GRAMAZIO
KOHLER
RESEARCH

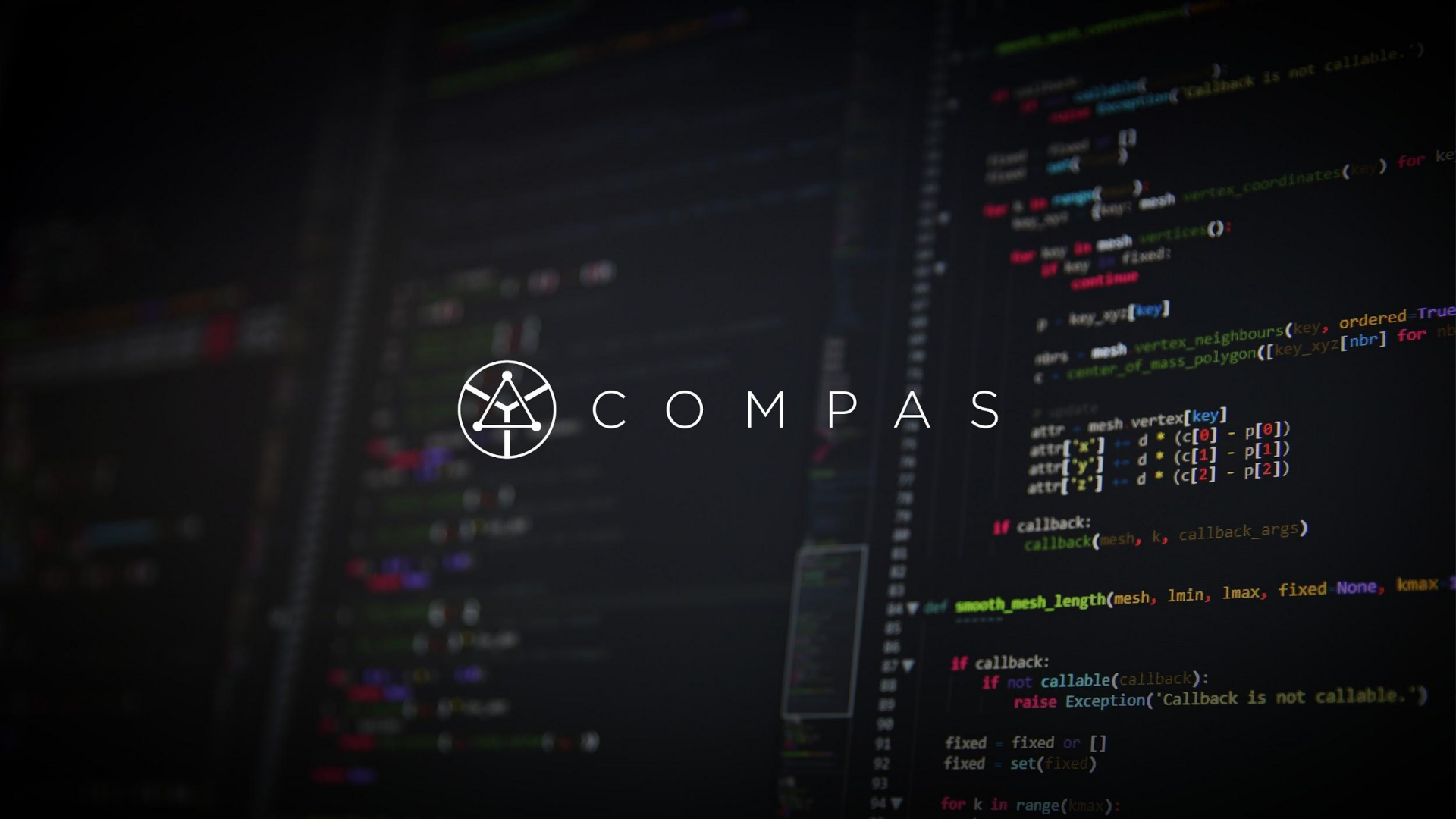
ETH zürich


COMPAS FAB

https://u.nu/tokyo20

AGENDA

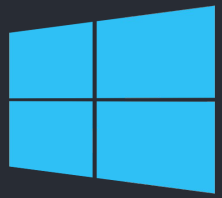
Intro & session goals
COMPAS 101
COMPAS FAB intro
Robotics
ROS primer
Kinematics & Motion planning


COMPAS

Getting started

https://u.nu/tokyo20

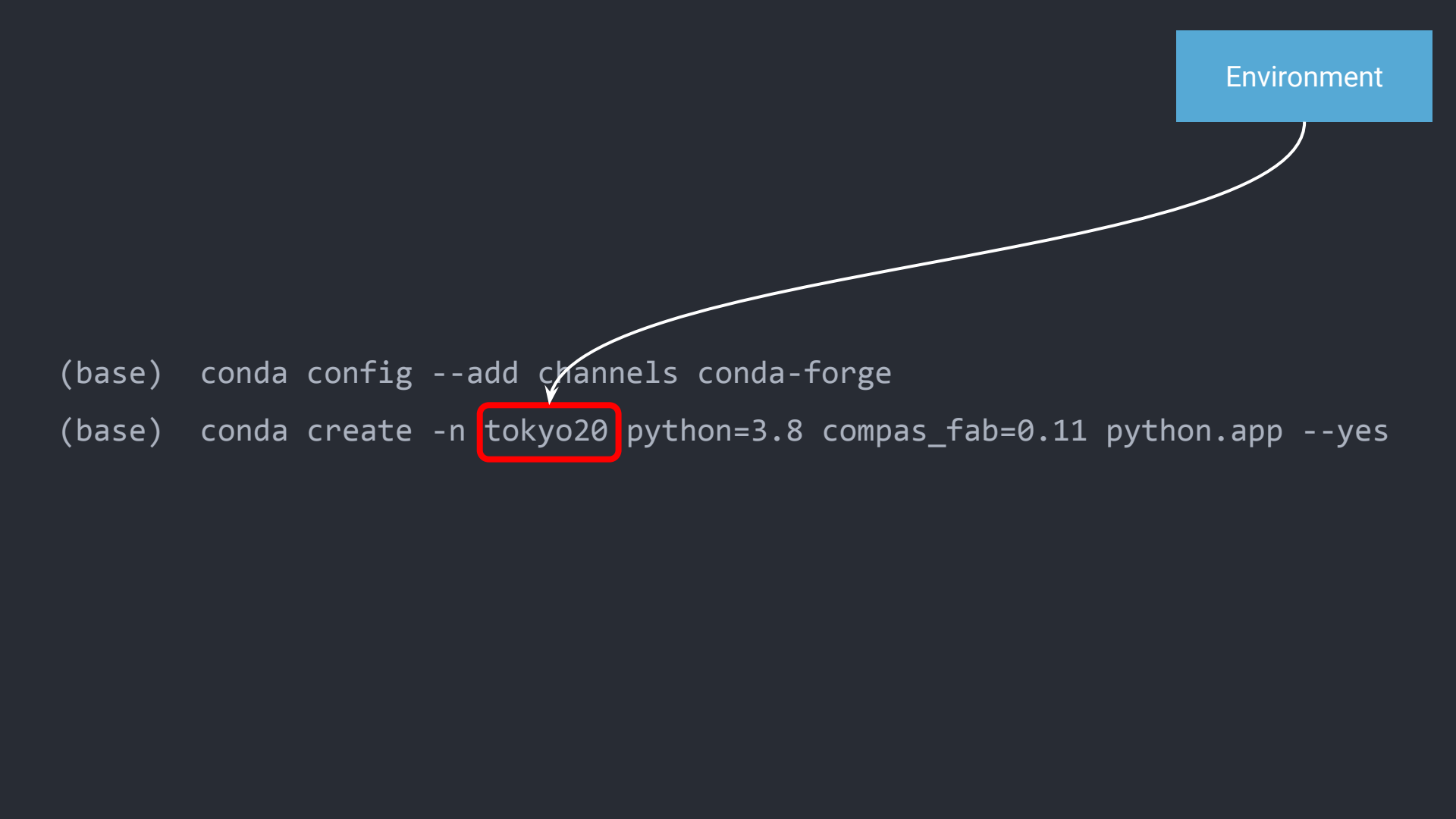
⌘ + SPACE

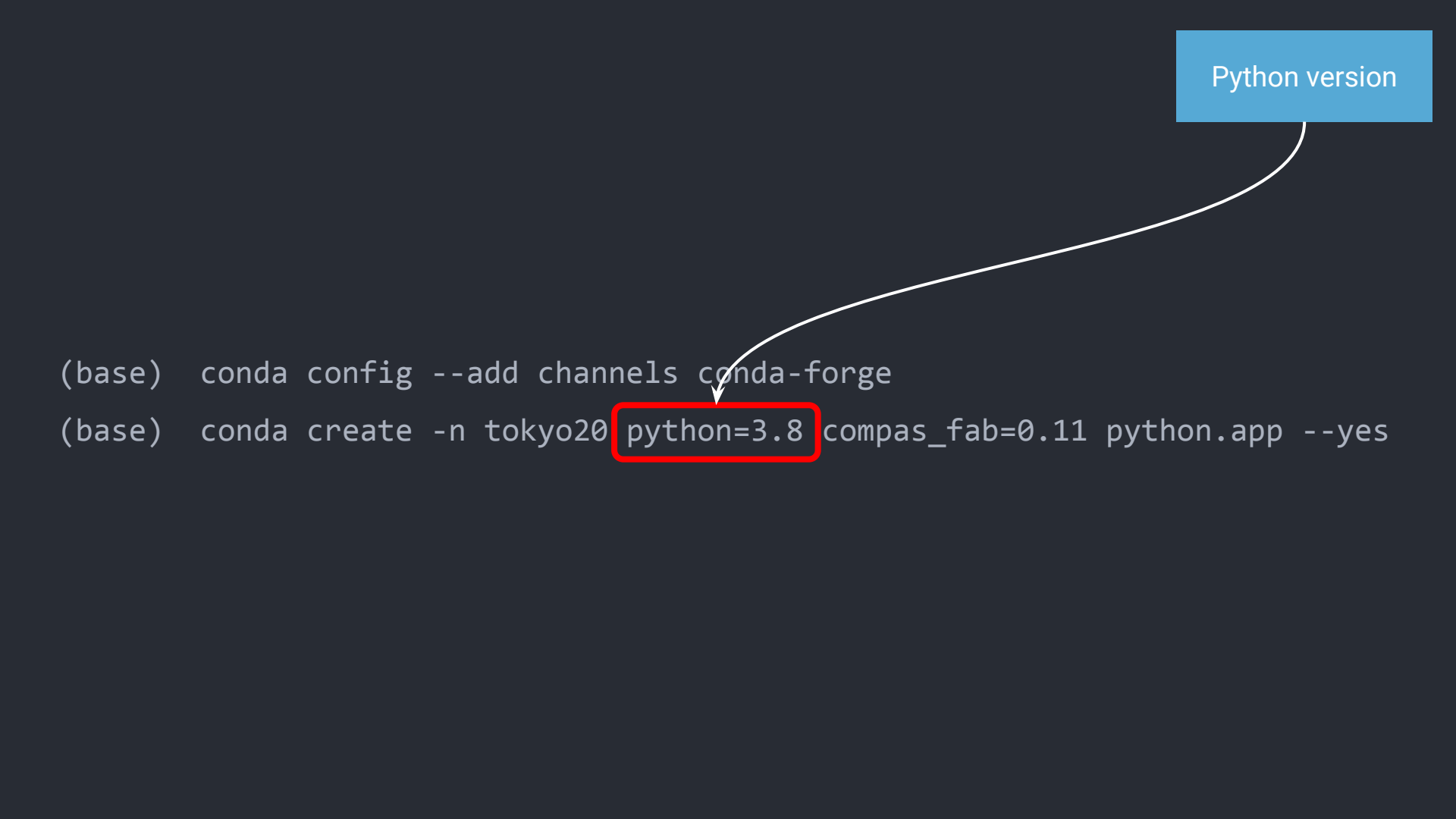


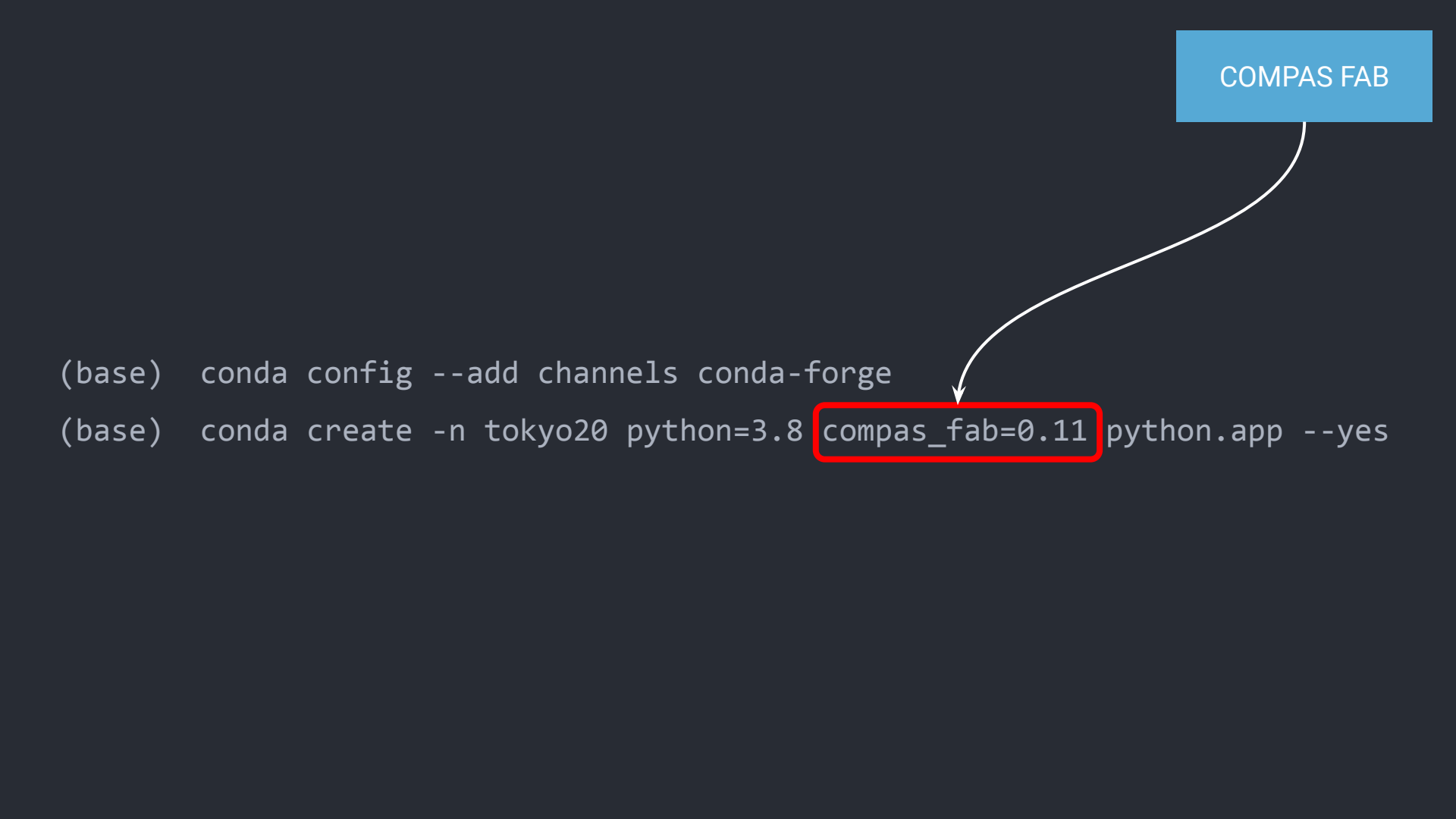
Terminal

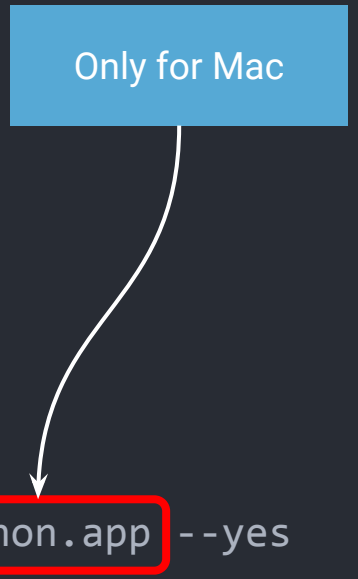
Anaconda Prompt

```
(base)  conda config --add channels conda-forge
(base)  conda create -n tokyo20 python=3.8 compas_fab=0.11 python.app --yes
```

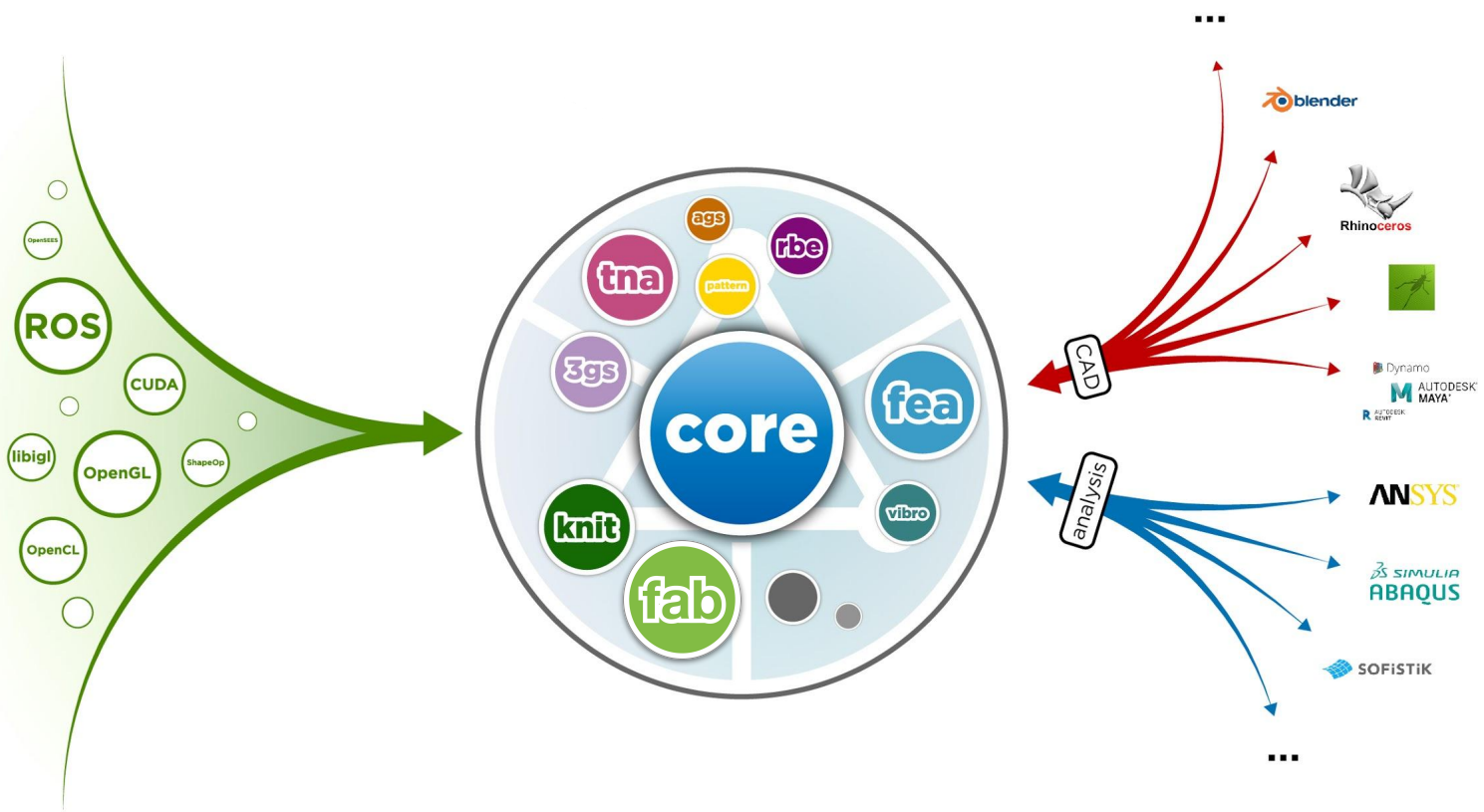

Environment
(base) conda config --add channels conda-forge
(base) conda create -n tokyo20 python=3.8 compas_fab=0.11 python.app --yes


