

# Hybrid Scripting/Vis Tool Development in Jupyter Notebooks

Leveraging human centric methods for performance analysis workflows

Scalable Tools - June 19, 2023

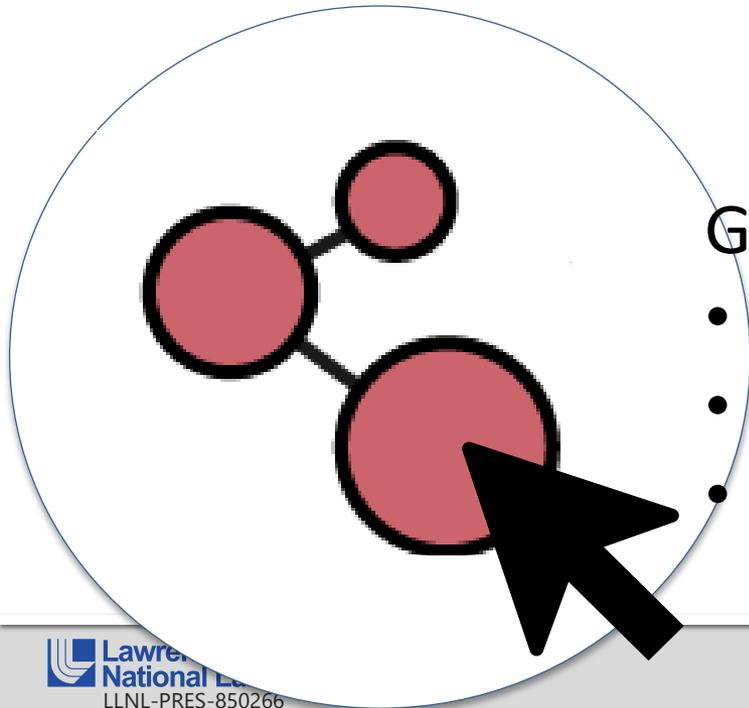
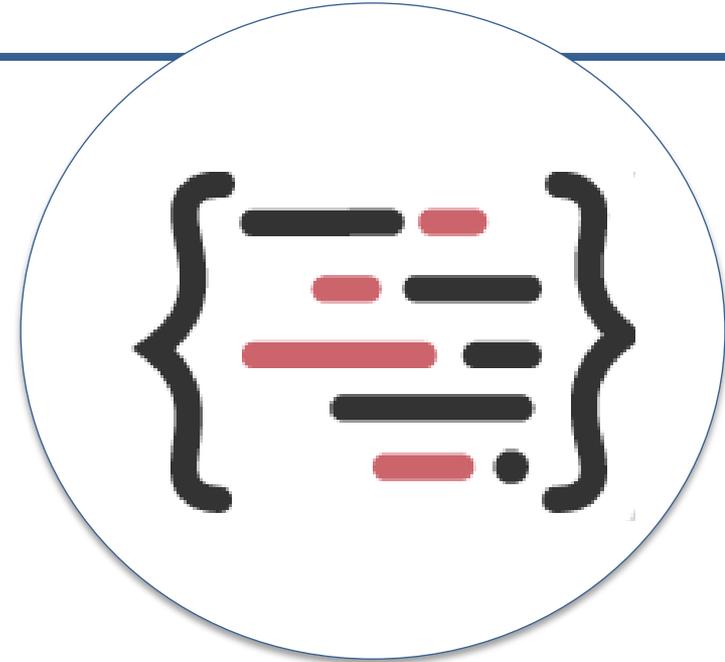
Connor Scully-Allison  
University of Utah/LLNL



# A Tale of Two Workflows

Script based tools for:

- Measuring Code/Generating Data
- Cleaning/Formatting Datasets
- Calculating Derived Metrics

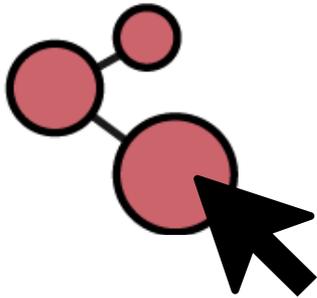
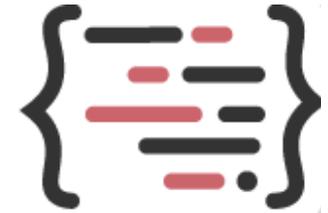


GUI-Based tools and visualizations for:

- Analyzing Metrics
- Communicating Work Done
- Identifying Bottlenecks

# Filling the Gap

Scripting can support a vast range of expressions and functionalities but can be cumbersome for analysis.



Interactive Visualization can support fluid exploration but is often limited to pre-determined tasks

