ScheduleStream: Temporal Planning with Samplers for GPU-Accelerated Multi-Arm Task and Motion Planning & Scheduling

Caelan Garrett, Fabio Ramos
NVIDIA Research

https://schedulestream.github.io/
https://github.com/NVlabs/ScheduleStream
https://youtu.be/0LyTPmAXaQY

Task and Motion Planning (TAMP)

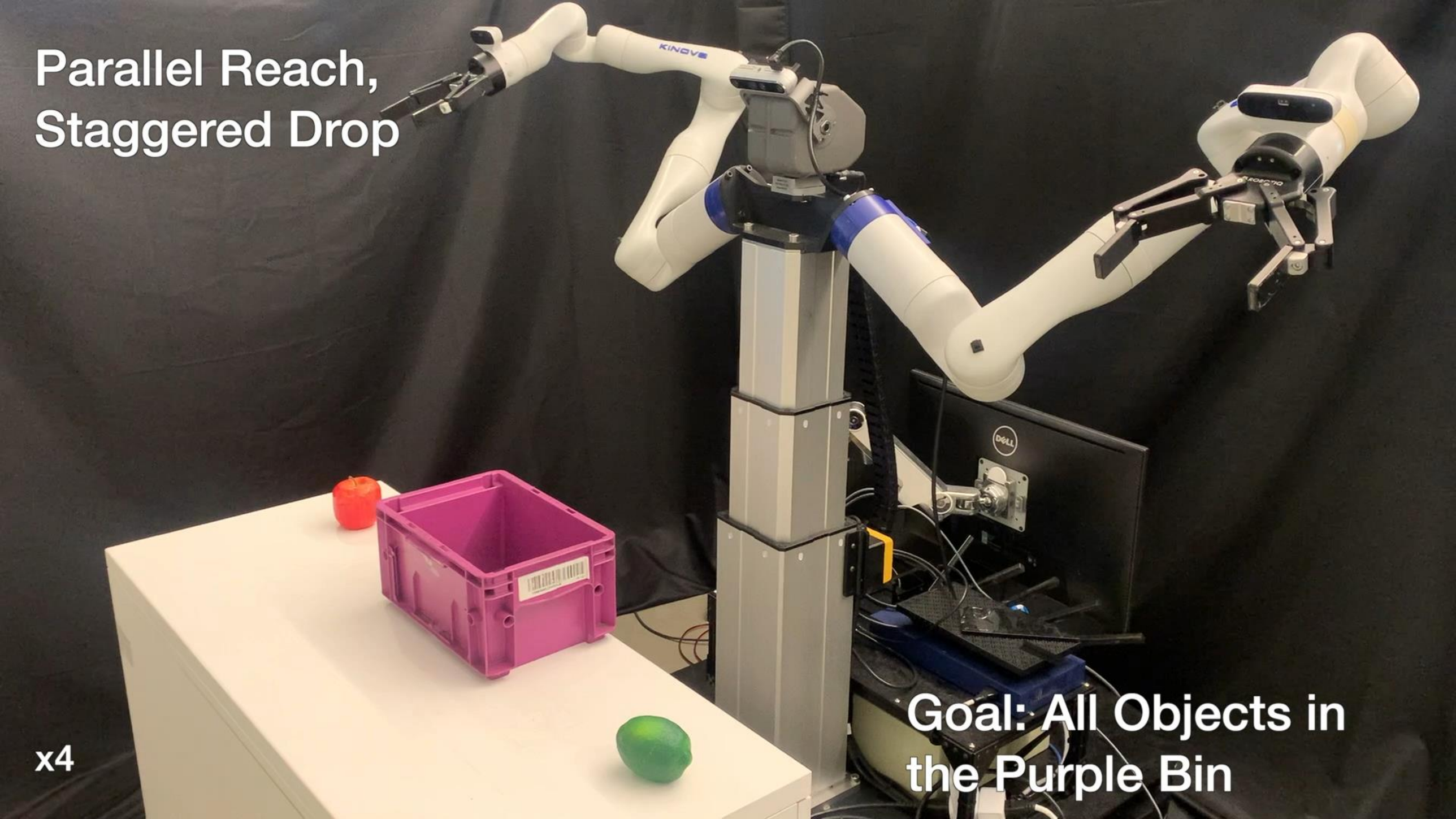
Manipulation Planning with Changes in Contact

- Motion planners (e.g. PRM, RRT, cuRobo) address continuous change in **robot state**
- To **manipulate** the world, must also reason about:
  1. Change in **object state**
  2. Change in robot-object **discrete contacts**
- TAMP: integrated **discrete** (“task”) and **continuous** (“motion”) planning
- **Challenges**: computationally expensive, single robot, inflexible frameworks

Our Contributions:

1. **Multi-Arm TAMP**: temporal planning & scheduling for concurrent robot motion
2. **GPU-Parallelized TAMP**: search, sampling, & optimization w/ hardware acceleration

