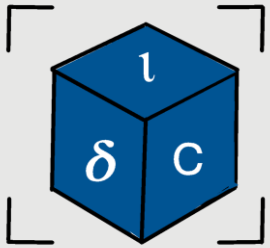
Reproducible Notebook Containers using Application Virtualization



[1]Raza Ahmad, [2]Naga Nithin Manne, [1]Tanu Malik

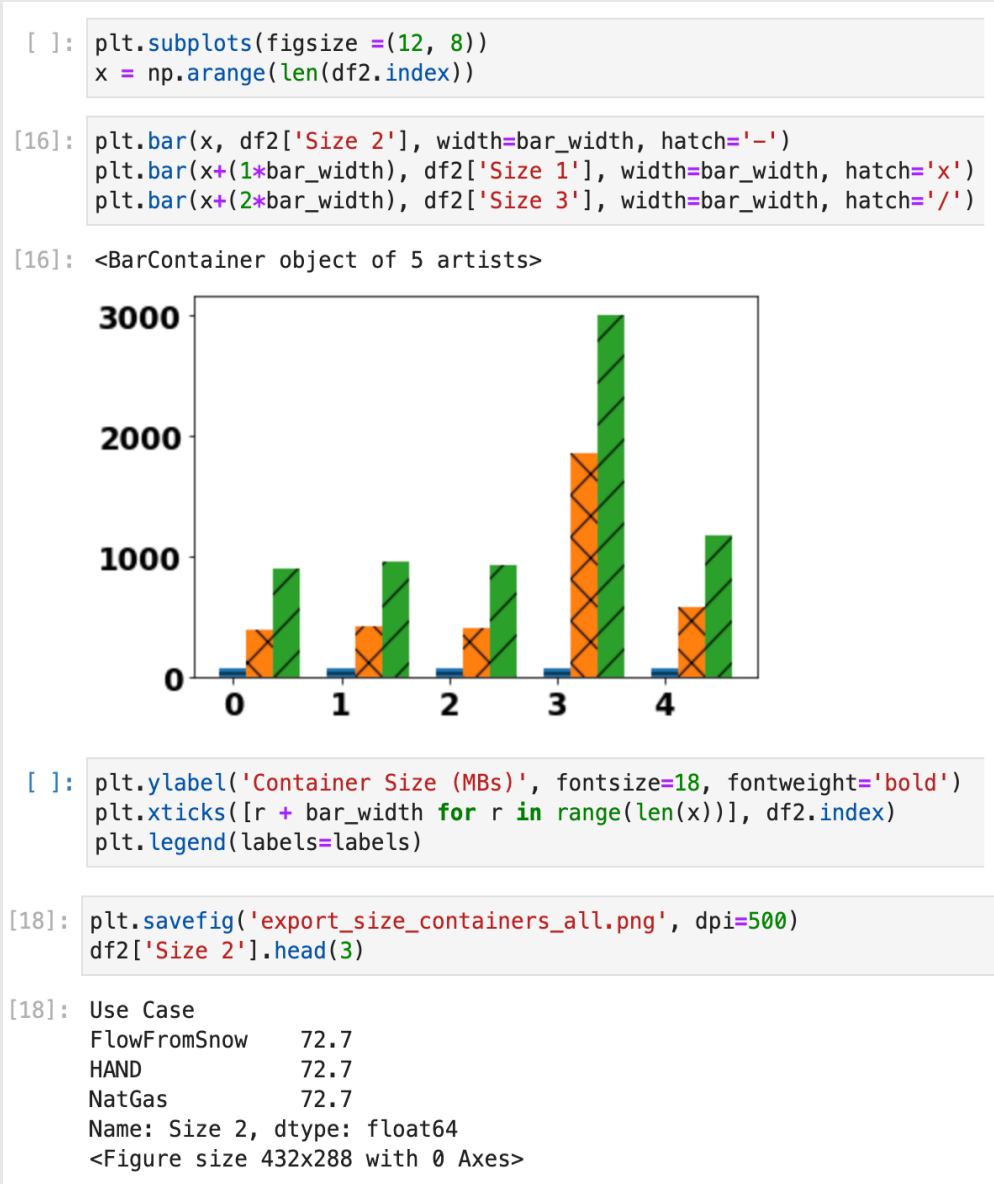
[1]School of Computing DePaul University,

[2]Argonne National Lab

Motivation

- Improve reproducibility of notebooks
- Notebooks are emerging programming paradigm in scientific computing.
  - Exploring data, executing models, visualizing results
- Reproducing notebooks is challenging
  - Hinders collaborative analytics, impacts data democratization
- We present FLINC, a system for notebook reproducibility using application virtualization
  - Enables notebook sharing using lighter weight containers
  - Provides flexible reproducibility

Notebook is Interactive and Shareable



```
[ ]: plt.subplots(figsize =(12, 8))
     x = np.arange(len(df2.index))

[16]: plt.bar(x, df2['Size 2'], width=bar_width, hatch='-')
      plt.bar(x+(1*bar_width), df2['Size 1'], width=bar_width, hatch='x')
      plt.bar(x+(2*bar_width), df2['Size 3'], width=bar_width, hatch='/')

[16]: <BarContainer object of 5 artists>
```



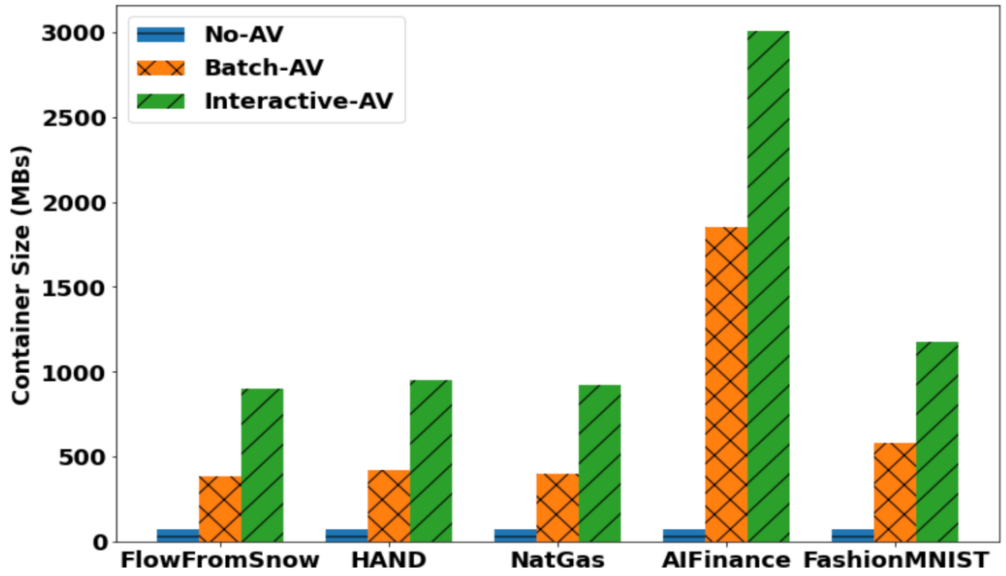
```
[ ]: plt.ylabel('Container Size (MBs)', fontsize=18, fontweight='bold')
     plt.xticks([r + bar_width for r in range(len(x))], df2.index)
     plt.legend(labels=labels)

[18]: plt.savefig('export_size_containers_all.png', dpi=500)
      df2['Size 2'].head(3)

[18]: Use Case
      FlowFromSnow    72.7
      HAND            72.7
      NatGas          72.7
      Name: Size 2, dtype: float64
      <Figure size 432x288 with 0 Axes>
```

Notebook is like a "Container"

```
[9]: plt.subplots(figsize =(12, 8))
     x = np.arange(len(df2.index))
     plt.bar(x, df2['Size 2'], width=bar_width, hatch='-')
     plt.bar(x+(1*bar_width), df2['Size 1'], width=bar_width, hatch='x')
     plt.bar(x+(2*bar_width), df2['Size 3'], width=bar_width, hatch='/')
```

```
[9]: plt.ylabel('Container Size (MBs)', fontsize=18, fontweight='bold')
     plt.xticks([r + bar_width for r in range(len(x))], df2.index)
     plt.legend(labels=labels)
     plt.savefig('export_size_containers_all.png', dpi=500)
     df2['Size 2'].head(3)
```

```
[9]: Use Case
     FlowFromSnow    72.7
     HAND            72.7
     NatGas          72.7
     Name: Size 2, dtype: float64
```

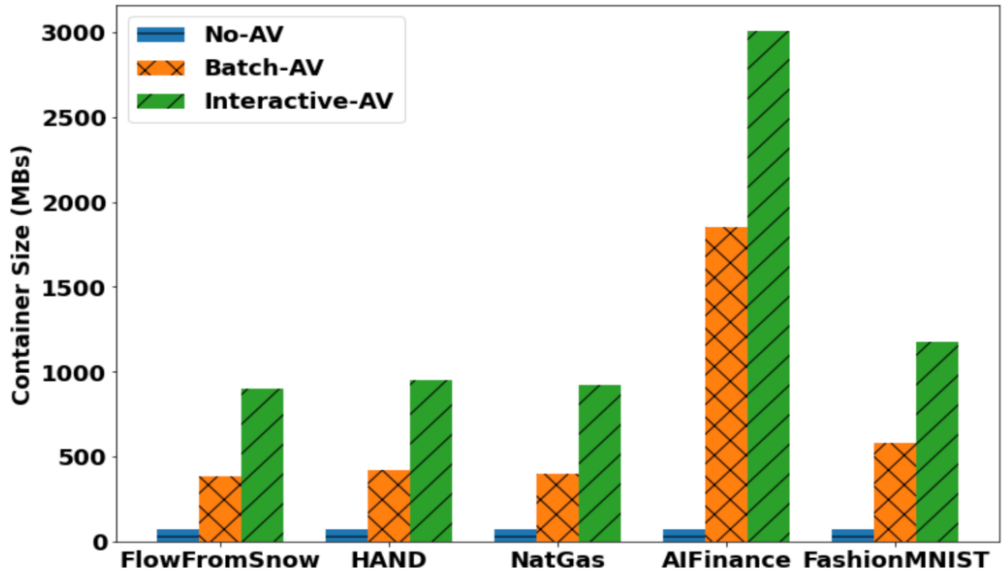


Notebook is like a "Container"

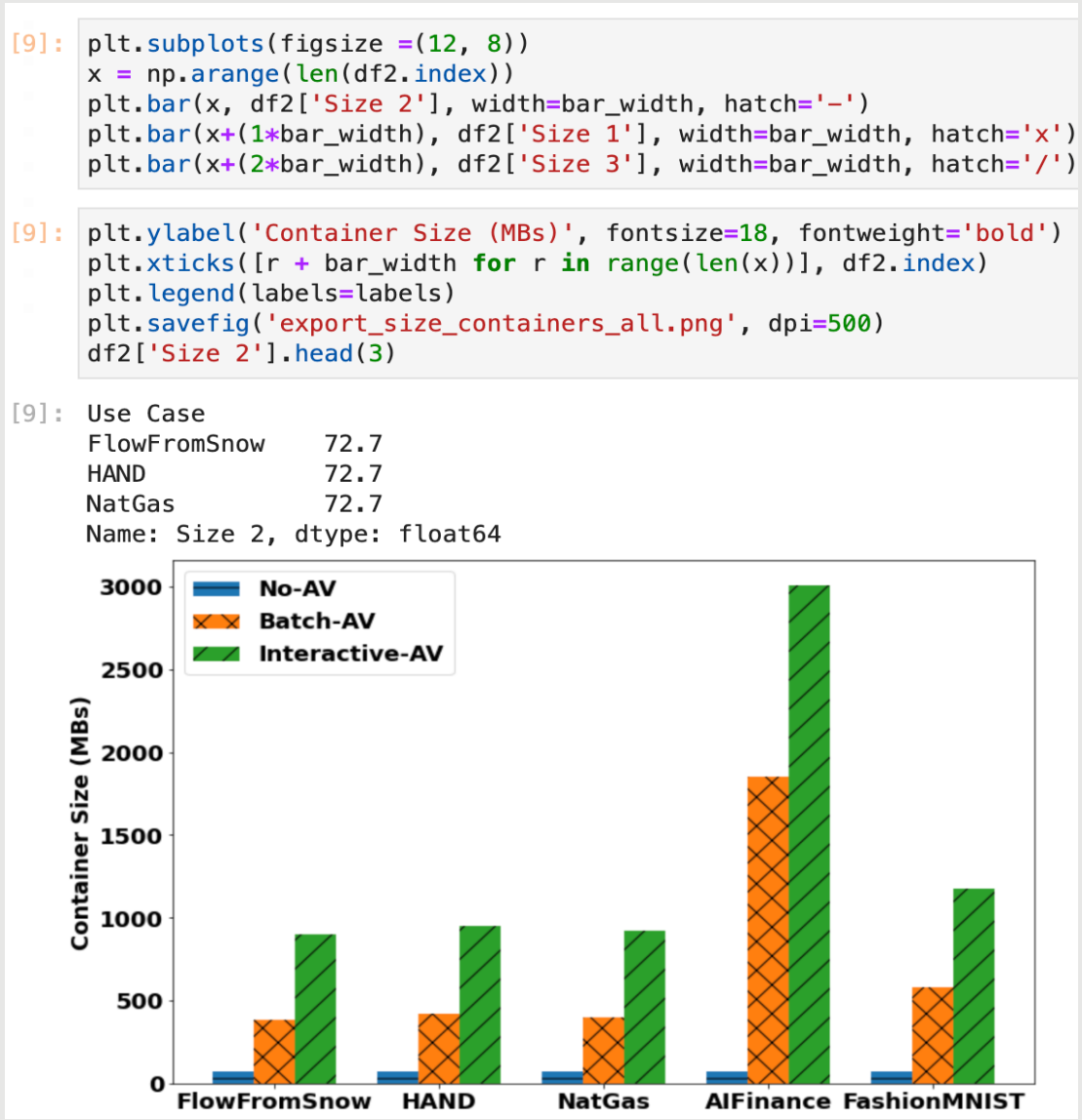
```
[9]: plt.subplots(figsize =(12, 8))
     x = np.arange(len(df2.index))
     plt.bar(x, df2['Size 2'], width=bar_width, hatch='-')
     plt.bar(x+(1*bar_width), df2['Size 1'], width=bar_width, hatch='x')
     plt.bar(x+(2*bar_width), df2['Size 3'], width=bar_width, hatch='/')

[9]: plt.ylabel('Container Size (MBs)', fontsize=18, fontweight='bold')
     plt.xticks([r + bar_width for r in range(len(x))], df2.index)
     plt.legend(labels=labels)
     plt.savefig('export_size_containers_all.png', dpi=500)
     df2['Size 2'].head(3)
```

