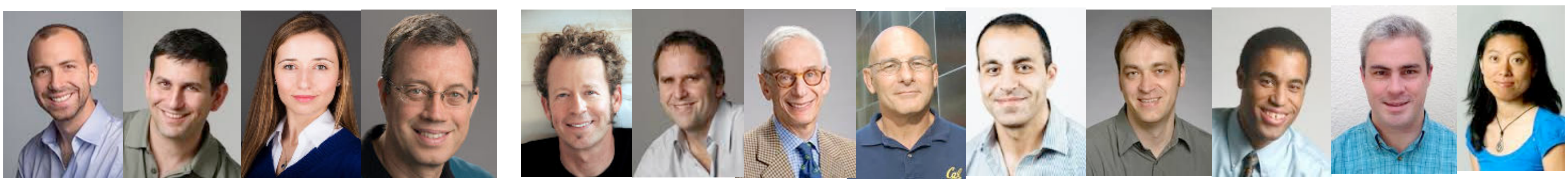
RISELab

(Real-time Intelligent Secure Execution)



Ion Stoica

ScaledML

March 25, 2017

RISELab

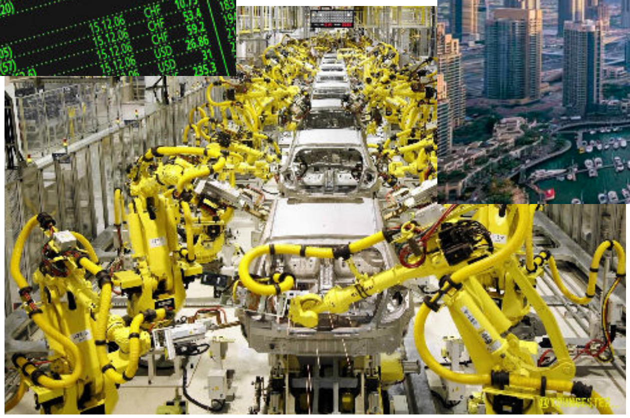
From live data to real-time decisions

AMPLab

From batch data to advanced analytics

Why?

Data only as valuable as the decisions (actions) it enables



Why?

Data only as valuable as the decisions (actions) it enables

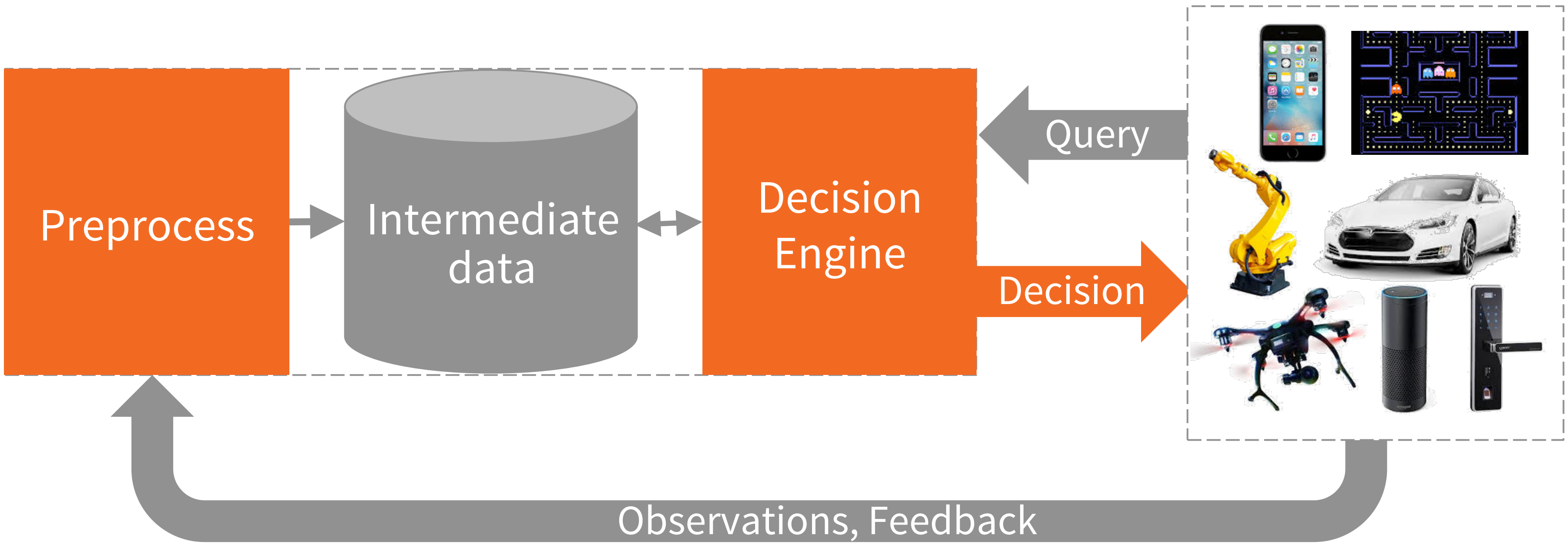
What is a good decision?

- Faster decisions better than slower decisions
- Decisions on fresh data better than decisions on stale data
- Decisions on personalized data better than on generic data

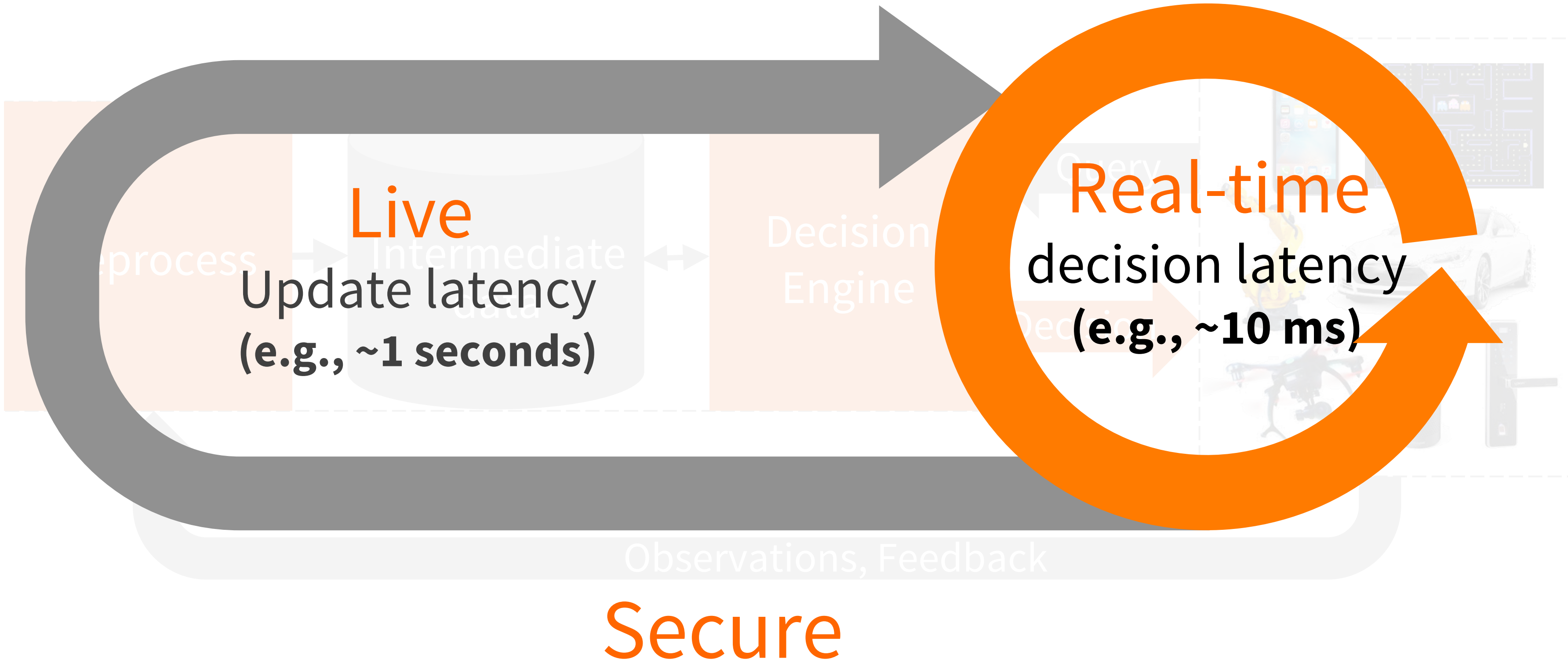
What we want?

Real-time decisions	*decide in ms*
on live data	*the current state of the environment*
with strong security	*privacy, confidentiality, integrity*

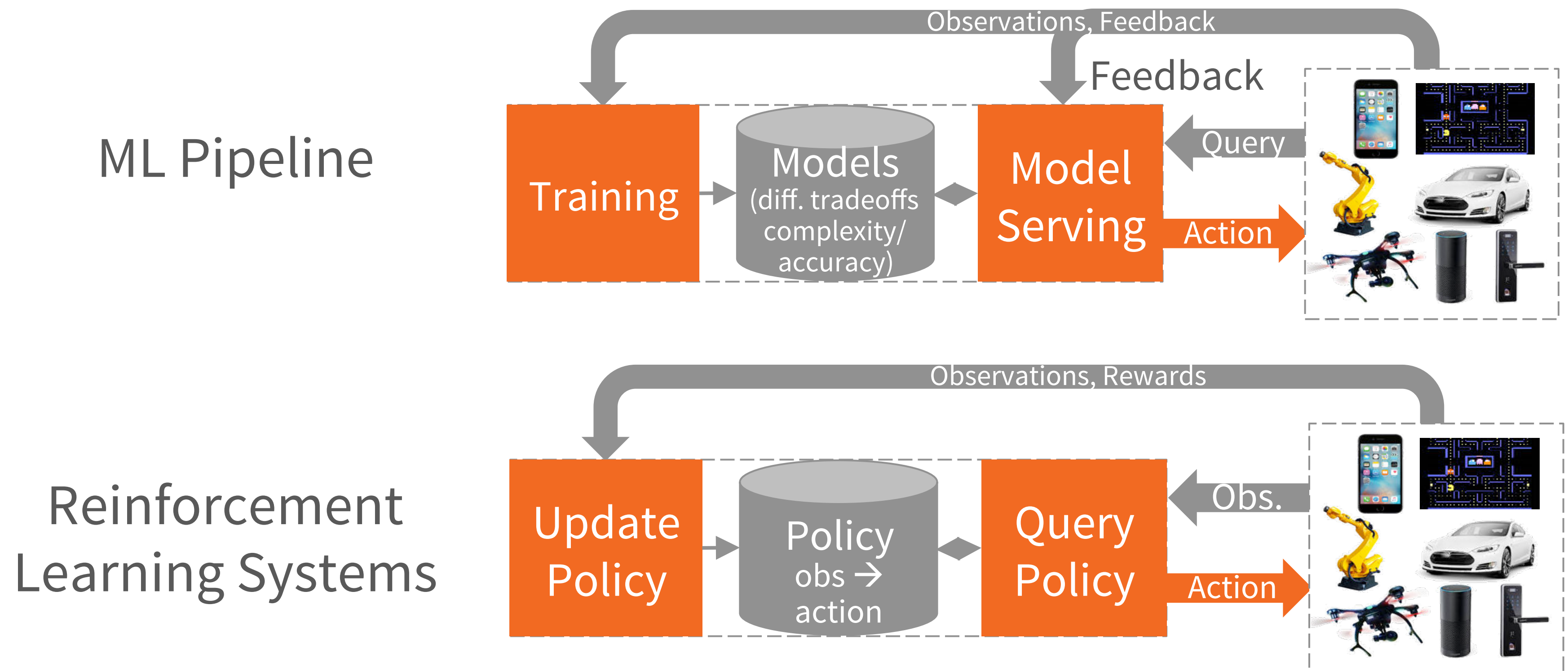
Typical decision system



Typical decision system



Example of decision systems



What else do we want from decisions?

Intelligent: complex decisions in uncertain environments

What else do we want from decisions?

Intelligent: complex decisions in uncertain environments

Robust: handle complex noise

What else do we want from decisions?

Intelligent: complex decisions in uncertain environments

Robust: handle complex noise, failures, unforeseen inputs

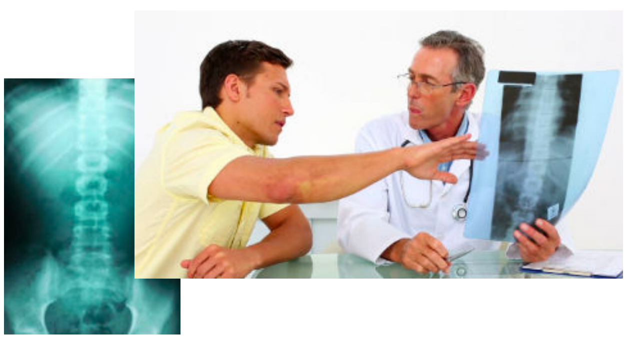
Need ability to say "I don't know!"

What else do we want from decisions?

Intelligent: complex decisions in uncertain environments

Robust: handle complex noise, failures, unforeseen inputs

Explainable: ability to explain non-obvious decisions



RISELab goal

Develop open source platforms, tools, and algorithms for intelligent real-time decisions on live-data

Some research directions

Secure Real-time Decisions Stack (SRDS)

- Open source platform to develop of RISE like apps
- Reinforcement Learning (RL) as one of key app patterns
- Secure from ground up

Learning control hierarchies: speedup learning, training

Shared learning: learn over confidential data

Some research directions

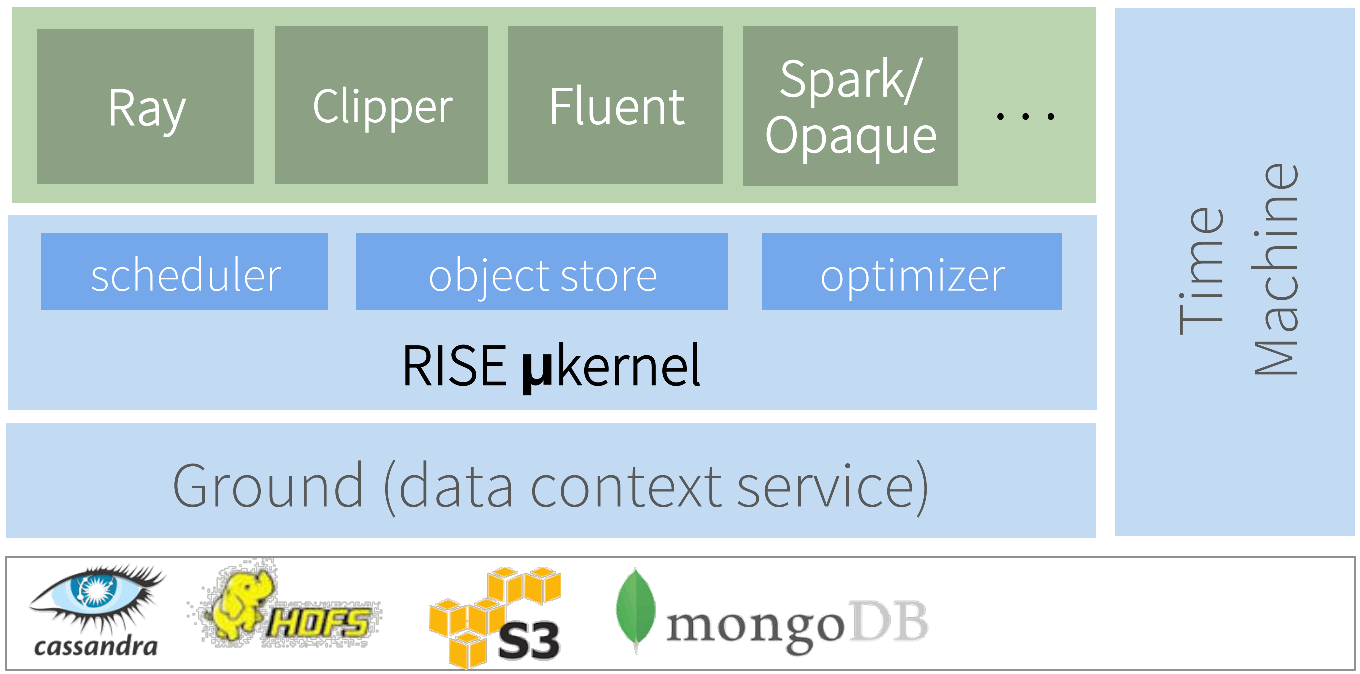
Secure Real-time Decisions Stack (SRDS)

- Open source platform to develop of RISE like apps
- Reinforcement Learning (RL) as one of key app patterns
- Secure from ground up

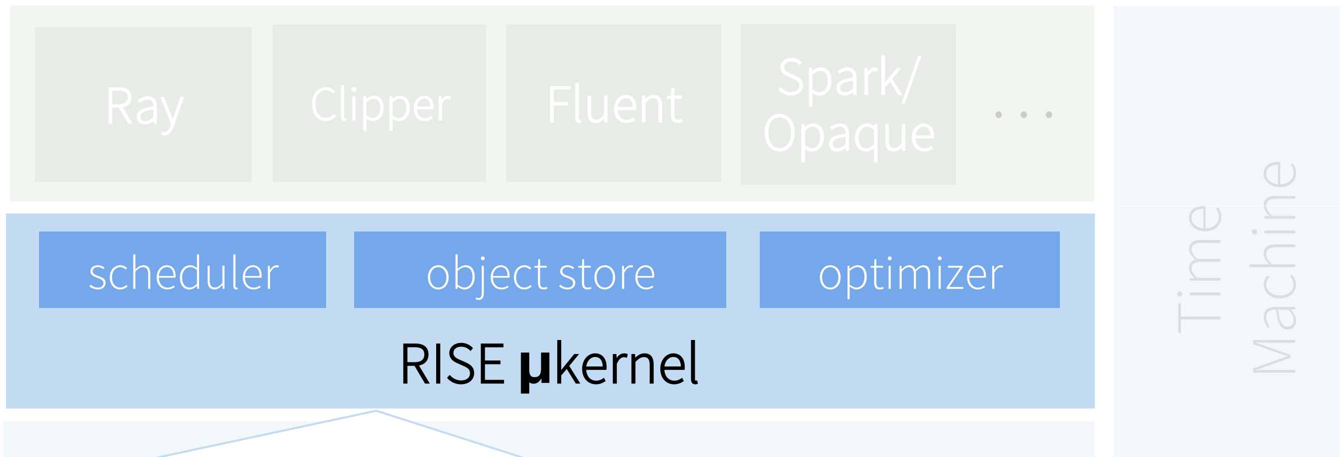
Learning control hierarchies: speedup learning, training

Shared learning: learn over confidential data

Secure Real-time Decision Stack (SRDS)



SRDS: Microkernel

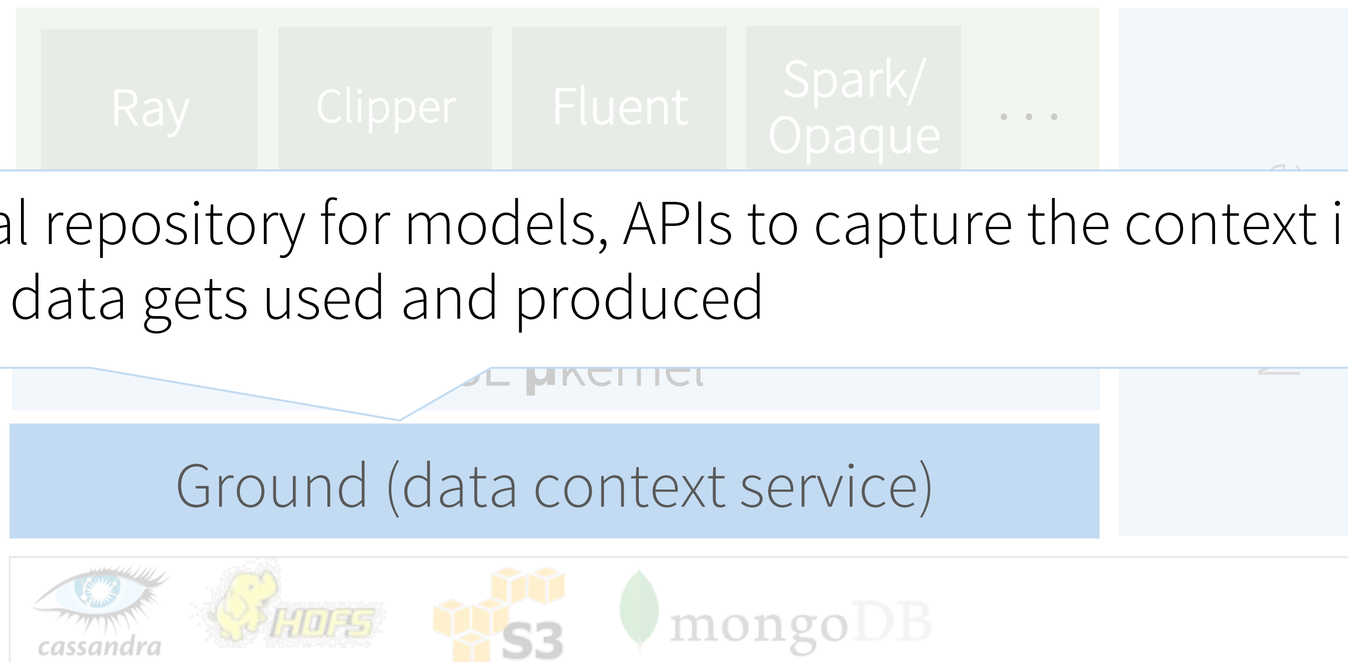


Minimalist execution engine:

- Support both data flow and task-parallel execution models
- High-throughput, low-latency scheduler

SRDS: Ground

Central repository for models, APIs to capture the context in which data gets used and produced



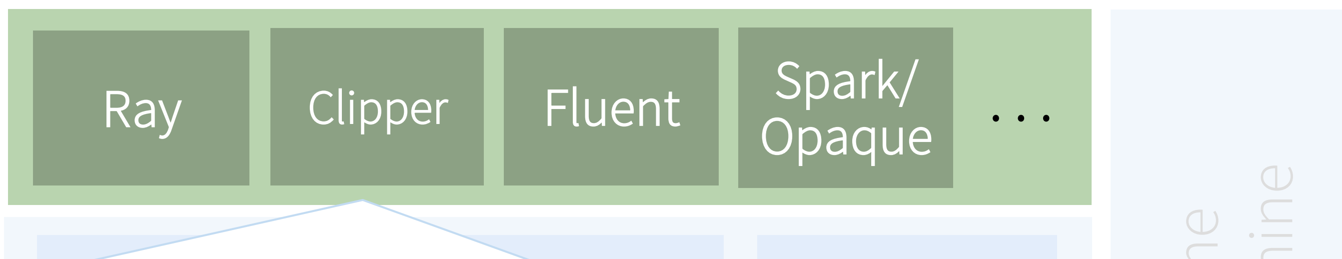
SRDS: Time machine

Replaying of apps at fine granularity

- Simplify development, debugging
- **Robustness**: replay against perturbed inputs
- **Explainability**: identify inputs causing decision
- **Security**: confirm vulnerabilities, test security patches, compliance auditing

Time Machine

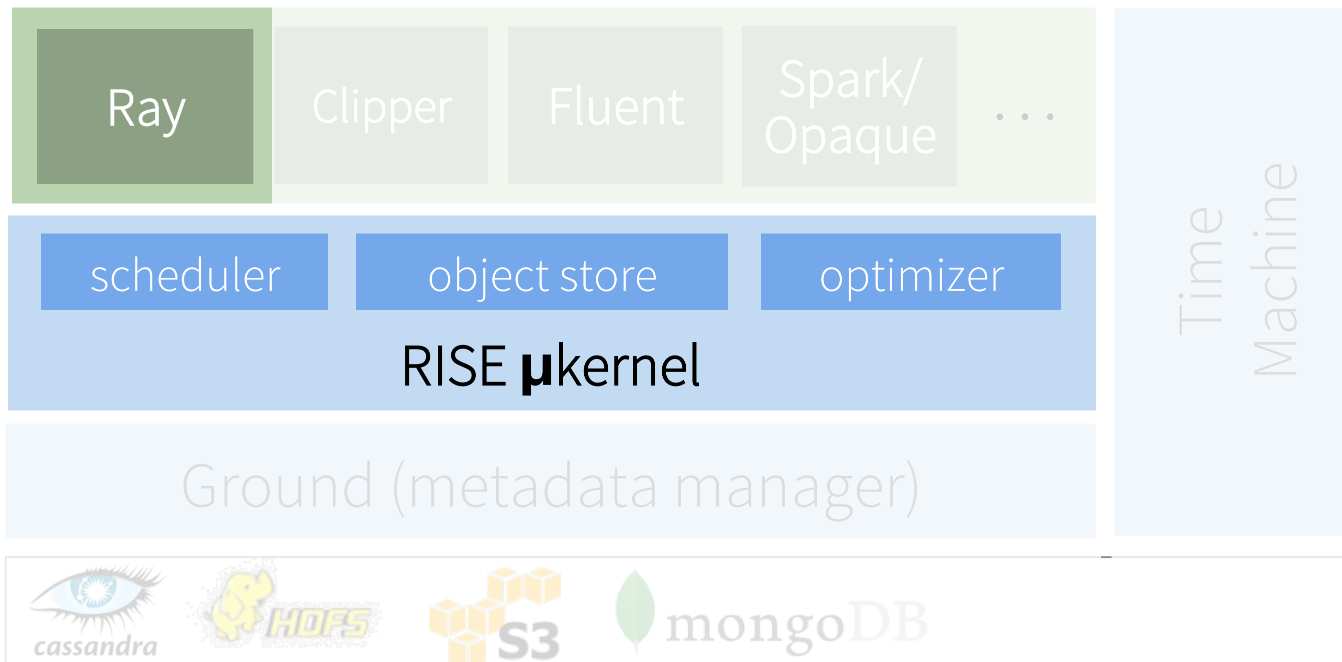
SRDS: Application frameworks



Computation frameworks to simplify development of RISE apps

- **Ray**: task-parallel framework to support RL workloads
- **Clipper**: model serving supporting ensembles & cascading models
- **Fluent**: fine grained data flow execution framework
- **Opaque**: secure SparkSQL

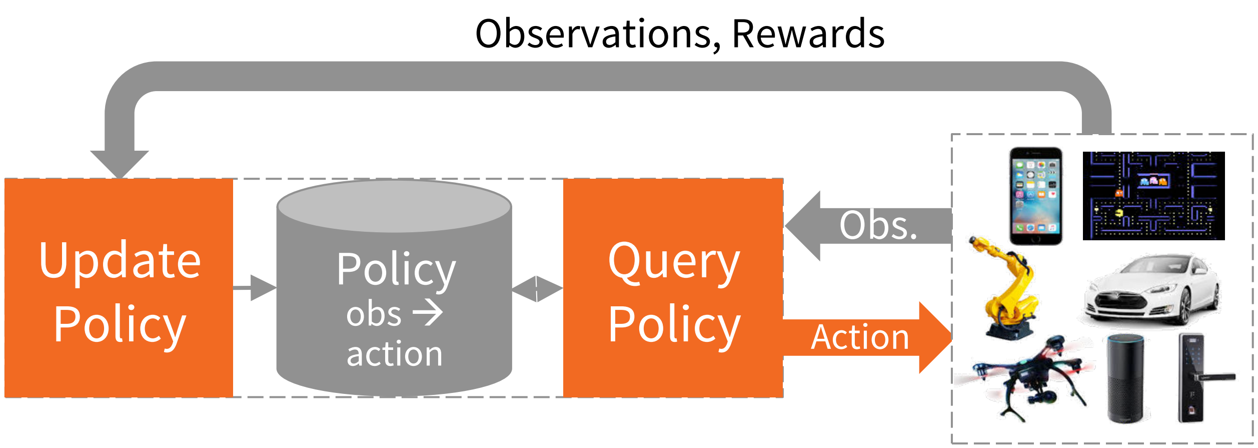
SRDS: Application frameworks



Ray

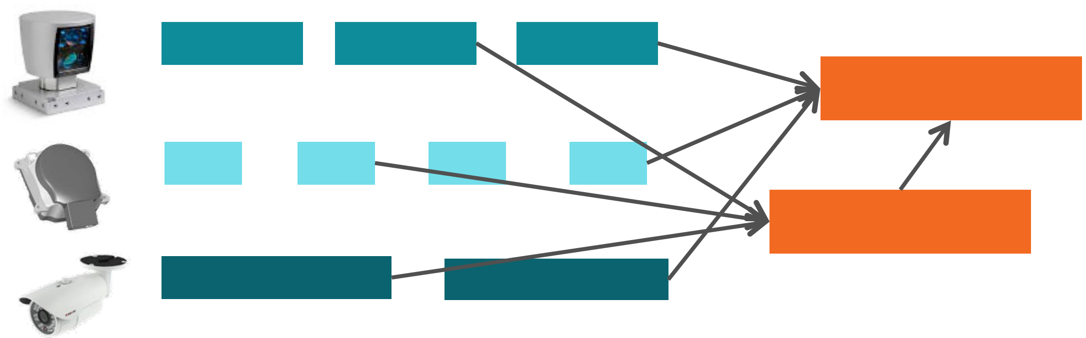
Targets Reinforcement Learning (RL) applications

Currently includes μkernel functionality



RL requirements

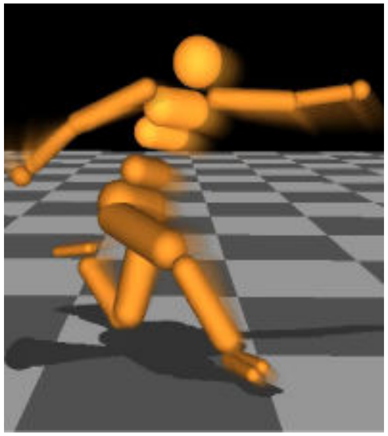
Process inputs from different sensors in parallel & real-time

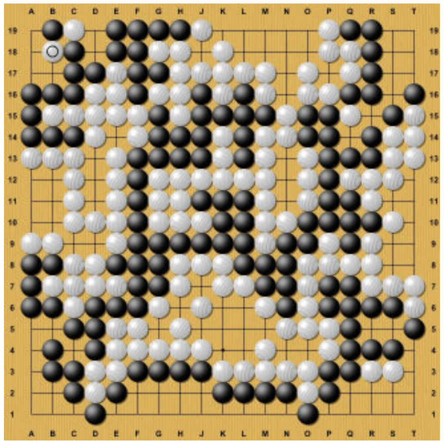


RL requirements

Process inputs from different sensor in parallel & real-time

Execute large number of rollouts (simulations)



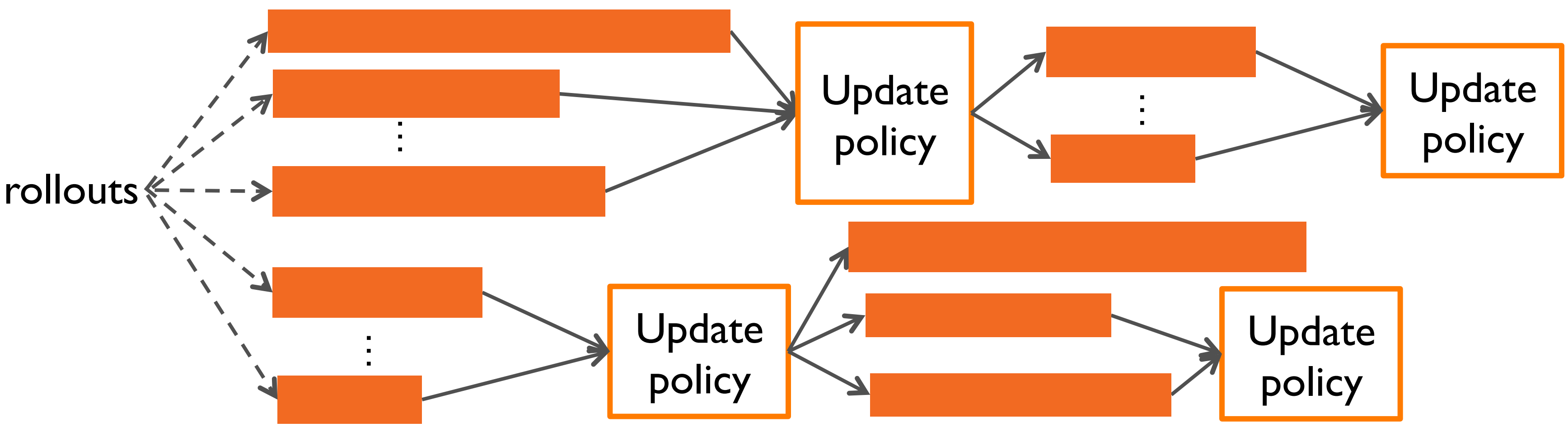


RL requirements

Process inputs from different sensor in parallel & real-time

Execute large number of rollouts (simulations)

Rollouts outcomes are used to update policy (e.g., SGD)



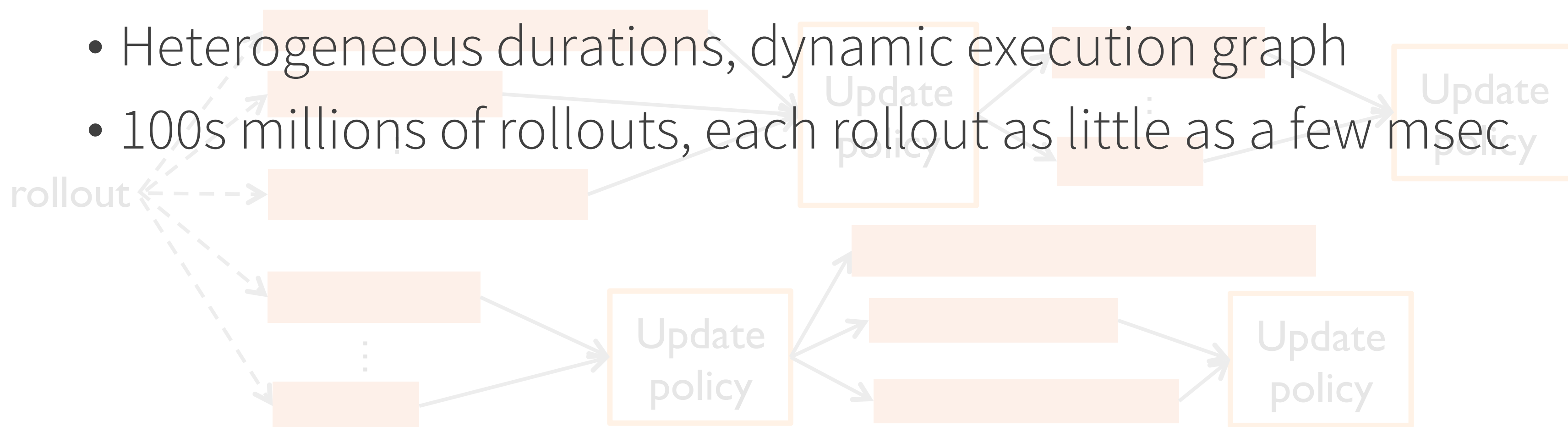
RL requirements

Process inputs from different sensor in parallel & real-time

Execute large number of rollouts (simulations)

Rollouts outcomes are used to update policy (e.g., SGD)

- Heterogeneous durations, dynamic execution graph
- 100s millions of rollouts, each rollout as little as a few msec



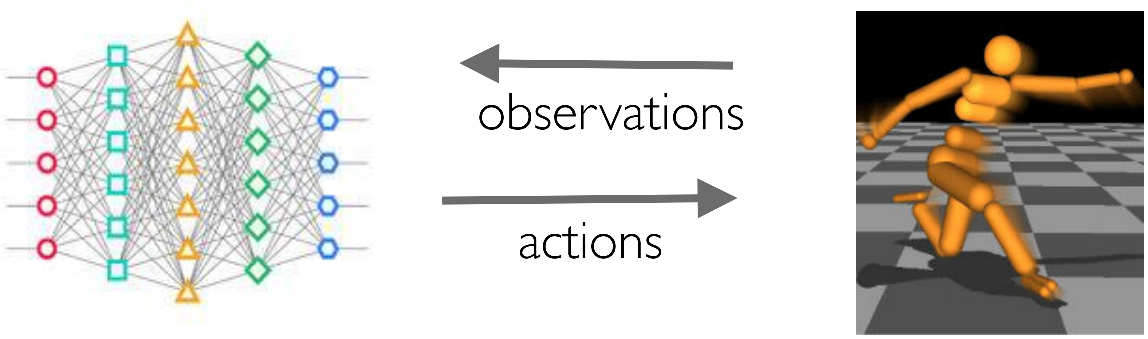
RL requirements

Process inputs from different sensor in parallel & real-time

Execute large number of rollouts (simulations)

Rollouts outcomes are used to update policy (e.g., SGD)

Often policies implemented by DNNs



Ray goals

Flexibility

- Combine neural networks, planning, search, simulation, etc
- Heterogeneous tasks: CPUs/GPUs, durations, computation
- Fine-grained data and task dependencies, dynamic execution

Performance

- Millions of tasks per second with msec level latencies
- Adapt to changing work in real-time

Easy of use

- Minimal changes to parallelize existing Python serial code

Ray architecture

Centralized control store

- Stateless components

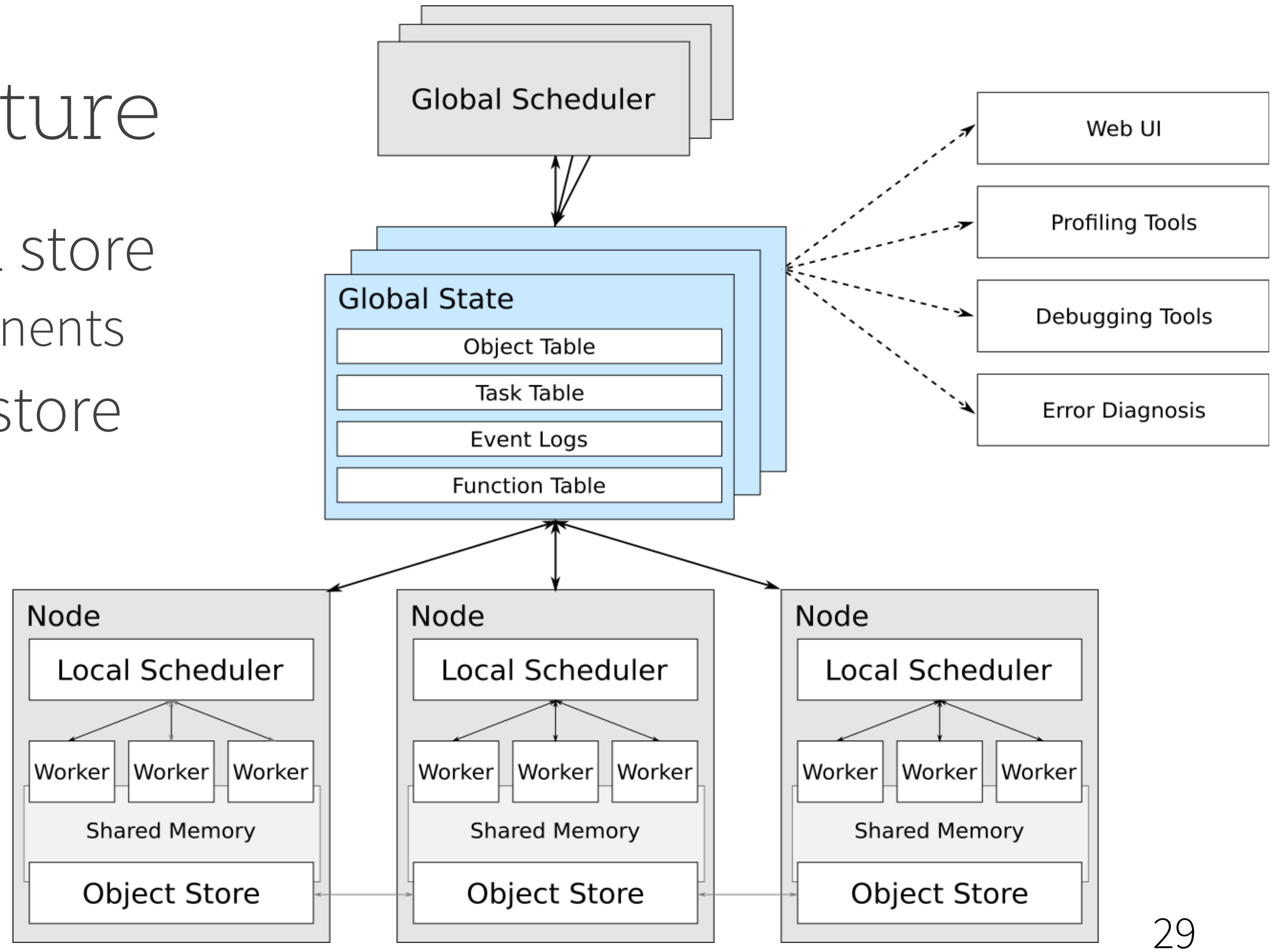
In-memory object store

- Leverage Arrow

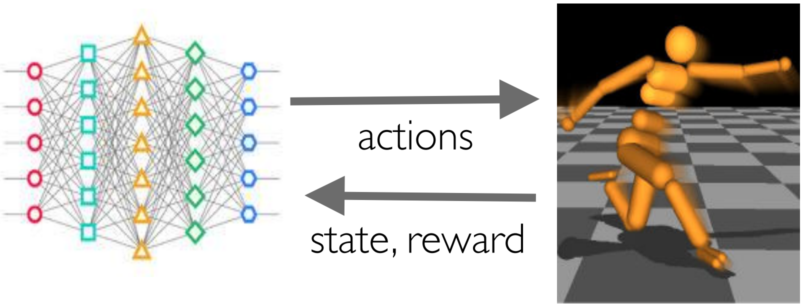
Dist. scheduling

Prototype

- Python bindings
- C/C++ backend

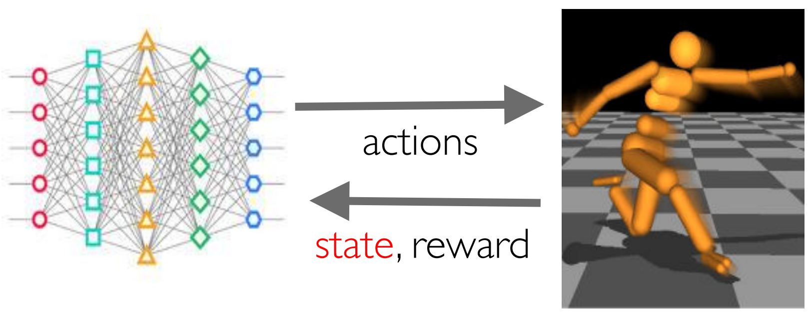


Example: Asynchronous RL with Ray



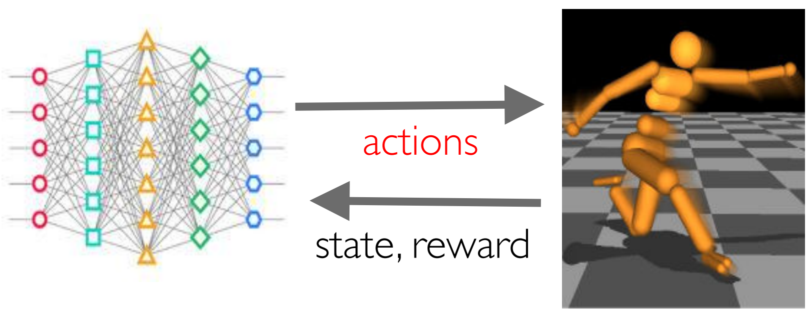

```
@ray.remote
def simulation(policy):
  trajectory = []
  state = simulator.initialize()
  for i in range(T):
    action = policy.compute(state)
    state, reward = simulator.step(action)
    trajectory.append((state, reward))
  return trajectory
```

Example: Asynchronous RL with Ray



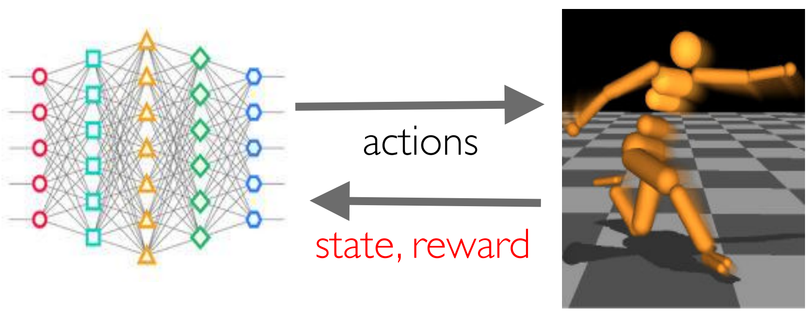

```python
@ray.remote
def simulation(policy):
  trajectory = []
  state = simulator.initialize()
  for i in range(T):
    action = policy.compute(state)
    state, reward = simulator.step(action)
    trajectory.append((state, reward))
  return trajectory
```

Example: Asynchronous RL with Ray



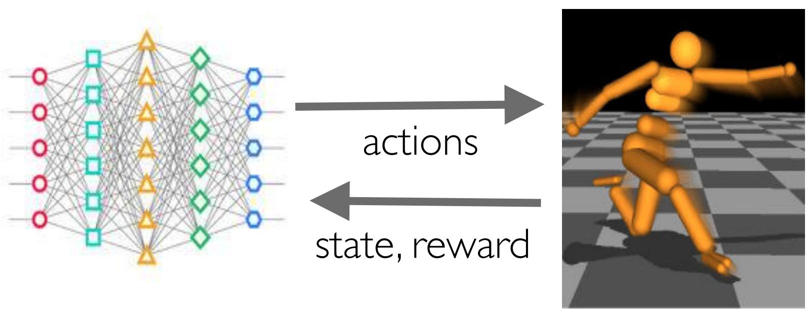

```
@ray.remote
def simulation(policy):
  trajectory = []
  state = simulator.initialize()
  for i in range(T):
    action = policy.compute(state)
    state, reward = simulator.step(action)
    trajectory.append((state, reward))
  return trajectory
```

Example: Asynchronous RL with Ray



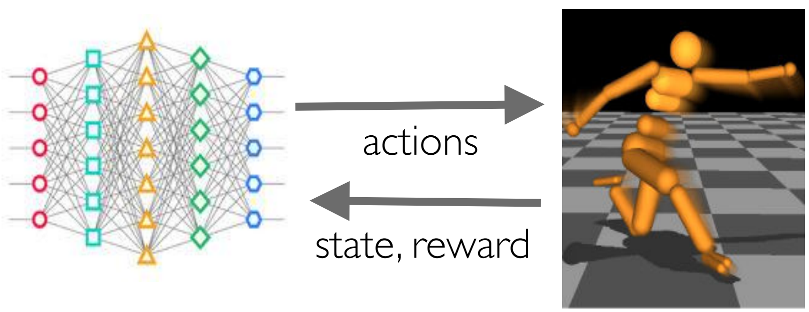

```
@ray.remote
def simulation(policy):
  trajectory = []
  state = simulator.initialize()
  for i in range(T):
    action = policy.compute(state)
    state, reward = simulator.step(action)
    trajectory.append((state, reward))
  return trajectory
```

Example: Asynchronous RL with Ray




```
@ray.remote
def simulation(policy):
  trajectory = []
  state = simulator.initialize()
  for i in range(T):
    action = policy.compute(state)
    state, reward = simulator.step(action)
    trajectory.append((state, reward))
  return trajectory
```

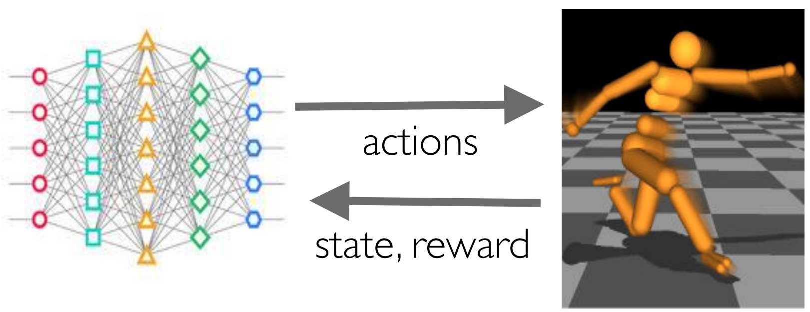
Example: Asynchronous RL with Ray




```
@ray.remote
def simulation(policy):
  trajectory = []
  state = simulator.initialize()
  for i in range(T):
    action = policy.compute(state)
    state, reward = simulator.step(action)
    trajectory.append((state, reward))
  return trajectory
```

```
# One step of the algorithm
trajectories = [simulation.remote(policy) for _ in range(10000)]
while True:
  # Wait for next trajectory to become ready
  trajectory, trajectories = ray.wait(trajectories)
  policy.update(ray.get(trajectory)) # Update model
  # Start new simulation
  trajectories.append(simulation.remote(policy))
```

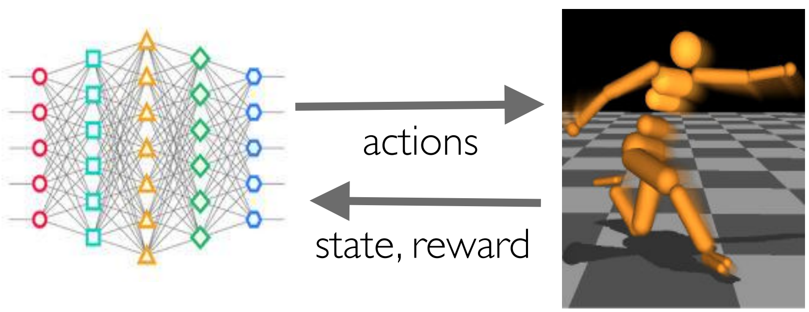
Example: Asynchronous RL with Ray




```
@ray.remote
def simulation(policy):
  trajectory = []
  state = simulator.initialize()
  for i in range(T):
    action = policy.compute(state)
    state, reward = simulator.step(action)
    trajectory.append((state, reward))
  return trajectory
```

```
# One step of the algorithm
trajectories = [simulation.remote(policy) for _ in range(10000)]
while True:
  # Wait for next trajectory to become ready
  trajectory, trajectories = ray.wait(trajectories)
  policy.update(ray.get(trajectory)) # Update model
  # Start new simulation
  trajectories.append(simulation.remote(policy))
```

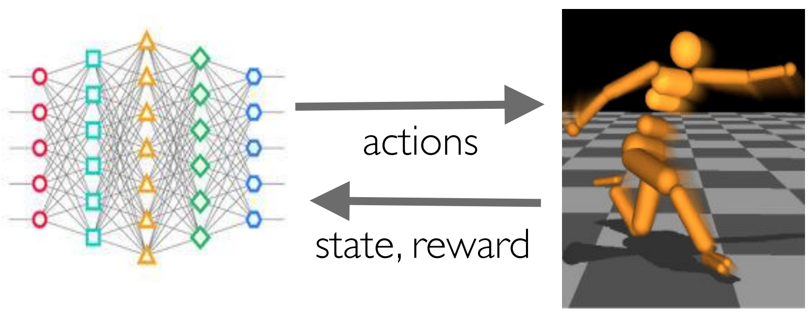
Example: Asynchronous RL with Ray




```
@ray.remote
def simulation(policy):
  trajectory = []
  state = simulator.initialize()
  for i in range(T):
    action = policy.compute(state)
    state, reward = simulator.step(action)
    trajectory.append((state, reward))
  return trajectory
```

```
# One step of the algorithm
trajectories = [simulation.remote(policy) for _ in range(10000)]
while True:
  # Wait for next trajectory to become ready
  trajectory, trajectories = ray.wait(trajectories)
  policy.update(ray.get(trajectory)) # Update model
  # Start new simulation
  trajectories.append(simulation.remote(policy))
```

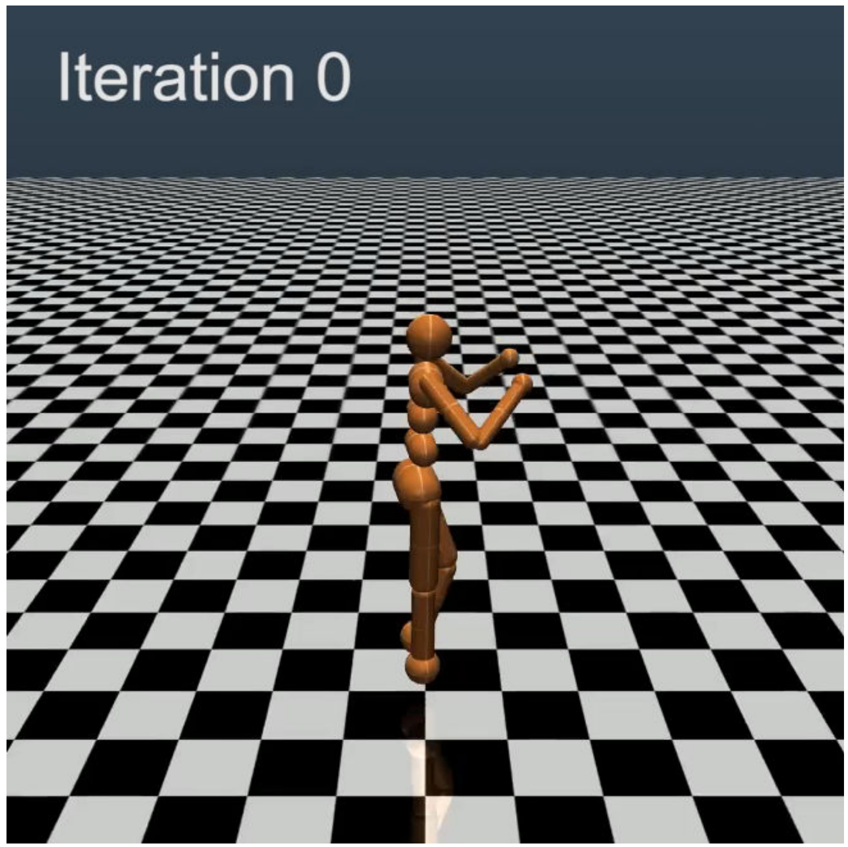
Example: Asynchronous RL with Ray




```
@ray.remote
def simulation(policy):
  trajectory = []
  state = simulator.initialize()
  for i in range(T):
    action = policy.compute(state)
    state, reward = simulator.step(action)
    trajectory.append((state, reward))
  return trajectory
```

```
# One step of the algorithm
trajectories = [simulation.remote(policy) for _ in range(10000)]
while True:
  # Wait for next trajectory to become ready
  trajectory, trajectories = ray.wait(trajectories)
  policy.update(ray.get(trajectory)) # Update model
  # Start new simulation
  trajectories.append(simulation.remote(policy))
```

Example: Learning to run



SRDS: Application frameworks

Ray | Clipper | Fluent | Spark/ Opaque | . . .

Flexibility

- Uniform interface across models
- Model life cycle management

Performance

- Prediction caching
- Ensembles, cascading models
- Control latency-accuracy tradeoff

Some research directions

Secure Real-time Decisions Stack (SRDS)

- Open source platform to develop of RISE like apps
- Reinforcement Learning (RL) as one of key app patterns
- Secure from ground up

Learning control hierarchies: speedup learning, training

Shared learning: learn over confidential data

Learning control hierarchies

Many apps consists of a sequence of decisions/actions, e.g.,

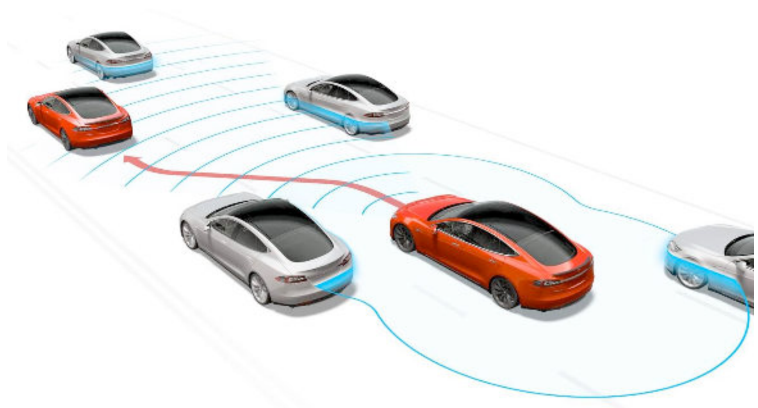
- Driving : turn wheel, step on gas or break, signal

Challenge: huge state and action spaces

- Expensive to learn and train

Approach: indentify sequence of actions, called options, e.g.,

- Driving: change lines



Learning control hierarchies

Many apps consists of a sequence of decisions/actions, e.g.,

- Driving : turn wheel, step on gas or break, signal

Challenge: huge solution space

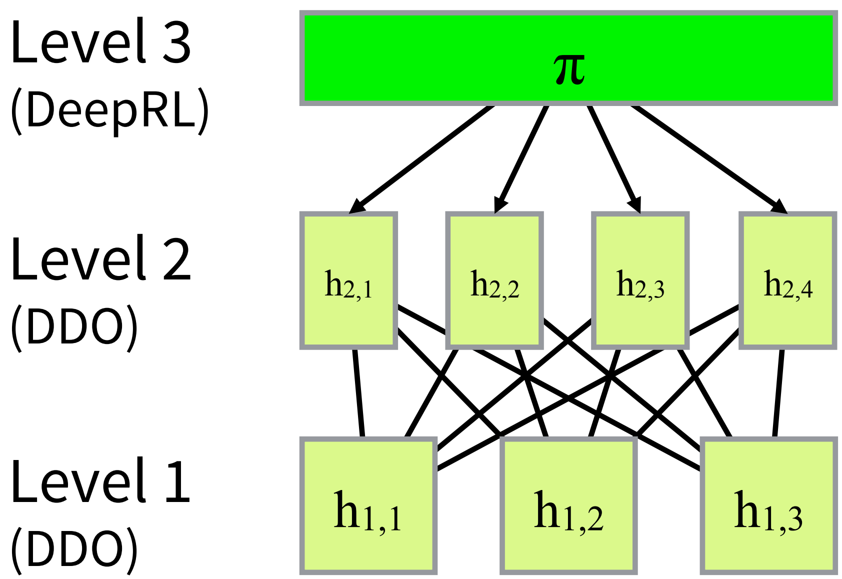
- Expensive to learn and train

Approach: indentify sequence of actions, called options, e.g.,

- Driving: change lines
- Advantage: reduce dramatically action space
  - Faster learning and generalization
- Most prior methods require human design, few end-to-end
- Our research: learn hierarchies of options **automatically**

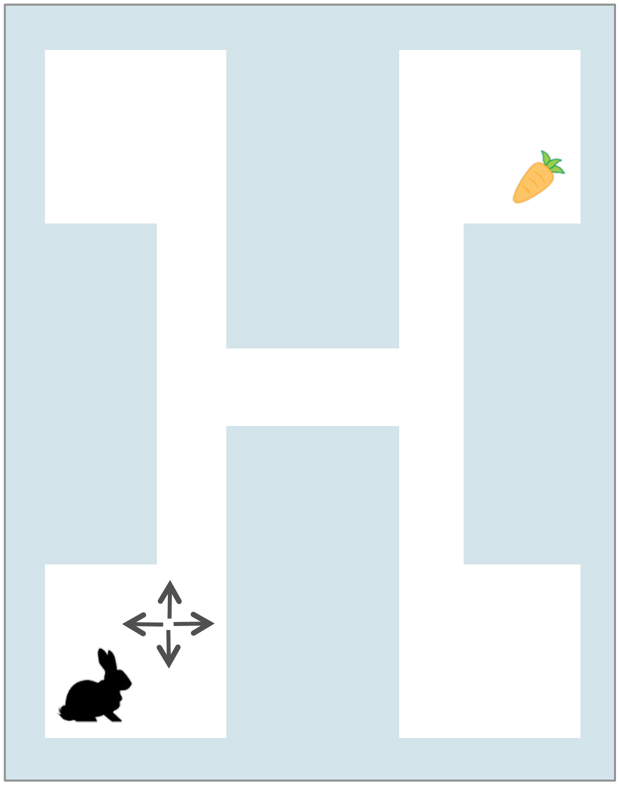
Discovery Deep Options (DDO)

Compute gradients with respect to policy parameters
Decouple levels and avoid joint training



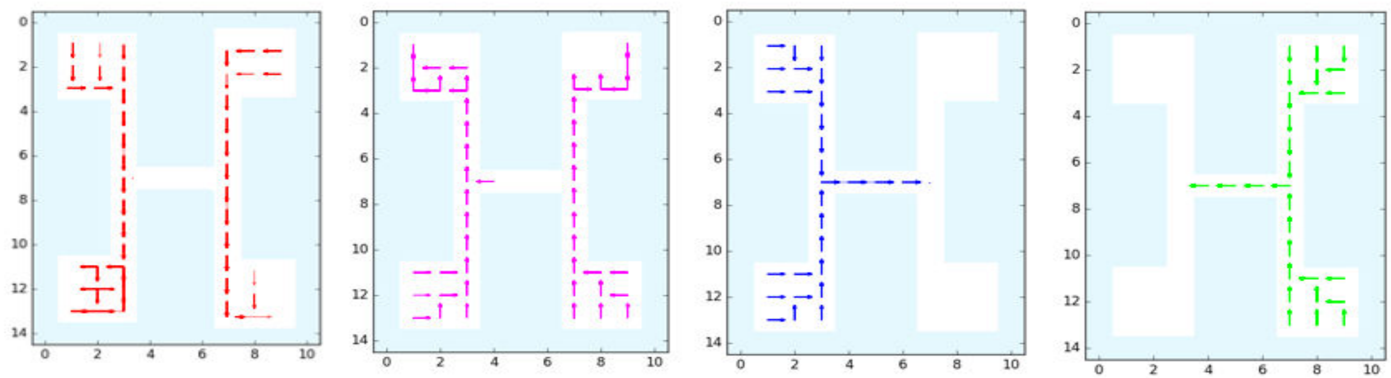
for $d = 1, \ldots, D-1$ **do**
  Initialize a set of options $\mathcal{H}_d = \{h_{d,1}, \ldots, h_{d,k_d}\}$
  DDO: train options $\langle \pi_h, \psi_h \rangle_{h \in \mathcal{H}_d}$ with η_d fixed
  Augment action space $\mathcal{A} \leftarrow \mathcal{A} \cup \mathcal{H}_d$
end for
Use RL algorithm to train high-level policy

Example: Four room maze example



Example: Four room maze example

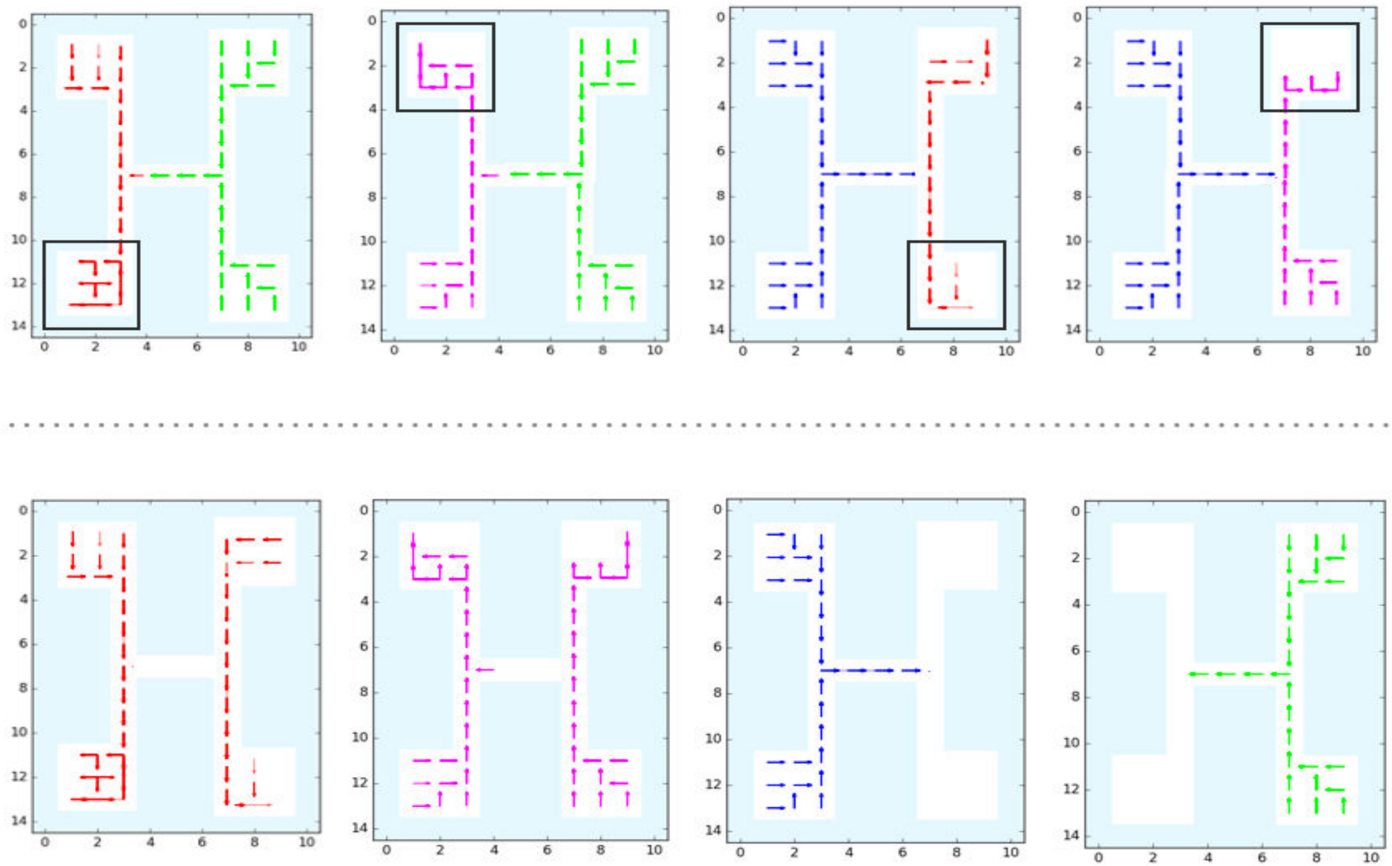
Level 1



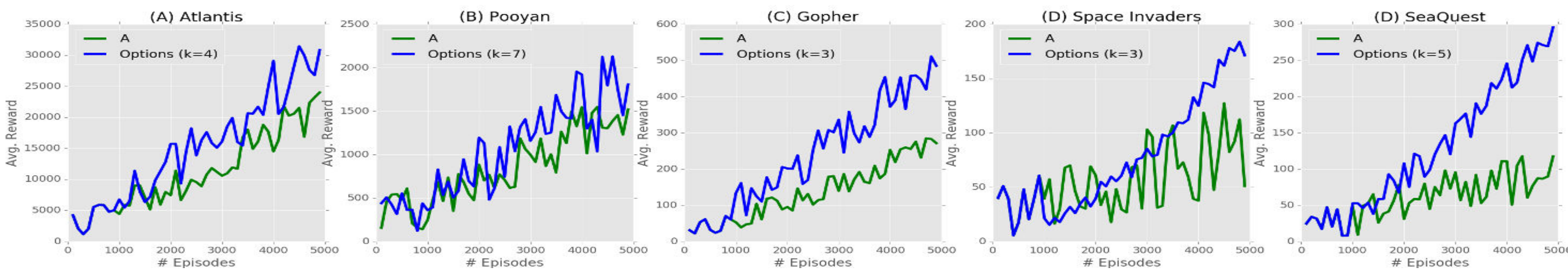
Example: Four room maze example

Level 2

Level 1



Results: Atari RAM



Results: SeaQuest RAM

Some research directions

Secure Real-time Decisions Stack (SRDS)

- Open source platform to develop of RISE like apps
- Reinforcement Learning (RL) as one of key app patterns
- Secure from ground up

Learning control hierarchies: speedup learning, training

Shared learning: learn over confidential data

Shared Learning

Every cloud provider wants to provide ML as a Service (MLaaS)

Cloud provider

ML as a Service (training & prediction)

Shared Learning

Every cloud provider wants to provide ML as a Service (MLaaS)

Every enterprise wants to get quality prediction on its data

Shared Learning

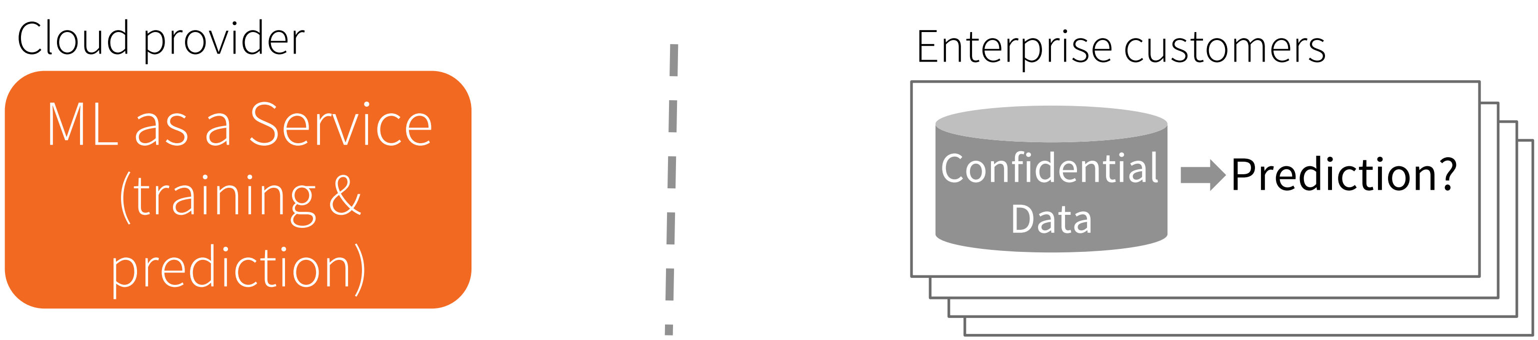
Every cloud provider wants to provide ML as a Service (MLaaS)

Every enterprise wants to get quality prediction on its data

Many enterprises wants to keep their data confidential

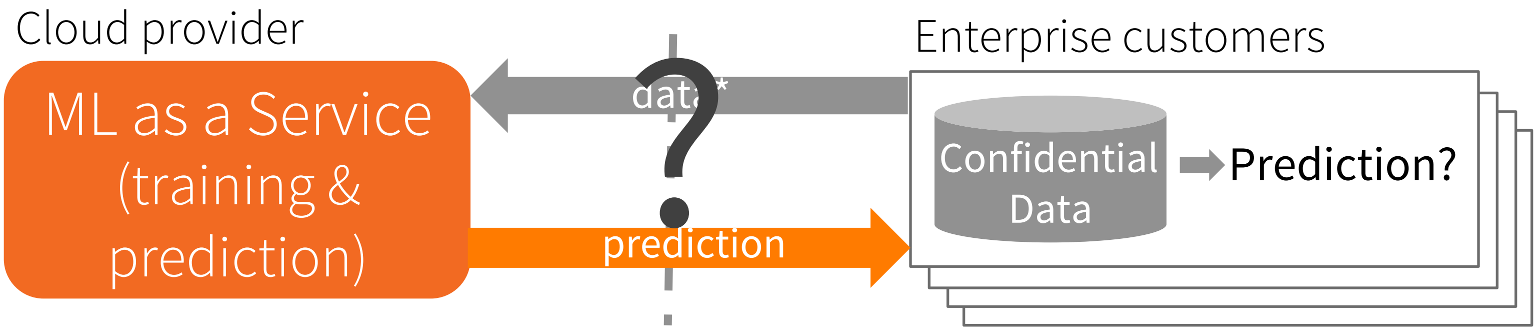
Shared Learning

Every cloud provider wants to provide ML as a Service (MLaaS)
Every enterprise wants to get quality prediction on its data
Many enterprises wants to keep their data confidential
Every cloud provider wants to learn across customers' data to improve pred.



Shared Learning

Every cloud provider wants to provide ML as a Service (MLaaS)
Every enterprise wants to get quality prediction on its data
Many enterprises wants to keep their data confidential
Every cloud provider wants to learn across customers' data to improve pred.



How can cloud providers learn across customers and perform predictions while preserving customer's data confidentiality?

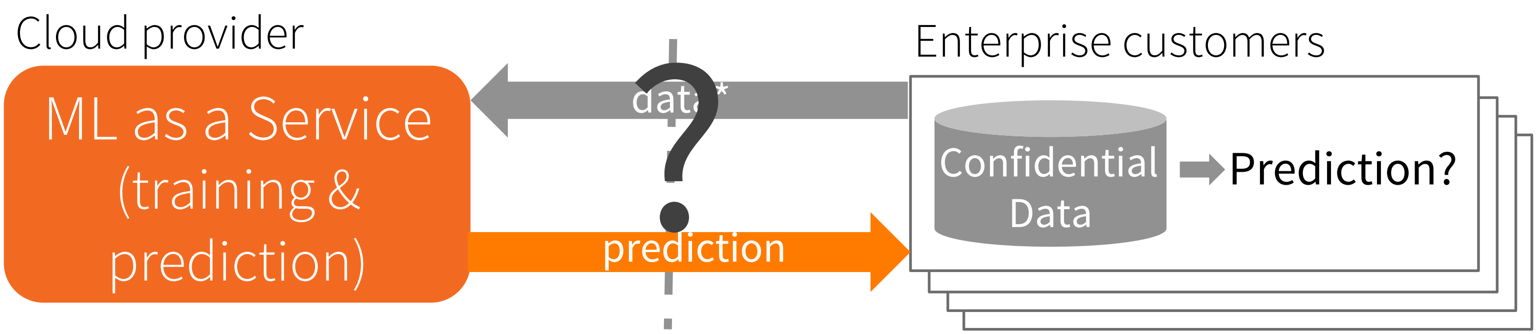
Shared Learning

Every cloud provider wants to provide ML as a Service (MLaaS)

Every enterprise wants to get quality prediction on its data

Many enterprises wants to keep their data confidential

Every cloud provider wants to learn across customers' data to improve pred.



How can cloud providers incentivize organizations to share data?

RISELab goal

Develop open source platforms, tools and algorithms for real-time decisions on live data with strong security

Many exciting challenges in ML/AI, systems, security, architectures

Thanks!