**So how do we reconcile these two needs for performance analysts and tie these workflows together?**

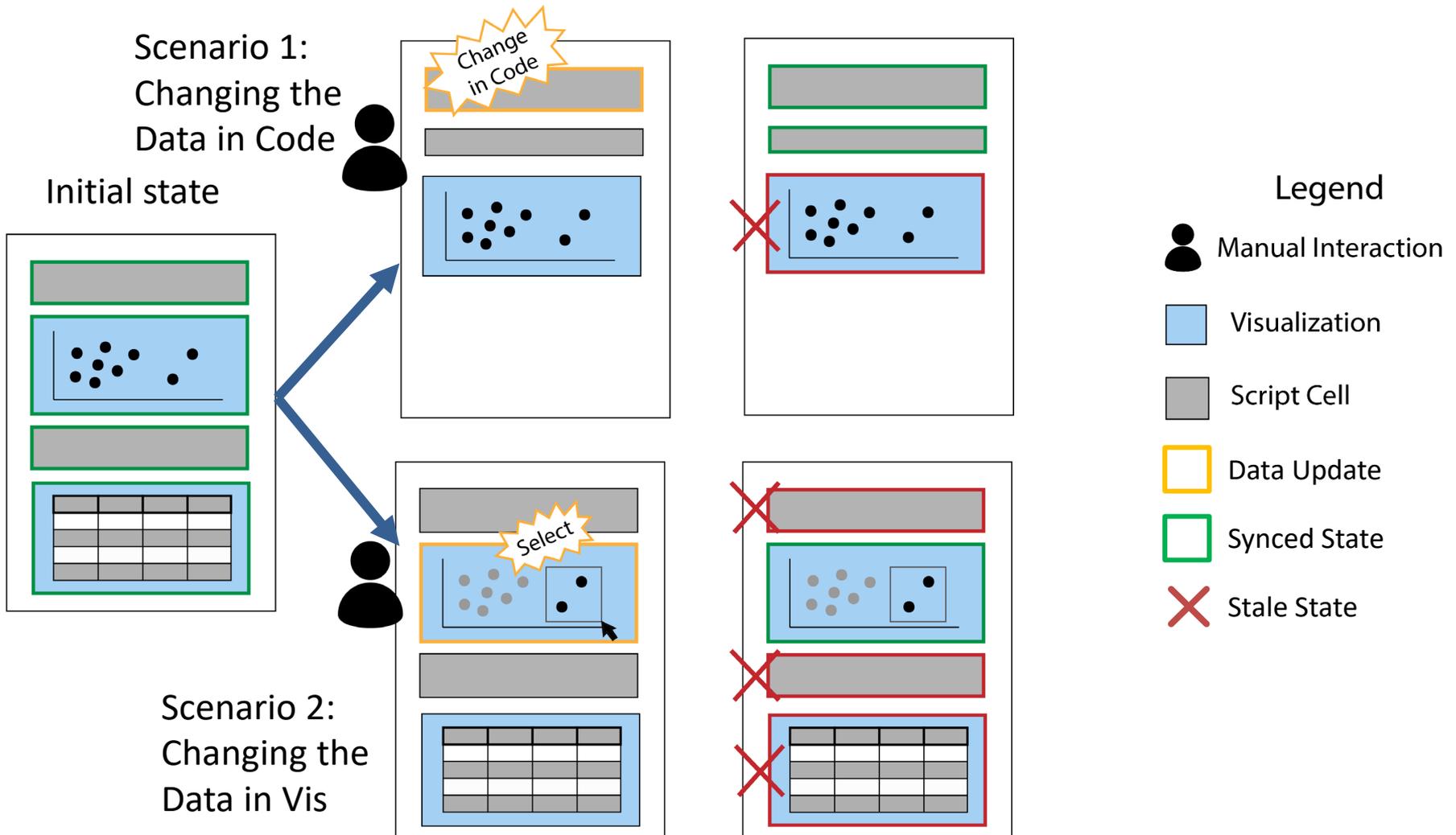


We develop tools embedded in Jupyter notebooks that leverage both **visualization** and **scripting** to give the users flexibility they seek.

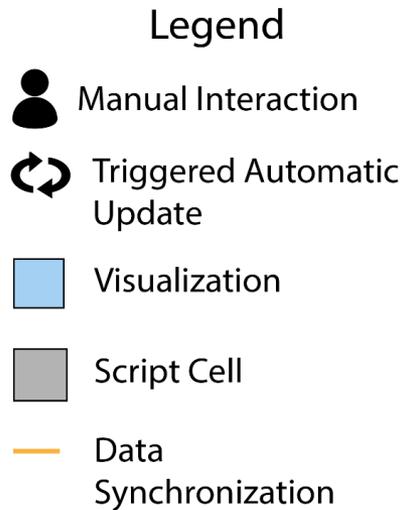
# Do Jupyter Notebooks alone fill this gap?



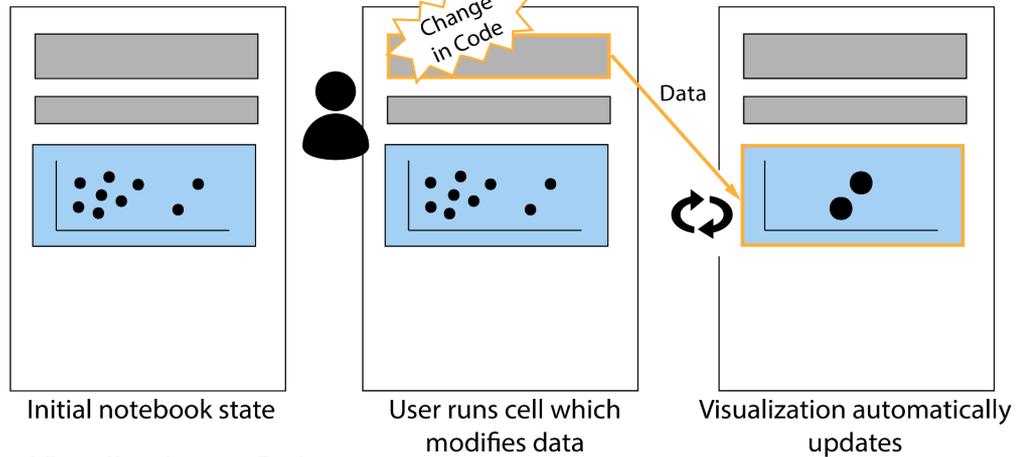
# Where Notebooks Fail to Support Hybrid Workflows



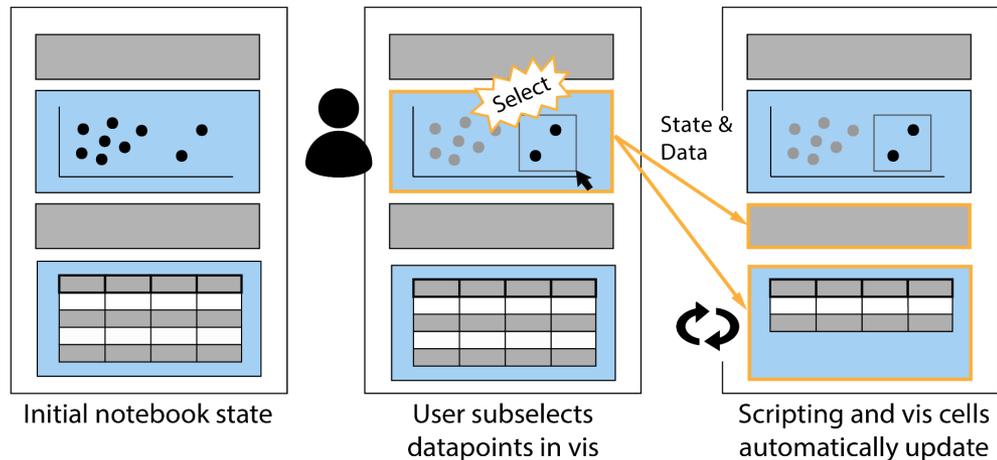
# What is the better model for a hybrid workflow?



