Hybrid Scripting/Vis Tool Development in Jupyter Notebooks

Leveraging human centric methods for performance analysis workflows

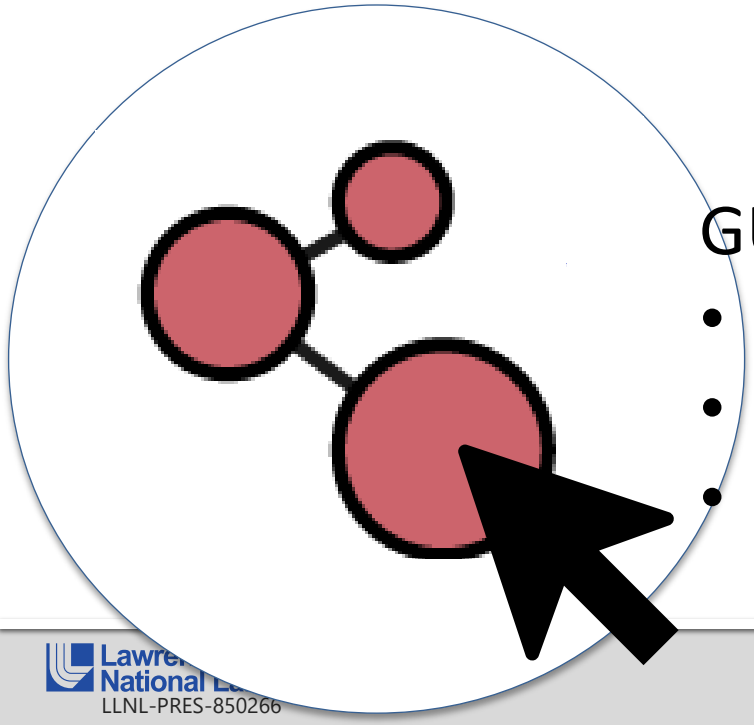
Scalable Tools - June 19, 2023

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

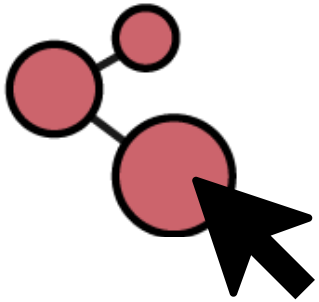


GUI-Based tools and visualizations for:

- 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.



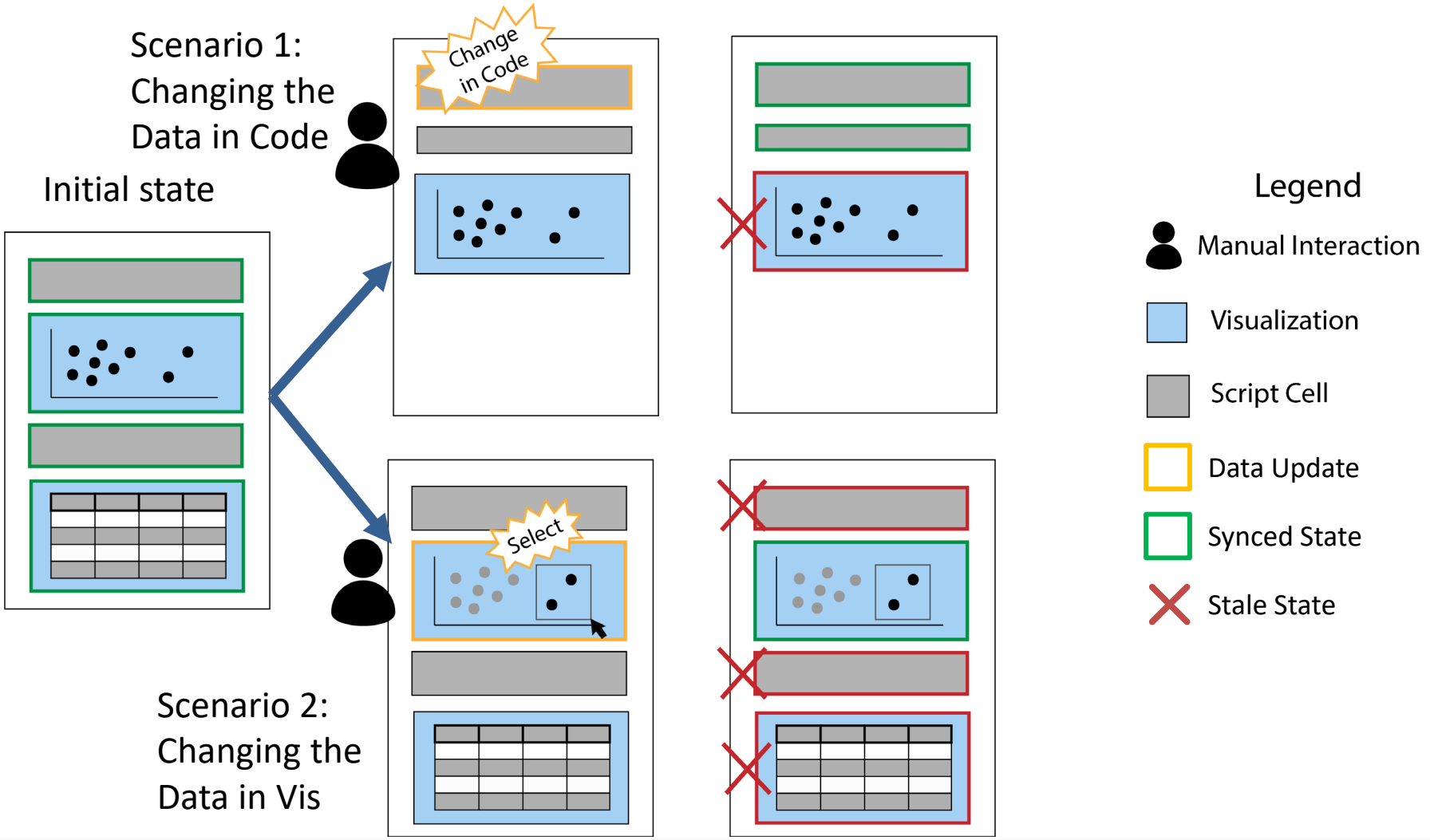
Interactive Visualization can support fluid exploration but is often limited to pre-determined tasks

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

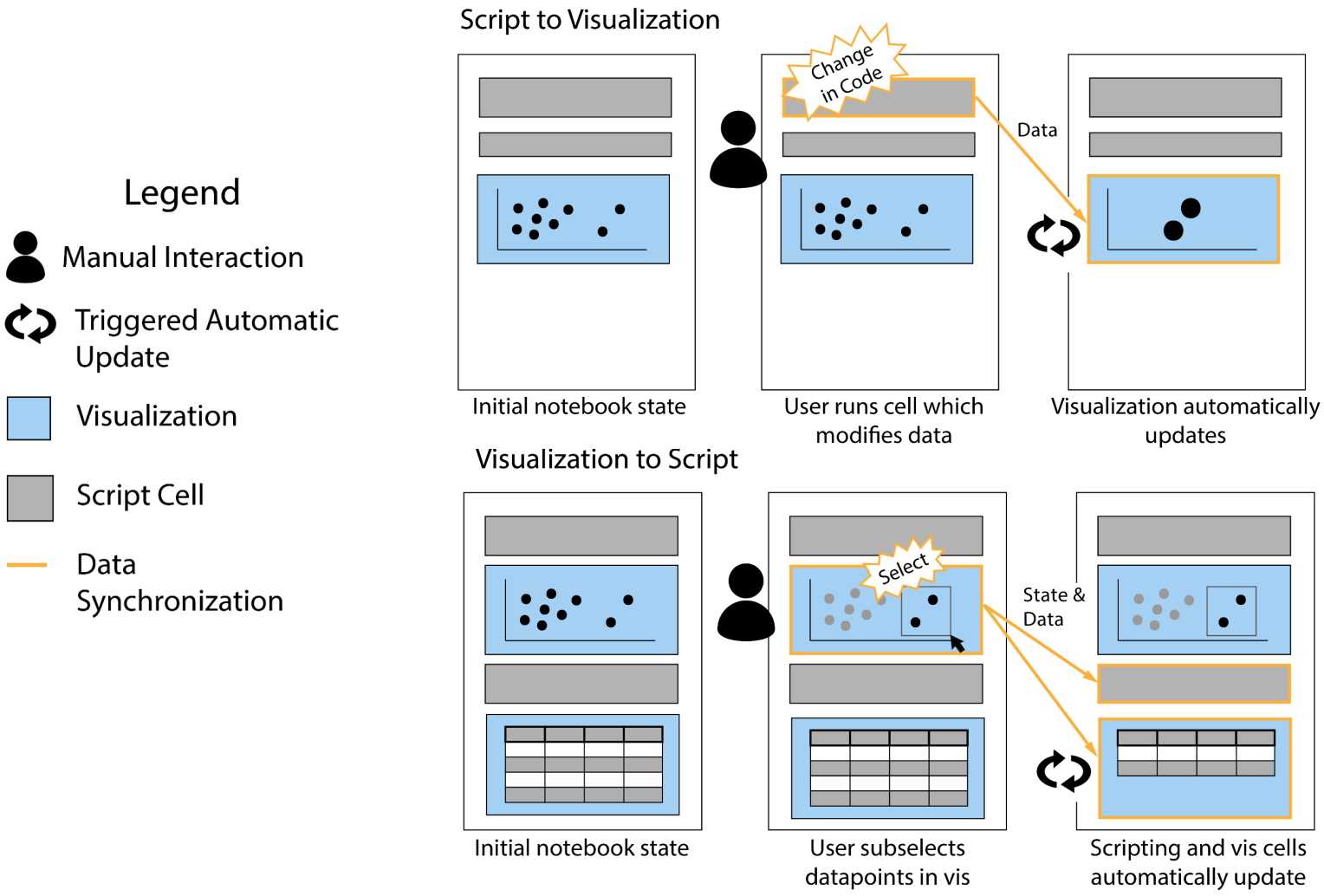
We develop tools embedded in Jupyter notebooks that leverage both visualization and scripting to give the users flexibility they seek.

Do Jupyter Notebooks alone fill this gap?

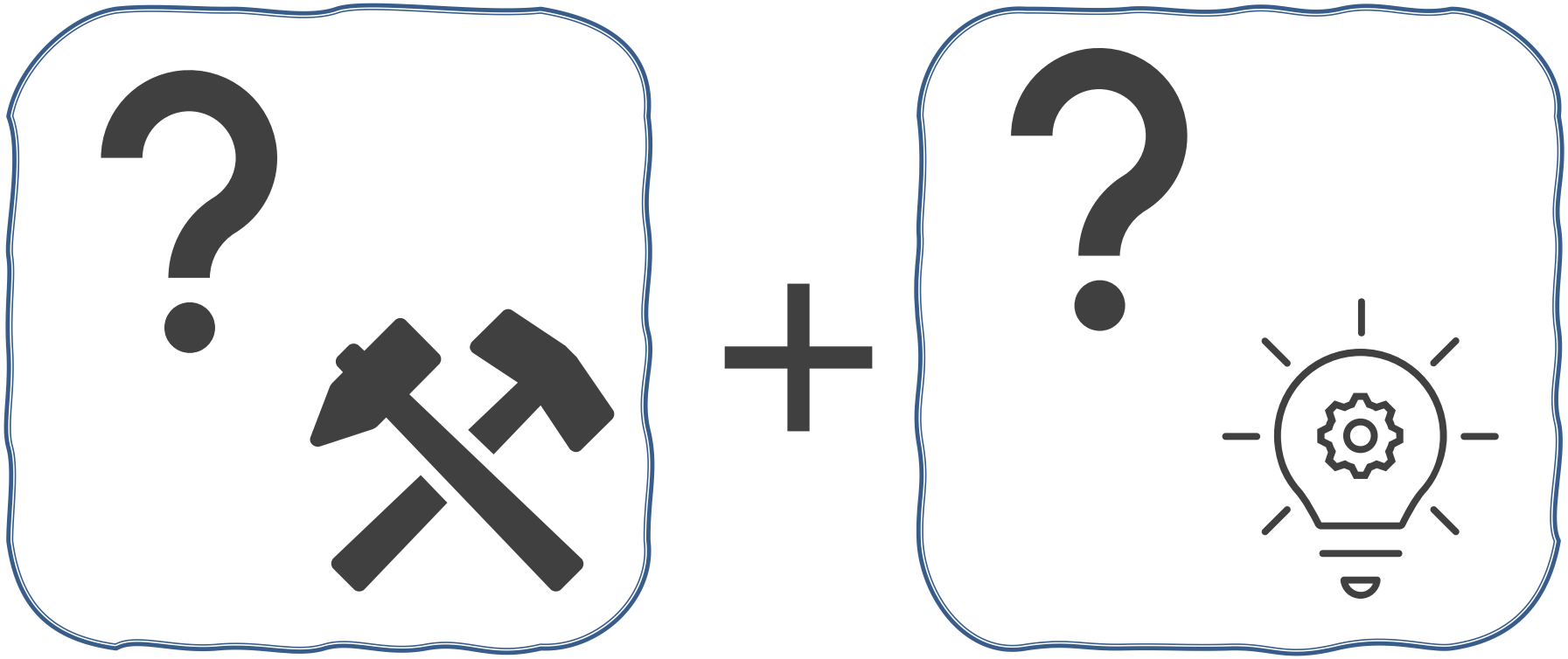
Where Notebooks Fail to Support Hybrid Workflows



What is the better model for a hybrid workflow?



The Equation of Good Hybrid Design

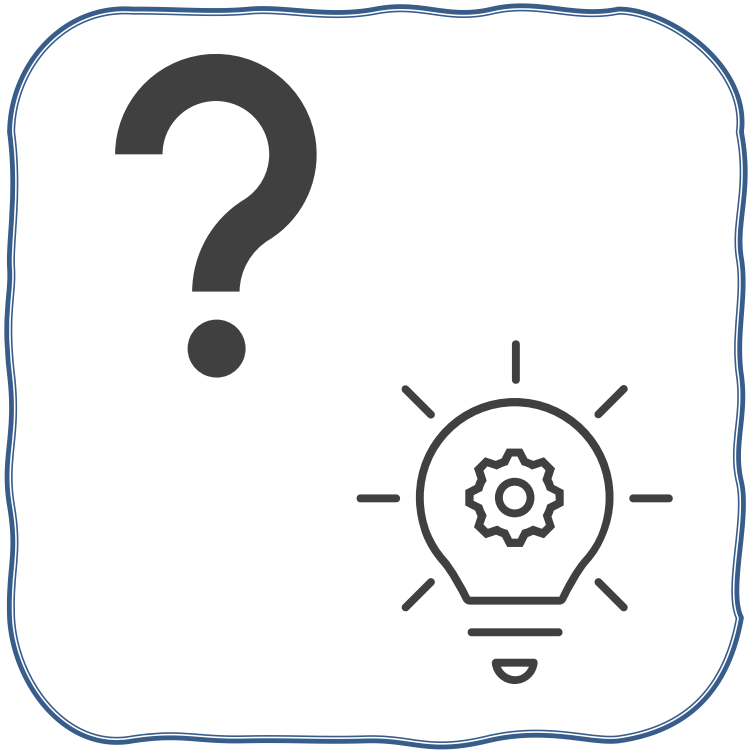


Technology for Implementation

Model of Design

The Equation of Good Hybrid Design

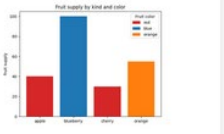
+

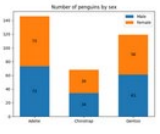


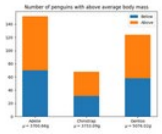
Technology for Implementation

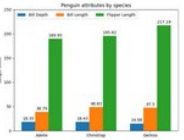
Model of Design

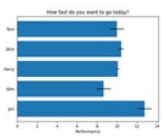
Python Vis alone is not enough

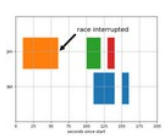

Bar color demo


Bar Label Demo


Stacked bar chart


Grouped bar chart with labels


Horizontal bar chart



Broken Barh

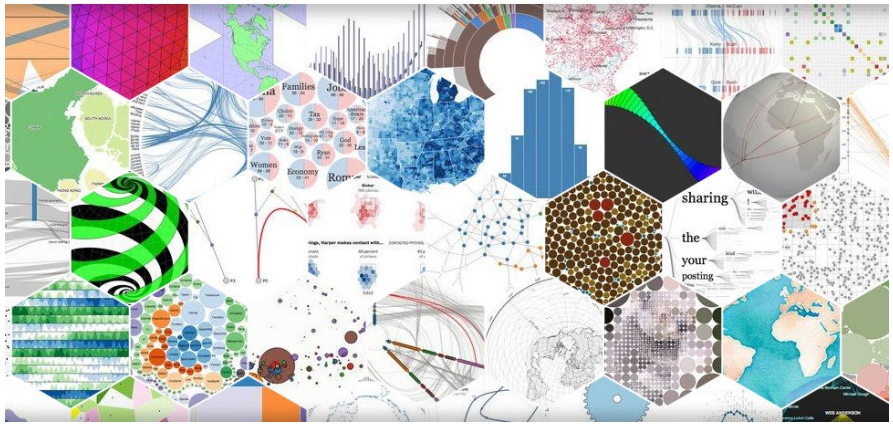
Python for Visualizations

- 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

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

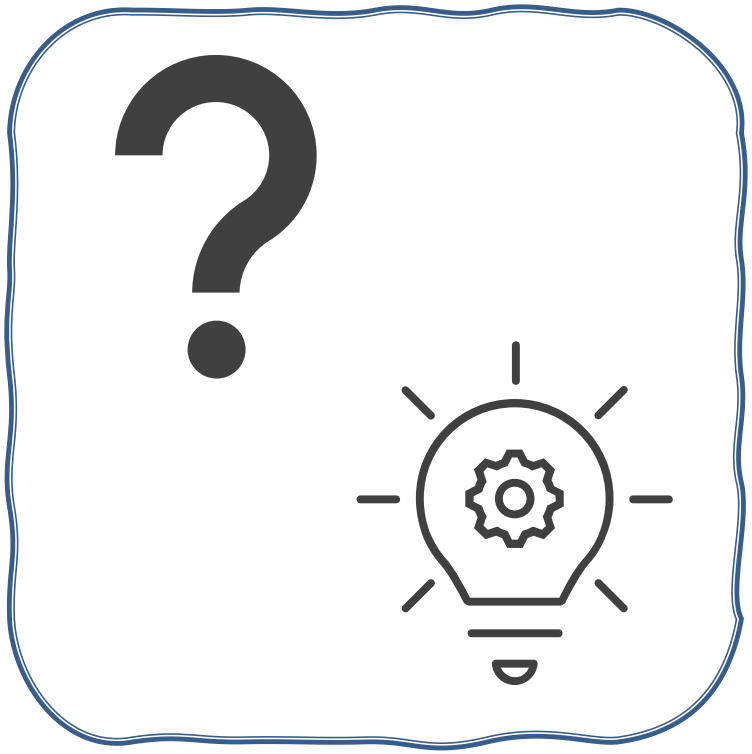
Data-Driven Documents



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

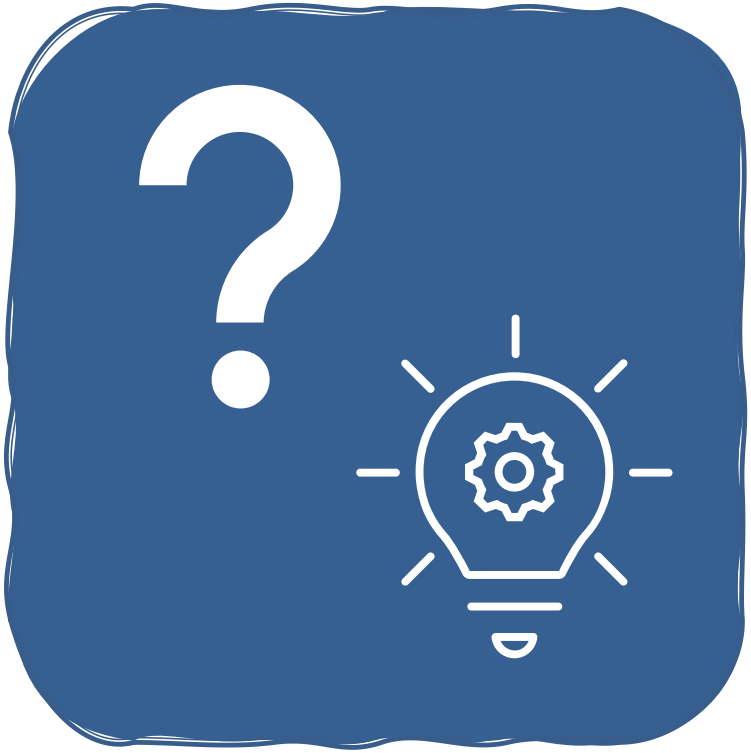


Technology for Implementation

Model of Design

The Equation of Good Hybrid Design

+

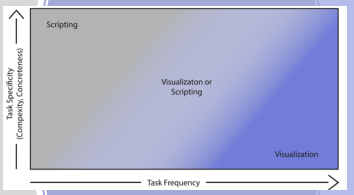


Roundtrip

Model of Design

Example Tasks – Performance Analysis

Tasks Naturally Suited to Scripting

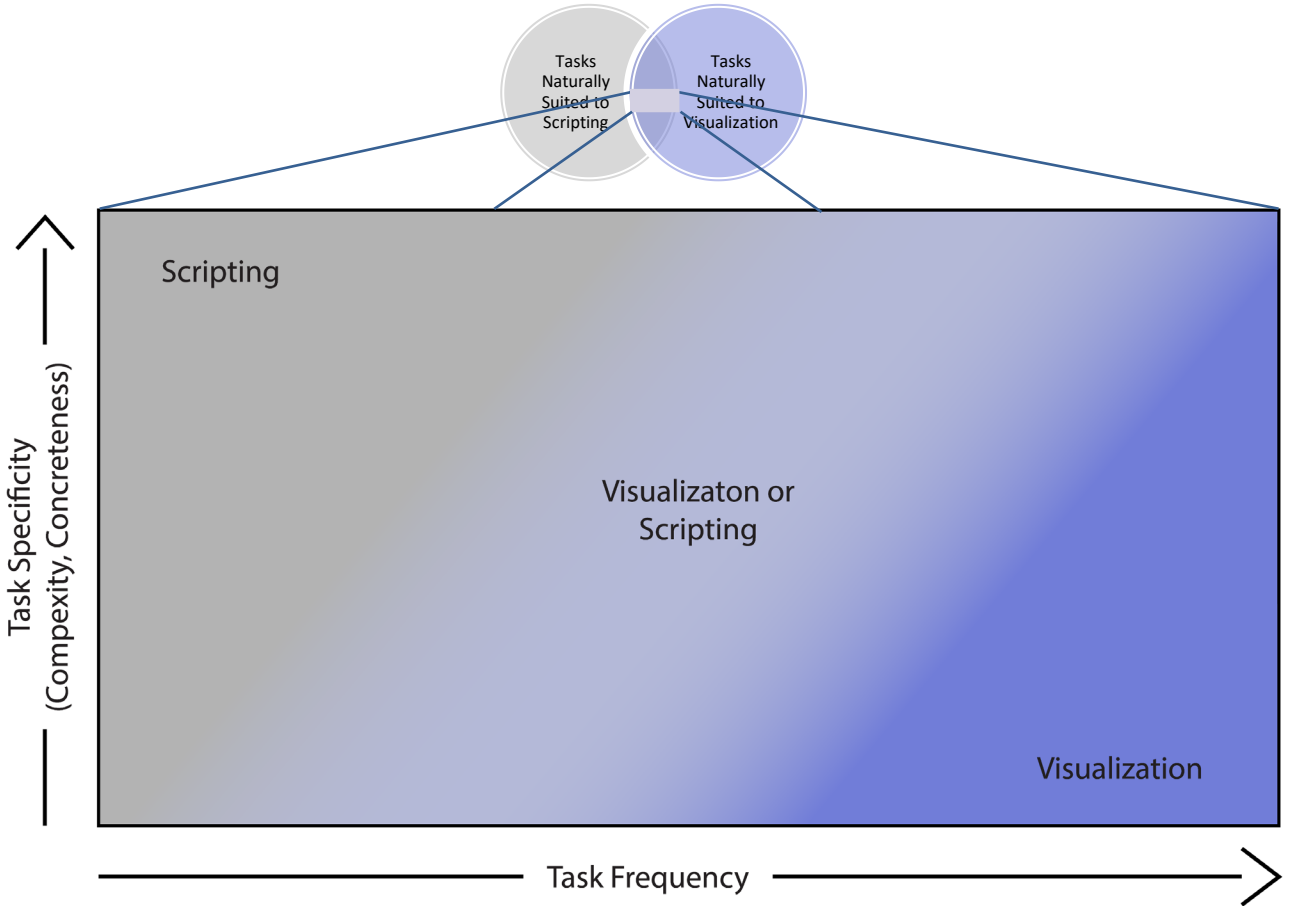


Tasks Naturally Suited to Visualization

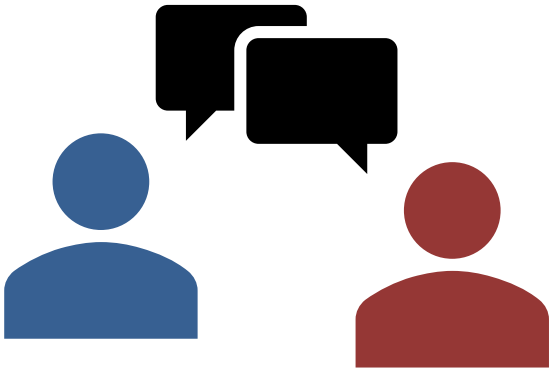
Task Categorization

Tasks

- Calculate Speedup (CPU/GPU)
- Find Optimization Opportunities
- Run Nightly Tests
- Report on Work Done

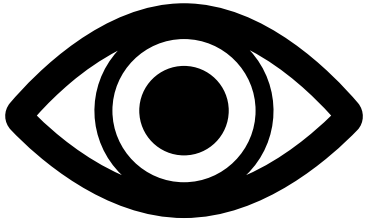


HCI 101 – Understanding Your User's Tasks



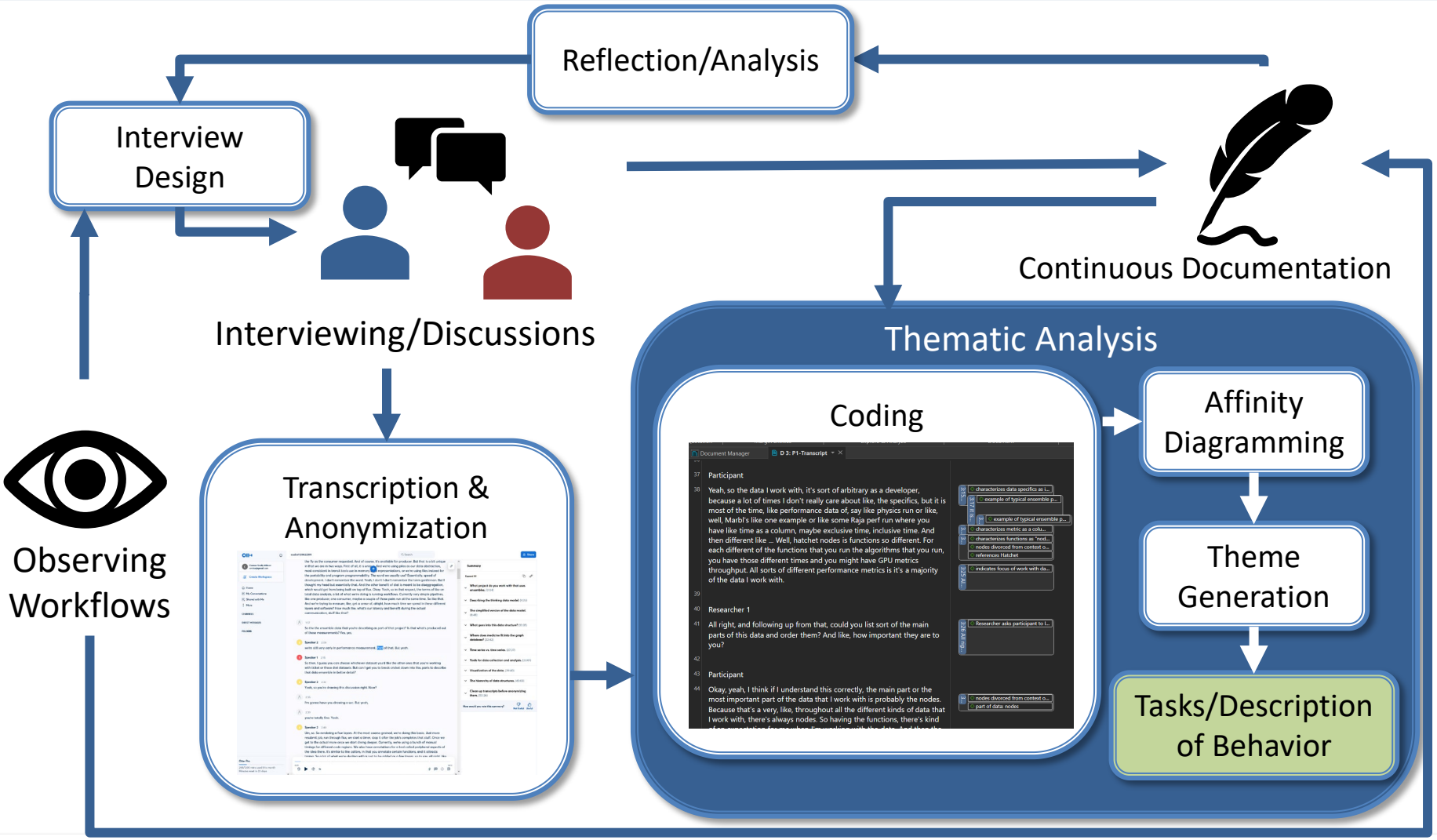
Interviewing/Discussions

Continuous Documentation



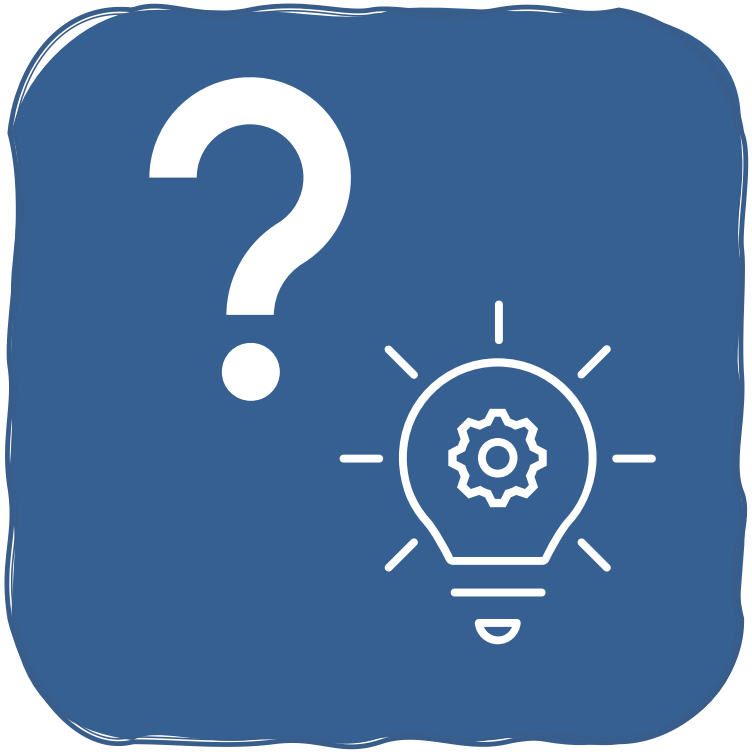
Observing Workflows

HCI 601 – What This Actually Entails . . .