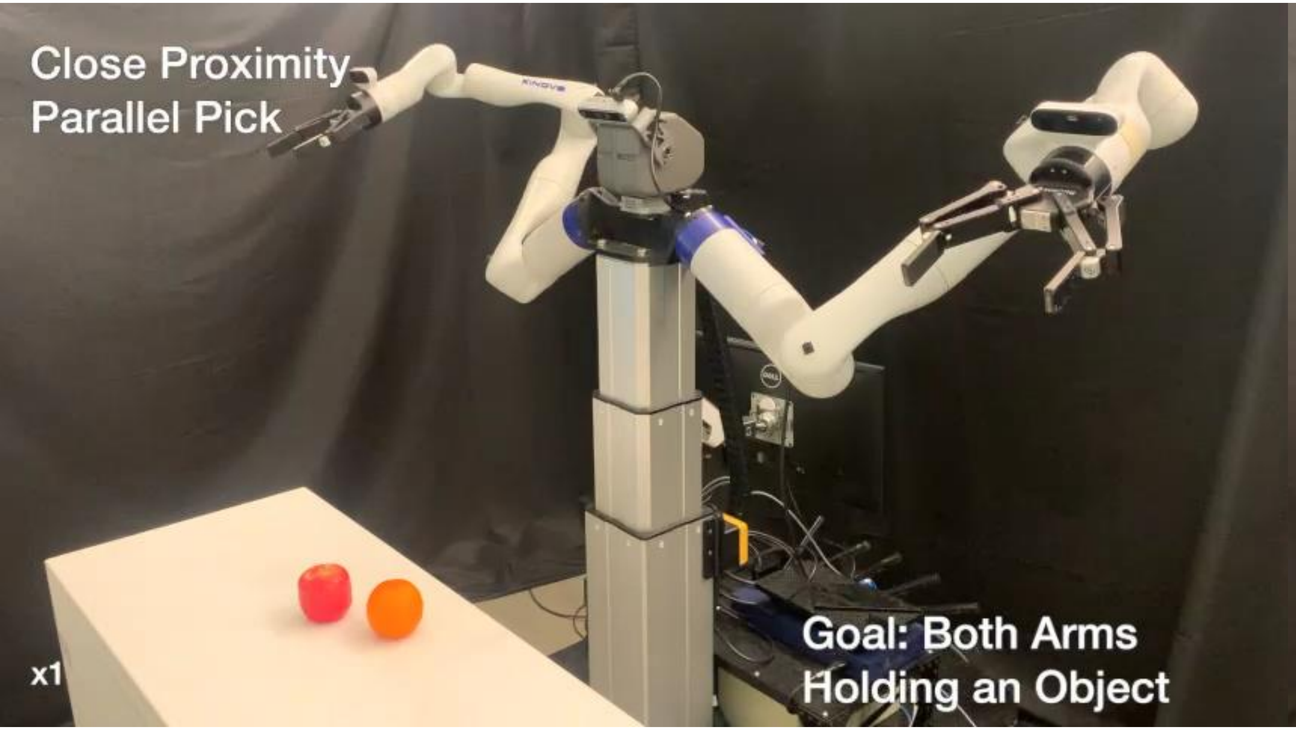
See https://youtu.be/0LyTPmAXaQY for all **videos**


Parallel Reach,
Staggered Drop
x4
Goal: All Objects in
the Purple Bin

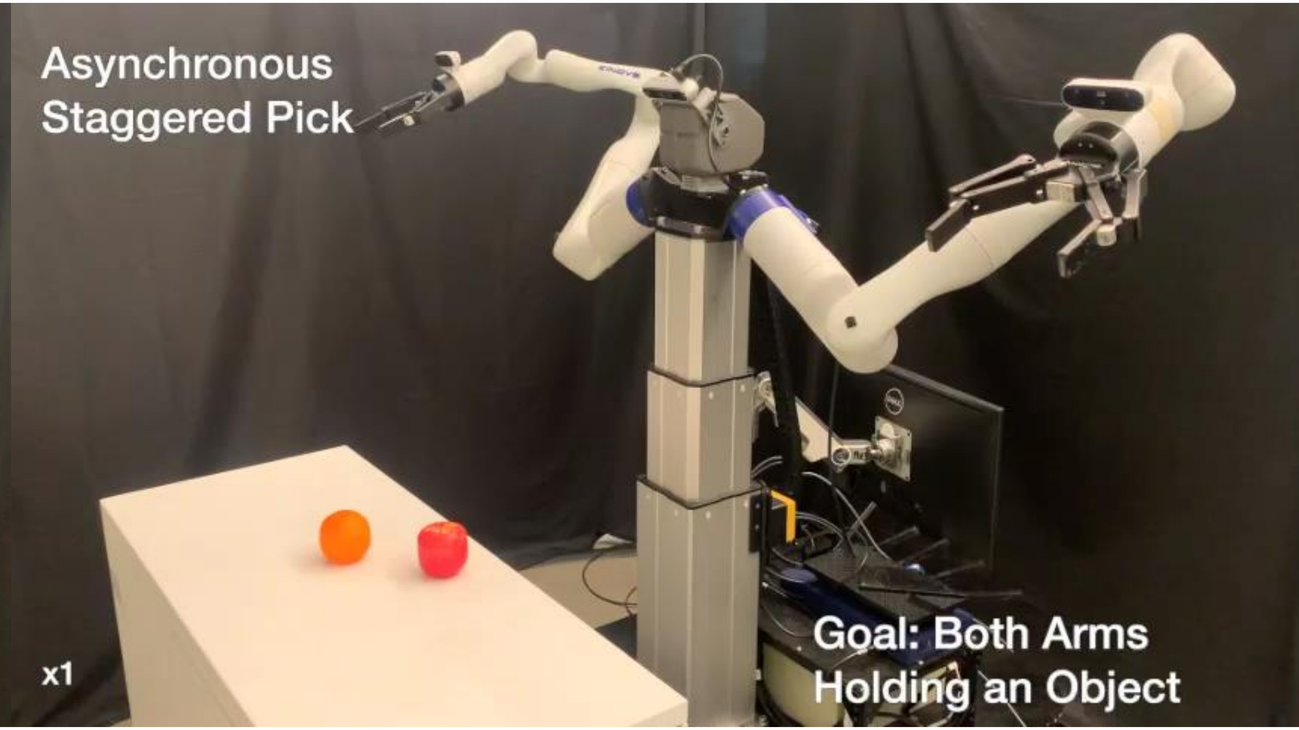
Task and Motion Planning & Scheduling

Planning Parallel yet Asynchronous Changes in Contact

- Existing TAMP approaches predominantly limited to **sequential** or synchronized plans
  - **Planning**: sequence of instantaneous actions. **Scheduling**: set of timed durative actions.
- **ScheduleStream**: planning language & algorithms, GPU-accelerated manipulation application