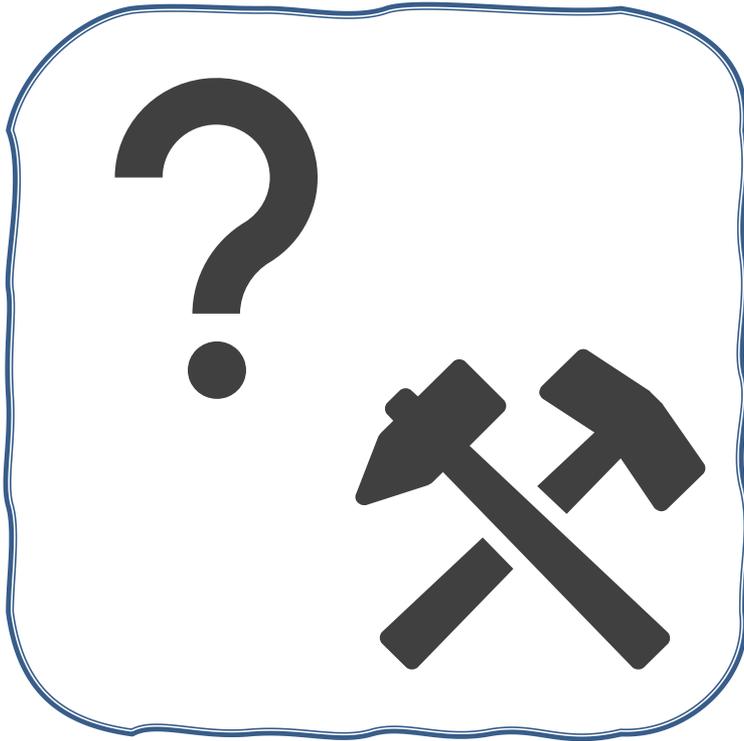
## Script to Visualization



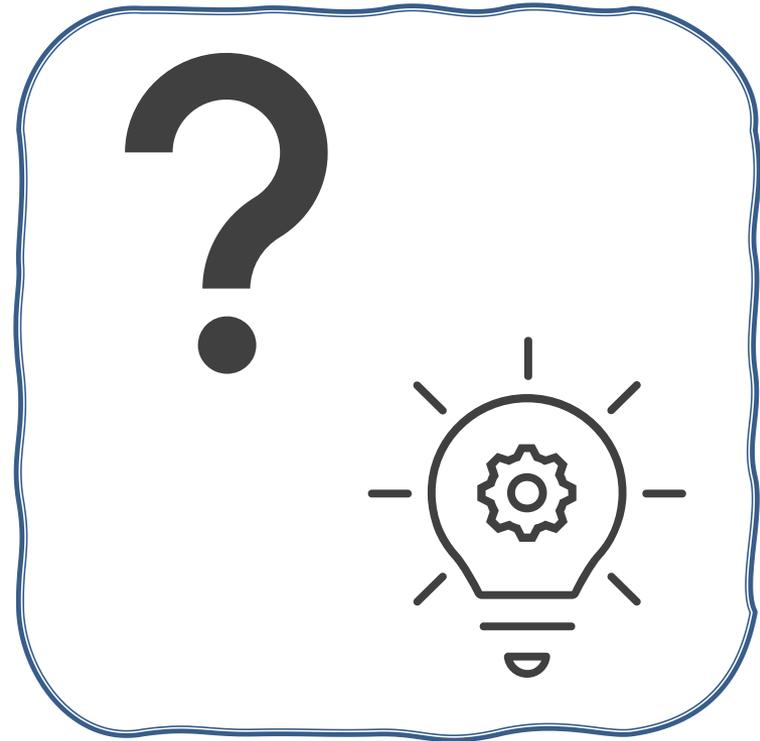
## Visualization to Script



# The Equation of Good Hybrid Design



+



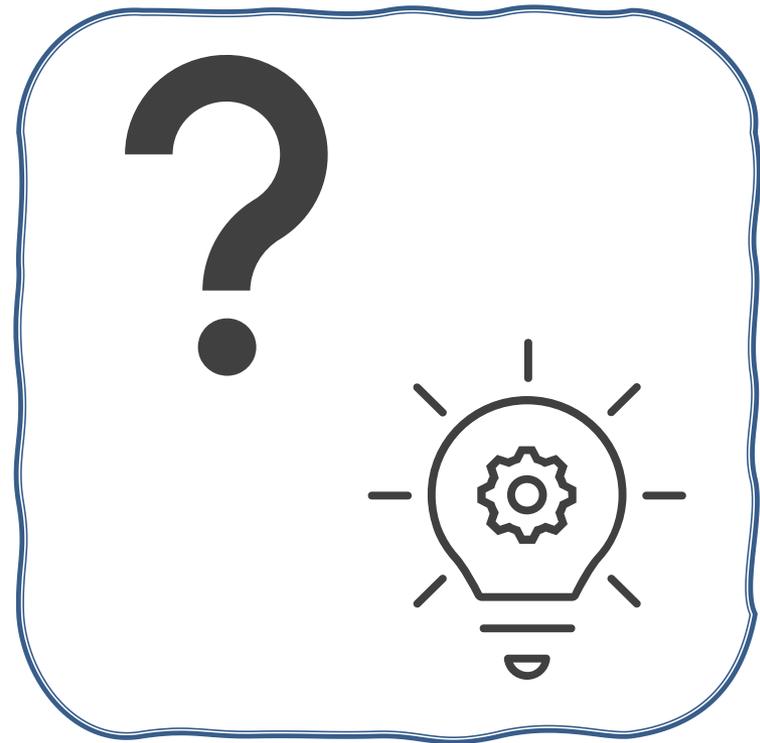
Technology for Implementation

Model of Design

# The Equation of Good Hybrid Design

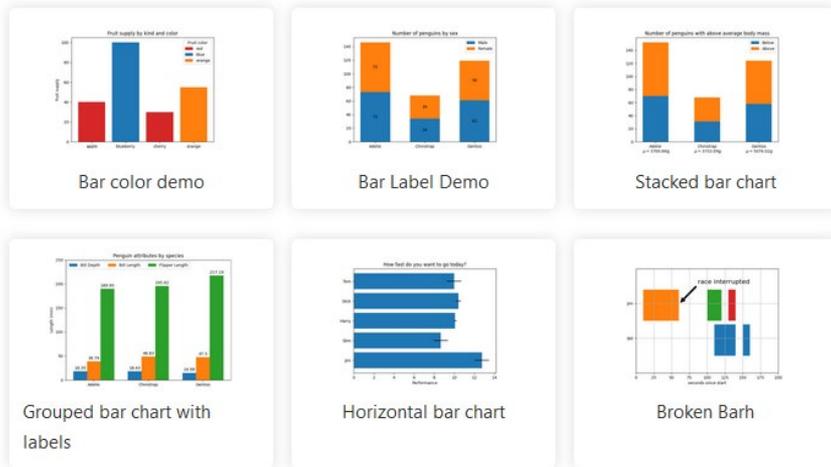


Technology for Implementation



Model of Design

# Python Vis alone is not enough



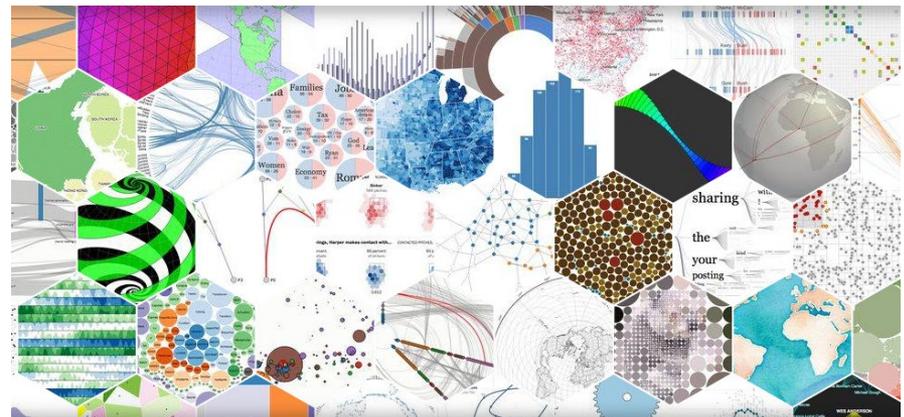
## JavaScript for Visualizations

- Leverages commonly known HTML Document Object Model (DOM)
- Libraries built for mapping data to visual elements intuitively (D3)
- Notebooks are run in browsers and already use JS

## Python for Visualizations

- VIS libraries are not flexible enough for fully custom visualizations
  - (I.E. Matplotlib, Bokeh)
- Use GUI rendering tools with various object/view models
- Low level syntax makes mapping data to visual elements difficult

## Data-Driven Documents



# Libraries for Loading Javascript in Notebooks

## Notebook JS

Library for loading JavaScript from individual notebook cells

**Pro:** Transparent syntax for notebook user

**Con:** No longer being supported/developed

## Roundtrip

Provides interfaces for managing data/state between notebook and JS vis.

