新一代物理引擎 Newton

解锁高精度仿真与机器人训练

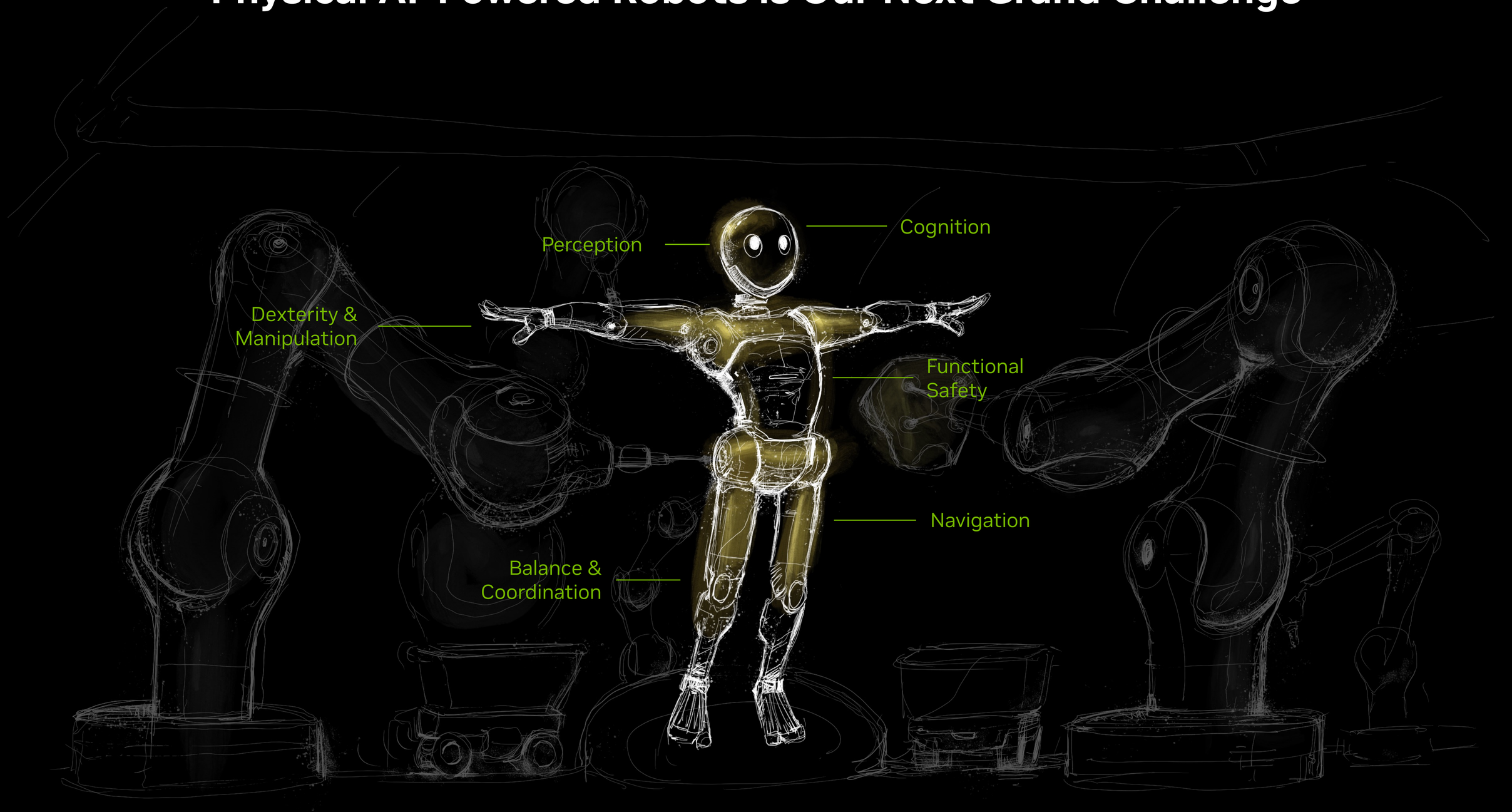
Juana Du，NVIDIA Robotics Solution Architect | Oct 2025

Agenda

- Newton overview
- Architecture design and GPU acceleration
- Advanced solvers and Multi-physics coupling support
- Isaac Lab integration

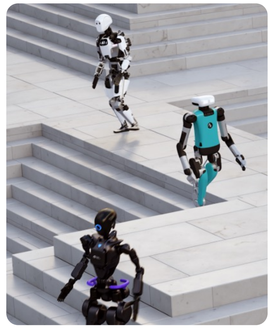
Newton overview

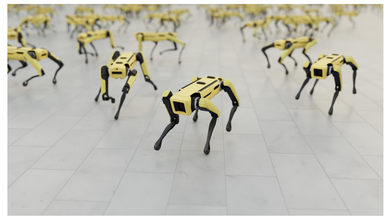
Physical AI-Powered Robots is Our Next Grand Challenge



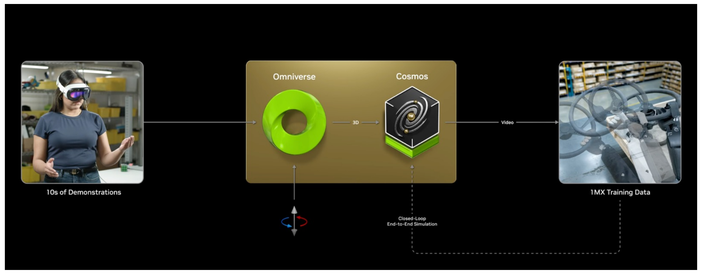
Simulation is Important for Robot Learning

GR00T: Generalist Robot 00 Technology

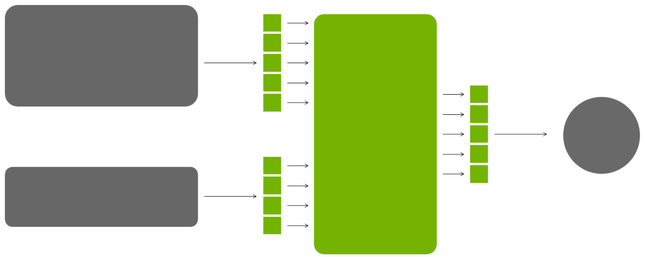




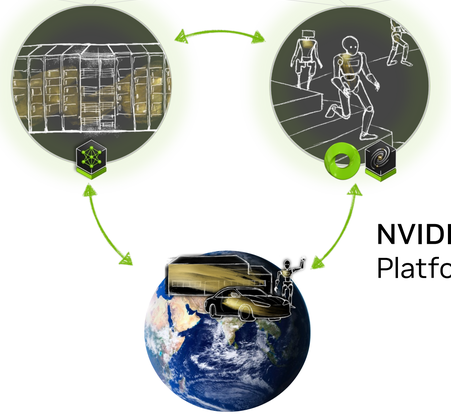
Isaac Sim & Isaac Lab



Synthetic Motion Generation Blueprint



GR00T Nx Foundation Model

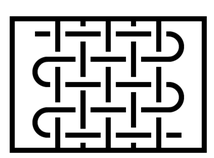


NVIDIA Isaac GR00T
Platform for Developing Humanoid Robots

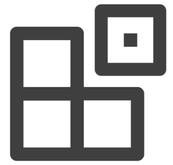
Next Generation Simulation for Robot Learning

Requirements

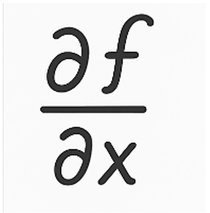
- **Modular & Extensible**
- **Community-built**
- **GPU Accelerated**
- **Industry-ready**
- **Common Data Interchange**
- **Unified API**
- **Differentiable**