Code Cells

```
[9]: Use Case
     FlowFromSnow    72.7
     HAND            72.7
     NatGas          72.7
     Name: Size 2, dtype: float64
```



Notebook is like a "Container"



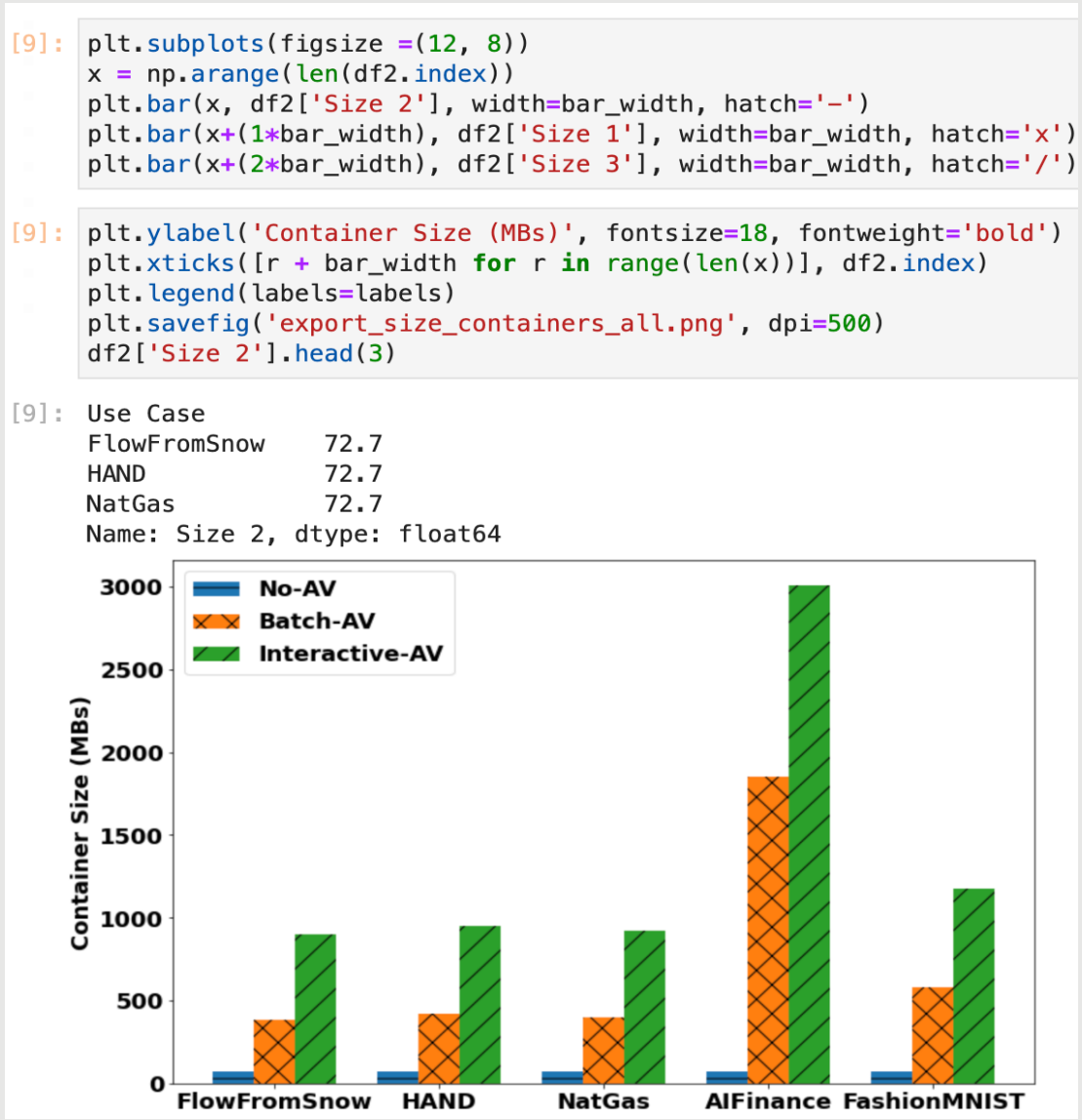
```
[9]: plt.subplots(figsize =(12, 8))
     x = np.arange(len(df2.index))
     plt.bar(x, df2['Size 2'], width=bar_width, hatch='-')
     plt.bar(x+(1*bar_width), df2['Size 1'], width=bar_width, hatch='x')
     plt.bar(x+(2*bar_width), df2['Size 3'], width=bar_width, hatch='/')

[9]: plt.ylabel('Container Size (MBs)', fontsize=18, fontweight='bold')
     plt.xticks([r + bar_width for r in range(len(x))], df2.index)
     plt.legend(labels=labels)
     plt.savefig('export_size_containers_all.png', dpi=500)
     df2['Size 2'].head(3)
```

Code Cells

```
[9]: Use Case
     FlowFromSnow    72.7
     HAND            72.7
     NatGas          72.7
     Name: Size 2, dtype: float64
```

Data Results

Notebook is like a "Container"



```
[9]: plt.subplots(figsize =(12, 8))
     x = np.arange(len(df2.index))
     plt.bar(x, df2['Size 2'], width=bar_width, hatch='-')
     plt.bar(x+(1*bar_width), df2['Size 1'], width=bar_width, hatch='x')
     plt.bar(x+(2*bar_width), df2['Size 3'], width=bar_width, hatch='/')

[9]: plt.ylabel('Container Size (MBs)', fontsize=18, fontweight='bold')
     plt.xticks([r + bar_width for r in range(len(x))], df2.index)
     plt.legend(labels=labels)
     plt.savefig('export_size_containers_all.png', dpi=500)
     df2['Size 2'].head(3)
```

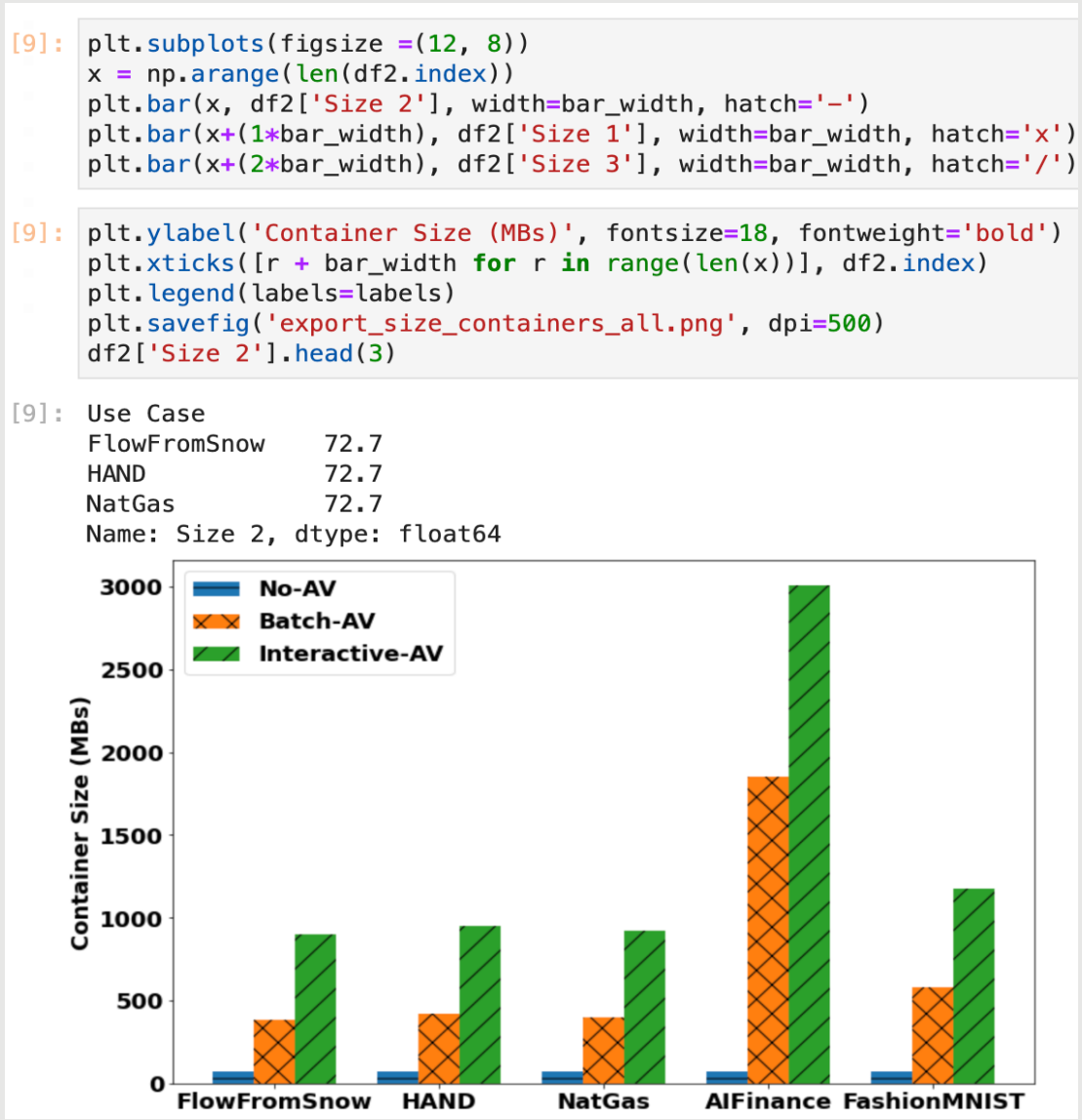
```
[9]: Use Case
     FlowFromSnow    72.7
     HAND            72.7
     NatGas          72.7
     Name: Size 2, dtype: float64
```


Code Cells

Data Results

Visualizations

Notebook is like a "Container"



```
[9]: plt.subplots(figsize =(12, 8))
     x = np.arange(len(df2.index))
     plt.bar(x, df2['Size 2'], width=bar_width, hatch='-')
     plt.bar(x+(1*bar_width), df2['Size 1'], width=bar_width, hatch='x')
     plt.bar(x+(2*bar_width), df2['Size 3'], width=bar_width, hatch='/')

[9]: plt.ylabel('Container Size (MBs)', fontsize=18, fontweight='bold')
     plt.xticks([r + bar_width for r in range(len(x))], df2.index)
     plt.legend(labels=labels)
     plt.savefig('export_size_containers_all.png', dpi=500)
     df2['Size 2'].head(3)
```

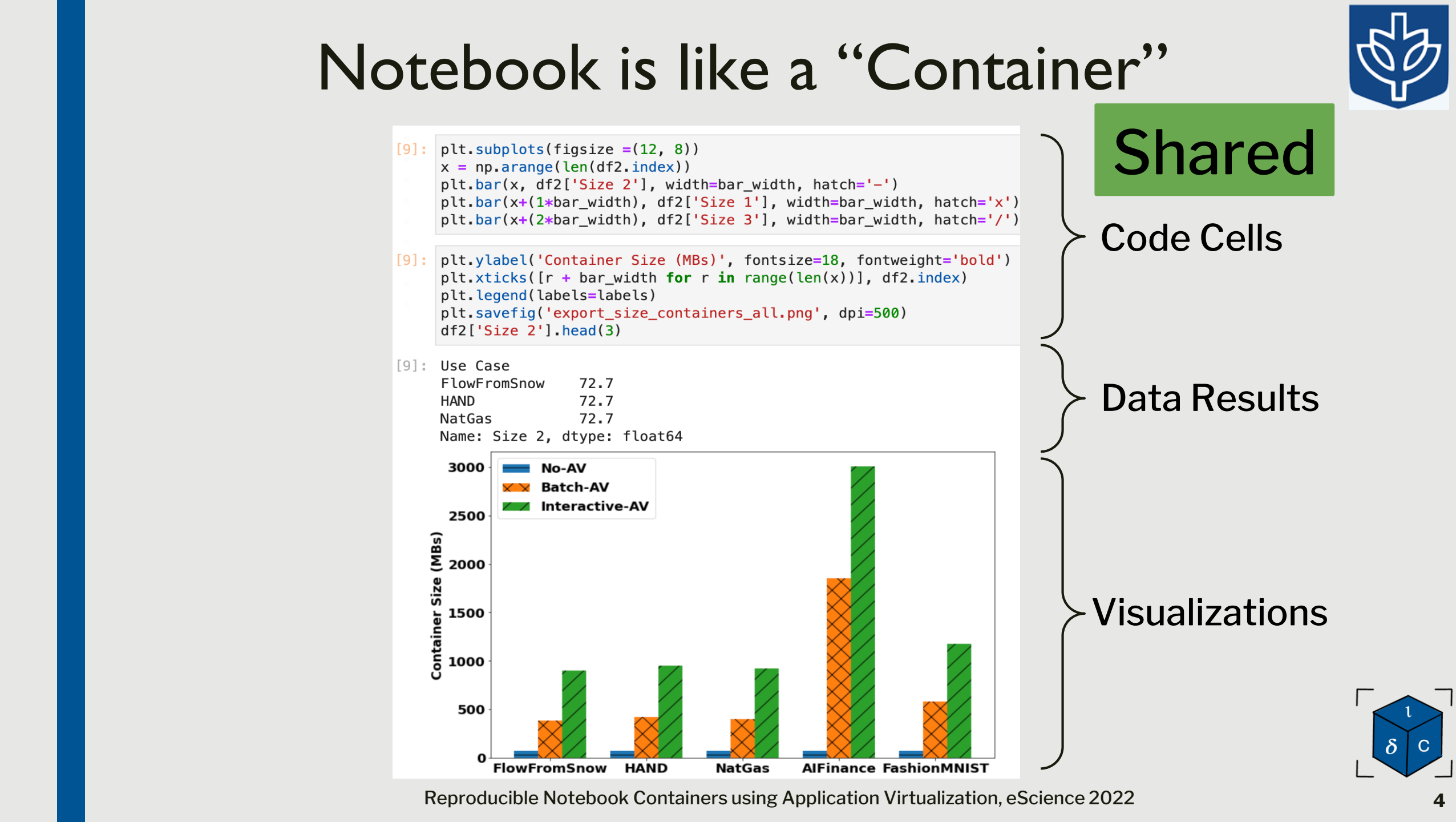
```
[9]: Use Case
     FlowFromSnow    72.7
     HAND            72.7
     NatGas          72.7
     Name: Size 2, dtype: float64
```