**Pro:** Supports complex data and state tracking

**Con:** Unoptimized research code

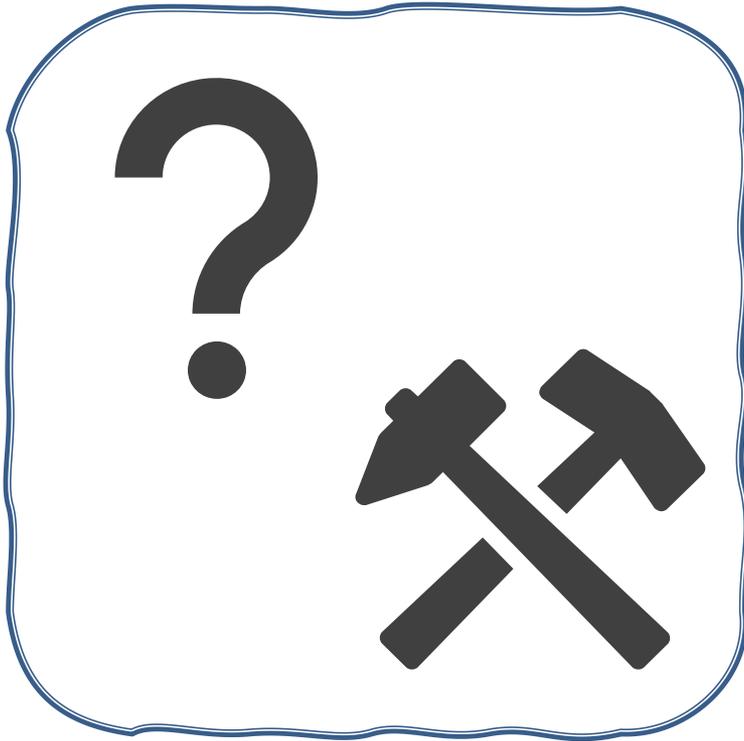
## Jupyter Widgets and Traitlets

Libraries which work together to load vis and manage data transfer between JS and Jupyter.

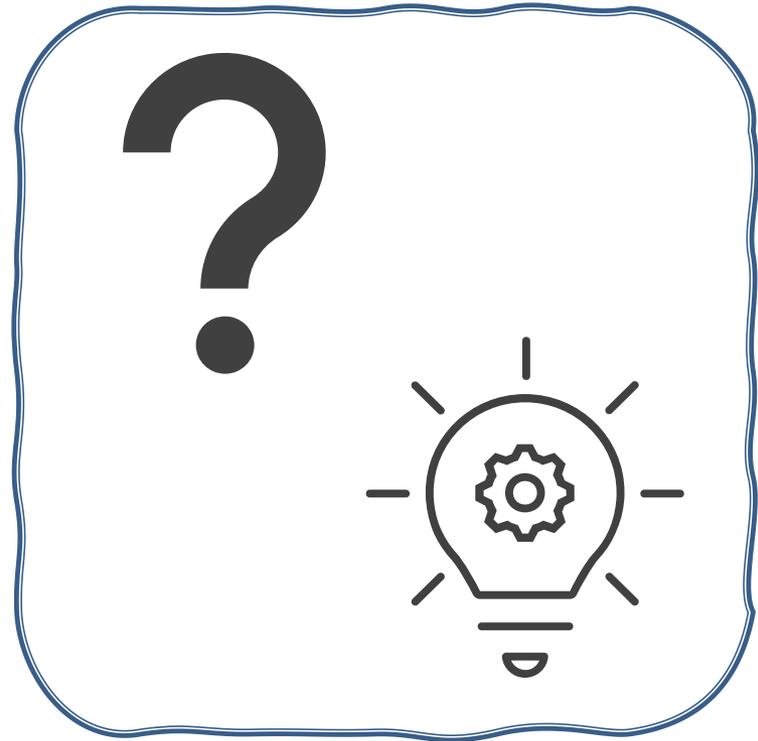
**Pro:** Intuitive object-oriented syntax

**Con:** Tight integration between JS and Python code

# The Equation of Good Hybrid Design



+



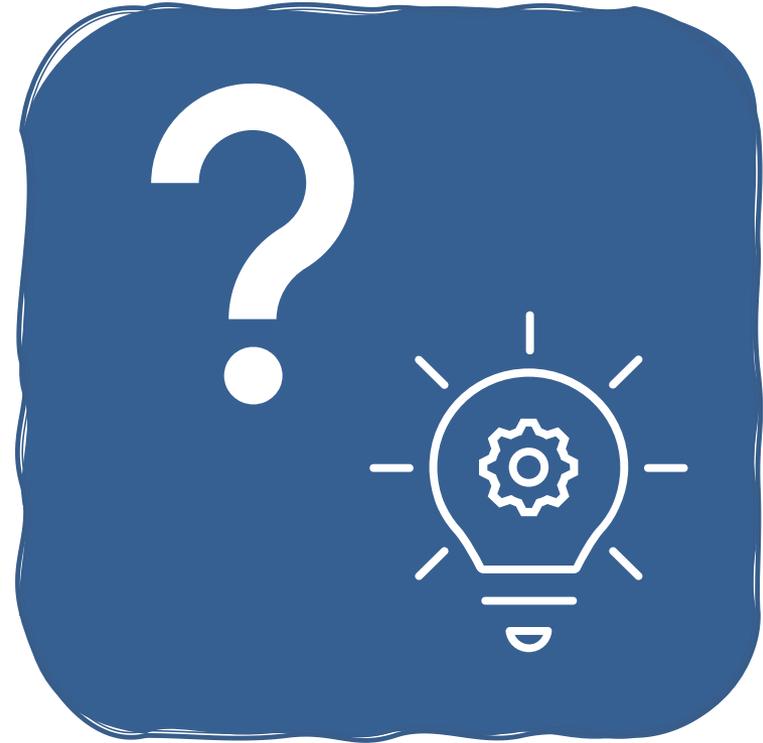
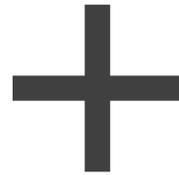
Technology for Implementation

Model of Design

# The Equation of Good Hybrid Design



Roundtrip



Model of Design

# Example Tasks – Performance Analysis

## Tasks

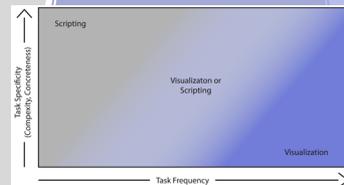
Calculate Speedup (CPU/GPU)

