Lecture 13:

Simulating Virtual Worlds for Training

Visual Computing Systems
Stanford CS348K, Spring 2024

Today

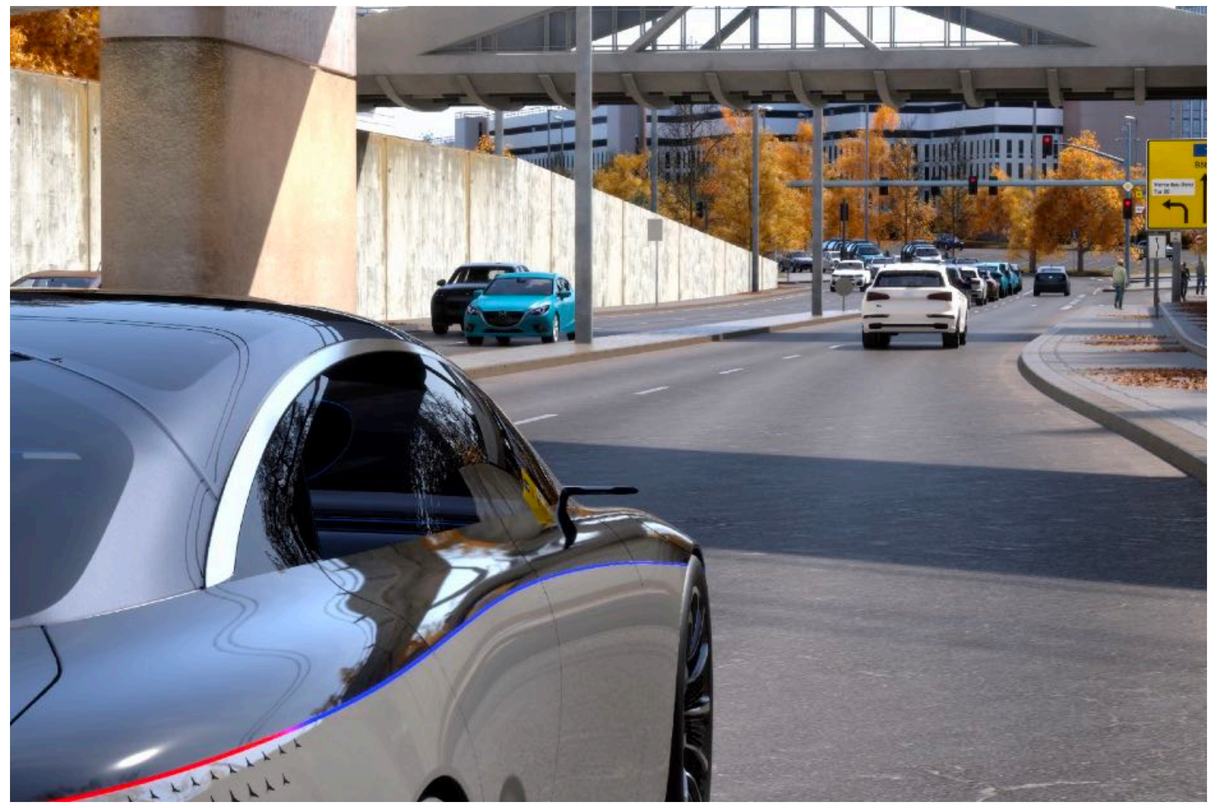
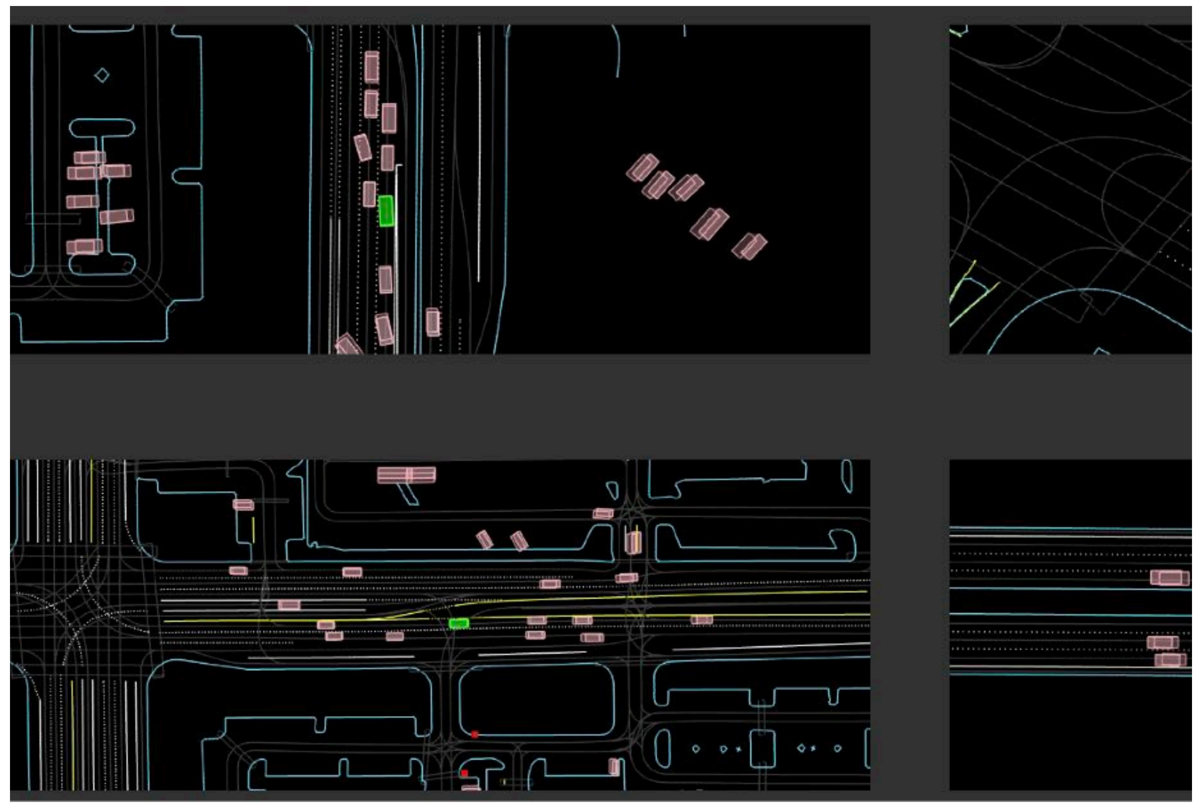
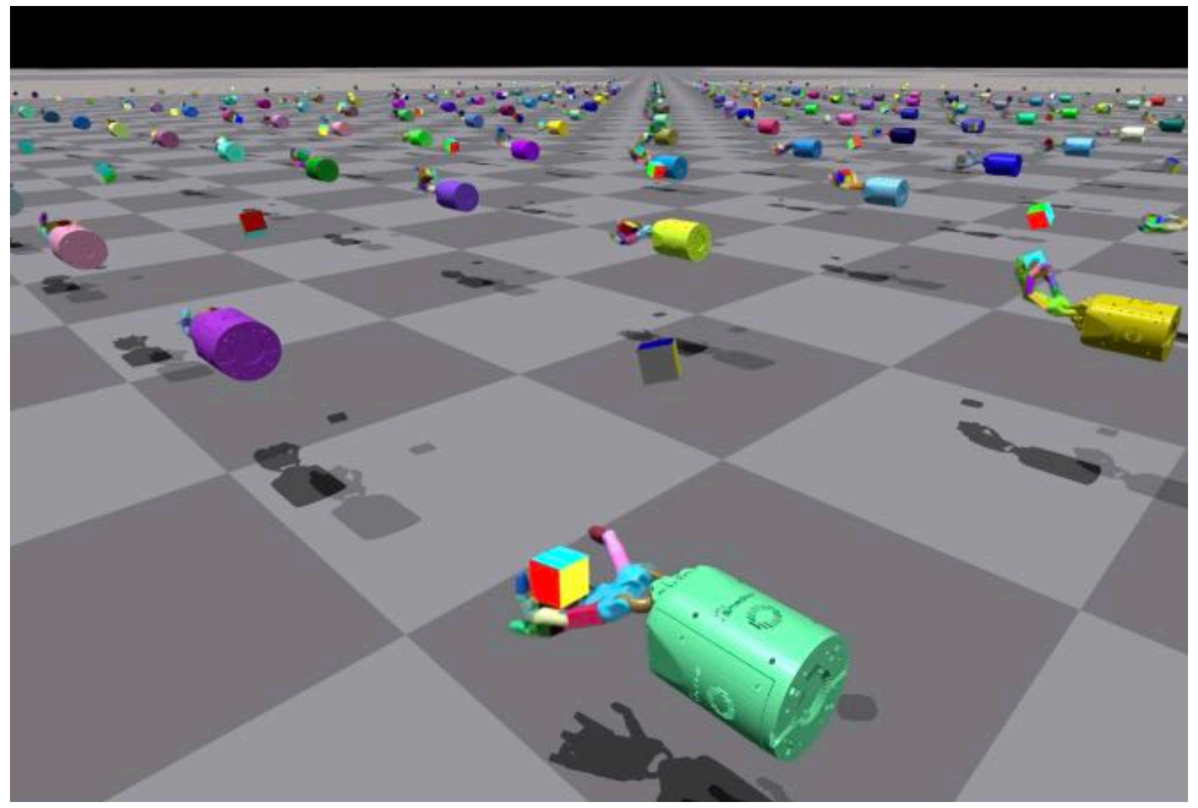
- Slides on the landscape of high-performance world simulation efforts designed for improving the efficiency of training embodied AI agents
- Discussion of the Madrona system
- Discussion: do we even need to simulate from a traditional world model at all?
  - Can't generative AI just make our training data?
  - Setup for ionight's reading: Genie

Recall from last time

1. Training and agent to learn complex skills can require many (millions, even billions) of trial-and-error steps (aka large amounts of "training experience")

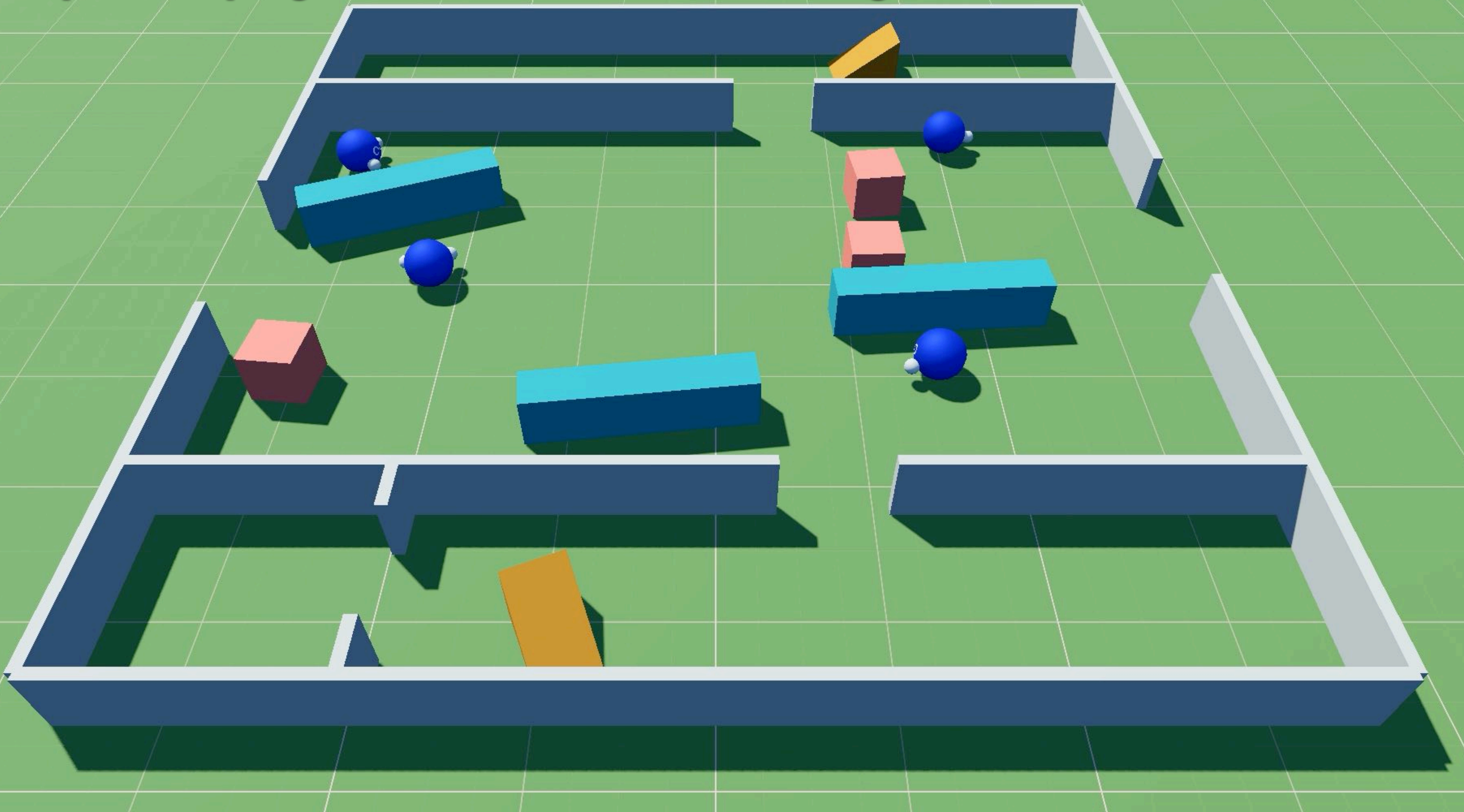


2. Researchers create virtual environments to simulate all this experience.


Basic training system components

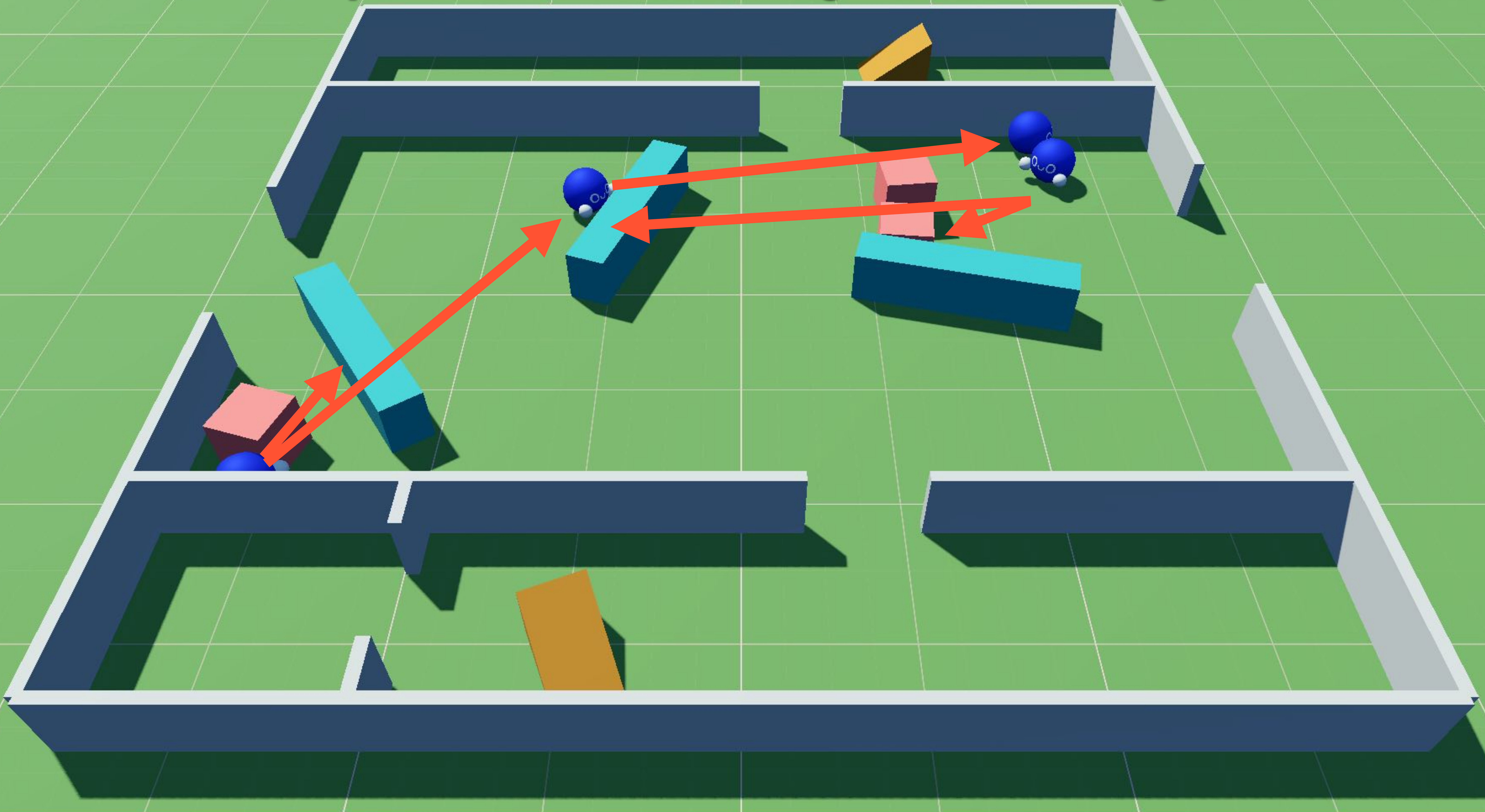
A simple example game: Hide and seek with four agents



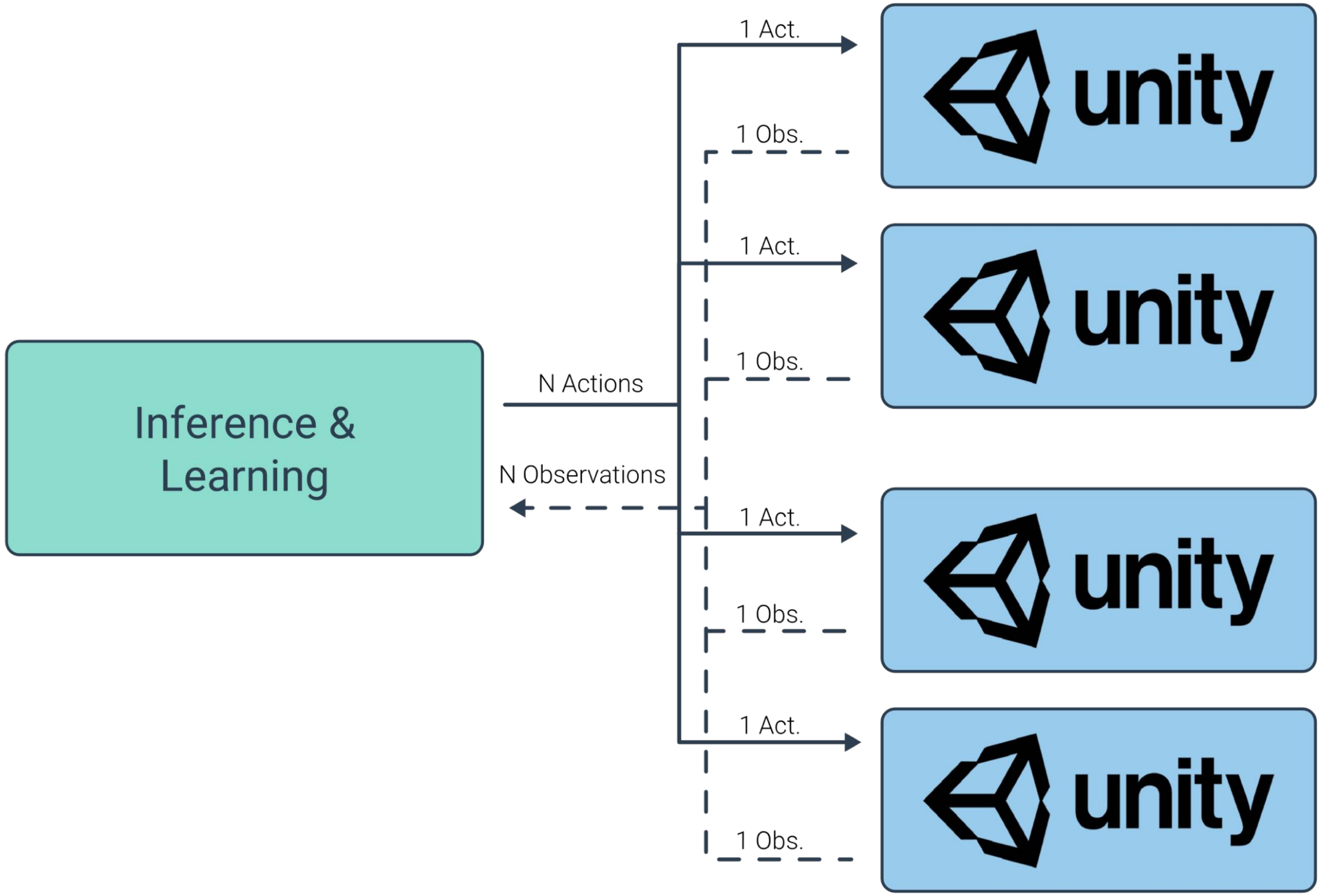
Hide and seek state: world entities

Hide and seek: examples of per-entity state (for agents)

Hide and seek: example observations (which agents does each agent see?)



