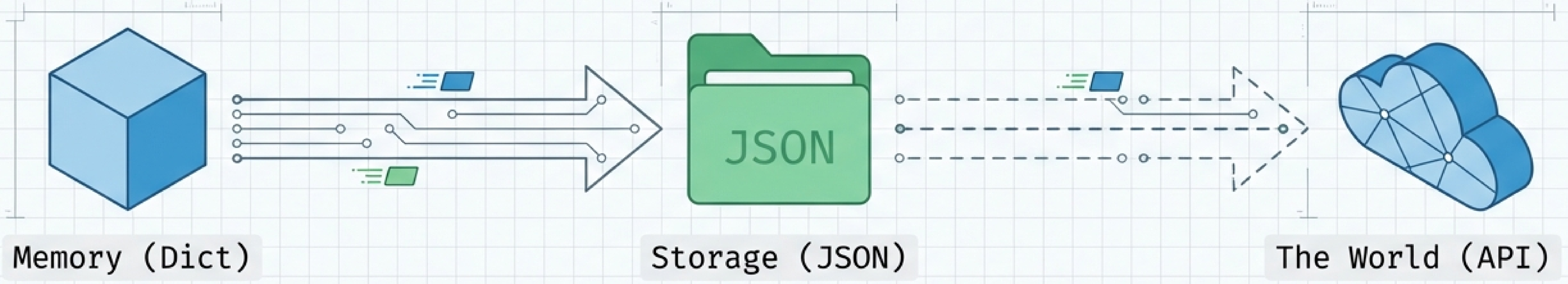
Python Data & APIs

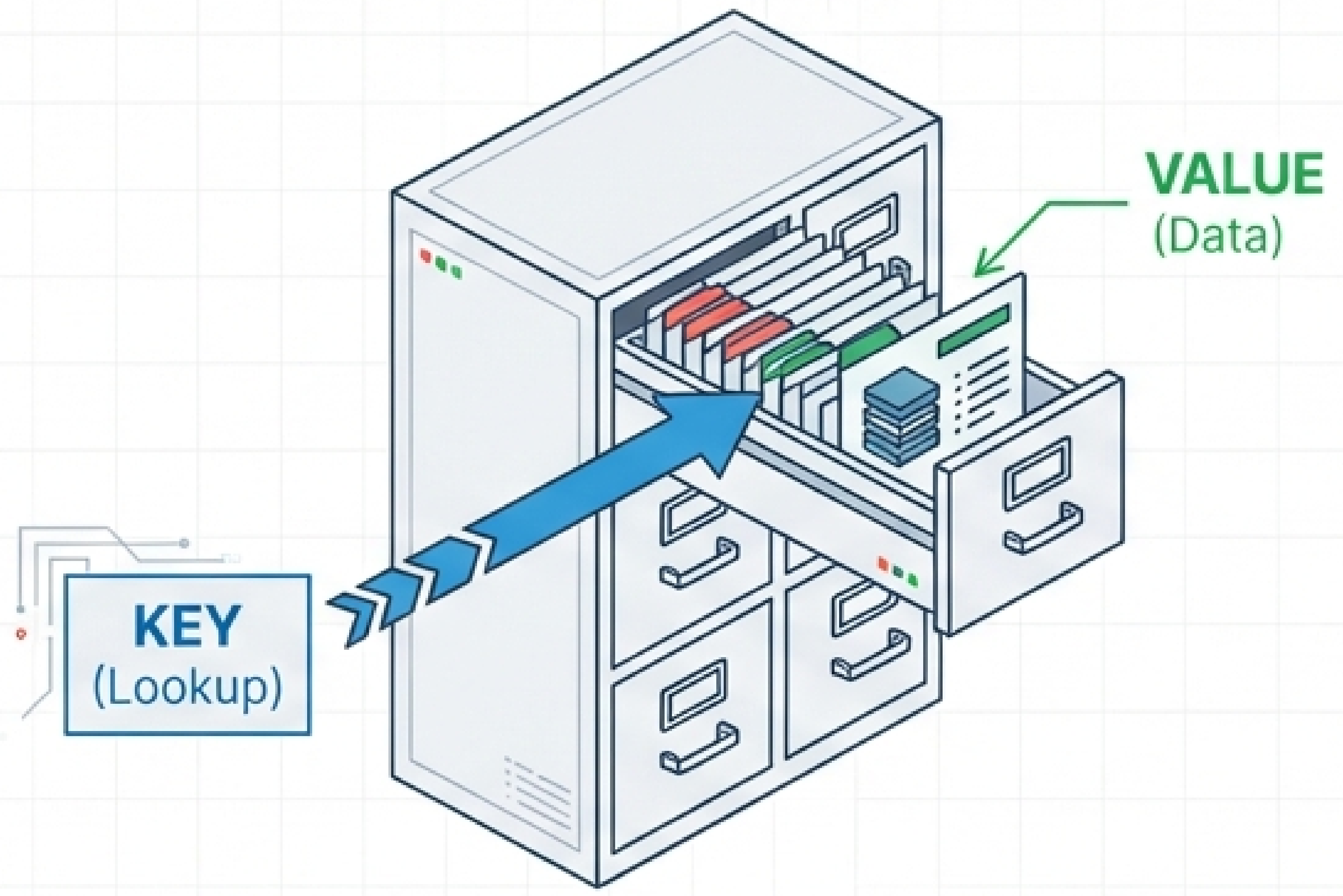
From Local Files to Web Requests



1. Dictionaries: The Structure
2. JSON: The Language