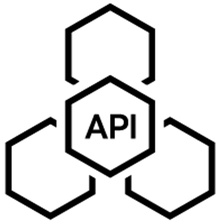


NVIDIA Warp

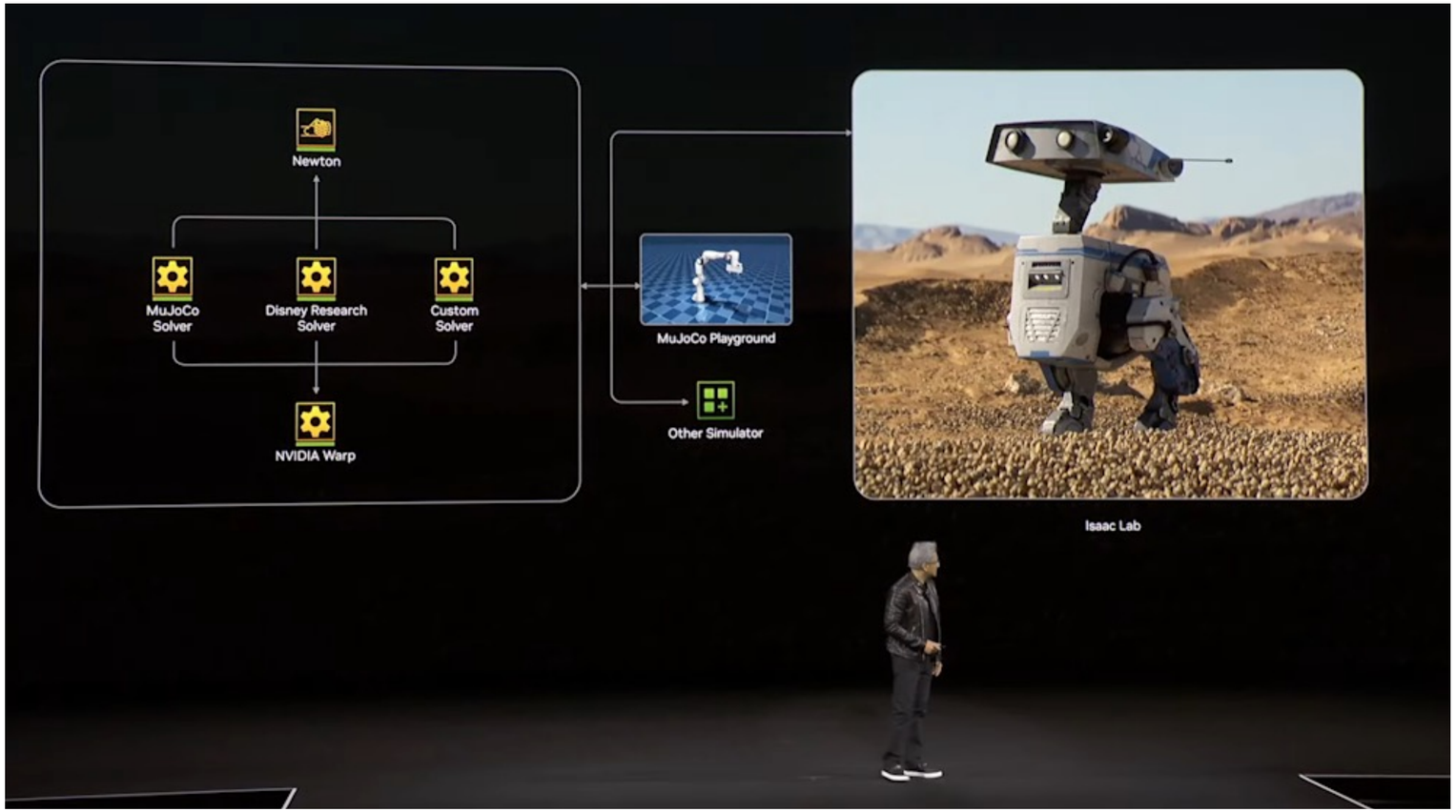


MuJoCo





Introducing Newton

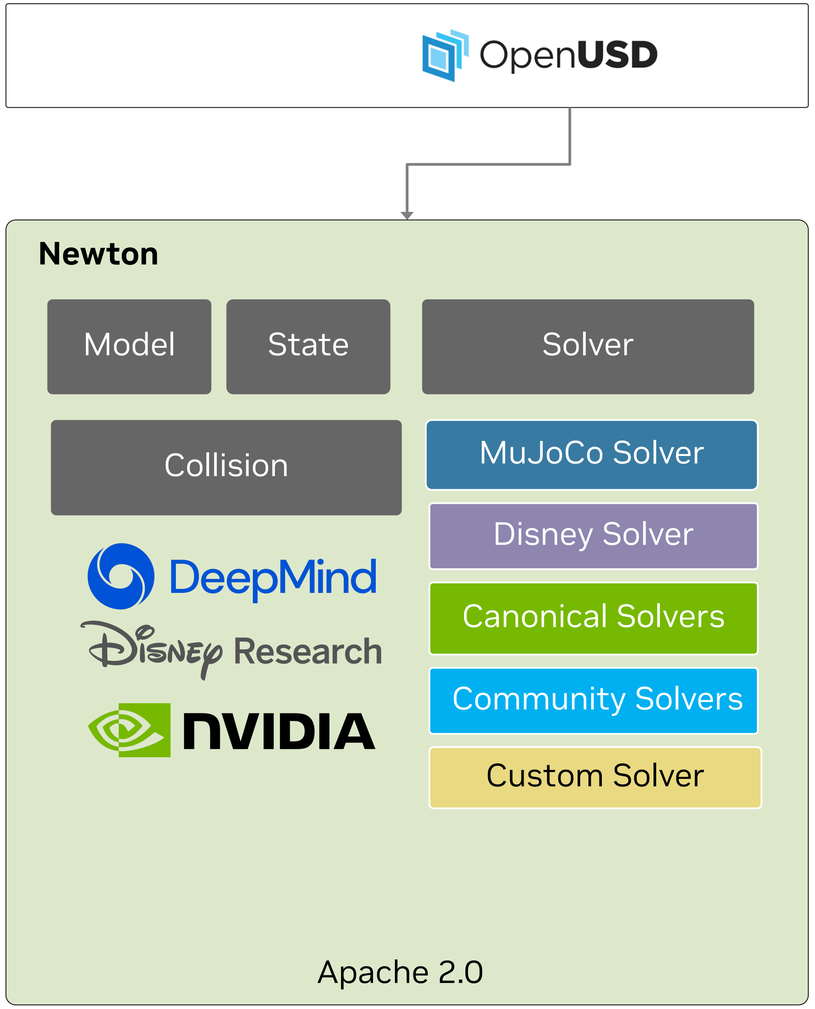


Disney Research

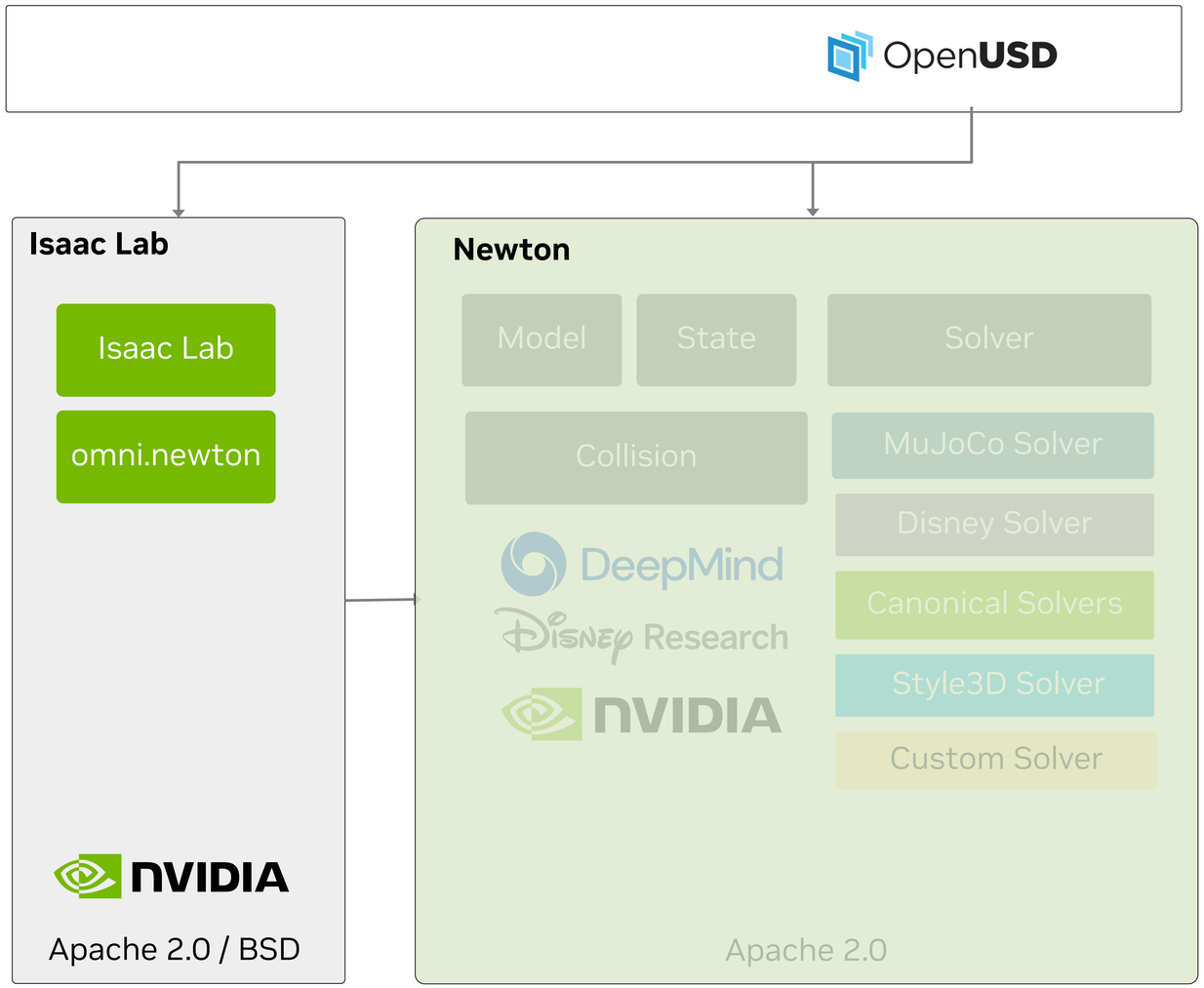
DeepMind

NVIDIA

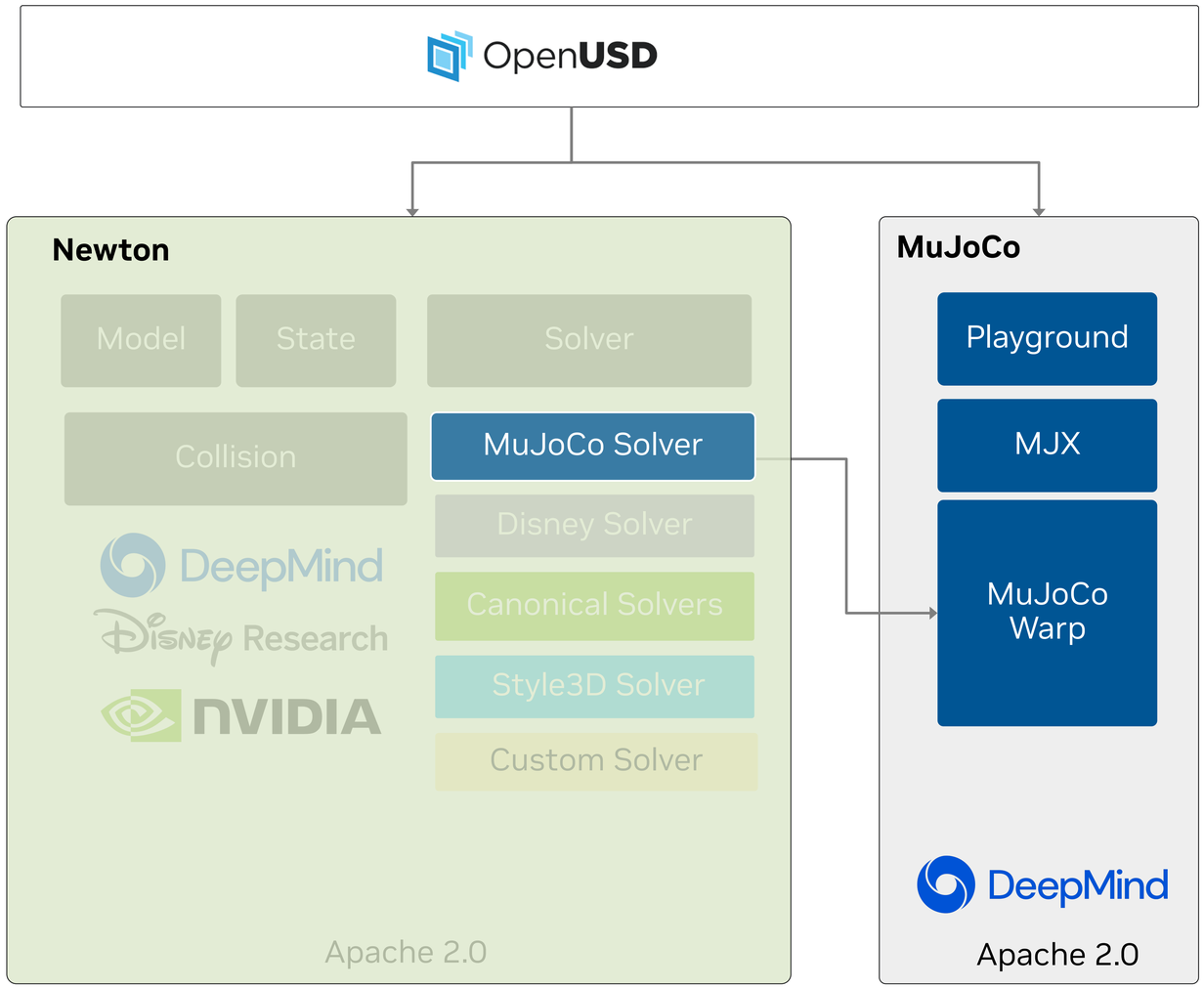
Architecture Overview



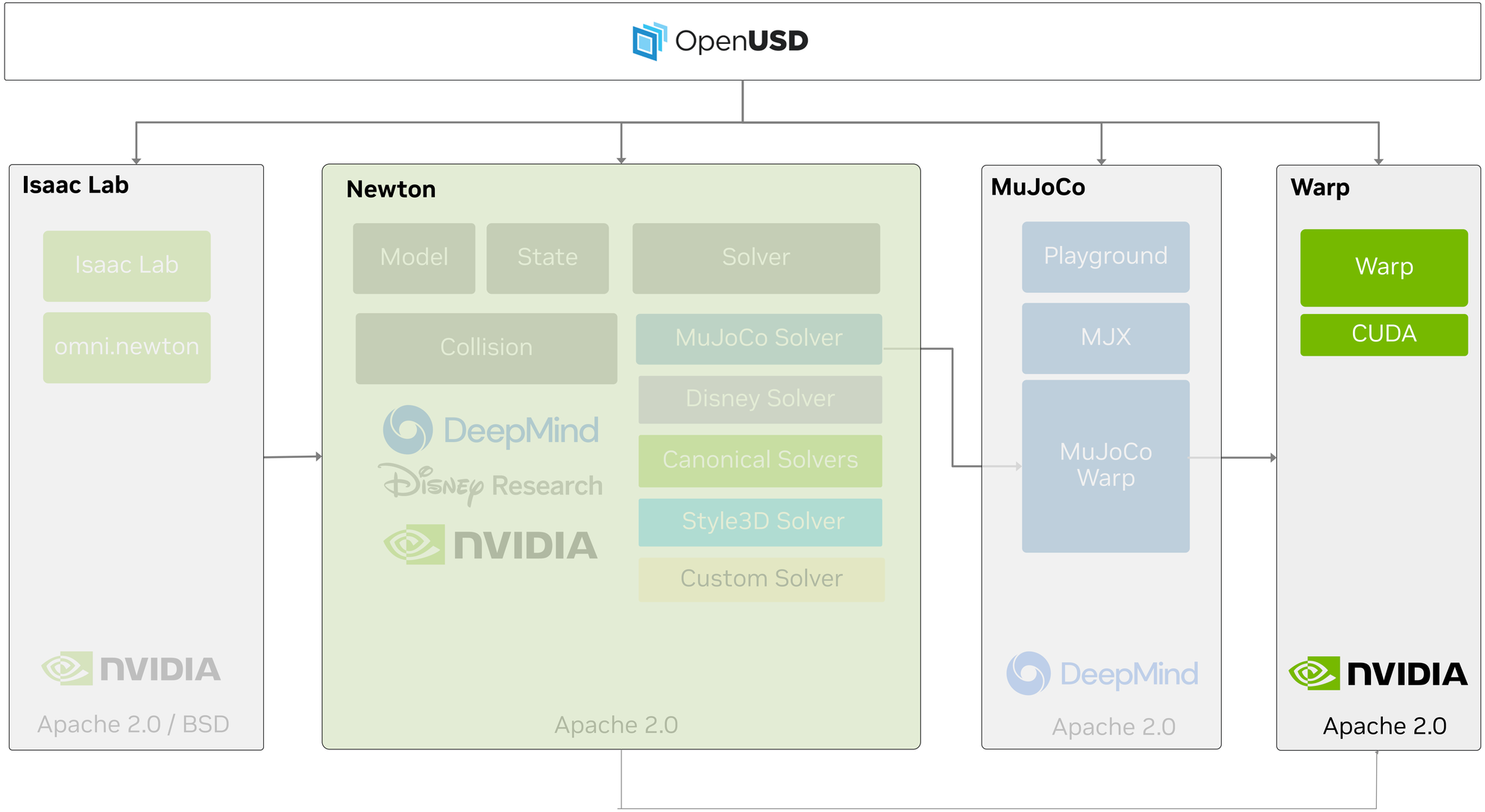
Architecture Overview



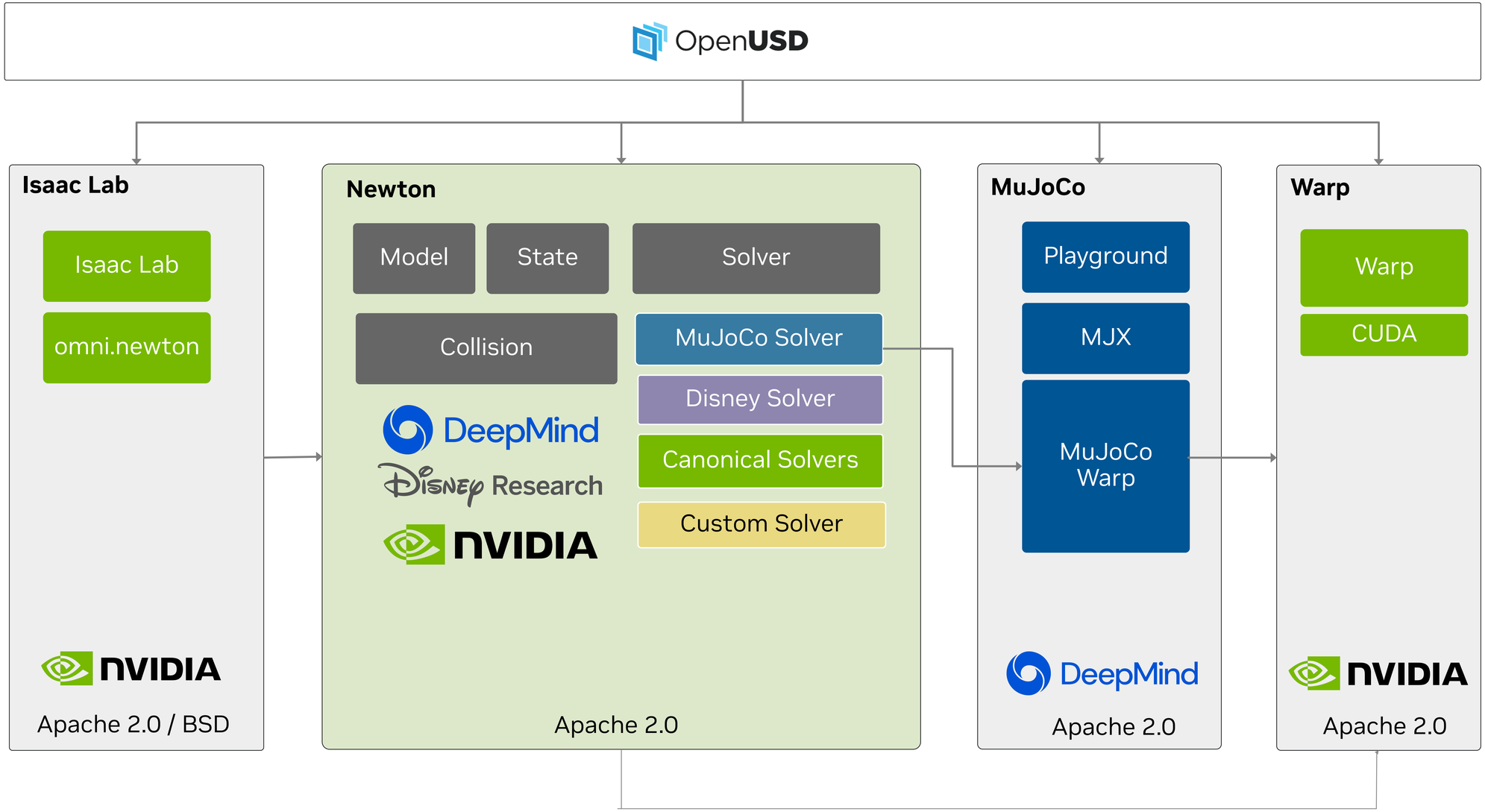
Architecture Overview



Architecture Overview



Architecture Overview



Introducing NVIDIA Warp

Performance

Native CUDA, JIT-compiled

CUDA-X libraries exposure

Ease of Use

Pure Python

Tile-based programming

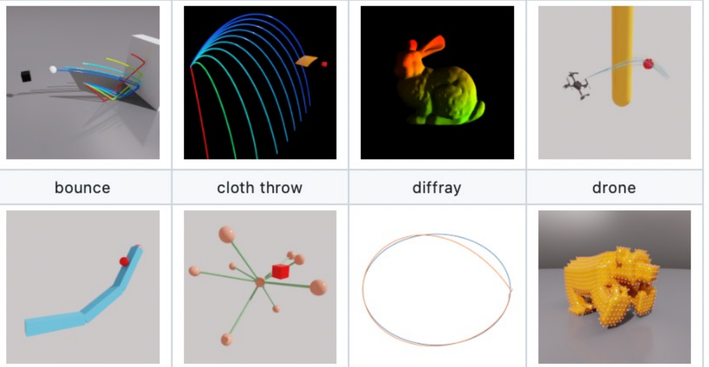
Native simulation primitives

Capability

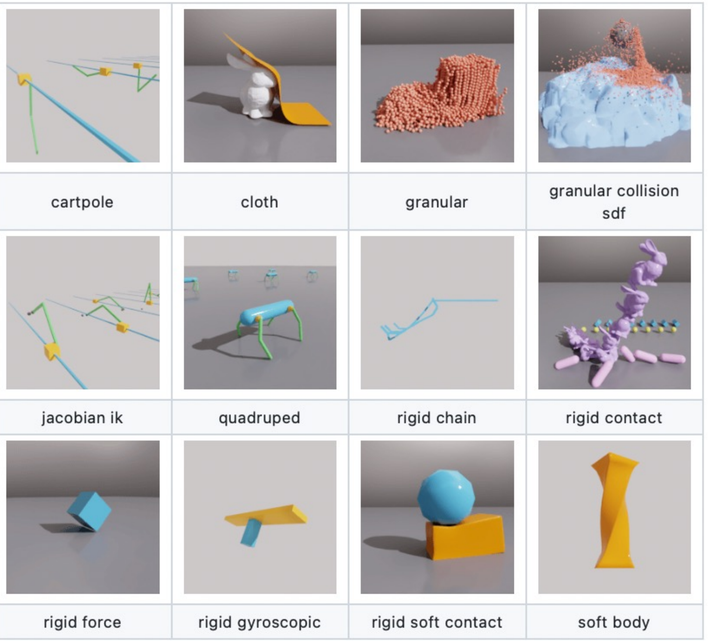
Automatic Differentiation

DL framework interop

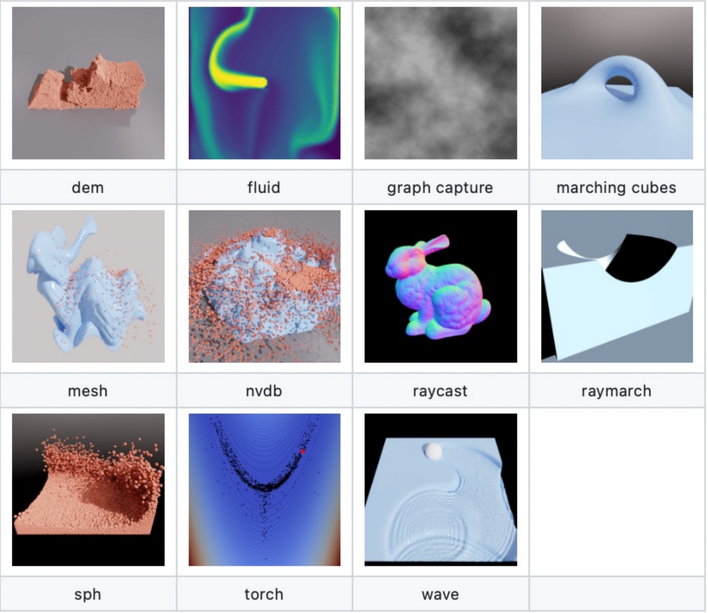
examples/optim



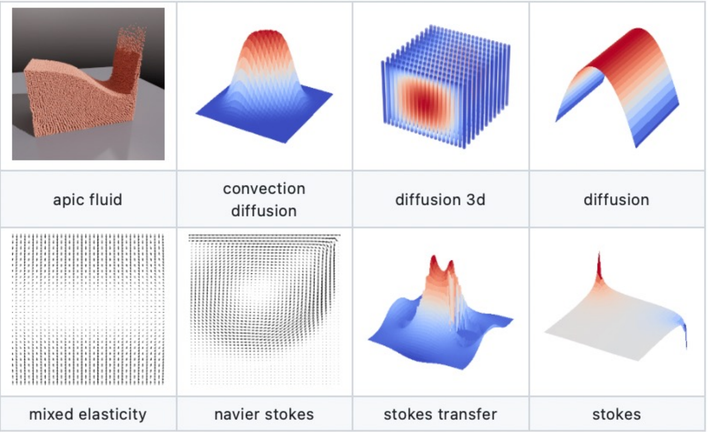
examples/sim



examples/core



examples/fem

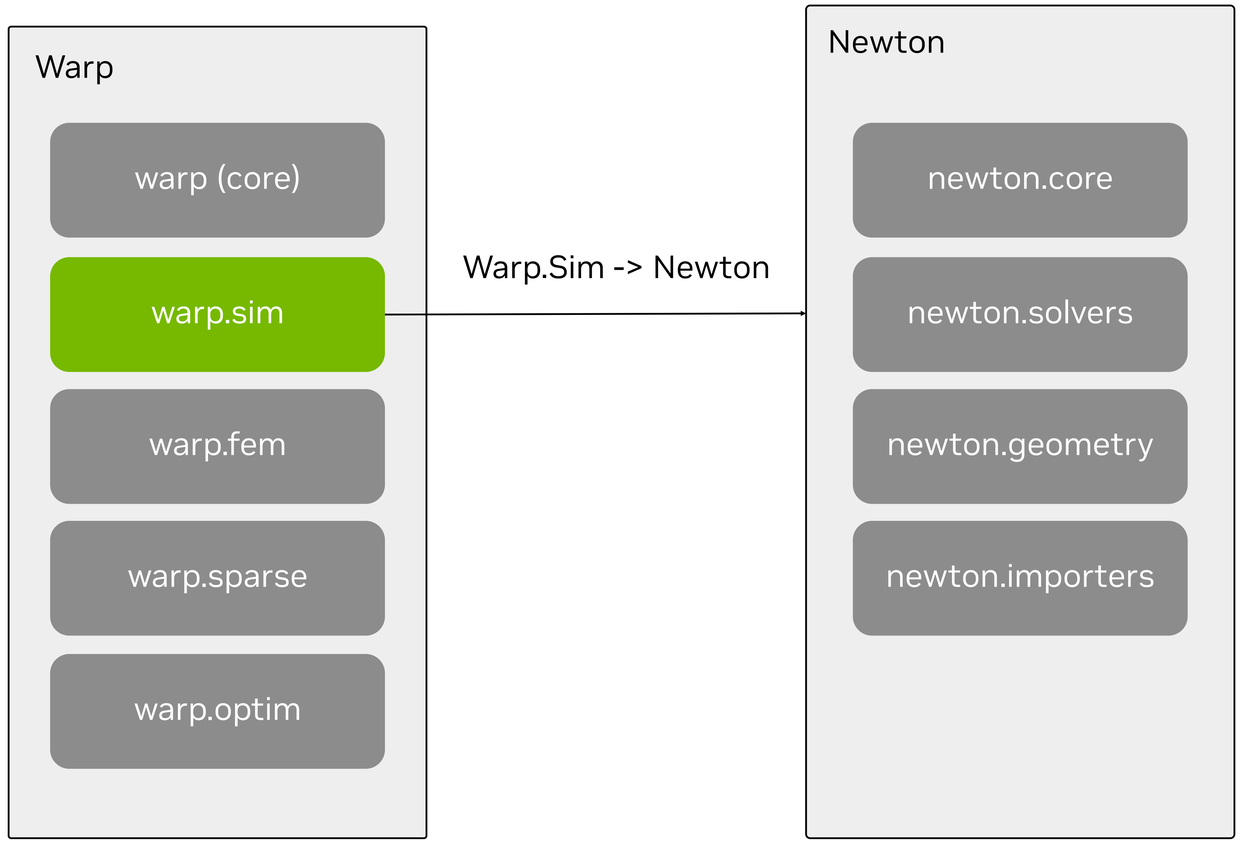


warp.sim Migration

- Newton architecture is derived from warp.sim
- Rebranding and deprecating over time
- Warp is a standalone framework

Open Source, Apache 2.0

pip install warp-lang



Warp Talks

Warp: A High-performance Python Framework for GPU Simulation and Graphics

Miles Macklin, NVIDIA

GTC, San Jose, March 2022

Warp: Advancing Simulation AI with Differentiable GPU Computing in Python

Miles Macklin, NVIDIA

GTC, San Jose, March 2024

Distributed and Differentiable Fluid Dynamics with Accelerated Python

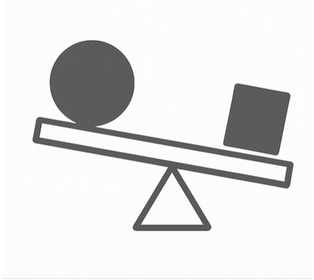
Massimiliano Meneghin, Autodesk
Lukasz Wawrzyniak, NVIDIA

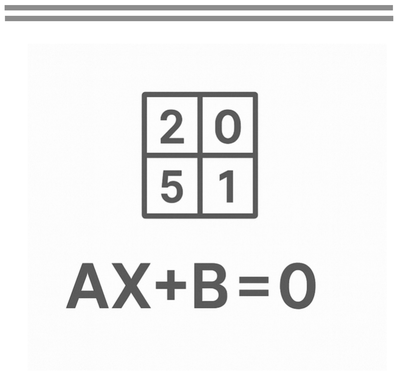
GTC, San Jose, March 2025

