Streamlit for Snowflake

Python Development of Streamlit Web Apps, Streamlit Apps and Snowflake Native Apps

Course Summary

- Streamlit Bootcamp
- Local Streamlit Web Apps
- Streamlit Community Cloud Web Apps
- Snowflake Client Apps with Streamlit
- Snowpark Web Apps with Streamlit
- Streamlit in Snowflake Apps
- Snowflake Native Apps Framework

Streamlit Application Types

- General-Purpose Local Web Apps
- Web Apps Deployed in Streamlit Cloud
- Snowflake Client Apps for Data
- Snowflake Client Apps for Metadata
- Streamlit Apps and Native Apps
- Snowflake Data Analysis Apps
- Data Science Apps for Snowflake

Hierarchical Data Viewer

- Simple Python Application on CSV Data
- Local Streamlit Web App on CSV Data
- Streamlit Community Cloud App
- Client App on Snowflake Tables
- Snowpark App with Recursive Queries
- Streamlit in Snowflake App
- Snowflake Native App

Hierarchical Metadata Viewer

- Snowflake Object Dependencies
- Snowflake Data Lineage
- Client App on Account Usage Views
- Multi-Page Streamlit Application
- Streamlit in Snowflake App
- Snowflake Native App

Other Streamlit Apps

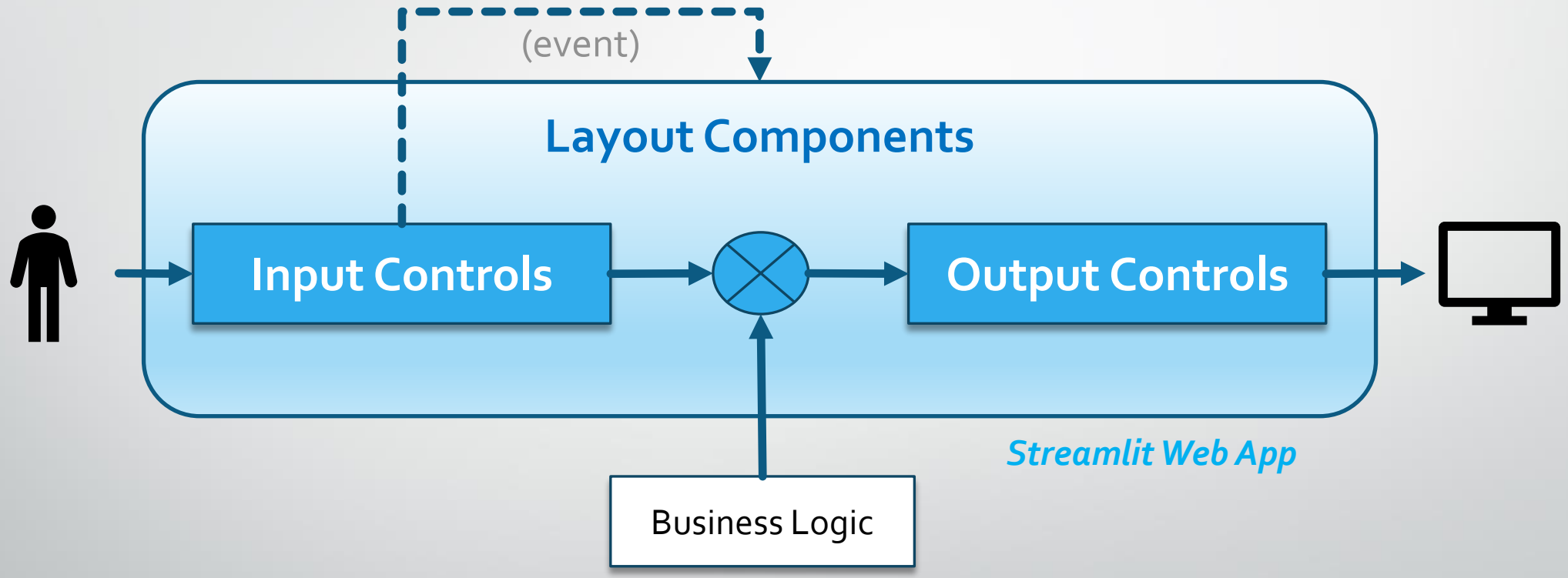
- Data Analytics with Dashboards
- Business Intelligence Applications
- Machine Learning Applications
- ChatGPT Integration with Snowflake
- Data Marketplace Enrichment
- ER Diagram Viewer for Snowflake

(1) Testing Local Streamlit Web Apps

Streamlit Front-End Bootcamp

- Introduction to Streamlit
- Output Controls
- Layout Components
- Input Controls
- Events & Page Reruns
- Data & Resource Caching
- Session State and Callbacks

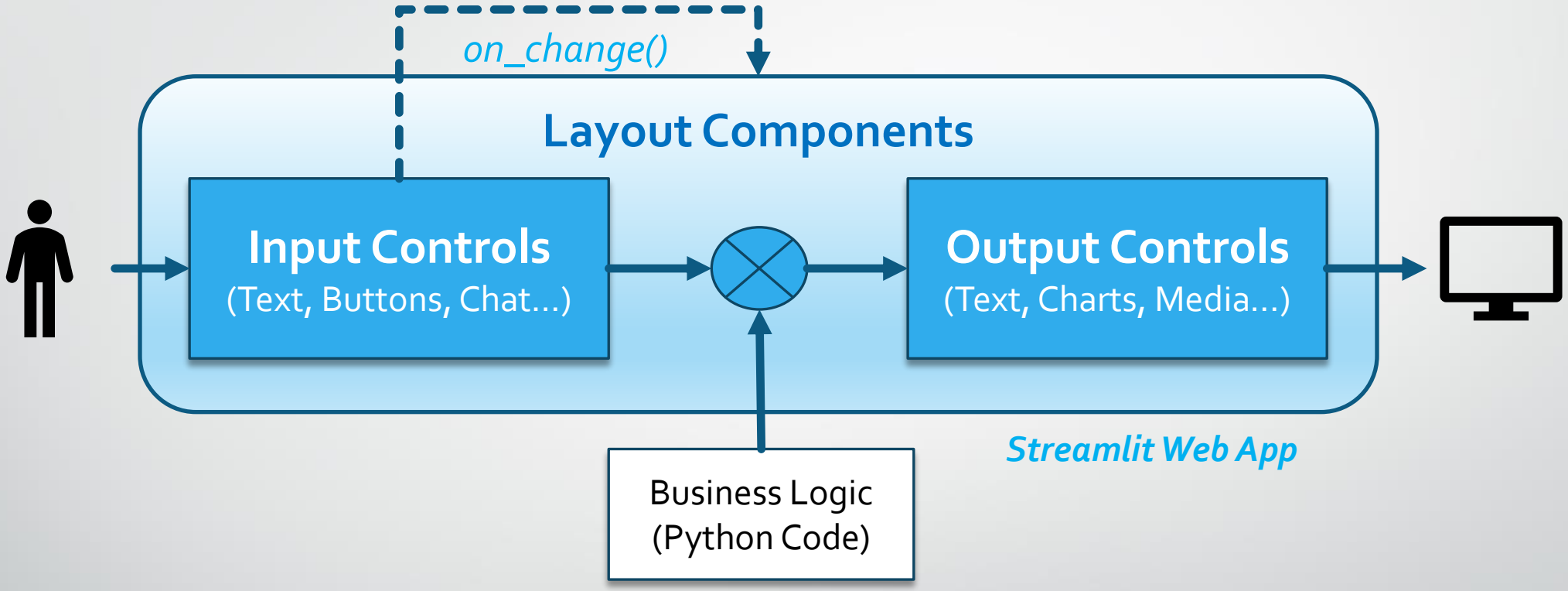
Streamlit Front-End Architecture



Introduction to Streamlit

- **History**
  - Bought by Snowflake in 2022 for $800M
  - Integrated with Snowpark: Streamlit Apps (SiS) → Native Apps
- **Features**
  - RAD framework for data science experiments (~VB, Access, Python at the app level)
  - Connect to all sorts of data sources (Snowflake etc)
  - Instant rendering as charts or HTML content, using rich third-party libraries
  - Provides front-end (I/O controls + components) and runtime support (as web app)
- **Development**
  - Great for prototyping and proof-of-concept simple apps (not like heavy React apps!)
  - Support for single and multi-page applications
  - Test as local web app (not standalone!)
  - Share and deploy as remote web app to Streamlit Cloud (for 100% free!)

Streamlit Architecture: Front-End, as a Library



- Simple input controls: single event per control → trigger full page rerun
- Rich third-party output libraries & minimalistic layout components (as containers)
- Business logic (external!): database access (Snowflake), ML (PyTorch, TensorFlow)

Output Controls (1)

- st.**write**('Most objects'), st.**write**(['st', 'is <', 3])
- st.**text**('Fixed width text')
- st.**title/header/subheader/caption**('My title')
- st.**code**('for i in range(8): foo()') ← source code (w/ optional line numbers)
- st.**markdown**('_Markdown_') ← write markdown
- st.**latex**(r''' e^{i\pi} + 1 = 0 ''') ← write formulas

- st.**divider** ← ~HR
- st.**link_button**("Go to gallery", url) ← no events
- st.**image** ← show image/list of images
- st.**audio/video** ← show audio/video player

Output Controls (2)

- st.**dataframe** ← w/ dataframes from Pandas, PyArrow, Snowpark, PySpark
- st.**table** ← show static table
- st.**json** ← show pretty-printed JSON string

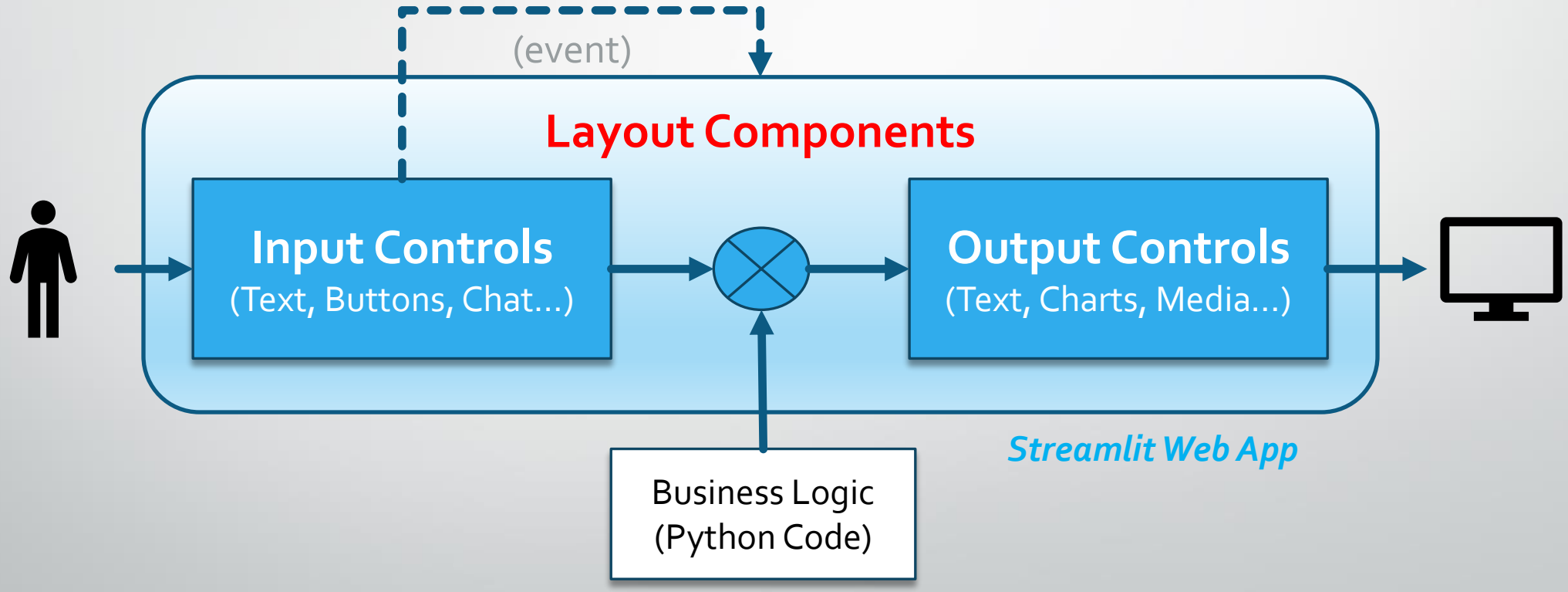
- st.**error/warning/info/success/toast**('Error message')
- st.**exception**(e)
- st.**balloons/snow**()

- with st.**spinner**(text='In progress'): …
- bar = st.**progress**(50) … bar.**progress**(100)
- with st.**status**('Authenticating…') as s: … s.update(label='Response')

Output Controls (3)

- st.**graphviz_chart**(fig) ← GraphViz encapsulation (Dagre-D3 lib)
- st.**area/bar/line/scatter_chart**(df) ← primitive Altair charts
- st.**map**(df) ← geo map w/ scatterplot (w/ PyDeck)
- st.**altair_chart**(chart) ← Altair lib
- st.**bokeh_chart**(fig) ← Bokeh lib
- st.**plotly_chart**(fig) ← interactive Plotly chart
- st.**pydeck_chart**(chart) ← free 3D maps (PyDeck lib)
- st.**pyplot**(fig) ← w/ matplotlib.pyplot figure
- st.**vega_lite_chart**(df) ← Vega-Lite lib
- st.**column_config** ← insert spark lines!
- st.**metric** ← show perf metric number in large

Streamlit Architecture: Front-End (as a Library)