Parallel Arm Execution



Sequential Arm Execution

ScheduleStream Planning Language

Temporal Planning with Samplers (as Streams)

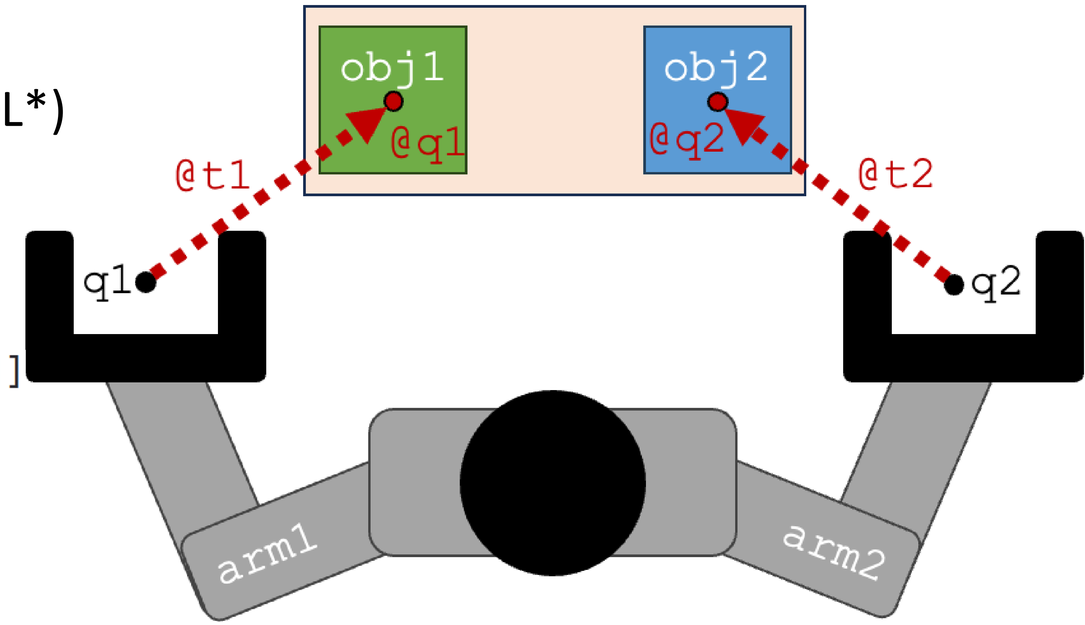
- Python constraints, costs, and samplers (Compared to PDDL*)
- Temporal dynamics (Compared to PDDLStream)

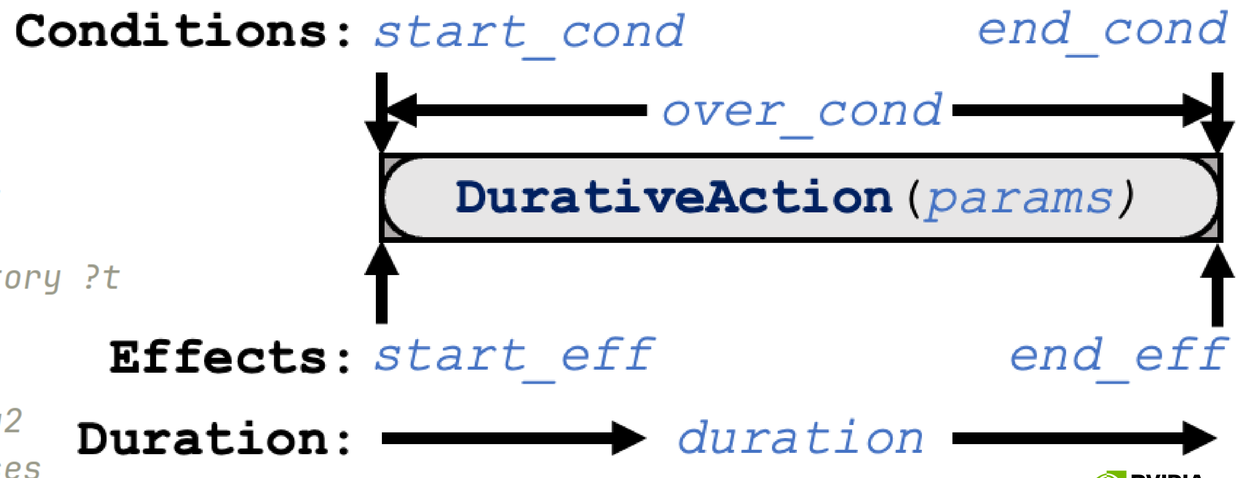
Python Multi-Arm Collision Constraints

```
arm_obj_collisions = [~ObjCollision("?t", Pose(o)) for o in [obj1, obj2]]
arm_arm_collisions = [~ArmCollision("?t", At(a)) for a in [arm1, arm2]]
```

Durative Move Action

```
move = DurativeAction(
    # robot arm ?arm, start conf ?q1, trajectory ?t, end conf ?q2
    parameters="?arm ?q1 ?t ?q2",
    start_condition=[
        Motion("?arm ?q1 ?t ?q2"),  # valid motion constraint
        At("?arm") == "?q1",  # ?arm is currently at start conf ?q1
    ],
    start_effect=[At("?arm") <= "?t"],  # ?arm is now along trajectory ?t
    over_condition=arm_obj_collisions + arm_arm_collisions
    end_condition=[],
    end_effect=[At("?arm") <= "?q2"],  # ?arm is now at end conf ?q2
    min_duration=Duration("?t"),  # Duration of trajectory ?t elapses
)
```