Python version
(base) conda config --add channels conda-forge
(base) conda create -n tokyo20 python=3.8 compas_fab=0.11 python.app --yes

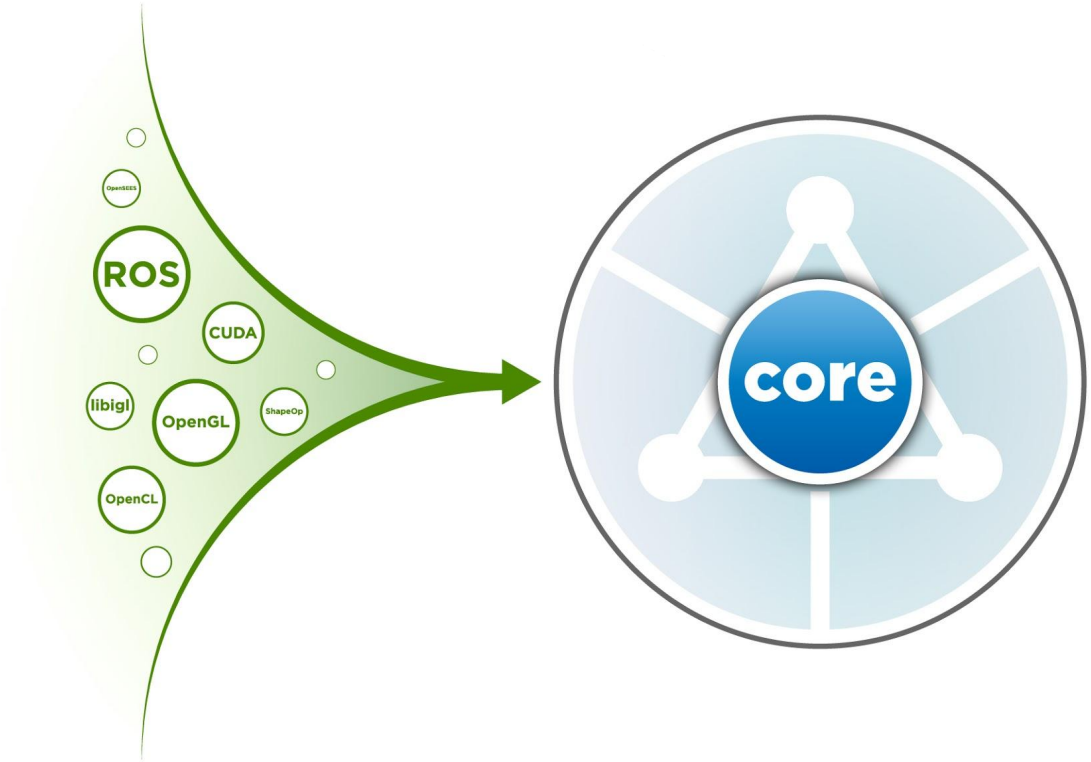

COMPAS FAB
(base) conda config --add channels conda-forge
(base) conda create -n tokyo20 python=3.8 compas_fab=0.11 python.app --yes

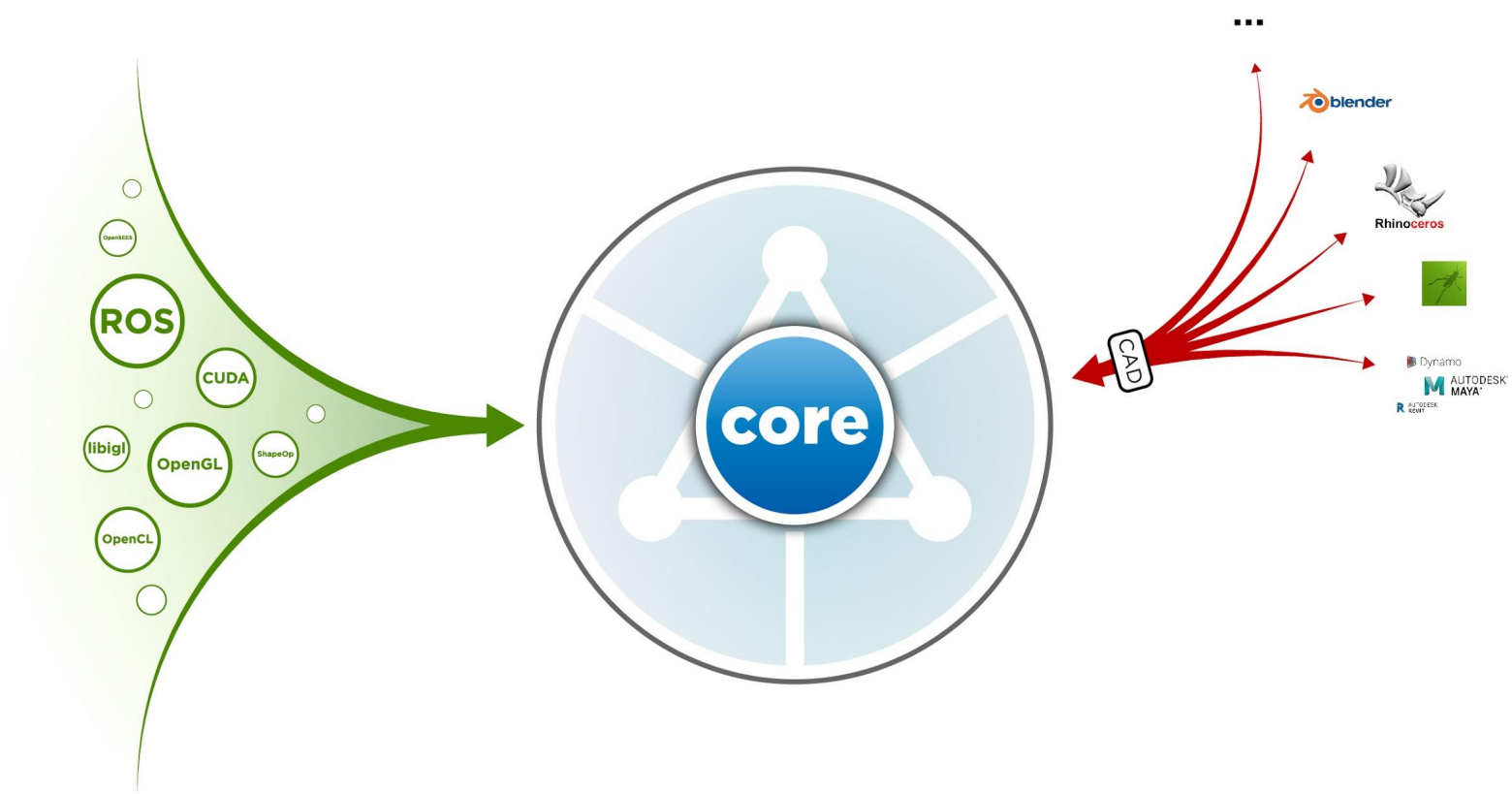


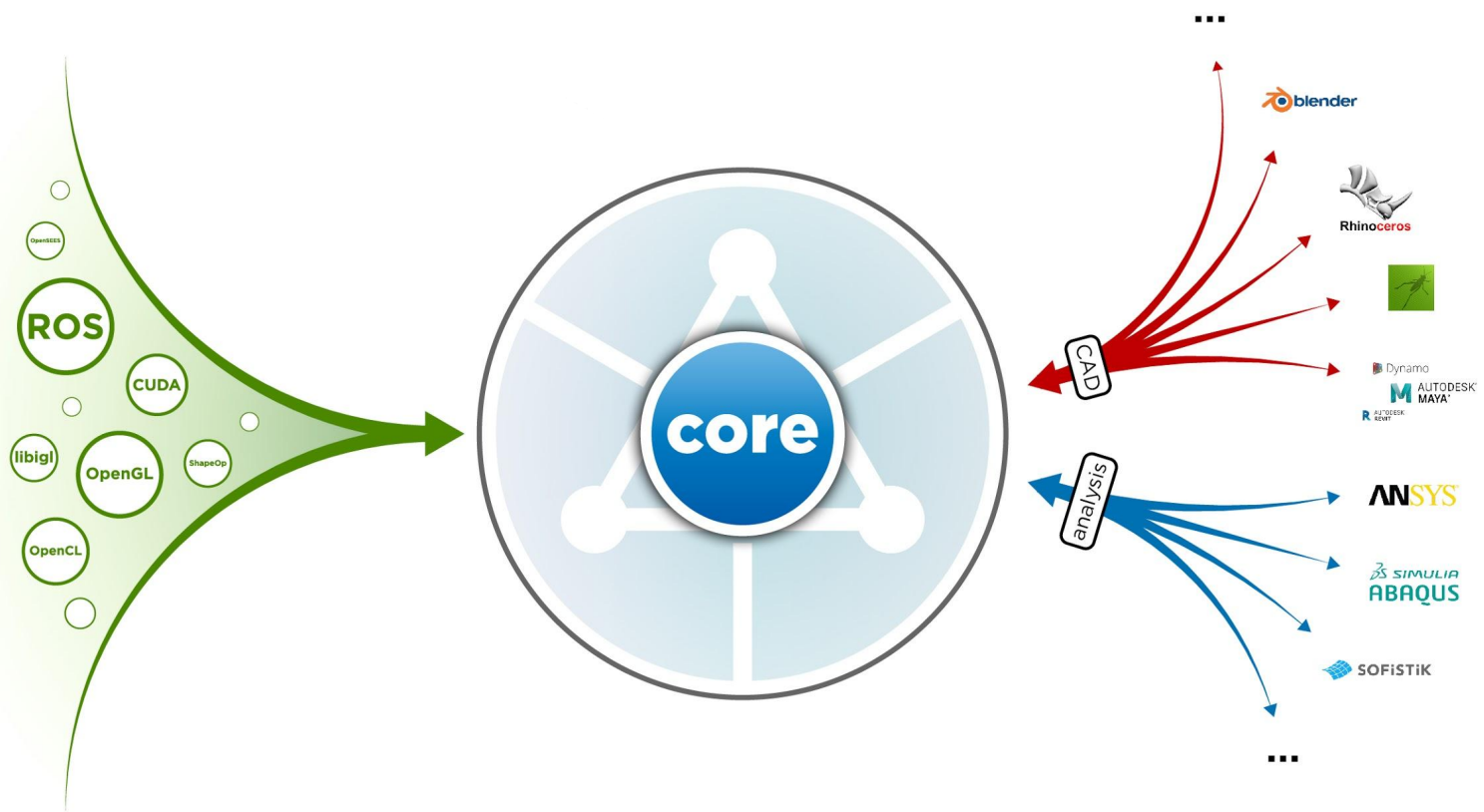

```
(base)  conda config --add channels conda-forge
(base)  conda create -n tokyo20 python=3.8 compas_fab=0.11 python.app --yes
```

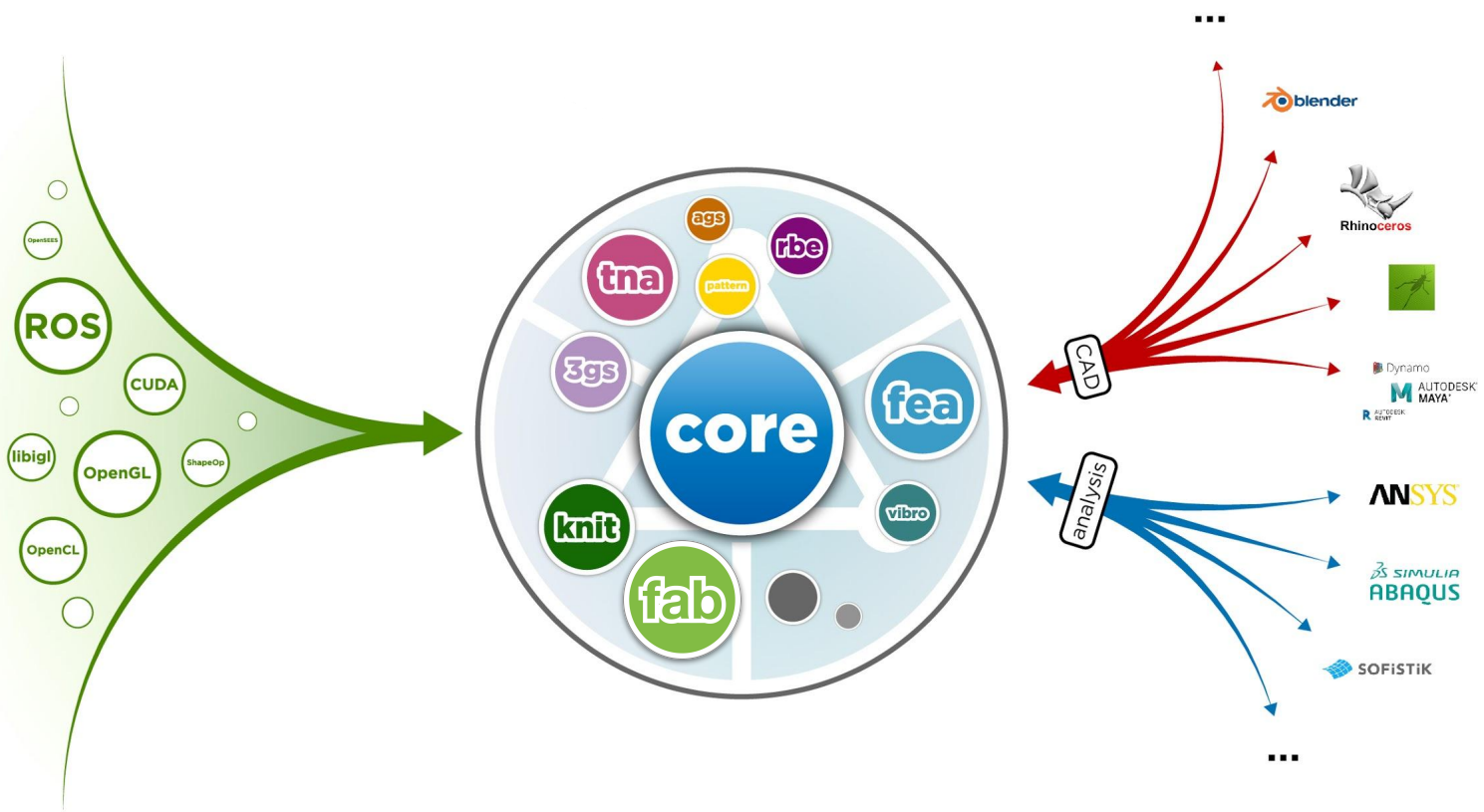

...
blender
Rhinoceros
Dynamo
AUTODESK MAYA
AUTODESK REVIT
CAD
analysis
ANSYS
SIMULIA ABAQUS
SOFiSTiK
...
OpenSEES
ROS
CUDA
libigl
OpenGL
ShapeOp
OpenCL
ags
rbe
tna
pattern
3gs
core
fea
knit
vibro
fab

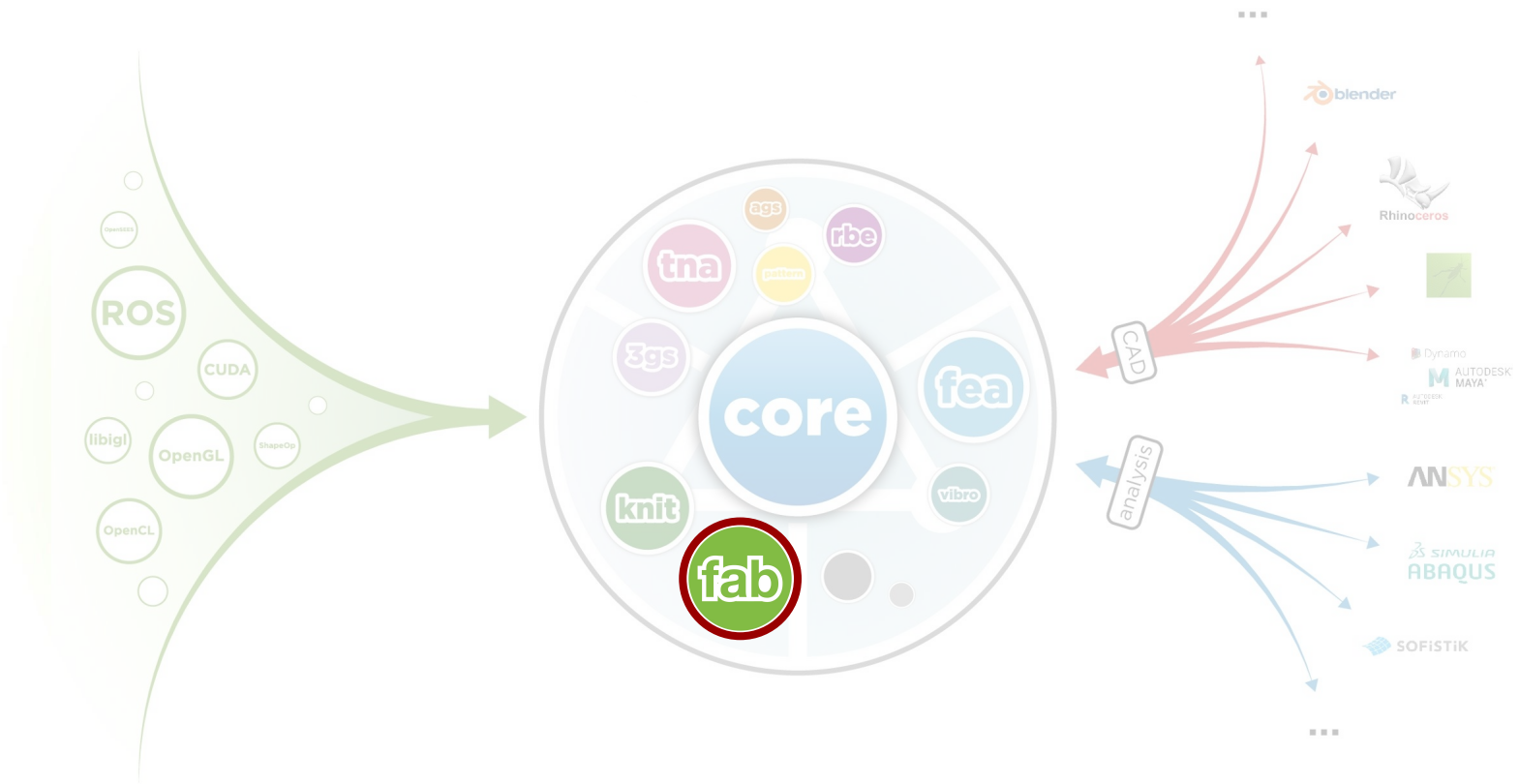
- Open source, Python
- Flexible data structures
- Algorithms & utilities for AEC
- Geometry processing
- CAD independent