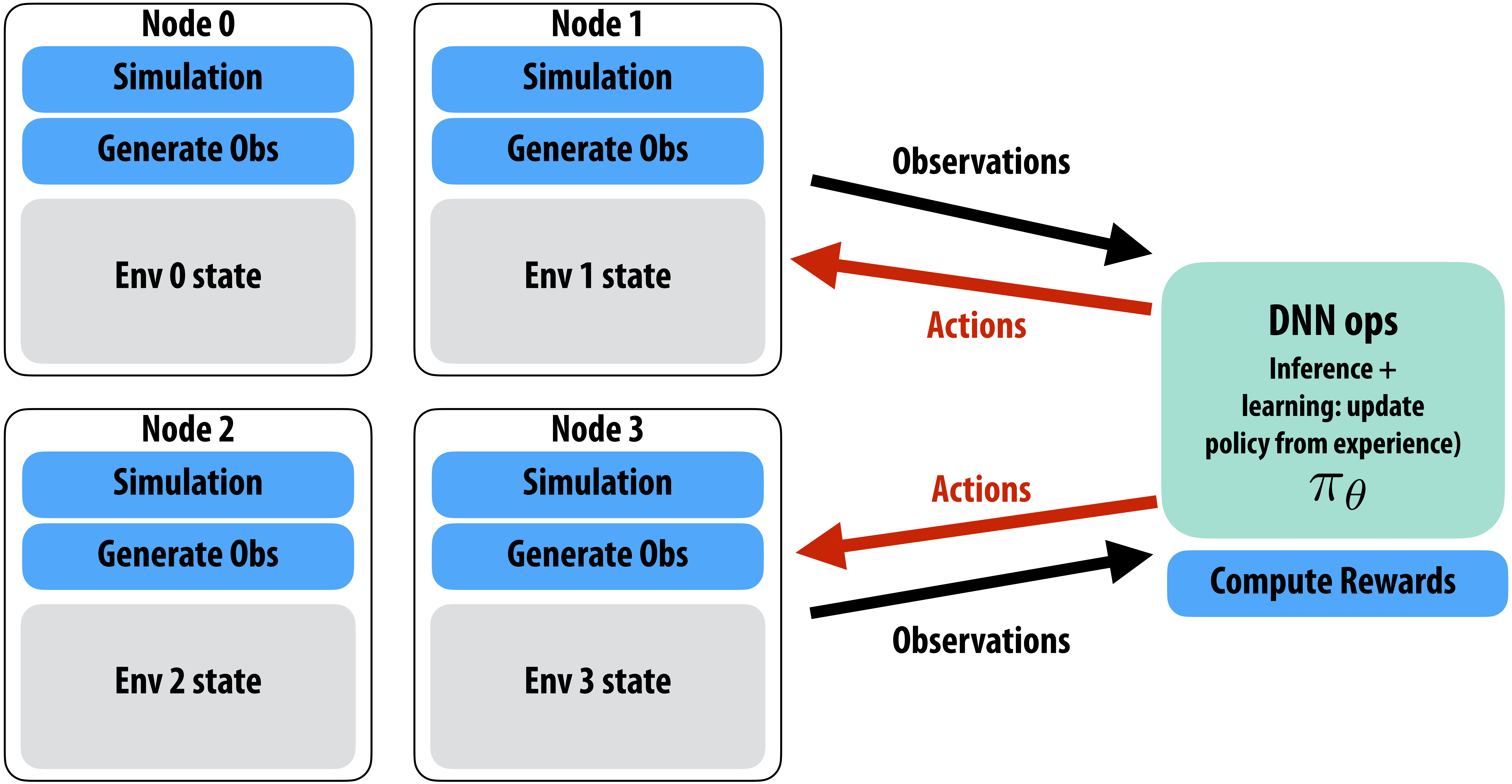
Common simulation approach: treat simulator as a black box, increase simulation throughput via "scale-out" parallelization



Treat existing simulation engines as an unmodifiable black box.

Run many copies of the black box in parallel.

A basic design: parallelize over workers



One example of this design: EnvPool (one multi-core node)

- **Pros:**
  - Use any existing simulator, unmodified
  - Collects observations from environments, provides them to Python as a Tensor
- **Cons:**
  - See upcoming slides (simulator-learning code sync costs, running many independent simulators is not optimal on high throughput machines)

README · Code of conduct · Apache-2.0 license

ENVPOOL

pypi v0.8.4 · PyPI Download 119k · arXiv 2206.10558 · docs passing · Bazel Build and Test passing · issues 49 open · Stars 1k · Forks 90 · license Apache-2.0

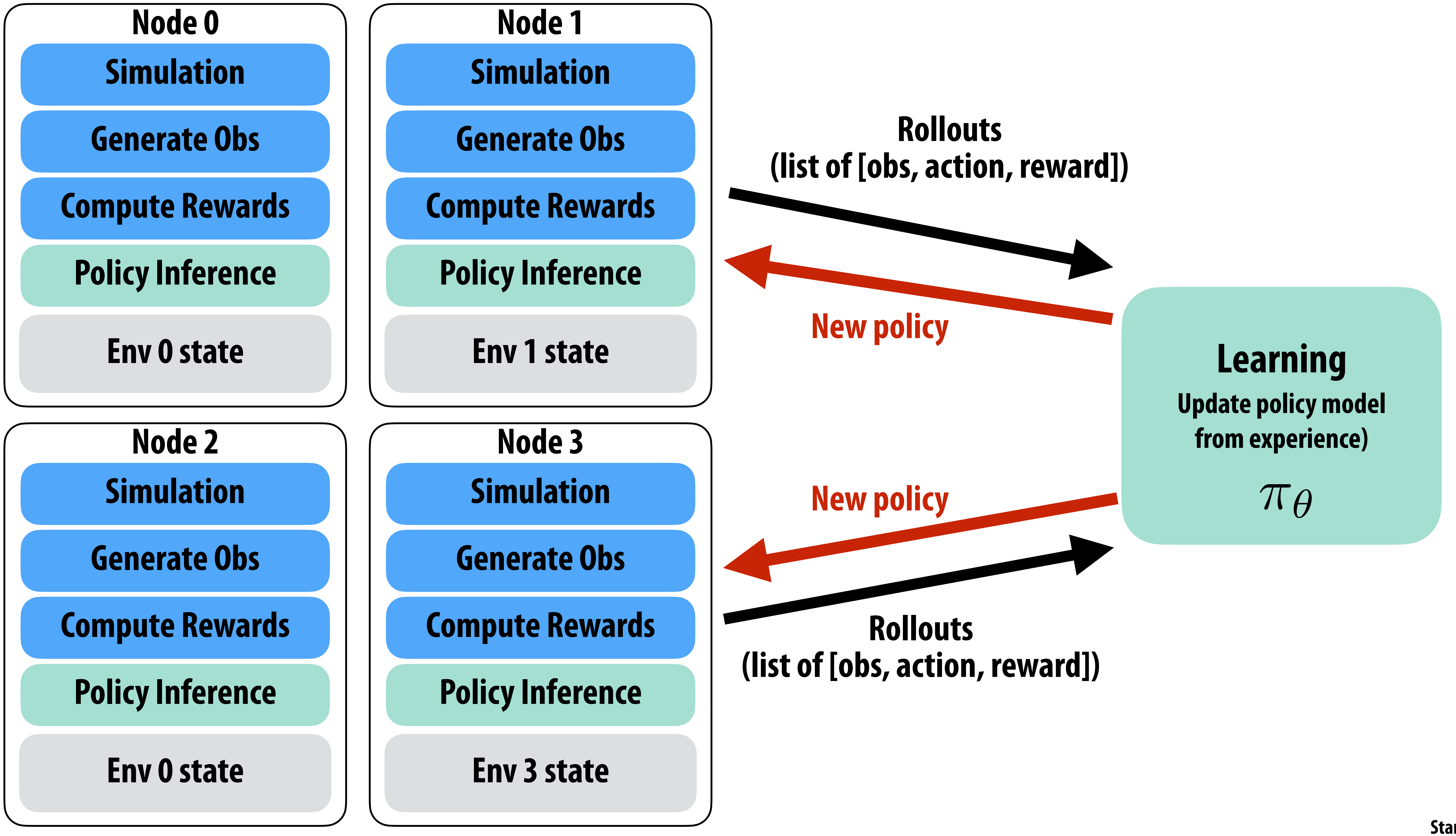
EnvPool is a C++-based batched environment pool with pybind11 and thread pool. It has high performance (~1M raw FPS with Atari games, ~3M raw FPS with Mujoco simulator on DGX-A100) and compatible APIs (supports both gym and dm_env, both sync and async, both single and multi player environment). Currently it supports:

- [x] Atari games
- [x] Mujoco (gym)
- [x] Classic control RL envs: CartPole, MountainCar, Pendulum, Acrobot
- [x] Toy text RL envs: Catch, FrozenLake, Taxi, NChain, CliffWalking, Blackjack
- [x] ViZDoom single player
- [x] DeepMind Control Suite
- [x] Box2D
- [x] Procgen
- [x] Minigrid

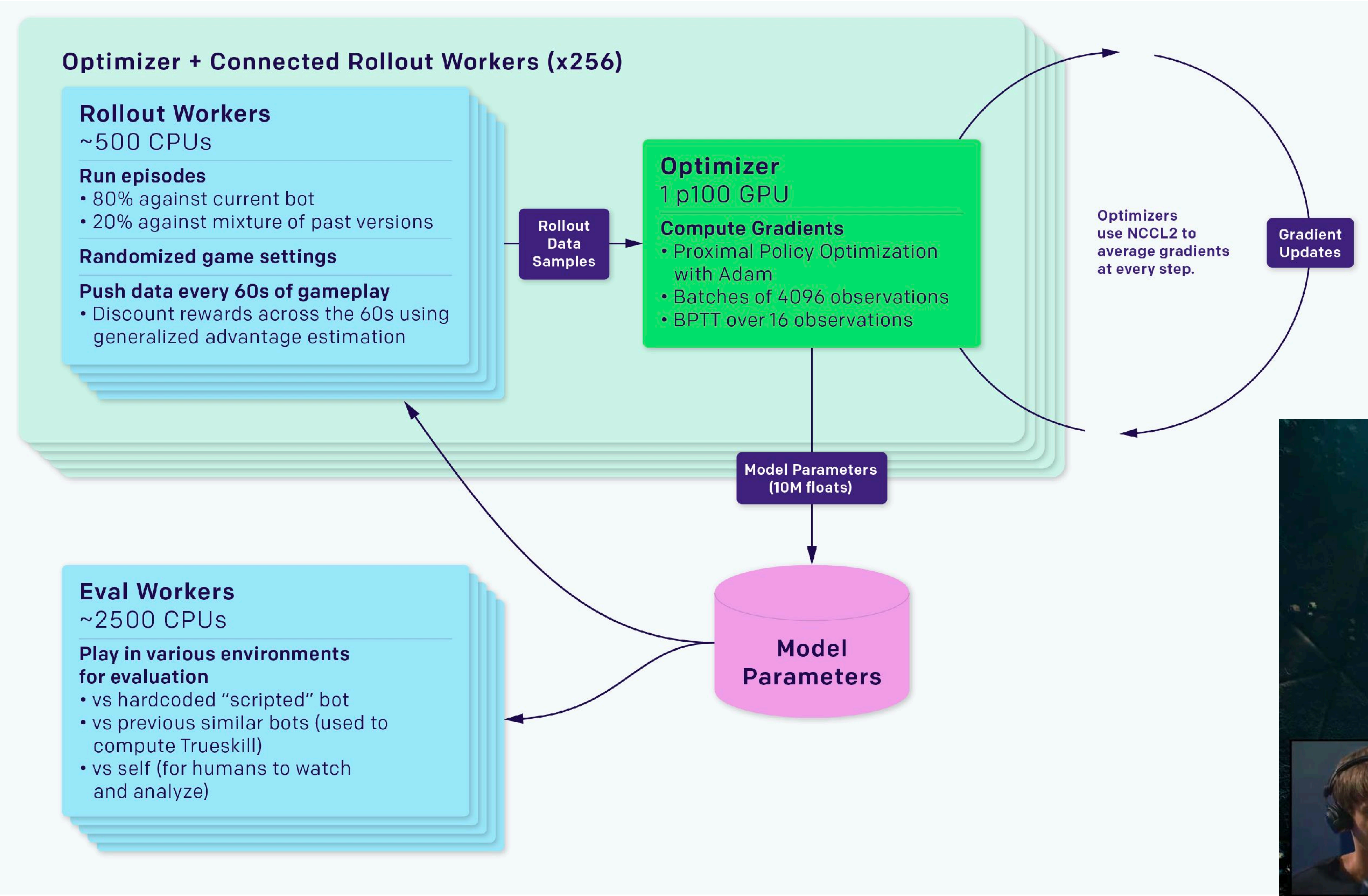
Here are EnvPool's several highlights:

- Compatible with OpenAI `gym` APIs, DeepMind `dm_env` APIs, and `gymnasium` APIs;
- Manage a pool of envs, interact with the envs in batched APIs by default;
- Support both synchronous execution and asynchronous execution;
- Support both single player and multi-player environment;
- Easy C++ developer API to add new envs: Customized C++ environment integration;
- Free **~2x** speedup with only single environment;
- **1 Million** Atari frames / **3 Million** Mujoco steps per second simulation with 256 CPU cores, **~20x** throughput of Python subprocess-based vector env;
- **~3x** throughput of Python subprocess-based vector env on low resource setup like 12 CPU cores;
- Comparing with existing GPU-based solution (Brax / Isaac-gym), EnvPool is a **general** solution for various kinds of speeding-up RL environment parallelization;

Similar design for distributed system: parallelize over workers



Example: Rapid by (OpenAI)

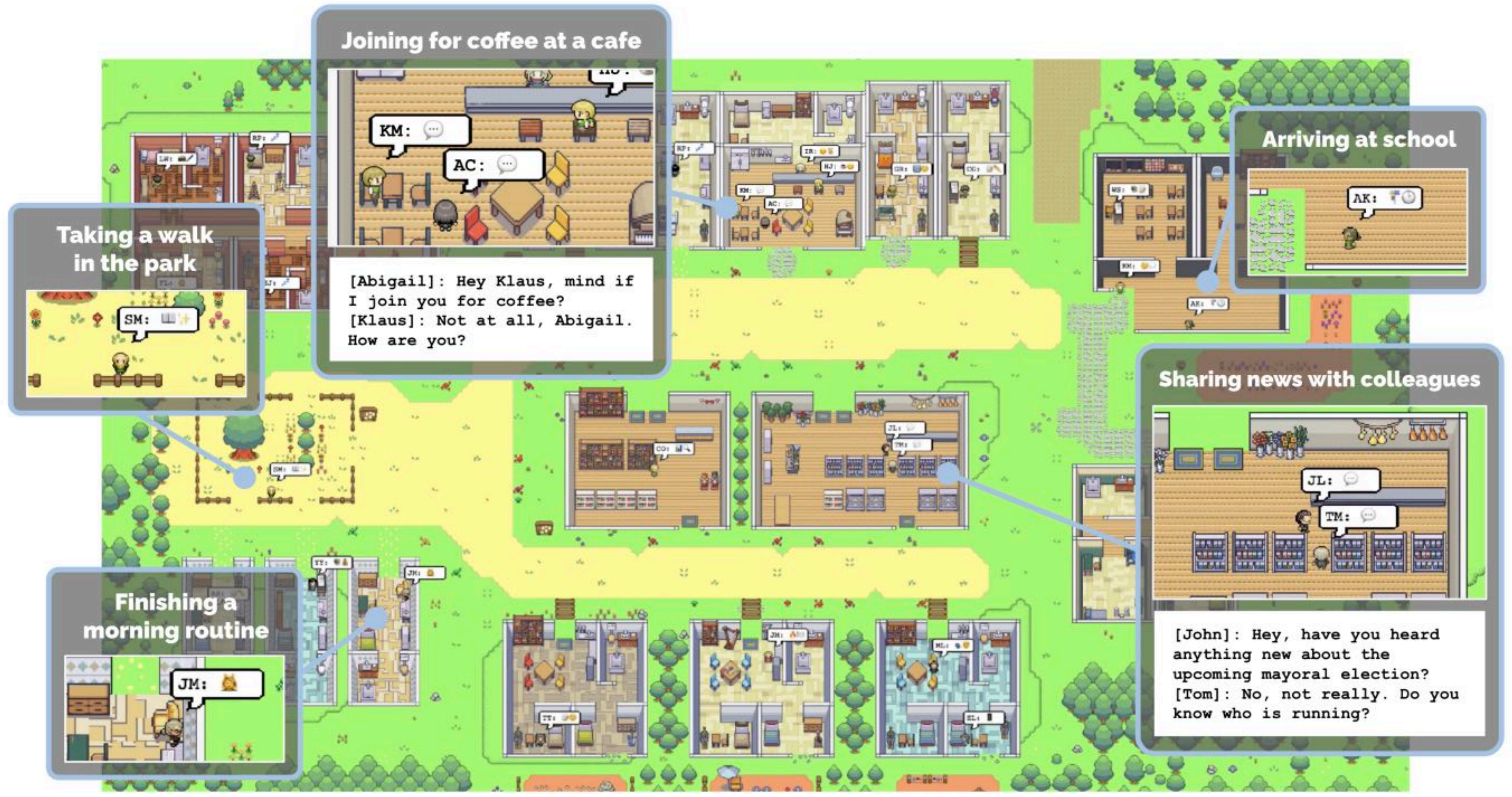


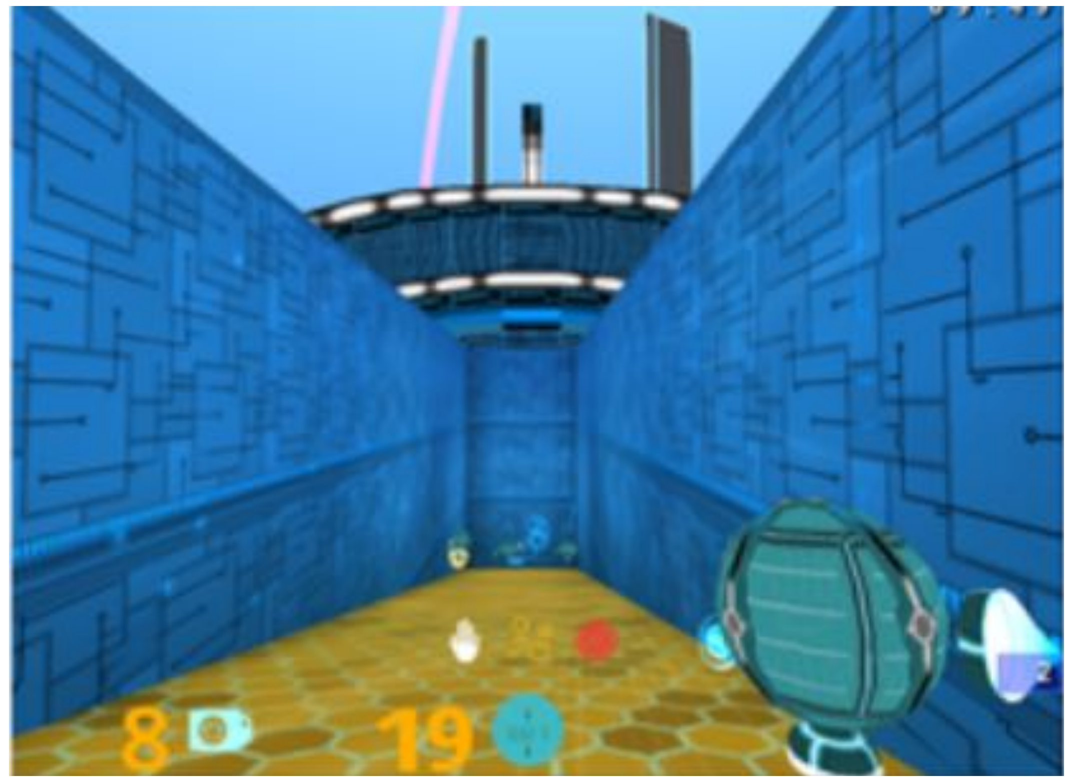
OpenAI Five stats (Dota 2)