3. Files: Storage & Safety
4. Exceptions: Resilience
5. Requests: The Bridge to the World

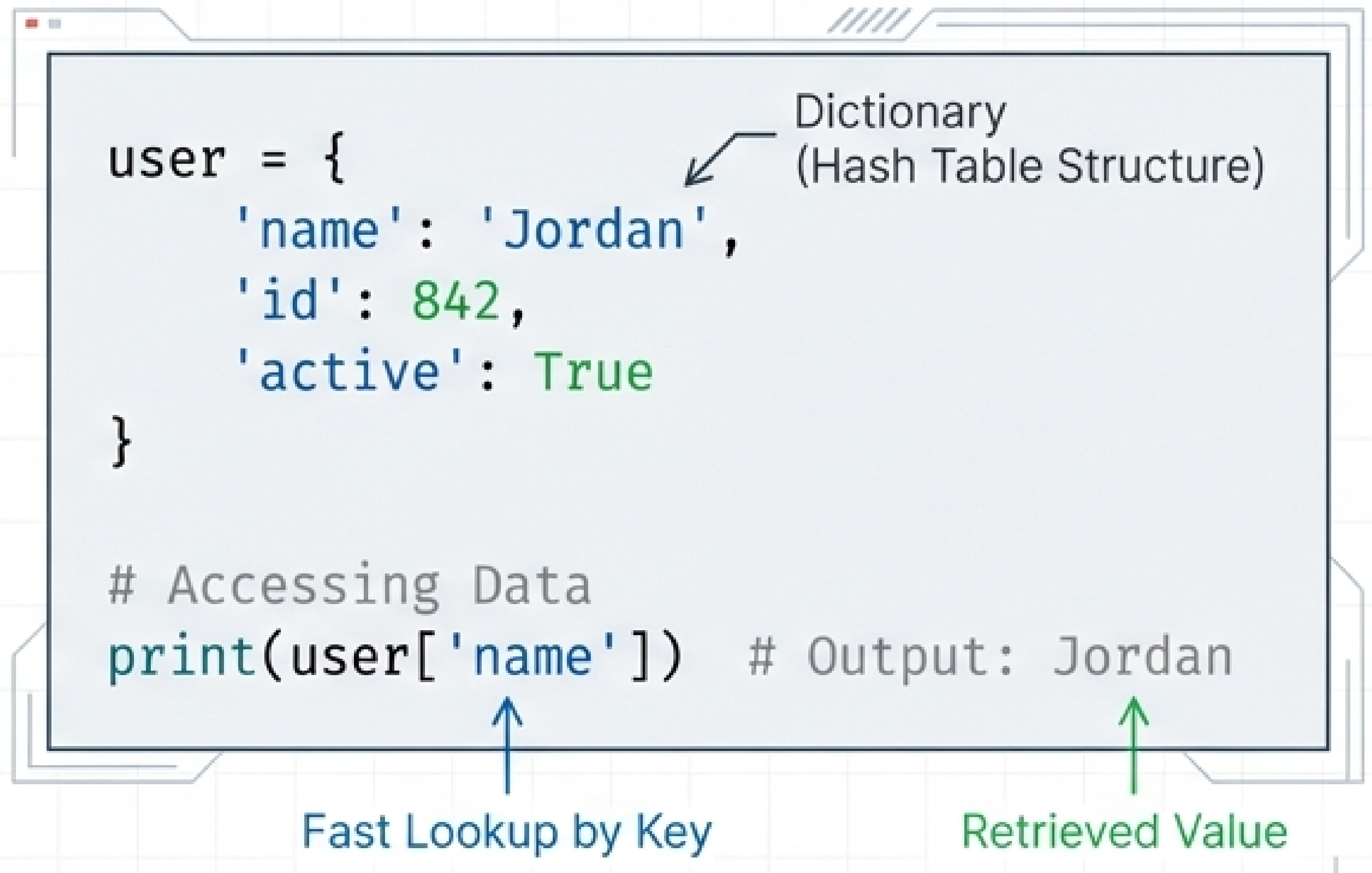
The Dictionary: Python’s Power Structure