Architecture Design and GPU Acceleration

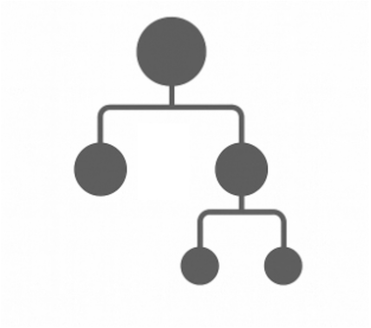
Newton Design Principles

Guidelines & Lessons Learned

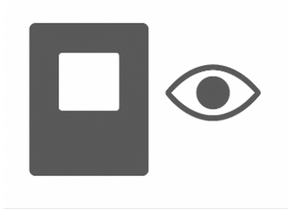


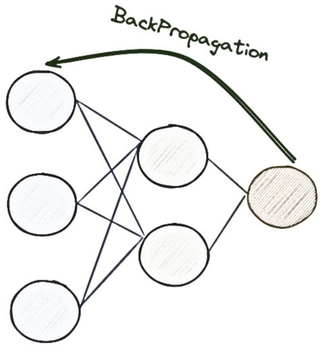


Separate physical model from numerical method

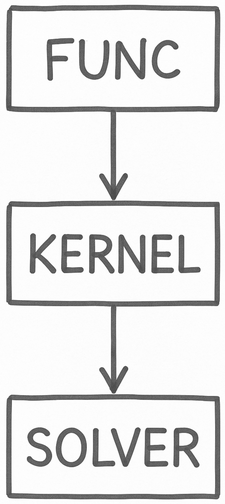


Flat data preferred over object-oriented programming





Avoid hidden state, more control over memory



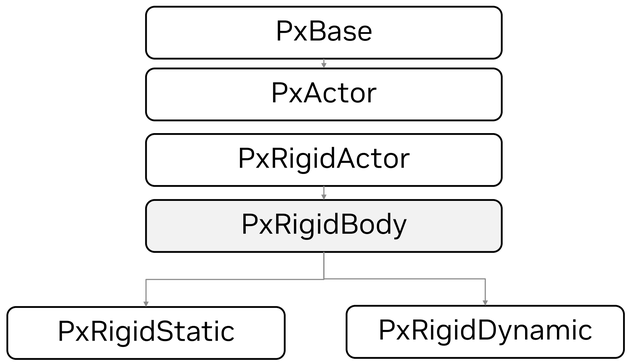
Take what you need, at each level

Flat Data Design

Remove layers of abstraction

PhysX 5

```
class PxRigidBody : public PxRigidActor
{
public:

   virtual void   setMass(PxReal mass) = 0;
   virtual PxReal getMass() const = 0;
   virtual PxReal getInvMass() const = 0;

   virtual void   setMassSpaceInertiaTensor(const PxVec3& m) = 0;
   virtual PxVec3 getMassSpaceInertiaTensor() const = 0;
   virtual PxVec3 getMassSpaceInvInertiaTensor() const = 0;

   virtual void   setLinearDamping(PxReal linDamp) = 0;
   virtual PxReal getLinearDamping() const = 0;
   virtual void   setAngularDamping(PxReal angDamp) = 0;
   virtual PxReal getAngularDamping() const = 0;
```



Newton

```
class Model:
"""Holds the definition of the simulation model
This class holds the non-time varying description of
the system, i.e.: all geometry, constraints, and
parameters used to describe the simulation. """

   body_com : wp.array(dtype=wp.vec3)
   body_inertia : wp.array(dtype=wp.mat33)
   body_inv_inertia : wp.array(dtype=wp.mat33)
   body_mass : wp.array(dtype=float)

   joint_type : wp.array(dtype=int)
   joint_parent : wp.array(dtype=int)
   joint_child : wp.array(dtype=int)
   joint_X_p : wp.array(dtype=wp.transform)
   joint_X_c : wp.array(dtype=wp.transform)