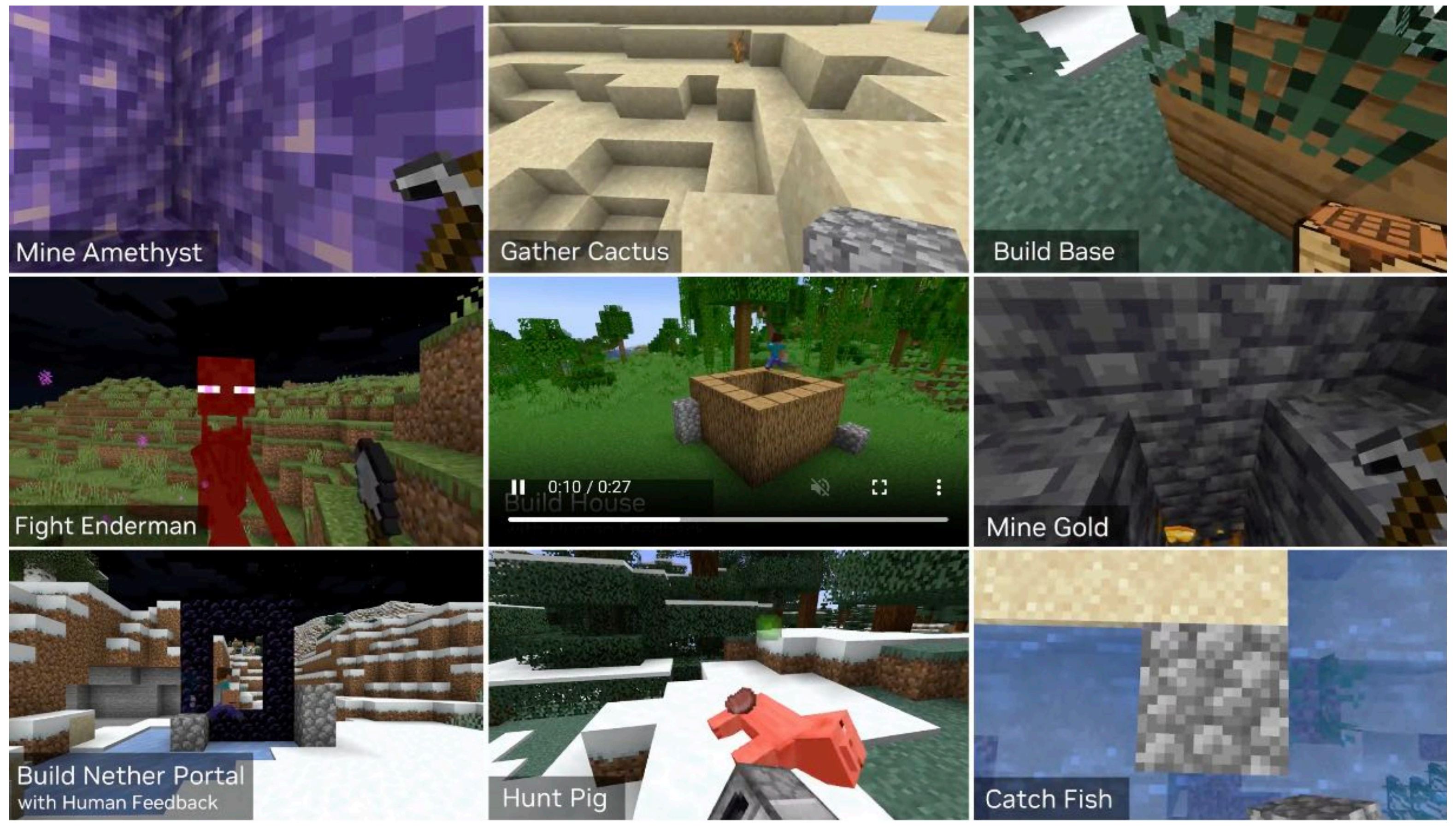
	OPENAI FIVE
CPUs	128,000 preemptible CPU cores on GCP
GPUs	256 P100 GPUs on GCP
Experience collected	~180 years per day (~900 years per day counting each hero separately)
Size of observation	~36.8 kB
Observations per second of gameplay	7.5
Batch size	1,048,576 observations
Batches per minute	~60

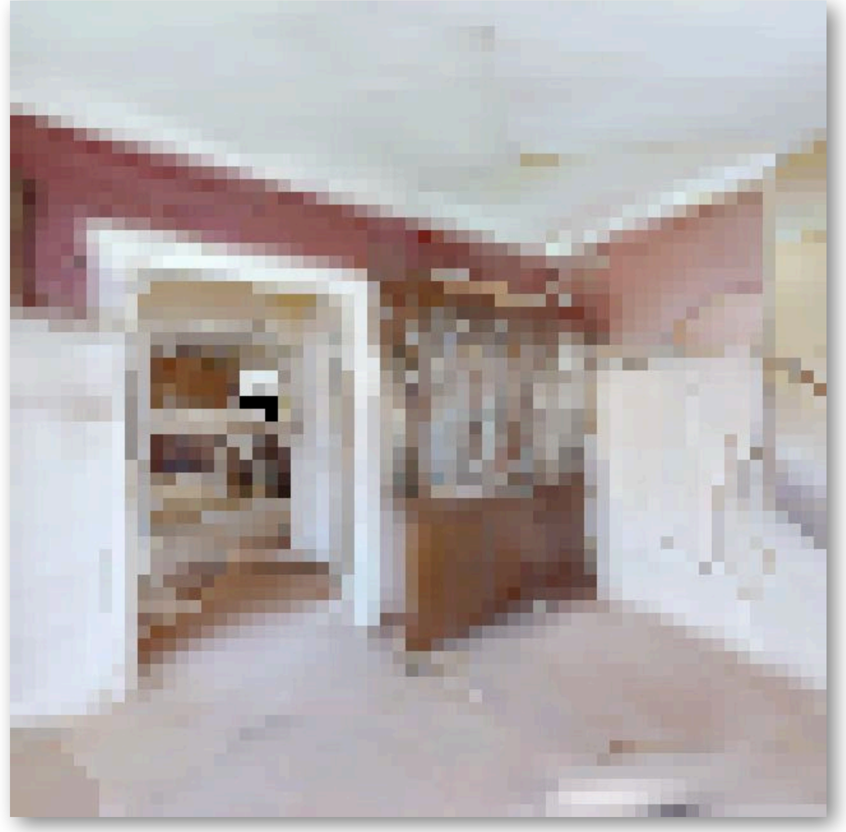
Example public RL learning environments

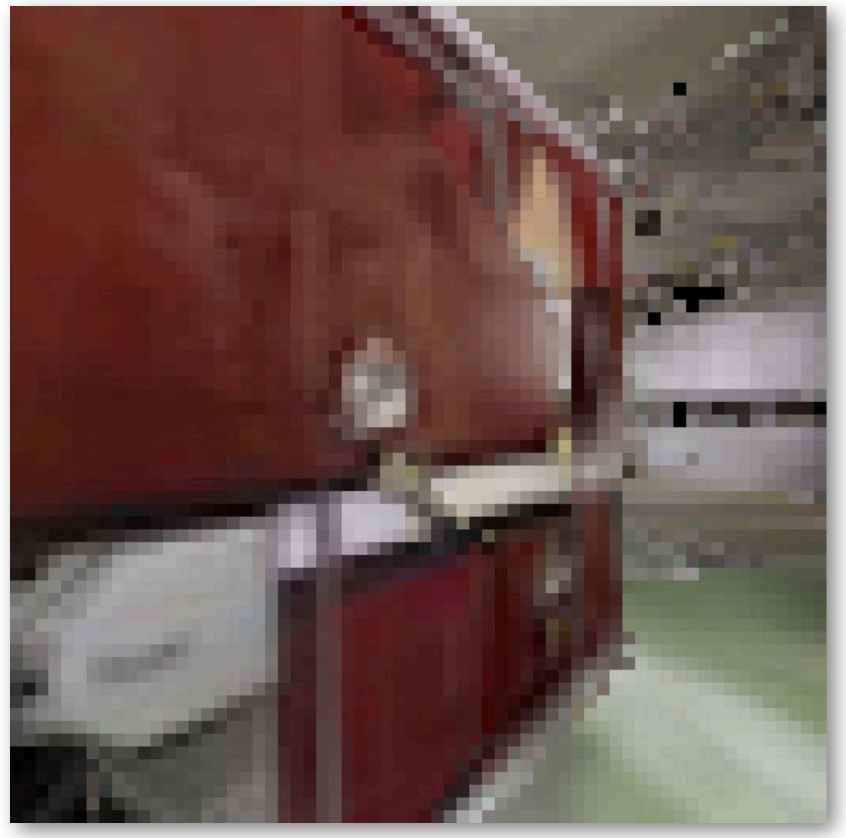
- Lower complexity worlds
- Lower fidelity observations

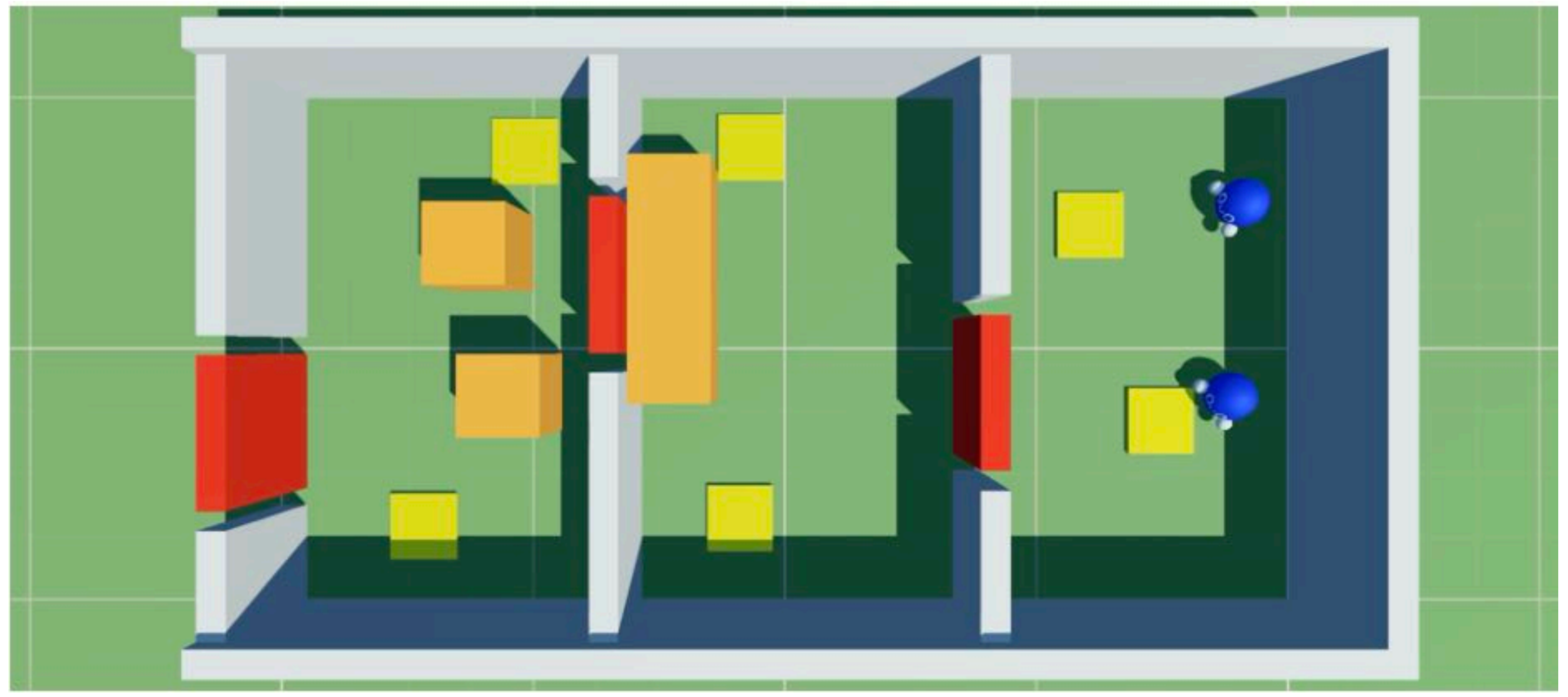


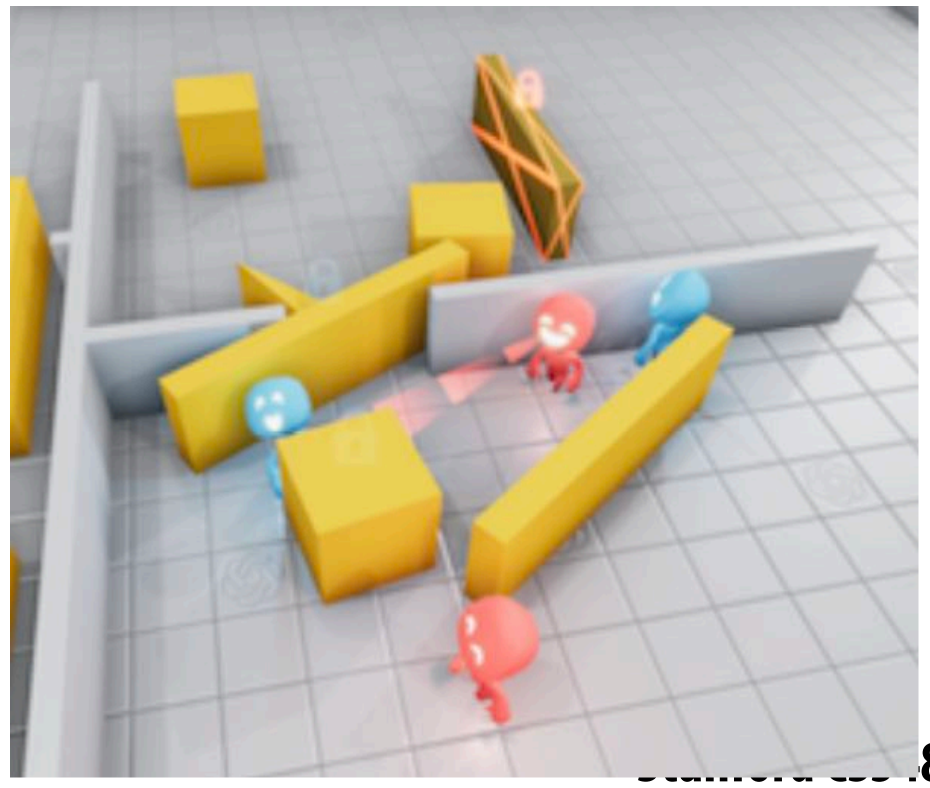










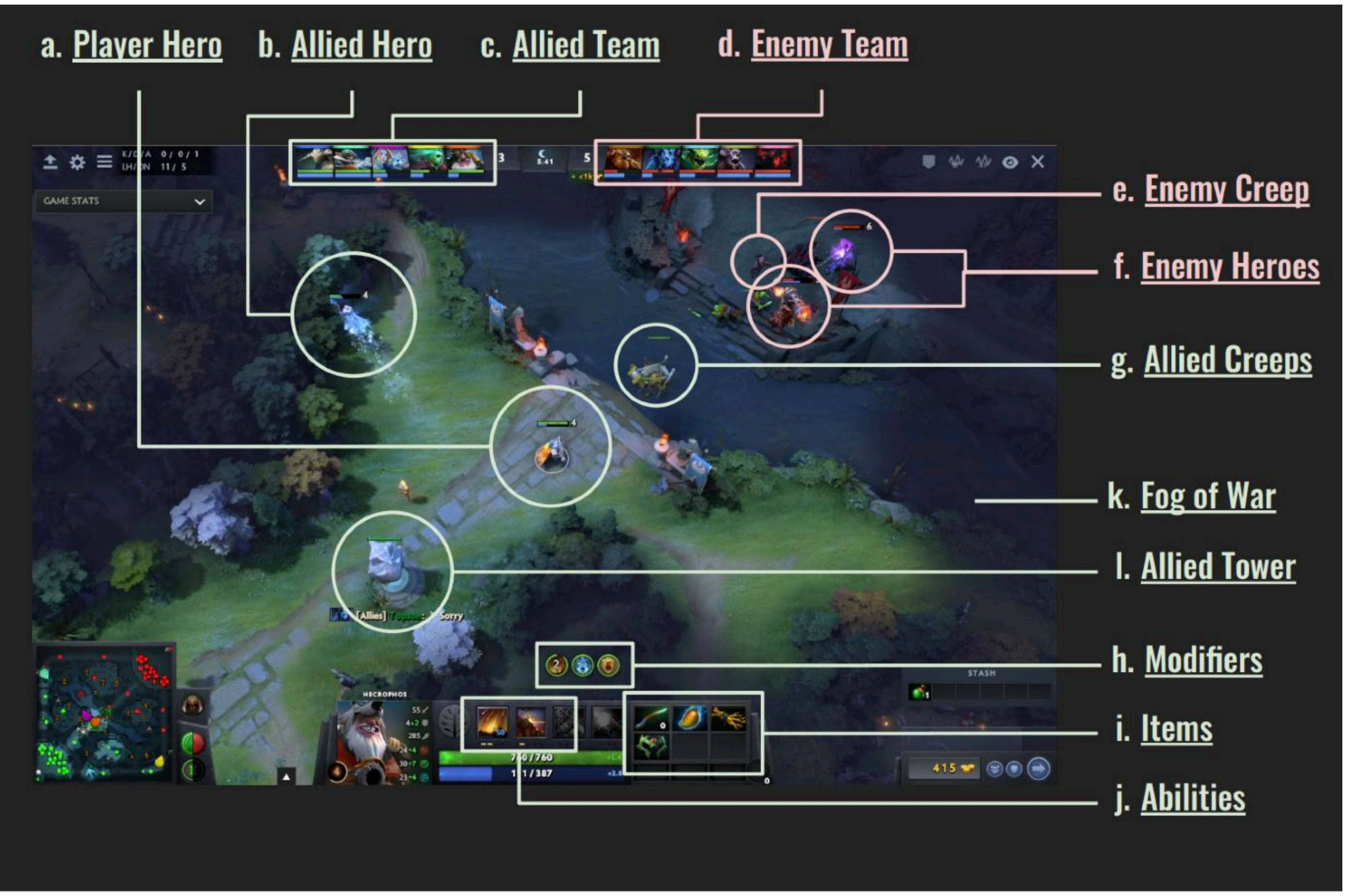


In contrast: what a modern GPU is designed to render
(Very high fidelity observations)

Unreal Nanite Demo

Large-scale agent training is expensive!

OpenAI Five



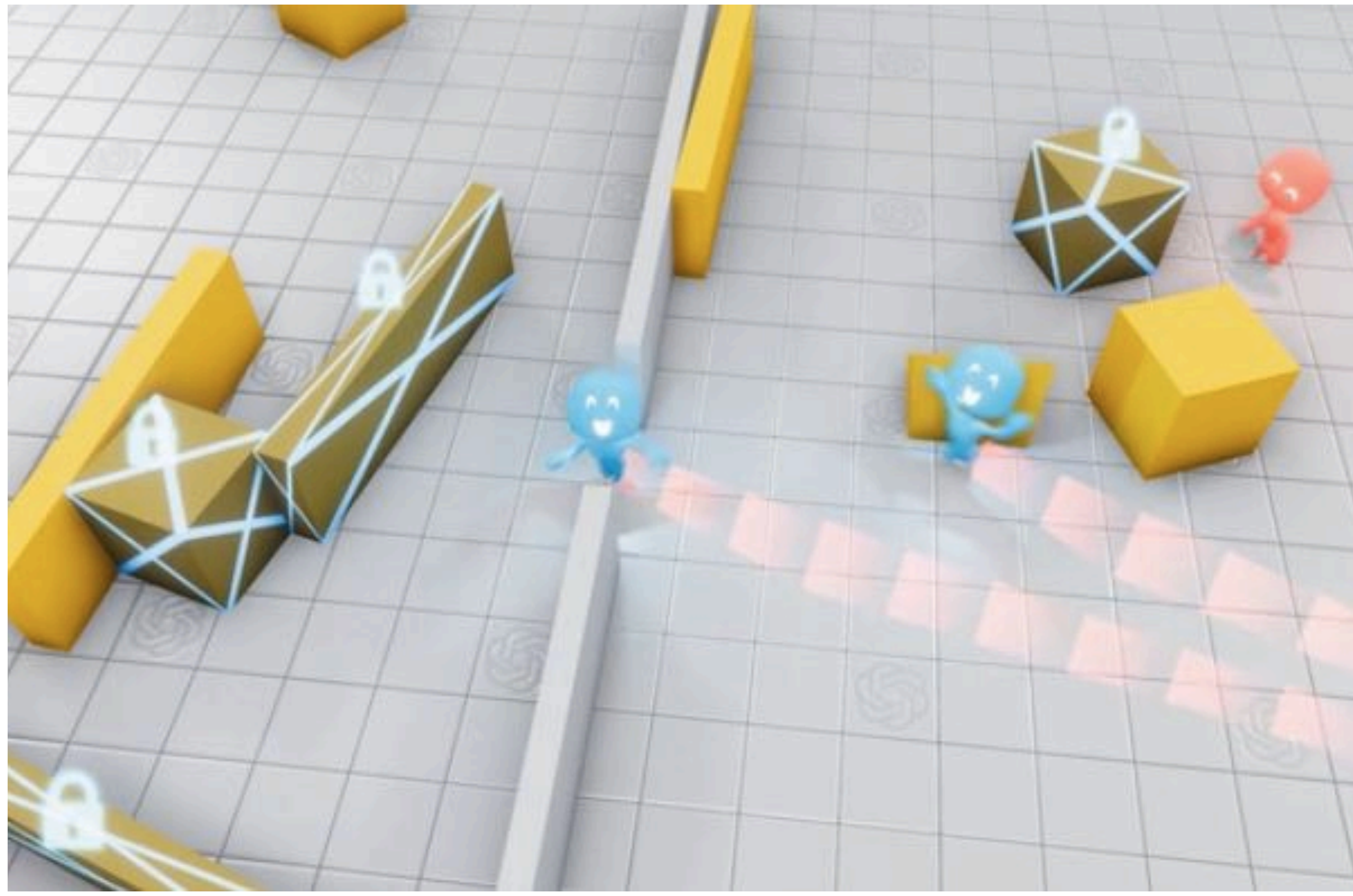
Learning Dota 2:
Months of training

CPUs	128,000 preemptible CPU cores on GCP
GPUs	256 P100 GPUs on GCP
Experience collected	~180 years per day (~900 years per day counting each hero separately)

Robotics in Virtual World

64 GPUs over 2.5 days
(2B experience samples)

OpenAI Hide and Seek



High-level strategies emerge
after billions of world time steps

Design issues of basic scale-out approach

- Inefficient simulation/rendering: low-complexity worlds do not make good use of a modern parallel processor's resources
  - GPUs won't achieve high-throughput rendering/physics with smaller workloads

- Inefficient communication between simulation and inference/training

- Duplication of computation and memory footprint (for scene data) across environment simulator instances

- Seems wasteful, right?