CONCEPT



A physical card catalog drawer or lookup table mechanism, demonstrating the key-to-value mapping process.

CODE



```python
user = {
    'name': 'Jordan',
    'id': 842,
    'active': True
}

# Accessing Data
print(user['name'])  # Output: Jordan
```

KEY TAKEAWAY: Dictionaries use hash tables for incredibly fast data lookup. They are the backbone of structured data.

JSON: The Language of APIs

JavaScript Object Notation

Python Syntax vs. JSON Syntax

Python Syntax

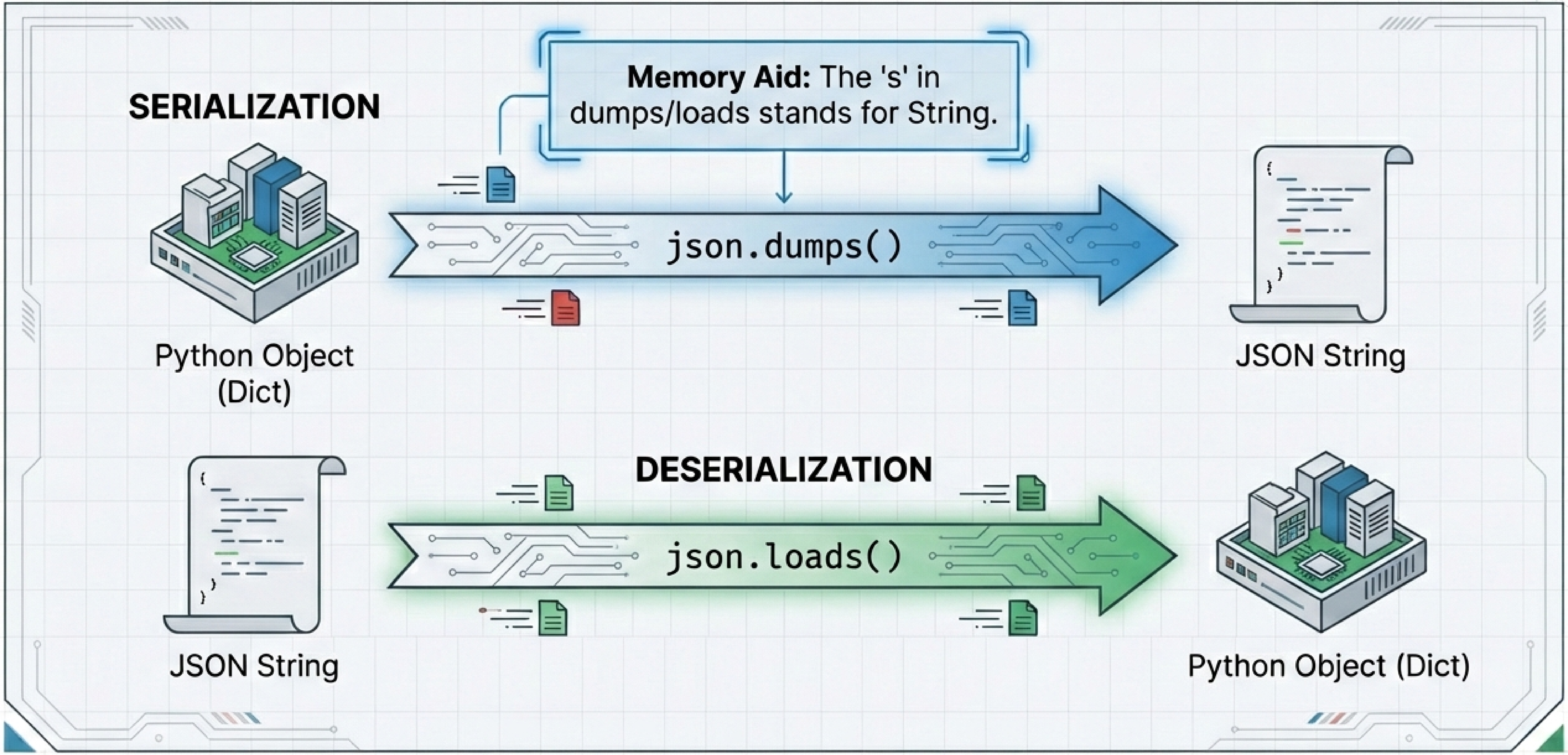
```
user = {
    'name': 'Jordan',
    'id': 842,
    'active': True,
    'data': None
}
```

JSON Syntax

```
{
  "name": "Jordan",
  "id": 842,
  "active": true,
  "data": null
}
```

Insight: JSON is a text string. Python Dictionaries are memory objects.

Serialization & Deserialization



NotebookLM

File Handling: The Basics

Mode	Description
'r'	Read (Default)
'w'	Write (Overwrites)
'a'	Append (Adds to end)

```
file = open('data.txt', 'w')
file.write('Hello World')
file.close()  # CRITICAL: Manual Close
```

⚠ **Danger:** Failing to close files leads to resource leaks.

Context Managers: The Pythonic Way

The Risky Way

```
file = open('data.json', 'w')
try:
    file.write(content)
finally:
    file.close()
```

Fira Code Medium

Context Manager

The Safe Way

```
with open('data.json', 'w') as f:
    f.write(content)
# Auto-closes automatically
```

The 'with' statement handles setup and teardown automatically, even if errors occur.

Writing JSON to Files

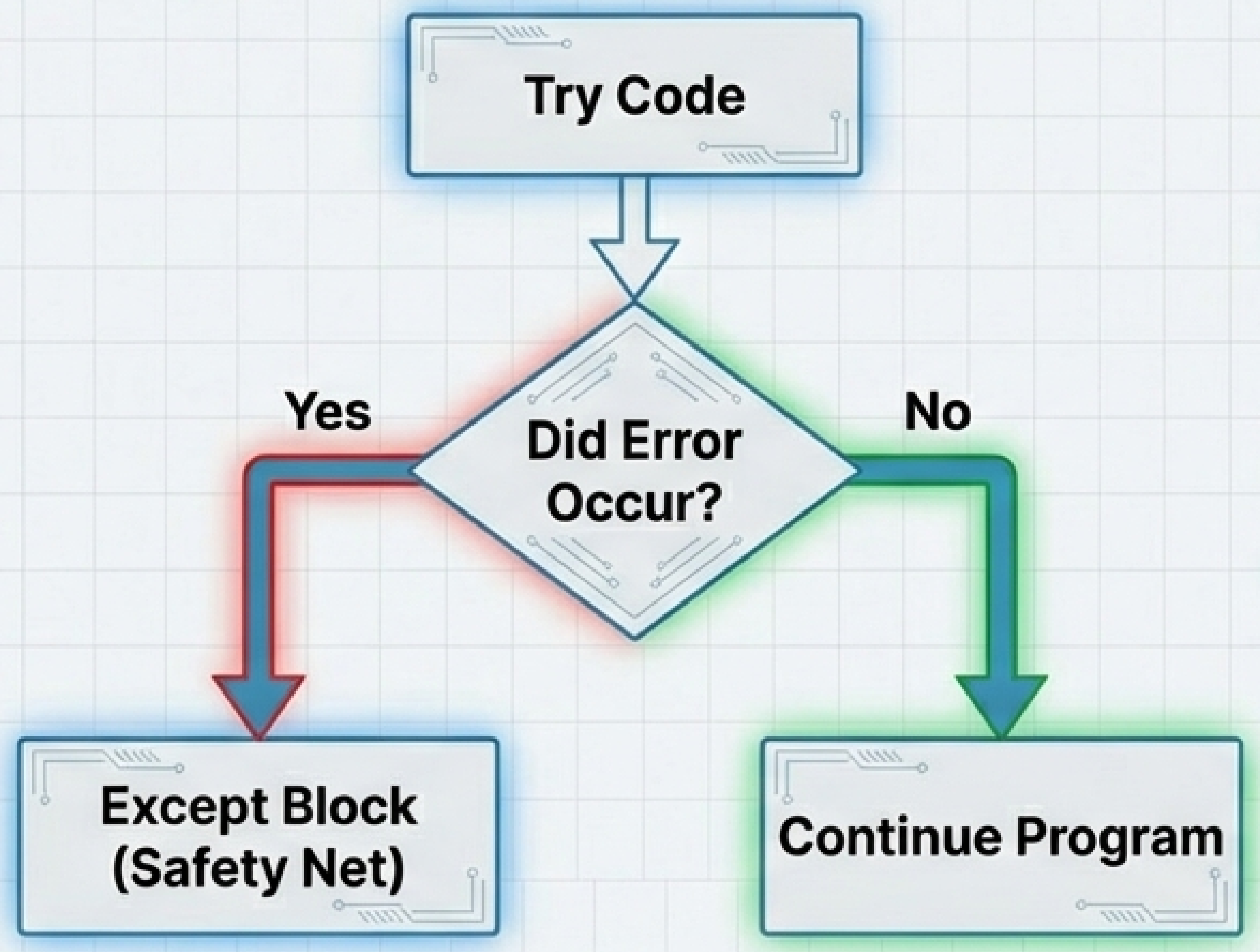
```
data = {'id': 1, 'status': 'active'}

with open('log.json', 'w') as f:
    json.dump(data, f)
```

Direct to File Object

Note: `json.dump()` (no `'s'`) writes directly to the file, skipping the string variable step.

Resilience: Asking for Forgiveness

Exception Handling with Try / Except



```
try:
    result = 10 / 0
except ZeroDivisionError:
    print('Cannot divide by zero!')
```

Fira Code Medium

Catching Specific Errors

`FileNotFoundError`
Missing files

`KeyError`
Missing dictionary keys

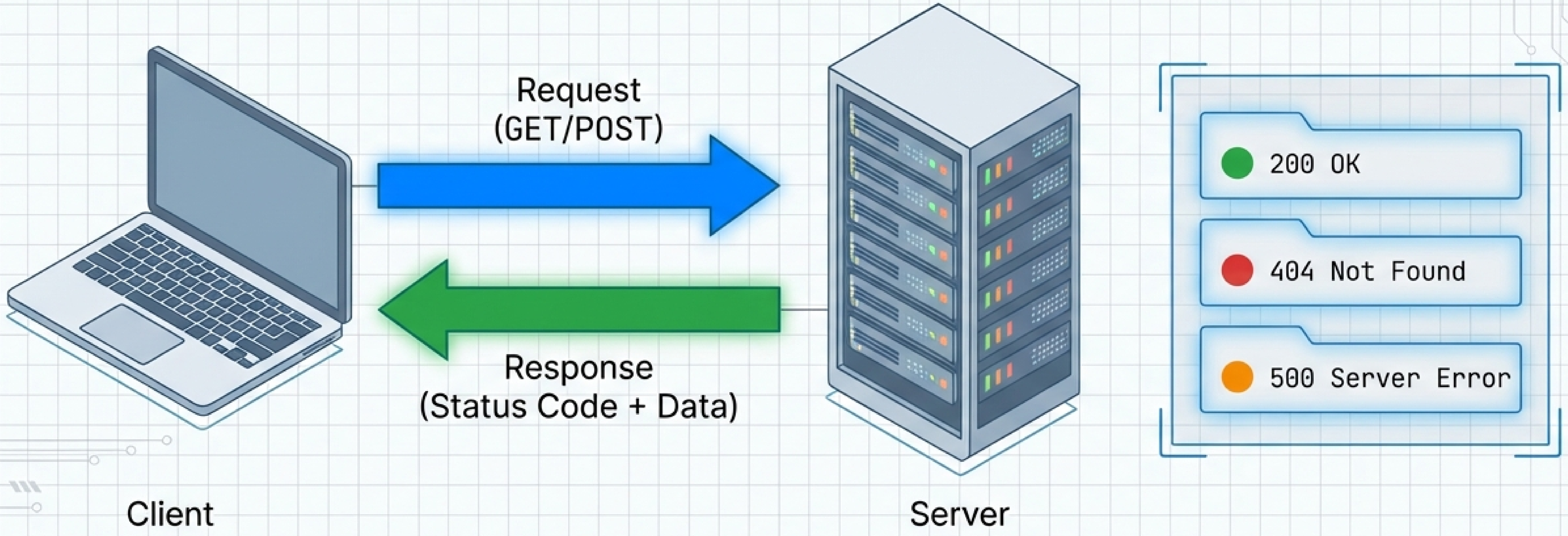
`json.JSONDecodeError`
Corrupt data

```python
try:
    data = json.loads(text)
except json.JSONDecodeError:
    print('Invalid JSON format')
```