Code Cells

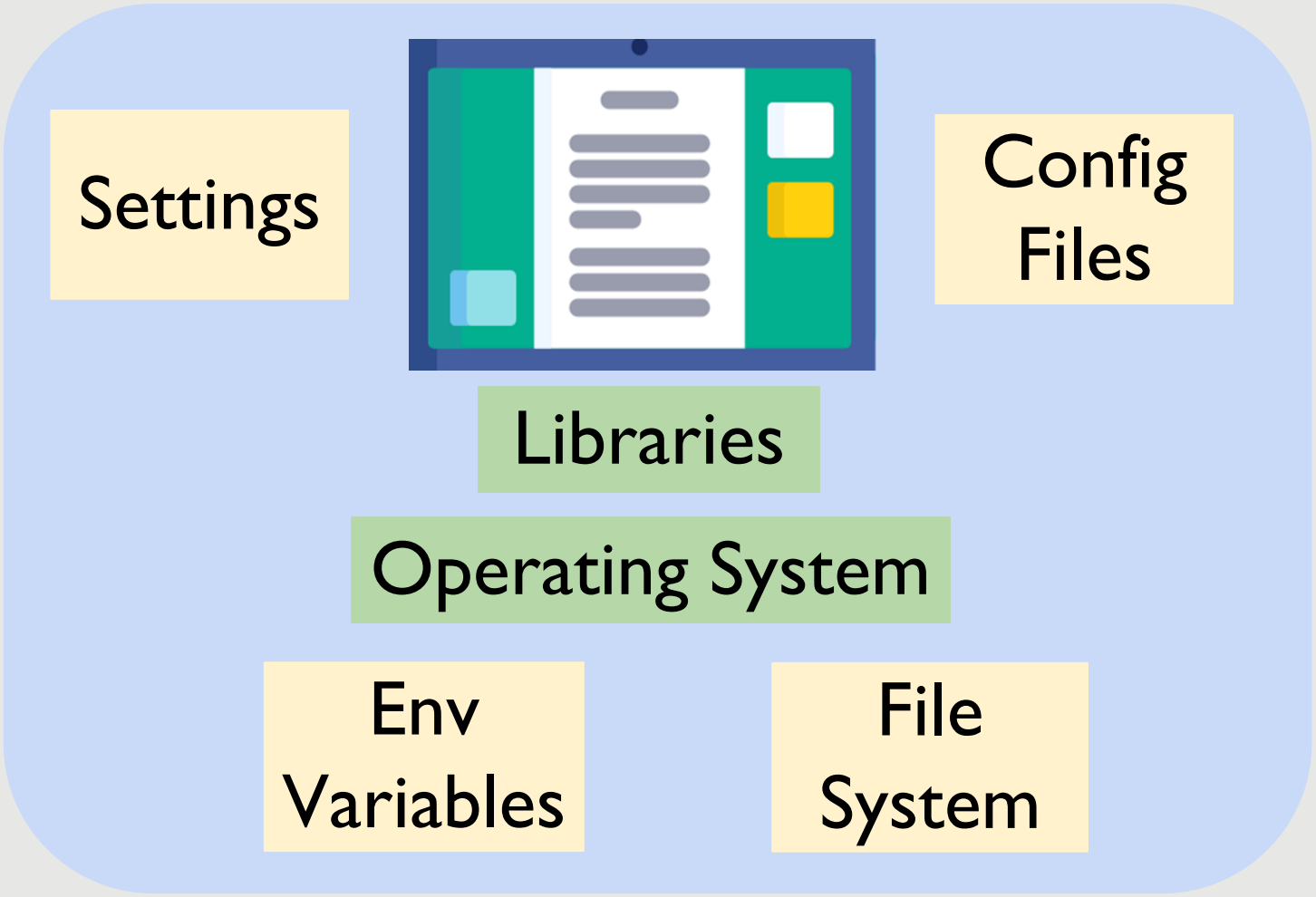
Data Results

Visualizations

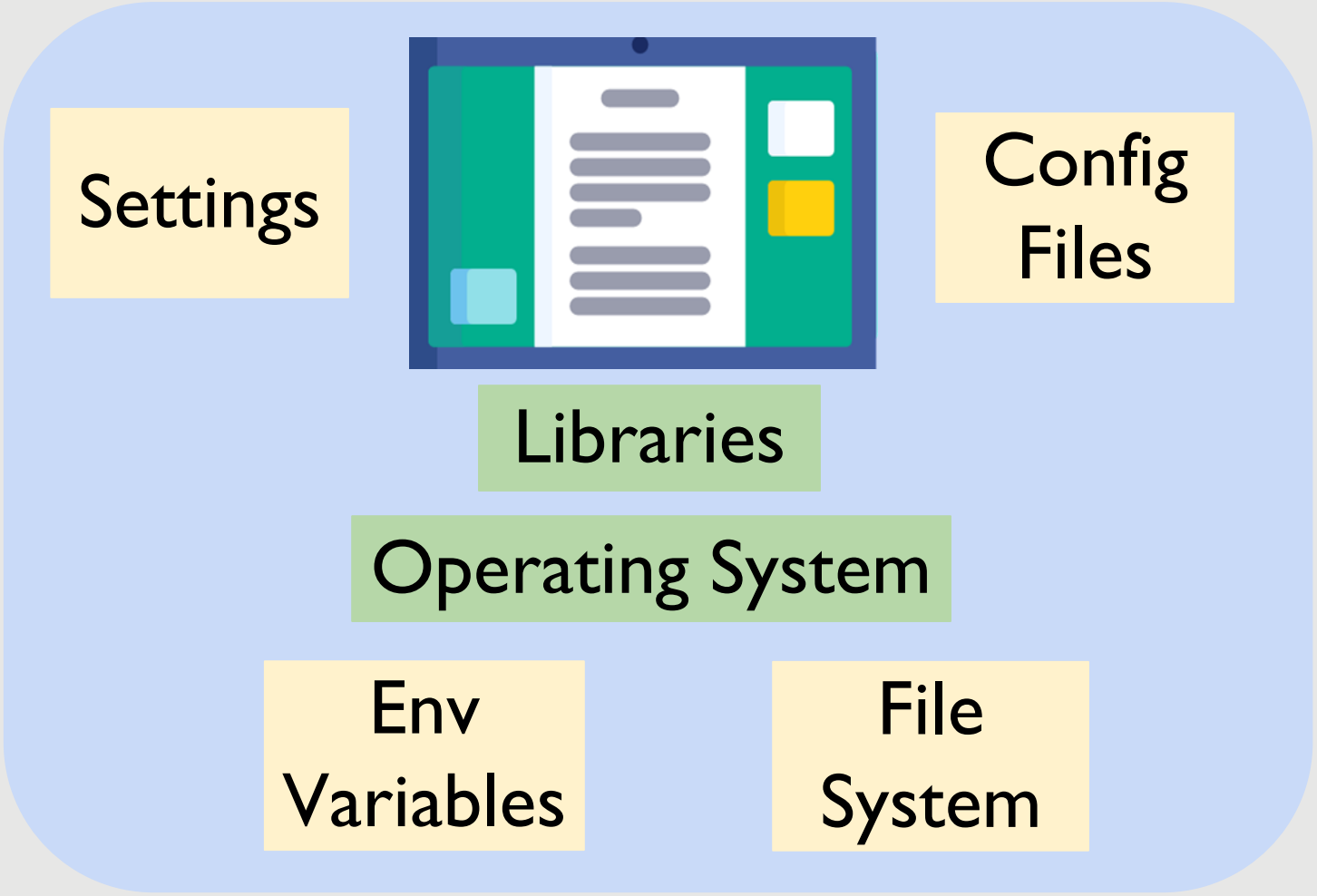
Notebook is like a "Container"



Notebook Environment

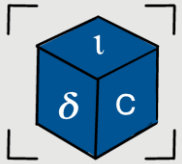


Notebook Environment

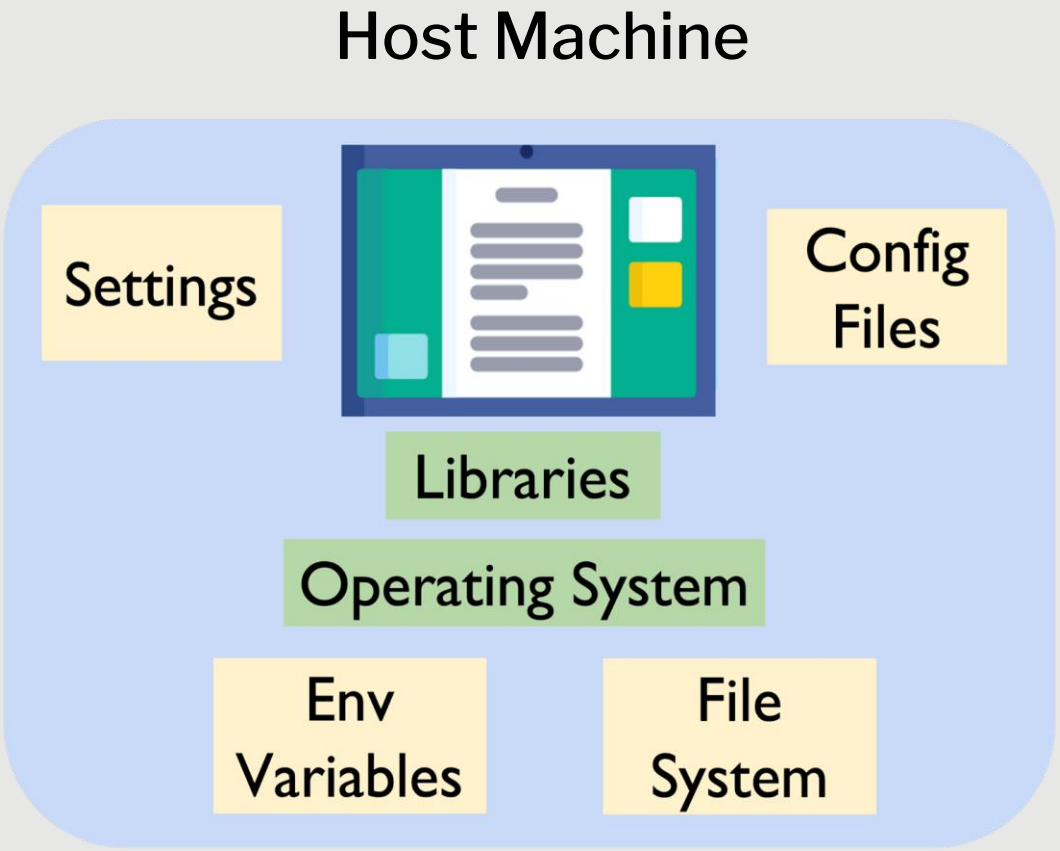


Not Shared

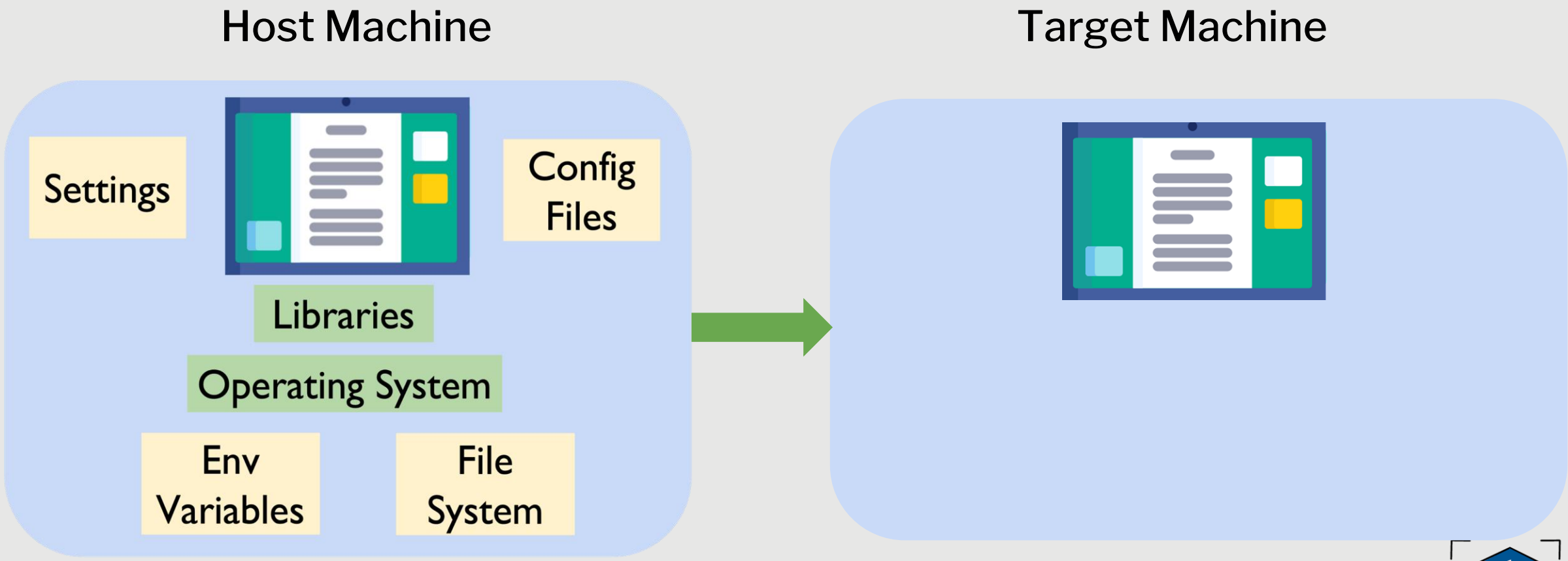
Notebook Environment is Not Shared



Notebook Environment is Not Shared



Notebook Environment is Not Shared



FLINC

- Creates lightweight notebook containers in an automated manner which are easy to use and share.
  - Preserves the interactive nature of notebooks
  - Guarantees reproducible execution
  - Supports flexibility to update and modify notebooks.