The Equation of Good Hybrid Design

+

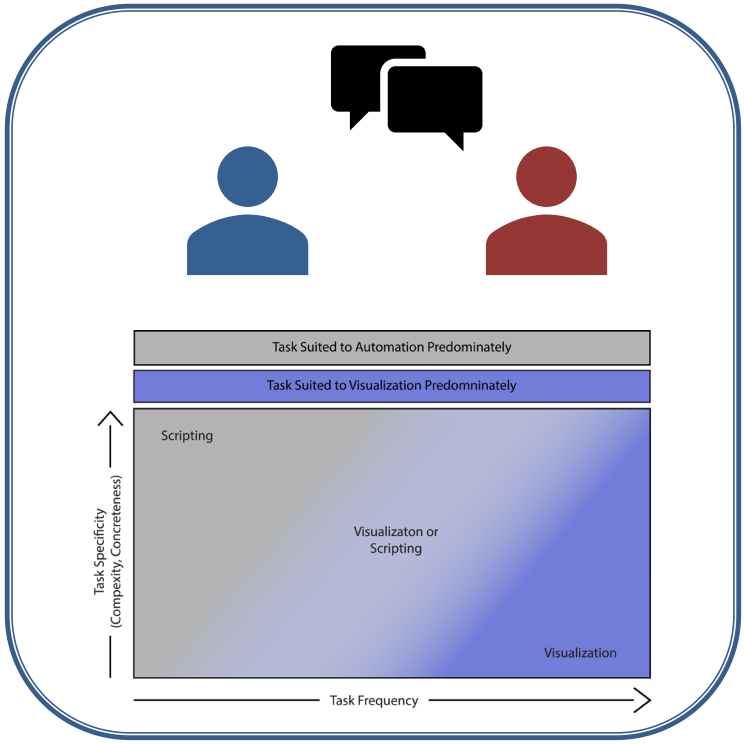


Roundtrip

Model of Design

The Equation of Good Hybrid Design

+

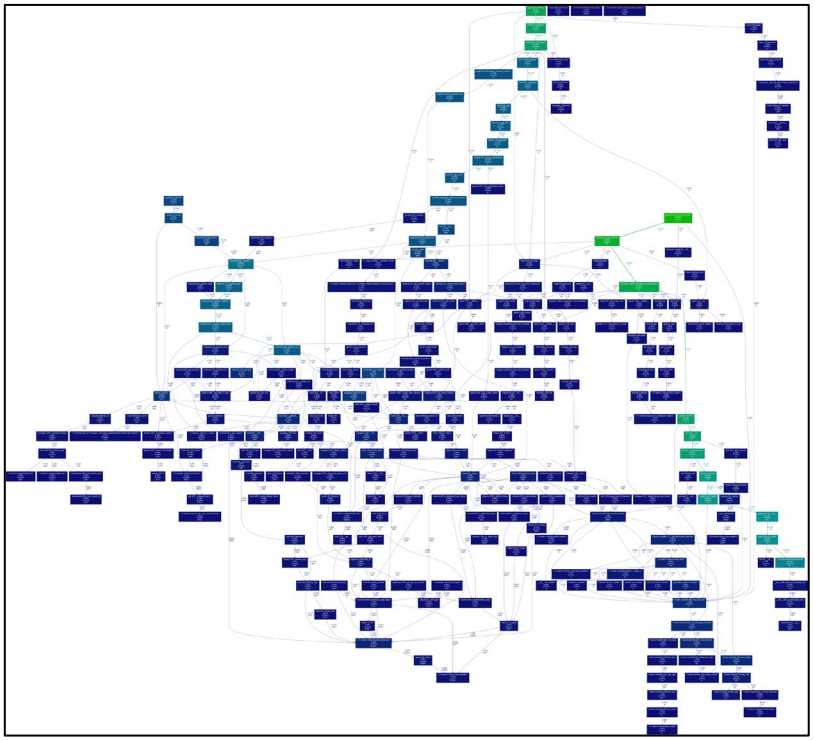


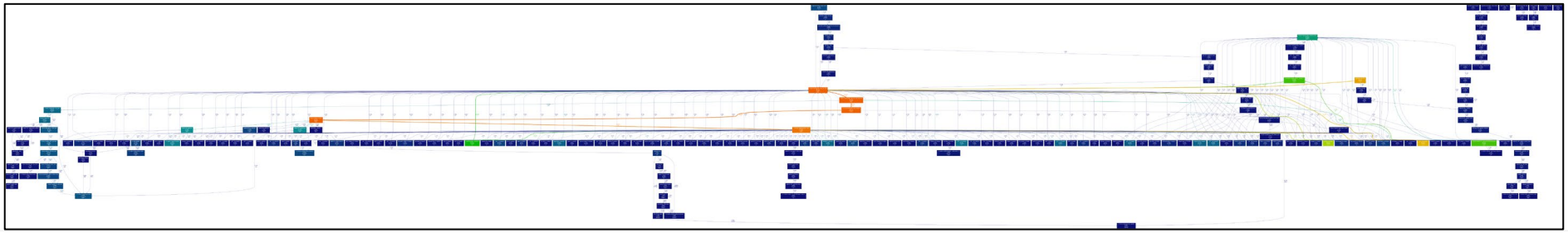
Roundtrip

Human Centered Methods +
Task Categorization

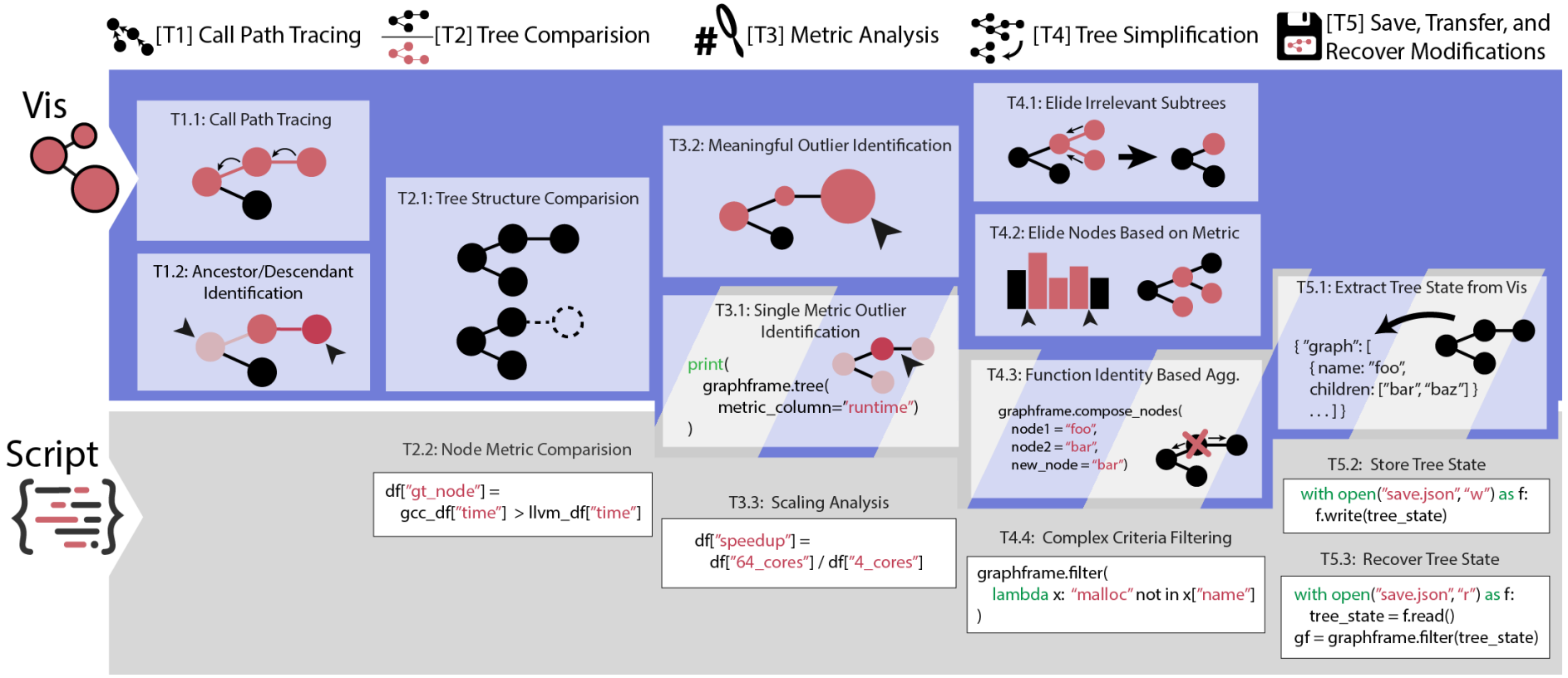
Our Specific Problem - Calling Context Tree Visualizations

```
0.000 foo
├─ 5.000 bar
│  ├─ 5.000 baz
│  └─ 10.000 grault
├─ 0.000 qux
│  └─ 5.000 quux
│     └─ 10.000 corge
│        ├─ 5.000 bar
│        │  ├─ 5.000 baz
│        │  └─ 10.000 grault
│        ├─ 10.000 grault
│        └─ 15.000 garply
└─ 0.000 waldo
   ├─ 5.000 fred
   │  ├─ 5.000 plugh
   │  └─ 5.000 xyzzy
   │     └─ 5.000 thud
   │        ├─ 5.000 baz
   │        └─ 15.000 garply
   └─ 15.000 garply
```