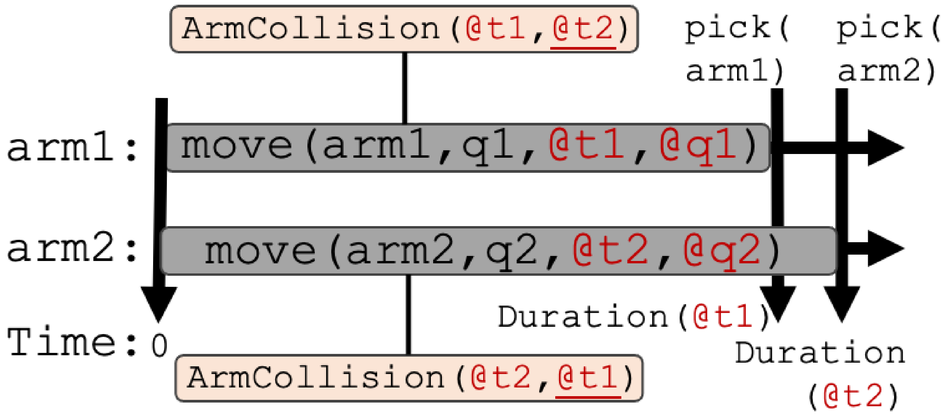


NVIDIA

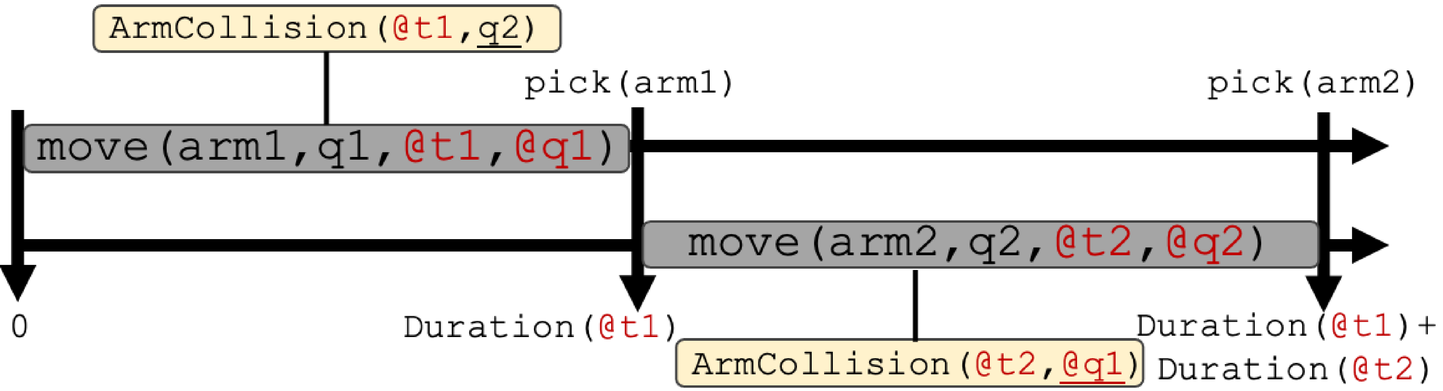
Arm-Arm Collisions Impose Temporal Mutex Constraints

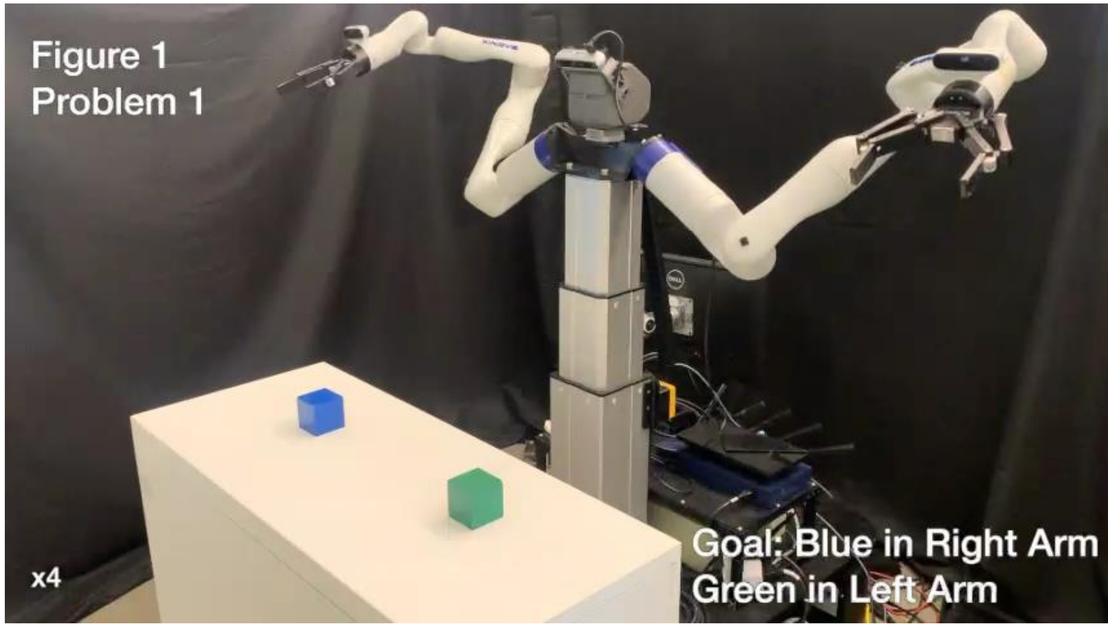
Collision and kinematic constraints limit **which** arms can do **what** manipulation **when**