FLINC

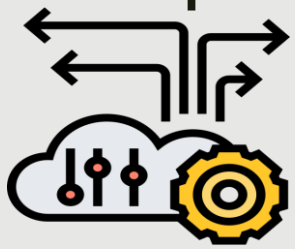
- Creates lightweight notebook containers in an automated manner which are easy to use and share.
  - Preserves the interactive nature of notebooks
  - Guarantees reproducible execution
  - Supports flexibility to update and modify notebooks.

Preserve Interactivity

FLINC

- Creates lightweight notebook containers in an automated manner which are easy to use and share.
  - Preserves the interactive nature of notebooks
  - Guarantees reproducible execution
  - Supports flexibility to update and modify notebooks.

Preserve Interactivity



Flexible for Modifications

FLINC

- Creates lightweight notebook containers in an automated manner which are easy to use and share.
  - Preserves the interactive nature of notebooks
  - Guarantees reproducible execution
  - Supports flexibility to update and modify notebooks.

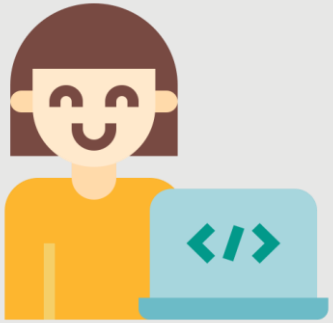
Preserve Interactivity

Flexible for Modifications

Efficient and Easy to Use

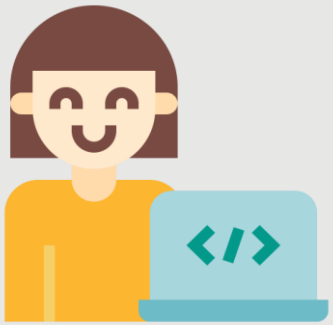
Notebook Reproducibility

Notebook Reproducibility



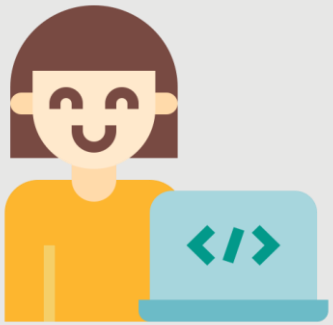
Alice

Notebook Reproducibility



Alice

Notebook Reproducibility

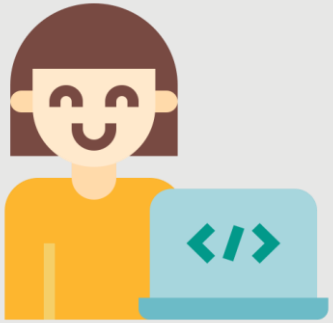


Alice

execute

modify

Notebook Reproducibility



Alice

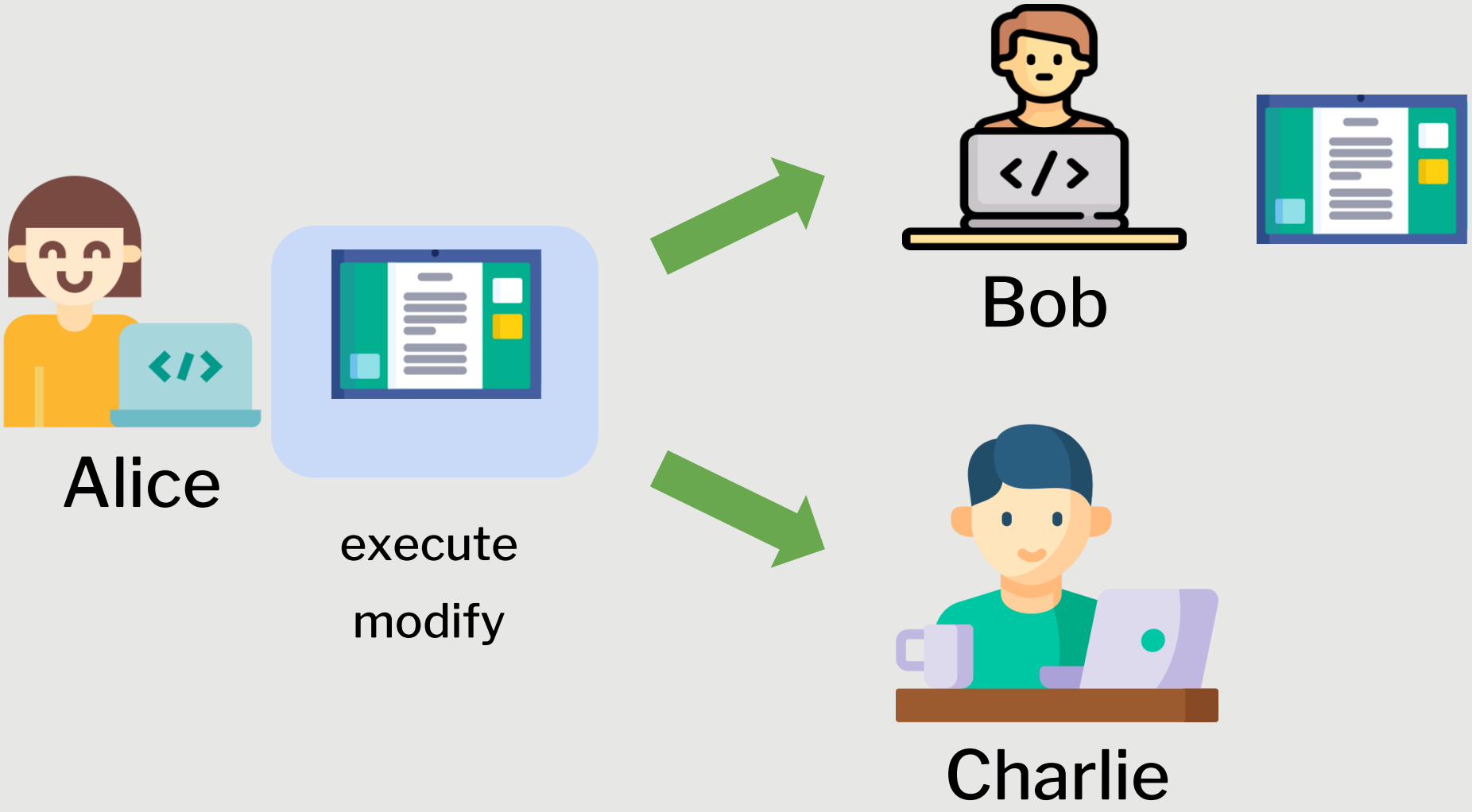
execute

modify

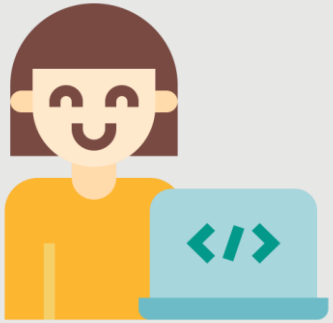
Bob

Charlie

Notebook Reproducibility



Notebook Reproducibility



Alice

execute

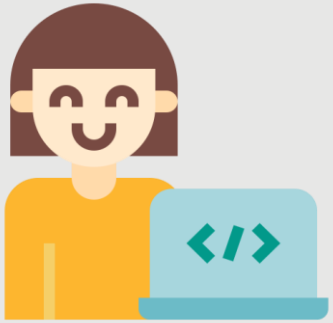
modify

Bob

modify

Charlie

Notebook Reproducibility



Alice

execute

modify

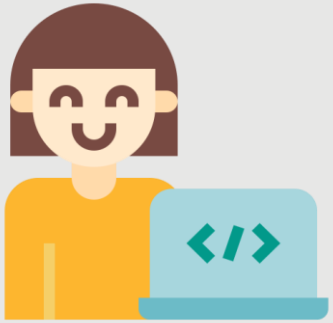
Bob

~~execute~~

modify

Charlie

Notebook Reproducibility



Alice

execute

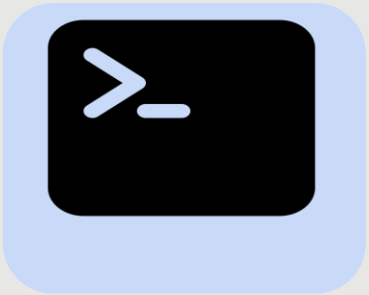
modify

Bob

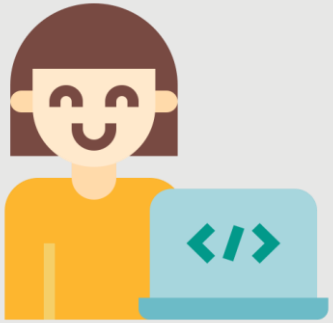
~~execute~~

modify

Charlie



Notebook Reproducibility



Alice

execute

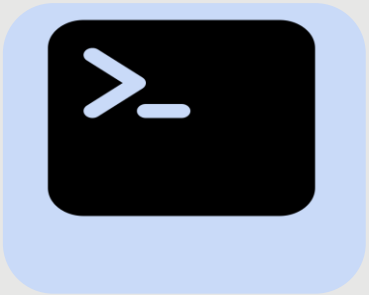
modify

Bob

~~execute~~

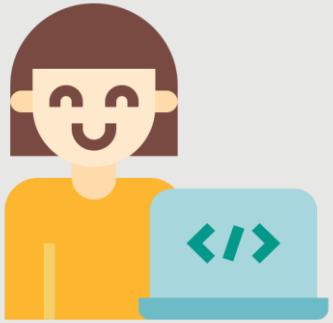
modify

Charlie



~~modify~~

Notebook Reproducibility



Alice

execute

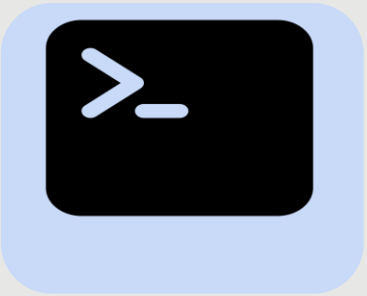
modify

Bob

~~execute~~

modify

Charlie



execute

~~modify~~

Current Solutions for Reproducibility

- Document the entire environment.

Current Solutions for Reproducibility

- Document the entire environment.

Problem

- Difficult and time consuming task.
- Increased complexity with multiple languages and custom libraries.

- Create a container image with all dependencies.

Current Solutions for Reproducibility

- Document the entire environment.

Problem

- Difficult and time consuming task.
- Increased complexity with multiple languages and custom libraries.

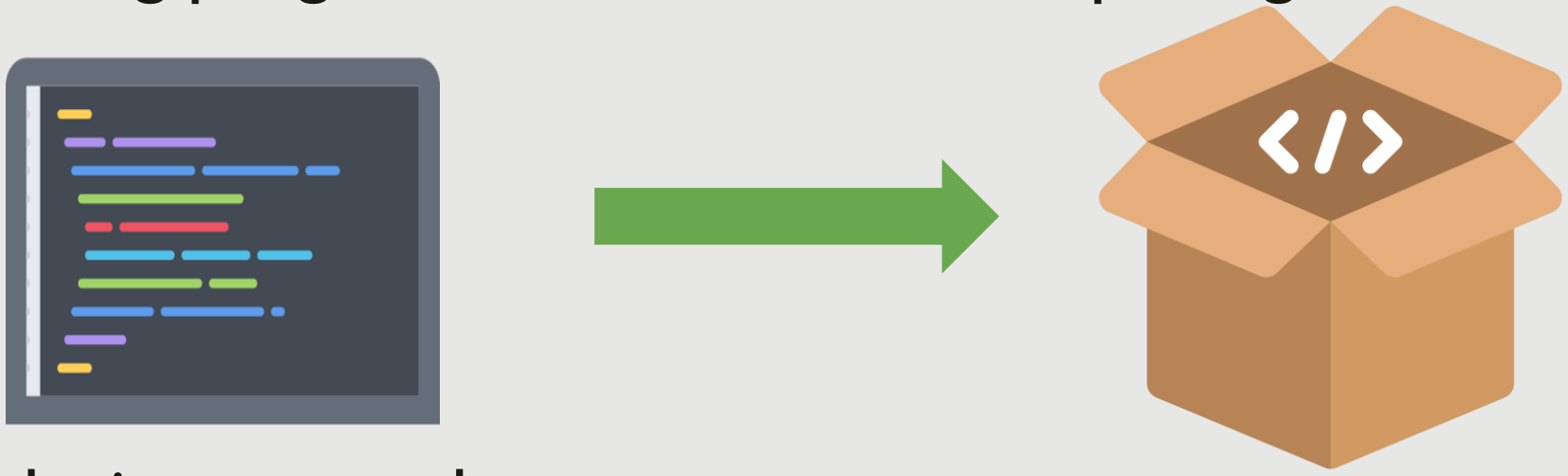
- Create a container image with all dependencies.