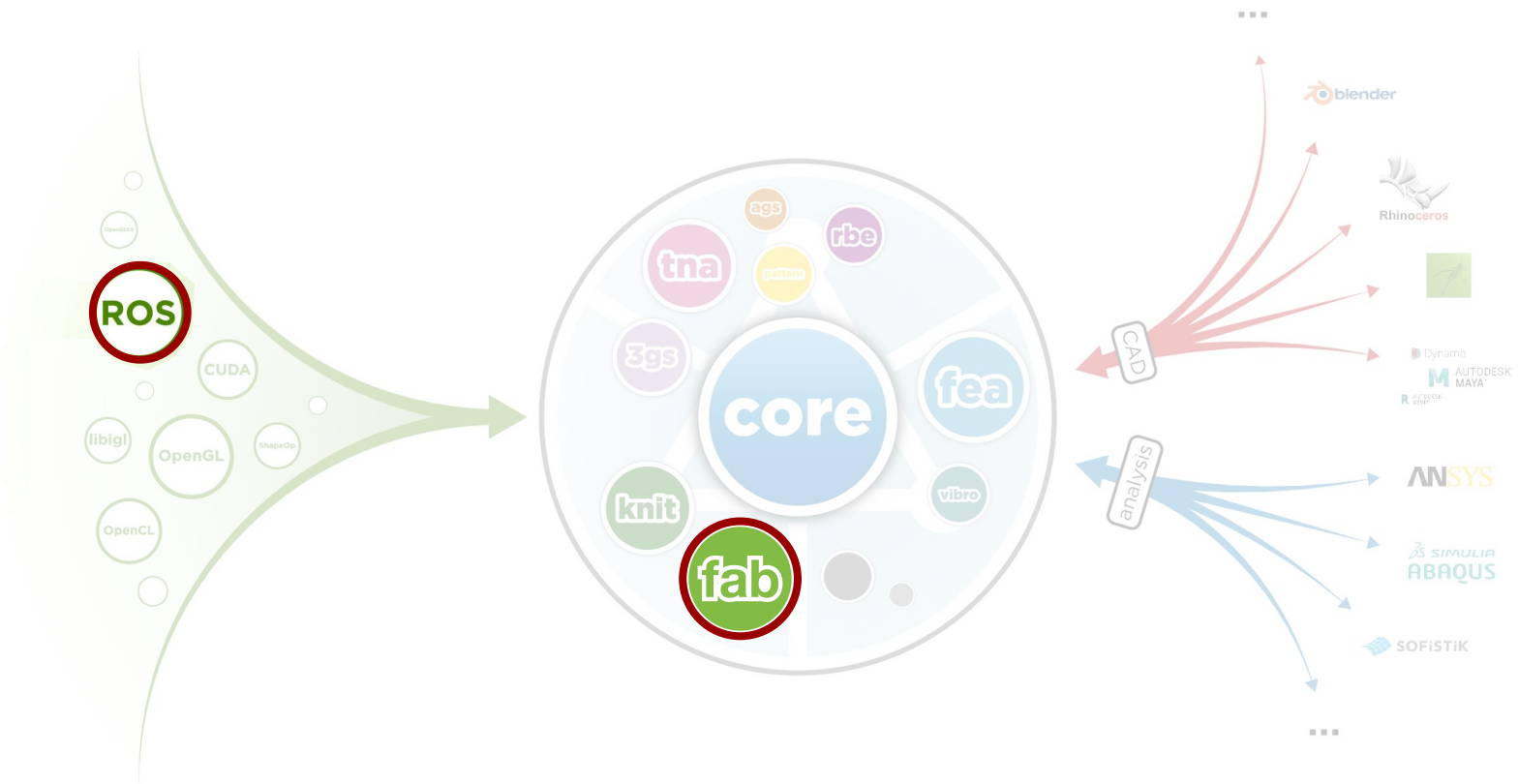

OpenSEES
ROS
CUDA
libigl
OpenGL
ShapeOp
OpenCL
core

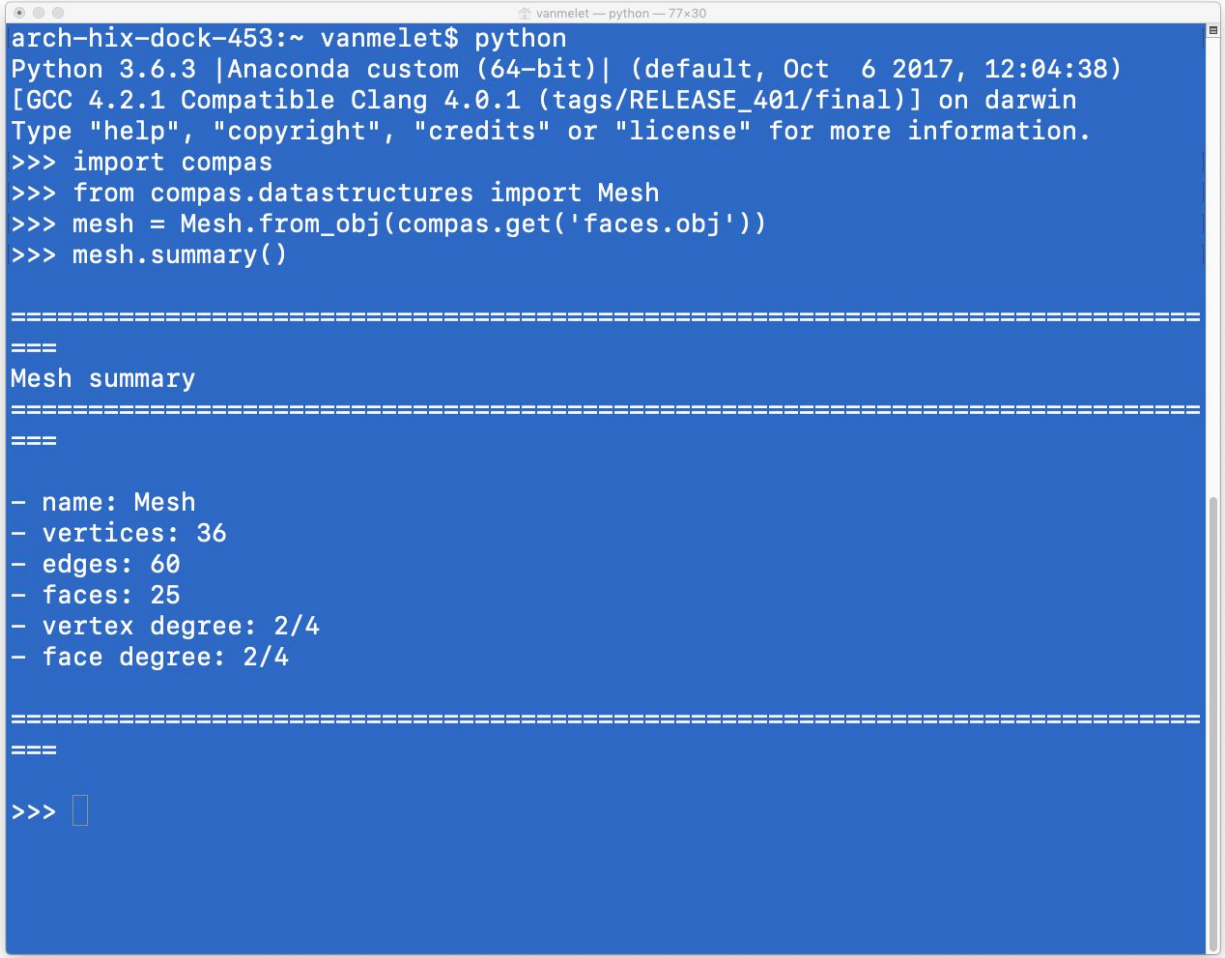

...
OpenSEES
ROS
CUDA
libigl
OpenGL
ShapeOp
OpenCL
core
CAD
blender
Rhinoceros
Dynamo
AUTODESK MAYA


...
blender
Rhinoceros
Dynamo
AUTODESK MAYA
CAD
analysis
ANSYS
SIMULIA ABAQUS
SOFiSTiK
...
core
ROS
CUDA
libigl
OpenGL
ShapeOp
OpenCL

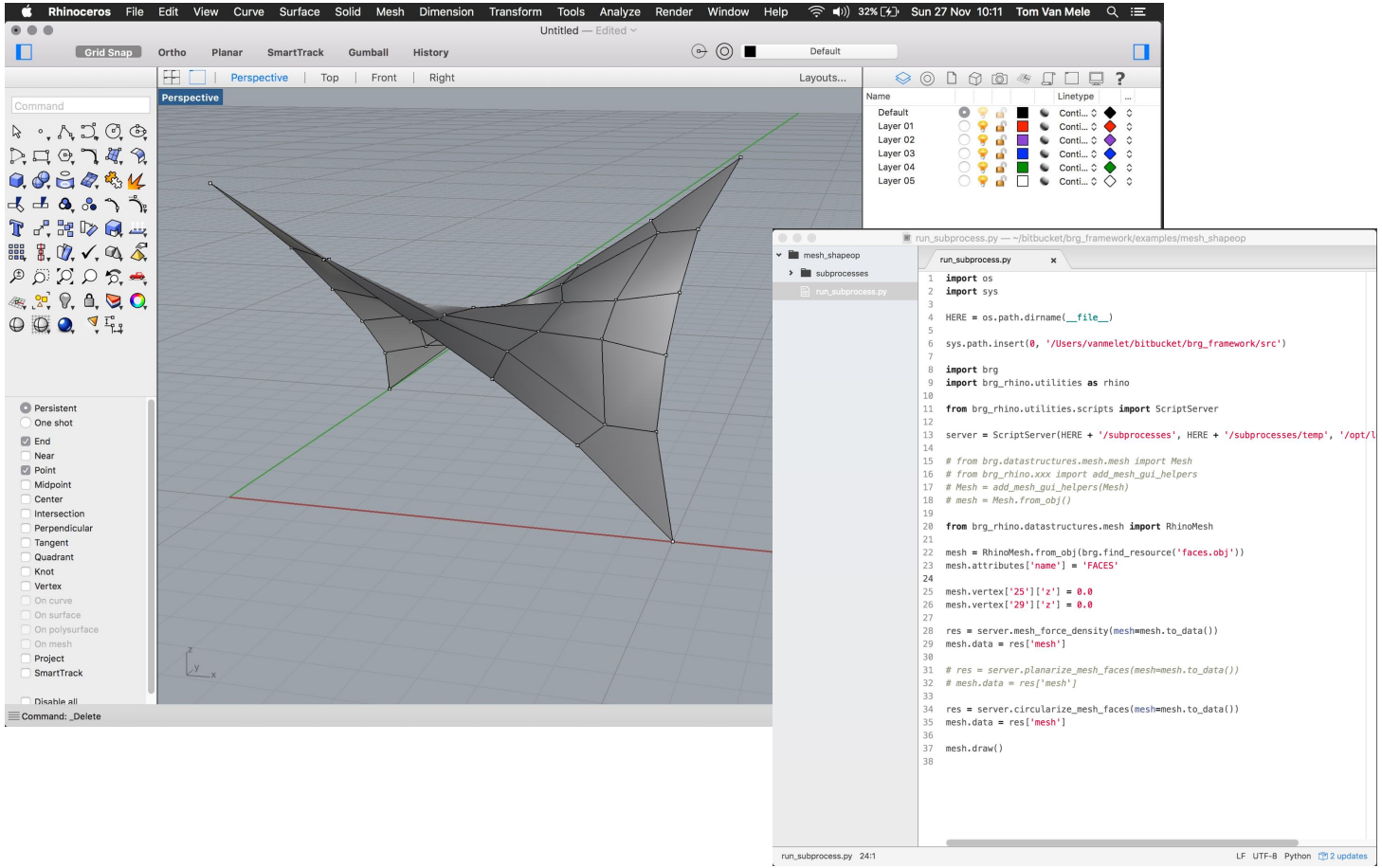

...
blender
Rhinoceros
Dynamo
AUTODESK MAYA
AUTODESK REVIT
CAD
analysis
ANSYS
SIMULIA ABAQUS
SOFiSTiK
...
OpenSEES
ROS
CUDA
libigl
OpenGL
ShapeOp
OpenCL
ags
rbe
tna
pattern
3gs
core
fea
knit
vibro
fab

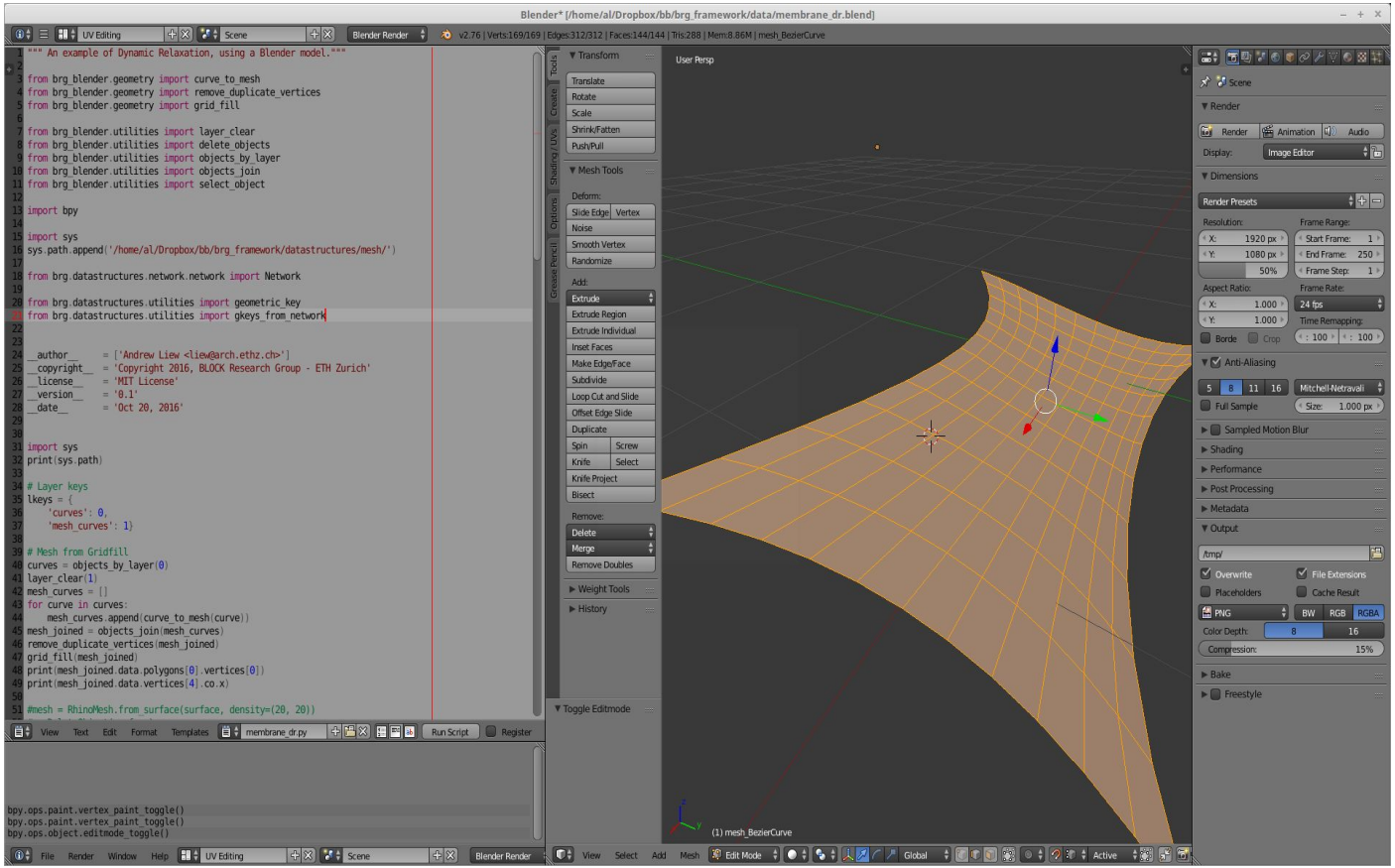

ROS
CUDA
libigl
OpenGL
ShapeOp
OpenCL
ags
rbe
tna
pattern
3gs
core
fea
knit
vibro
fab
CAD
analysis
blender
Rhinoceros
Dynamo
AUTODESK MAYA
ANSYS
SIMULIA ABAQUS
SOFiSTiK


ROS
CUDA
libigl
OpenGL
ShapeOp
OpenCL
ags
rbe
tna
pattern
3gs
core
fea
knit
vibro
fab
CAD
analysis
...
blender
Rhinoceros
Dynamo
AUTODESK MAYA
ANSYS
SIMULIA ABAQUS
SOFiSTiK
...


vanmelet — python — 77×30
arch-hix-dock-453:~ vanmelet$ python
Python 3.6.3 |Anaconda custom (64-bit)| (default, Oct 6 2017, 12:04:38)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import compas
>>> from compas.datastructures import Mesh
>>> mesh = Mesh.from_obj(compas.get('faces.obj'))
>>> mesh.summary()
==========
===
Mesh summary
==========
===
- name: Mesh
- vertices: 36
- edges: 60
- faces: 25
- vertex degree: 2/4
- face degree: 2/4
==========
===
>>>


dr_numpy.py — compas-dev
FOLDERS
compas-dev
compas
build
data
dist
docs
libs
src
compas
__pycache__
com
datastructures
files
geometry
interop
numerical
__pycache__
_alglib
alglib
algorithms
descent
devo
dr
drx
__pycache__
__init__.py
drx_numpy.py
fd
ga
lma
mma
pca
solvers
topop
__init__.py
linalg.py
matrices.py
operators.py
utilities.py
plotters
robots
rpc
topology
utilities
viewers
__init__.py
_os.py
COMPAS.egg-info
compas_blender
compas_ghpython
compas_hpc
compas_revit
compas_rhino
temp
tests
.bumpversion.cfg
.editorconfig
.gitignore
.travis.yml
CONTRIBUTORS.md
LICENSE
dr_numpy.py
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
import compas
try:
from numpy import array
from numpy import isnan
from numpy import isinf
from numpy import ones
from numpy import zeros
from scipy.linalg import norm
from scipy.sparse import diags
except ImportError:
compas.raise_if_not_ironpython()
from compas.numerical import connectivity_matrix
from compas.numerical import normrow
__all__ = ['dr_numpy']
K = [
[0.0],
[0.5, 0.5],
[0.5, 0.0, 0.5],
[1.0, 0.0, 0.0, 1.0],
]
class Coeff():
def __init__(self, c):
self.c = c
self.a = (1 - c * 0.5) / (1 + c * 0.5)
self.b = 0.5 * (1 + self.a)
def dr_numpy(vertices, edges, fixed, loads, qpre, fpre, lpre, linit, E, radius,
callback=None, callback_args=None, **kwargs):
"""Implementation of the dynamic relaxation method for form findong and analysis
of articulated networks of axial-force members.
Parameters
vertices : list
XYZ coordinates of the vertices.
edges : list
Connectivity of the vertices.
fixed : list
Indices of the fixed vertices.
loads : list
XYZ components of the loads on the vertices.
qpre : list
Prescribed force densities in the edges.
fpre : list
Prescribed forces in the edges.
lpre : list
Prescribed lengths of the edges.
linit : list
Initial length of the edges.
E : list
Stiffness of the edges.
radius : list
Radius of the edges.
callback : callable, optional
User-defined function that is called at every iteration.
callback_args : tuple, optional
Additional arguments passed to the callback.
Notes
For more info, see [1]_.
References
.. [1] De Laet L., Veenendaal D., Van Mele T., Mollaert M. and Block P.,
«Bending incorporated: designing tension structures by integrating
Proceedings of Tensinet Symposium 2013,Istanbul, Turkey, 2013.
Examples
.. plot::
:include-source:
import compas
from compas.datastructures import Network
Git branch: master, index: ✓, working: 10●, Line 33, Column 1