   joint_axis : wp.array(dtype=wp.vec3)

   spring_indices : wp.array(dtype=int)
   spring_rest_length : wp.array(dtype=float)
   spring_stiffness : wp.array(dtype=float)
   spring_damping : wp.array(dtype=float)
   spring_control : wp.array(dtype=float)
```

newton.sim.Model

System description

- Flat Python struct of 1D, 2D Warp arrays
- Stores non-time varying data
- Fundamentally, we store data grouped by frequencies:
  - per-Joint (e.g.: parent body)
  - per-DOF (e.g.: armature)
  - per-Articulation (e.g.: joint dof offset)
  - per-Body (e.g.: mass, inertia)
  - per-Shape (e.g.: friction coefficients)
- Flexible thanks to Python's dynamic typing
- E.g.: add shape lubrication for custom friction
- model.shape_lubrication = wp.ones(dtype=float, length=n)

```
class Model:
    """Holds the definition of the simulation model

    This class holds the non-time varying description of
    the system, i.e.: all geometry, constraints, and
    parameters used to describe the simulation.

    Attributes:

        body_com : array(dtype=wp.vec3)
        body_inertia : array(dtype=wp.mat33)
        body_inv_inertia : array(dtype=wp.mat33)
        body_mass : array(dtype=float)

        joint_type : array(dtype=int)
        joint_parent : array(dtype=int)
        joint_child : array(dtype=int)
        joint_X_p : array(dtype=wp.transform)
        joint_X_c : array(dtype=wp.transform)
        joint_axis : array(dtype=wp.vec3)

        spring_indices : array(dtype=int)
        spring_rest_length : array(dtype=float)
        spring_stiffness : array(dtype=float)
        spring_damping : array(dtype=float)
        spring_control : array(dtype=float)

        ...
```