Problem

- Manual and time consuming.
- Requires technical knowledge.
- Conflict with existing environment.

Another Solution: Application Virtualization

- Application Virtualization(AV) is a method to convert executing programs into self-contained packages.



- Works in two modes:
  - Audit Mode - capture and encapsulate execution
  - Repeat Mode - repeat the captured execution

Limitations of AV

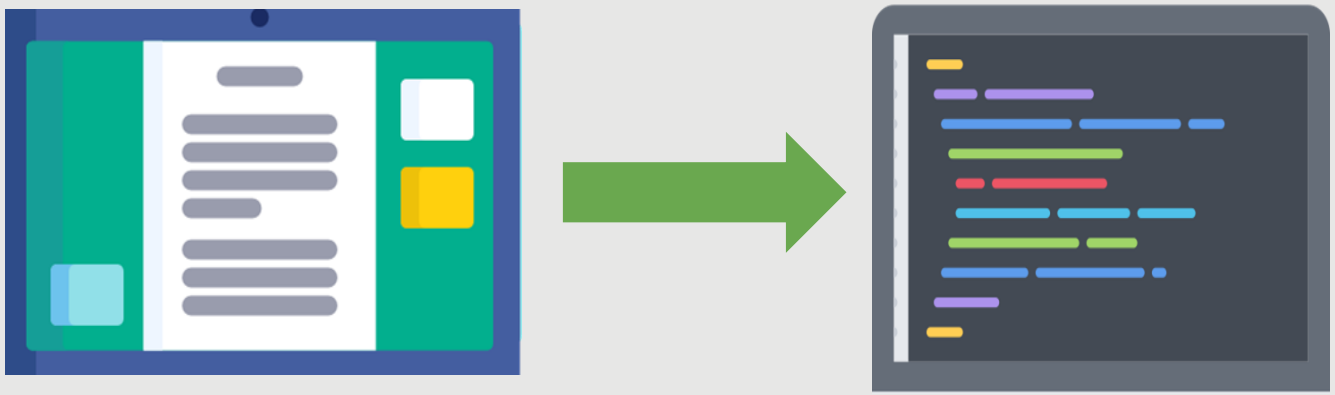
- Most AV systems assume batch workflows.
  - Not suited to capturing interactive notebooks.
- Notebooks need to be converted to batch scripts

Limitations of AV

- Most AV systems assume batch workflows.
  - Not suited to capturing interactive notebooks.
- Notebooks need to be converted to batch scripts

Limitations of AV

- Most AV systems assume batch workflows.
  - Not suited to capturing interactive notebooks.
- Notebooks need to be converted to batch scripts



Limitations of AV

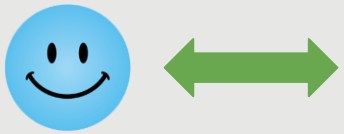
- Most AV systems assume batch workflows.
  - Not suited to capturing interactive notebooks.
- Notebooks need to be converted to batch scripts

Notebook Client Server Architecture

- Extends application virtualization(AV) to the client-server architecture of notebooks.

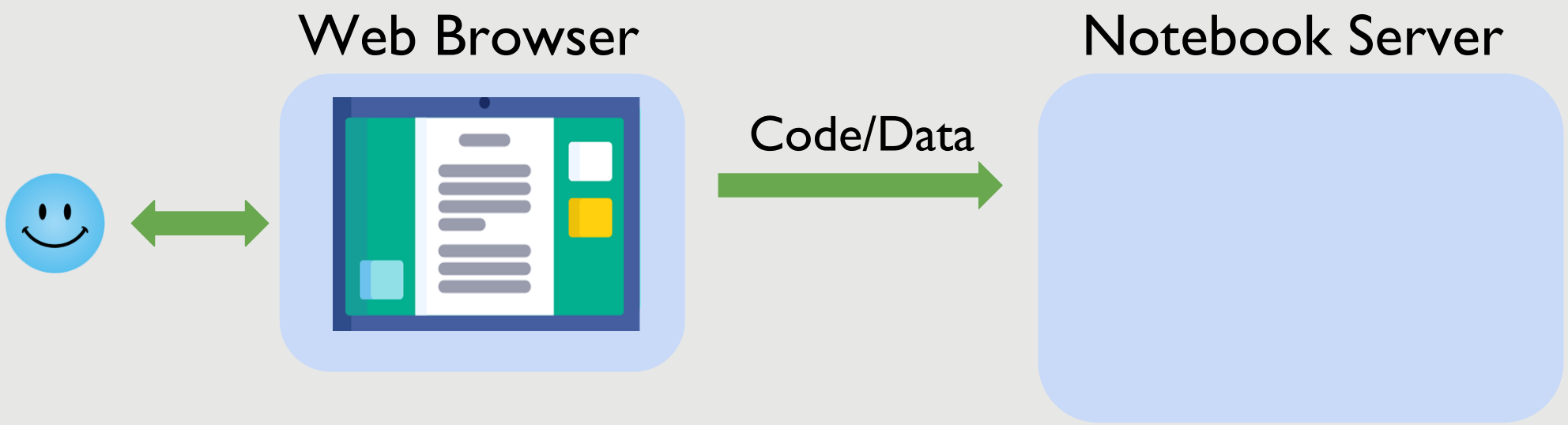
Notebook Client Server Architecture

- Extends application virtualization(AV) to the client-server architecture of notebooks.



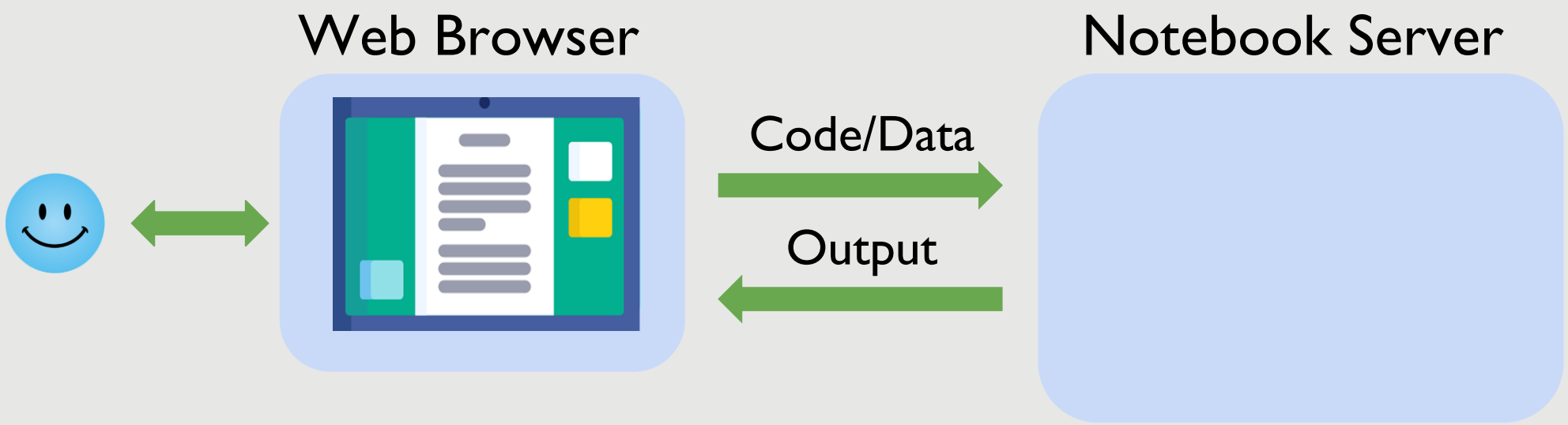
Notebook Client Server Architecture

- Extends application virtualization(AV) to the client-server architecture of notebooks.



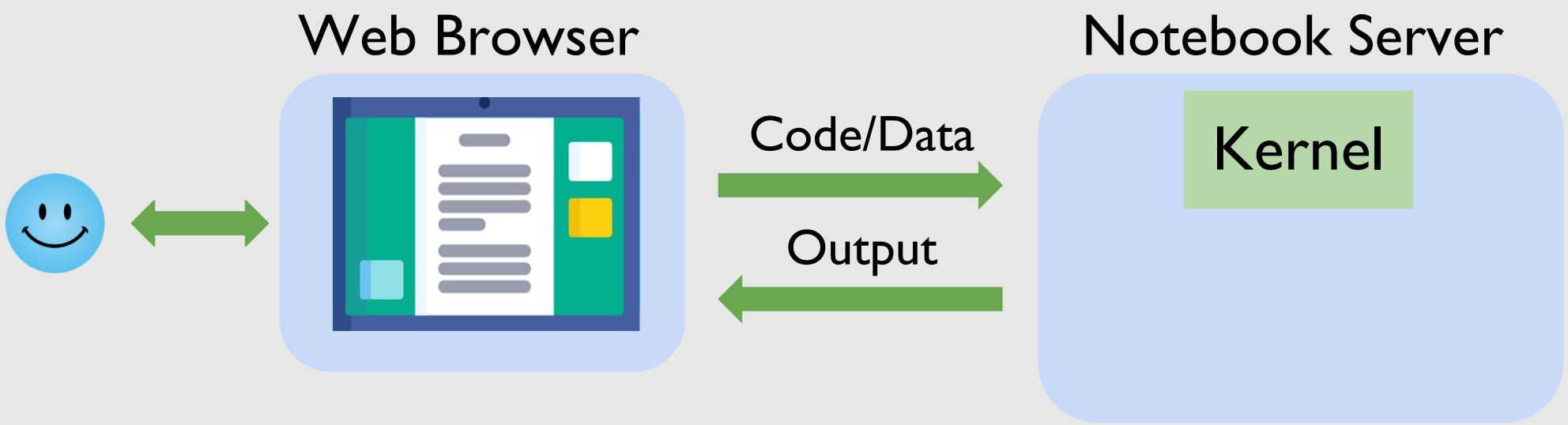
Notebook Client Server Architecture

- Extends application virtualization(AV) to the client-server architecture of notebooks.



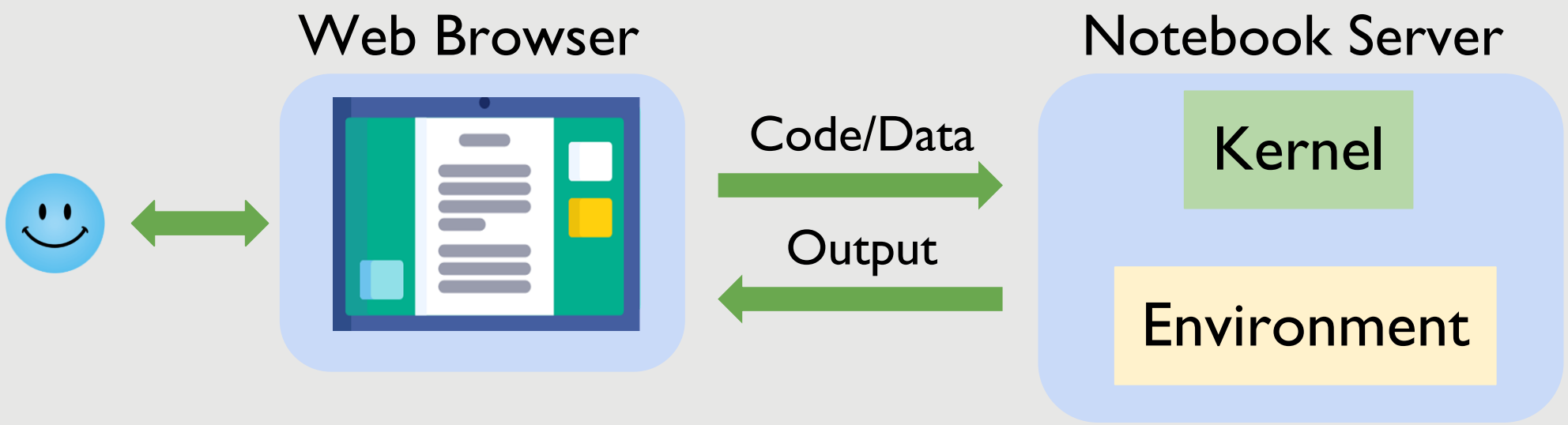
Notebook Client Server Architecture

- Extends application virtualization(AV) to the client-server architecture of notebooks.



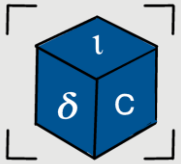
Notebook Client Server Architecture

- Extends application virtualization(AV) to the client-server architecture of notebooks.



Extend AV to Notebook Client-Server Architecture

- Creates two additional kernels:
  - *Audit kernel - executes and audits notebook code*
  - *Repeat kernel – repeats the containerized code*

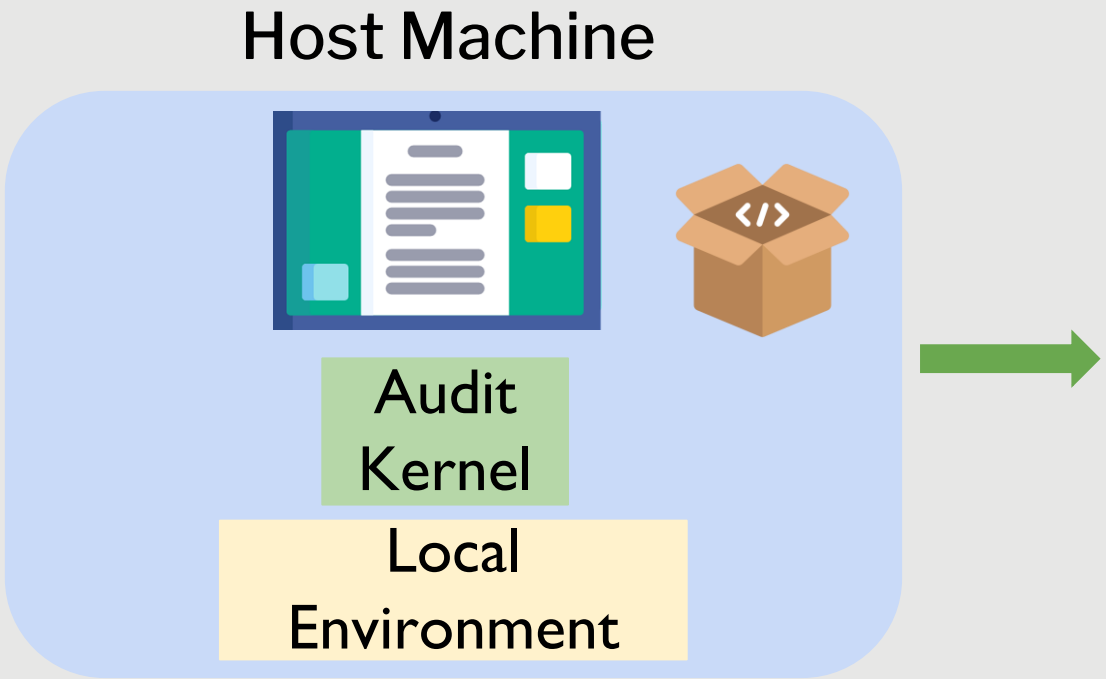


Extend AV to Notebook Client-Server Architecture

- Creates two additional kernels:
  - *Audit kernel - executes and audits notebook code*
  - *Repeat kernel – repeats the containerized code*

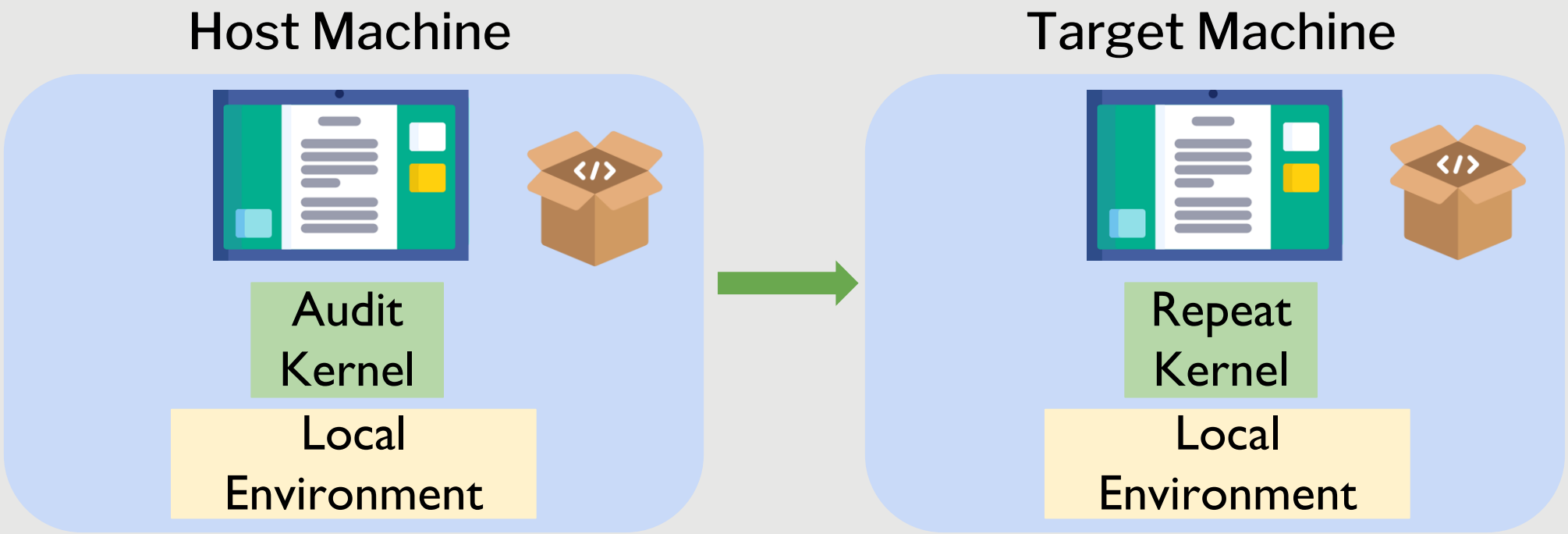
Extend AV to Notebook Client-Server Architecture

- Creates two additional kernels:
  - *Audit kernel - executes and audits notebook code*
  - *Repeat kernel – repeats the containerized code*



Extend AV to Notebook Client-Server Architecture

- Creates two additional kernels:
  - *Audit kernel - executes and audits notebook code*
  - *Repeat kernel – repeats the containerized code*



But, is Extending AV Enough? NO!

Bob

~~execute~~
modify

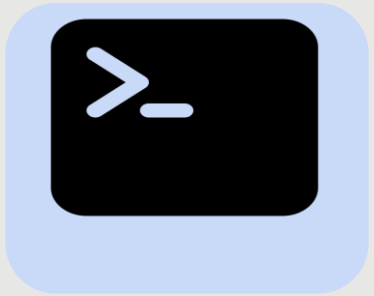
1. No Guarantee of Reproducible Execution
   - Notebook and container are separately shared

But, is Extending AV Enough? NO!

Bob

~~execute~~
modify

Charlie



execute
~~modify~~

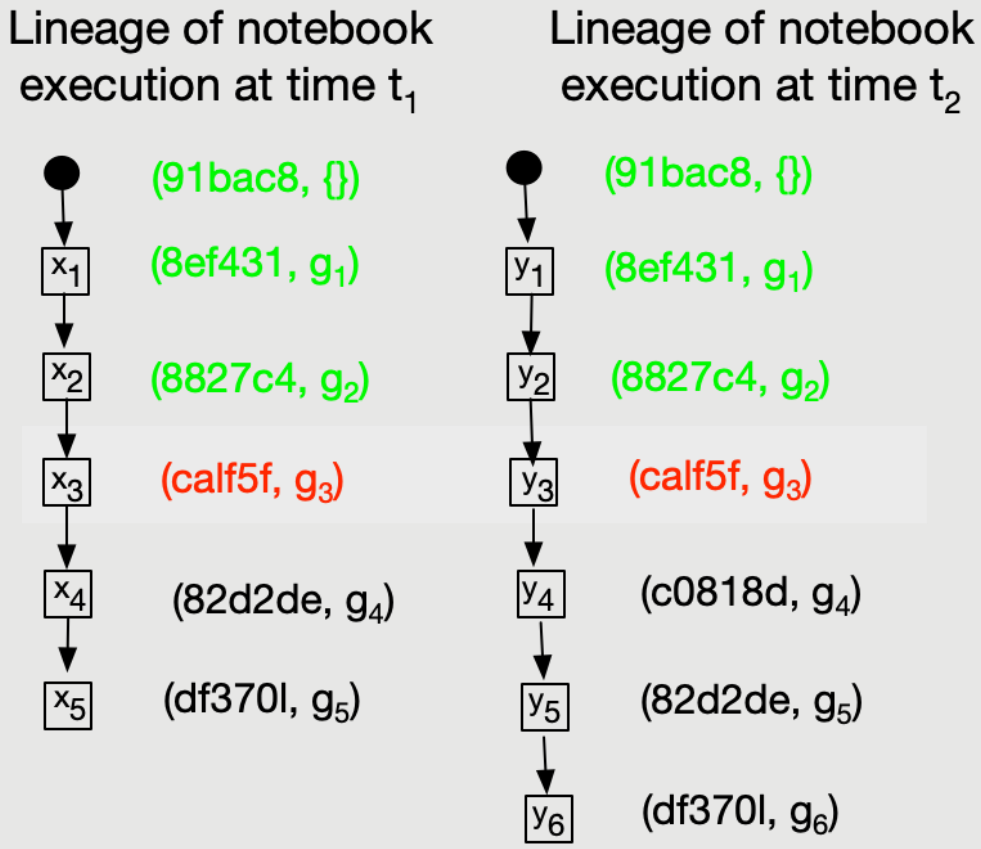
1. No Guarantee of Reproducible Execution
   - Notebook and container are separately shared

2. No Flexibility with Reproducible Execution
   a. No way to export dependencies

Ensuring Reproducible Execution

- **Key Idea: Lineage-based notebook cell equivalence**
- Compare *per cell* lineage during audit time with repeat time
  - Cell lineage = cell code hashes and state lineage
  - State lineage =
    - the predecessor cell's lineage
    - the sequence of system events triggered by program instructions in the cell, and
    - the hashes of the associated external data dependencies.

Lineage Equivalence



E_3 in L_1	E_3 in L_2
4183 fork 4184 4184 exec '/usr/torch/dataloader.py' 4184 open '/home/alice/new_fashion' 4183 mem 567894567 4184 read b2e1772	6254 fork 6255 6255 exec '/usr/torch/dataloader.py' 6255 open '/home/alice/new_fashion' 6255 read 6789b34 6254 mem 589715363

Given two notebook executions E_1 and E_2, cell state ps_i in E_1 is equal to cell state ps_j in E_2, denoted $ps_i = ps_j$, iff

(i) $h_i = h_j$, and

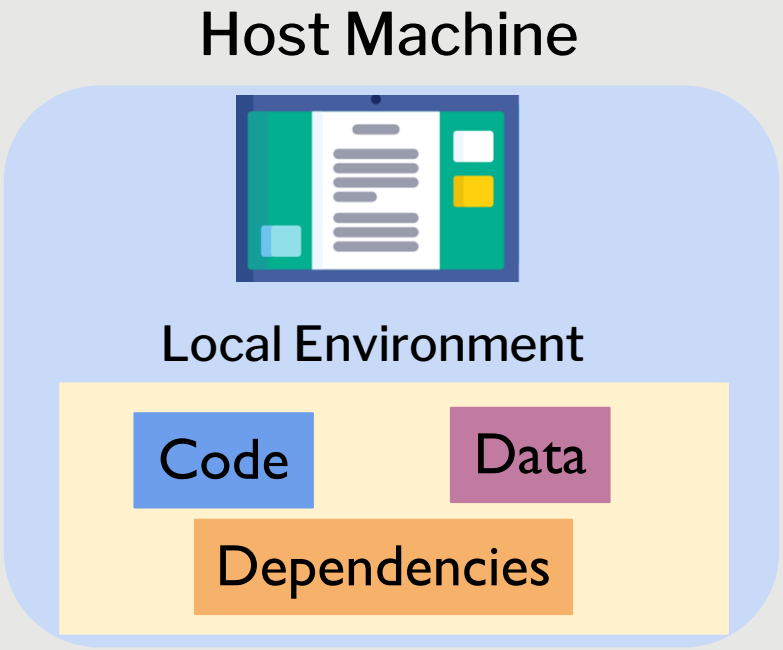
(ii) $g_i = g_j$

Flexible Reproducibility

Flexible Reproducibility

Export
Functionality

Flexible Reproducibility



Export
Functionality

Flexible Reproducibility

Host Machine

Local Environment

Code

Data

Dependencies

Export Functionality

Generates

Provenance Log

```
CLOSE /lib/x86_64-linux-gnu/libpthread.so.0
EXECVE 28567 /data/software/GLib/2.62.0-GCCcore-8.3.
EXECVE 28567 /data/software/GDAL/3.0.2-foss-2019b/bi
EXECVE 28567 /data/software/libiconv/1.16-GCCcore-8.
EXECVE 28567 /data/software/netCDF-C++4/4.3.1-gompi-
EXECVE 28567 /data/software/libdap/3.20.6-GCCcore-8.
EXECVE 28567 /data/software/netCDF-Fortran/4.5.2-gom
EXECVE 28567 /data/software/GRASS/7.8.3-foss-2019b/b
EXECVE 28567 /data/software/Java/11.0.2/ip /data/con
READ /data/lib/python3.7/site-packages/matplotlib-3.
CLOSE /data/lib/python3.7/site-packages/rasterio-1.2
READ /data/lib/python3.7/site-packages/xarray-0.16.2
READ /data/lib/python3.7/site-packages/numpy-1.20.1.
READ /data/lib/python3.7/site-packages/geopandas-0.1
READ /data/lib/python3.7/site-packages/pandas-1.2.2.
```

Flexible Reproducibility

Host Machine

Local Environment

Code

Data

Dependencies

Export Functionality

Generates

Provenance Log

```
CLOSE /lib/x86_64-linux-gnu/libpthread.so.0
EXECVE 28567 /data/software/GLib/2.62.0-GCCcore-8.3.
EXECVE 28567 /data/software/GDAL/3.0.2-foss-2019b/bi
EXECVE 28567 /data/software/libiconv/1.16-GCCcore-8.
EXECVE 28567 /data/software/netCDF-C++4/4.3.1-gompi-
EXECVE 28567 /data/software/libdap/3.20.6-GCCcore-8.
EXECVE 28567 /data/software/netCDF-Fortran/4.5.2-gom
EXECVE 28567 /data/software/GRASS/7.8.3-foss-2019b/b
EXECVE 28567 /data/software/Java/11.0.2/ip /data/con
READ /data/lib/python3.7/site-packages/matplotlib-3.
CLOSE /data/lib/python3.7/site-packages/rasterio-1.2
READ /data/lib/python3.7/site-packages/xarray-0.16.2
READ /data/lib/python3.7/site-packages/numpy-1.20.1.
READ /data/lib/python3.7/site-packages/geopandas-0.1
READ /data/lib/python3.7/site-packages/pandas-1.2.2.
```

Extract

```
numpy==1.19.1
memory_profiler==0.57.0
mccabe==0.6.1
pyzmq==19.0.1
pyepsg==0.4.0
decorator==4.4.2
pycparser==2.20
astroid==2.4.2
sklearn==0.0
pysumma==3.0.0
MarkupSafe==1.1.1
PyYAML==5.3.1
```

Notebook Dependencies

Flexible Reproducibility

Host Machine

Local Environment

Code

Data

Dependencies

Export Functionality

Target Machine

Local Env

Virtual Environment

Code

Data

Dependencies

Generates

Installs

Provenance Log

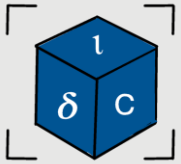
```
CLOSE /lib/x86_64-linux-gnu/libpthread.so.0
EXECVE 28567 /data/software/GLib/2.62.0-GCCcore-8.3.
EXECVE 28567 /data/software/GDAL/3.0.2-foss-2019b/bi
EXECVE 28567 /data/software/libiconv/1.16-GCCcore-8.
EXECVE 28567 /data/software/netCDF-C++4/4.3.1-gompi-
EXECVE 28567 /data/software/libdap/3.20.6-GCCcore-8.
EXECVE 28567 /data/software/netCDF-Fortran/4.5.2-gom
EXECVE 28567 /data/software/GRASS/7.8.3-foss-2019b/b
EXECVE 28567 /data/software/Java/11.0.2/ip /data/con
READ /data/lib/python3.7/site-packages/matplotlib-3.
CLOSE /data/lib/python3.7/site-packages/rasterio-1.2
READ /data/lib/python3.7/site-packages/xarray-0.16.2
READ /data/lib/python3.7/site-packages/numpy-1.20.1.
READ /data/lib/python3.7/site-packages/geopandas-0.1
READ /data/lib/python3.7/site-packages/pandas-1.2.2.
```

Extract

```
numpy==1.19.1
memory_profiler==0.57.0
mccabe==0.6.1
pyzmq==19.0.1
pyepsg==0.4.0
decorator==4.4.2
pycparser==2.20
astroid==2.4.2
sklearn==0.0
pysumma==3.0.0
MarkupSafe==1.1.1
PyYAML==5.3.1
```

Notebook Dependencies

Experiments



Experiments

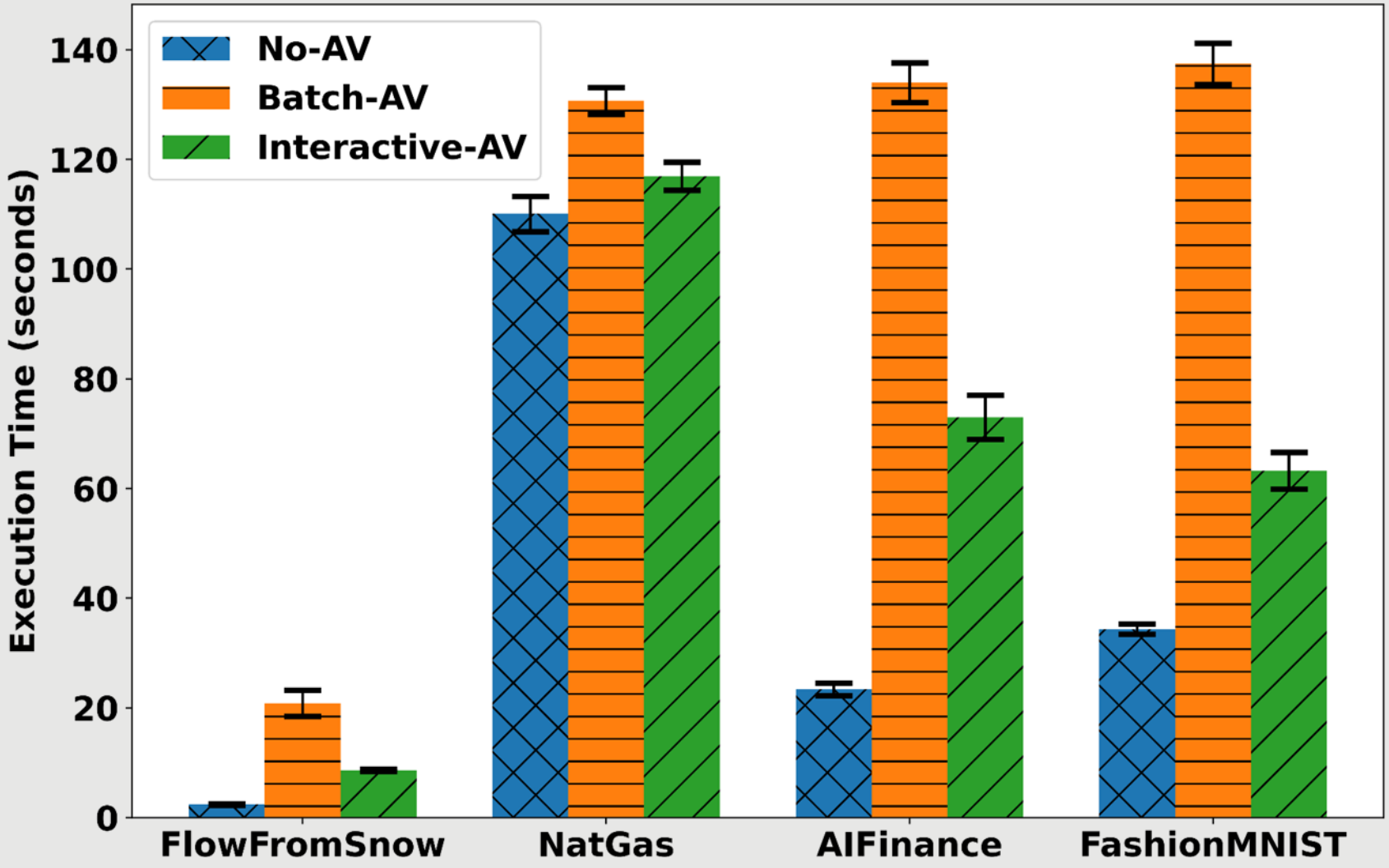
- Analyze storage and running time for AV and export.
- Experiment using 4 Python notebooks.
- Compare three methods for AV performance.

  1. Interactive-AV: FLINC notebook auditing

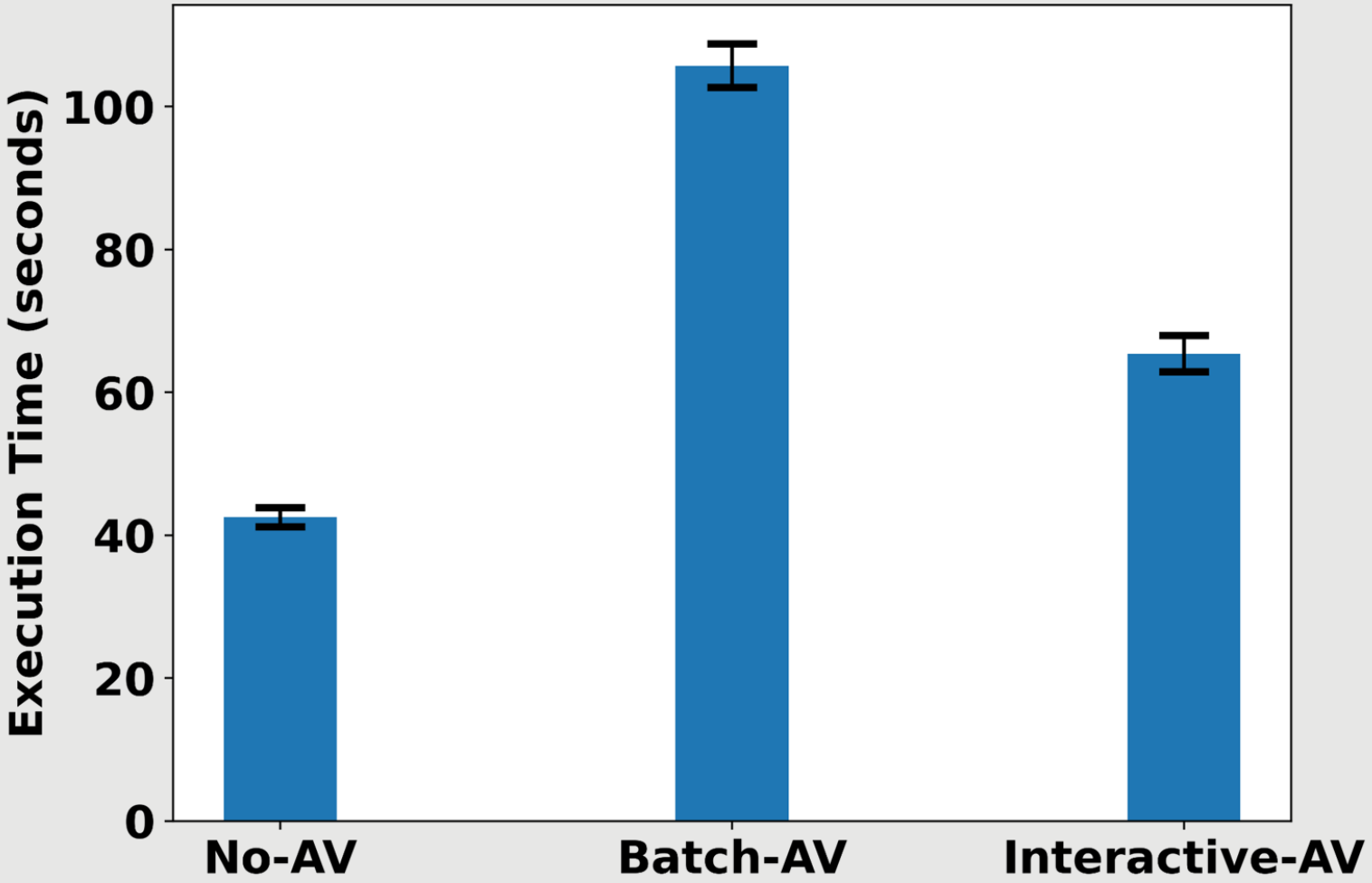
  2. Batch-AV: Python file of notebook audited using AV