Rhinoceros File Edit View Curve Surface Solid Mesh Dimension Transform Tools Analyze Render Window Help
Sun 27 Nov 10:11 Tom Van Mele
Untitled — Edited
Grid Snap Ortho Planar SmartTrack Gumball History
Perspective Top Front Right Layouts...
Perspective
Command
Persistent
One shot
End
Near
Point
Midpoint
Center
Intersection
Perpendicular
Tangent
Quadrant
Knot
Vertex
On curve
On surface
On polysurface
On mesh
Project
SmartTrack
Disable all
Command: _Delete
Name
Linetype
Default
Layer 01
Layer 02
Layer 03
Layer 04
Layer 05
run_subprocess.py — ~/bitbucket/brg_framework/examples/mesh_shapeop
mesh_shapeop
subprocesses
run_subprocess.py
import os
import sys
HERE = os.path.dirname(__file__)
sys.path.insert(0, '/Users/vanmelet/bitbucket/brg_framework/src')
import brg
import brg_rhino.utilities as rhino
from brg_rhino.utilities.scripts import ScriptServer
server = ScriptServer(HERE + '/subprocesses', HERE + '/subprocesses/temp', '/opt/l
# from brg.datastructures.mesh.mesh import Mesh
# from brg_rhino.xxx import add_mesh_gui_helpers
# Mesh = add_mesh_gui_helpers(Mesh)
# mesh = Mesh.from_obj()
from brg_rhino.datastructures.mesh import RhinoMesh
mesh = RhinoMesh.from_obj(brg.find_resource('faces.obj'))
mesh.attributes['name'] = 'FACES'
mesh.vertex['25']['z'] = 0.0
mesh.vertex['29']['z'] = 0.0
res = server.mesh_force_density(mesh=mesh.to_data())
mesh.data = res['mesh']
# res = server.planarize_mesh_faces(mesh=mesh.to_data())
# mesh.data = res['mesh']
res = server.circularize_mesh_faces(mesh=mesh.to_data())
mesh.data = res['mesh']
mesh.draw()
run_subprocess.py 24:1
LF UTF-8 Python 2 updates


Blender* [/home/al/Dropbox/bb/brg_framework/data/membrane_dr.blend]
UV Editing
Scene
Blender Render
v2.76 | Verts:169/169 | Edges:312/312 | Faces:144/144 | Tris:288 | Mem:8.86M | mesh_BezierCurve
""" An example of Dynamic Relaxation, using a Blender model."""
from brg_blender.geometry import curve_to_mesh
from brg_blender.geometry import remove_duplicate_vertices
from brg_blender.geometry import grid_fill
from brg_blender.utilities import layer_clear
from brg_blender.utilities import delete_objects
from brg_blender.utilities import objects_by_layer
from brg_blender.utilities import objects_join
from brg_blender.utilities import select_object
import bpy
import sys
sys.path.append('/home/al/Dropbox/bb/brg_framework/datastructures/mesh/')
from brg.datastructures.network.network import Network
from brg.datastructures.utilities import geometric_key
from brg.datastructures.utilities import gkeys_from_network
__author__ = ['Andrew Liew <liew@arch.ethz.ch>']
__copyright__ = 'Copyright 2016, BLOCK Research Group - ETH Zurich'
__license__ = 'MIT License'
__version__ = '0.1'
__date__ = 'Oct 20, 2016'
import sys
print(sys.path)
# Layer keys
lkeys = {
'curves': 0,
'mesh_curves': 1}
# Mesh from Gridfill
curves = objects_by_layer(0)
layer_clear(1)
mesh_curves = []
for curve in curves:
mesh_curves.append(curve_to_mesh(curve))
mesh_joined = objects_join(mesh_curves)
remove_duplicate_vertices(mesh_joined)
grid_fill(mesh_joined)
print(mesh_joined.data.polygons[0].vertices[0])
print(mesh_joined.data.vertices[4].co.x)
#mesh = RhinoMesh.from_surface(surface, density=(20, 20))
View
Text
Edit
Format
Templates
membrane_dr.py
Run Script
Register
bpy.ops.paint.vertex_paint_toggle()
bpy.ops.paint.vertex_paint_toggle()
bpy.ops.object.editmode_toggle()
File
Render
Window
Help
UV Editing
Scene
Blender Render
Transform
Translate
Rotate
Scale
Shrink/Fatten
Push/Pull
Mesh Tools
Deform:
Slide Edge
Vertex
Noise
Smooth Vertex
Randomize
Add:
Extrude
Extrude Region
Extrude Individual
Inset Faces
Make Edge/Face
Subdivide
Loop Cut and Slide
Offset Edge Slide
Duplicate
Spin
Screw
Knife
Select
Knife Project
Bisect
Remove:
Delete
Merge
Remove Doubles
Weight Tools
History
Toggle Editmode
Tools
Create
Shading / UVs
Options
Grease Pencil
User Persp
(1) mesh_BezierCurve
View
Select
Add
Mesh
Edit Mode
Global
Active
Scene
Render
Render
Animation
Audio
Display:
Image Editor
Dimensions
Render Presets
Resolution:
X: 1920 px
Y: 1080 px
50%
Frame Range:
Start Frame: 1
End Frame: 250
Frame Step: 1
Aspect Ratio:
X: 1.000
Y: 1.000
Frame Rate:
24 fps
Time Remapping:
Borde
Crop
Anti-Aliasing
5
8
11
16
Mitchell-Netravali
Full Sample
Size: 1.000 px
Sampled Motion Blur
Shading
Performance
Post Processing
Metadata
Output
/tmp/
Overwrite
File Extensions
Placeholders
Cache Result
PNG
BW
RGB
RGBA
Color Depth:
8
16
Compression:
15%
Bake
Freestyle

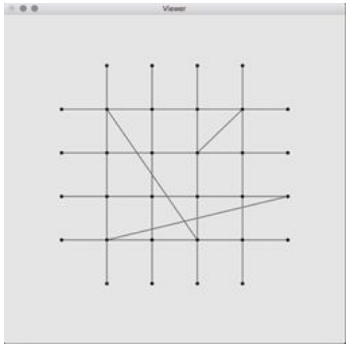
```python
# Rhino
from compas.datastructures import Mesh
from compas_rhino.artists import MeshArtist

mesh = Mesh.from_obj('https://u.nu/hypar')

artist = MeshArtist(mesh)

artist.draw_mesh()
```

```python
# Blender
from compas.datastructures import Mesh
from compas_blender.artists import MeshArtist

mesh = Mesh.from_obj('https://u.nu/hypar')

artist = MeshArtist(mesh)

artist.draw_mesh()
```

```python
# Rhino
from compas.datastructures import Mesh
from compas_rhino.artists import MeshArtist

mesh = Mesh.from_obj('https://u.nu/hypar')

artist = MeshArtist(mesh)

artist.draw_mesh()
```

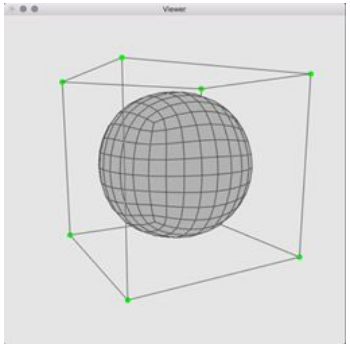
```python
# Blender
from compas.datastructures import Mesh
from compas_blender.artists import MeshArtist

mesh = Mesh.from_obj('https://u.nu/hypar')

artist = MeshArtist(mesh)

artist.draw_mesh()
```

Data structures



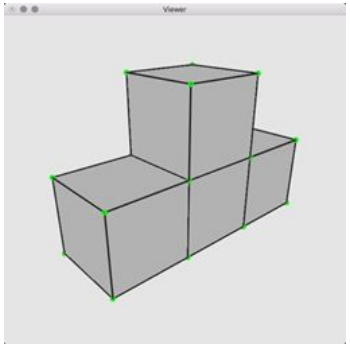
Network

- General networks
- Directed edge graph



Mesh

- Surface meshes
- Half-edge



Volumetric Mesh

- Cellular meshes
- Half-plane

Algorithms

Geometry

- Smoothing
- Planarization
- Convex hull
- Triangulation
- Transformations
- Bounding boxes
- Isolines
- ...

Numerical

- Dynamic relaxation
- Force density
- Topology optimization
- Genetic algorithms
- Principal component analysis
- Differential evolution
- ...

Topology

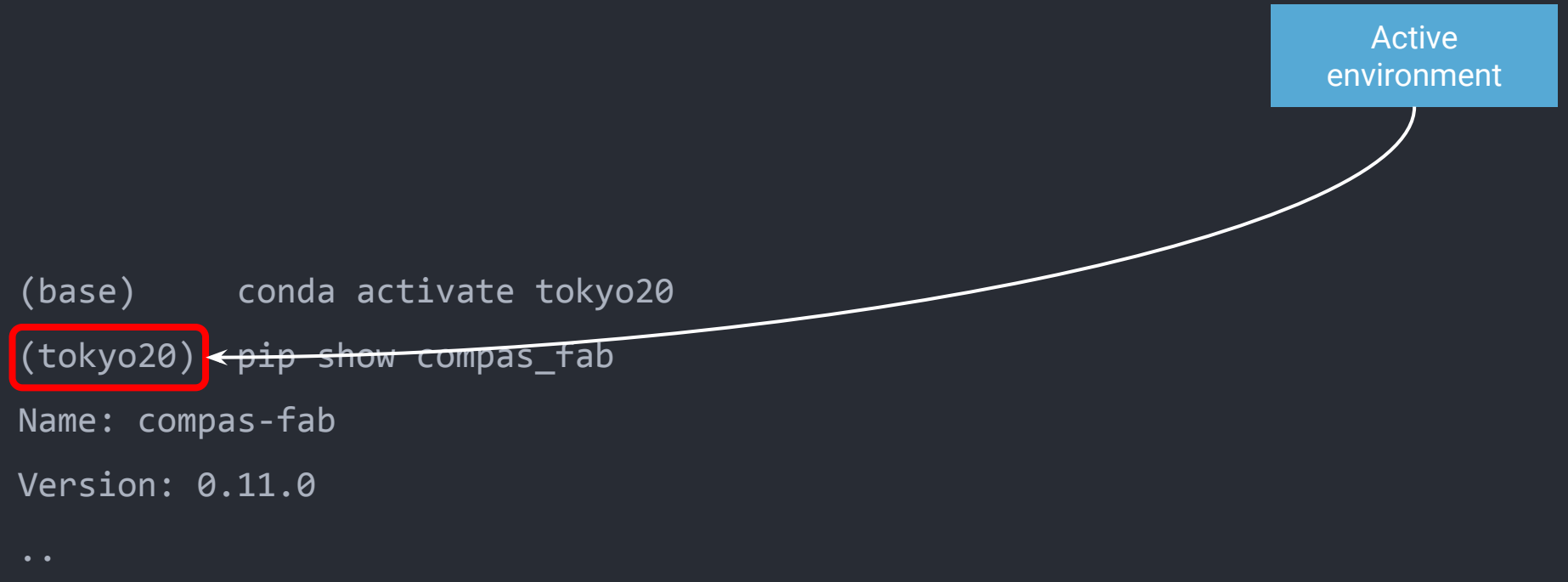
- Traversal
- Graph colouring
- Planarity
- Conway operators
- ...

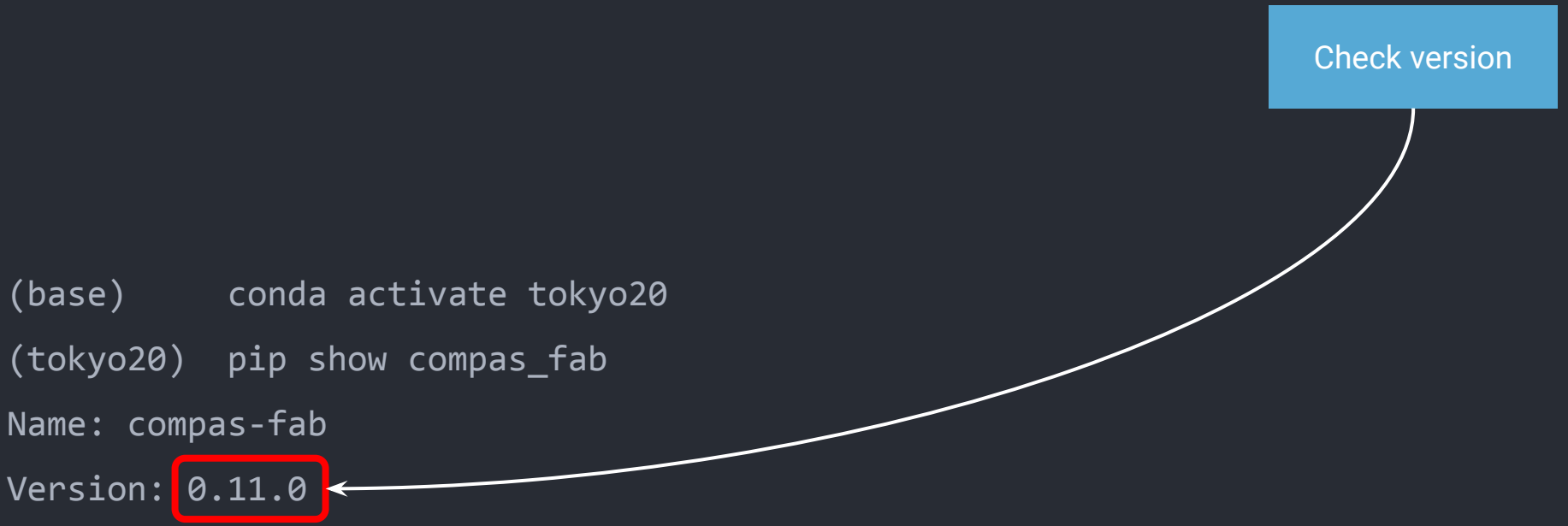
COMPAS **FAB**

Installation

https://u.nu/tokyo20

```
(base)     conda activate tokyo20
(tokyo20)  pip show compas_fab
Name: compas-fab
Version: 0.11.0
..
```


Active environment
(base) conda activate tokyo20
(tokyo20) pip show compas_fab
Name: compas-fab
Version: 0.11.0
..


Check version
(base) conda activate tokyo20
(tokyo20) pip show compas_fab
Name: compas-fab
Version: 0.11.0
..

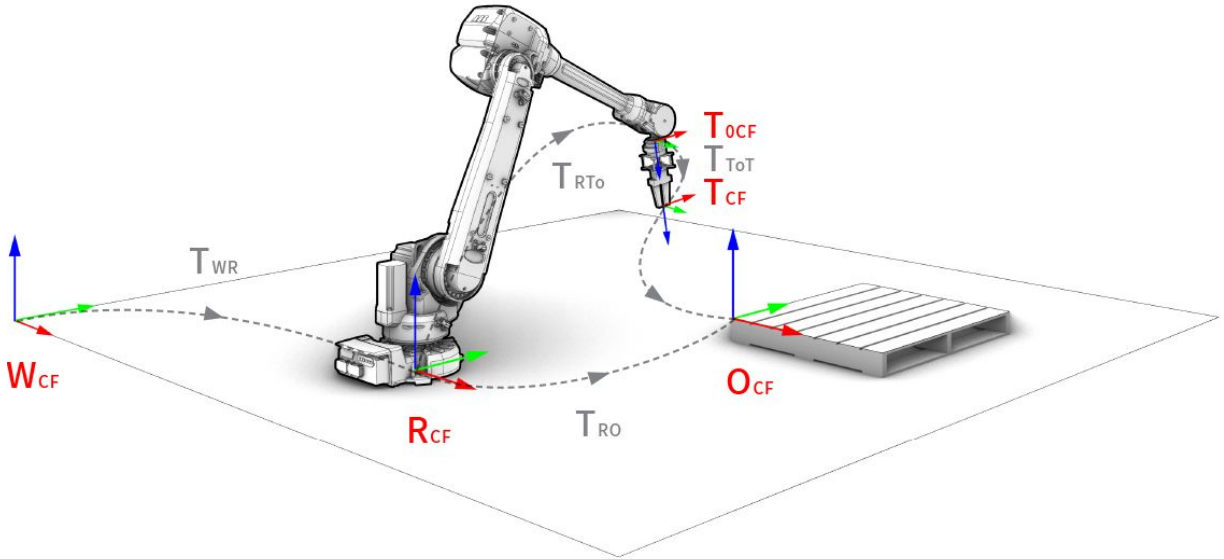
```
(tokyo20)  python -m compas_fab.rhino.install
```

Robotics

Fundamentals | Models | Backends

Robotic **fundamentals**

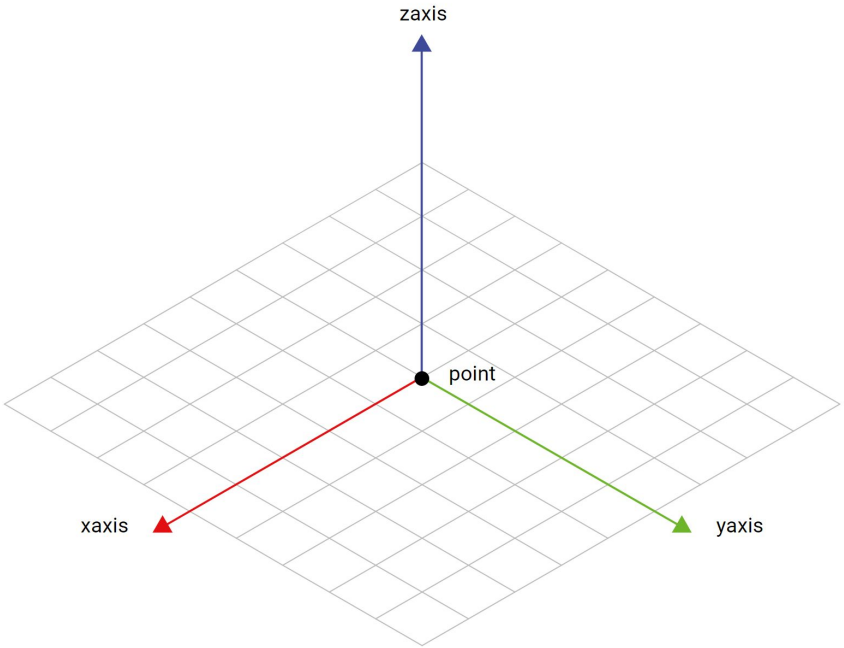
Frame and Transformation



Frame

A frame is defined by a base point and two orthonormal base vectors (xaxis, yaxis), which specify the normal (zaxis).