newton.sim.State

Physical configuration

- State objects hold time-varying data
  - Primarily joint_q, body_q, particle_q
  - Flat struct of wp.arrays
- Create state objects by asking the model for a new state object using model.state()
- Can create as many state objects as you require, easy to save / restore state
- User can add fields
  - e.g.: state.particle_temperature

```
class State:
    """The State object holds all *time-varying* data for
     a model.

    Attributes:

        joint_q (array): Array of joint positions
        joint_qd (array): Array of joint velocities

        body_q (array): Array of body coordinates
        body_qd (array): Array of body coordinates

        particle_q (array): Array of 3D particle
        particle_qd (array): Array of 3D particle
```

newton.sim.Control

External forces and actuation

- Represents actuation inputs over a timestep
- Includes joint actuation, PD targets, activations
- Supports both implicit & explicit methods
- Create Control object by calling model.control()

```
class Control:
"""Time-varying control data for a :class:`Model`.

Time-varying control data includes joint control
inputs, and activation forces for triangle and
tetrahedral elements.

Attributes:

    # joint actuation
    joint_f (array): Generalized joint forces
    joint_target (array): Joint PD targets

    # triangle activations
    tri_activations (array):

    # tetrahedra activations
    tet_activations (array):
```

newton.sim.Contacts

Contact Geometry

- Generated by collision pipeline
- User controls when/how to compute contacts
- Passed to solver.step()

```
class Contacts:
    """Provides contact information to be consumed by
    a solver. Stores the contact distance, position,
    frame, and geometry for each contact point.
    .. note::

    Attributes:

        rigid_contact_count : array(dtype=int)
        rigid_contact_point_id : array(dtype=int)
        rigid_contact_shape0 : array(dtype=int)
        rigid_contact_shape1 : array(dtype=int)
        rigid_contact_point0 : array(dtype=wp.vec3)
        rigid_contact_point1 : array(dtype=wp.vec3)
        rigid_contact_normal : array(dtype=wp.vec3)
        rigid_contact_thickness : array(dtype=float)
        rigid_contact_tids : array(dtype=int)

        soft_contact_count : array(dtype=int)
        soft_contact_particle : array(dtype=int)
        soft_contact_shape : array(dtype=int)
        soft_contact_body_pos : array(dtype=wp.vec3)
        soft_contact_body_vel : array(dtype=wp.vec3)
        soft_contact_normal : array(dtype=wp.vec3)
        soft_contact_tids : array(dtype=int)
```

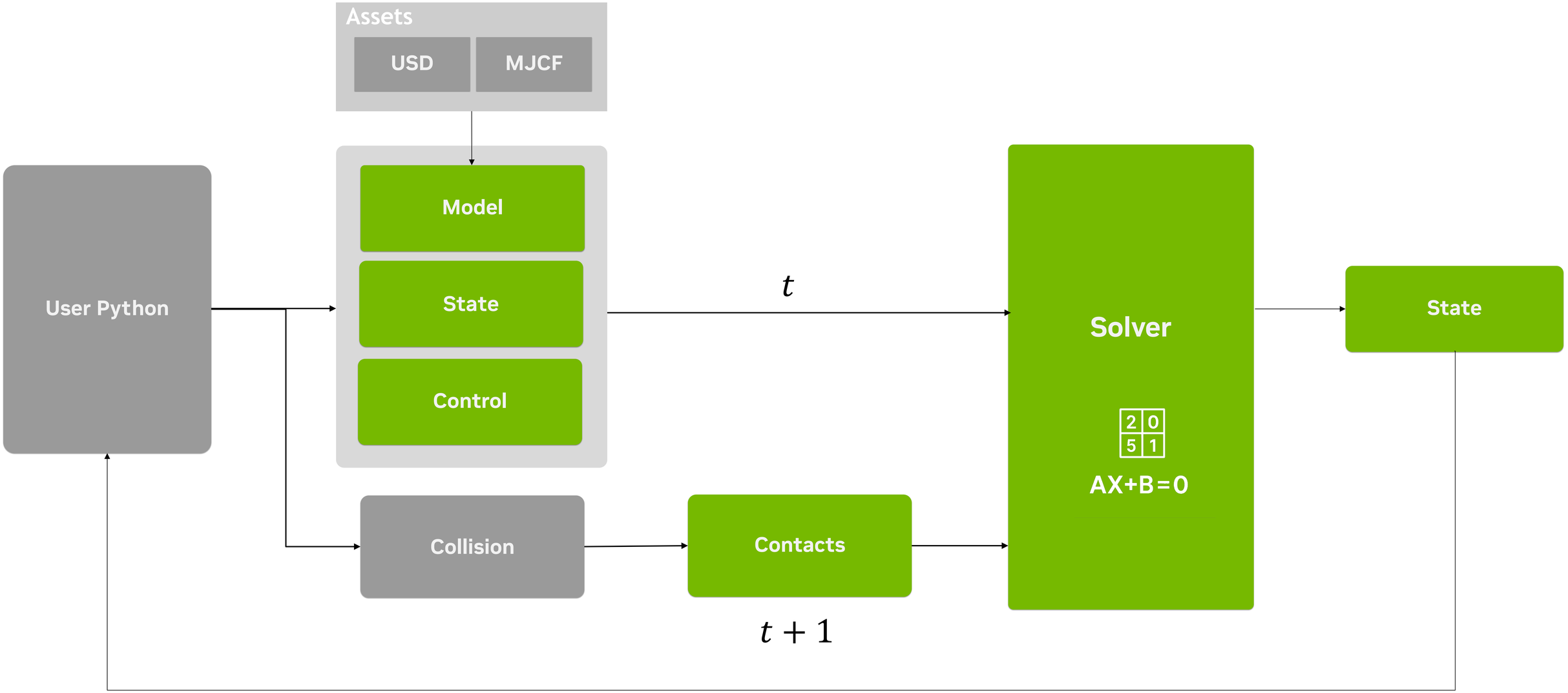
Geometry Library

newton.geometry

- Goal: reusable collision detection and contact generation
- Split into standalone module for flexibility
- High-level entry points (broad phase, narrow phase)
- Expose low-level routines to users (SDF funcs, queries, GJK)

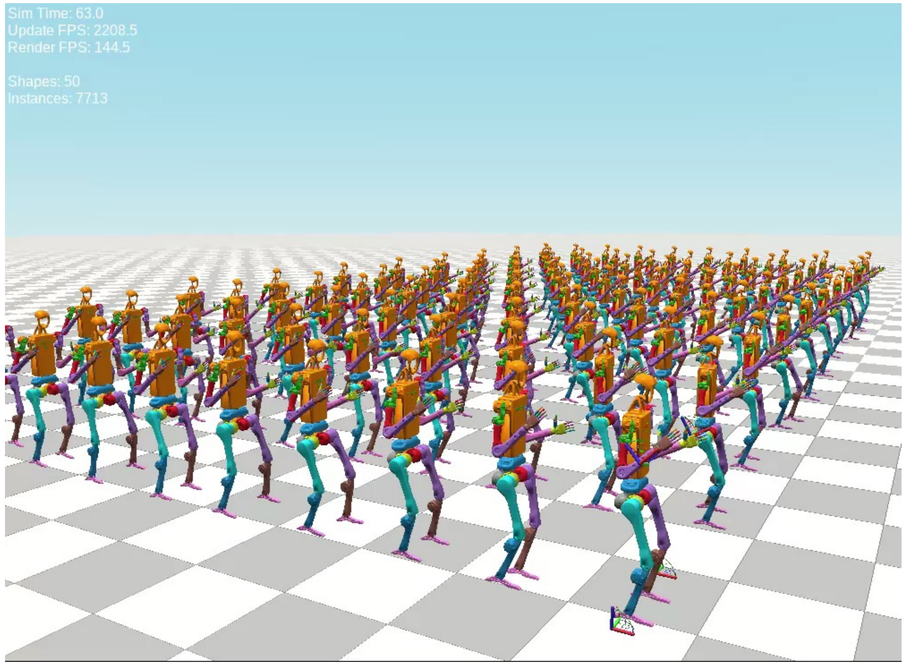
```
# geometry types
GEO_SPHERE = wp.constant(0)
GEO_BOX = wp.constant(1)
GEO_CAPSULE = wp.constant(2)
GEO_CYLINDER = wp.constant(3)
GEO_CONE = wp.constant(4)
GEO_MESH = wp.constant(5)
GEO_SDF = wp.constant(6)
GEO_PLANE = wp.constant(7)
GEO_NONE = wp.constant(8)

# collision pipeline
def broad_phase(
    # input
    bounds_origin: wp.array(dtype=wp.vec3),
    bounds_size: wp.array(dtype=wp.vec3)
    bounds_type: wp.array(dtype=int),
    # output
    pairs: wp.array(dtype=wp.vec2i))

def narrow_phase(
    # input
    shape_transform: wp.array(dtype=wp.transform),
    shape_type: wp.array(dtype=int)
    shape_scale: wp.array(dtype=wp.vec3),
    pairs: wp.array(dtype=wp.vec2i) ,
    # output
    contact_position: wp.array(dtype=wp.vec2i),
    contact_frame: wp.array(dtype=wp.quat),
    contact_depth: wp.array(dtype=float))


# SDF functions
@wp.func
def sphere_sdf(center: wp.vec3, radius: float, p: wp.vec3):

@wp.func
def box_sdf(upper: wp.vec3, p: wp.vec3):
...
```

A Reference Simulation Pipeline



Inverse Kinematics (IK) Solver

- Forward/Inverse kinematics map between representations:
  - newton.ik.IKSolver()
  - newton.ik.IKPositionObjective()
  - newton.ik.IKRotationObjective()
  - newton.ik. IKJointLimitObjective()
  - Newton.ik. IKJacobianMode