Run Nightly Tests

Find Optimization Opportunities

Report on Work Done

Tasks  
Naturally  
Suited to  
Scripting



Tasks  
Naturally  
Suited to  
Visualization

# Task Categorization

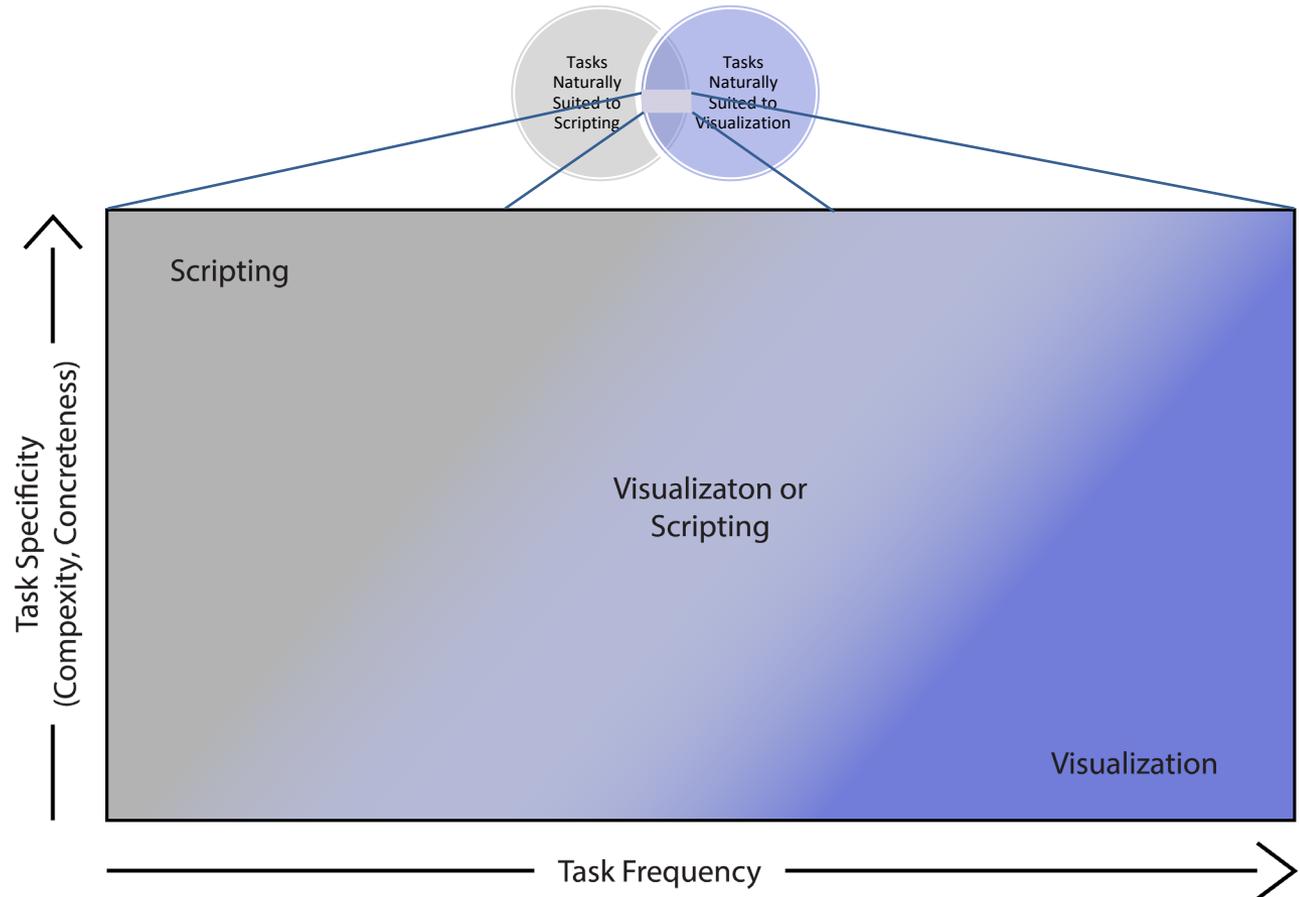
## Tasks

Calculate Speedup  
(CPU/GPU)

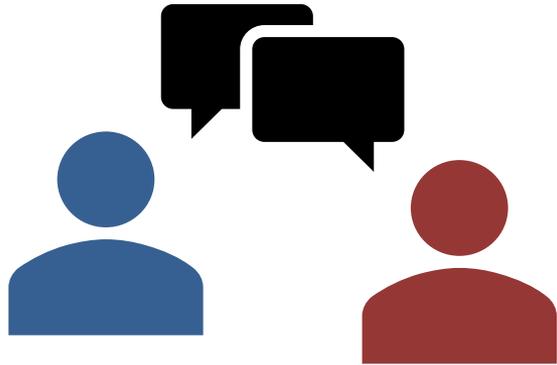
Find Optimization  
Opportunities

Run Nightly Tests

Report on Work  
Done



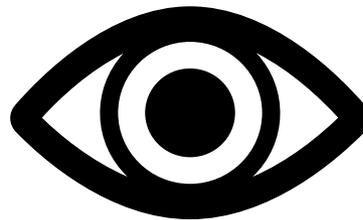
# HCI 101 – Understanding Your User’s Tasks