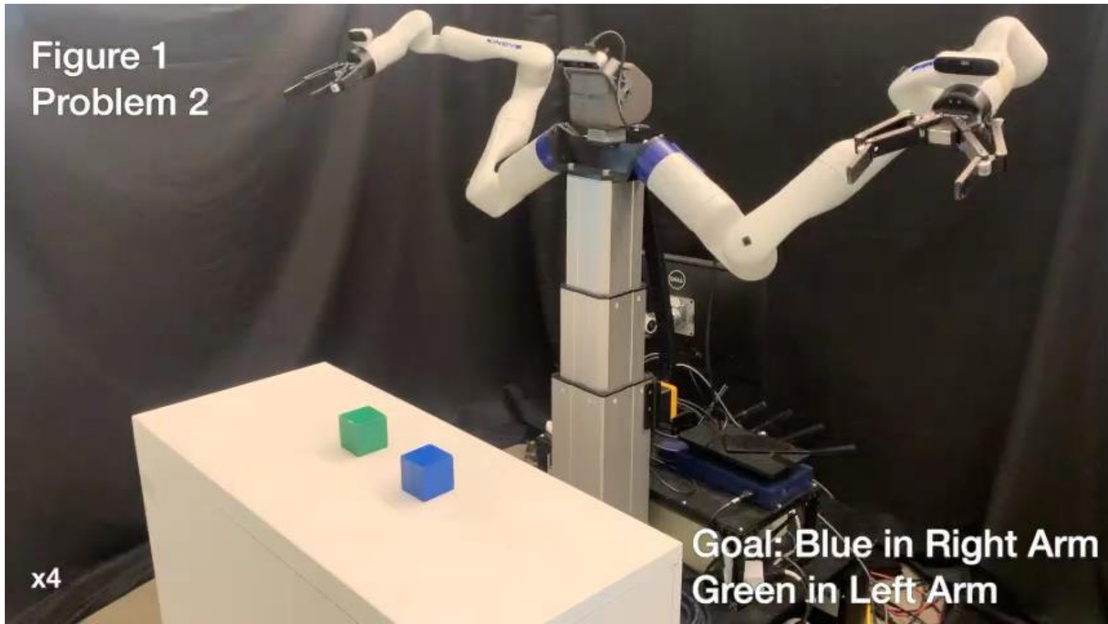
Problem 1: Safe *Parallel* Arm Schedule Exists