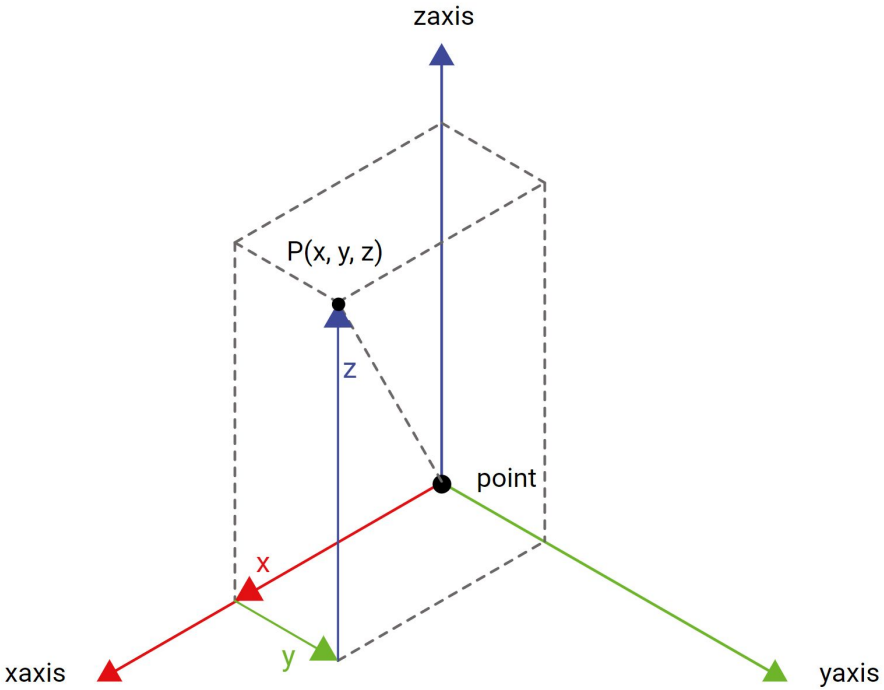
It describes location and orientation in a (right-handed) cartesian coordinate system.



```python
"""There are several ways to construct a `Frame`.
"""
from compas.geometry import Point
from compas.geometry import Vector
from compas.geometry import Frame
from compas.geometry import Plane

# Frame autocorrects axes to be orthonormal
F = Frame(Point(1, 0, 0), Vector(-0.45, 0.1, 0.3), Vector(1, 0, 0))

F = Frame([1, 0, 0], [-0.45, 0.1, 0.3], [1, 0, 0])

F = Frame.from_points([1, 1, 1], [2, 3, 6], [6, 3, 0])
F = Frame.from_plane(Plane([0, 0, 0], [0.5, 0.2, 0.1]))
F = Frame.from_euler_angles([0.5, 1., 0.2])
F = Frame.worldXY()
```


```python
"""There are several ways to construct a `Frame`.
"""
from compas.geometry import Point
from compas.geometry import Vector
from compas.geometry import Frame
from compas.geometry import Plane

# Frame autocorrects axes to be orthonormal
F = Frame(Point(1, 0, 0), Vector(-0.45, 0.1, 0.3), Vector(1, 0, 0))

F = Frame([1, 0, 0], [-0.45, 0.1, 0.3], [1, 0, 0])

F = Frame.from_points([1, 1, 1], [2, 3, 6], [6, 3, 0])
F = Frame.from_plane(Plane([0, 0, 0], [0.5, 0.2, 0.1]))
F = Frame.from_euler_angles([0.5, 1., 0.2])
F = Frame.worldXY()
```


Frame as a cartesian coordinate system

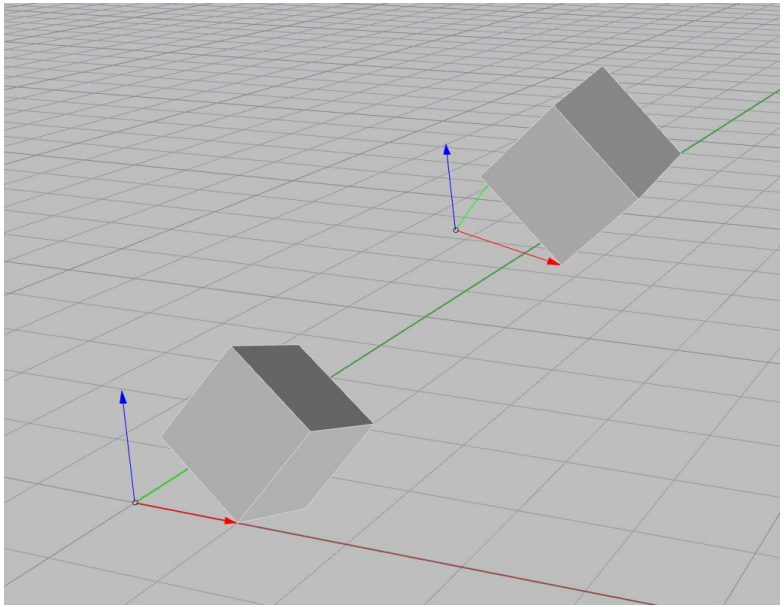


```
"""Example: 'point in frame'
"""
from compas.geometry import *

point = Point(146.00, 150.00, 161.50)
xaxis = Vector(0.9767, 0.0010, -0.214)
yaxis = Vector(0.1002, 0.8818, 0.4609)

F = Frame(point, xaxis, yaxis)  # coordinate system F
P = Point(35., 35., 35.)  # point in F (local coordinates)

P_ = F.to_world_coords(P)  # point in global (world) coordinates
print("The point's world coordinates: {}".format(P_))

P2 = F.to_local_coords(P_)
print("The point's local coordinates: {}".format(P2))  # should equal P
print(allclose(P2, P))
```


Example

1. Bring a box from the world coordinate system into another coordinate system

2. Draw frames and boxes in Rhino



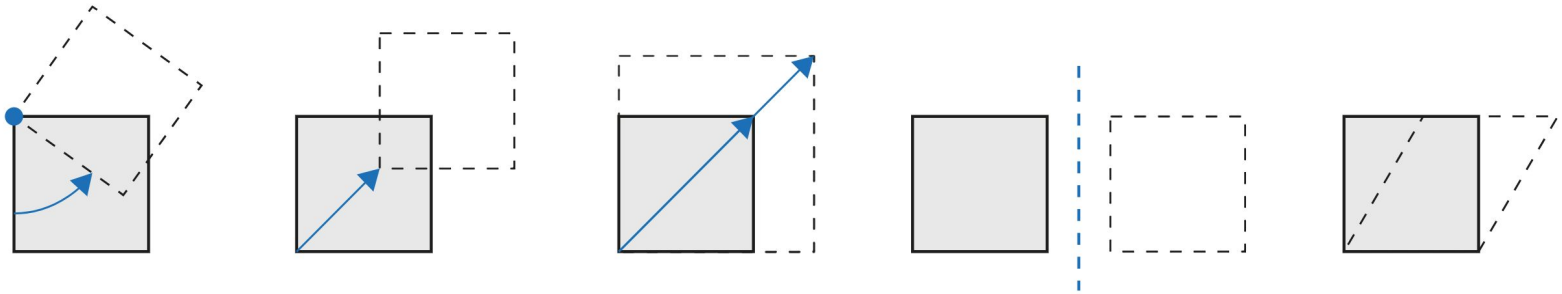
```
"""Example: Bring a box from the world coordinate system into another
coordinate system.
"""
from compas.geometry import Frame
from compas.geometry import Box

# Box in the world coordinate system
frame = Frame([1, 0, 0], [-0.45, 0.1, 0.3], [1, 0, 0])
width, length, height = 1, 1, 1
box = Box(frame, width, length, height)

# Frame F representing a coordinate system
F = Frame([2, 2, 2], [0.978, 0.010, -0.210], [0.090, 0.882, 0.463])

# Represent box frame in frame F and construct new box
box_frame_transformed = F.to_world_coords(box.frame)
box_transformed = Box(box_frame_transformed, width, length, height)
print("Box frame transformed:", box_transformed.frame)
```

```python
from compas.datastructures import Mesh
from compas_rhino.artists import FrameArtist
from compas_rhino.artists import MeshArtist

# create artists
artist1 = FrameArtist(Frame.worldXY())
artist2 = MeshArtist(Mesh.from_shape(box))
artist3 = FrameArtist(F)
artist4 = MeshArtist(Mesh.from_shape(box_transformed))

# draw
artist1.draw()
artist2.draw_faces()
artist3.draw()
artist4.draw_faces()
```

Transformation

Transformations refer to operations such as moving, rotating, and scaling objects. They are stored using 4x4 transformation matrices.

Most transformations preserve the parallel relationship among the parts of the geometry. For example collinear points remain collinear after the transformation. Also points on one plane stay coplanar after transformation. This type of transformation is called an *affine transformation* and concerns transformations such as `Rotation`, `Translation`, `Scale`, `Reflection`, `Shear` and orthogonal and parallel `Projection`. Only perspective `Projection` is not *affine*.



```python
"""Transformation examples
"""
from compas.geometry import *

axis, angle = [0.2, 0.4, 0.1], 0.3
R = Rotation.from_axis_and_angle(axis, angle)

translation_vector = [5, 3, 1]
T = Translation(translation_vector)

scale_factors = [0.1, 0.3, 0.4]
S = Scale(scale_factors)

point, normal = [0.3, 0.2, 1], [0.3, 0.1, 1]
R = Reflection(point, normal)

point, normal = [0, 0, 0], [0, 0, 1]
perspective = [1, 1, 0]
P = Projection.perspective(point, normal, perspective)
```

```
"""Example: Bring a box from the world coordinate system into another coordinate
system.
"""
from compas.geometry import Frame
from compas.geometry import Transformation
from compas.geometry import Box

# Box in the world coordinate system
frame = Frame([1, 0, 0], [-0.45, 0.1, 0.3], [1, 0, 0])
width, length, height = 1, 1, 1
box = Box(frame, width, length, height)

# Frame F representing a coordinate system
F = Frame([2, 2, 2], [0.978, 0.010, -0.210], [0.090, 0.882, 0.463])

# Get transformation between frames and apply transformation on box.
T = Transformation.from_frame_to_frame(Frame.worldXY(), F)
box_transformed = box.transformed(T)
print("Box frame transformed", box_transformed.frame)
```

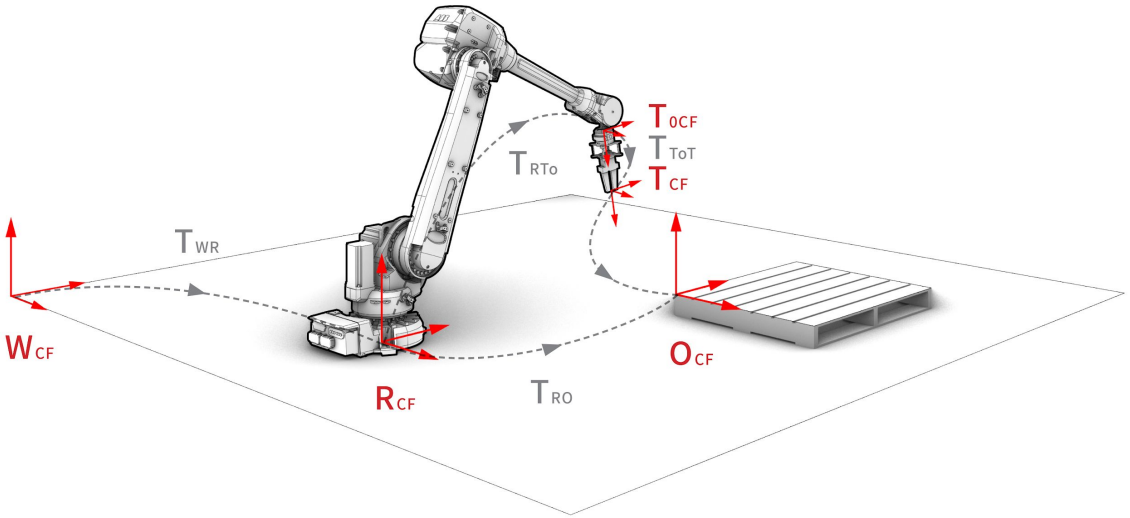

Rotation and orientation

A `Rotation` is a circular movement of an object around a point of rotation. A three-dimensional object can always be rotated around an infinite number of imaginary lines called *rotation axes*.

```python
"""There are several ways to construct a `Rotation`.
"""
import math
from compas.geometry import Frame
from compas.geometry import Rotation

R = Rotation.from_axis_and_angle([1, 0, 0], math.radians(30))
R = Rotation.from_axis_and_angle([1, 0, 0], math.radians(30), point=[1, 0, 0])
R = Rotation.from_basis_vectors([0.68, 0.68, 0.27], [-0.67, 0.73, -0.15])
R = Rotation.from_frame(Frame([1, 1, 1], [0.68, 0.68, 0.27], [-0.67, 0.73, -0.15]))
R = Rotation.from_axis_angle_vector([-0.043, -0.254, 0.617])
R = Rotation.from_quaternion([0.945, -0.021, -0.125, 0.303])
R = Rotation.from_euler_angles([1.4, 0.5, 2.3], static=True, axes='xyz')

print(R)
```

```
"""Example: Different Robot vendors use different conventions to describe TCP
orientation."""

from compas.geometry import Point
from compas.geometry import Vector
from compas.geometry import Frame

point = Point(0.0, 0.0, 63.0)
xaxis = Vector(0.68, 0.68, 0.27)
yaxis = Vector(-0.67, 0.73, -0.15)
F = Frame(point, xaxis, yaxis)

print(F.quaternion)  # ABB
print(F.euler_angles(static=False, axes='xyz'))  # Staubli
print(F.euler_angles(static=False, axes='zyx'))  # KUKA
print(F.axis_angle_vector)  # UR
```

Robot coordinate frames



World (WCF)

Robot (RCF)

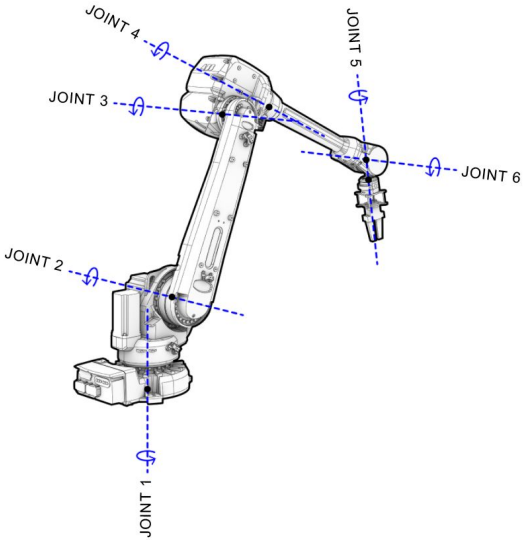
Tool0 (T0CF)

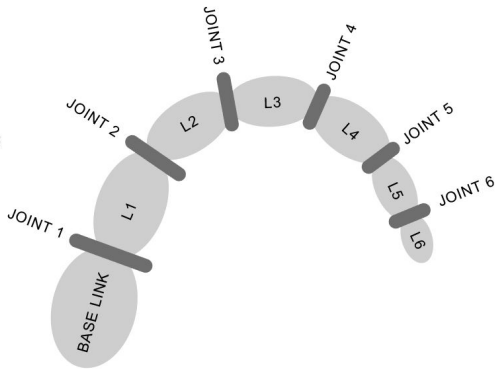
Tool (TCF)

Object (OCF)

Robot **models**

Robotic fabrication: models





- Links connected by Joints
- Coordinate frames at joints
- Tree structure
- URDF
  - Kinematics & Dynamics
  - Visual representation
  - Collision representation
  - Semantics

Robotic fabrication: models

compas.robots

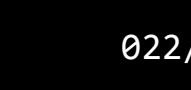
- Robot
  - Link
    - Inertial
    - Visual
    - Collision
  - Joint
    - Origin
    - Parent
    - Child
    - Dynamics

```python
# Load robot model
from compas.robots import RobotModel

RobotModel.from_urdf_file('ur5.urdf')
```

```xml
<?xml version="1.0" ?>
<robot name="rfl" xmlns:xacro="http://wiki.ros.org/xacro">
  <link name="rail">
   <inertial>
     <origin xyz="0.0 0.0 0.0"/>
     <mass value="120"/>
     <inertia ixx="-4.98" ixy="-0.0" ixz="2.7" iyy="-5.95" iyz="-0.0" izz="0.828"/>
   </inertial>
   <visual>
     <geometry>
       <mesh filename="package://rfl/meshes/visual/rail.stl"/>
     </geometry>
   </visual>
   <collision>
     <geometry>
       <mesh filename="package://rfl/meshes/collision/rail.stl"/>
     </geometry>
   </collision>
 </link>
<joint name="gantry_joint" type="prismatic">
   <parent link="rail"/>
   <child link="gantry"/>
   <origin rpy="0 0 0 " xyz="0 0 0"/>
   <axis xyz="1 0 0"/>
   <limit effort="100000" lower="0.0" upper="37.206" velocity="2.618"/>
   <dynamics damping="0.2" friction="0"/>
 </joint>
...
```

```python
# Rhino
from compas.robots import *
from compas_fab.rhino import RobotArtist

r = 'ros-industrial/abb'
p = 'abb_irb6600_support'
b = 'kinetic-devel'

github = GithubPackageMeshLoader(r,p,b)
urdf = github.load_urdf('irb6640.urdf')

model = RobotModel.from_urdf_file(urdf)
model.load_geometry(github)

RobotArtist(model).draw_visual()
```

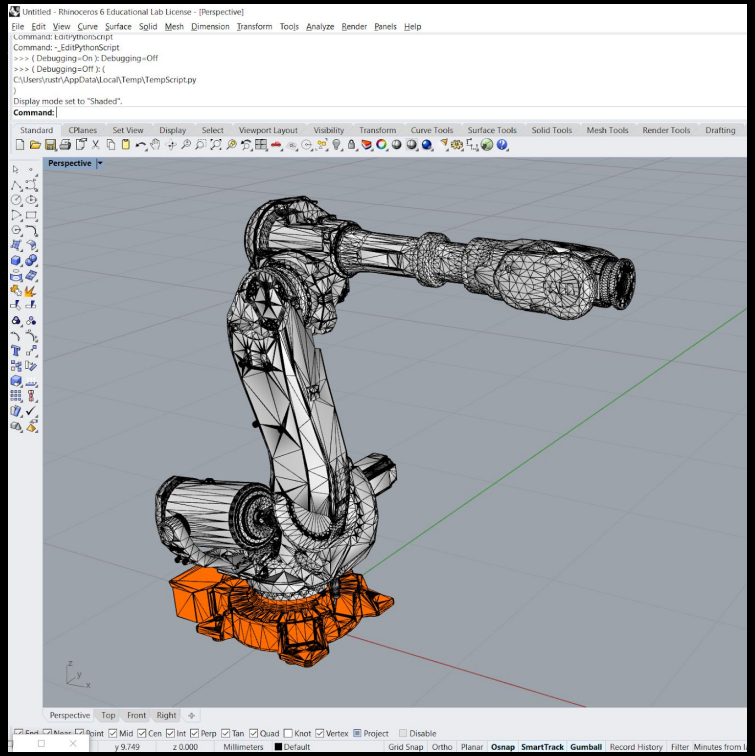
```python
# Blender
from compas.robots import *
from compas_fab.blender import RobotArtist

r = 'ros-industrial/abb'
p = 'abb_irb6600_support'
b = 'kinetic-devel'

github = GithubPackageMeshLoader(r,p,b)
urdf = github.load_urdf('irb6640.urdf')

model = RobotModel.from_urdf_file(urdf)
model.load_geometry(github)

RobotArtist(model).draw_visual()
```



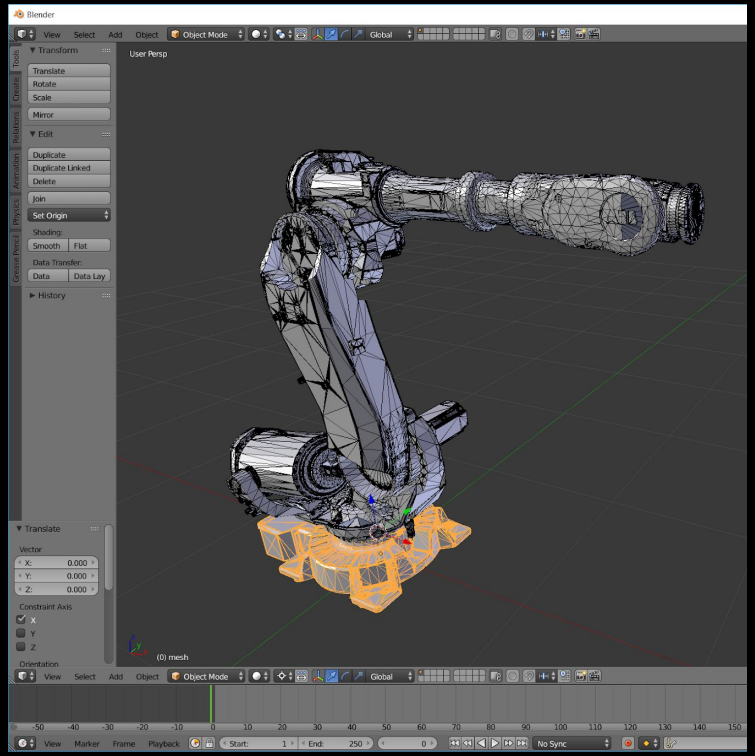
```python
# Rhino
from compas.robots import *
from compas_fab.rhino import RobotArtist

r = 'ros-industrial/abb'
p = 'abb_irb6600_support'
b = 'kinetic-devel'

github = GithubPackageMeshLoader(r,p,b)
urdf = github.load_urdf('irb6640.urdf')

robot = Robot.from_urdf_file(urdf)
robot.load_geometry(github)

RobotArtist(robot).draw_visual()
```

```python
# Blender
from compas.robots import *
from compas_fab.blender import RobotArtist

r = 'ros-industrial/abb'
p = 'abb_irb6600_support'
b = 'kinetic-devel'

github = GithubPackageMeshLoader(r,p,b)
urdf = github.load_urdf('irb6640.urdf')

robot = Robot.from_urdf_file(urdf)
robot.load_geometry(github)

RobotArtist(robot).draw_visual()
```

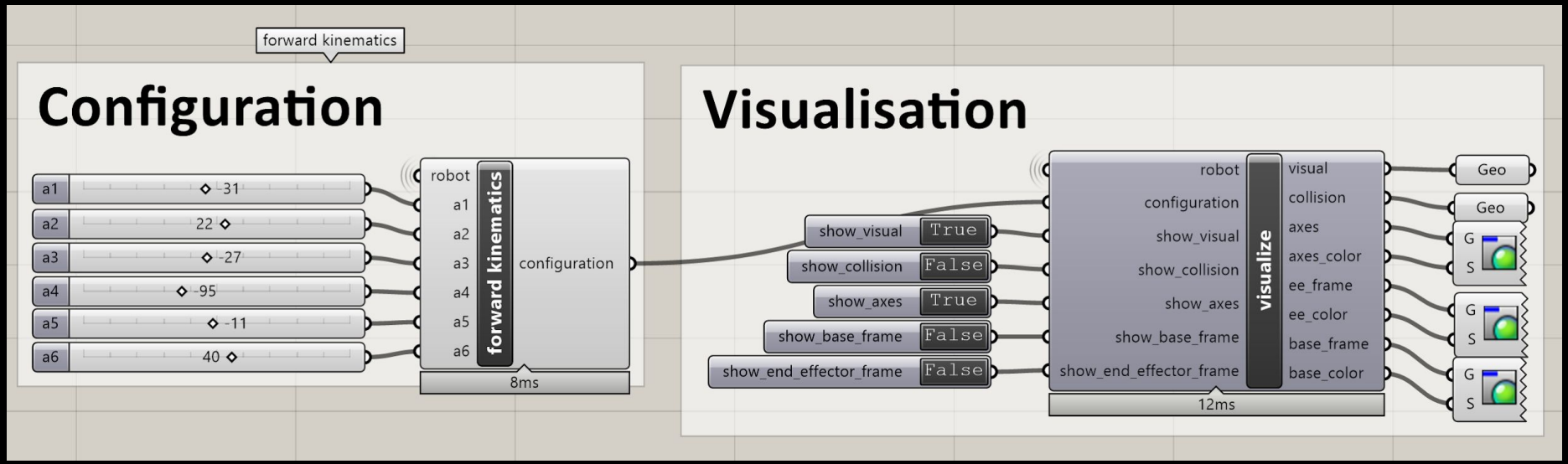
Rhino



Blender



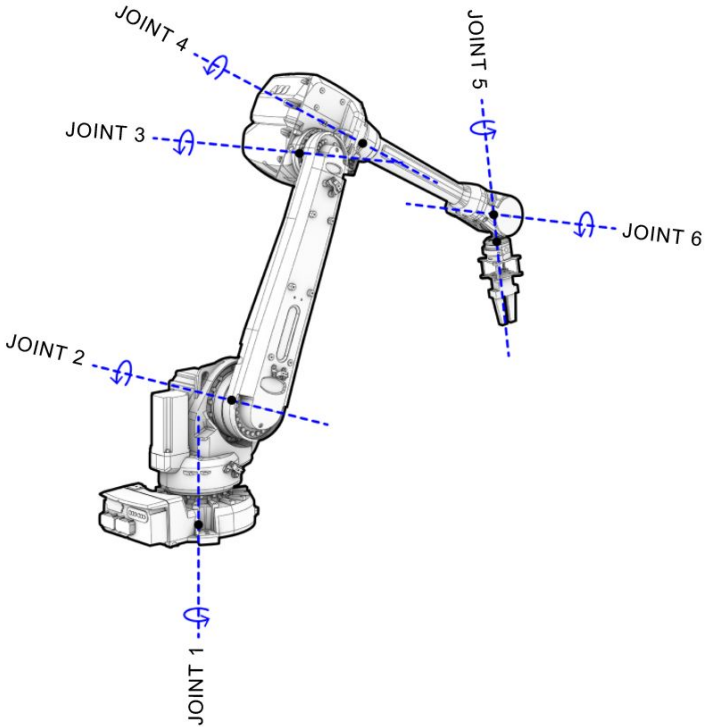
Grasshopper



Loading external models

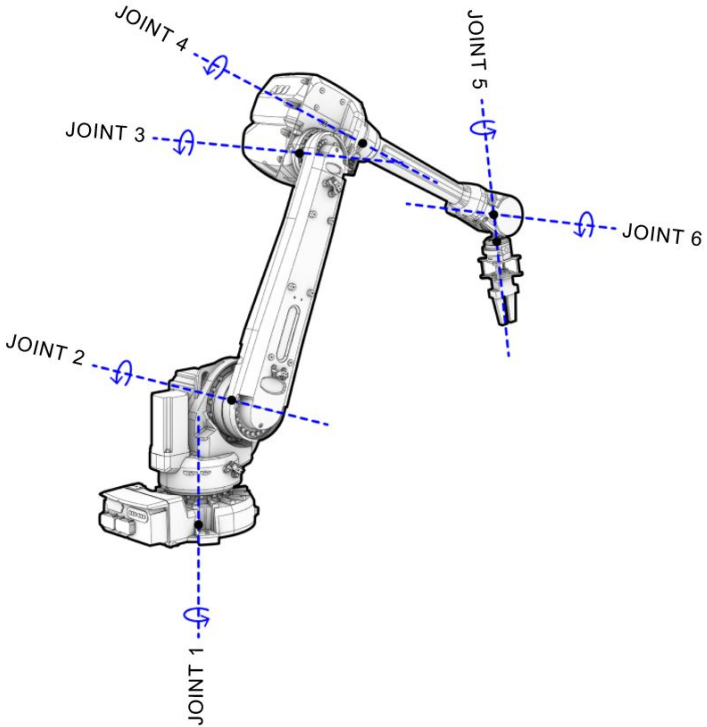
```
# Set high precision to import meshes defined in meters
compas.PRECISION = '12f'

# Load robot and its geometry
with RosClient('localhost') as ros:
    robot = ros.load_robot(load_geometry=True)
    print(robot.model)
```

Configuration




```python
from compas.robots import Joint

print(Joint.REVOLUTE)
print(Joint.PRISMATIC)
print(Joint.FIXED)
```

Configuration




```
from math import pi
from compas_fab.robots import Configuration

values = [1.5, pi]
types = [Joint.PRISMATIC, Joint.REVOLUTE]
config = Configuration(values, types)

config =
Configuration.from_revolute_values([pi/2, 0., 0.,
pi/2, pi, 0])

config =
Configuration.from_prismatic_and_revolute_values(
[8.312], [pi/2, 0., 0., 0., 2*pi, 0.8])
```

Building your own robot

```
# create robot
robot = RobotModel("robot", links=[], joints=[])

# add links
link0 = robot.add_link("world")
link1 = robot.add_link("link1", visual_mesh=mesh1,)
link2 = robot.add_link("link2", visual_mesh=mesh2,)

# add the joints between the links
robot.add_joint("joint1", Joint.REVOLUTE, link0, link1, origin1, axis1)
robot.add_joint("joint2", Joint.REVOLUTE, link1, link2, origin2, axis2)
```

Backends

Robotic fabrication: backends

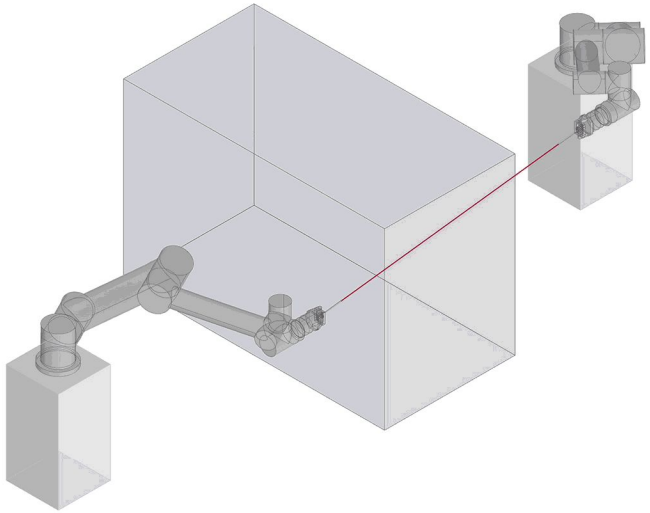
V-REP
Robot simulator

ROS
Robot Operating System

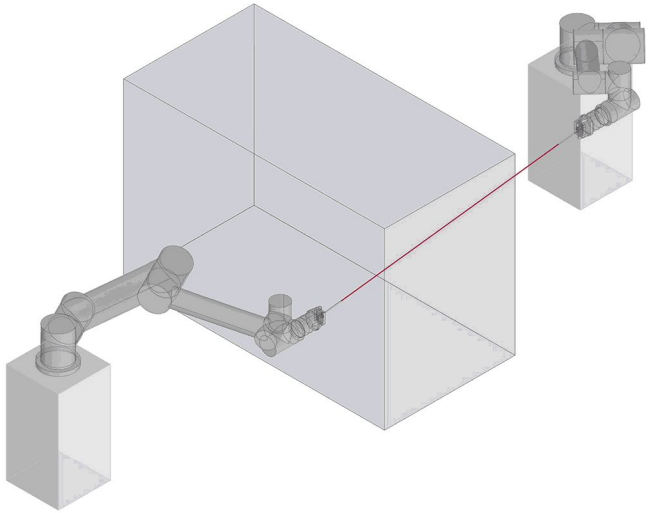
Moveit!
Motion Planning Framework

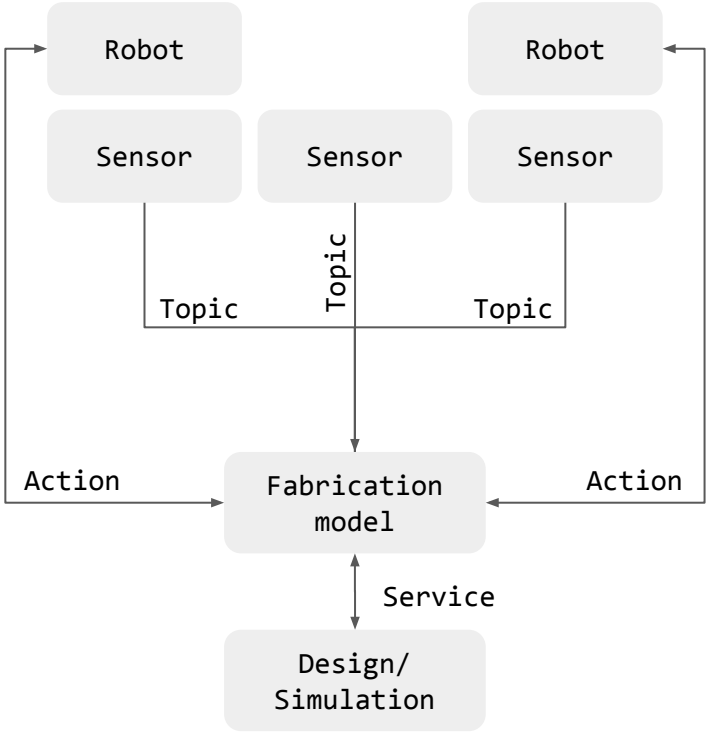

ROS
Open Source Robotics Foundation

Fabrication processes

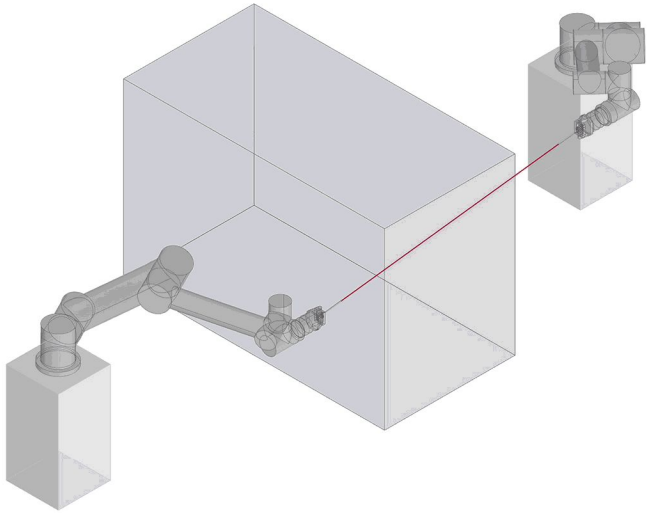


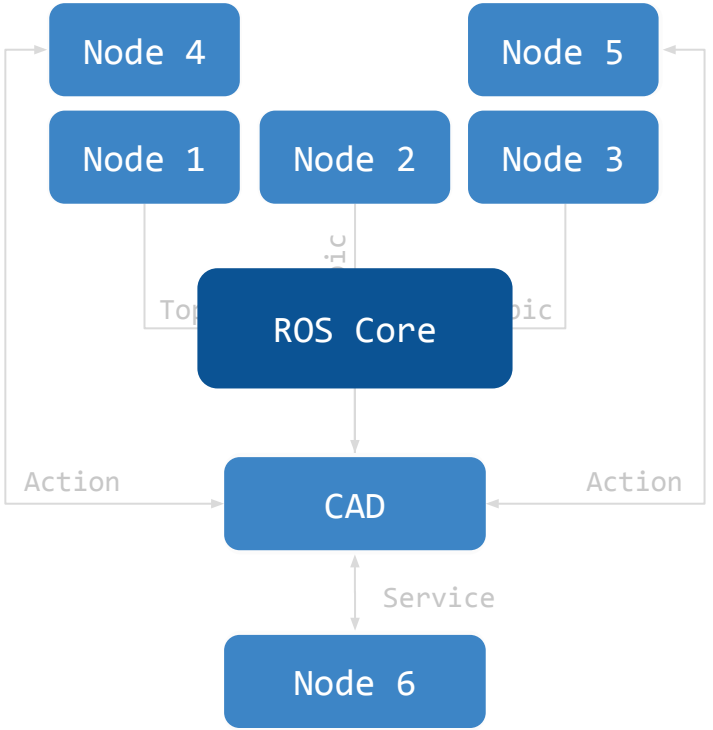
Fabrication processes



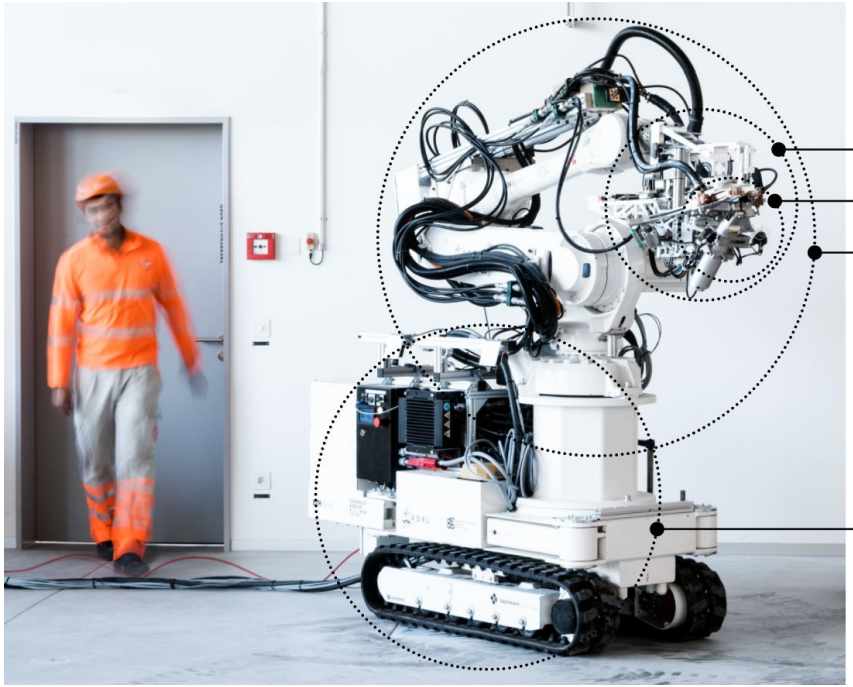


Fabrication processes





What is ROS?



ROS Node 1: End Effector

ROS Node 2: Localization

ROS Node 3: Robot Arm

ROS Node 4: Mobile Base

What is ROS?