Interviewing/Discussions

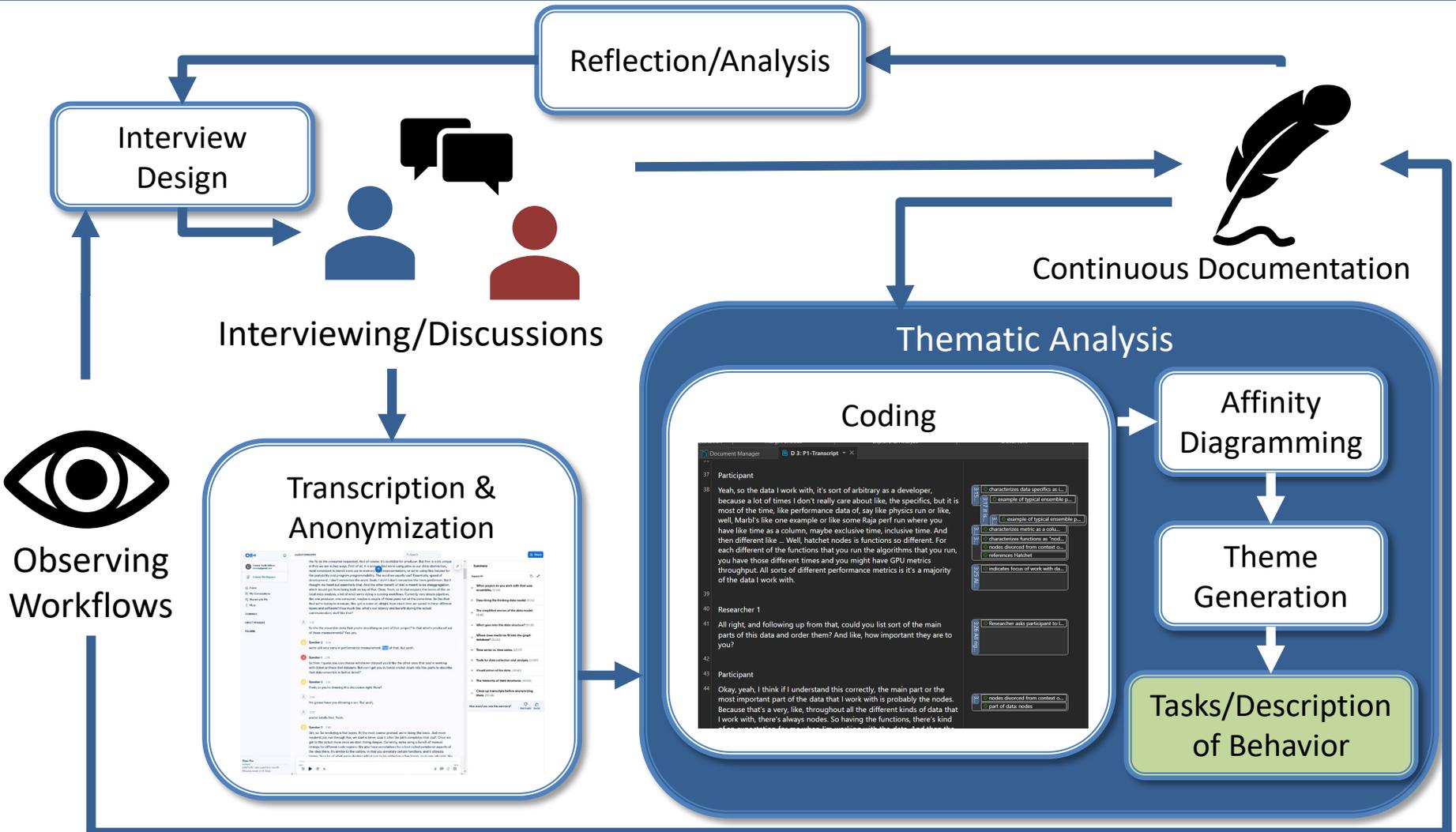


Continuous Documentation



Observing Workflows

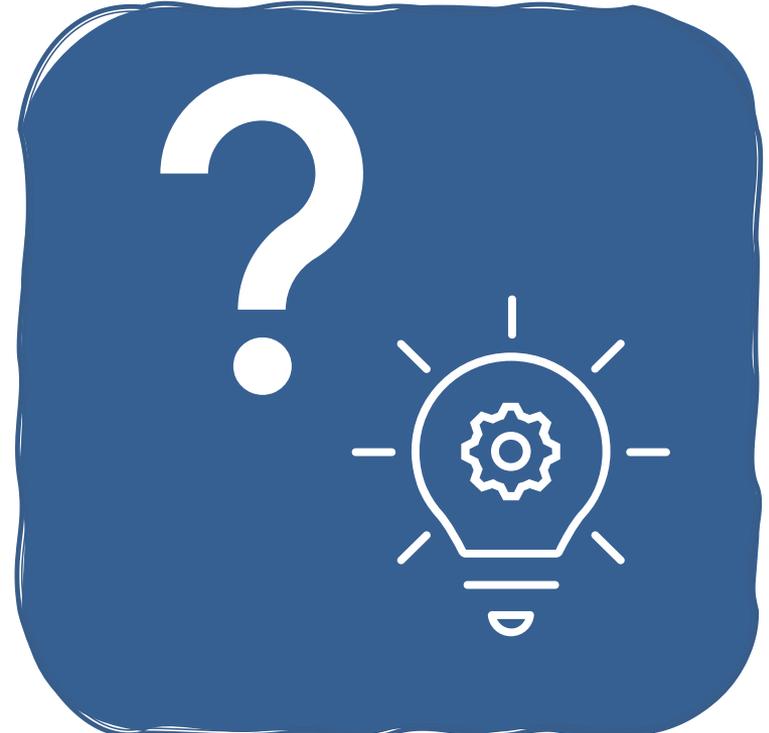
# HCI 601 – What This Actually Entails . . .



# The Equation of Good Hybrid Design



Roundtrip

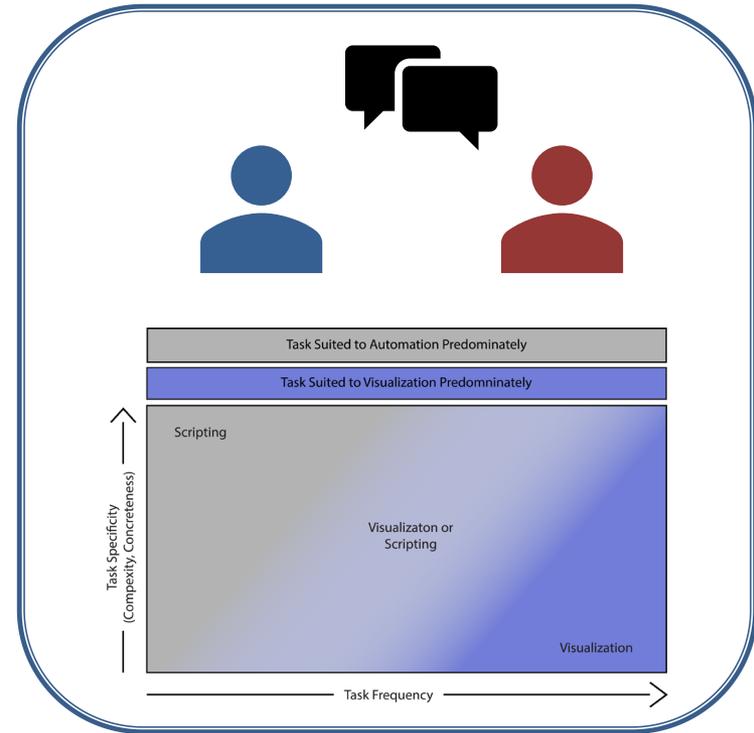


Model of Design

# The Equation of Good Hybrid Design



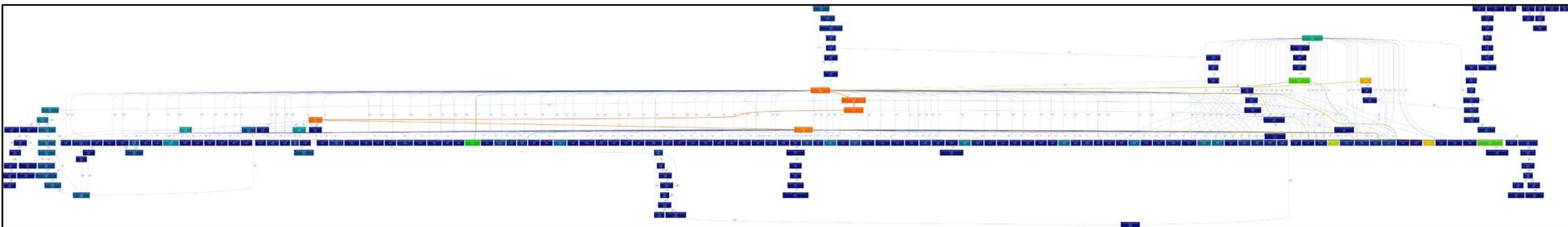
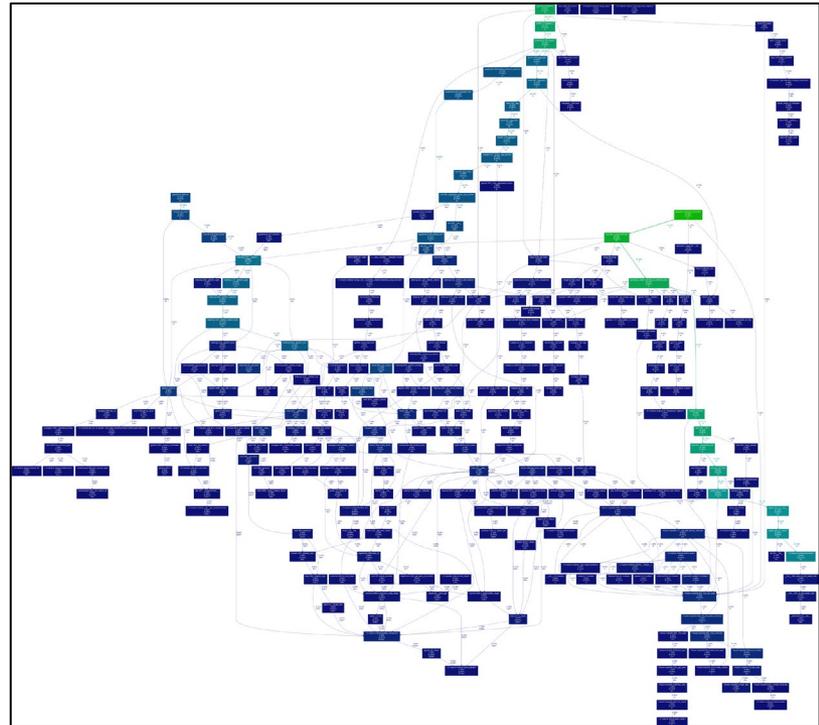
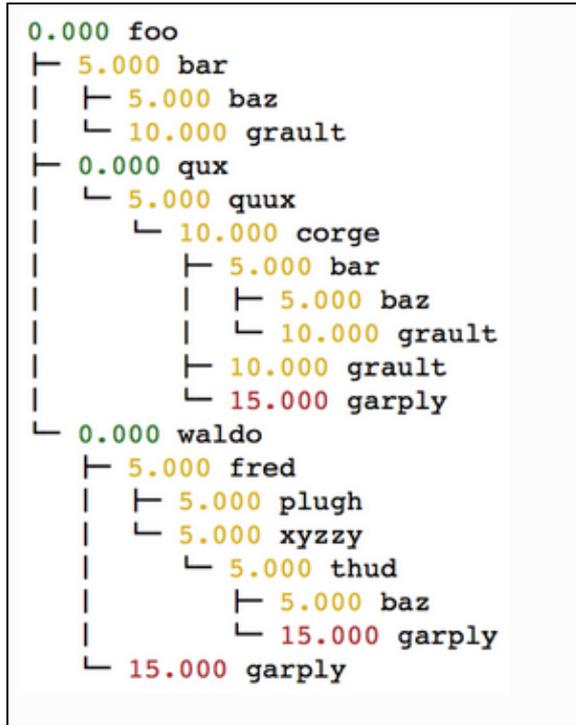
+