Best Practice: Avoid bare **'except:'**. Only catch the errors you expect and can handle.

NotebookLM

HTTP Basics: Talking to the World



The requests Library

HTTP for Humans™

```
import requests

# Make the call
r = requests.get('https://api.example.com/data')

# Inspect Results
print(r.status_code)  # 200
print(r.text)         # Raw string
print(r.json())       # Python Dict
```

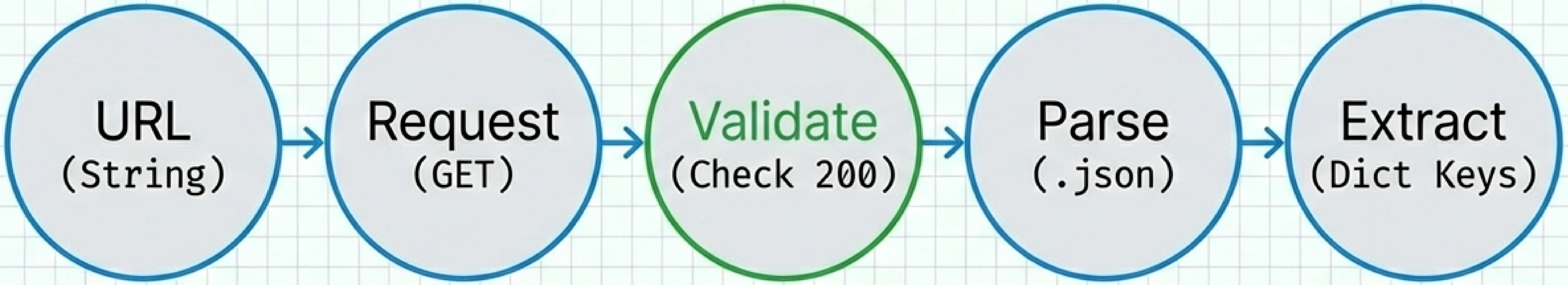
Auto-Deserialization

Making Robust API Calls

```python
try:
    r = requests.get(url, timeout=5)
    r.raise_for_status()  # Check for 200 OK
except requests.exceptions.HTTPError:
    print('Web error occurred')
except requests.exceptions.ConnectionError:
    print('No internet connection')
```