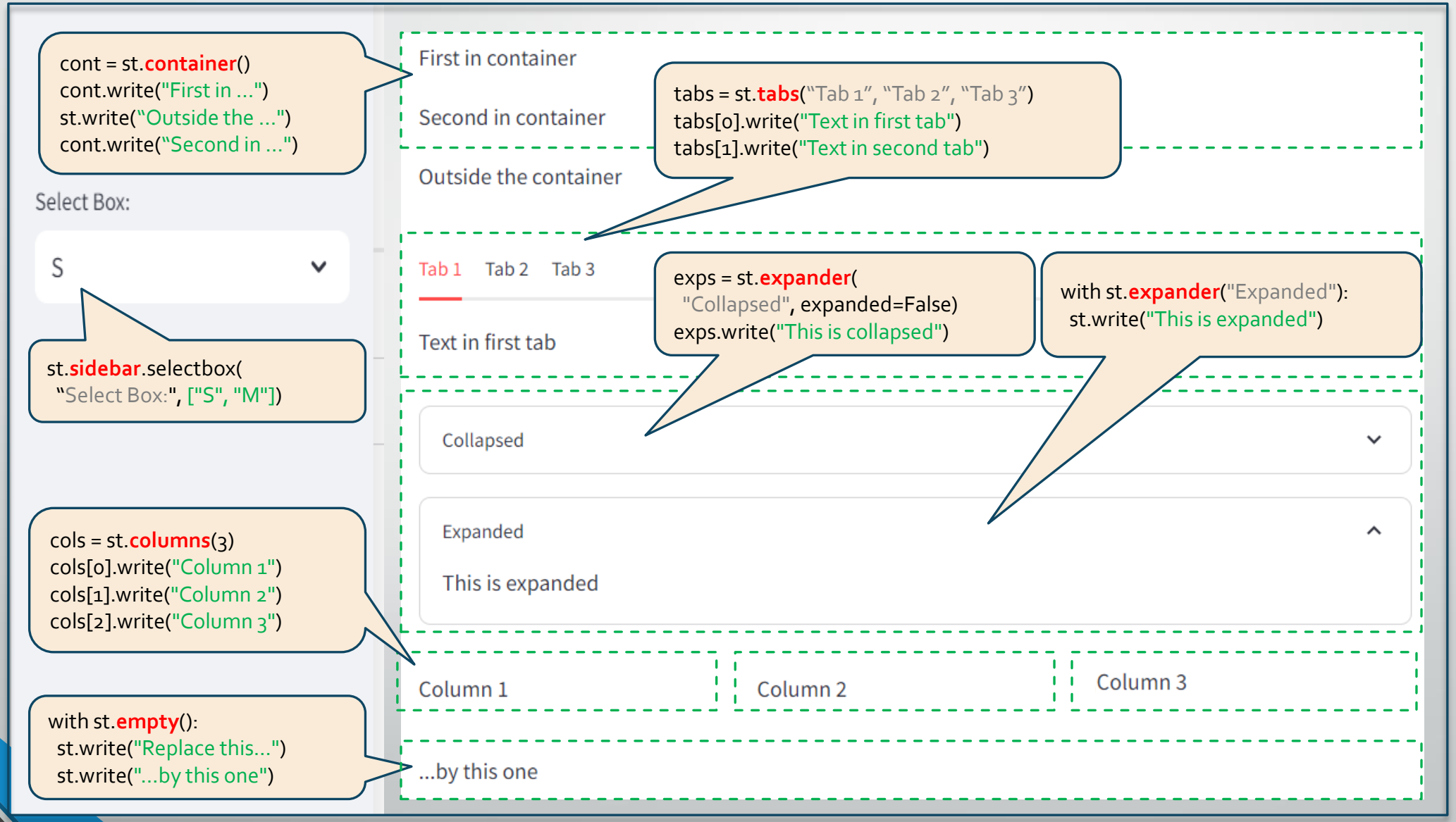


Layout Components

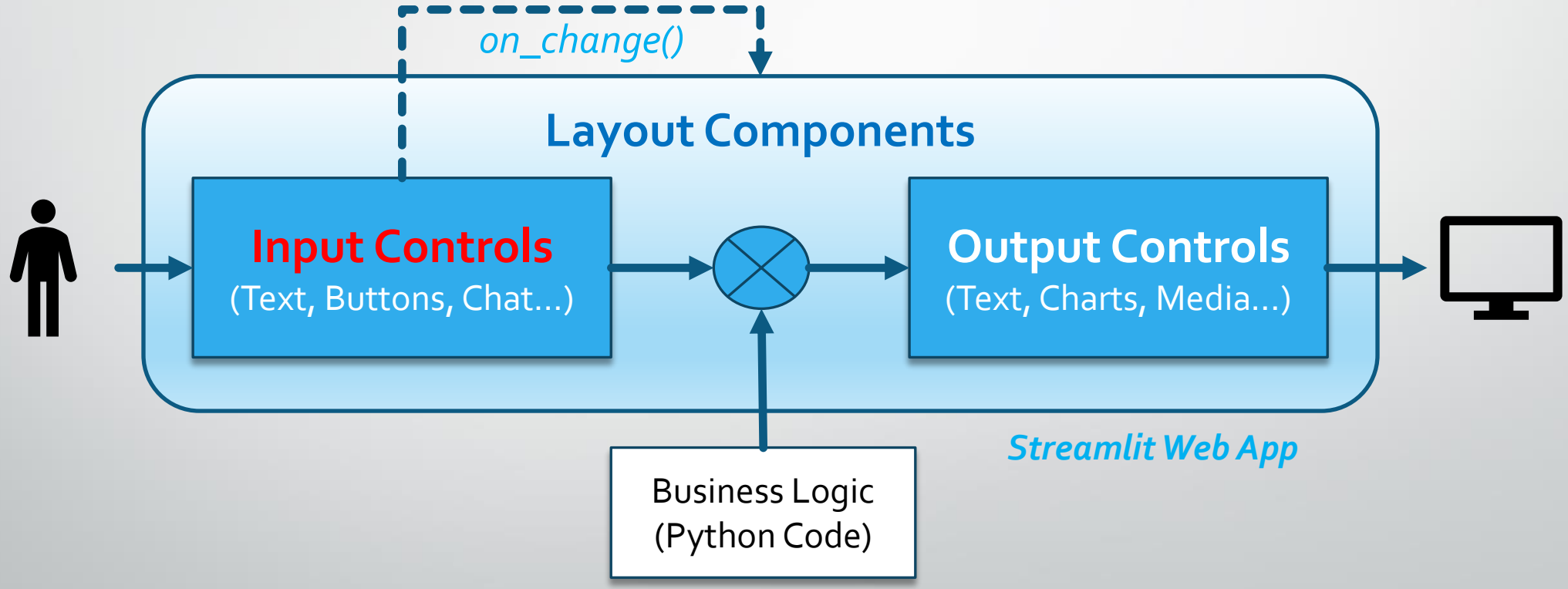
- st.*sidebar* ← collapsible left sidebar
- st.**tabs** ← group of tab pages

- st.**columns** ← group of side-by-side horizontal containers
- st.**expander** ← collapsible container (collapsed by default)
- st.**container** ← multi-element container
- st.**empty** ← single-element placeholder (w/ text replace)

Layout Components



Streamlit Architecture: Front-End (as a Library)



Input Controls (1)

- st.**text/number/date/time_input**("First name") ← on_change() event
- st.**text_area**("Text to translate") ← on_change() event

- st.**selectbox**("Pick one", ["cats", "dogs"]) ← on_change() event
- st.**multiselect**("Buy", ["milk", "apples", "potatoes"]) ← on_change() event
- st.**slider**("Pick a number", 0, 100) ← on_change() event
- st.**select_slider**("Pick a size", ["S", "M", "L"]) ← on_change() event
- st.**color_picker**("Pick a color") ← on_change() event

Input Controls (2)

- st.**button**("Click me") ← buttons on single line, on_click() event
- st.**toggle**("Enable") ← on_change() event
- st.**checkbox**("I agree") ← on_change() event
- st.**radio**("Pick one", ["cats", "dogs"]) ← on_change() event

- st.**file_uploader**("Upload a CSV") ← on_change() event
- st.**download_button**("Download file", data) ← on_click() event

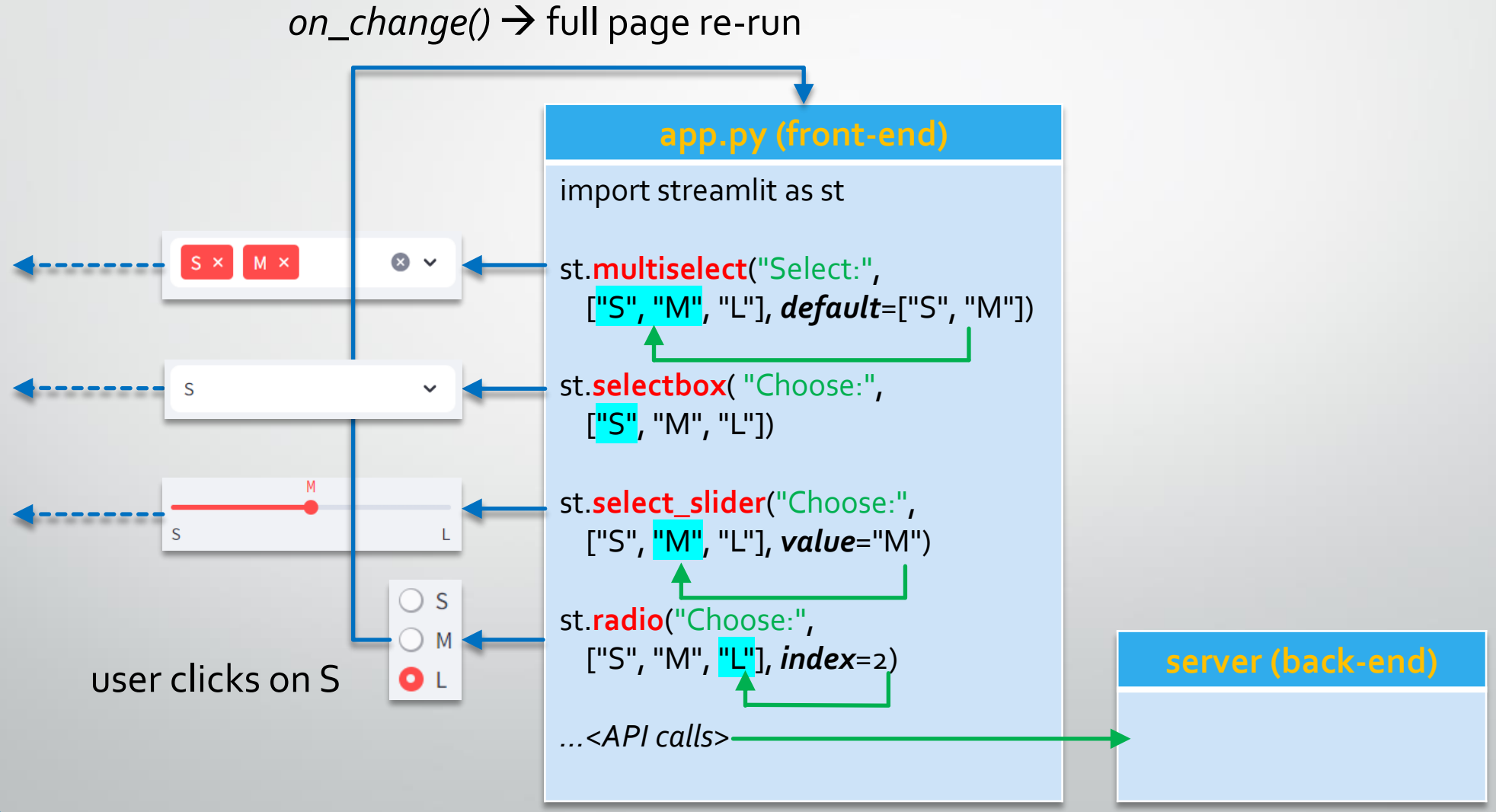
Input Controls (3)

- st.**data_editor** ← show widget, on_change() event
- st.**camera_input**("Take a picture") ← on_change() event

- st.**chat_input**("Say something") ← prompt chat widget, on_submit() event
- with st.**chat_message**("user"): … ← response to a chat message

- st.**form_submit_button** ("Submit") ← on_click() event (in a form!)
- with st.**form**(…): … ← container for multiple input controls