Roundtrip

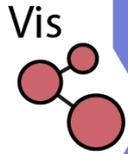
Human Centered Methods +  
Task Categorization

# Our Specific Problem - Calling Context Tree Visualizations

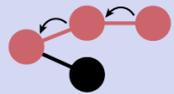


# Our Tasks for Calling Context Tree Analysis

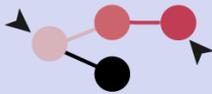
 [T1] Call Path Tracing  
  [T2] Tree Comparison  
  [T3] Metric Analysis  
  [T4] Tree Simplification  
  [T5] Save, Transfer, and Recover Modifications



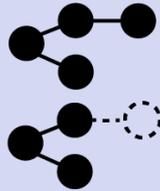
T1.1: Call Path Tracing



T1.2: Ancestor/Descendant Identification



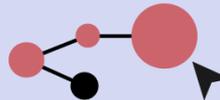
T2.1: Tree Structure Comparison



T2.2: Node Metric Comparison

```
dff["gt_node"] =
gcc_df["time"] > llvm_df["time"]
```

T3.2: Meaningful Outlier Identification



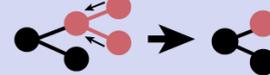
T3.1: Single Metric Outlier Identification

```
print(
graphframe.tree(
metric_column="runtime"
)
)
```

T3.3: Scaling Analysis

```
dff["speedup"] =
dff["64_cores"] / dff["4_cores"]
```

T4.1: Elide Irrelevant Subtrees



T4.2: Elide Nodes Based on Metric



T4.3: Function Identity Based Agg.

```
graphframe.compose_nodes(
node1 = "foo",
node2 = "bar",
new_node = "bar"
)
```

T4.4: Complex Criteria Filtering

```
graphframe.filter(
lambda x: "malloc" not in x["name"]
)
```

 [T5] Save, Transfer, and Recover Modifications

T5.1: Extract Tree State from Vis

```
{ "graph": [
{ name: "foo",
children: ["bar", "baz"]
...}] }
```

T5.2: Store Tree State

```
with open("save.json", "w") as f:
f.write(tree_state)
```

T5.3: Recover Tree State

```
with open("save.json", "r") as f:
tree_state = f.read()
gf = graphframe.filter(tree_state)
```

Script



# %cct for Hatchet

Hatchet  
(Scripting Side)



%cct ?graphframe ?selections\_and\_state

Metrics
Display
Query
Interactive Calling Context Tree

Legend for metric: speedup

- 1.60 - 1.92
- 1.28 - 1.60
- 0.96 - 1.28
- 0.64 - 0.96
- 0.32 - 0.64
- 0.00 - 0.32

Legend for metric: time

- 7.10M - 8.52M
- 5.68M - 7.10M
- 4.26M - 5.68M
- 2.84M - 4.26M
- 1.42M - 2.84M
- 8.00 - 1.42M

Detailed View on Selected Nodes

name	speedup	time	time (inc)
CalcFBHourglassForceForElems	0.79	3.83M	3.83M
IntegrateStressForElems	0.79	3.13M	3.13M
CalcHourglassControlForElems	0.64	8.52M	12.36M

Traditional Tree Layout

Mass Prune

Showing two metrics at once

## Calling Context Tree Example Notebook

```
In [1]: import os, sys
        from IPython.display import HTML, display

        import hatchet as ht
        %load_ext hatchet.vis.loader
```

```
In [2]: gf = ht.GraphFrame.from_hpctoolkit('datasets/kripke-scaling/hpctoolkit-kripke-64-cores/')
```

```
In [ ]: %cct ?gf ?selections_and_state
```

```
In [ ]: %table ?gf ?selections_and_state
```

```
In [ ]:
```

## Calling Context Tree Example Notebook ¶

```
In [ ]: import os, sys
        from IPython.display import HTML, display

        import hatchet as ht
        %load_ext hatchet.vis.loader
```

```
In [ ]: gf = ht.GraphFrame.from_hpctoolkit('datasets/kripke-scaling/hpctoolkit-kripke-64-cores/')
```

```
In [ ]: %cct gf
```

```
In [ ]:
```

## Calling Context Tree Example Notebook

```
In [12]: import os, sys
from IPython.display import HTML, display
```

```
import hatchet as ht
%load_ext hatchet.vis.loader
```

The hatchet.vis.loader extension is already loaded. To reload it, use:  
%reload\_ext hatchet.vis.loader

```
In [13]: gf = ht.GraphFrame.from_hpctoolkit('datasets/kripke-scaling/hpctoolkit-kripke-64-cores/')
```