`plumbing`

- Process management
- Interprocess communication
- Device drivers

`tools`

- Simulation
- Visualization
- Graphical UI
- Data logging

`capabilities`

- Control
- Planning
- Perception
- Mapping
- Manipulation

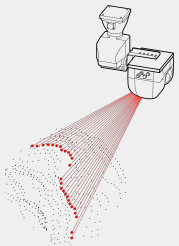
`ecosystem`

- Package organization
- Software distribution
- Community
- Documentation
- Tutorials

ROS Concepts

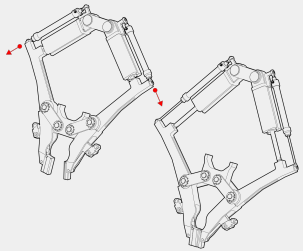
Topics

- Nodes communicate over topics
- **Publish/subscribe** model
- **One-way** data stream
- e.g: *robot states, sensor data*



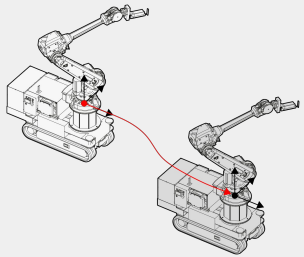
Services

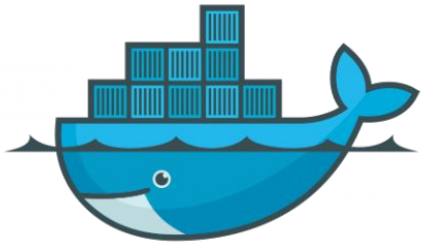
- **Blocking** call per request
- Request/response model
- Short trigger or calculation
- e.g: *calculate path, open gripper*

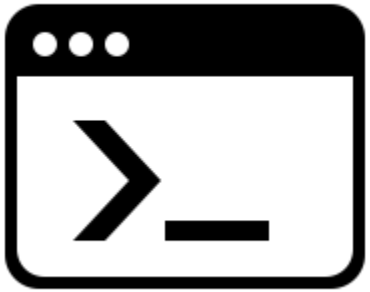


Actions

- **Non-blocking** requests
- Goal-oriented and cancellable
- Implemented with topics
- e.g: *navigation, motion execution*




docker



`docker-compose up -d`

Right-click → Compose Up

docker/docker-compose.yml

http://localhost:8080/vnc.html?resize=scale&autoconnect=true

http://192.168.99.100:8080/vnc.html?resize=scale&autoconnect=true

Verify connection to ROS

```python
from compas_fab.backends import RosClient

with RosClient('localhost') as client:
    print('Connected:', client.is_connected)
```

Verify connection to ROS

```
from compas_fab.backends import RosClient

with RosClient('192.168.99.100') as client:
    print('Connected:', client.is_connected)
```

Publish to topics

Publish to topic: **connect**

```
import time

from roslibpy import Topic
from compas_fab.backends import RosClient

with RosClient('localhost') as client:
    talker = Topic(client, '/messages', 'std_msgs/String')
    talker.advertise()

    while client.is_connected:
        print('Sending...')
        talker.publish({'data': 'Hello'})
        time.sleep(1)
```

Publish to topic: **advertise**

```
import time

from roslibpy import Topic
from compas_fab.backends import RosClient

with RosClient('localhost') as client:
    talker = Topic(client, '/messages', 'std_msgs/String')
    talker.advertise()

    while client.is_connected:
        print('Sending...')
        talker.publish({'data': 'Hello'})
        time.sleep(1)
```

Publish to topic: **publish**

```
import time

from roslibpy import Topic
from compas_fab.backends import RosClient

with RosClient('localhost') as client:
    talker = Topic(client, '/messages', 'std_msgs/String')
    talker.advertise()

    while client.is_connected:
        print('Sending...')
        talker.publish({'data': 'Hello'})
        time.sleep(1)
```

Subscribe to topics

Subscribe to topic: **connect**

```
import time

from roslibpy import Topic
from compas_fab.backends import RosClient

def receive_message(message):
    print('Heard talking: ' + message['data'])

with RosClient('localhost') as client:
    listener = Topic(client, '/messages', 'std_msgs/String')
    listener.subscribe(receive_message)

    while client.is_connected:
        time.sleep(1)
```

Subscribe to topic: **subscribe**

```
import time

from roslibpy import Topic
from compas_fab.backends import RosClient

def receive_message(message):
    print('Heard talking: ' + message['data'])

with RosClient('localhost') as client:
    listener = Topic(client, '/messages', 'std_msgs/String')
    listener.subscribe(receive_message)

    while client.is_connected:
        time.sleep(1)
```

Robot **planning**

`compas.robots.RobotModel`

Describes the kinematics, linkage geometry and dynamics of a robot cell.

`compas_fab.robots.Robot`

Integrates a robot model, additional semantic information for planning, a backend client and artist instance.

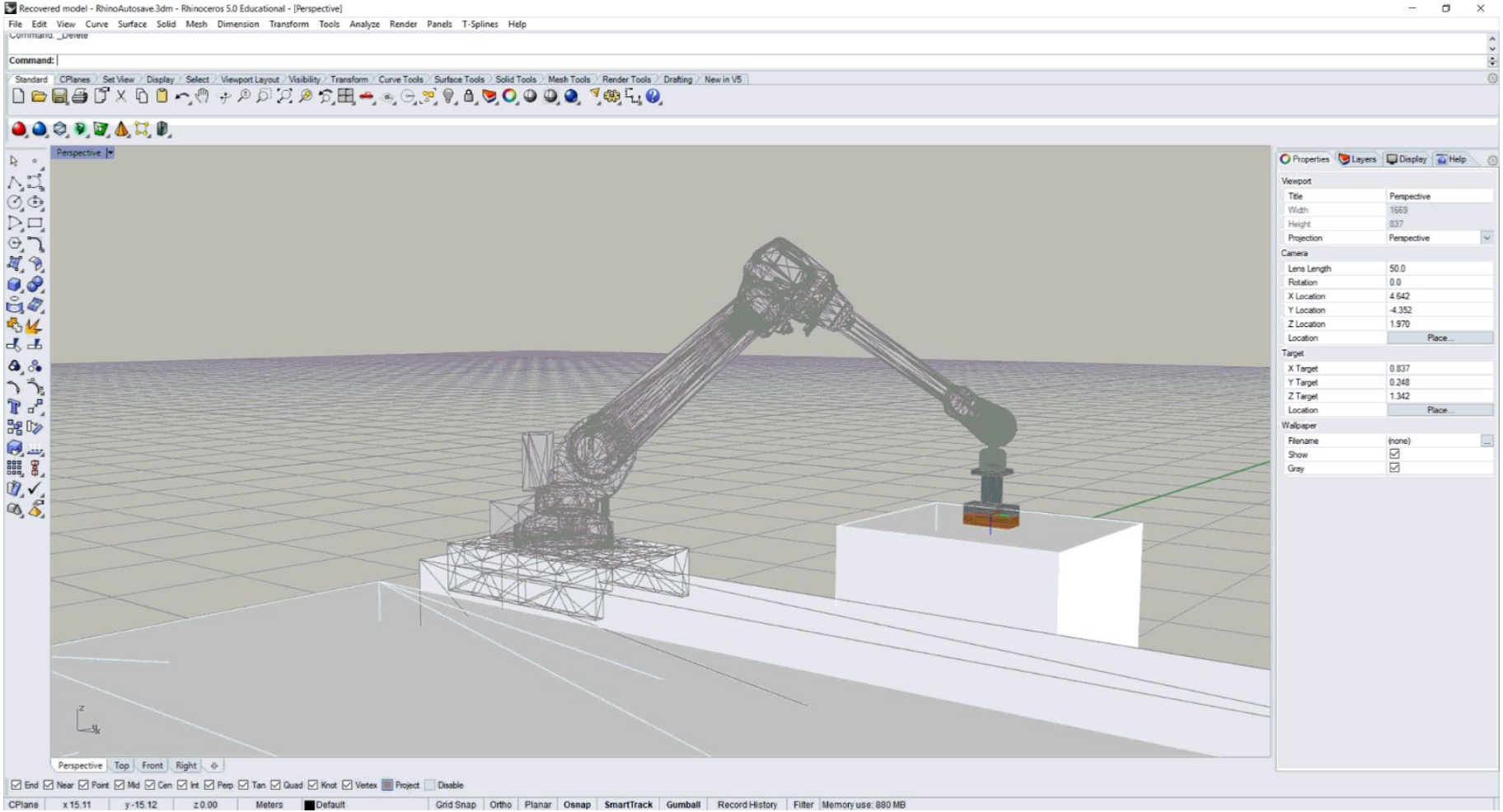
Planning

MoveIt!

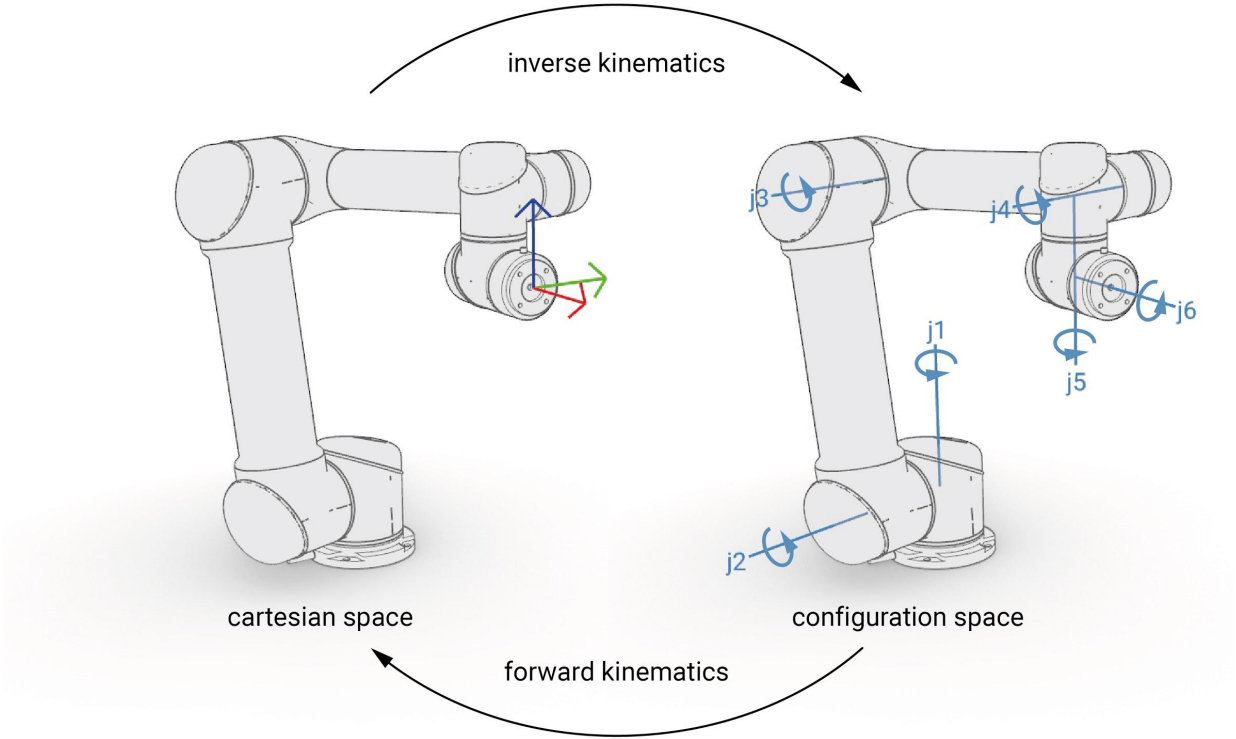
Kinematics

Motion planning

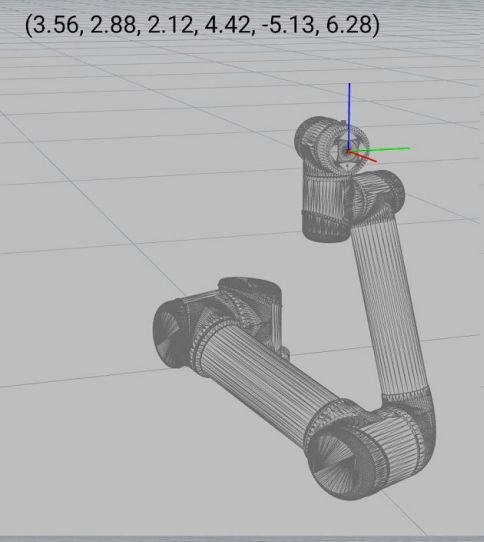
Planning scene


Recovered model - RhinoAutosave.3dm - Rhinoceros 5.0 Educational - [Perspective]
File Edit View Curve Surface Solid Mesh Dimension Transform Tools Analyze Render Panels T-Splines Help
Command:
Standard CPlanes Set View Display Select Viewport Layout Visibility Transform Curve Tools Surface Tools Solid Tools Mesh Tools Render Tools Drafting New in V5
Perspective
Properties Layers Display Help
Viewport
Title Perspective
Width 1669
Height 837
Projection Perspective
Camera
Lens Length 50.0
Rotation 0.0
X Location 4.642
Y Location -4.352
Z Location 1.970
Location Place...
Target
X Target 0.837
Y Target 0.248
Z Target 1.342
Location Place...
Wallpaper
Filename (none)
Show
Gray
Perspective Top Front Right
End Near Point Mid Cen Int Perp Tan Quad Knot Vertex Project Disable
CPlane x 15.11 y -15.12 z 0.00 Meters Default Grid Snap Ortho Planar Osnap SmartTrack Gumball Record History Filter Memory use: 880 MB

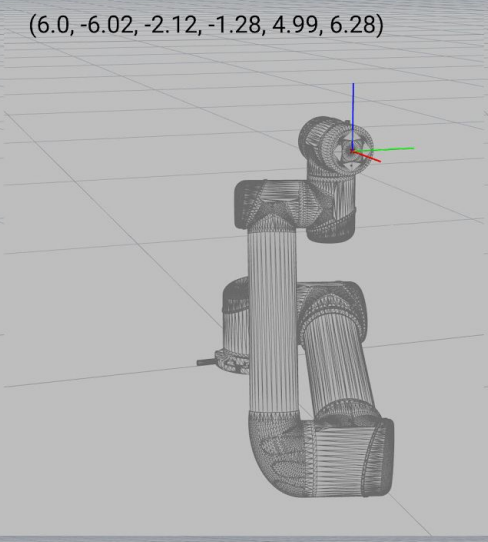
Kinematics


inverse kinematics
j3
j4
j6
j1
j5
j2
cartesian space
configuration space
forward kinematics

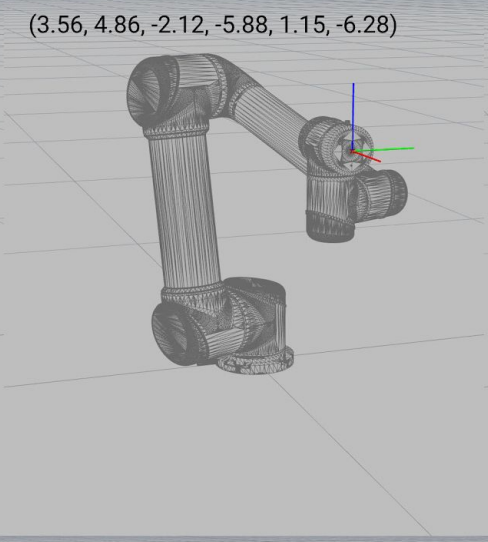
(3.56, 2.88, 2.12, 4.42, -5.13, 6.28)



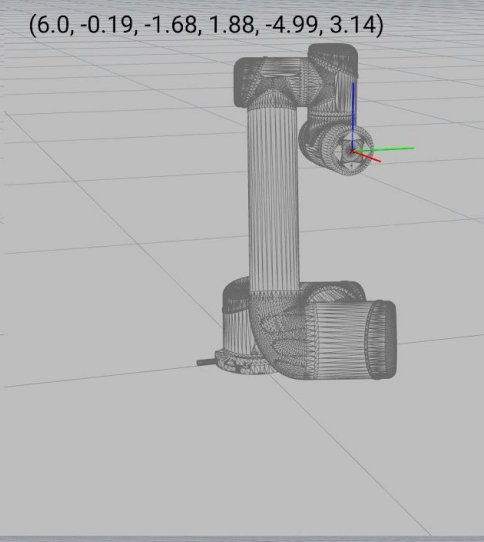
(6.0, -6.02, -2.12, -1.28, 4.99, 6.28)



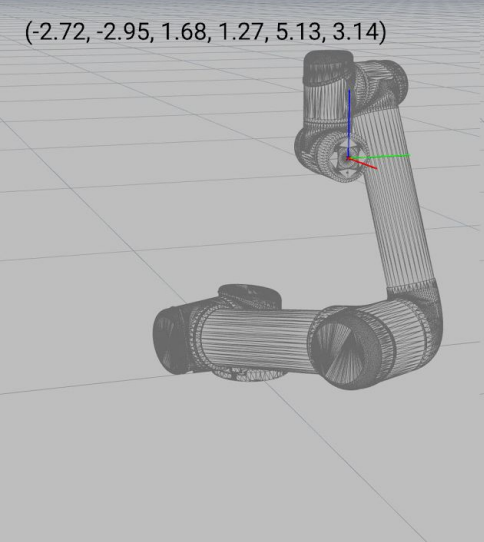
(3.56, 4.86, -2.12, -5.88, 1.15, -6.28)



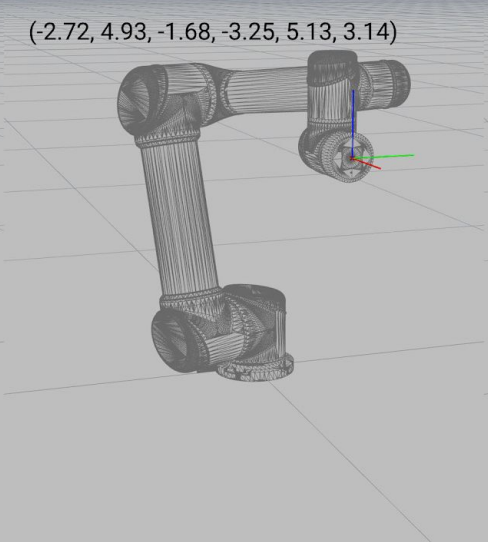
(6.0, -0.19, -1.68, 1.88, -4.99, 3.14)



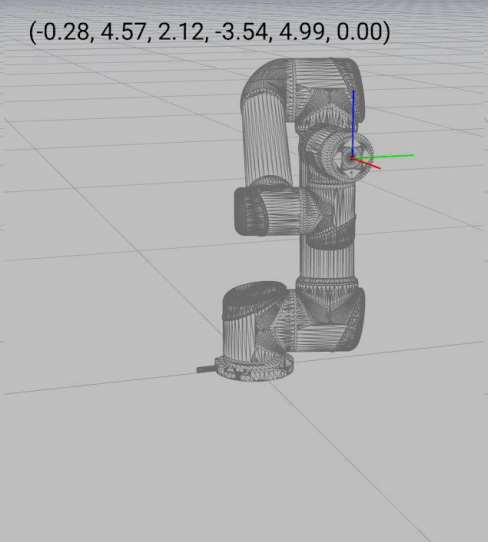
(-2.72, -2.95, 1.68, 1.27, 5.13, 3.14)



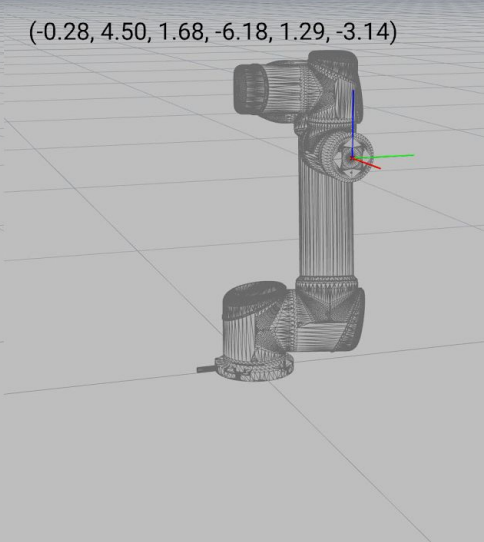
(-2.72, 4.93, -1.68, -3.25, 5.13, 3.14)



(-0.28, 4.57, 2.12, -3.54, 4.99, 0.00)



(-0.28, 4.50, 1.68, -6.18, 1.29, -3.14)



Forward kinematics

For a given configuration, calculate the frame of the end-effector expressed in the world coordinate frame.