Interactive Widgets

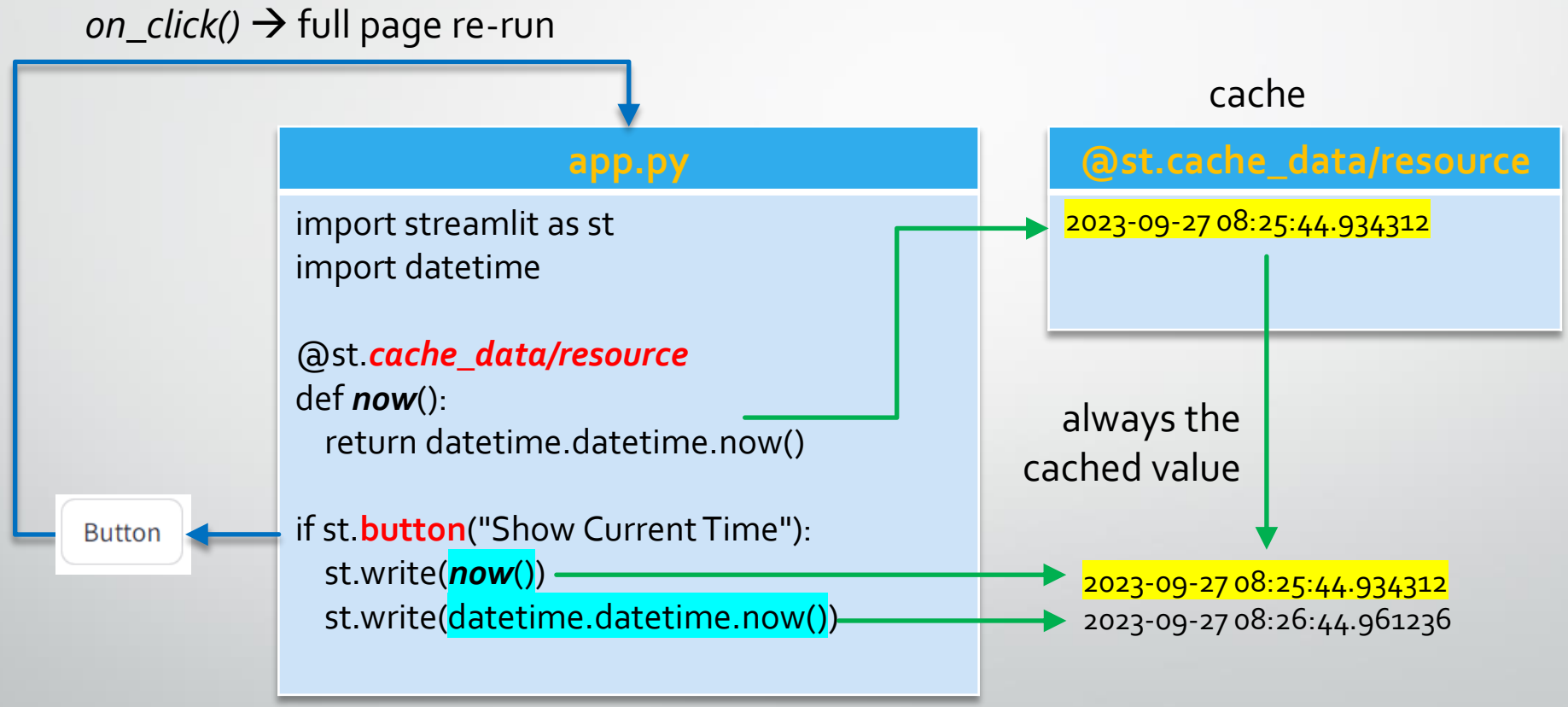


Caching

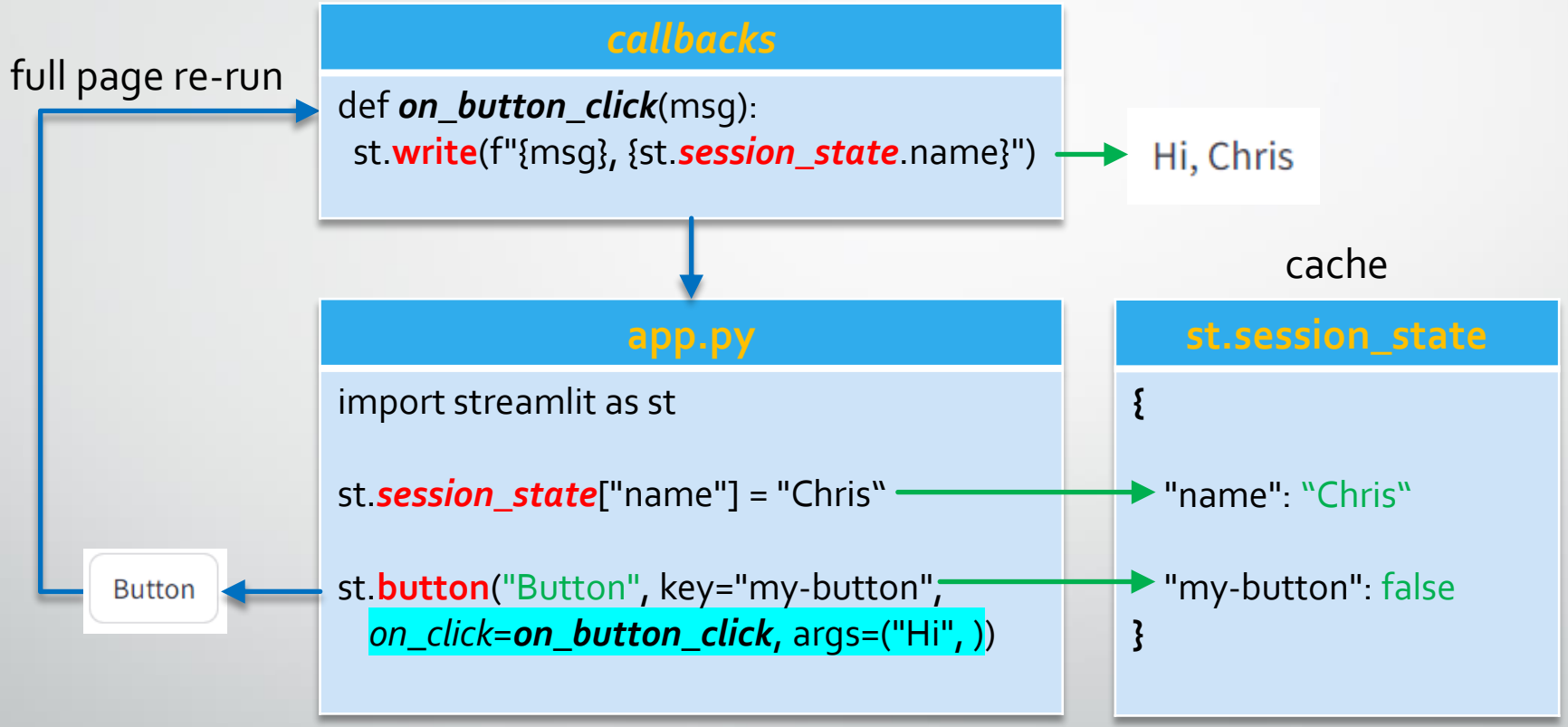
- @st.**cache_data** ← for serializable objects, always copy!
- @st.**cache_resource** ← for non-serializable "live" objects, no copy
- ~~@st.**cache**~~ ← deprecated (must replace)

- *arguments*
  - **ttl**=3600 ← entry expires after 1h
  - **max_entries**=10 ← keeps max 10 records (LRU cache)
  - **show_spinner**=False ← no visible spinner
  - **show_spinner**="Loading data..." ← custom spinner message

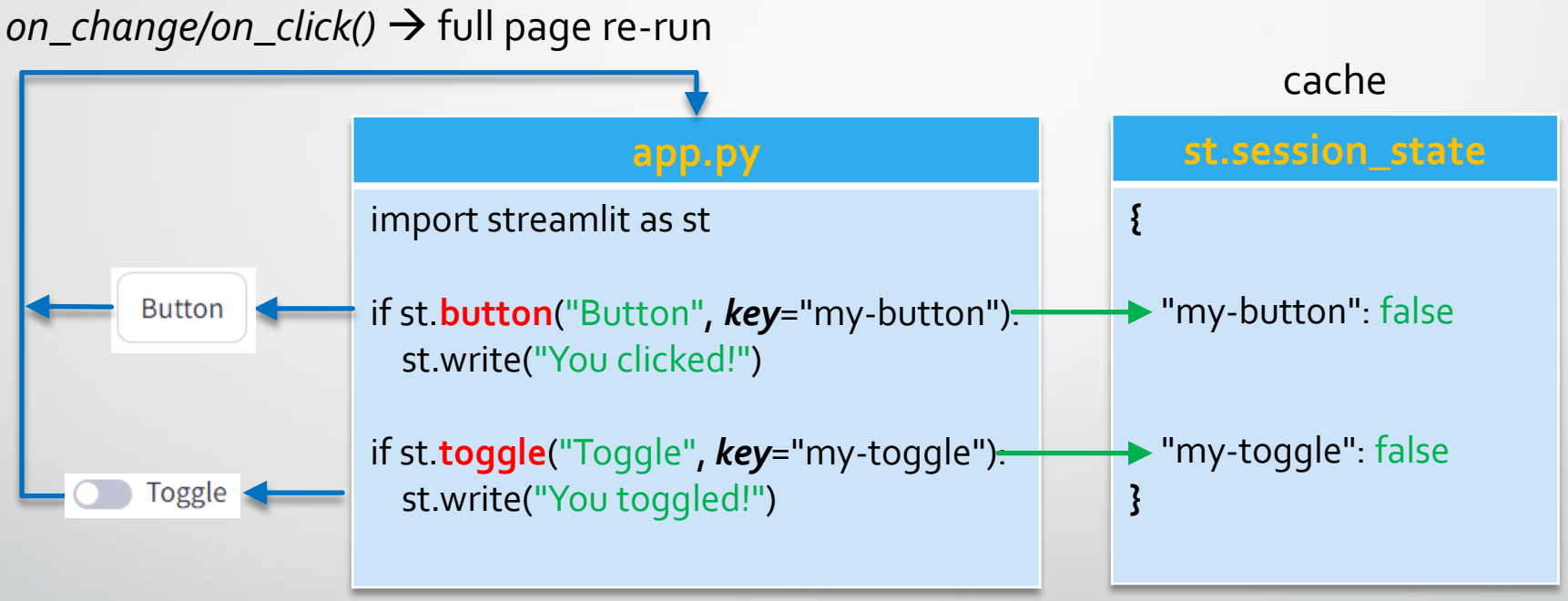
Caching Example



Session State and Callbacks



Buttons

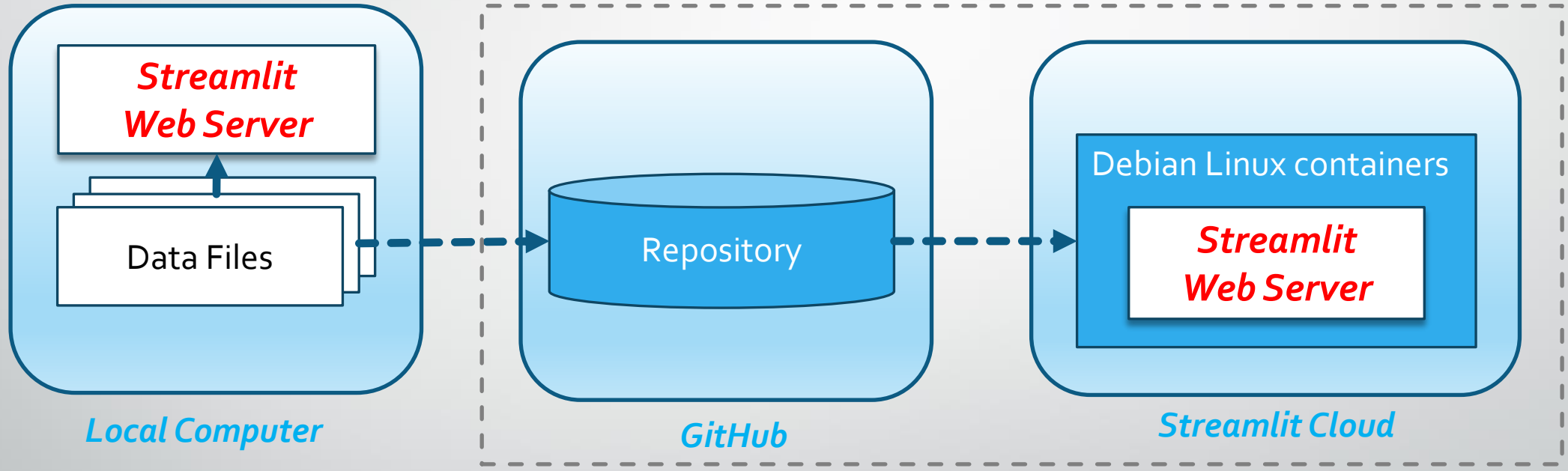

on_change/on_click() → full page re-run
cache
app.py
st.session_state
import streamlit as st
if st.button("Button", key="my-button"):
st.write("You clicked!")
if st.toggle("Toggle", key="my-toggle"):
st.write("You toggled!")
Button
Toggle
{
"my-button": false
"my-toggle": false
}

(2) Sharing Streamlit Apps in Streamlit Cloud

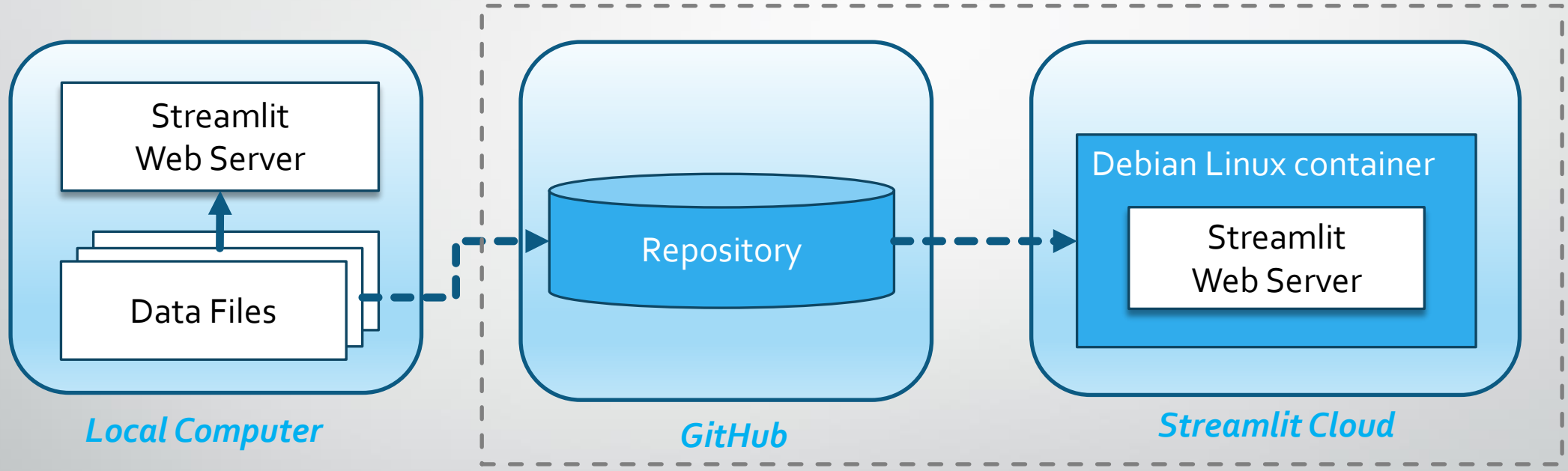
Streamlit Runtime Bootcamp

- Streamlit Local Web Applications
- Streamlit Cloud Web Applications
- Streamlit Secrets
- Sharing Private Apps
- Protecting Public Apps

Streamlit Runtime Architecture



Streamlit Runtime: Local to Streamlit Cloud



Publish to GitHub

- Streamlit will create access keys! → need authorization
- your app will automatically refresh on each new GitHub push
- can later add "Open in Streamlit" button in GitHub