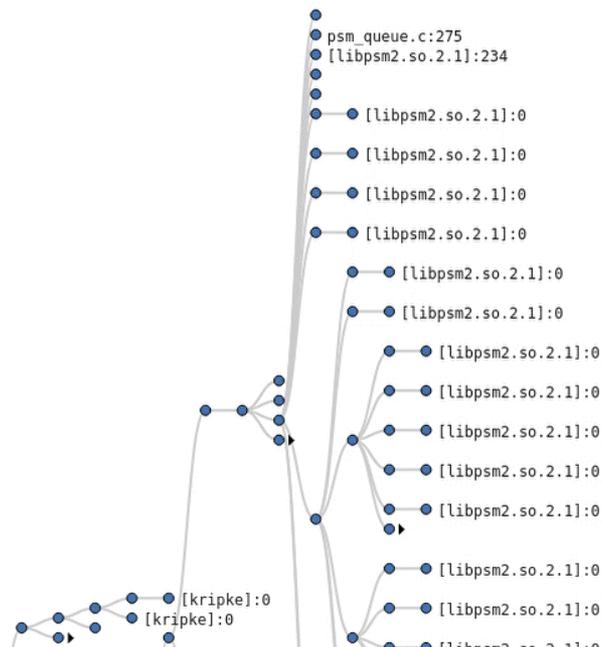
```
In [14]: %cct gf
```

Metrics Display Query Interactive Calling Context Tree

Legend for metric: time Legend for metric: time (inc)

64.32M - 77.19M  
51.46M - 64.32M  
38.59M - 51.46M  
25.73M - 38.59M  
12.86M - 25.73M  
0.00 - 12.86M

159.89M - 191.87M  
127.91M - 159.89M  
95.94M - 127.91M  
63.96M - 95.94M  
31.98M - 63.96M  
0.00 - 31.98M



## Calling Context Tree Example Notebook

```
In [1]: import os, sys
        from IPython.display import HTML, display

import hatchet as ht
%load_ext hatchet.vis.loader
```

```
In [2]: """
        The following are convenience functions provided to you for this tutorial, and define some common operations.
        They cannot operate on dataframes produced from eachother,
        so only use them on dataframes directly loaded from a dataset
        """

def affixColumnToGraphframe(dest_gf, src_gf, colname_dest, colname_src):
    """
    Attaches a column from one graph frame to another. Returns a new
    graphframe with the requested column.
    Note: will not produce meaningful results if node names and node id's are not aligned
    between datasets

    Params:
    dest_gf: the destination graphframe for the column
    src_gf: the source graphframe for the column
    colname_dest: the target column name on the desination graphframe
    colname_src: the name of the column we would like to transfer from source
    """
    gf_new = dest_gf.copy()
    src_gf = src_gf.copy()

    src_gf.dataframe[colname_dest] = src_gf.dataframe[colname_src]
    src_gf.dataframe = src_gf.dataframe.drop(columns=['time (inc)', 'time'])

    gf_new.dataframe = gf_new.dataframe \
        .reset_index() \
        .join(\
            src_gf.dataframe.reset_index().set_index(['nid', 'name']),
            on=['nid', 'name'],
            lsuffix='_l',
            rsuffix='_r'
        )

    if('_missing_node' in gf_new.dataframe.columns):
        gf_new.dataframe = gf_new.dataframe.drop(columns=['_missing_node'])

    removes = [c for c in gf_new.dataframe.columns if '_r' in c]
    renames = {}

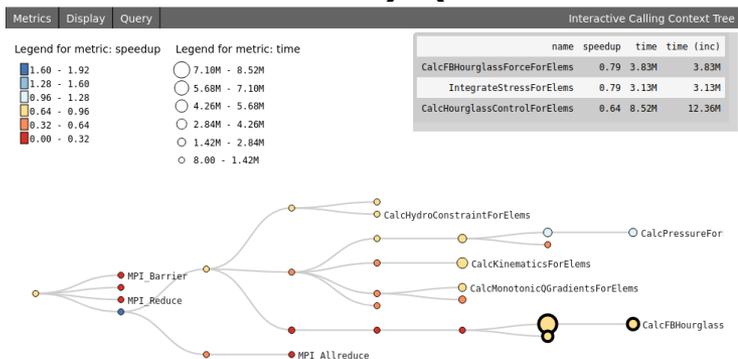
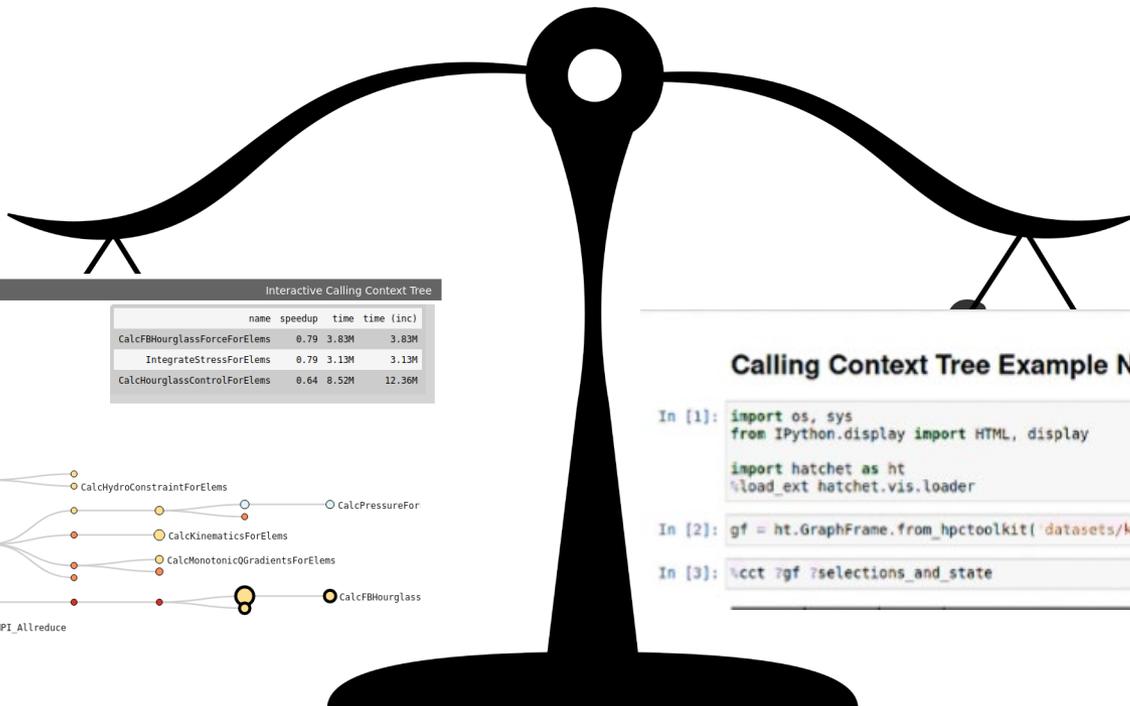
    for c in gf_new.dataframe.columns:
        if c[-2:] == '_l':
            renames[c] = c[:-2]

    gf_new.dataframe = gf_new.dataframe.drop(columns=removes).rename(columns=renames).set_index(['node'])

    gf_new.exc_metrics.append(colname_dest)

    return gf_new

def calcSpeedup(gf1, gf2):
    # Calculates the speedup between two graph frames
    # with the same function calls
```



### Calling Context Tree Example Notebook

```
In [1]: import os, sys
        from IPython.display import HTML, display

        import hatchet as ht
        %load_ext hatchet.vis.loader

In [2]: gf = ht.GraphFrame.from_hpctoolkit('datasets/kripke-scaling/hpctoolkit-kripke-64-cores/')

In [3]: %cct ?gf ?selections_and_state
```

We want to integrate scripting and visualization better for this community to support their workflows.



Thicket

<https://github.com/LLNL/thicket>



## Proposal

Working group here to further human-oriented discussions of performance analysis



Hatchet

<https://github.com/LLNL/hatchet>



**Disclaimer**

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC