    agents=[
        RLAgent(idx=0, machine_groups=['Diffusion']), RLAgent(idx=1, machine_families=['DefMet_BE_33', 'DefMet_BE_42']),
        GreedyAgent(policy=GreedyAgent.Policy.CombinedCR),
    ],
    simulator_params=dict(plugins=[WandBPlugin()], run_to=3600 * 24 * max_days, ), dataset='SMT2020_LVHM',
    reward=5 * R.Dense.LotWipCount() + R.Dense.LotTimeliness() + 20 * R.Sparse.LotCompletion(), max_steps=max_steps, )
```

Future Work

- Development of optimised RL agents for the environment
- Analyse adaptivity of agents to evolving factories
- Integration of datasets with physical fabs, transfer learning from current dataset to industrial ones

Time for Questions

Thank You for Your Attention

Download PySCFabSim at
https://github.com/prosysscience/PySCFabSim-release

This work was partially funded by KWF project 28472, cms electronics GmbH, FunderMax GmbH, Hirsch Armbänder GmbH, incubed IT GmbH, Infineon Technologies Austria AG, Isovolta AG, Kostwein Holding GmbH, and Privatstiftung Kärntner Sparkasse.

Contact
Benjamin Kovács
University of Klagenfurt
Benjamin.Kovacs@aau.at

References

(1) D. Kopp, M. Hassoun, A. Kalir and L. Mönch, "SMT2020—A Semiconductor Manufacturing Testbed," in *IEEE Transactions on Semiconductor Manufacturing*, vol. 33, no. 4, pp. 522-531, Nov. 2020, doi: 10.1109/TSM.2020.3001933.

(2) B. Kovács, P. Tassel, R. Ali, M. El-Kholany, M. Gebser and G. Seidel, "A Customizable Simulator for Artificial Intelligence Research to Schedule Semiconductor Fabs," *2022 33rd Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, 2022, pp. 1-6, doi: 10.1109/ASMC54647.2022.9792520.