Deploy to Streamlit Cloud

- sign-up with your Google email at **share.streamlit.io**
- make sure app can be shared (for free) → see limits! use subdomain
- replace any ***app\myapp.py*** to ***app/myapp.py***, if from subfolder (\ → /)
- prefix with ***os.path.dirname(__file__)}/*** any relative file names
- make sure **requirements.txt** is updated → check black sidebar log
- add any pwds or confidential data as Secrets (see ***Advanced Options***)
- make sure you'll run the same version of Python when deployed
- can later add the link in Medium posts → expanded as gadget

(3) Connecting Streamlit Apps to Snowflake

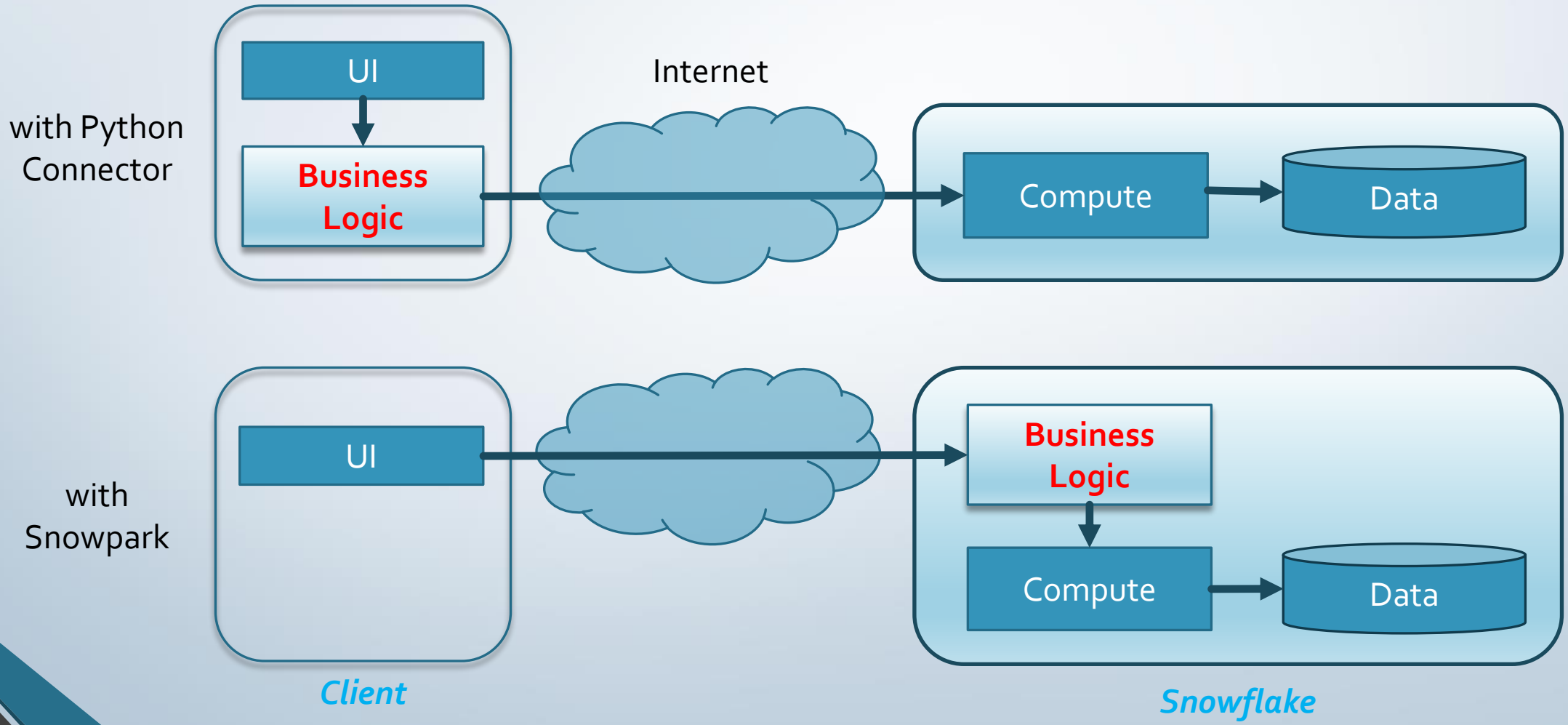
Streamlit and Snowflake

- Creating a Free Snowflake Account
- Snowflake Connector for Python
- Snowpark for Python
- Streamlit Connector to Snowflake
- Recursive SQL Queries
- Stored Procs and UDFs

Streamlit Multi-Page Apps

- Multiple Snowflake Connectors
- Hierarchical Metadata Viewer
- Dashboard with Vega-Lite Charts
- Dashboard with Altair Charts
- Run a Native App from Marketplace

Python Connector vs Snowpark



SnowSQL (CLI Client)

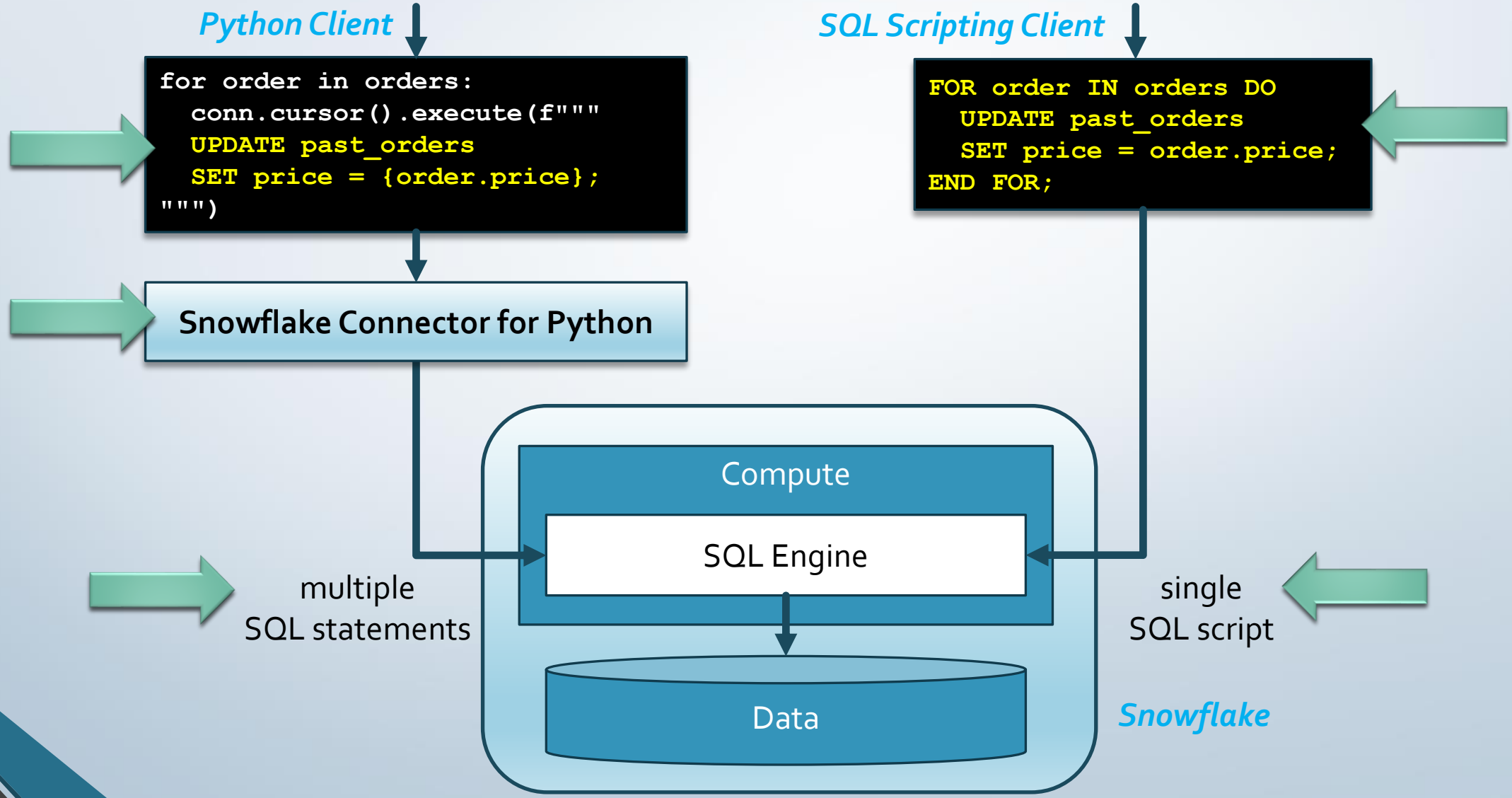
- **Overview**
  - Installs as command line tool for Linux/macOS/Windows
  - Developed with ***Snowflake Connector for Python***
- **Features**
  - Connect to Snowflake account
  - Perform SQL (DDL/DML) operations
  - Run SQL deploy scripts (including **PUT/GET** for local files)
  - Automate SQL deployments from bash shell through batch scripts
  - Abort/pause running queries
- **Hidden Gems**
  - **~/.snowsql/config** file may be re-used to connect to Snowflake from client apps
  - Can use **SNOWSQL_PWD** env var to save local password
  - Customize scripts through ***variable substitution*** (for partner databases)
  - Generate ***JWT tokens*** for key pair authentication (for SQL REST API)

Client Drivers

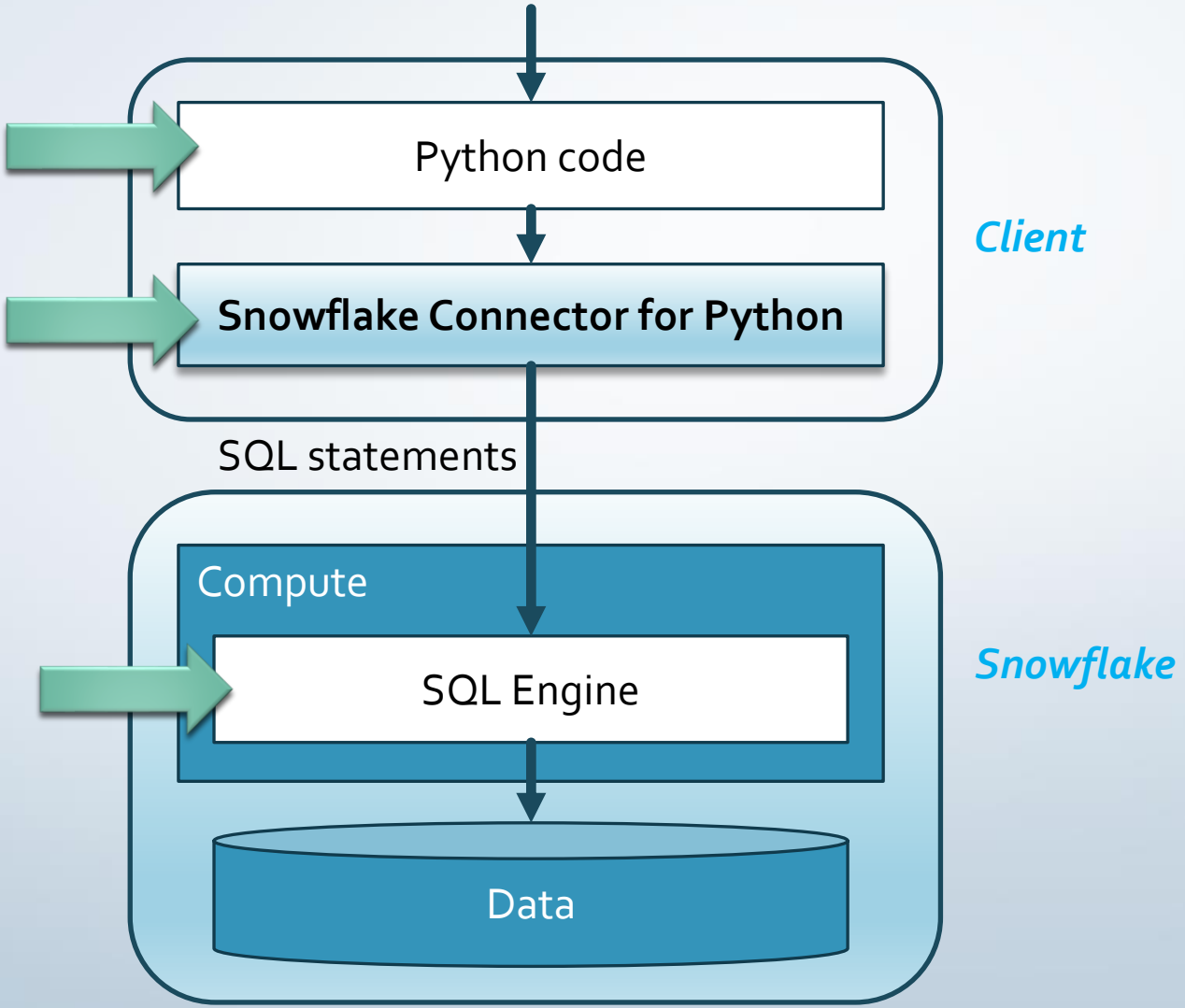
- **Snowflake Connector for Python**
- **.NET Driver** – for C#
- **Node.js Driver** – for JavaScript
- **Go Snowflake Driver**
- **PHP PDO Driver for Snowflake**