  3. Baseline: No-AV: Regular notebook execution

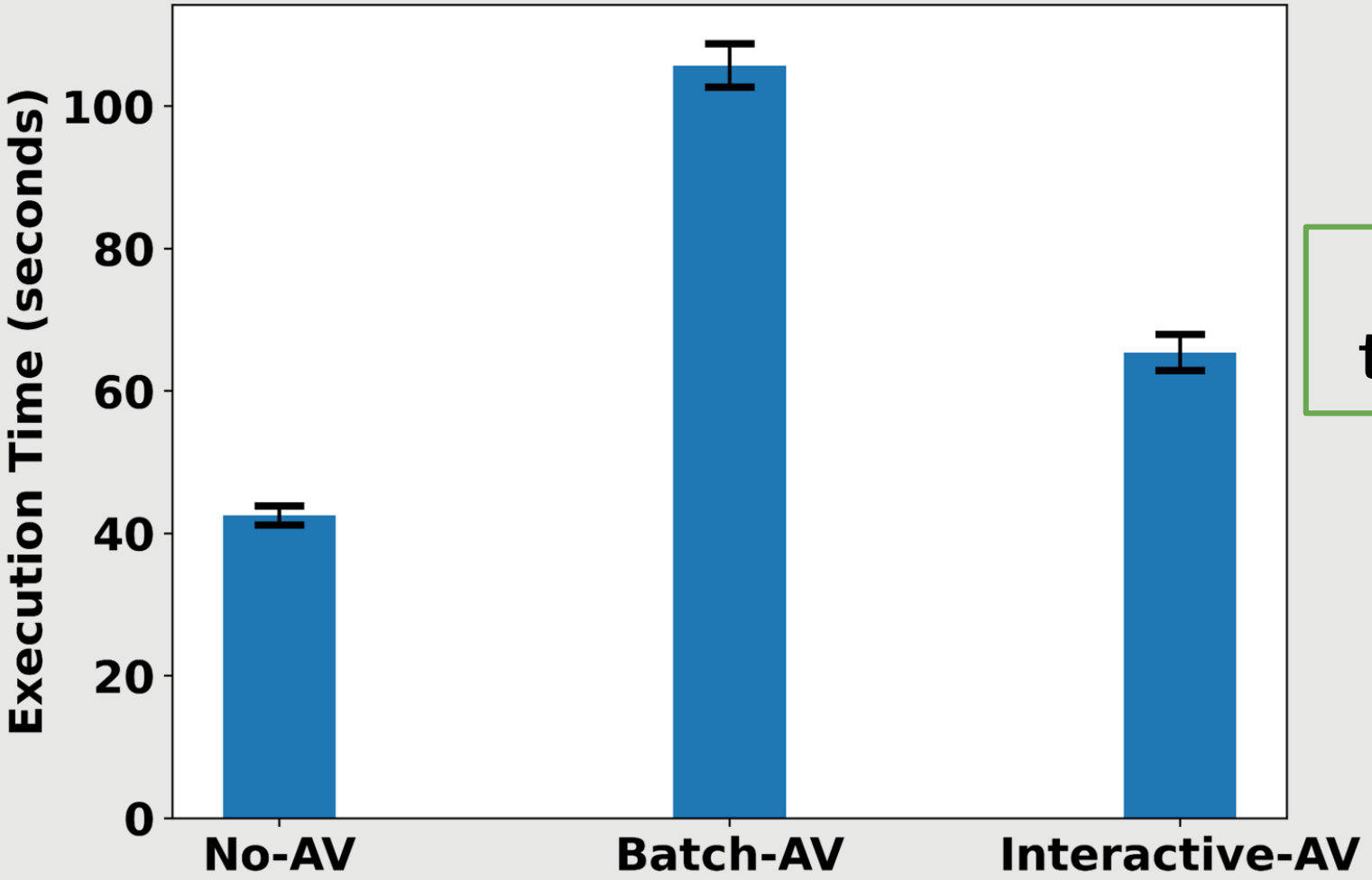
Notebook Execution Time



Average Execution Time

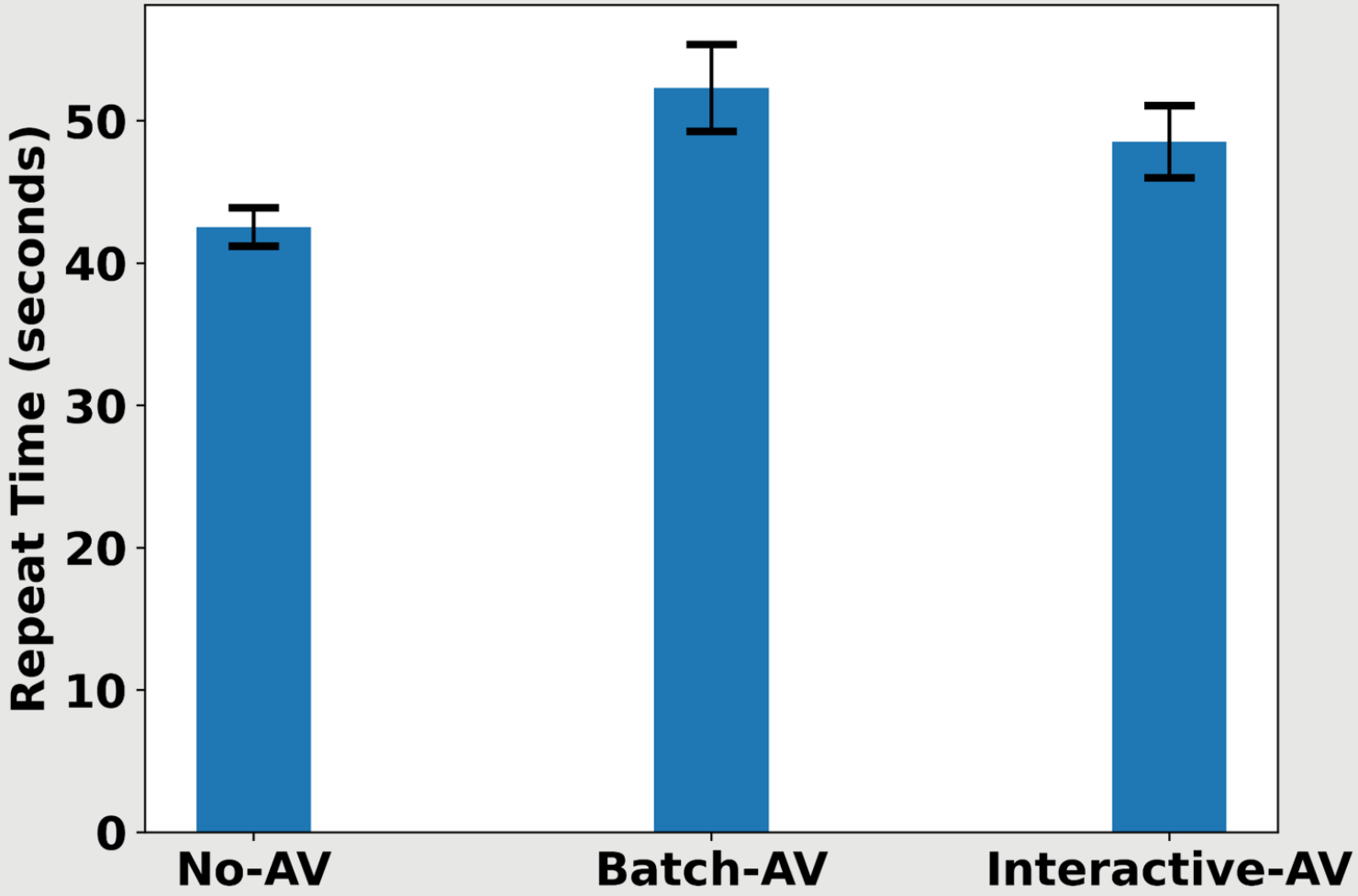


Average Execution Time



38% less than Batch-AV

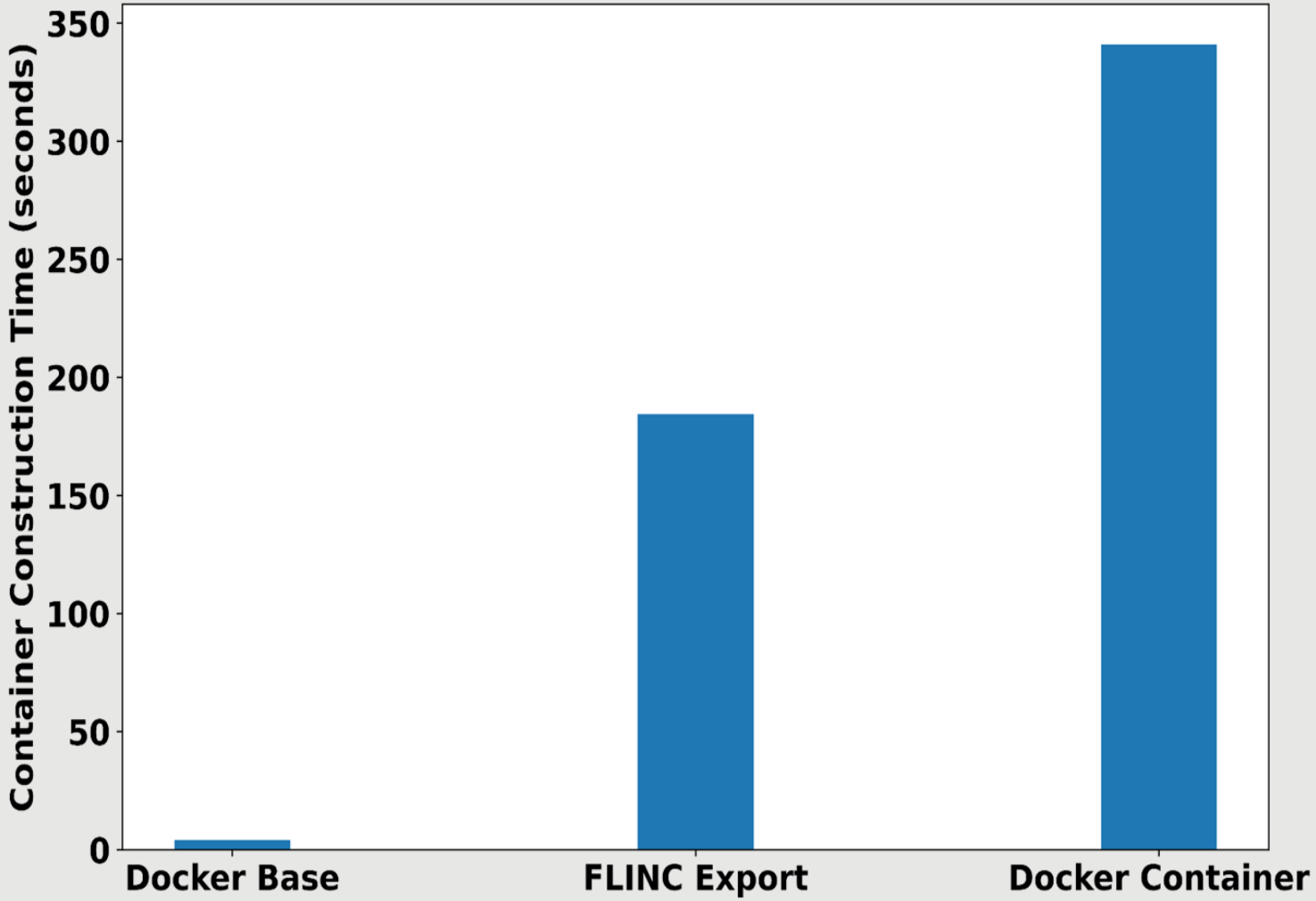
Average Repeat Time



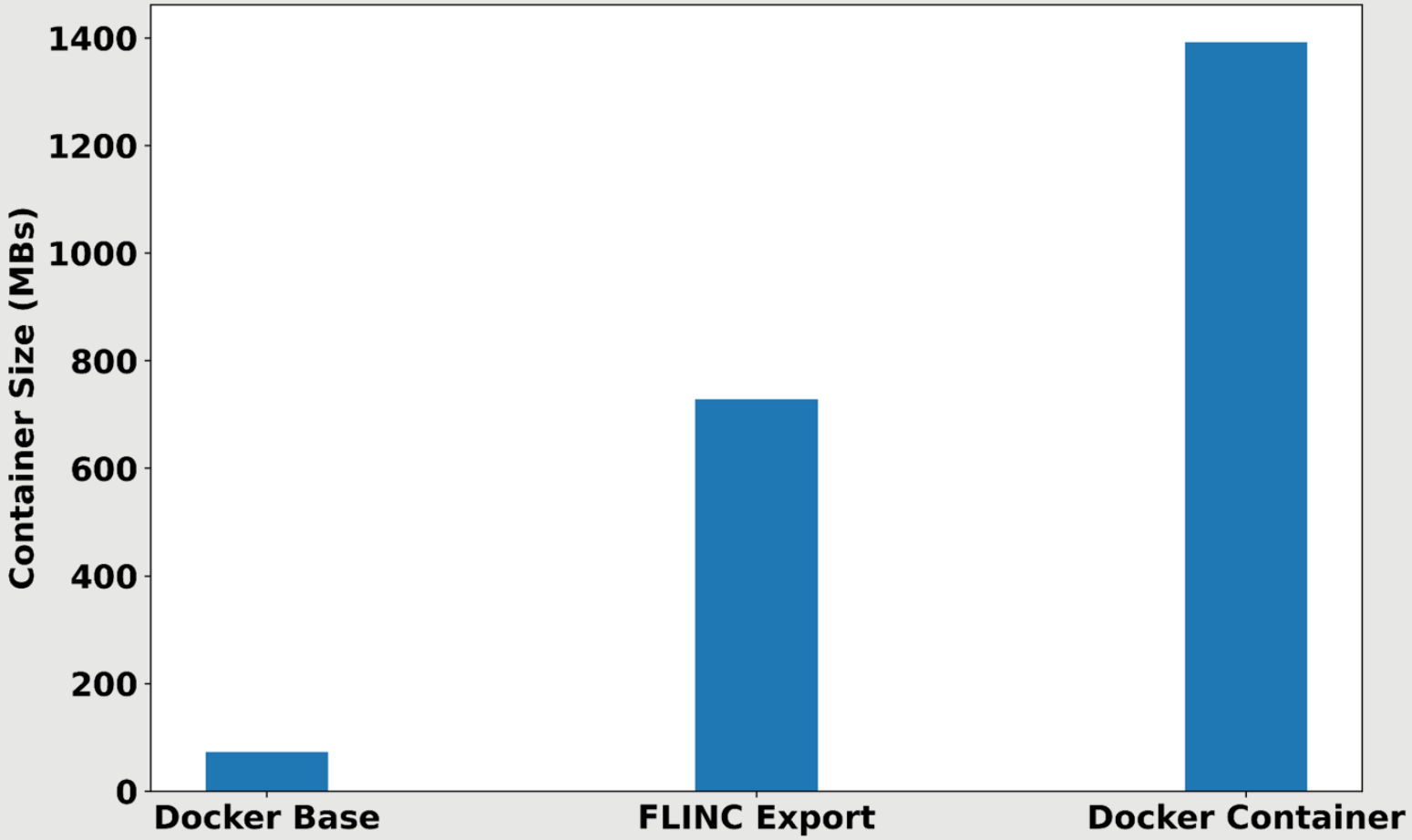
FLINC Export Overhead

- Compare three methods for export functionality.
  1. FLINC Export
     - FLINC virtual environment with dependencies
  2. Docker Container
     - Docker container with same dependencies
  3. Docker Base
     - Docker base Ubuntu image

Export Execution Time



Export Size


Container Size (MBs)
0
200
400
600
800
1000
1200
1400
Docker Base
FLINC Export
Docker Container

Future Work

- Detailed analysis on the performance of FLINC.
- Beyond current reliance on Linux *strace* for auditings.
- Extendible to other event tracing systems like ETW/procmon for Windows and OpenBSM/dtrace for MacOS.
- Extendible to other packaging formats such as Flatpak or MacOS Bundles.

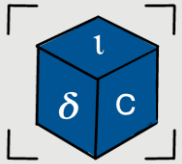
Conclusion

- Motivate the need for sharing and reproducing notebooks successfully.
- Elaborate reproducibility of notebooks for different scenarios.
- Introduce FLINC to create reproducible notebook containers that work across multiple environments.
- Experiments on notebooks using FLINC.

Acknowledgements

- NSF CNS-1846418, NSF ICER-1639759, ICER-1661918.

- David Tarboton, Jonathan L. Goodall, Shilvi Satpati, YoungDon Choi, Ayman Nassar, Binata Roy, Iman Maghami, Zhiyu Li, and Anthony Castronova.



THANK YOU!

For questions, please contact:

Raza Ahmad
raza.ahmad@depaul.edu

Tanu Malik
tanu.malik@depaul.edu

Slides:
https://docs.google.com/presentation/d/1K4nzO4TWhTSqCI8niW8niSHwtNVqHaK7

Slide Credits

- Girl icons created by mynamepong - Flaticon
- Software engineer icons created by Freepik - Flaticon
- Terminal icons created by icon_small - Flaticon
- Laptop icons created by Freepik - Flaticon
- Code icons created by Nikita Golubev - Flaticon>
- Code icons created by juicy_fish - Flaticon
- Tick icons created by Freepik - Flaticon
- Efficiency icons created by Eucalyp - Flaticon
- Flexible icons created by Eucalyp - Flaticon
- User interface icons created by Eucalyp - Flaticon