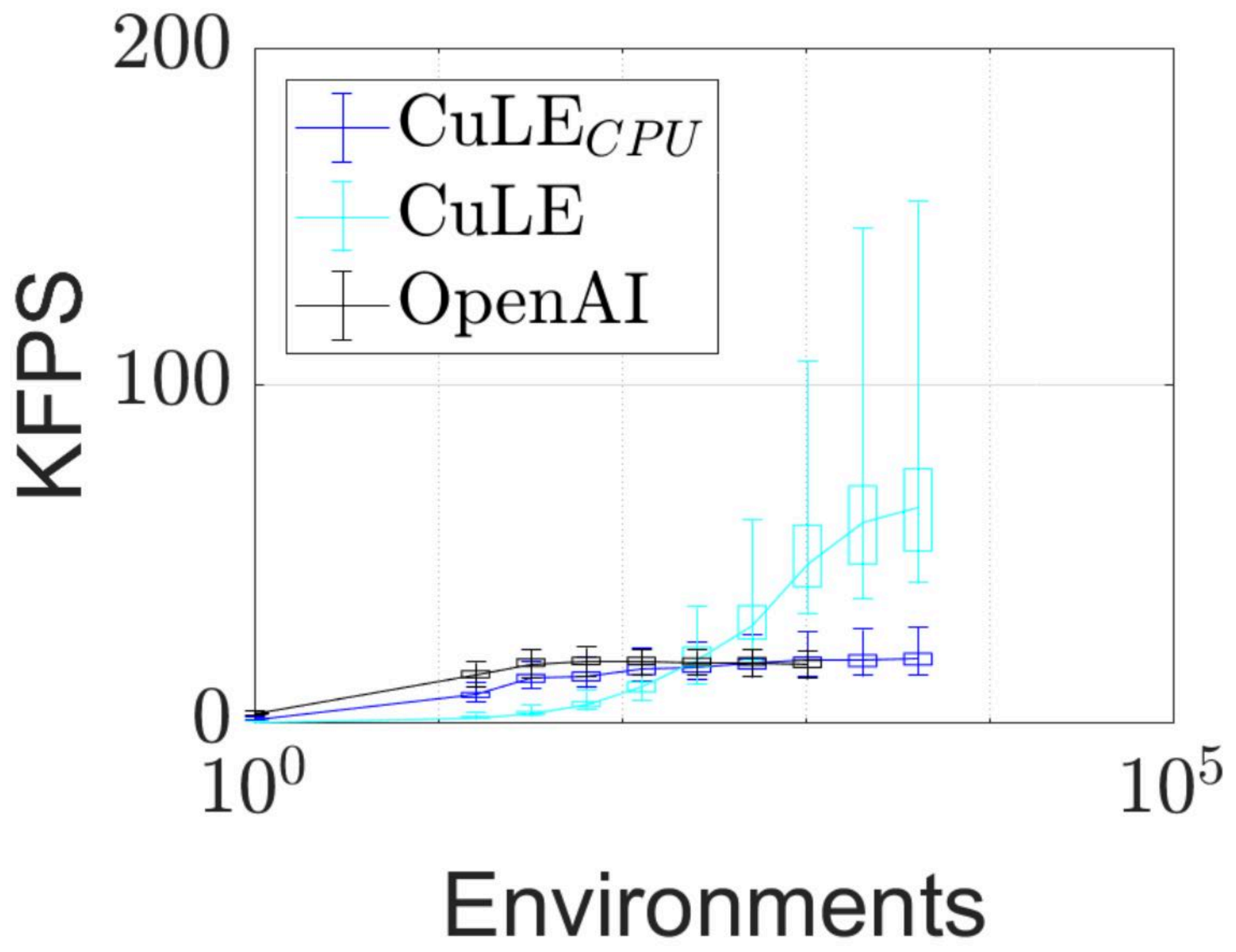
A new visual computing systems research question:

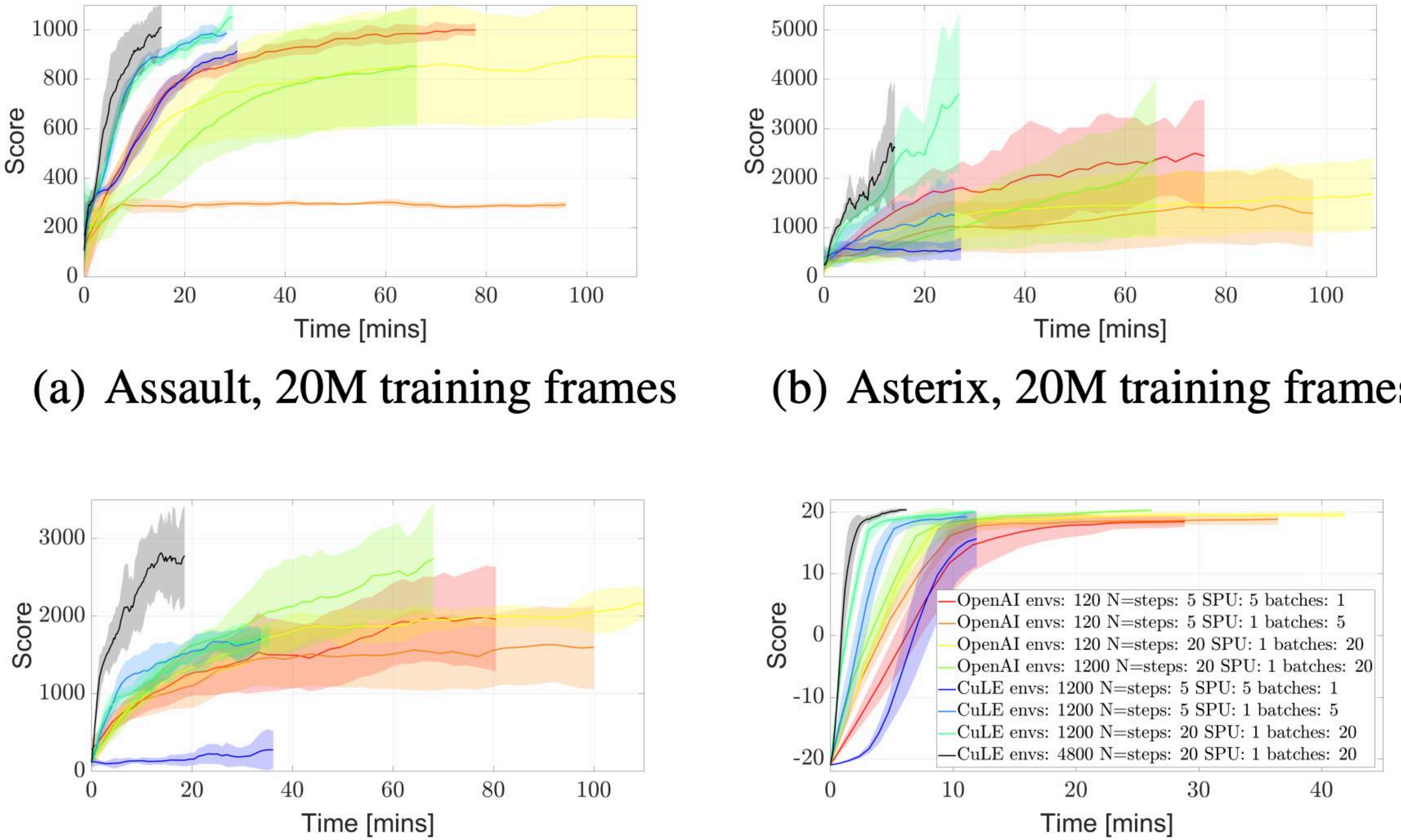
Can we execute embodied AI training more efficiently if we architect a world simulation engine from the ground up to process many independent worlds at once?

Batch environment simulation

CuLE: Rewriting an Atari emulator in CUDA

- One CUDA thread = work for one enumerator instance
- Large numbers of threads execute





(a) Assault, 20M training frames

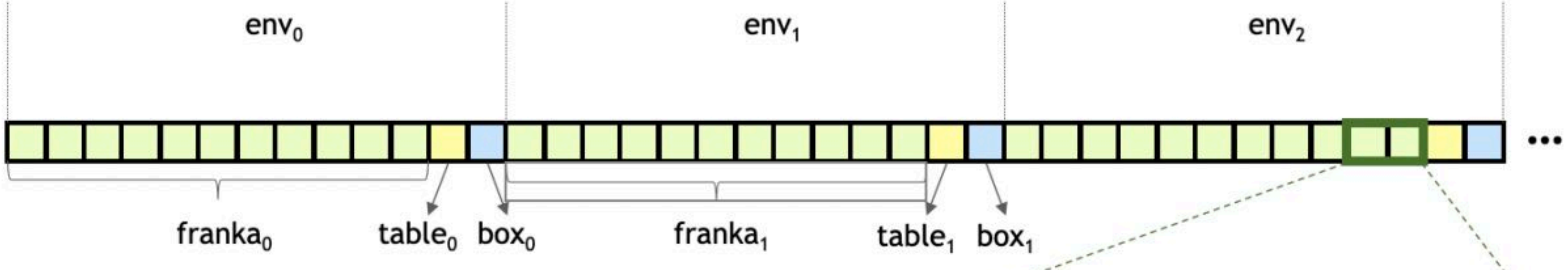
(b) Asterix, 20M training frames

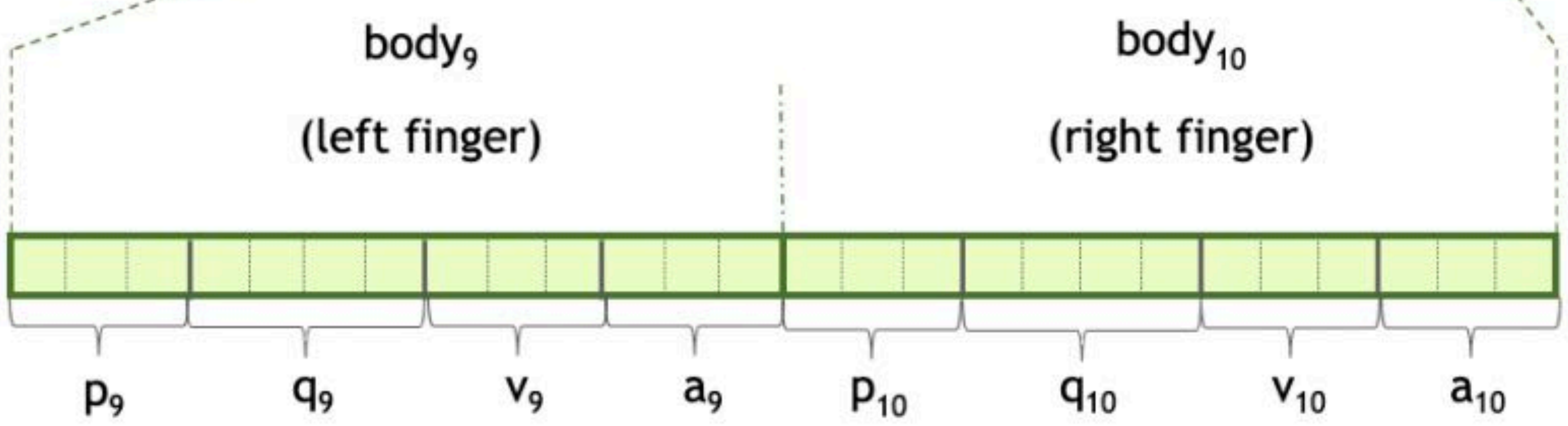
(c) Ms-Pacman, 20M training frames

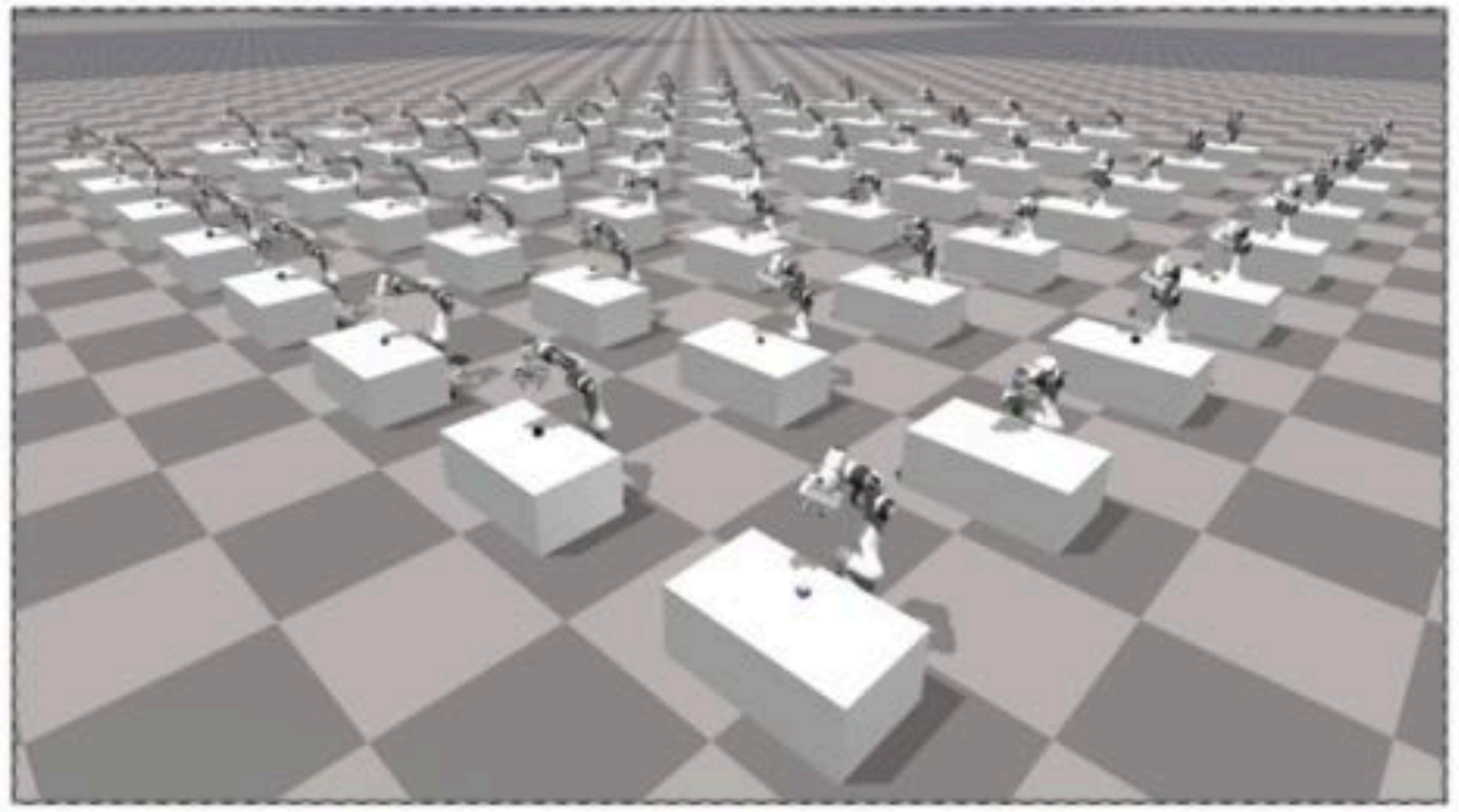
(d) Pong, 8M training frames

NVIDIA Issac Gym

- Batched many-environment execution applied to rigid body physics sim
- Simulate 100's to 1000's of world environments simultaneously on the GPU
- Current state for all environments packaged in a single PyTorch tensor
- User can write GPU-accelerated loss/reward functions in PyTorch on this tensor
- Result: tight loop of simulate/infer/train

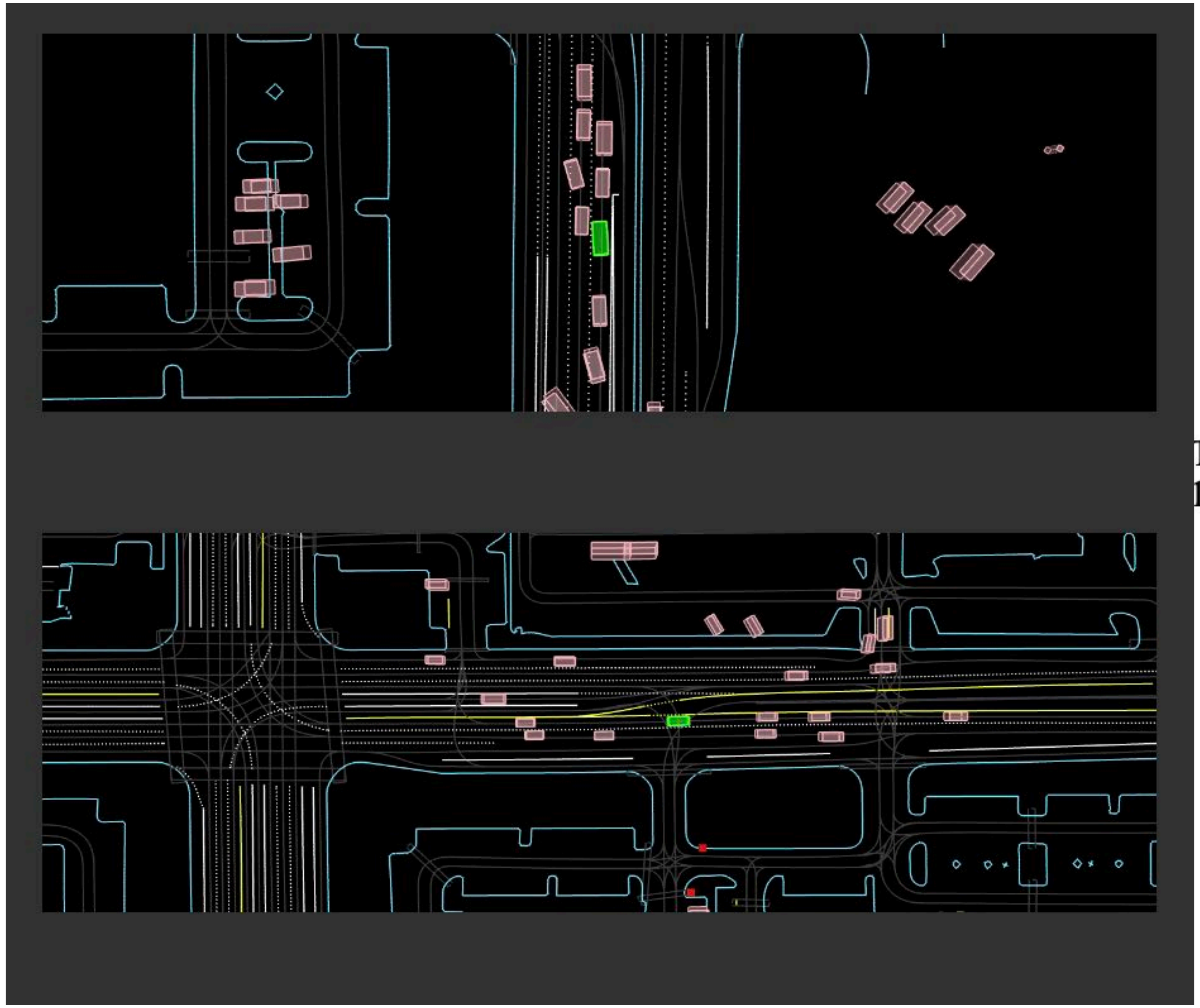






Waymax

- Self-driving car simulator built using Jax programming environment
- Environment state stored in JAX tensors (max number of objects across all environments in a batch)