- **JDBC Driver** ← for Java, Scala…
- **ODBC Driver** ← from Windows

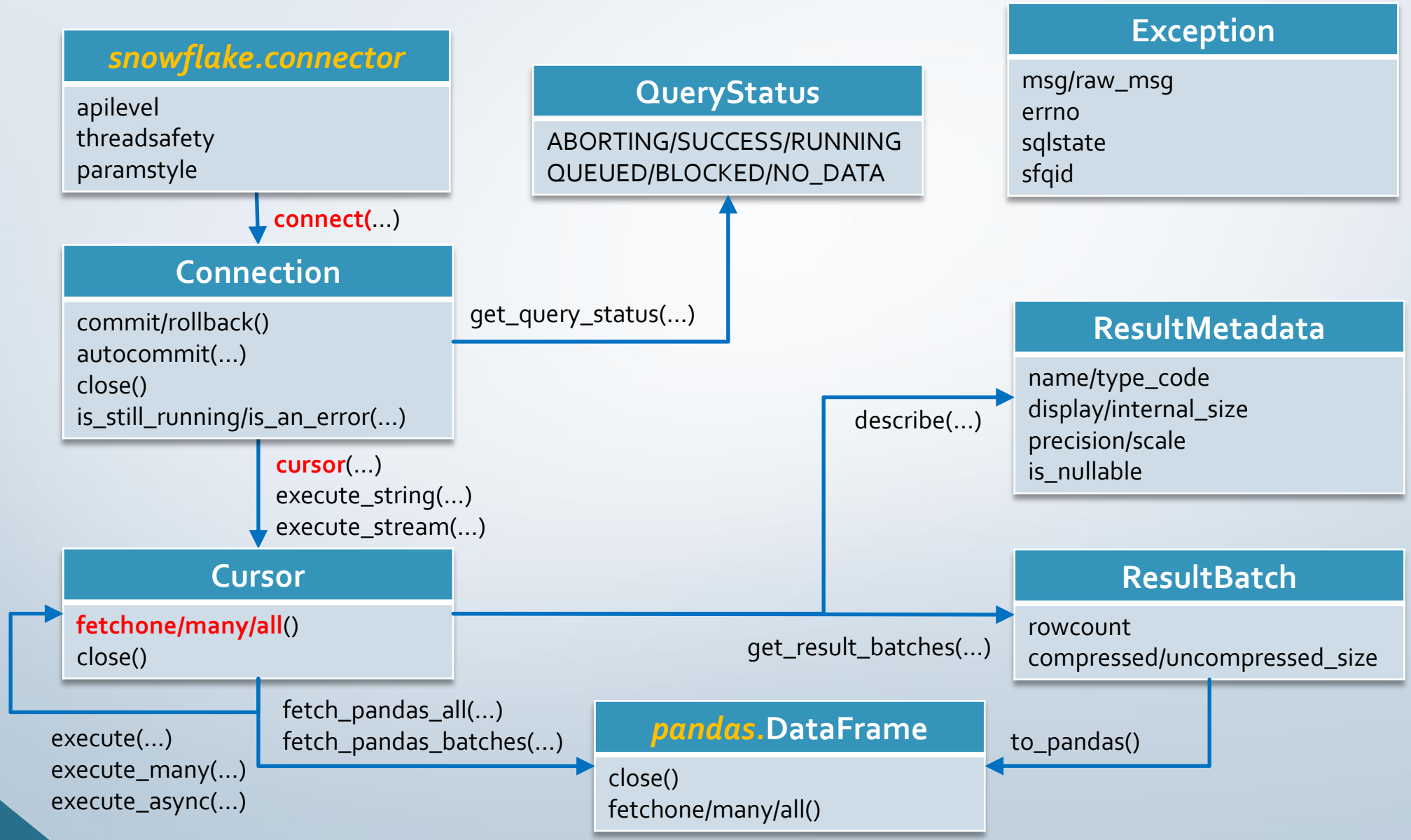
Client-Side vs Server-Side Cursors



Snowflake Connector for Python



Python Connector: API



Python Connector: Common Pattern

Python Client

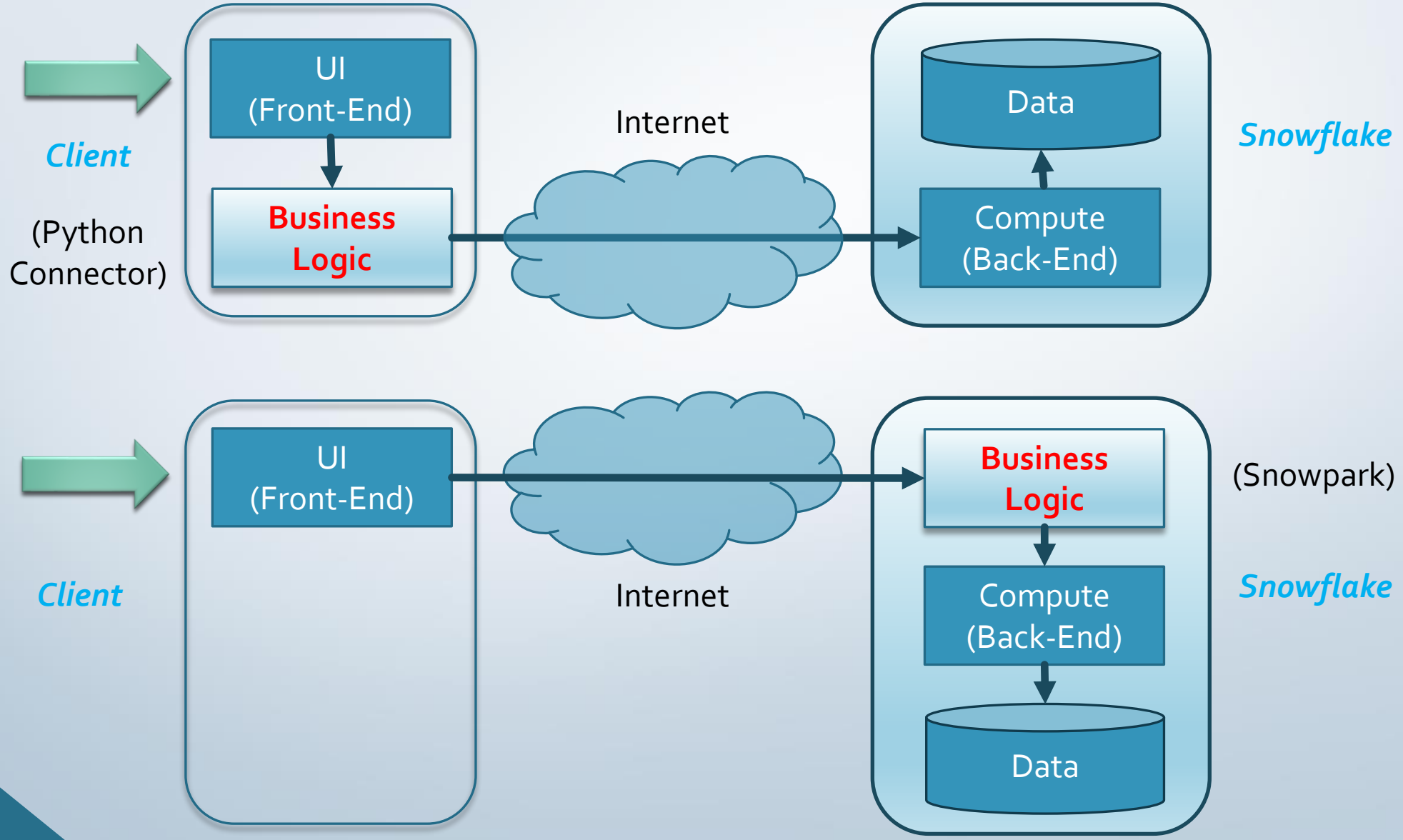
```
import snowflake.connector

# show all property=value pairs for current user
with snowflake.connector.connect(…) as conn:
    with conn.cursor() as cur:
        cur.execute(“show parameters")
        for row in cur:
            print(f'{str(row[0])}={str(row[1])}')
```

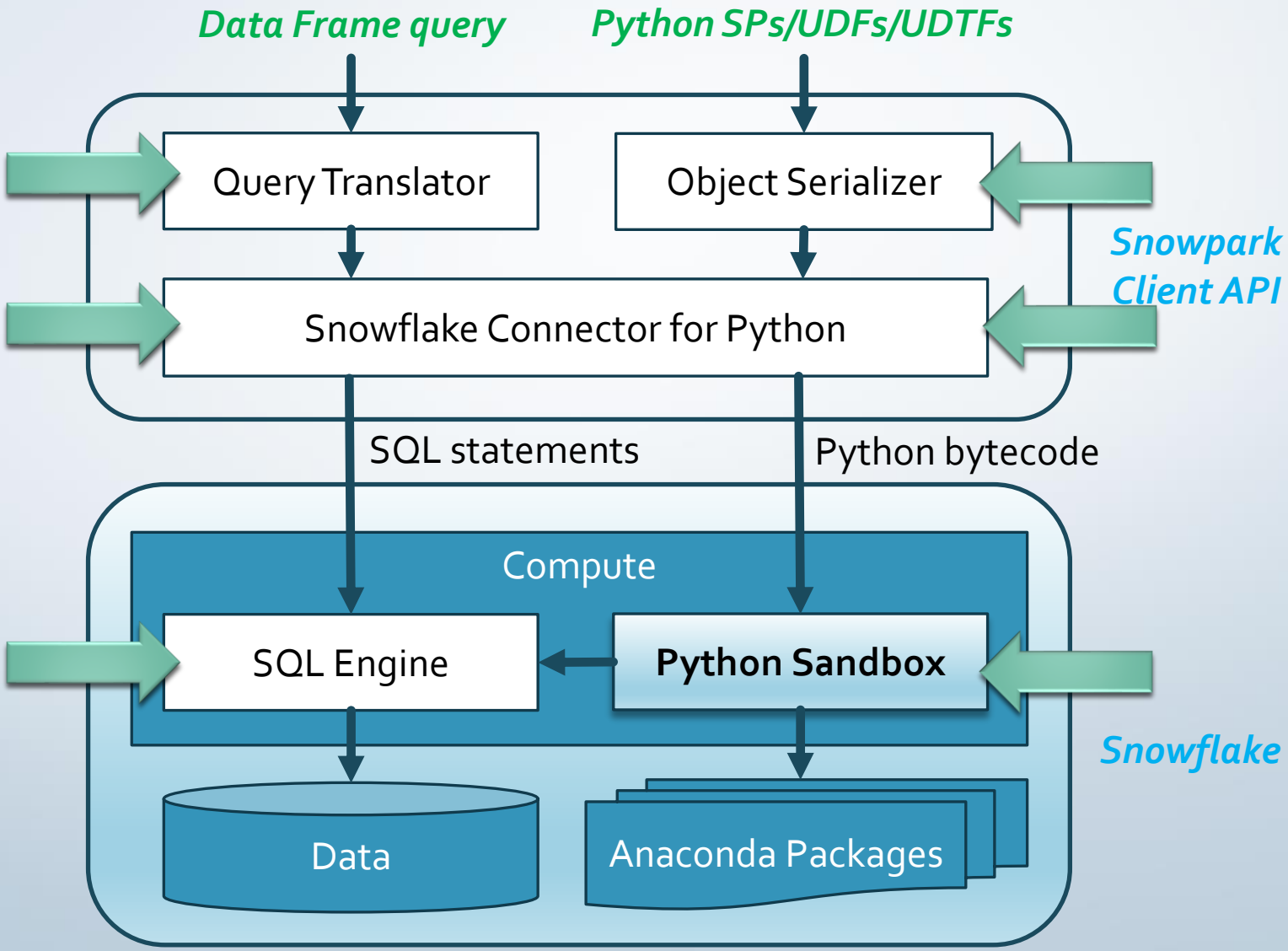
PROBLEMS OUTPUT TERMINAL PORTS CODEWHISPERER REFERENCE LOG

```
SNOWFLAKE_LOCK=false
SNOWFLAKE_SUPPORT=false
DAYS_TO_EXPIRY=null
MINS_TO_UNLOCK=null
DEFAULT_WAREHOUSE=COMPUTE_WH
DEFAULT_NAMESPACE=null
DEFAULT_ROLE=ACCOUNTADMIN
DEFAULT_SECONDARY_ROLES=null
EXT_AUTHN_DUO=false
EXT_AUTHN_UID=null
```

Client-Side vs Server-Side Programming

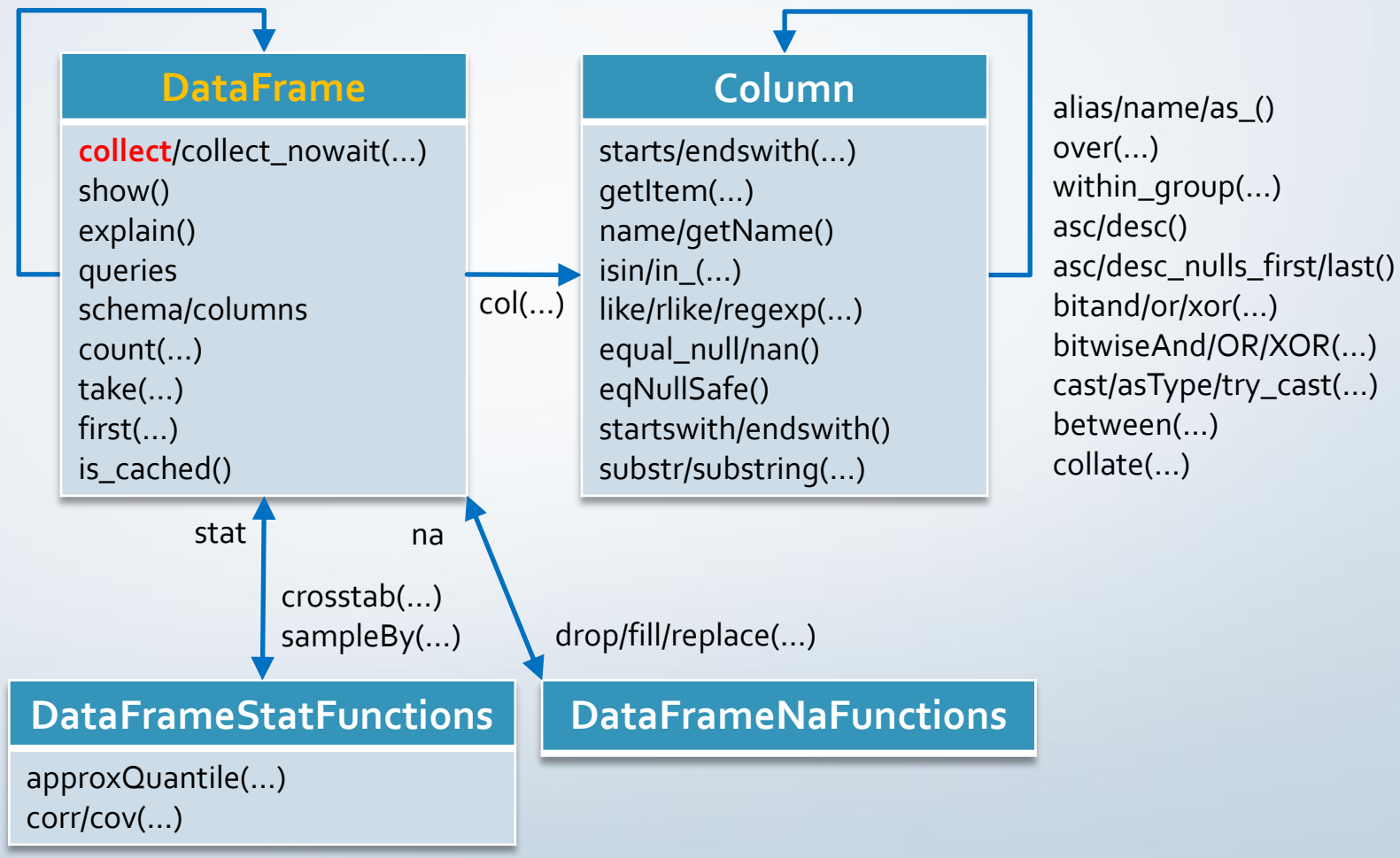


Snowpark for Python



DataFrame Class

select/selectExpr(...)
filter/where(...)
sort/orderBy(...)
union/unionAll/unionByName(...)
intersect/except_/minus/substract
withColumn/withColumnRenamed(...)
with_columns(...)
join/crossJoin/natural_join(...)
join_table_function/flatten(...)
limit(...)
agg(...)
drop/dropna/fillna(...)
crosstab/unpivot(...)
describe/rename/replace(...)
sample/sampleBy(...)
randomSplit(...)
toDF(...)




```
rows = get_active_session().sql("SELECT ...").collect()
```

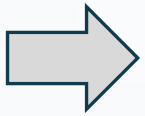
Create Query with DataFrame

Python Code using DataFrame

```
emps = (session.table("EMP")
  .select("DEPTNO", "SAL"))
depts = (session.table("DEPT")
  .select("DEPTNO", "DNAME"))
q = emps.join(depts,
  emps.deptno == depts.deptno)

q = q.filter(q.dname != 'RESEARCH')
q = (q.select("DNAME", "SAL")
  .group_by("DNAME")
  .agg({"SAL": "sum"})
  .sort("DNAME"))

q.show()
```

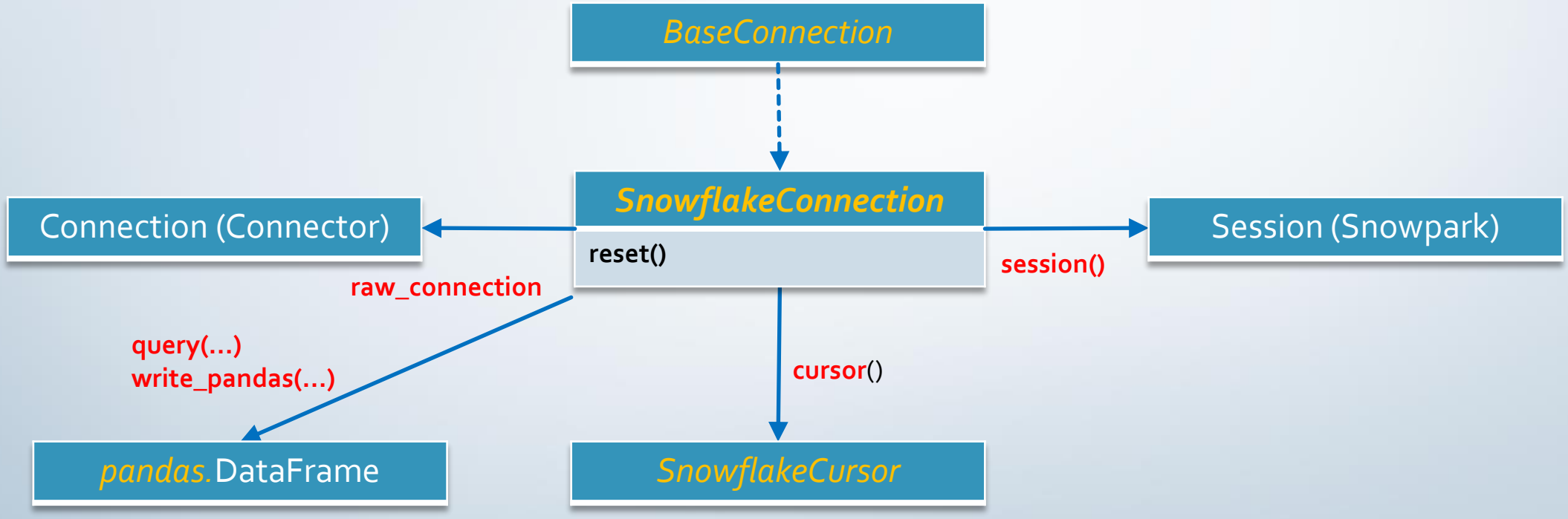


Generated SQL

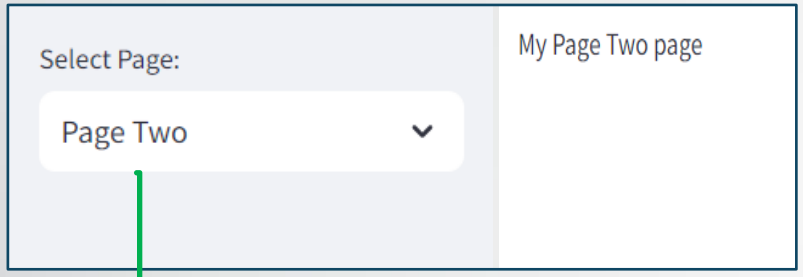
```
select dname, sum(sal)
  from emp join dept
  on emp.deptno = dept.deptno
  where dname <> 'RESEARCH'
  group by dname
  order by dname;
```

DNAME	SUM(SAL)
ACCOUNTING	8,750.5
SALES	9,400

st.connections.SnowflakeConnection Class



Multi-Page Applications: w/ SelectBox



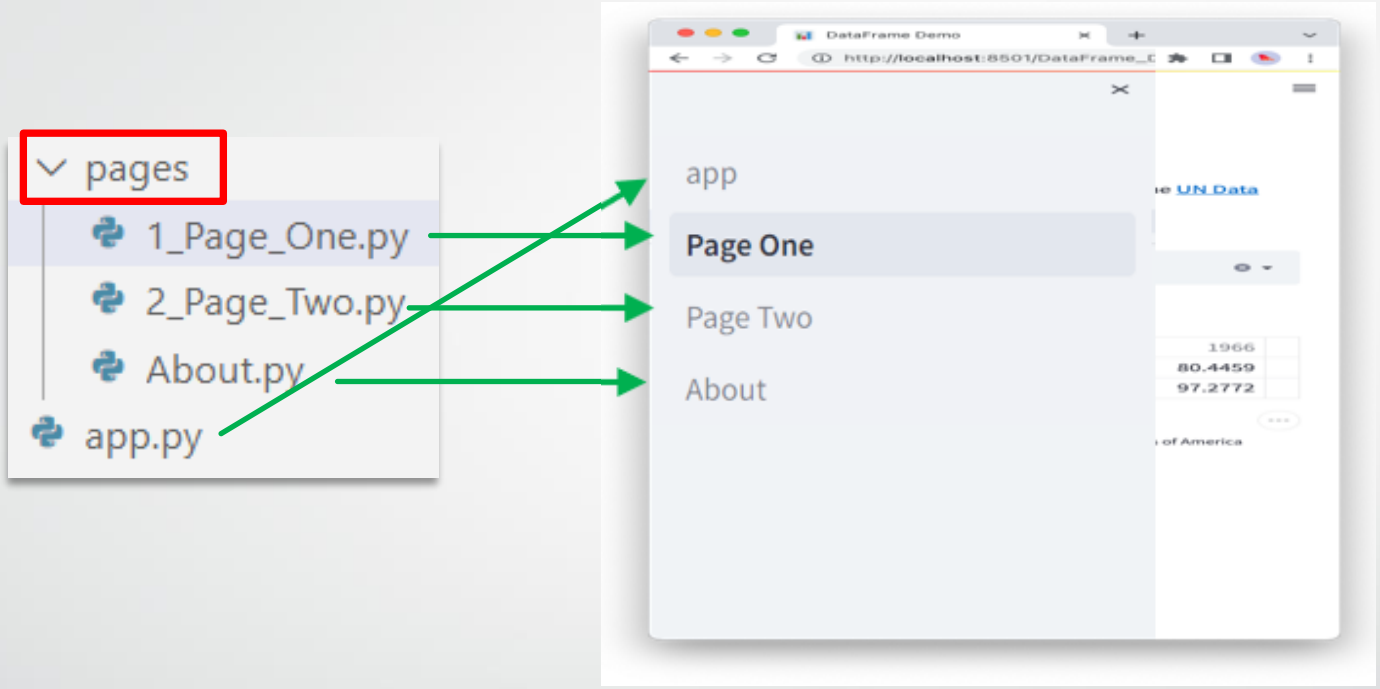
app.py

```
import streamlit as st

def main(): st.write("My main page")
def page_one(): st.write("My Page One page")
def page_two(): st.write("My Page Two page")
def about(): st.write("My About page")

funcs = {
   "-": main,
   "Page One": page_one,
   "Page Two": page_two,
   "About": about }
name = st.sidebar.selectbox(
   "Select Page:", funcs.keys())
funcs[name]()
```

Multi-Page Applications: w/ Pages Folder



1_Page_One.py

```
import streamlit as st

st.set_page_config(
    page_title="Plotting Demo",
    page_icon="📈")
```

(4) Deploying Streamlit Apps to Snowflake

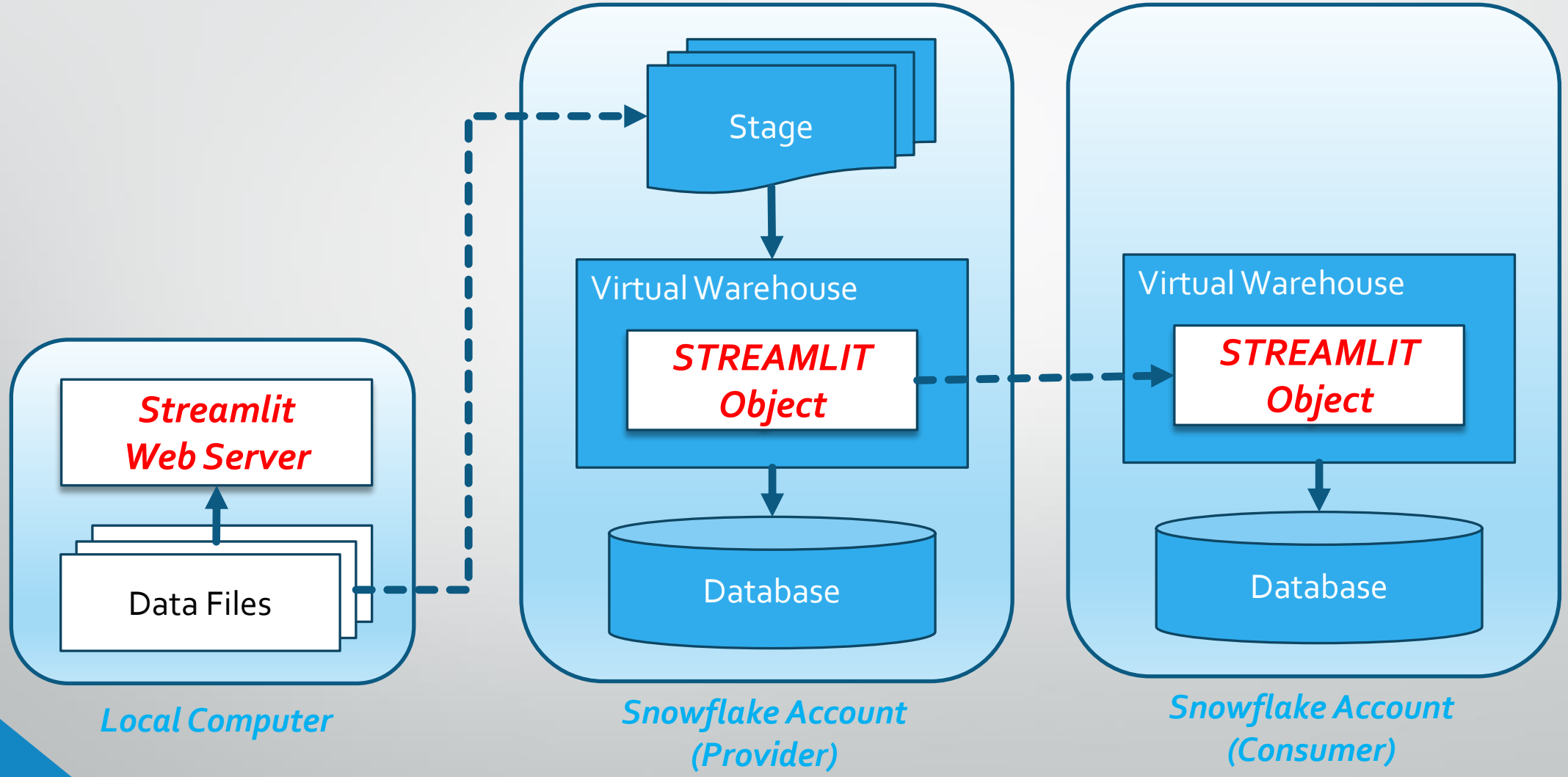
Streamlit in Snowflake Apps

- How to Deploy Streamlit to Snowflake
- Hierarchical Data Viewer as Streamlit App
- Hierarchical Metadata Viewer as Streamlit App
- Multi-Page Dashboards as Streamlit Apps
- Data Science Applications as Streamlit Apps