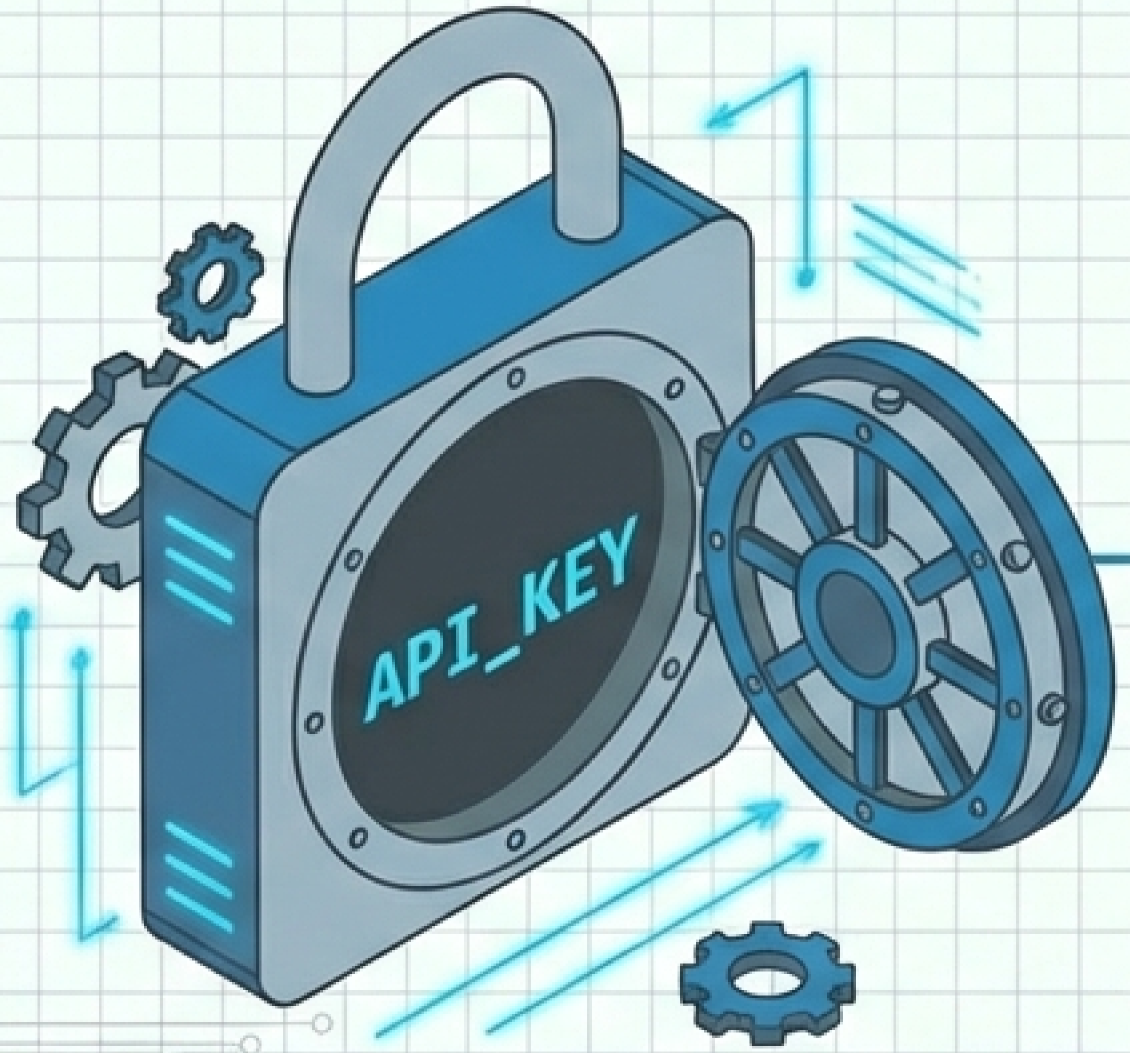
The Guard Rail

The API Workflow



String → Response Object → Python Dictionary → Specific Value

Authentication & Security



```python
import os

# Load from Environment
key = os.getenv('API_KEY')

headers = {'Authorization': f'Bearer {key}'}
requests.get(url, headers=headers)
```

Golden Rule: Never hardcode secrets in your script. Use Environment Variables.

Best Practices Checklist

- ✓ **Files**: Use `'with open(...)'` to prevent leaks.
- ✓ **JSON**: Use `'loads()'` for strings, `'load()'` for files.
- ✓ **Exceptions**: Catch specific errors (`FileNotFound`, `ConnectionError`).
- ✓ **Requests**: Always use `'raise_for_status()'`.
- ✓ **Security**: Keep secrets in Environment Variables.

Robust code handles data gracefully and fails safely.