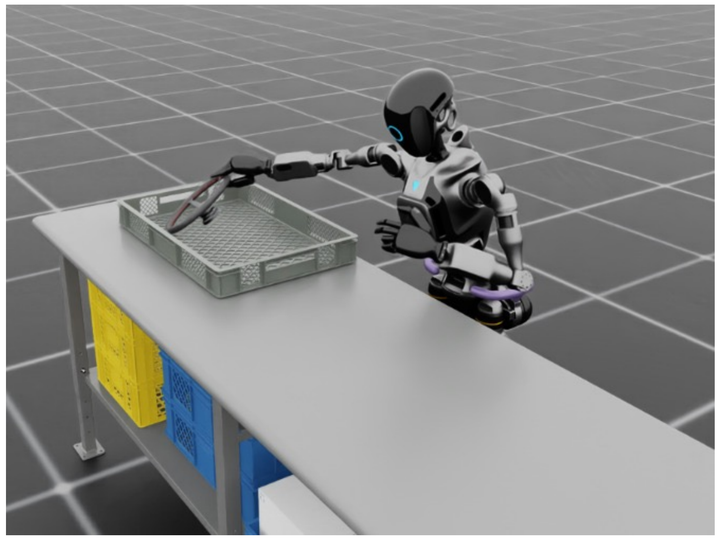


H1 Whole Body IK @ 0.03ms in Newton

GPU Acceleration

Kernal based

- Key challenges for physics simulation:
  - Possible Contacts scales O(n^2)
  - Actual contacts are only known at runtime
- Challenges from Tensor based frameworks
  - Branching (conditional execution)
  - Sparsity (fine-grained memory allocation)



```python
@wp.kernel
def divergence(u: wp.array2d(dtype=wp.vec2),
               div: wp.array2d(dtype=float)):

    # 2D thread indices
    i, j = wp.tid()

    # boundary conditions
    if i == grid_width - 1:
        return
    if j == grid_height - 1:
        return

    # compute divergence
    dx = (u[i + 1, j][0] - u[i, j][0]) * 0.5
    dy = (u[i, j + 1][1] - u[i, j][1]) * 0.5
    div[i, j] = dx + dy

# 2d kernel launch
wp.launch(divergence, dim=[512, 512], inputs=[u, div])
```

Example: 2D Divergence Calculation in Warp

GPU Acceleration

Advanced Features

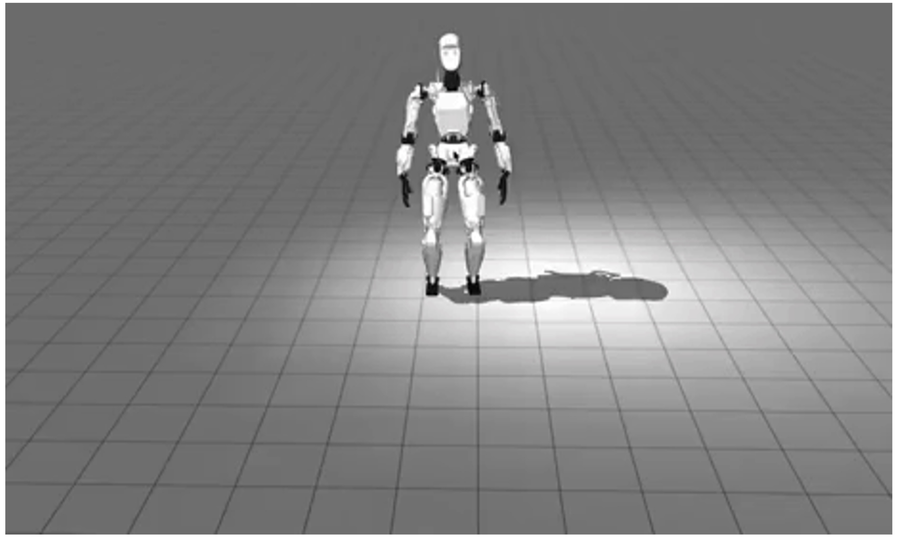
- CUDA Graph Capture
  - Reduce kernel launch overhead
  - Speedups for iterative workloads like simulation loops
  - wp.ScopedCapture() and wp.capture_launch(graph)

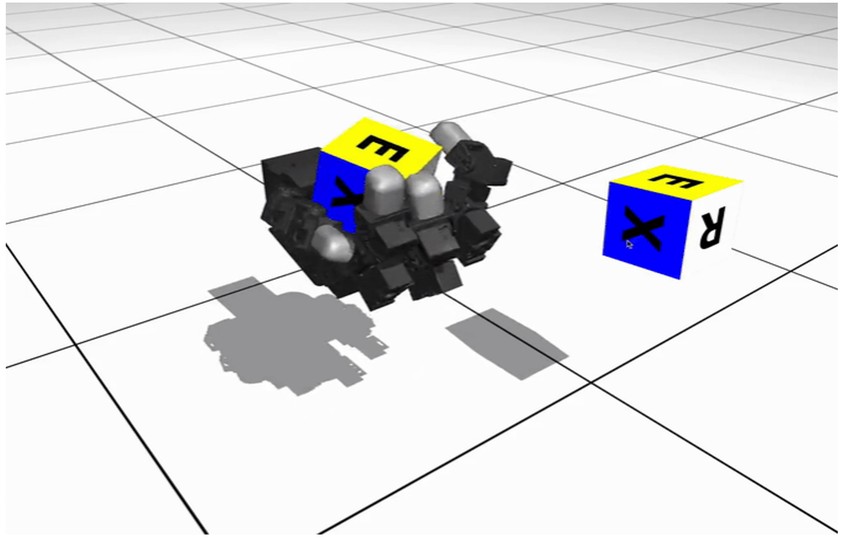
- Tile-based Programming
  - Make use of GPU shared memory
  - Speedups for GEMM and processing adjacent elements
  - wp.tile_matmul() and wp.tile_ reduce()

- NVIDIA MathDx Integration
  - Leveraging cuBLASDx and cuFFTDx
  - Build-Time Integration

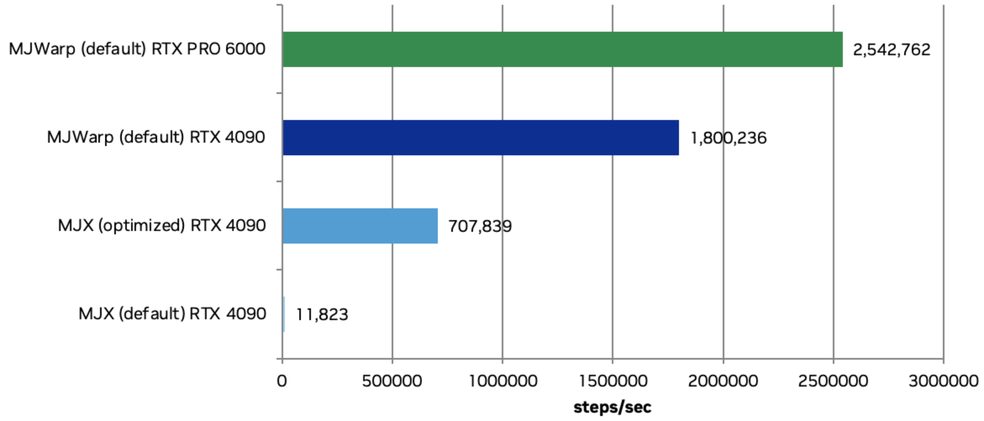
DeepMind

MuJoCo Solver
(MJWarp)

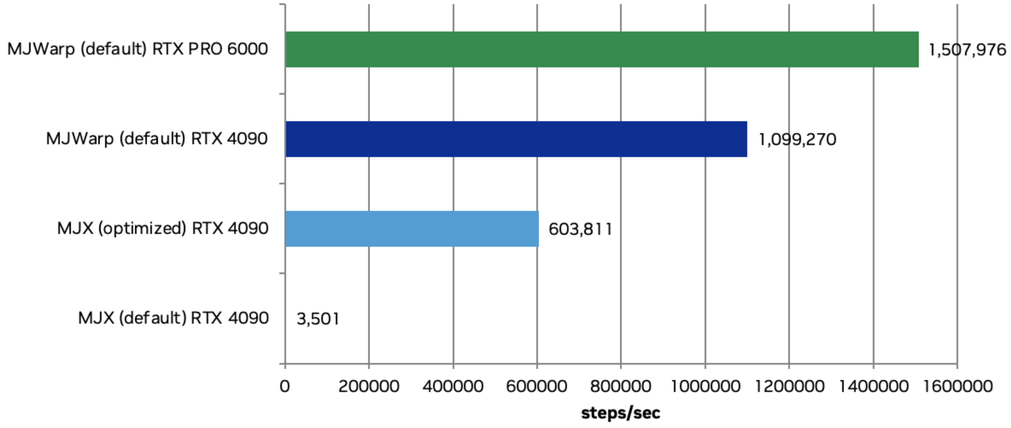




Apptronik locomotion, physics step/sec



Leap hand manipulation, physics step/sec



NVIDIA

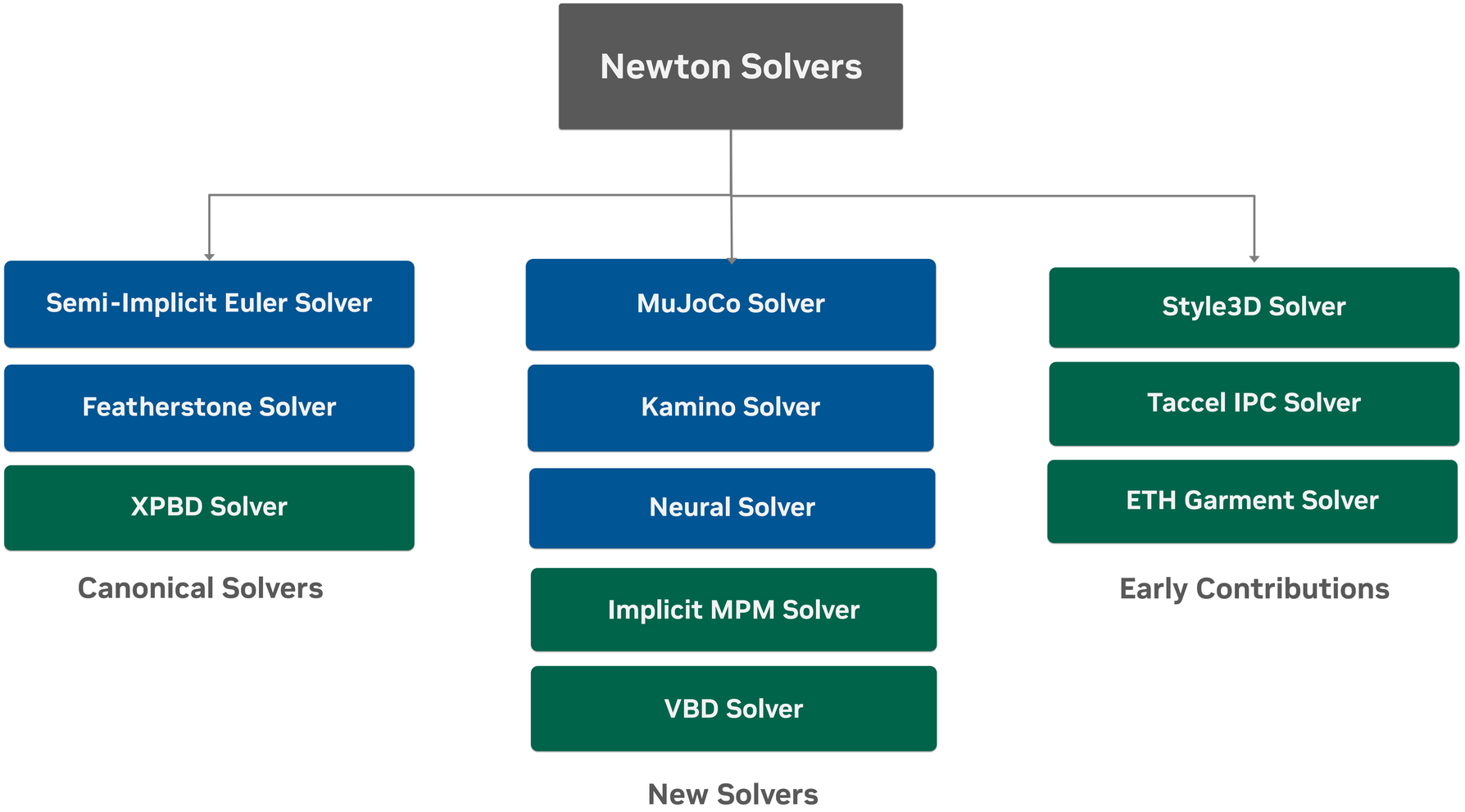
Advanced solvers and Multi-physics coupling support

Newton Solvers

Available as Plugins

Rigid Body

Deformable / Hybrid



Style3D

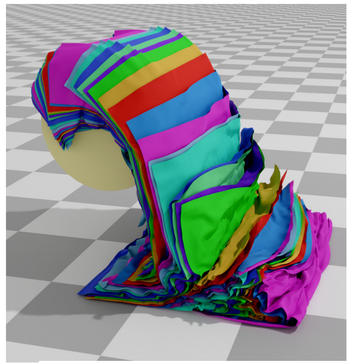
PEKING UNIVERSITY

ETH zürich

VBD Solver

Vertex Block Descent

SIGGRAPH 2025
Vancouver+ 10-14 August



Offset
Geometric
Contact

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

Vertex
Block
Descent

Implicit MPM Solver

Material Point Method

Render the Possibilities
SIGGRAPH2016

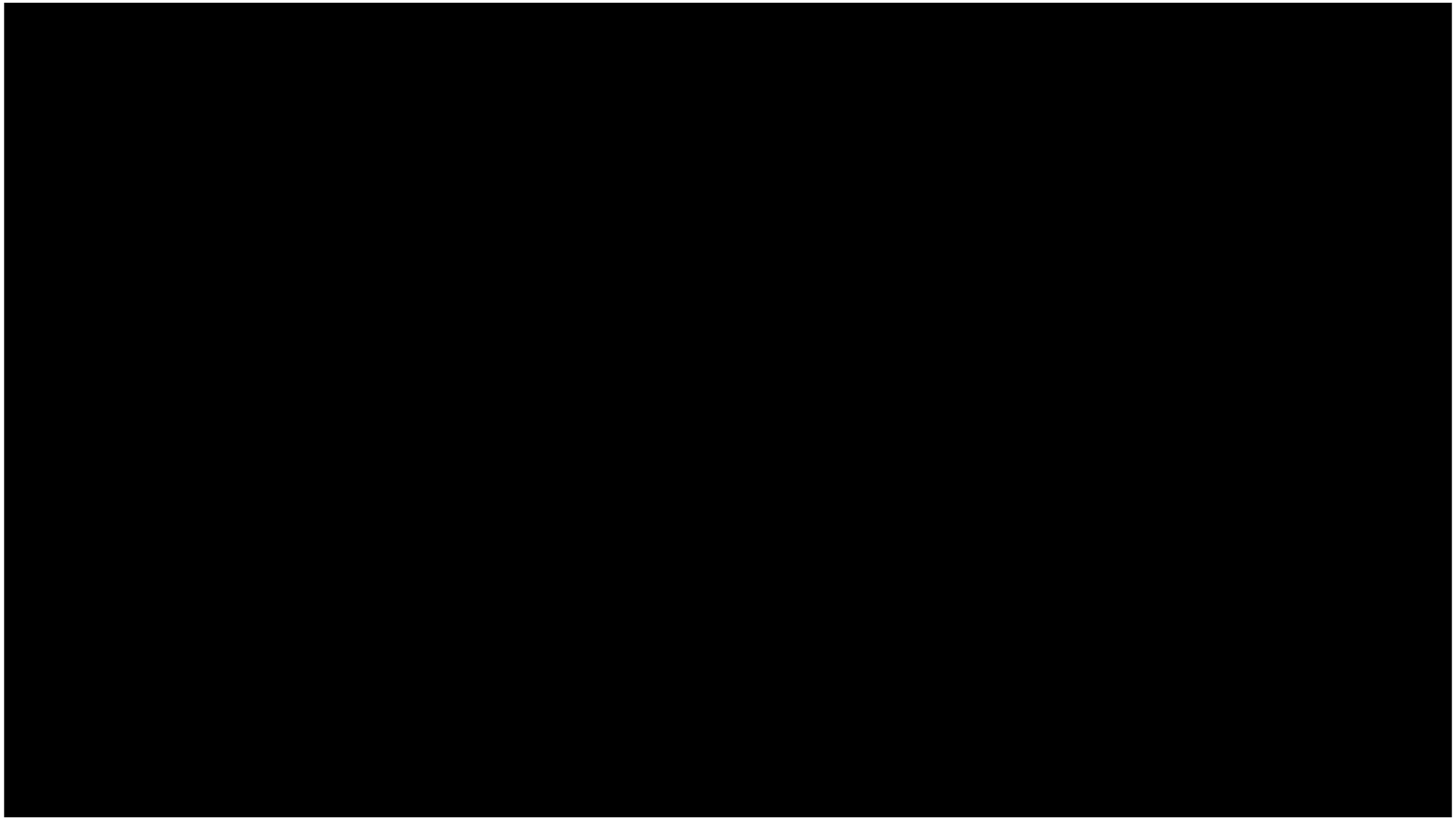
A Semi-Implicit Material Point Method for the Continuum Simulation of Granular Materials

Style3D Solver

cloth simulation

PKU : Taccel IPC Solver

vision-based tactile sensing



Solver Coupling

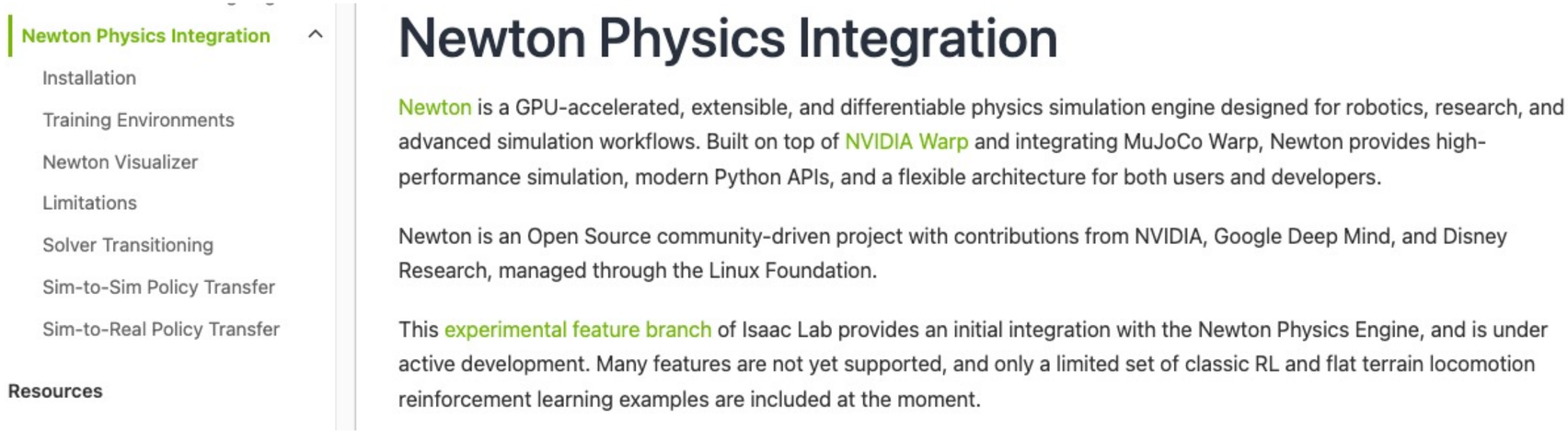
- Model supports mixed systems: rigid, cloth, soil
- Multiple solvers can run independently
- Currently one-way coupling

```
import newton

builder = newton.sim.ModelBuilder()

# import franka
newton.utils.parse_urdf("franka.urdf", builder)

# add table + cloth
builder.add_body(xform=(0.0, 1.0, 0.0))
builder.add_cloth_grid(pos=(0.0, 2.0, 0.0), dim_x=100,
dim_y=100)

mjc_solver->step(...)
vbd_solver->step(...)
```

Isaac Lab Integration

Experimental Feature Branch

- Under active development, only support limited features and workflows
- Purpose is to let the users understand how Newton integration works in Isaac Lab
- Future updates: Performance improvements and more solver integration



Selection API

Flexible views onto simulation data

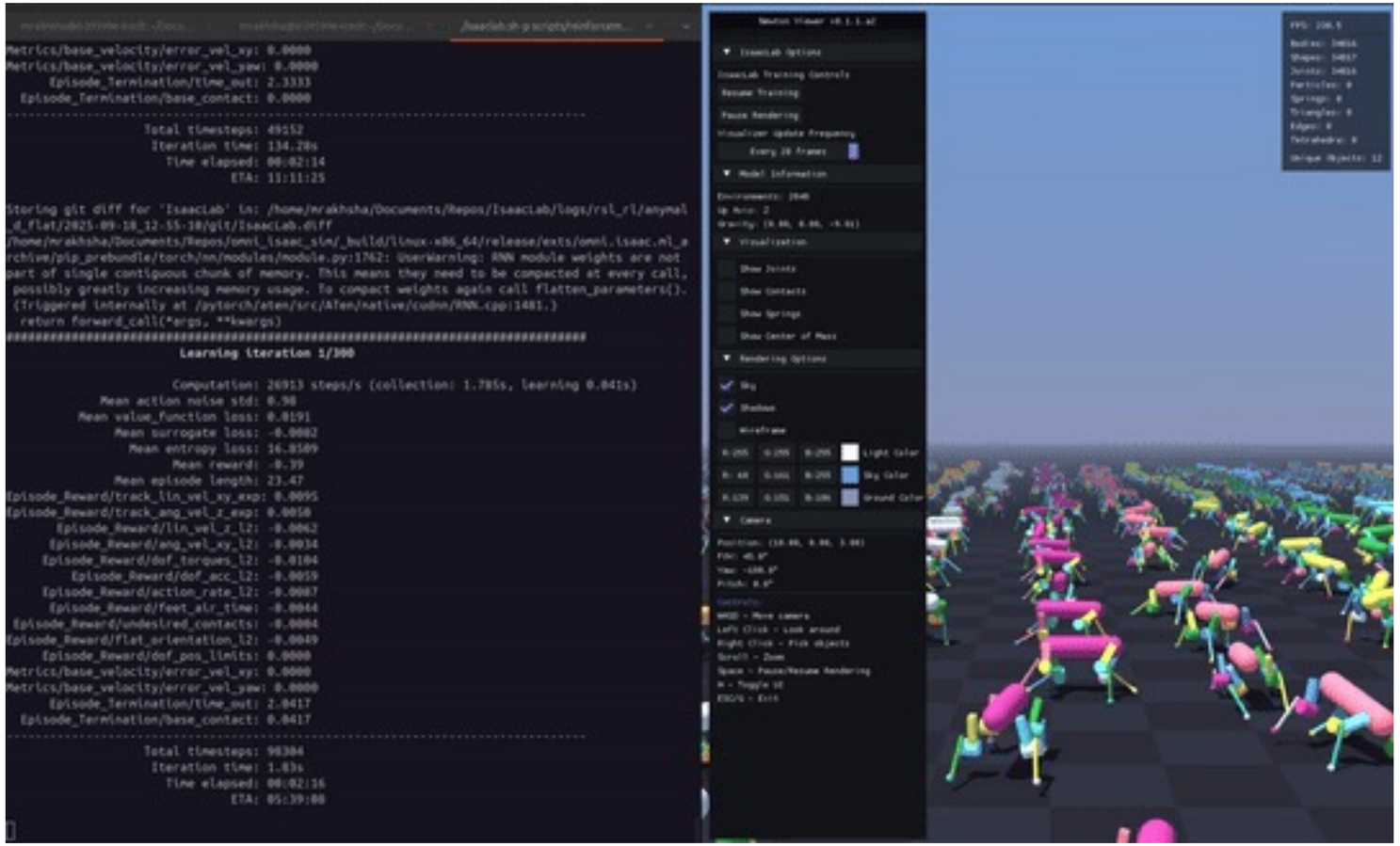
- For RL we often need to select a subset of DOFs batched by environment
- Query based on name instead of index to avoid relying on joint orderings
- The Selection API allows creating batched ”views” onto the Model and State properties
- Use regex & paths to create ArticulationViews
- Views isolate data & handle strides automatically

```
from newton.selection import ArticulationView

# create view onto data for one articulation in the Model
ants = ArticulationView(model, "/World/envs/*/Robot/torso")

# get strided joint arrays
ant_q = ants.get_attribute("joint_q", state_0)
ant_qd = ants.get_attribute("joint_qd", state_0)

# set dof states (masked)
ants.set_attribute("joint_q", state_0, xform, mask=mask)
ants.set_attribute("joint_qd", state_0, velocity, mask=mask)

# helpers for manipulating root state
ants.set_root_transforms(state_0, xform, mask=mask)
ants.set_root_velocities(state_0, velocity, mask=mask)
```