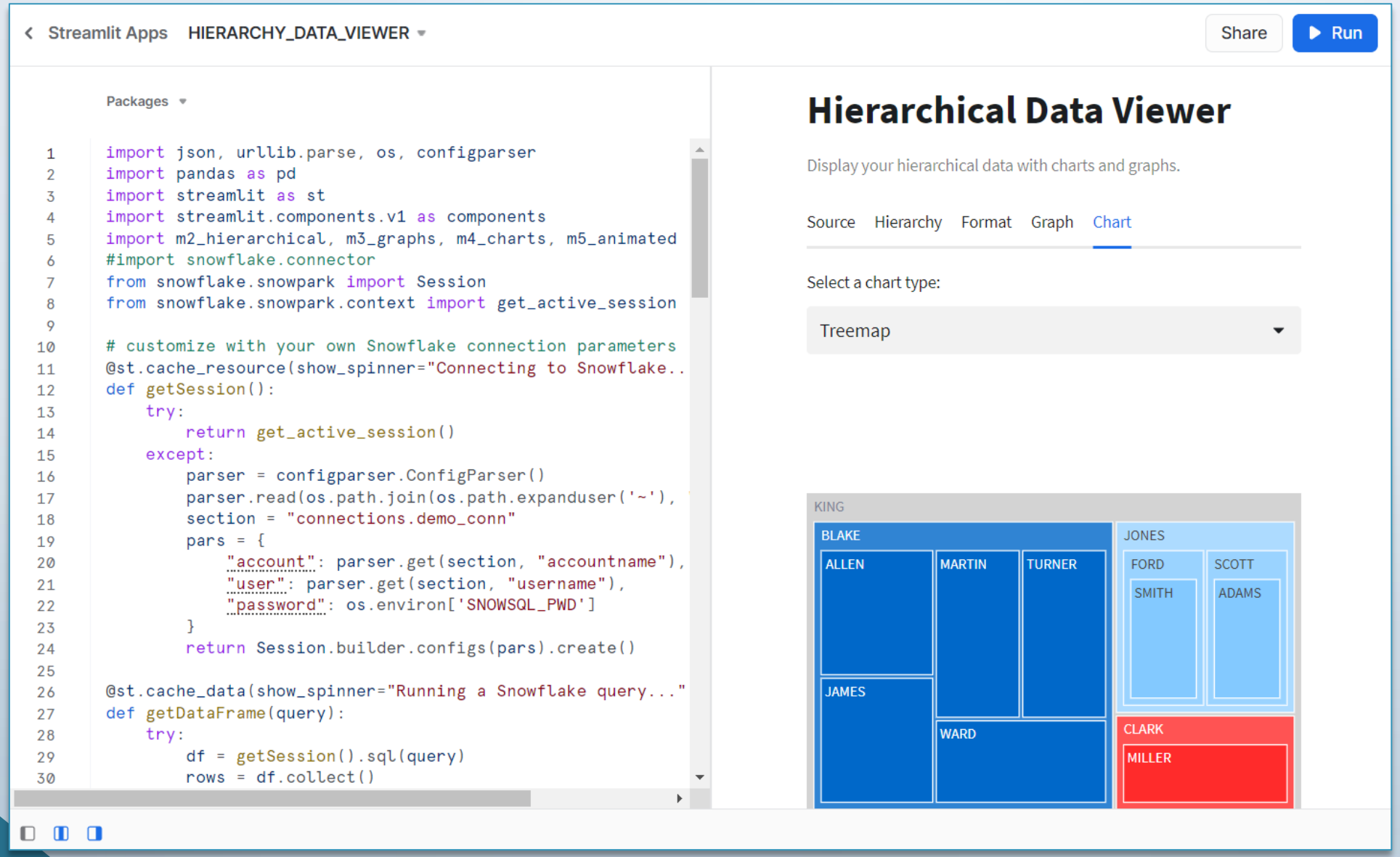
Snowflake Native App Framework

- How to Share a Snowflake Native App
- Private and Public Sharing of Native Apps
- Hierarchical Data Viewer as Native App
- Hierarchical Metadata Viewer as Native App
- Data Enrichment Application as Native App
- Run Native Apps from Snowflake Marketplace

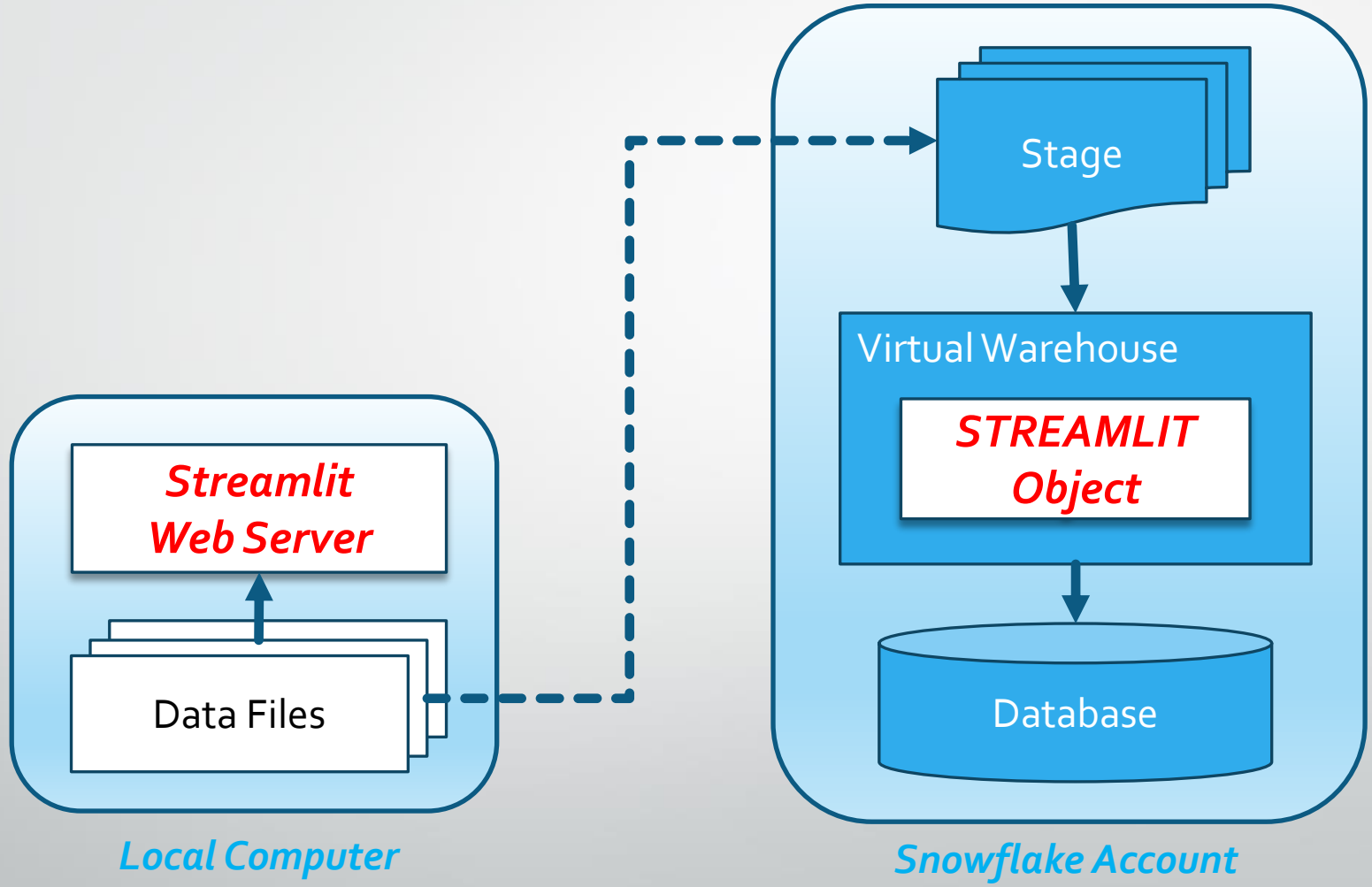
Deployed Streamlit to Snowflake Architecture



Streamlit in Snowflake



Streamlit Runtime: Local to Snowflake Account



Streamlit in Snowflake

- **Create and test as a local Streamlit web app**
  - Create a local Streamlit app, with one or more Python files.
  - Connect locally to Snowflake through Snowpark.
  - Test your application as a local Streamlit web app.

- **Deploy as a Streamlit in Snowflake app**
  - Create a Snowflake database with a named stage.
  - Upload your Python and other app files into this stage.
  - Create a **STREAMLIT** object, mentioning the entry point file.
  - In Snowsight, start your new app in the new *Streamlit* tab.
  - Connect to Snowflake through *get_active_session()*
  - Continue editing, running, and testing the app in Snowsight.

Native App: Prepare and Upload

- ***Create and test first as a local Streamlit web app***
  - Create a script.sql file, to prepare data on the consumer's side.
  - Create a readme.md file, for the first info page of the app.
  - Create a manifest.yml file, pointing to the two previous files.

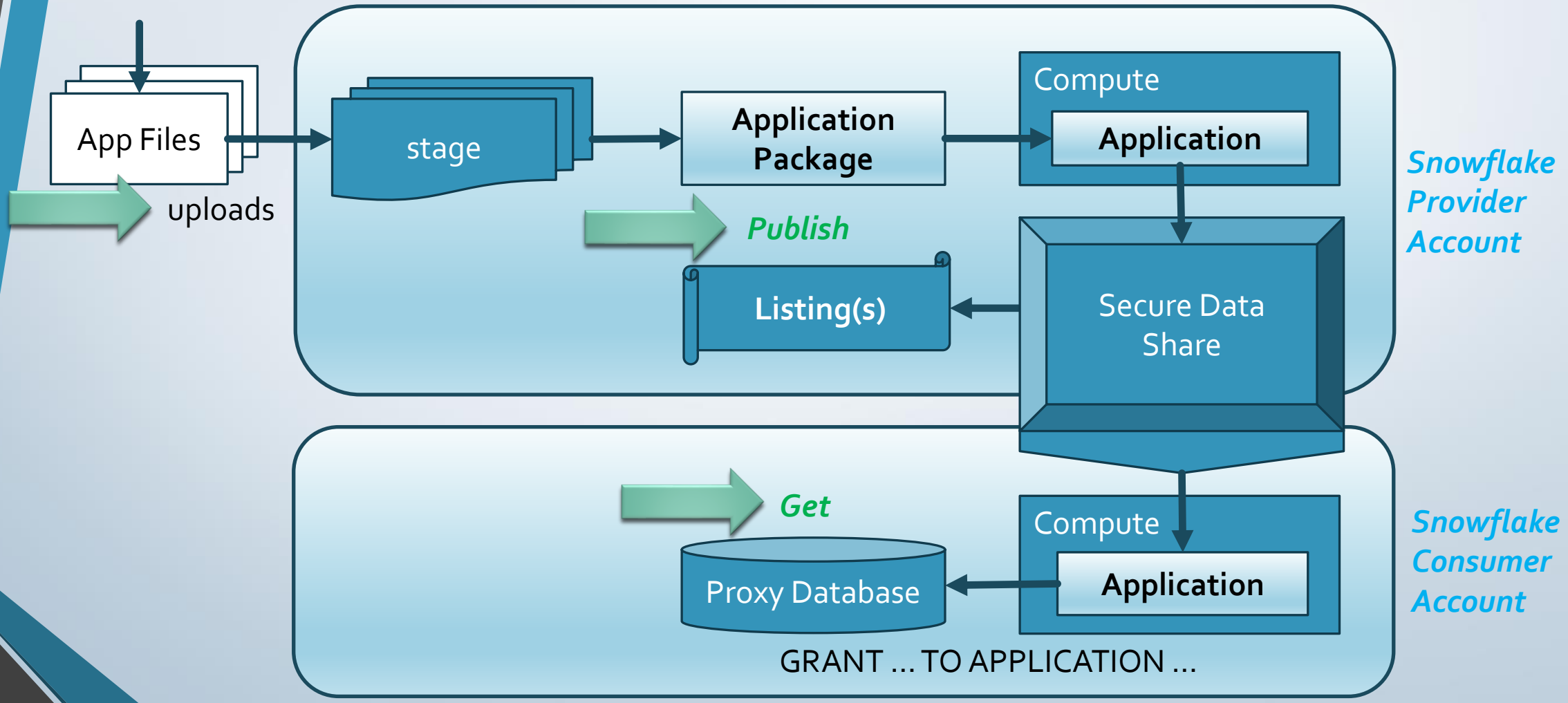
- ***In Snowflake***
  - Upload all your app files into a **named stage**.
  - Create an **APPLICATION PACKAGE** with the files uploaded in the stage.
  - Create an **APPLICATION** for this package.
  - Create a **STREAMLIT** object for the code.