Problem 2: Collisions Force *Sequential* Arm Schedule

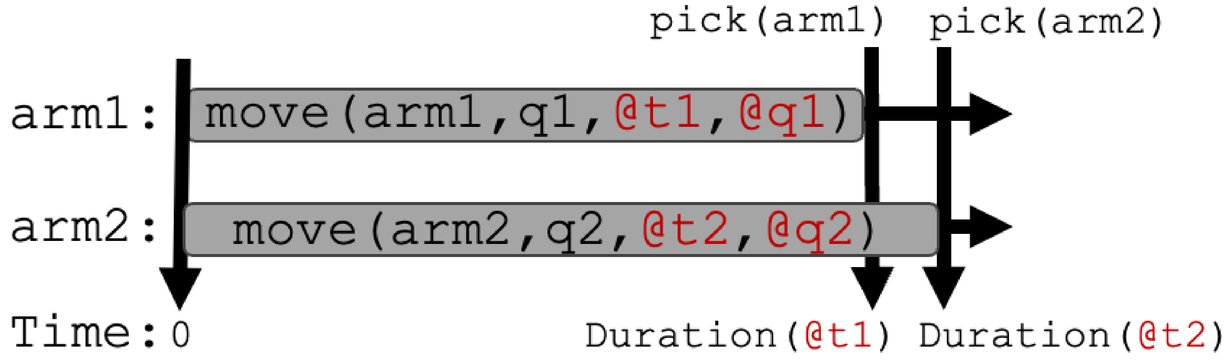






ScheduleStream Algorithms

Plan a Sequence of
Start and End Actions

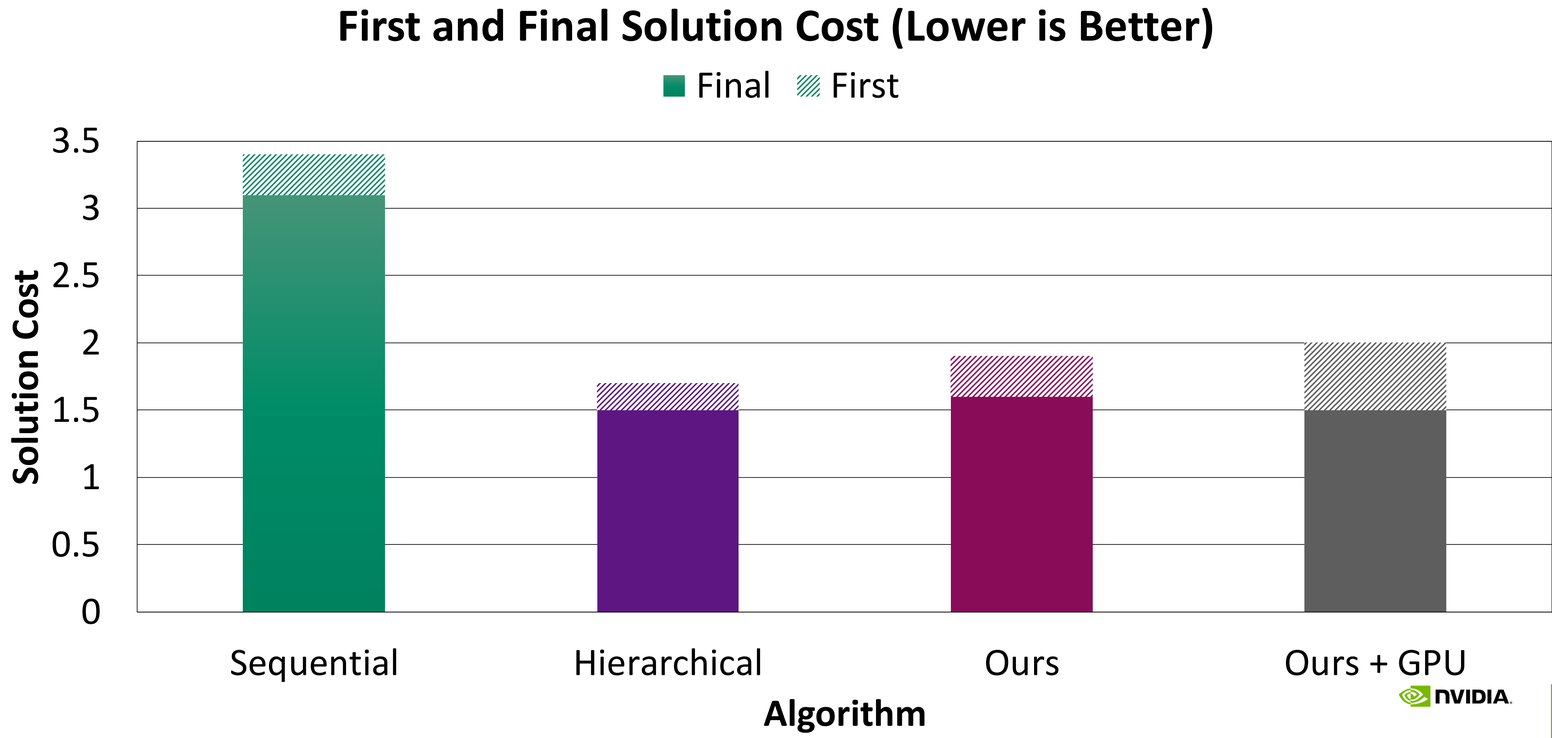


Time	Start & End Plan	State Sequence
0		[At(arm1)=q1,At(arm2)=q2,…]
	start_move(arm1,q1,@t1,@q1)	
0		[At(arm1)=@t1,At(arm2)=q2,…]
	start_move(arm2,q2,@t2,@q2)	
Duration(@t1)		[At(arm1)=@t1,At(arm2)=@t2,…]
	end_move(arm1,q1,@t1,@q2)	
Duration(@t2)		[At(arm1)=@q2,At(arm2)=@t2,…]
	end_move(arm2,q2,@t2,@q2)	
Duration(@t2)		[At(arm1)=@q2,At(arm2)=@q2,…]
	pick(arm1,@q1,obj1,@g1,p1)	
Duration(@t2)		[…,Holding(arm1)=obj1]
	pick(arm2,@q2,obj2,@g2,p2)	
Duration(@t2)		

Experiment Results: Solution Cost

Scheduling Induces Parallelism, Halving Solution Duration

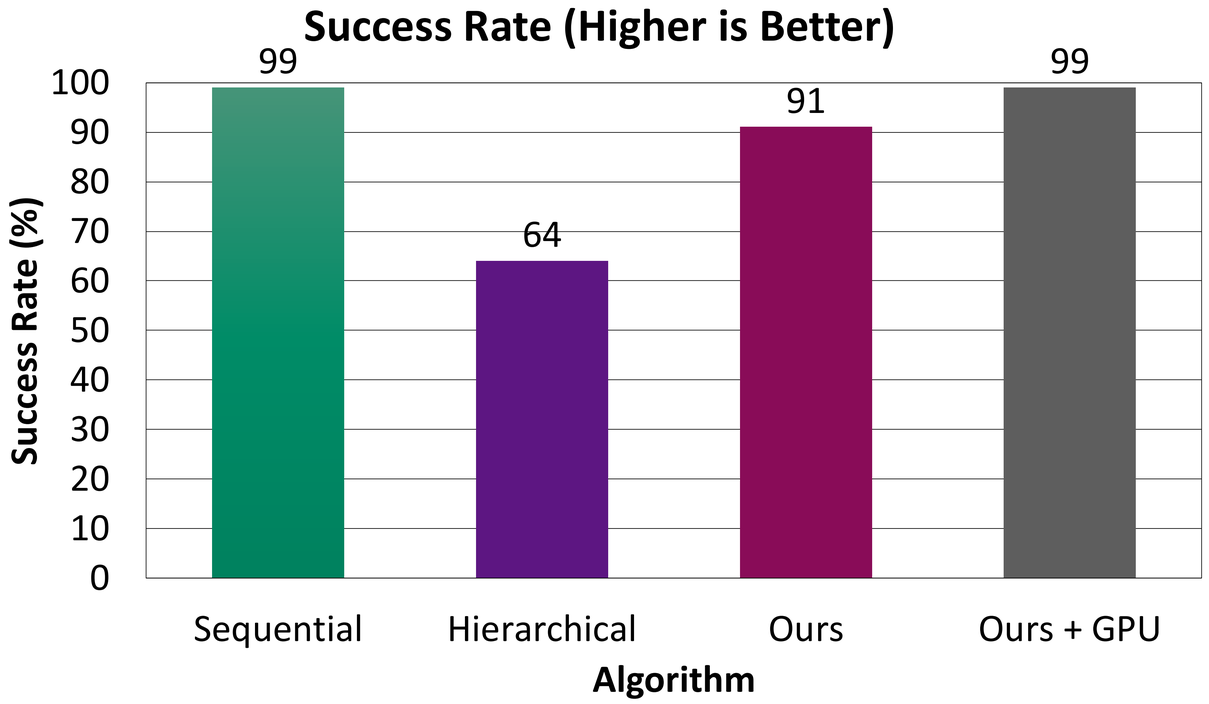
1. **Sequential**: traditional serial TAMP (e.g. cuTAMP, PDDLStream)
2. **Hierarchical**: strict task scheduling *then* motion planning decomposition
3. **Ours**: ScheduleStream
4. **Ours + GPU**: ScheduleStream w/ batched GPU accelerated samplers