Isaac Lab Integration Highlights

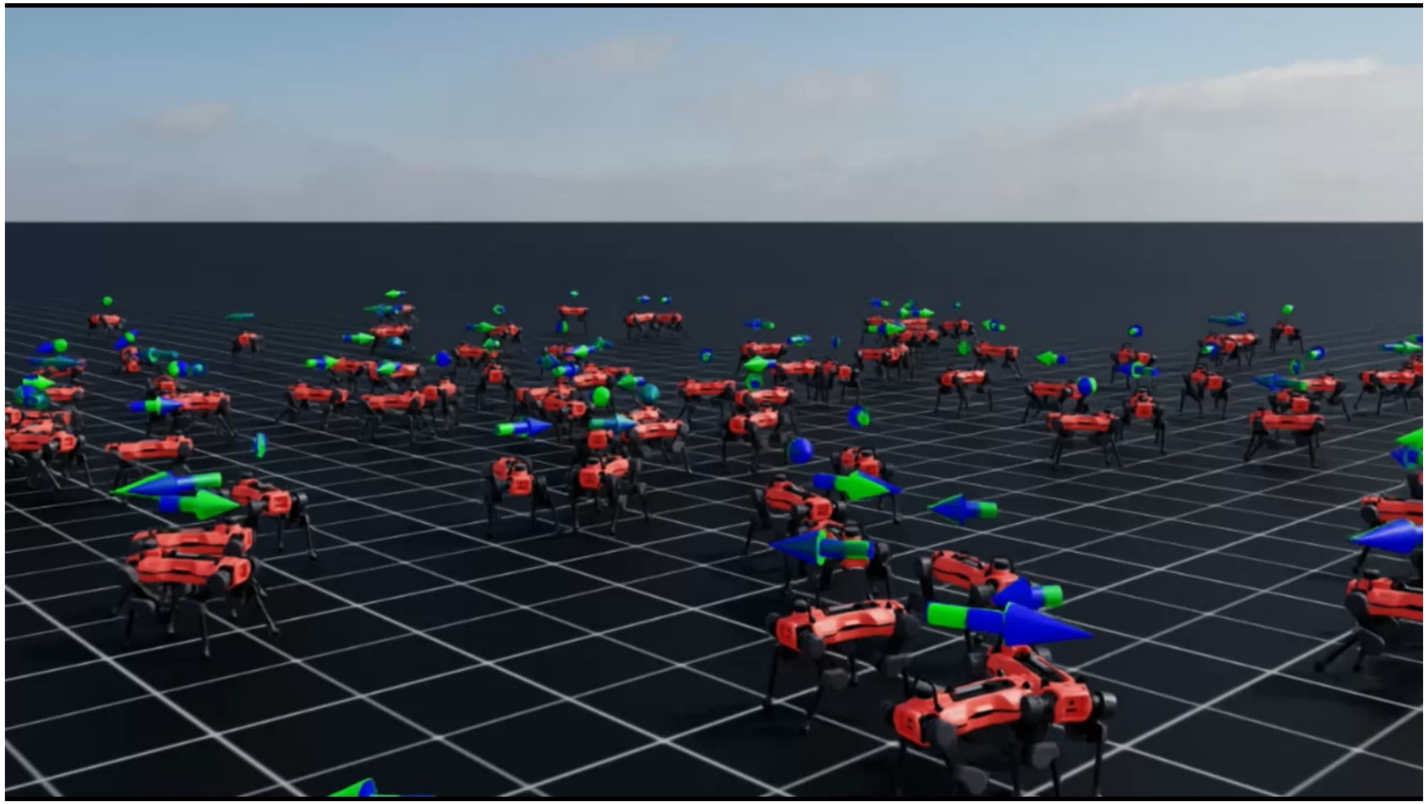
Training

```
./isaaclab.sh -p scripts/reinforcement_learning/rsl_rl/train.py
--task Isaac-Velocity-Flat-Anymal-D-v0 --num_envs 4096 --headless
--newton_visualizer
```



Isaac Lab Integration Highlights

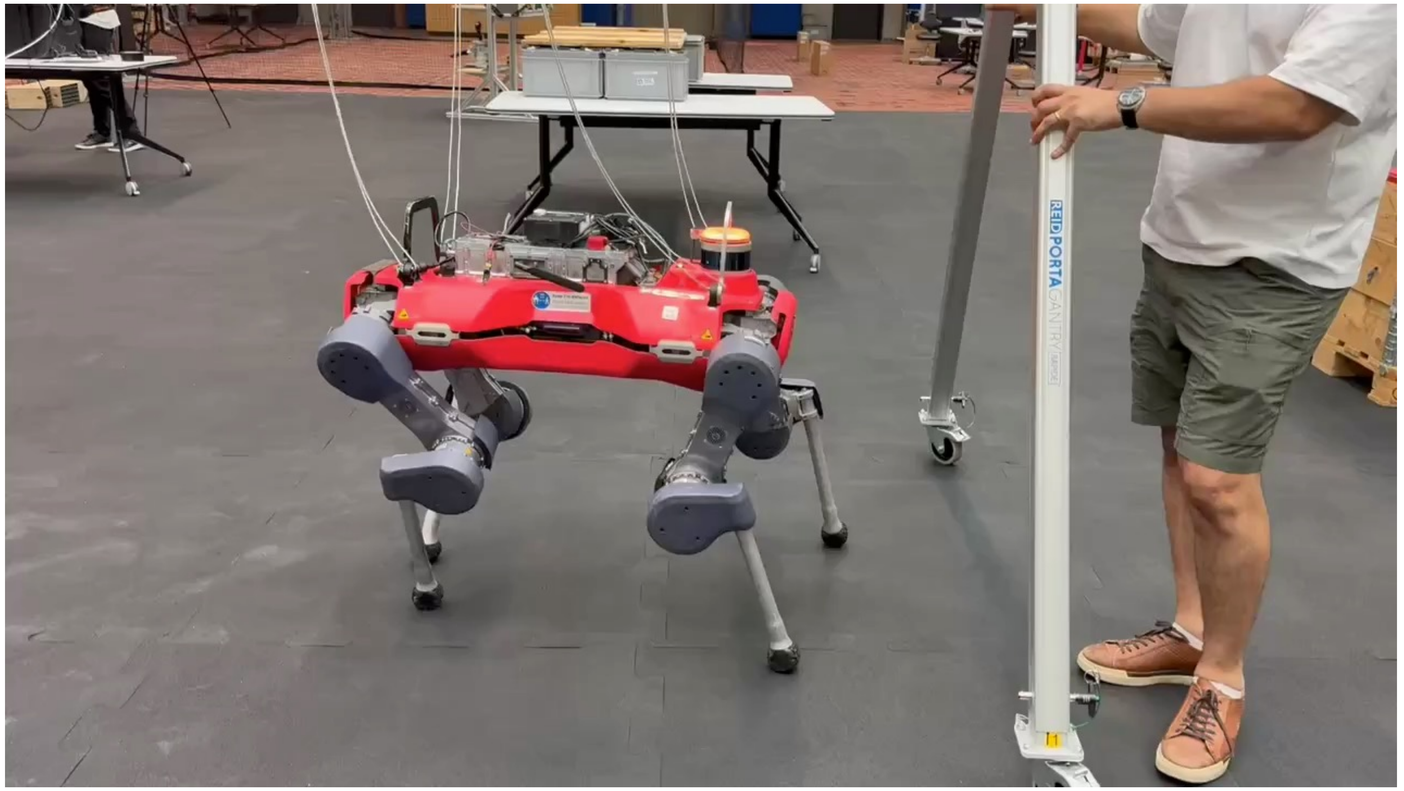
Sim2Sim



Newton to PhysX

Isaac Lab Integration Highlights

Sim2Real



Newton Timeline

MVP, Alpha, Beta, GA

2025 – June

Newton - MVP

2025 – Aug

Newton - Alpha
Early contributors
Official Linux Foundation Project
Lab integration, locomotion focus

2025 – Sept

CoRL - Beta Release
MuJoCo Warp – GA
Neural Solvers
Lab integration, manipulation focus

< 2026 - Mar

Newton 1.0 (GA)
Isaac Lab 3.0 Release
Kamino (DR) Solver

Newton is Open Source under Apache 2.0

Disney Research

DeepMind

NVIDIA

https://github.com/newton-physics

Newton Talks

Announcing MuJoCo-Warp and Newton: How Google DeepMind and NVIDIA are Supercharging Robotics Development

Erik Frey, Google DeepMind

GTC, San Jose, March 2025

How Robots Learn To Be Robots in Simulation With the Newton Physics Engine

Yuval Tassa, Google DeepMind
Miles Macklin, NVIDIA

GTC/VivaTech, Paris, June 2025

How Disney Droids Come to Life With Physics Simulation

Moritz Bächer, Disney Research
Mohammad Mohajerani, NVIDIA

SIGGRAPH, Vancouver, August 2025


NVIDIA
Thanks

More Resources

- Newton: https://github.com/newton-physics/newton
- Warp: https://nvidia.github.io/warp/ , https://nvidia.github.io/warp/publications.html
- Mujoco Warp: https://github.com/google-deepmind/mujoco_warp
- IsaacLab Integration: https://isaac-sim.github.io/IsaacLab/main/source/experimental-features/newton-physics-integration/index.html
- Neural Robot Dynamics : https://neural-robot-dynamics.github.io/
- Taccel: https://taccel-simulator.github.io/supercharging