- **Input:** Configuration, i.e. `compas_fab.robots.Configuration()`
- **Output:** Frame in WCF, i.e. `compas.geometry.Frame()`
- **Usage:**

```
config = Configuration.from_revolute_values([0, 0, 0, 3.14, 0, 0])
frame_wcf = robot.forward_kinematics(config)
```

Inverse kinematics

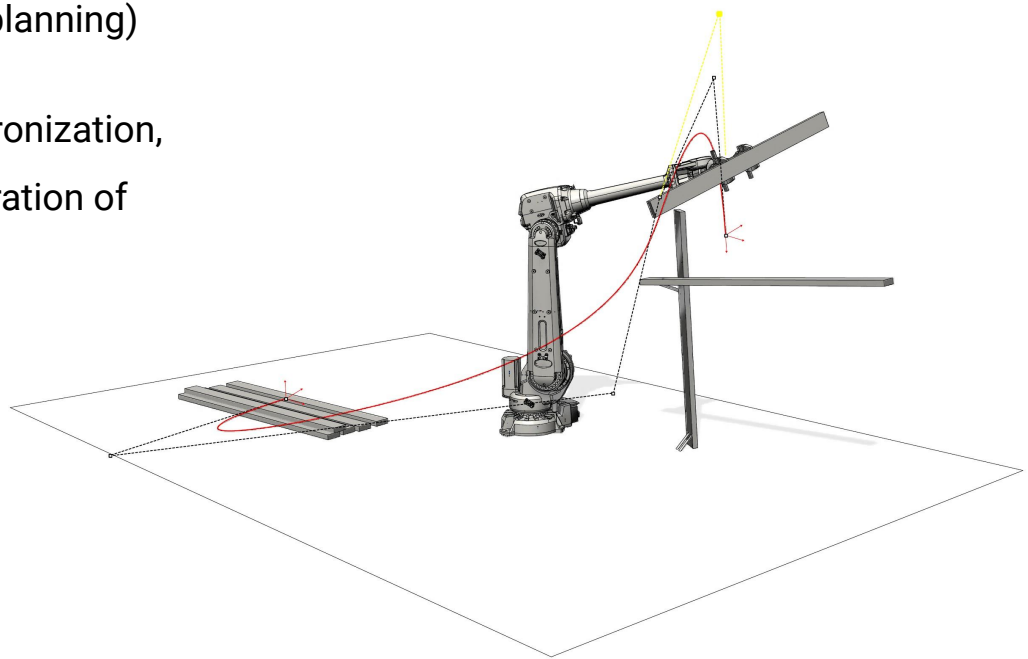
For a given end-effector frame expressed in world coordinate frame, find the corresponding robot configuration(s) to reach it.

- **Input:** Frame in WCF and start config, i.e. `compas.geometry.Frame()`
- **Output:** Configuration, i.e. `compas_fab.robots.Configuration()`
- **Usage:**

```
frame_wcf = Frame([0.3, 0.1, 0.5], [1, 0, 0], [0, 1, 0])
start_config = robot.init_configuration()
config = robot.inverse_kinematics(frame_wcf, start_config)
```

Motion planning

- Collision checking (= path planning)
- Trajectory checking (synchronization, consider speed and acceleration of moving objects)



Motion planning

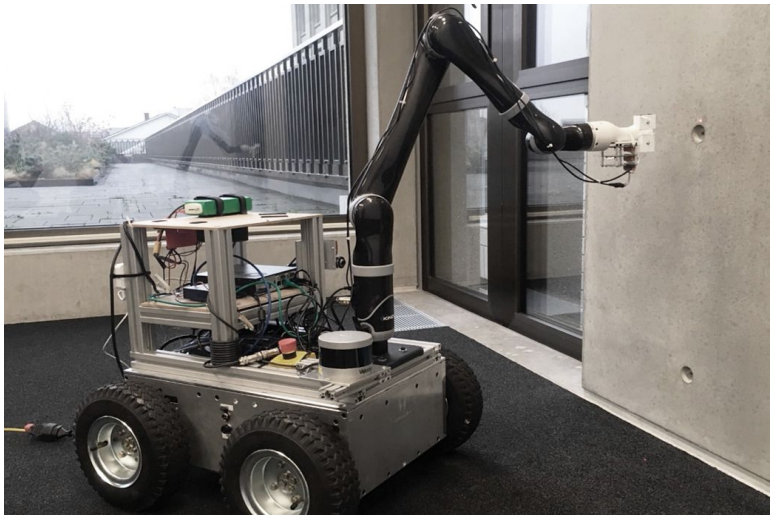
Collision checks

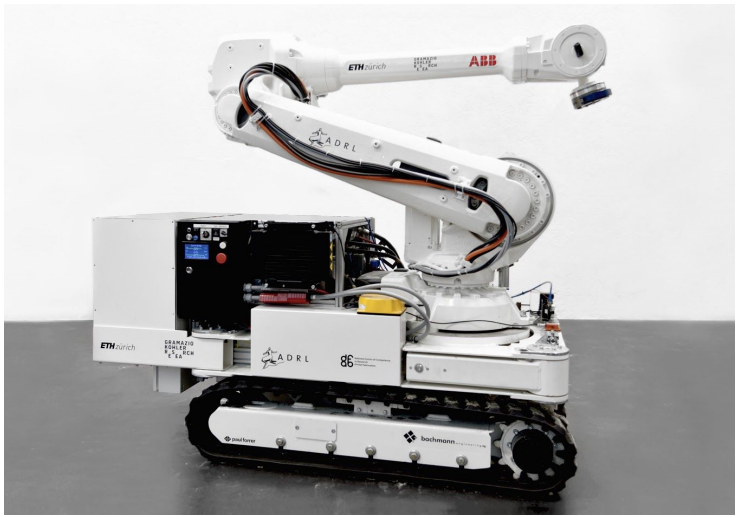
- Intricate positions (spatial assembly)
- Multiple robots working closely

Motion planning

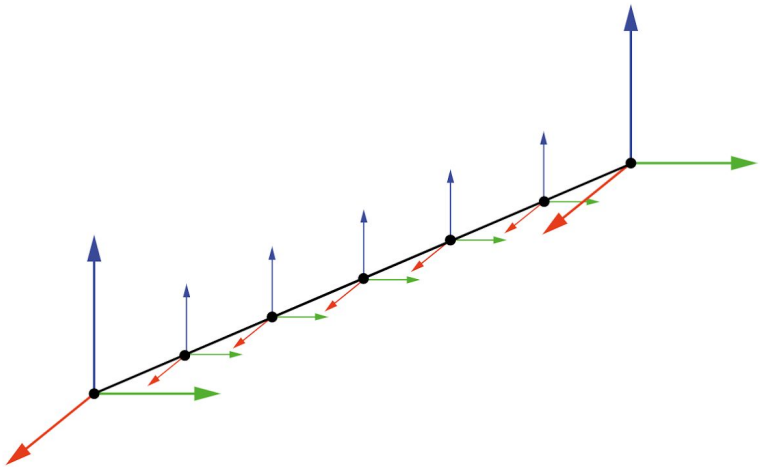
Trajectory checks

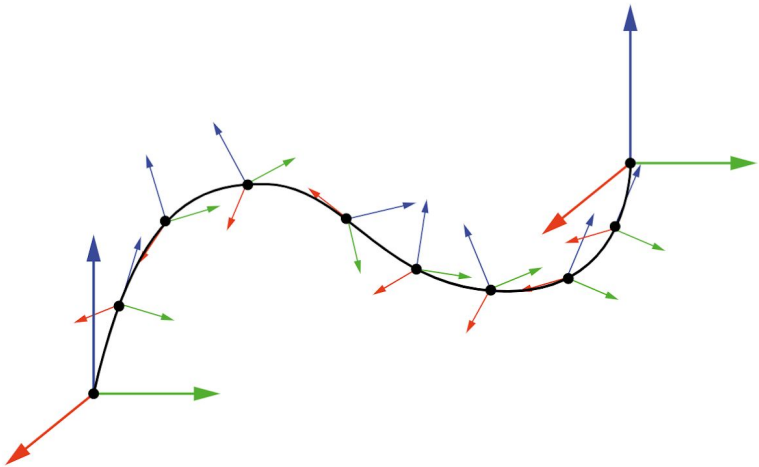
- Synchronisation
- Continuous processes





Cartesian motion vs free-space motion





Plan cartesian motion

For a given list of waypoints, calculate a linear trajectory in task space.

- **Input:** List of frame in WCF and start config, i.e. `compas.geometry.Frame()`
- **Output:** Joint trajectory, i.e. `compas_fab.robots.JointTrajectory()`
- **Usage:**

```
f1 = Frame([0.3, 0.1, 0.5], [1, 0, 0], [0, 1, 0])
f2 = Frame([0.6, 0.1, 0.4], [1, 0, 0], [0, 1, 0])
start_config = robot.zero_configuration()

trajectory = robot.plan_cartesian_motion([f1, f2], start_config)
```

Plan kinematic motion

For a given start configuration and a goal (defined as constraints), calculate a joint trajectory in joint space.

- **Input:** Goal constraints and start cfg, e.g. `compas_fab.robots.PositionConstraint()`
- **Output:** Joint trajectory, i.e. `compas_fab.robots.JointTrajectory()`
- **Usage:**

```
pc = robot.constraints_from_frame(...)
start_config = robot.zero_configuration()

trajectory = robot.plan_motion(pc, start_config, planner_id='RRT')
```

Defining constraints

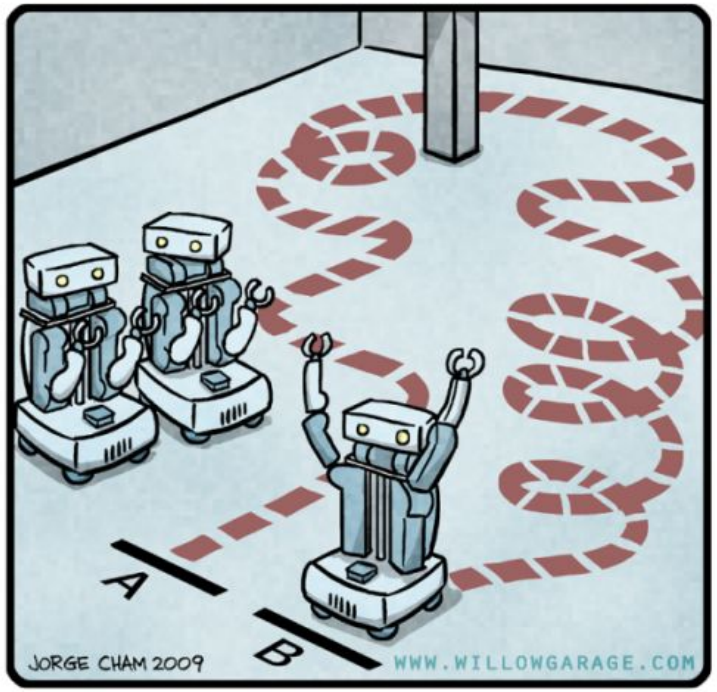
- **`JointConstraint`**
  Constrains the value of a joint to be within a certain bound.

- **`OrientationConstraint`**
  Constrains a link to be within a certain orientation.

- **`PositionConstraint`**
  Constrains a link to be within a certain bounding volume.

```
frame = Frame([0.4, 0.3, 0.4], [0, 1, 0], [0, 0, 1])
tolerance_position = 0.001
tolerance_axes = [math.radians(1)] * 3

start_configuration = Configuration.from_revolute_values([-0.042, 4.295, 0, -3.327, 4.755, 0.])
group = robot.main_group_name

# create goal constraints from frame
goal_constraints = robot.constraints_from_frame(frame,
                                                 Tolerance_position,
                                                 Tolerance_axes,
                                                 group)
```

R.O.B.O.T. Comics



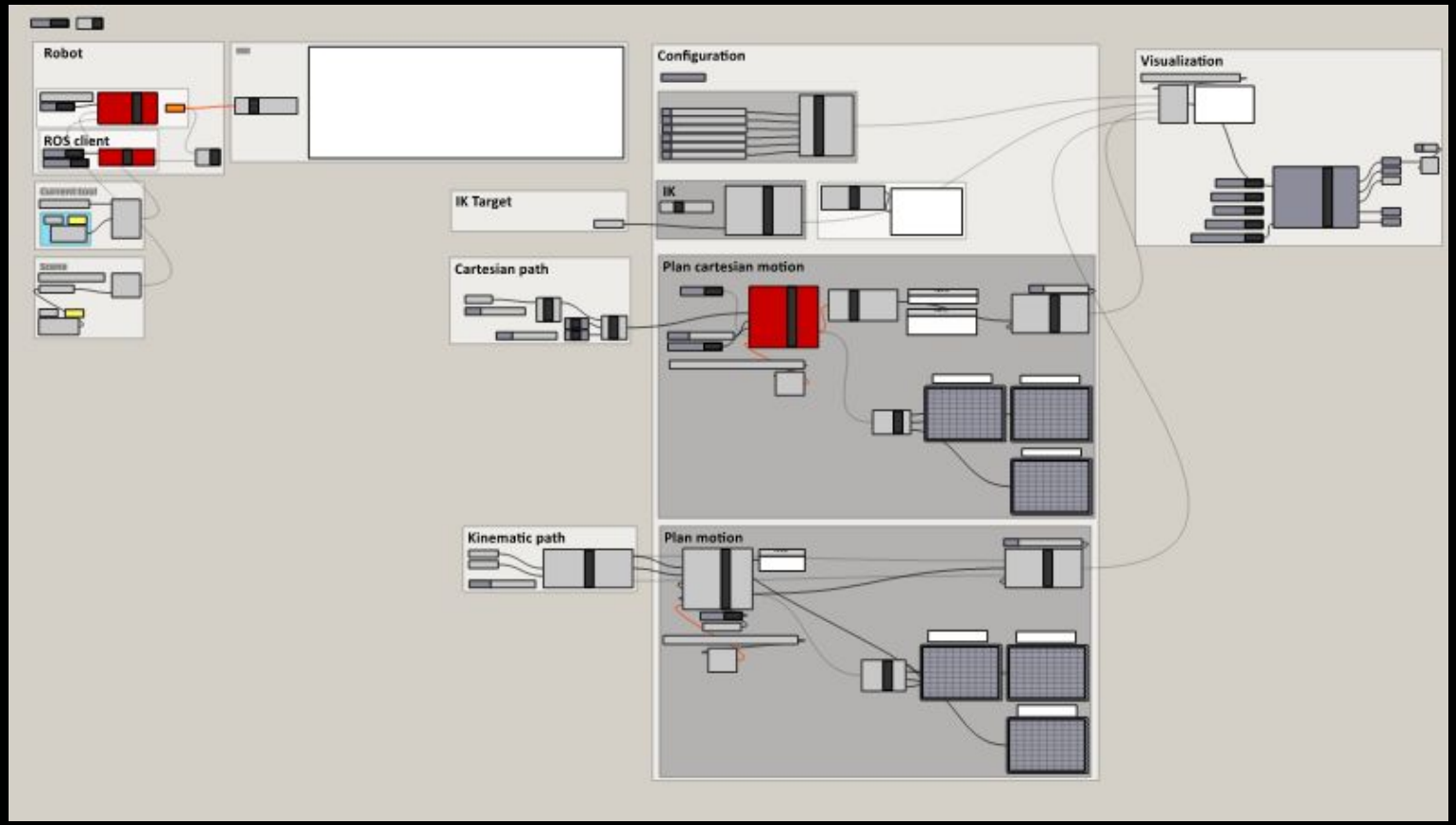
"HIS PATH-PLANNING MAY BE SUB-OPTIMAL, BUT IT'S GOT FLAIR."

Planning scene operations

Add/append/remove collision meshes (i.e. obstacles) and add/remove attached collision meshes (i.e. meshes attached to the end effector).

- **Usage:**

```
scene = compas_fab.robots.PlanningScene(robot)
scene.add_collision_mesh(..)
scene.remove_collision_mesh(..)
scene.append_collision_mesh(..)

scene.add_attached_collision_mesh(..)
scene.remove_attached_collision_mesh(..)
scene.attach_collision_mesh_to_robot_end_effector(..)
```


Robot
ROS client
Configuration
Visualization
IK Target
IK
Cartesian path
Plan cartesian motion
Kinematic path
Plan motion

Robot **control**

Control

MoveIt!

COMPAS RRC

Offline control

Online real-time control

Online non-real-time control

Trajectory execution

MoveIt!

High level interface, that hands over execution to underlying drivers.

Allows to execute a previously computed joint trajectory.

- **Input:** Joint trajectory, i.e. `compas_fab.robots.JointTrajectory()`
- **Output:** Future result, i.e. `compas_fab.backends.CancellableFutureResult()`
- **Usage:**

```
wait = robot.client.execute_joint_trajectory(trajectory)

wait.result()
```

Joint trajectory action

Similar to MoveIt! Executor, but lower level. It bypasses MoveIt! pipeline, but serves the same purpose and has almost identical interface.

- **Input:** Joint trajectory, i.e. `compas_fab.robots.JointTrajectory()`
- **Output:** Future result, i.e. `compas_fab.backends.CancellableFutureResult()`
- **Usage:**

```
wait = robot.client.follow_joint_trajectory(trajectory)

wait.result()
```

ROS-I Drivers

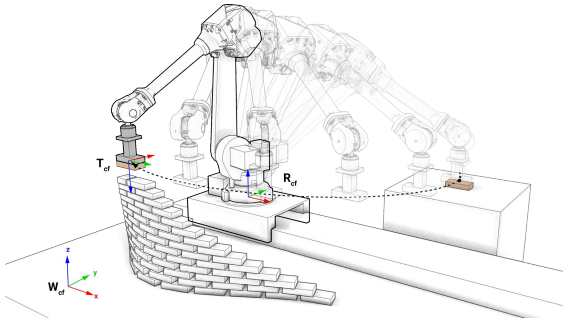
Different drivers implement different control strategies:

- UR: real-time control
- Franka Emika: real-time control
- ABB: non-realtime downloading driver
- Staubli: non-real-time streaming driver

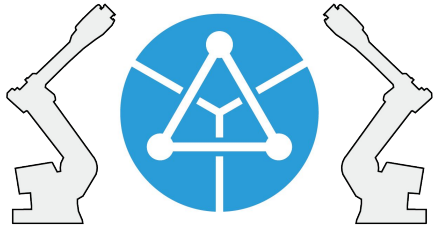
ROS-I Drivers

Some drivers' features are **not** accessible over a unified interface.

Use `RosClient` and `Topics` & `Services` to use them.



compas_fab



compas_rrc

FAB **0.12**

New backend extensibility
PyBullet integration
More backends

RRC **1.0**

Cyclic instruction support
New execution levels
More compact protocol

Thanks!