Native App: Test and Deploy

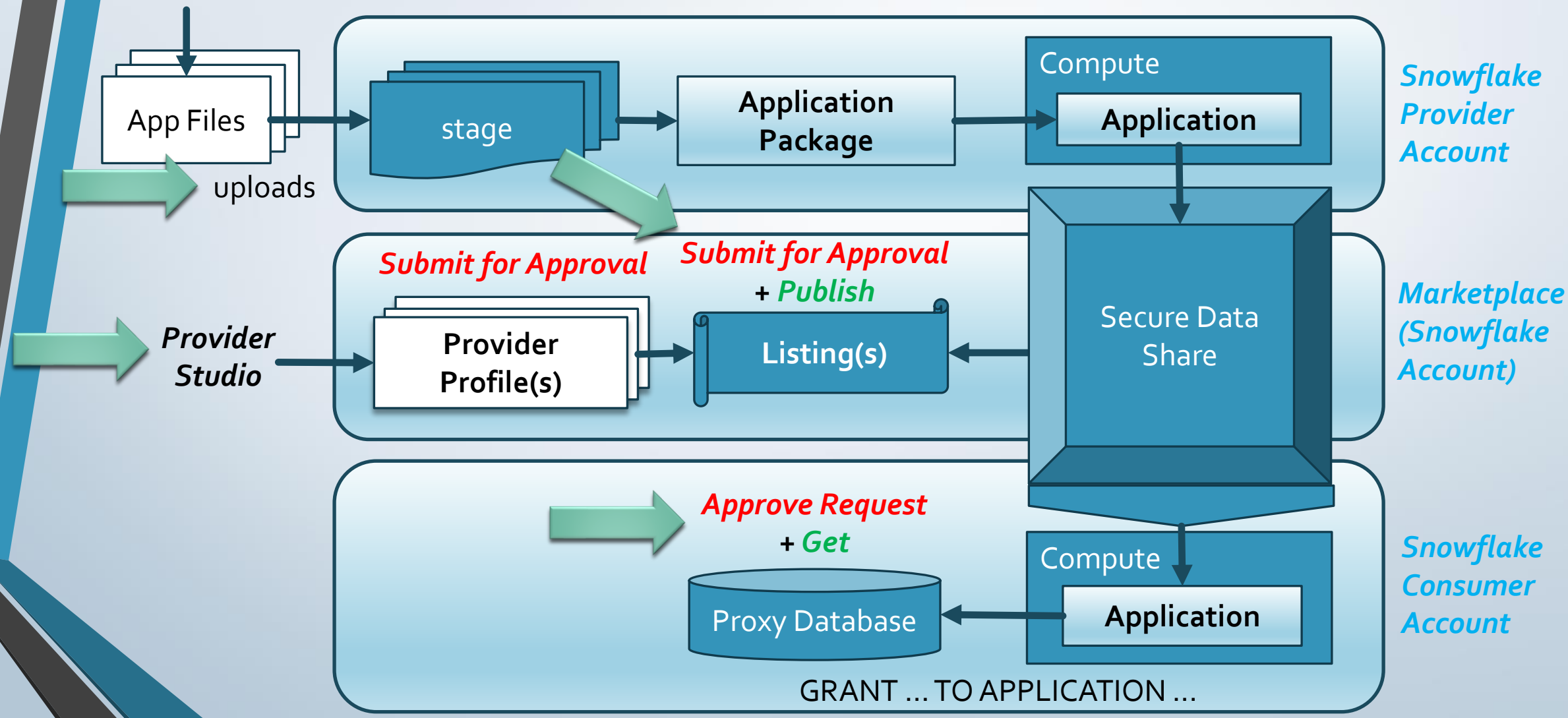
- ***In Snowsight***
  - Start your new app in the new *Apps* tab.
  - Connect to Snowflake through *get_active_session()*
  - Continue editing, running, testing the app in Snowsight, as a producer.

- ***In the Marketplace/Data Exchange*** → *public/private share*
  - Create [and get approved by Snowflake] a **provider profile**.
  - Publish your app [and get approved in the Marketplace] as a Native App.

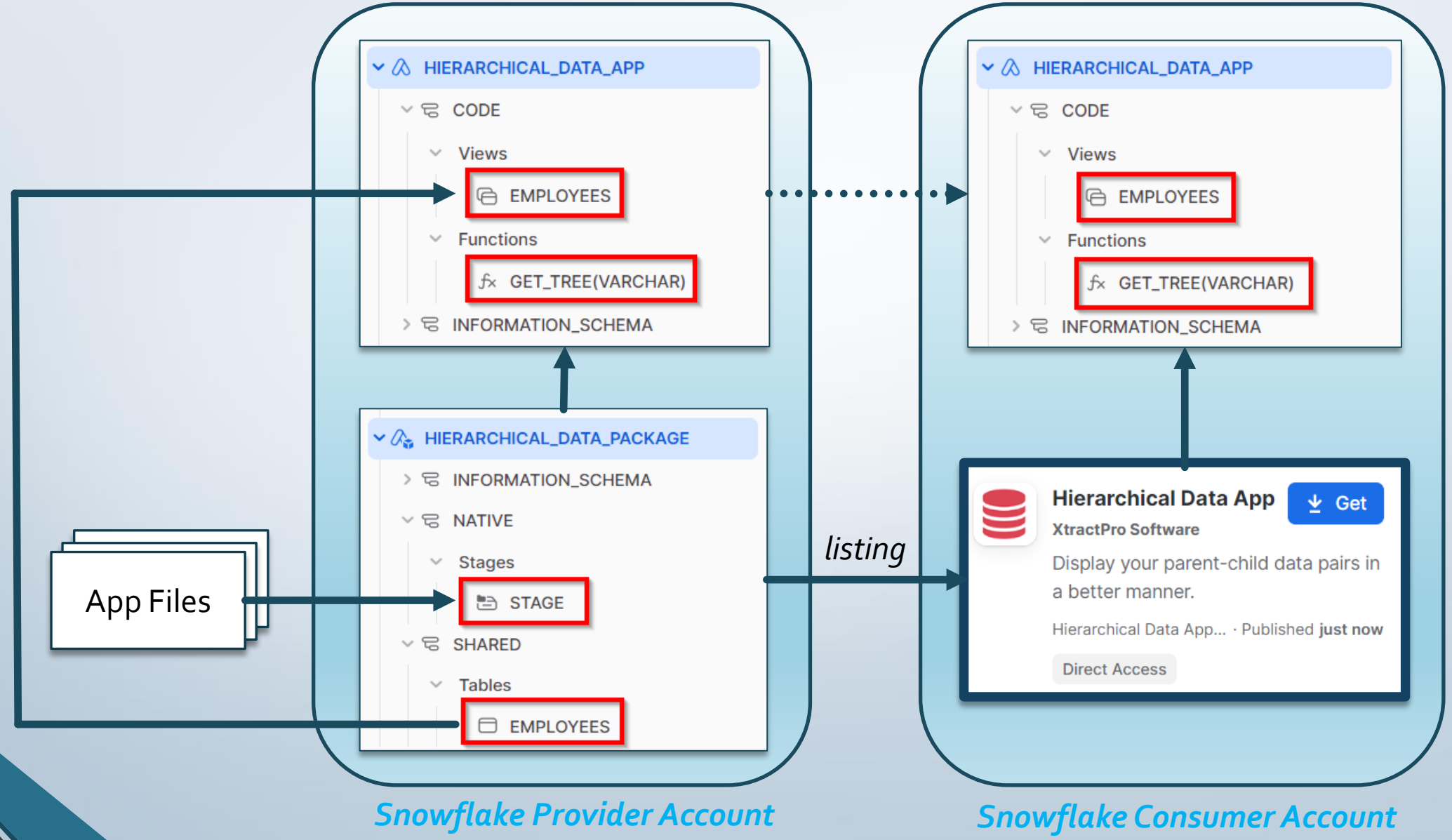
Native App: Private Share (Data Exchange)



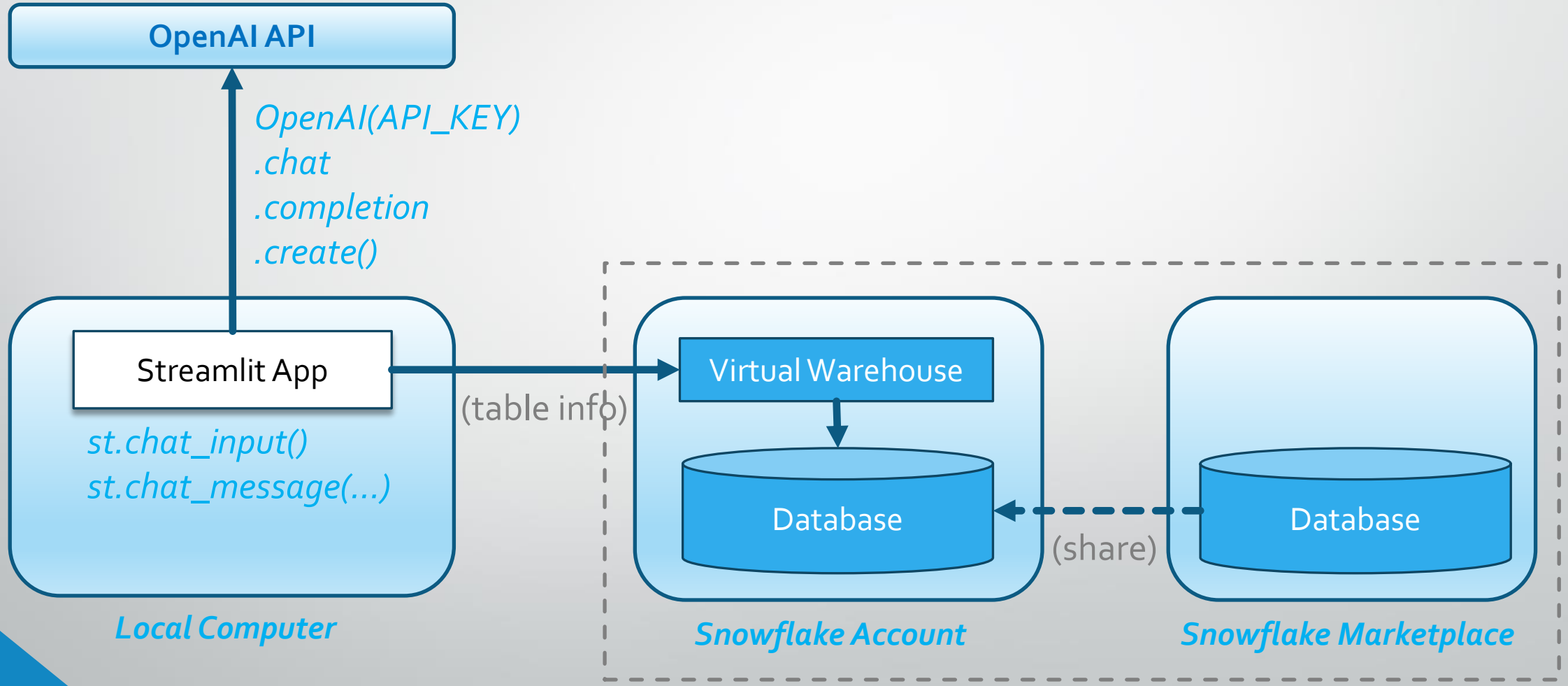
Native App: Public Share (Snowflake Marketplace)



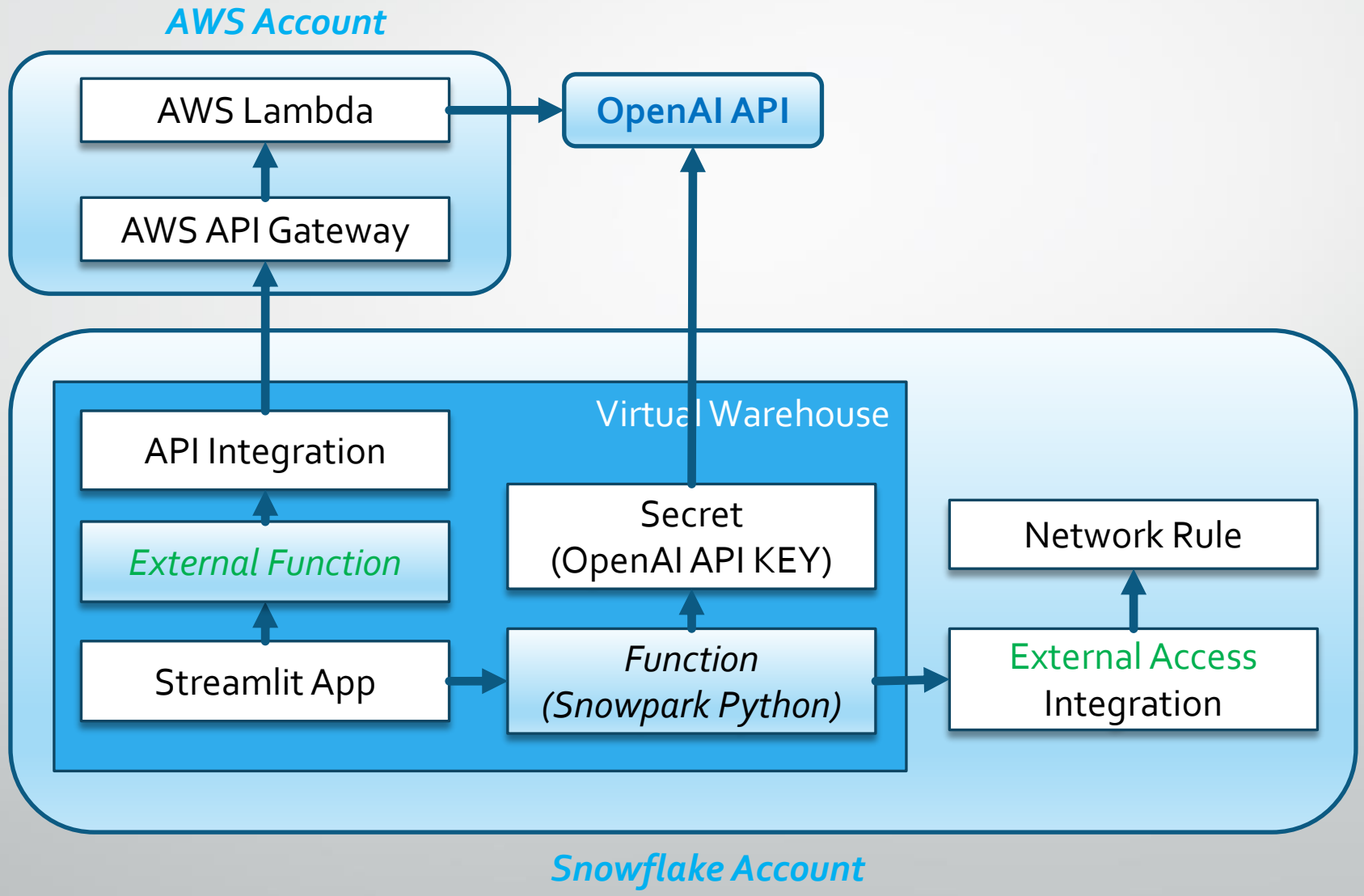
Native App: Provider-Consumer



OpenAI's ChatGPT Bot



OpenAI Access: w/ External Access/Function



Streamlit for Snowflake

Python Development of Streamlit Web Apps, Streamlit Apps and Snowflake Native Apps