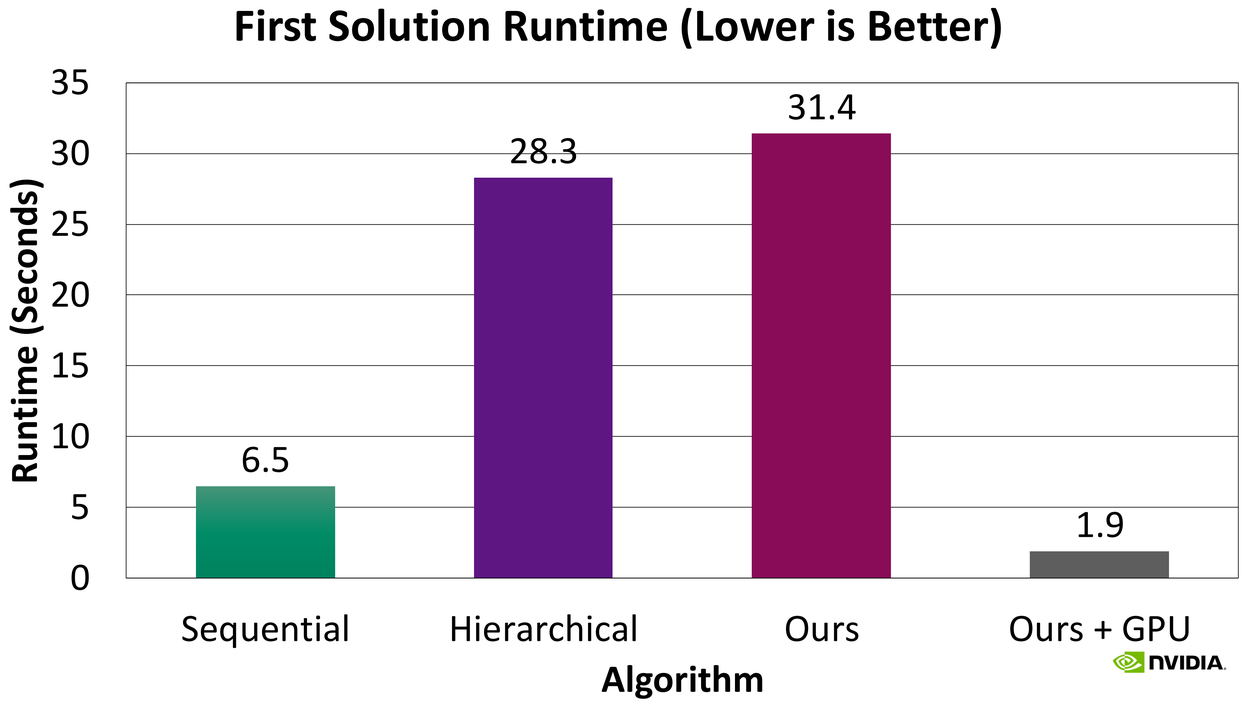


Experiment Results: Success Rate & Runtime

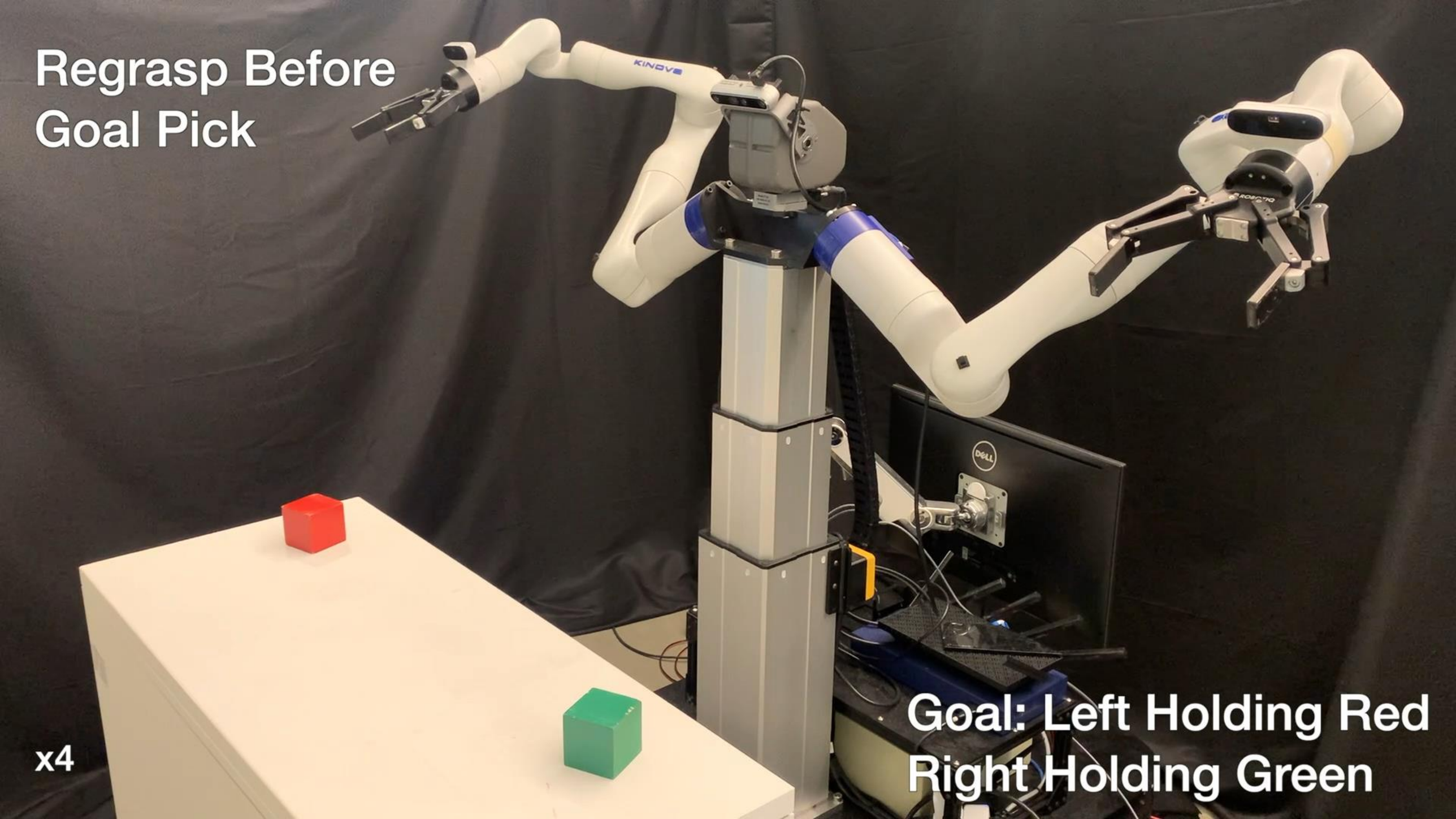
GPU Acceleration Combats Increase in Action Space

- **Hierarchical** is theoretically incomplete and empirically has a poor success rate
  - Task scheduling *then* motion planning fails when, for example, non-trivial reachability constraints
- **Ours + GPU** provides a dramatic speed up over **Ours**
  - GPU acceleration reduces overhead arising to robot-robot trajectory collision checking





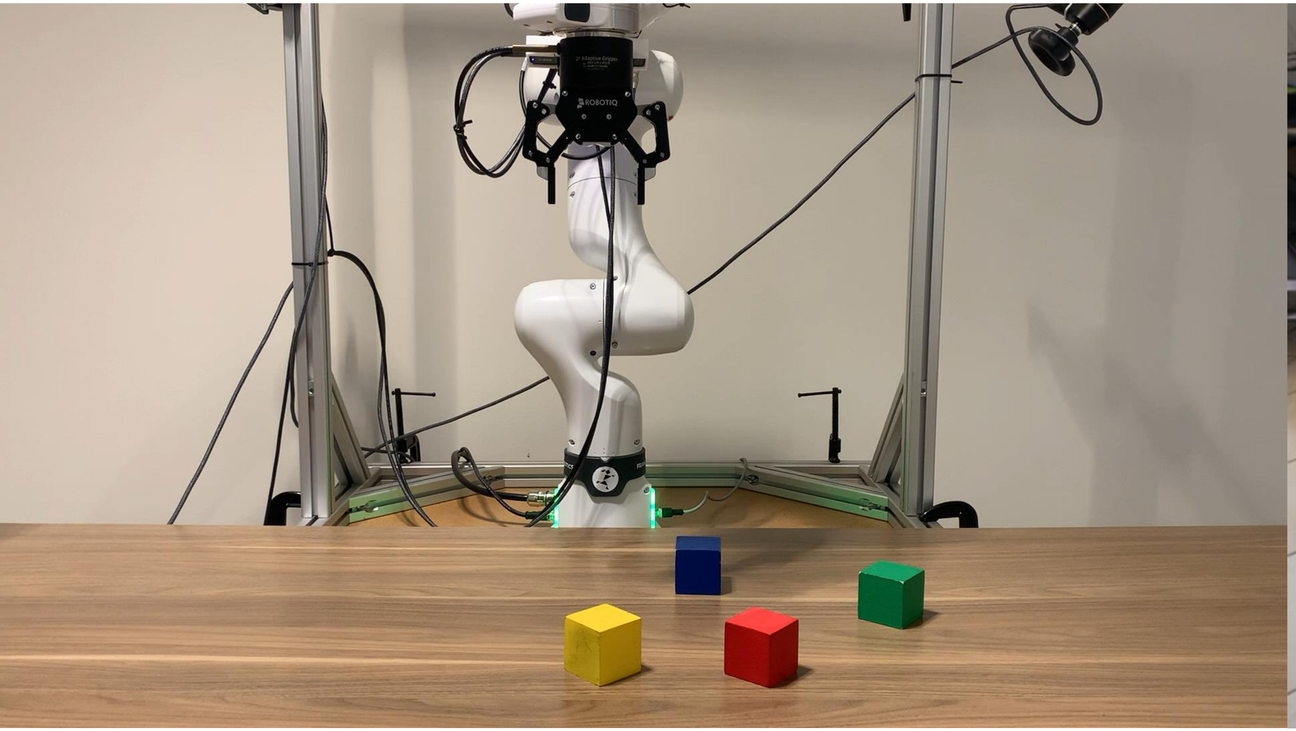
NVIDIA


Regrasp Before
Goal Pick
x4
Goal: Left Holding Red
Right Holding Green

Takeaways

ScheduleStream Temporal Planning Framework

- Scheduling allows for **simultaneous** yet **asynchronous** manipulation
- GPU acceleration **parallelizes** continuous manipulation planning
- Applications to automated **demonstration generation** for imitation learning



DROID

RoboLab