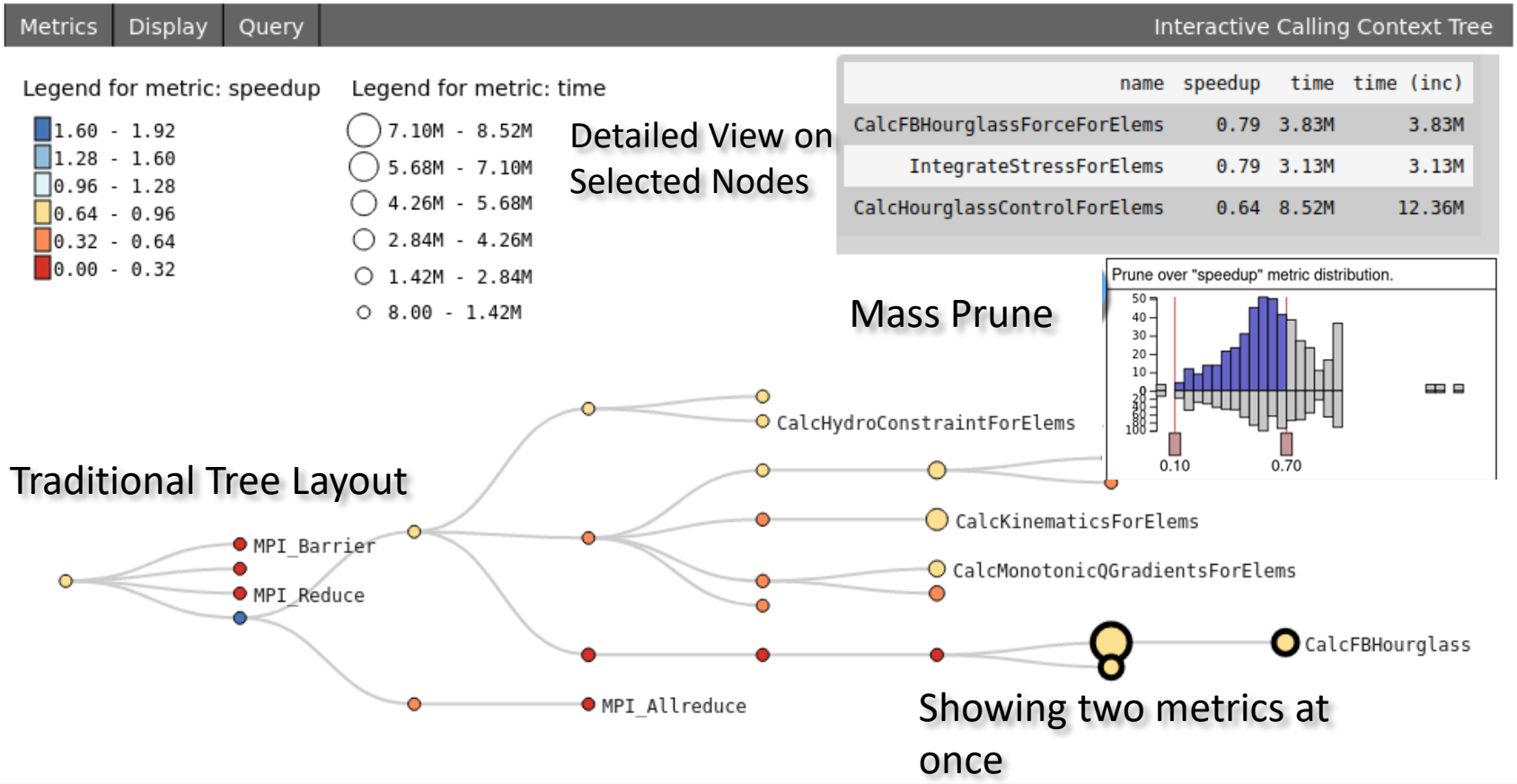
Our Tasks for Calling Context Tree Analysis



%cct for Hatchet

Hatchet
(Scripting Side)

%cct ?graphframe ?selections_and_state



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
```

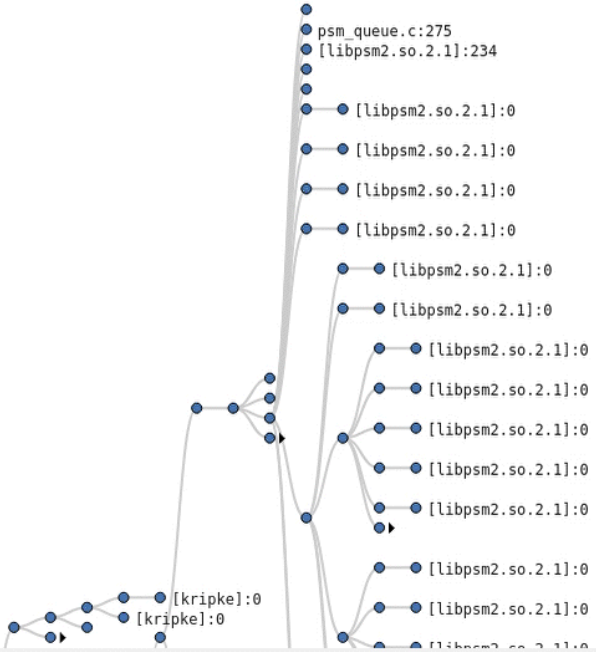
Metrics | Display | Query — Interactive Calling Context Tree

Legend for metric: time

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

Legend for metric: time (inc)

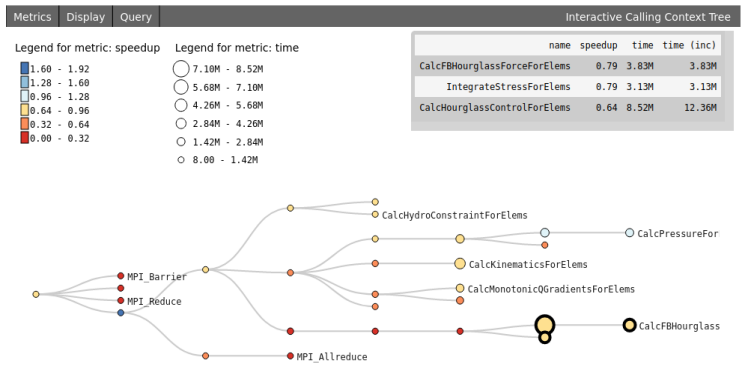
- 159.89M - 191.87M
- 127.91M - 159.89M
- 95.94M - 127.91M
- 63.96M - 95.94M
- 31.98M - 63.96M
- 0.00 - 31.98M



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 dastaset
"""

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
```



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

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