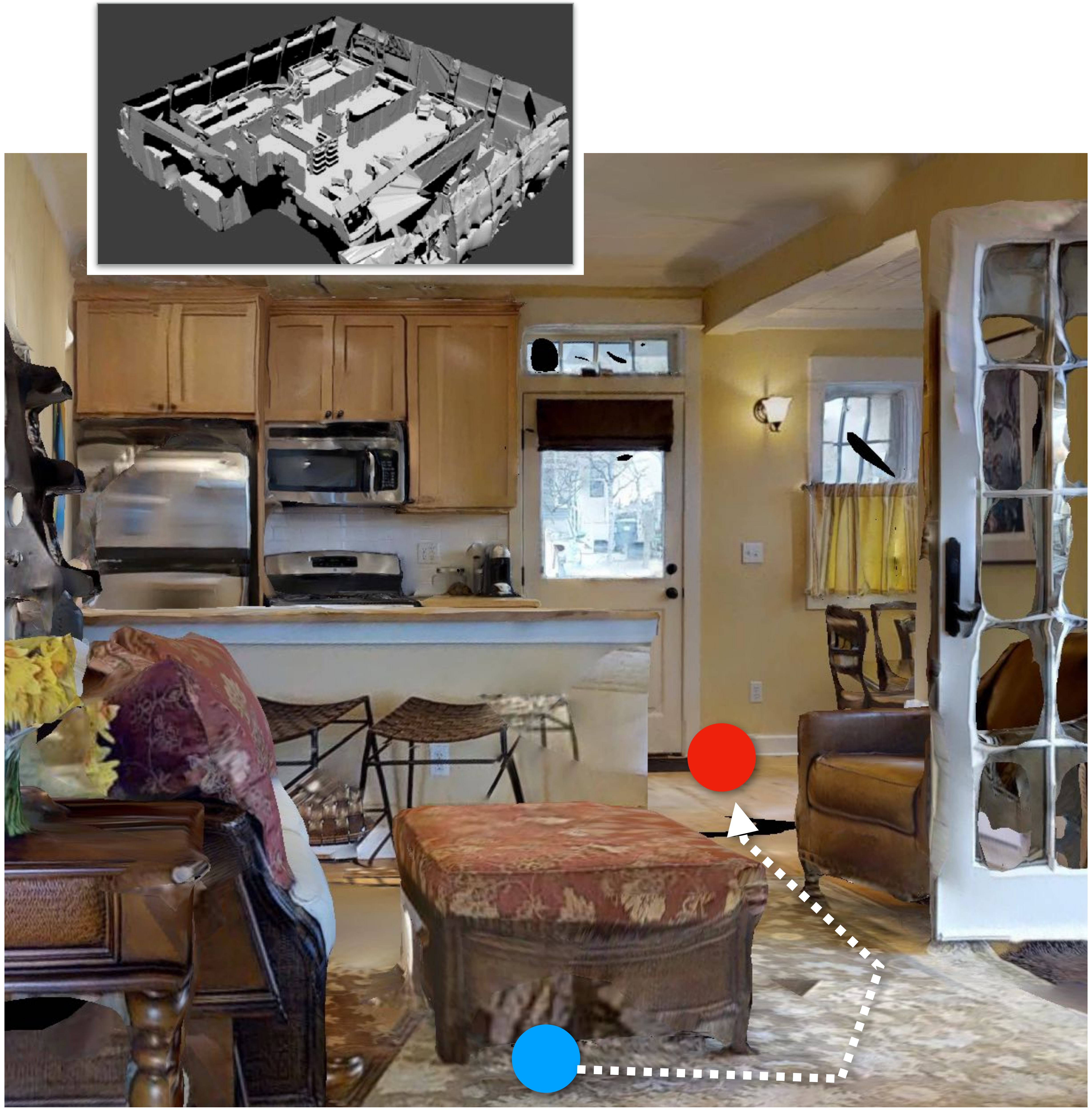


	Device	BS-1	BS-16	Reset	Step	Transition	Metrics	RolloutExpert
Single-Agent Env	CPU	✓		1.09	131	0.90	112	1.0×10^4
	CPU		✓	12.2	1.7×10^3	10.9	1.69×10^3	1.4×10^5
	GPU-v100	✓		0.58	0.75	0.47	0.21	56.2
	GPU-v100		✓	0.67	2.48	0.52	2.27	279
Multi-Agent Env	CPU	✓		6.23	129	1.01	112	1.1×10^4
	CPU		✓	49.8	1.1×10^3	14.3	1.72×10^3	1.6×10^5
	GPU-v100	✓		0.64	0.92	0.53	0.19	73.3
	GPU-v100		✓	0.81	2.86	0.51	2.24	OOM

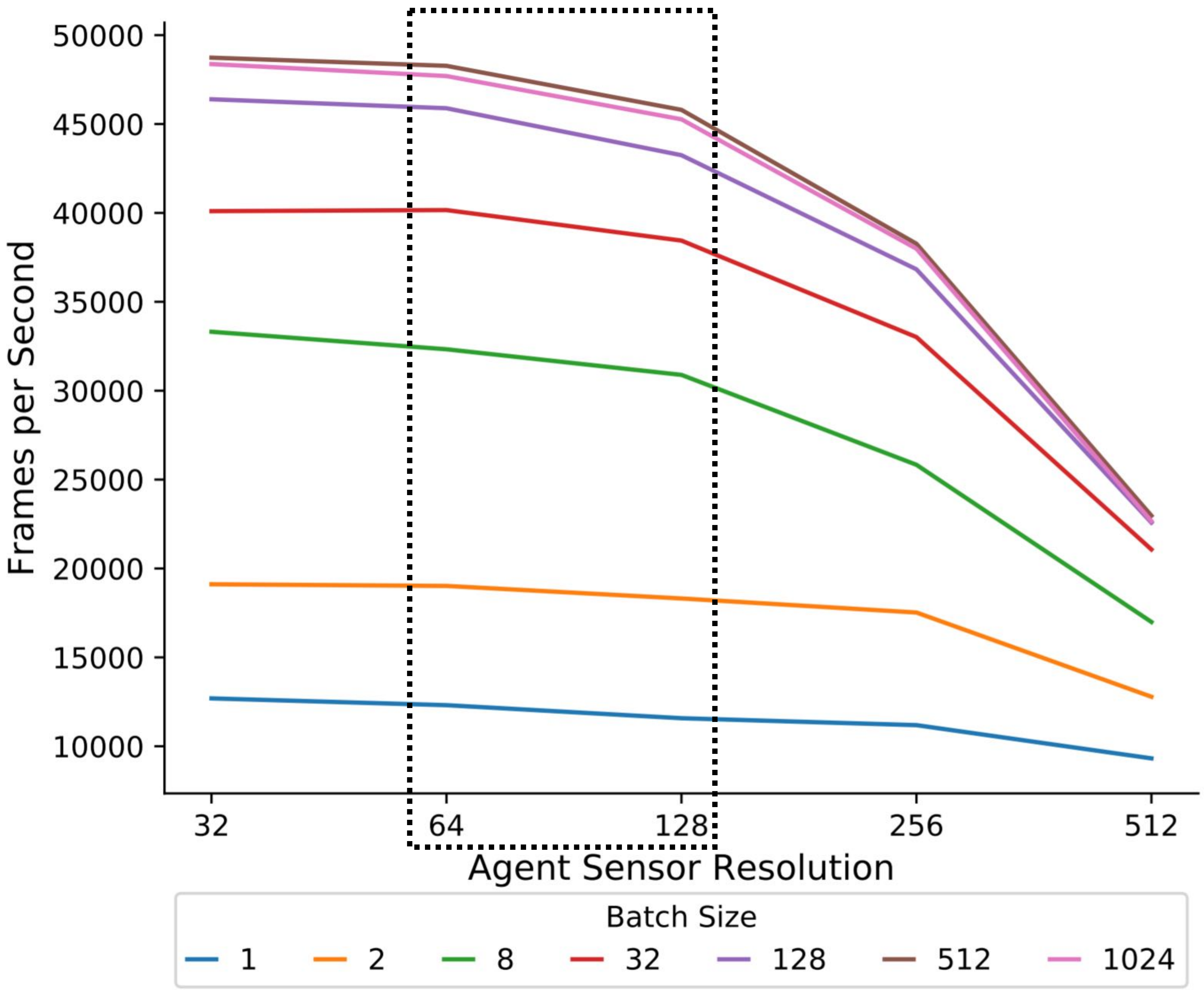
Table 2: Runtime benchmark in milliseconds: the environment controls all objects in the scene (up to 128 as defined in WOD).

Example: navigation in 3D scanned environments

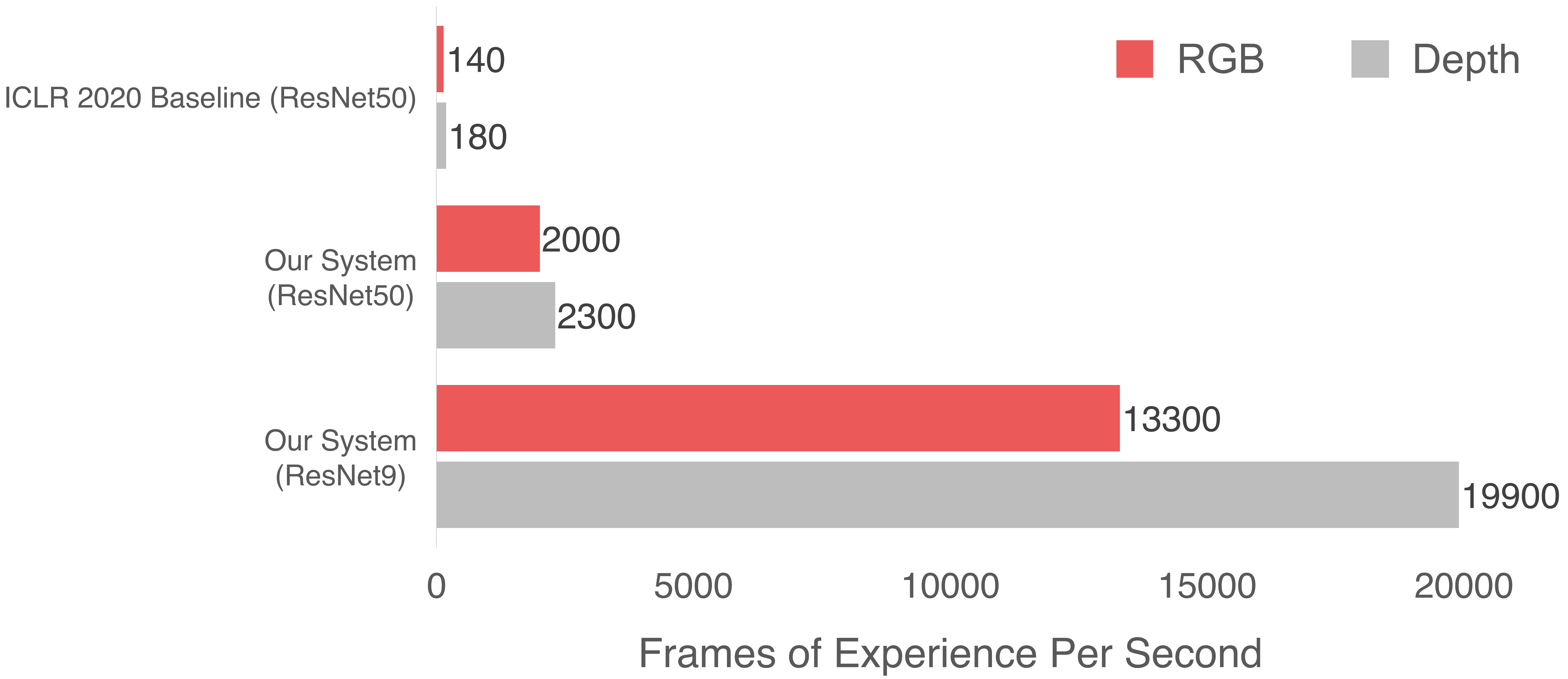
(these are multi-room floorplans…)



40,000 fps batch rendering of small images from popular 3D scanned virtual environments (Gibson/Matterport)



100x End-to-End Gain with Optimized Policy DNN

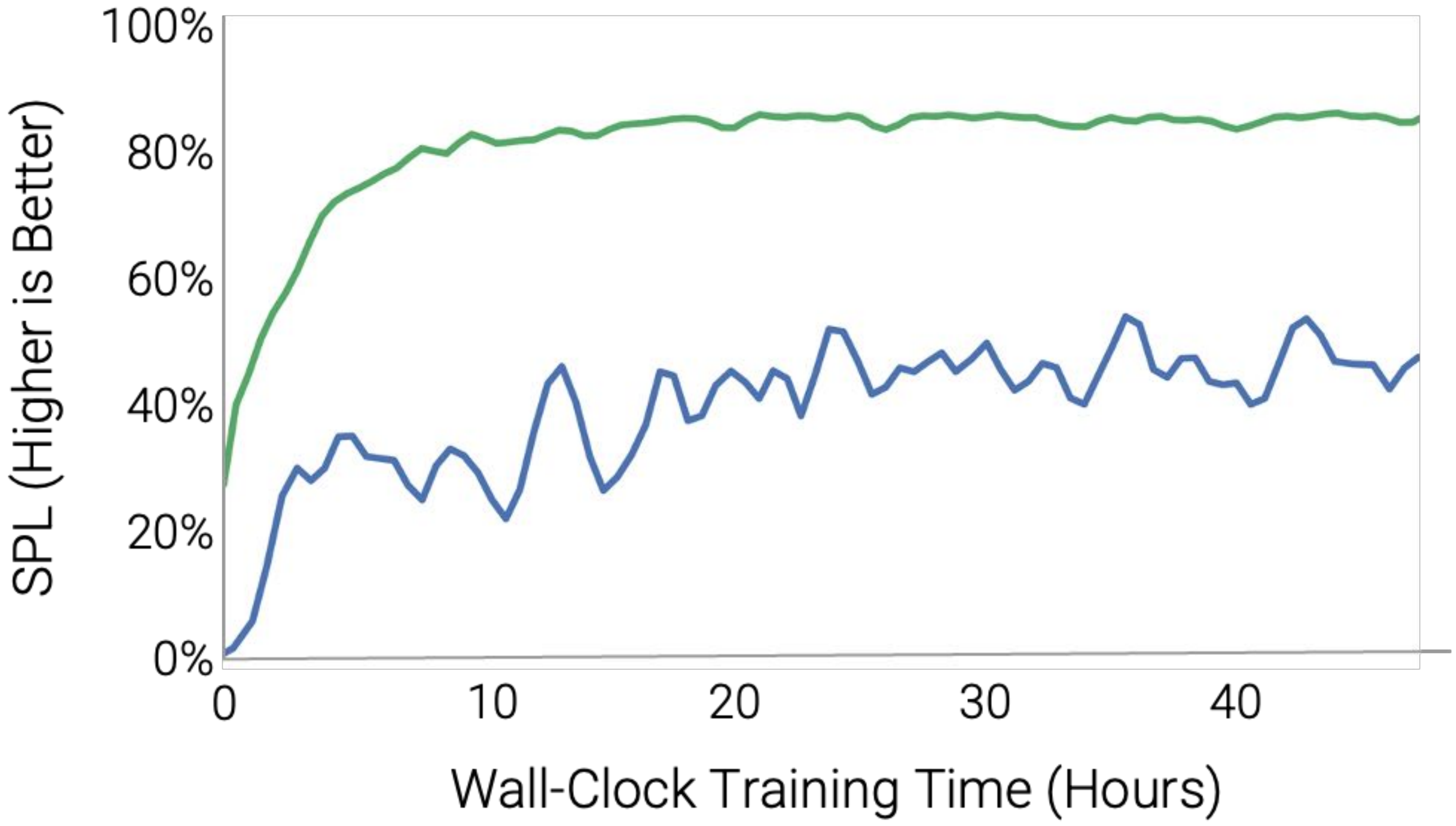


Benchmarked with RTX 3090

2.5B frames of experience in 48 hrs on a single RTX 3090

“PointGoal Navigation” in Gibson environments (Habitat Labs)

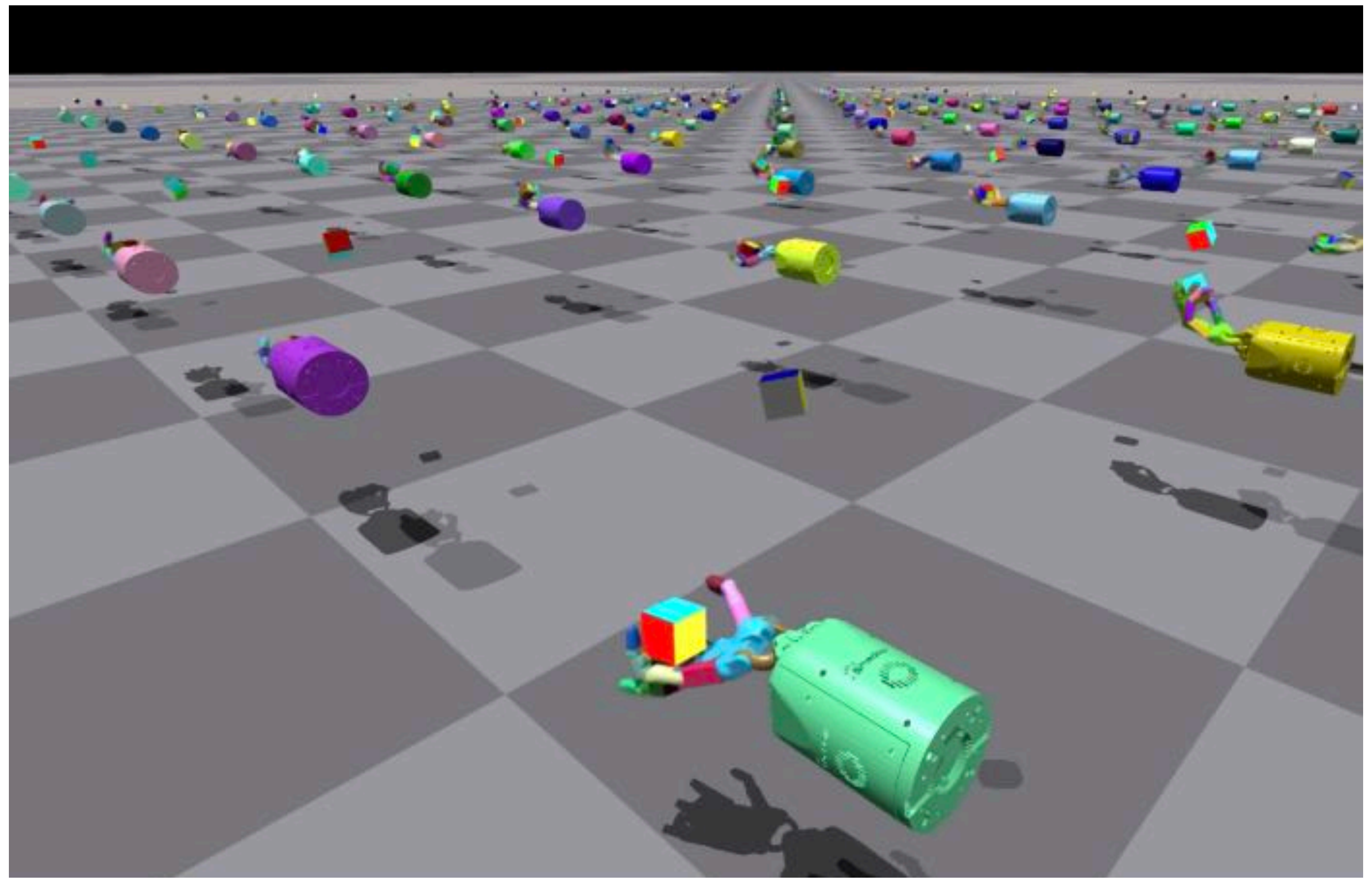
Run render->infer->train loop runs at 13,000 fps per GPU



"Large Batch Simulation for Deep Reinforcement Learning", B. Shacklett, E. Wijmans, A.Petrenko, M. Savva, D. Batra, V. Koltun, K. Fatahalian, ICLR 2021

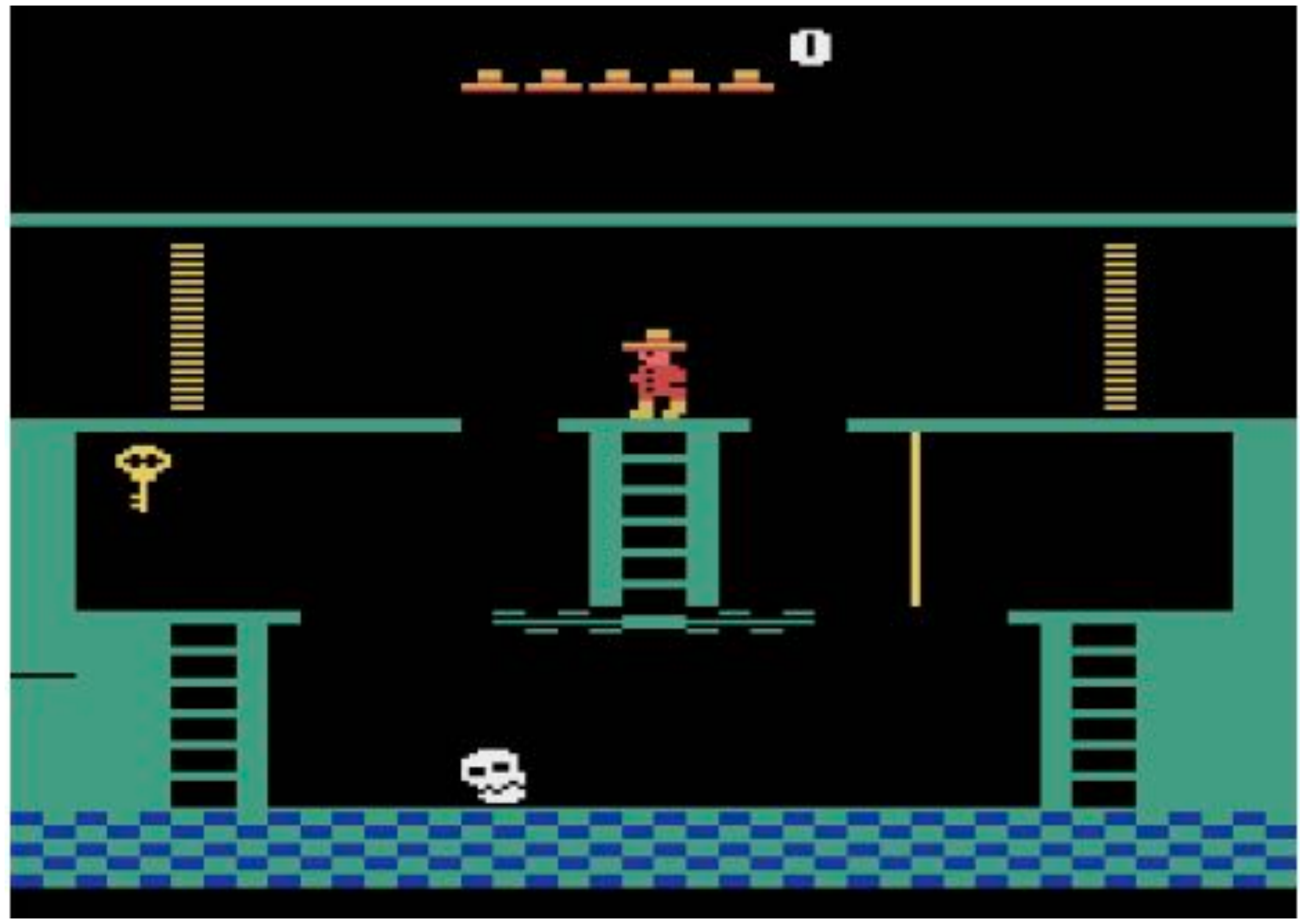
The story so far... reimplement learning environments as “batch simulators" for 100x Speedups

Isaac Gym: GPU Physics



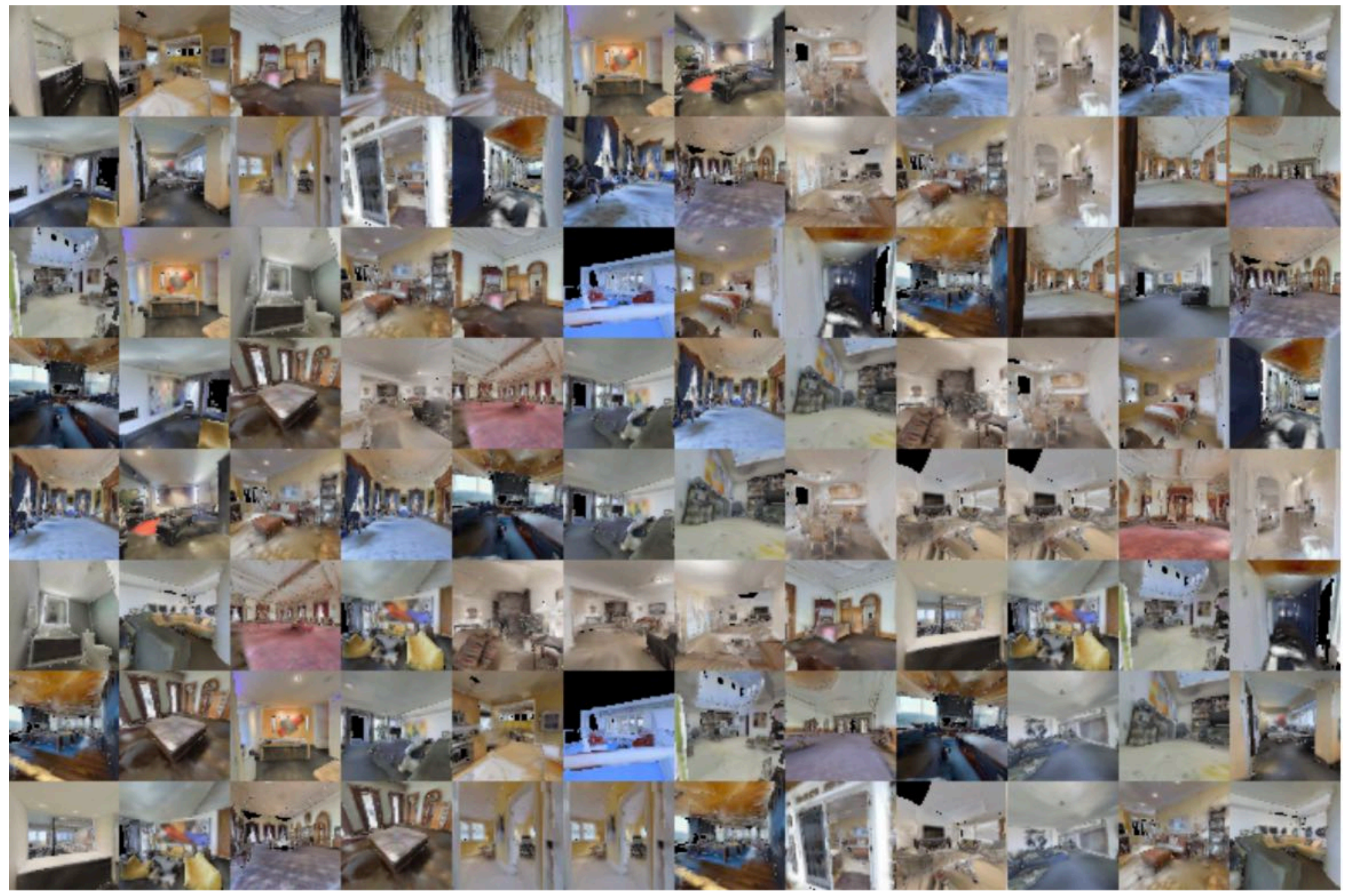
~10000 envs per GPU

CuLE: Atari



~5000 envs per GPU

BPS3D: Home Navigation



~1000 envs per GPU

MuJoCo MJX

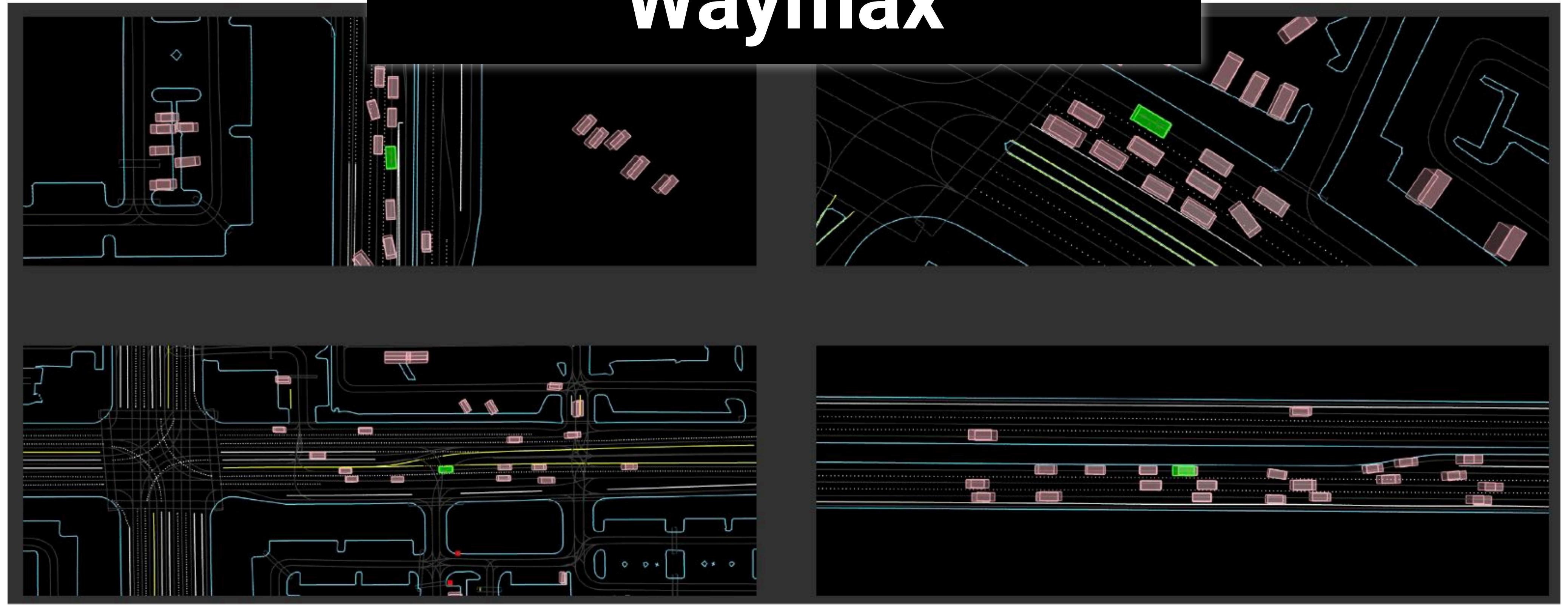
MuJoCo XLA (MJX)

Starting with version 3.0.0, MuJoCo includes MuJoCo XLA (MJX) under the mjx directory. MJX allows MuJoCo to run on compute hardware supported by the XLA compiler via the JAX framework. MJX runs on a all platforms supported by JAX: Nvidia and AMD GPUs, Apple Silicon, and Google Cloud TPUs.

The MJX API is consistent with the main simulation functions in the MuJoCo API, although it is currently missing some features. While the API documentation is applicable to both libraries, we indicate features unsupported by MJX in the notes below.

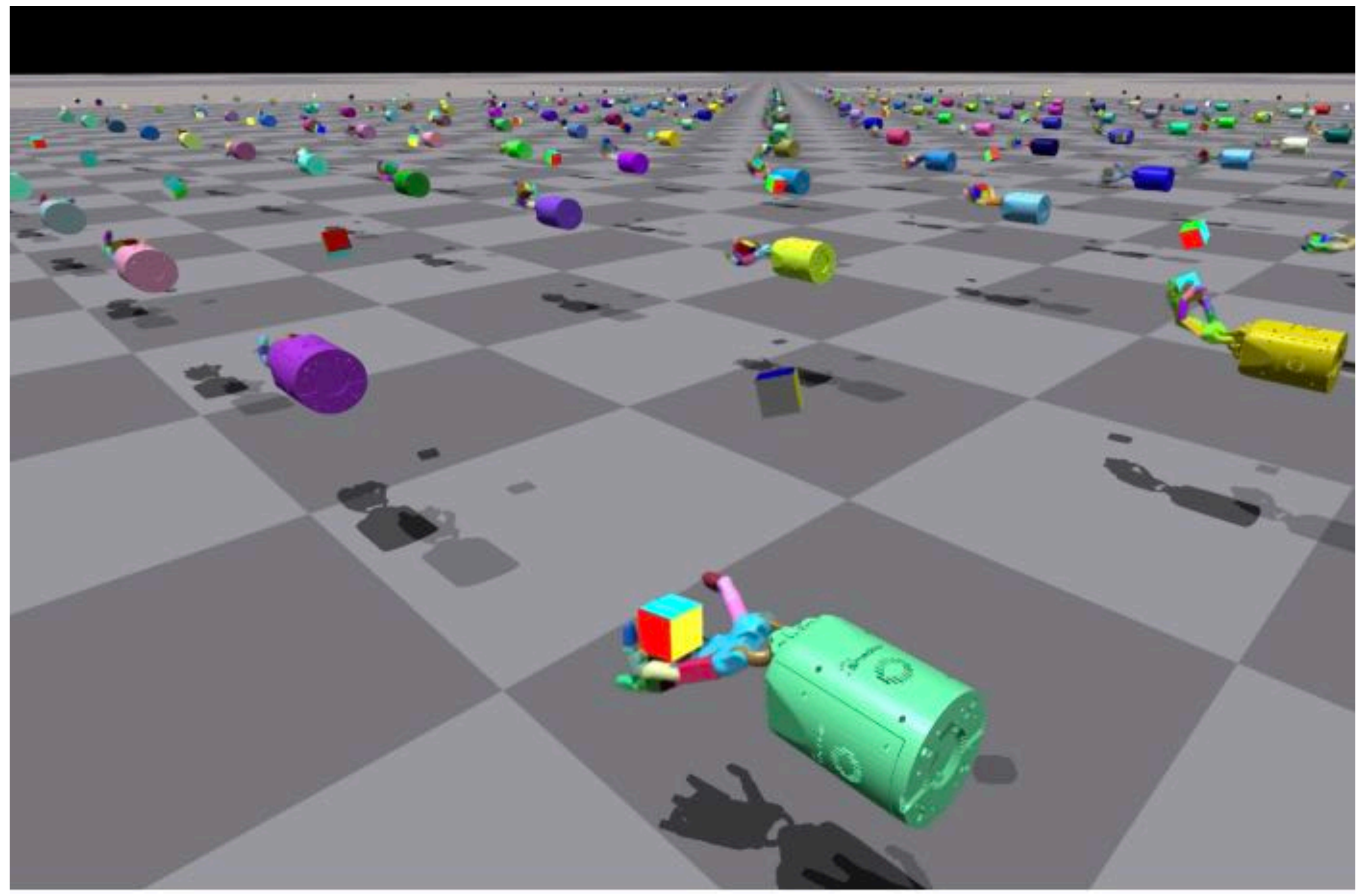
MJX is distributed as a separate package called `mujoco-mjx` on PyPI. Although it depends on the main `mujoco` package for model compilation and visualization, it is a re-implementation of MuJoCo

Waymax

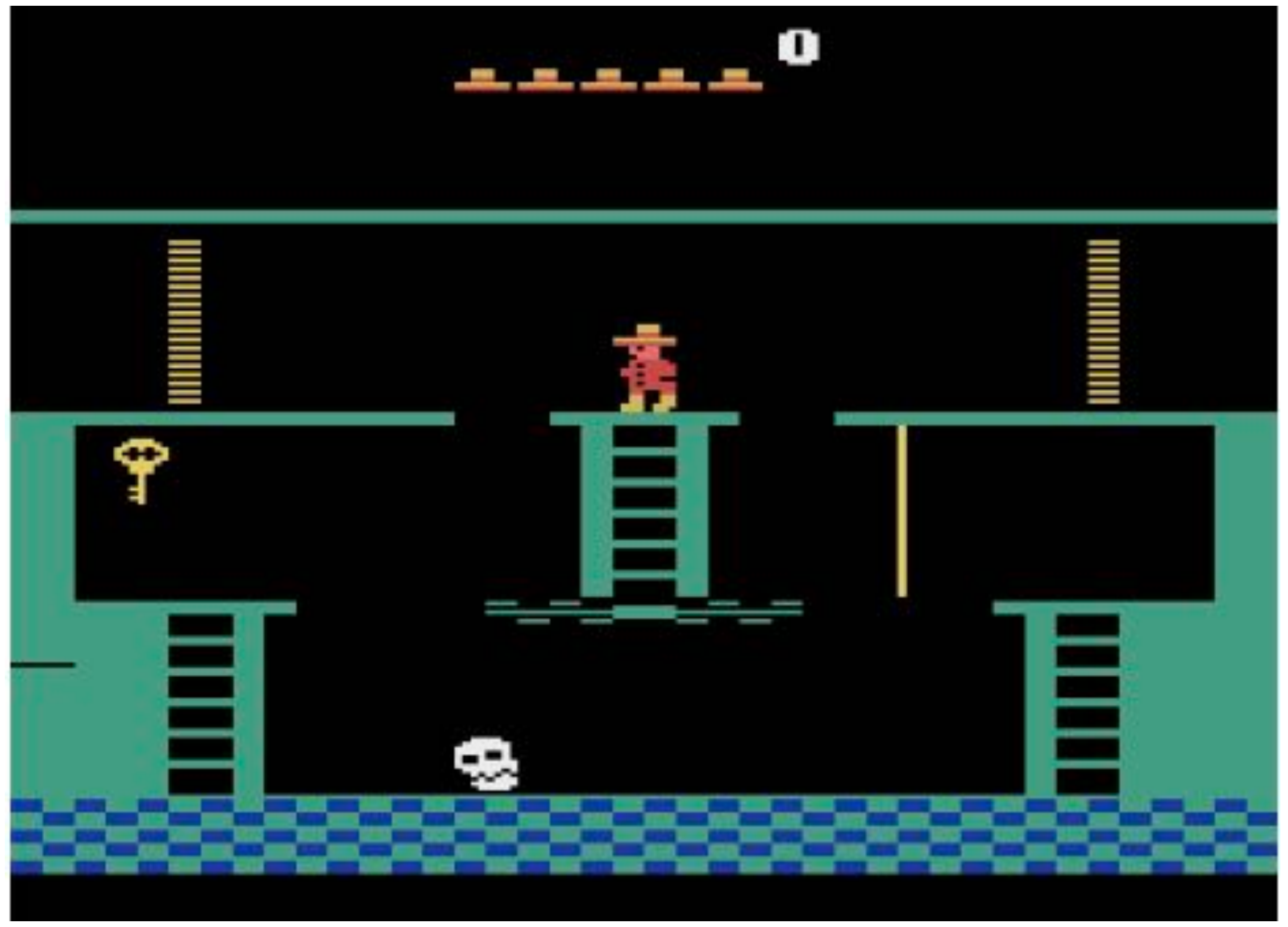


What about training new kinds of agents for new tasks?

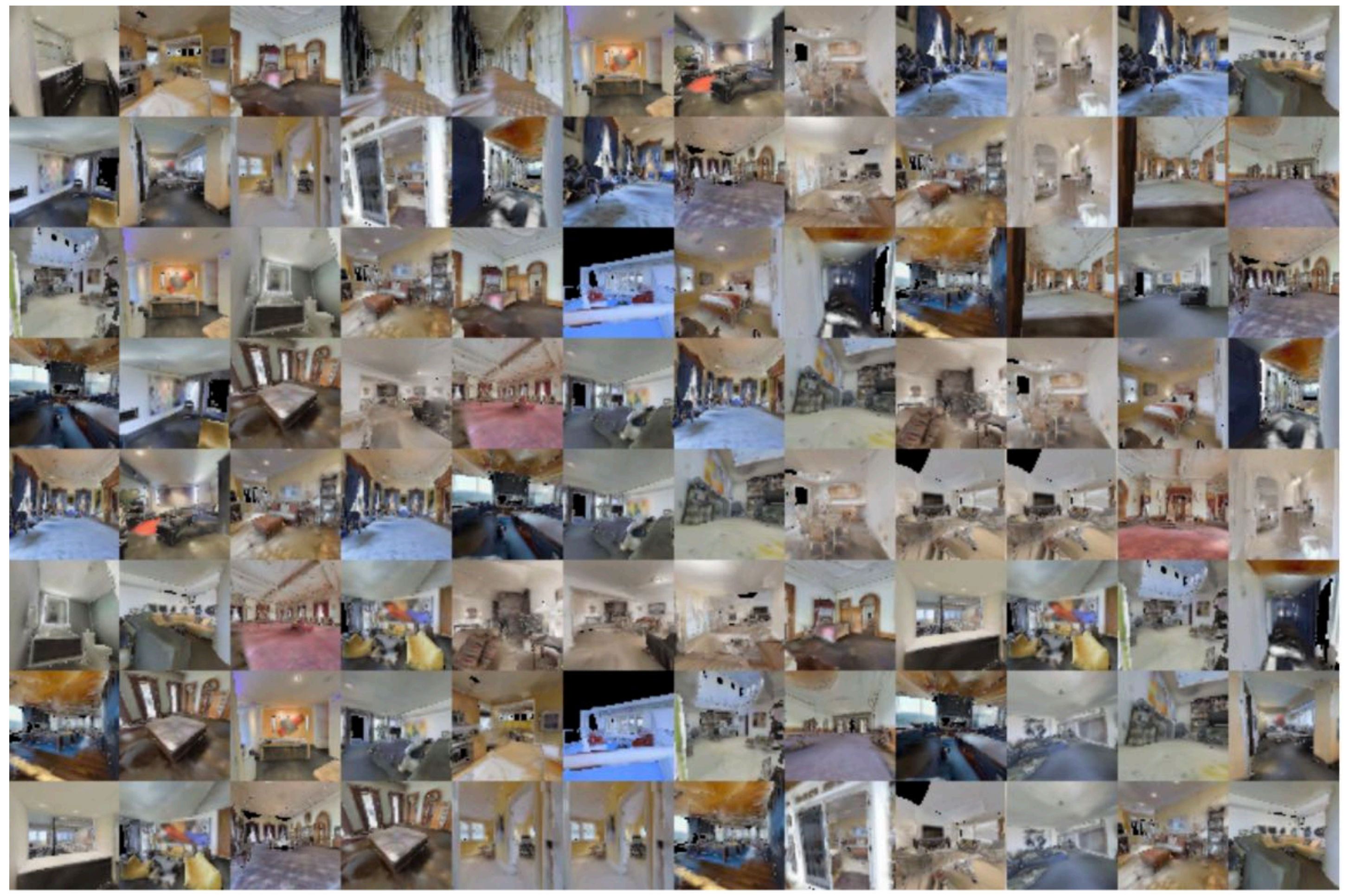
GPU Physics



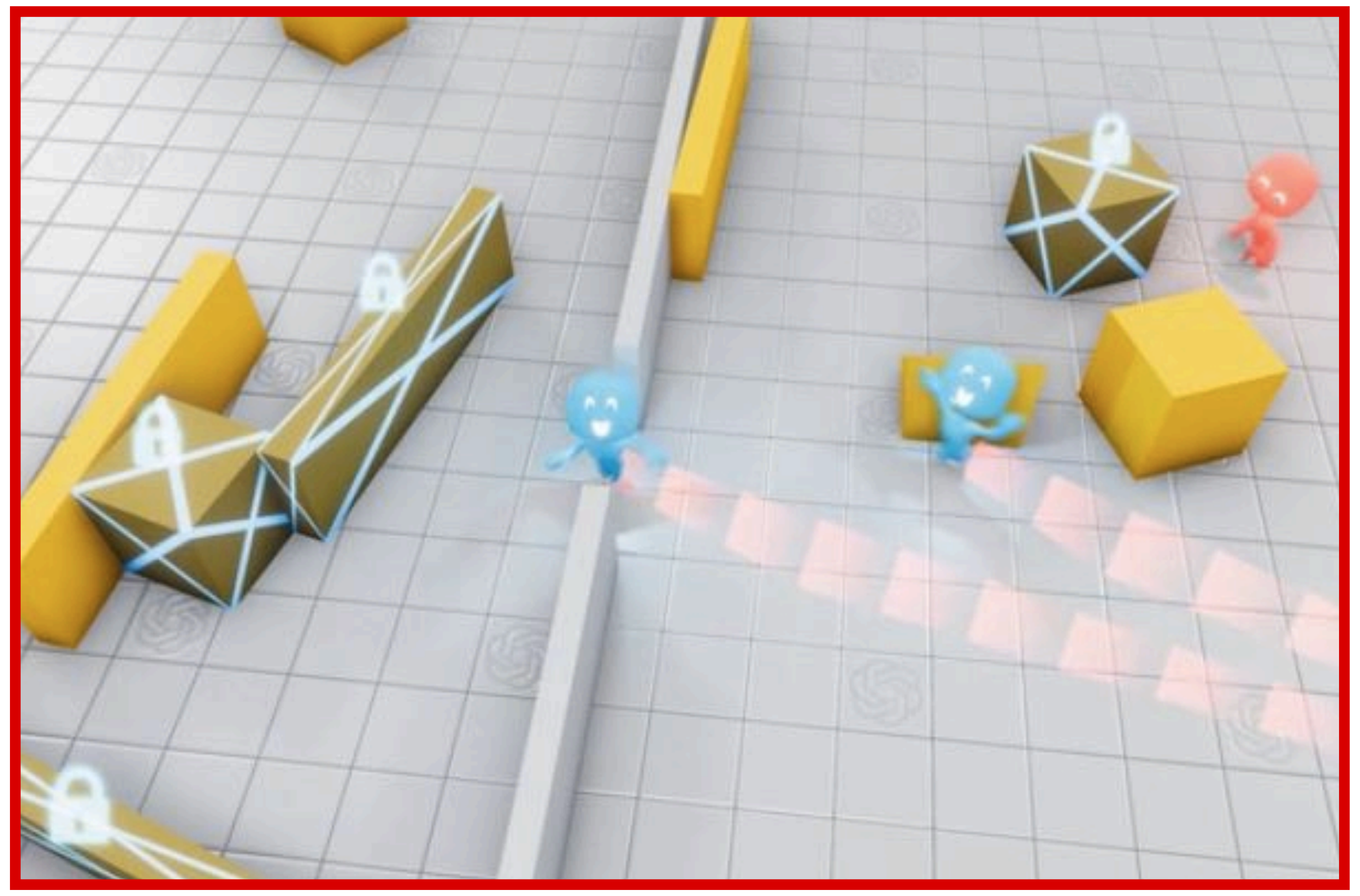
Atari



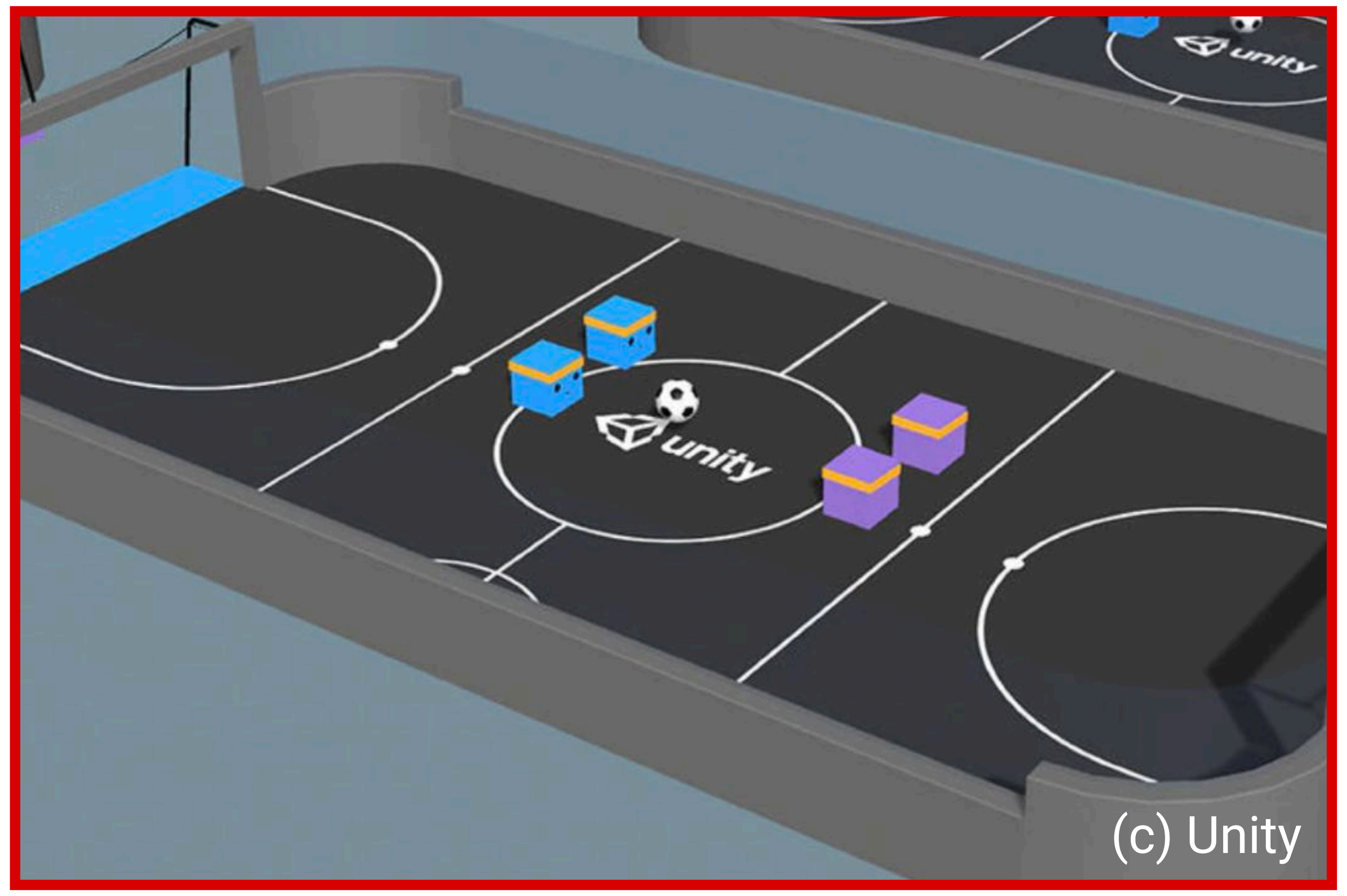
Floorplan Navigation



Novel Research Task



New Types of Games



We need a "game engine" for building batch simulators!

Maximize throughput: millions of sim steps/sec for simple 3D environments
(When running many environments in parallel)

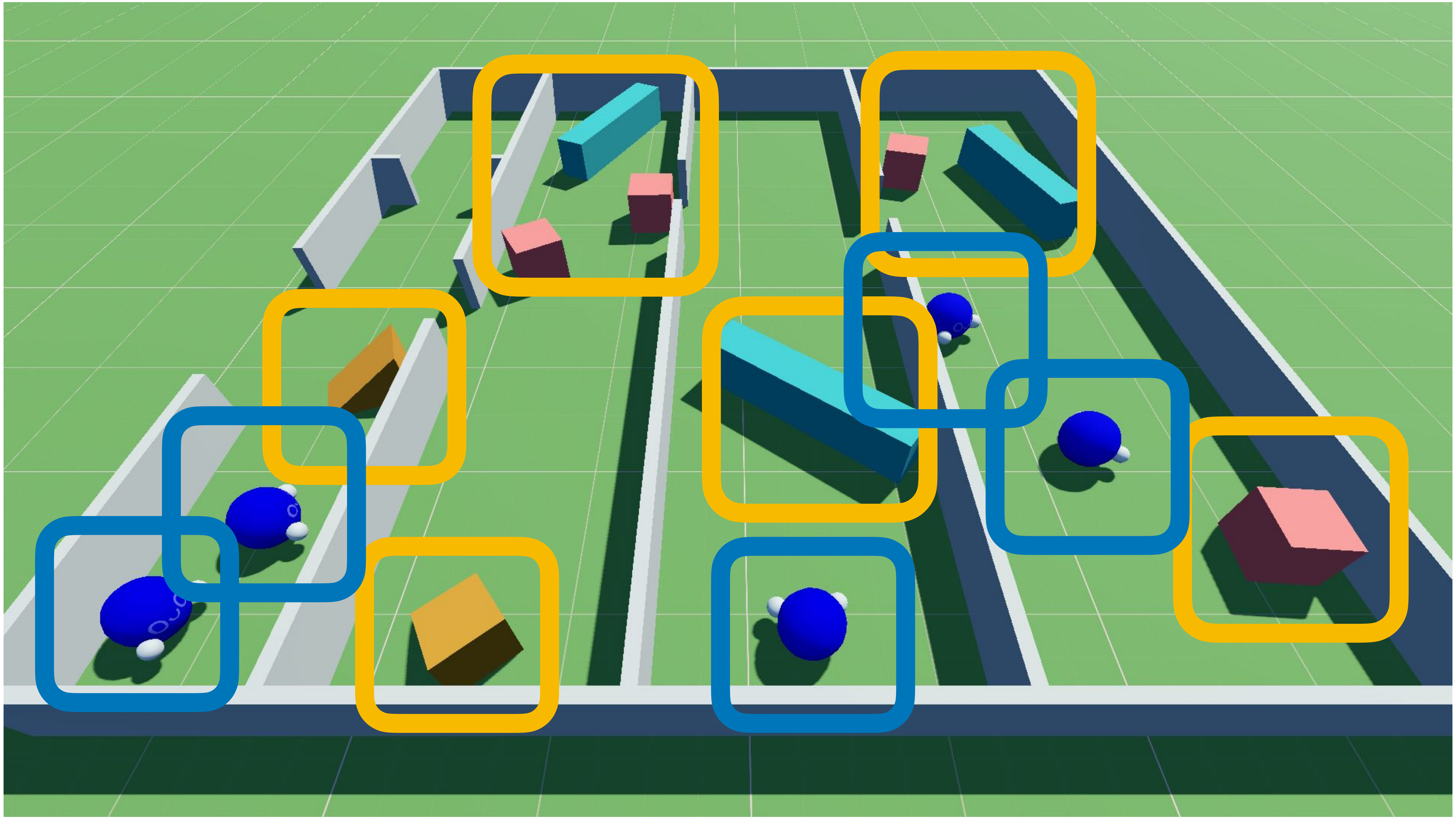
Programmable: environment creators should be able to author diverse set of worlds, define custom world rules/behavior

Productive: quickly be able to create novel worlds

Madrona

- **Key idea: the Entity Component System (ECS) architecture, which is a programming structure used in some games today, provides useful structure needed to build a many-world game engine on the GPU**

Entities (ECS)



Components (ECS)

Agents						
Id	EnvID	Pos	Bbox	Action	Reward	
12	0	[0,0,.5]	{min...	LEFT	0.1	...
32	0	[2,1,0]	{min...	FWD	-0.1	...
51	0	[1.5,0,1]	{min...	FWD	-2.5	...

Obstacles				
Id	EnvID	Pos	Bbox	
13	0	[0.5,0,.5]	{min...	...
72	0	[0,1,3]	{min...	...
61	0	[1,1,2]	{min...	...

Batch ECS: Store Data Across All Environments in Unified Tables in GPU Memory

Agents

Id	EnvID	Pos	Bbox	Action	Reward	
12	0	[0,0,.5]	{min...	LEFT	0.1	...
32	0	[2,1,0]	{min...	FWD	-0.1	...
51	0	[1.5,0,1]	{min...	FWD	-2.5	...
62	1	[1.5,0.5,1]	{min...	RIGHT	0.3	...
65	1	[-0.5,0,0]	{min...	RIGHT	0.5	...
20	2	[0.5,1,1]	{min...	LEFT	1.5	...
23	2	[-1.5,0,0]	{min...	LEFT	10.5	...
41	2	[0.5,1,0]	{min...	BACK	-10	...

Obstacles

Id	EnvID	Pos	Bbox	
13	0	[0.5,0,.5]	{min...	...
72	0	[0,1,3]	{min...	...
61	0	[1,1,2]	{min...	...
49	1	[0.5,1,2]	{min...	...
70	1	[-0.5,1,0]	{min...	...
33	2	[1.5,0,2.5]	{min...	...
81	3	[2.5,0.5,2]	{min...	...
11	3	[1.5,0.5,2]	{min...	...

Systems (ECS)

Agents

Id	EnvID	Pos	Bbox	Action	Reward	
12	0	[0,0,.5]	{min...	LEFT	0.1	...
32	1	[2,1,0]	{min...	FWD	-0.1	...
51	2	[1.5,0,1]	{min...	FWD	2.5	...
22	2	[2.5,0,1.5]	{min...	RIGHT	1.1	...

Obstacles

Id	EnvID	Pos	Bbox	
13	0	[0.5,0,.5]	{min...	...
72	1	[0,1,3]	{min...	...
61	1	[1,1,2]	{min...	...
25	2	[1.5,1,2.5]	{min...	...

ProcessActions

Pos, Action

Collisions

Id, Pos, Bbox

ComputeRewards

Pos, Reward

Systems (ECS)

Agents

Id	EnvID	Pos	Bbox	Action	Reward	
12	GPU Thread	[0,0,.5]	{min...	LEFT	0.1	...
32	GPU Thread	[2,1,0]	{min...	FWD	-0.1	...
51	GPU Thread	[1.5,0,1]	{min...	FWD	2.5	...
22	GPU Thread	[2.5,0,1.5]	{min...	RIGHT	1.1	...

Obstacles

Id	EnvID	Pos	Bbox	
13	0	[0.5,0,.5]	{min...	...
72	1	[0,1,3]	{min...	...
61	1	[1,1,2]	{min...	...
25	2	[1.5,1,2.5]	{min...	...

ProcessActions

Pos, Action

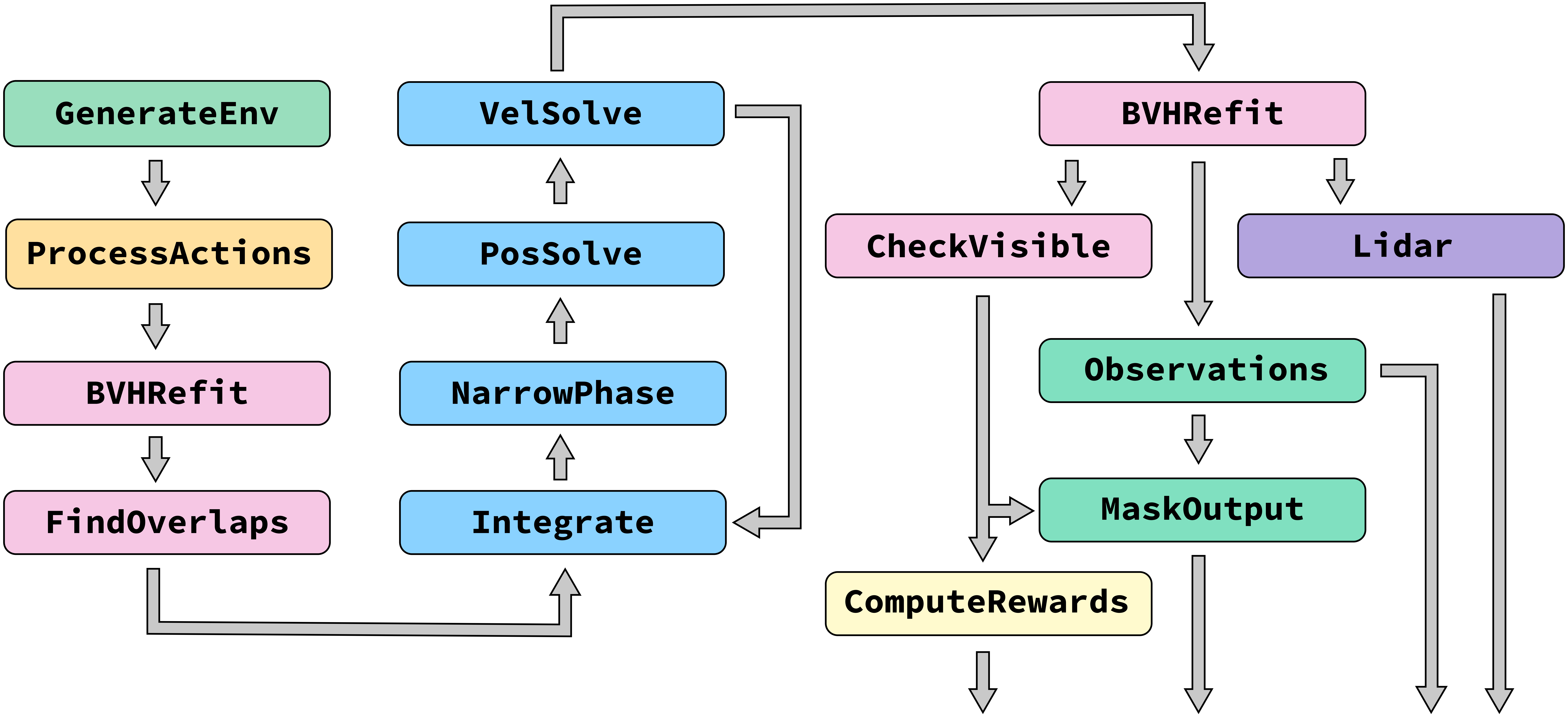
Collisions

Id, Pos, Bbox

ComputeRewards

Pos, Reward

ECS Systems Combined into Task Graph and Executed in Parallel on the GPU



Scheduling the ECS on the GPU

Fully GPU-Driven Scheduling Challenges

Dynamic GPU-Driven Memory Allocation:

- Game logic needs to create entities at runtime (during a simulation step)
- Entity lifetimes can vary wildly (<1 frame to hundreds)

How to Efficiently Execute Task Graph Given Dynamic Workload Each Frame?

- Task graph can contain > 100 nodes!
- # of entities matching each system may depend on prior nodes in task graph

Growable ECS Table Storage By (Ab)using GPU Virtual Memory Support

Obstacles

Id	EnvID	Pos	Bbox	
13	0	[0.5,0,.5]	{min...	...
72	1	[0,1,3]	{min...	...
61	1	[1,1,2]	{min...	...
25	2	[1.5,1,2.5]	{min...	...

Unbacked
Virtual Memory

New Entities
Created

Obstacles

Id	EnvID	Pos	Bbox	
13	0	[0.5,0,.5]	{min...	...
72	1	[0,1,3]	{min...	...
61	1	[1,1,2]	{min...	...
25	2	[1.5,1,2.5]	{min...	...
27	2	[1.5,1,2.5]	{min...	...
28	2	[1,0,3.5]	{min...	...

Unbacked
Virtual Memory

Achieving Dynamic GPU Memory Allocation & Improving Coherence Using Parallel Radix Sort

Id	EnvID	Pos	
12	0	[0,0,.5]	...
32	1	[2,1,0]	...
51	2	[1.5,0,1]	...
22	2	[2.5,0,1.5]	...

ECS Systems Execute

Id	EnvID	Pos	
12	0	[0,0,.5]	...
32	X	[2,1,0]	...
51	X	[1.5,0,1]	...
22	2	[2.5,0,1.5]	...
23	0	[2,1,1.5]	...
24	1	[0,1,2.5]	...

Radix Sort

Id	EnvID	Pos	
12	0	[0,0,.5]	...
23	0	[2,1,1.5]	...
24	1	[0,1,2.5]	...
22	2	[2.5,0,1.5]	...
32	X	[2,1,0]	...
51	X	[1.5,0,1]	...

Reclaim Memory

Id	EnvID	Pos	
12	0	[0,0,.5]	...
23	0	[2,1,1.5]	...
24	1	[0,1,2.5]	...
22	2	[2.5,0,1.5]	...

Low-Overhead Dynamic Task Graph Execution Using Persistent Megakernel Design

Megakernel(taskgraph, envs):

```
while true:
    system_id, invocation_id = getNext(taskgraph)
    switch system_id:
        case 0: physicsSystemEntry(envs, invocation_id)
        case 1: ProcessActionEntry(envs, invocation_id)
        case 2: visibilitySystemEntry(envs, invocation_id)
        case -1: break

    threadFinished(taskgraph)
```

getNext(taskgraph):

```
node = taskgraph.currentNode()
if node.currentInvocation < node.numInvocations:
  return node.systemID, node.currentInvocation++


return taskgraph.advanceNode()
```

Example ECS System: Mapping GPU Threads to Hide & Seek ProcessAction

ProcessAction(env, id, pos, force, team, action):

```
if action.type == MOVE:
    force = computeMovementForce(action.dir)
if action.type == LOCK:
    hit_obj = raycastForward(env, pos)
    if hit_obj:
        ...
```

ProcessActionEntry(envs, ecs_state, gpu_thread_idx):

```
ids, world_ids, positions, forces, teams, actions =
   ecs_state.getColumns<Id, EnvID, Pos, Force, Team, Action>()

row = gpu_thread_idx

env_id = env_ids[row]
if env_id is not valid:
  return

ProcessAction(envs[env_id], ids[row], positions[row],
         forces[row], teams[row], actions[row])
```

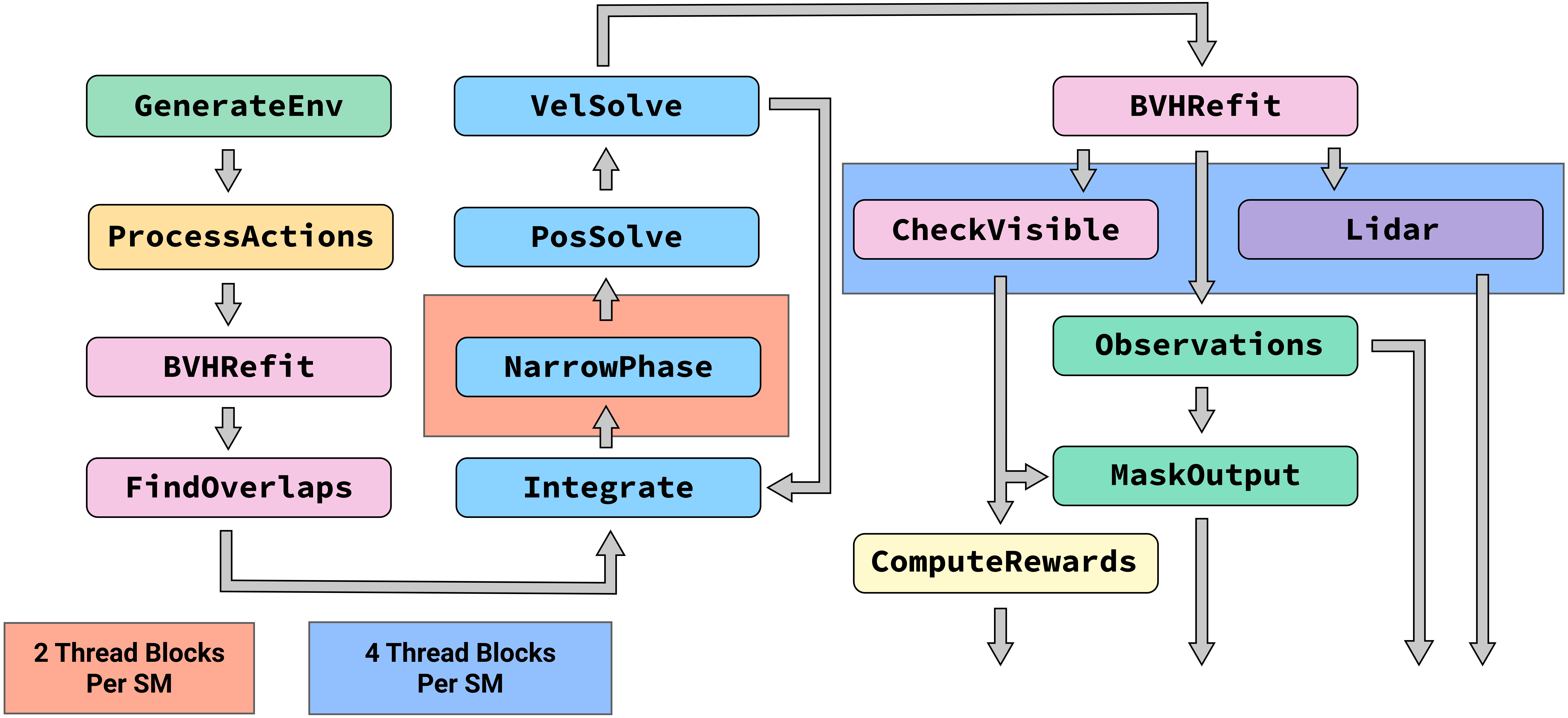
Id	EnvID	Pos	
12	0	[0,0,.5]	...
32	X	[2,1,0]	...
51	X	[1.5,0,1]	...
22	2	[2.5,0,1.5]	...
23	0	[2,1,1.5]	...
24	1	[0,1,2.5]	...

Mitigating Megakernel Inefficiencies Using Profile-Guided Optimization

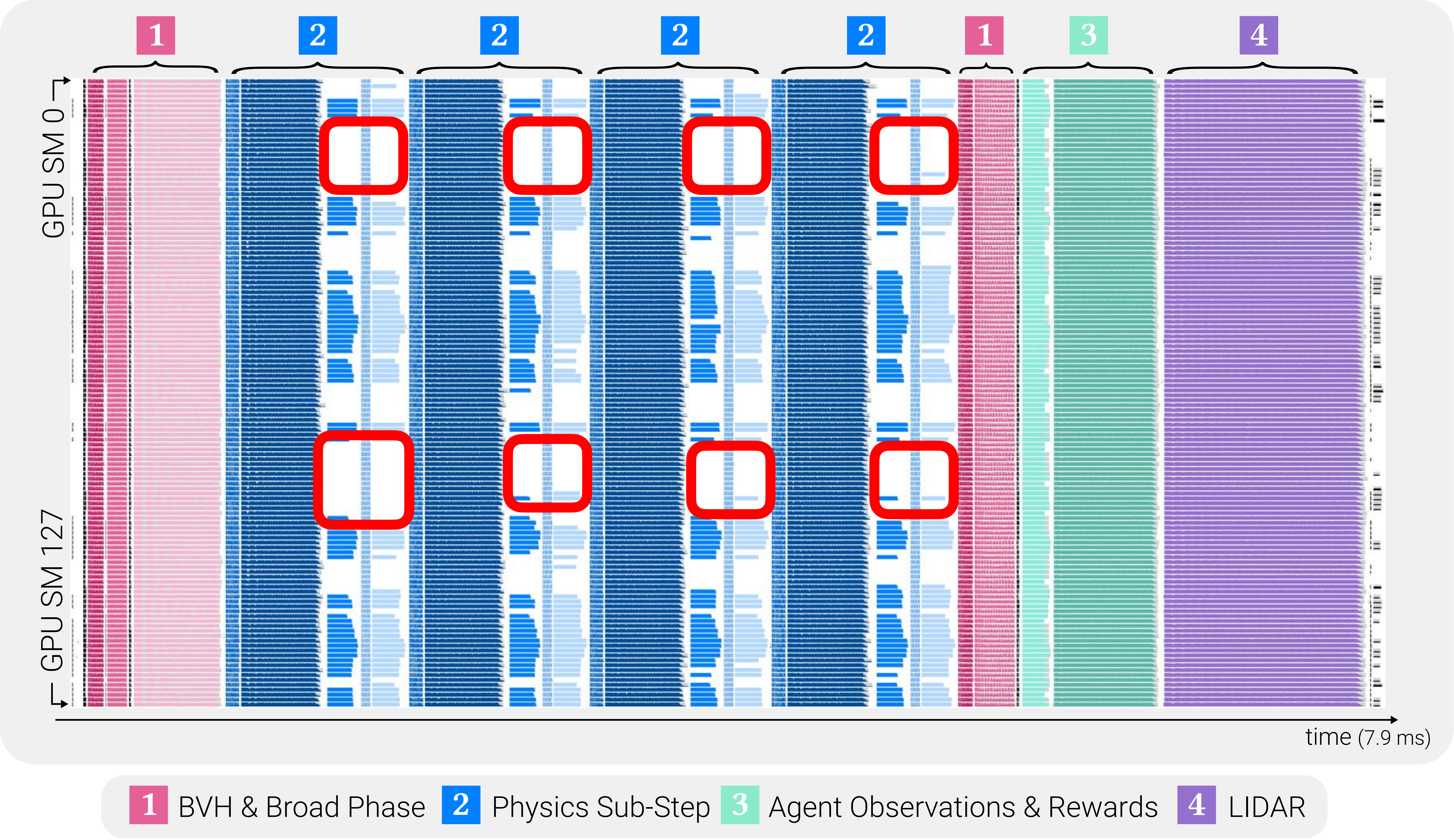
- **Megakernel Implies One-Size-Fits-All Register Allocation**
  - Observation: Can afford more than 1 kernel launch per batched simulation step

- **Profile-Guided Optimization:** Empirically test performance of each system with different register allocations & choose best!
  - Negligible cost in a 100 million step training run

Mitigating Megakernel Inefficiencies Using Profile-Guided Optimization



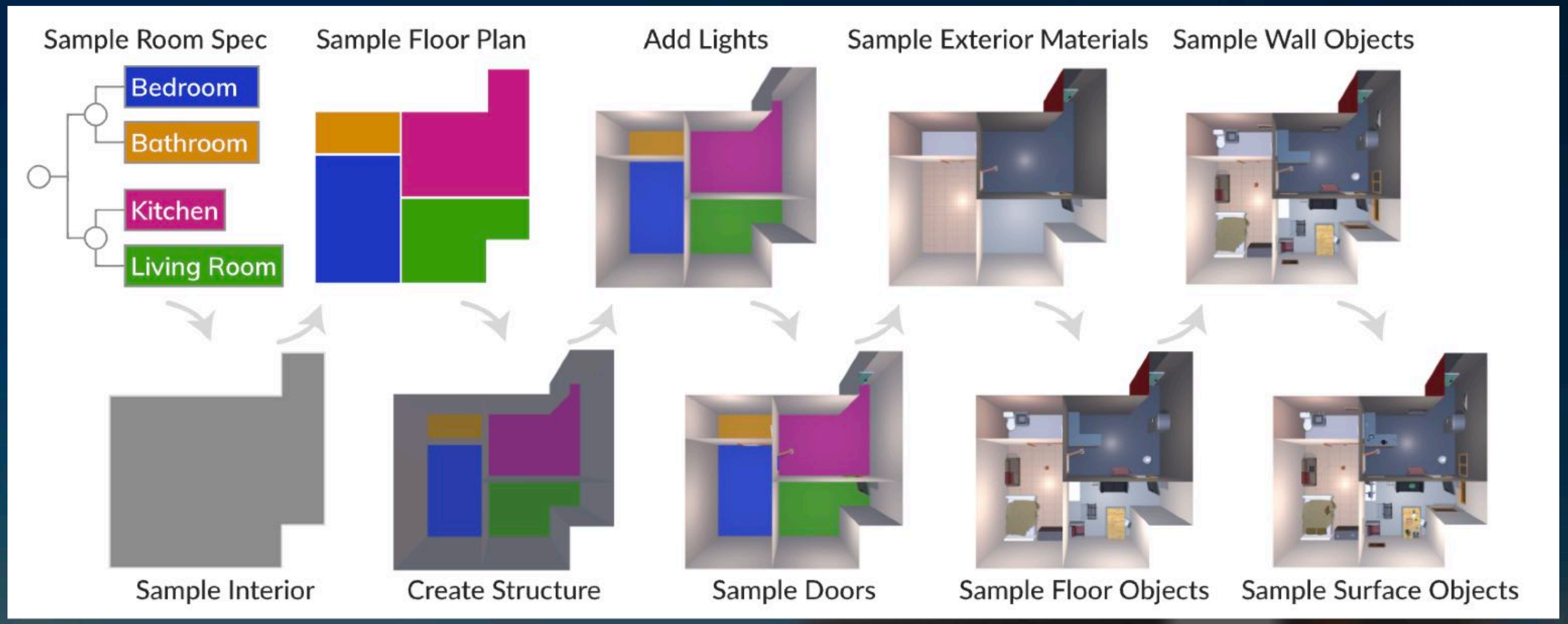
Performance Analysis: One Step Across 16K Environments



Procedural content creation

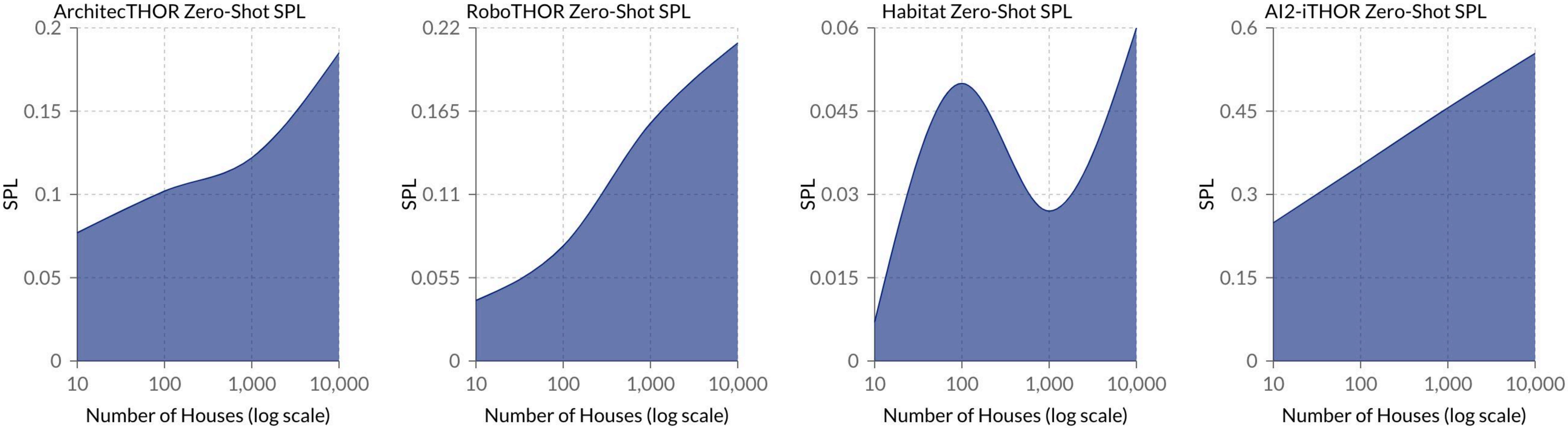
Significant value in diversity of scenes

Example: ProcTHOR



Procedurally generated floorpans, furniture arrangements, random material assigmments, etc.

Greater diversity of scenes wins



Better off training on a large number of highly diverse scenes, than a small number of photorealistic ones

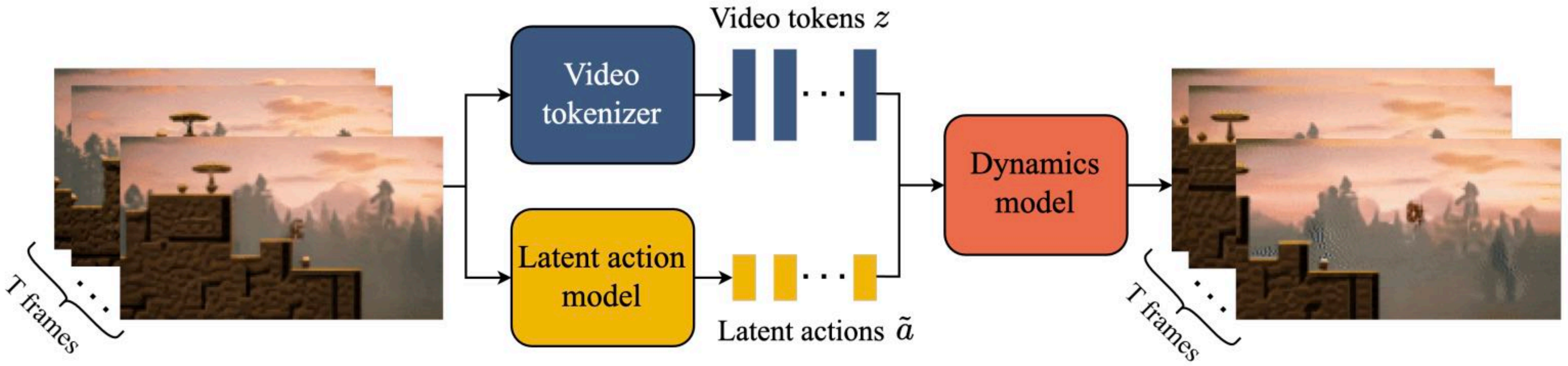
Generative AI as a means to generate world simulation output

Enhancing CG images to look like real-world images using image-to-image transfer

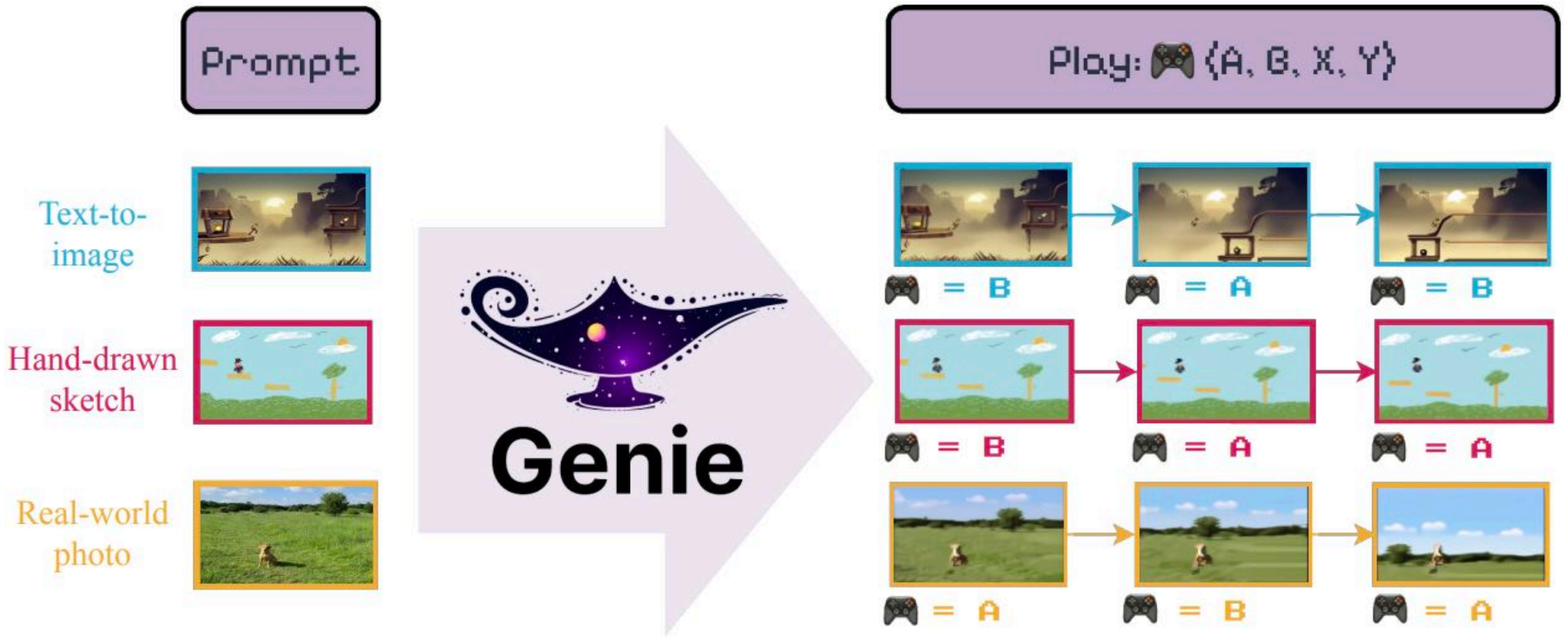
Modifying real-world images to create novel situations

Genie

- Key idea: learn a world simulator from videos of video game play
  - From video, learn latency user actions, and dynamics model that steps work given (current state, action)



  - Then at "test time" given a novel world state (perhaps one generated from a prompt), and given user input, time step the novel world forward in time



Next time: the great debate

- Class debate: you have to choose a side!

- What's the right way to build a world simulator?
  - Like a "game engine" : humans model and build a simulator
  - Data driven: just learn it from big data

- What are the pros/cons?

- In what situations might one be preferable?