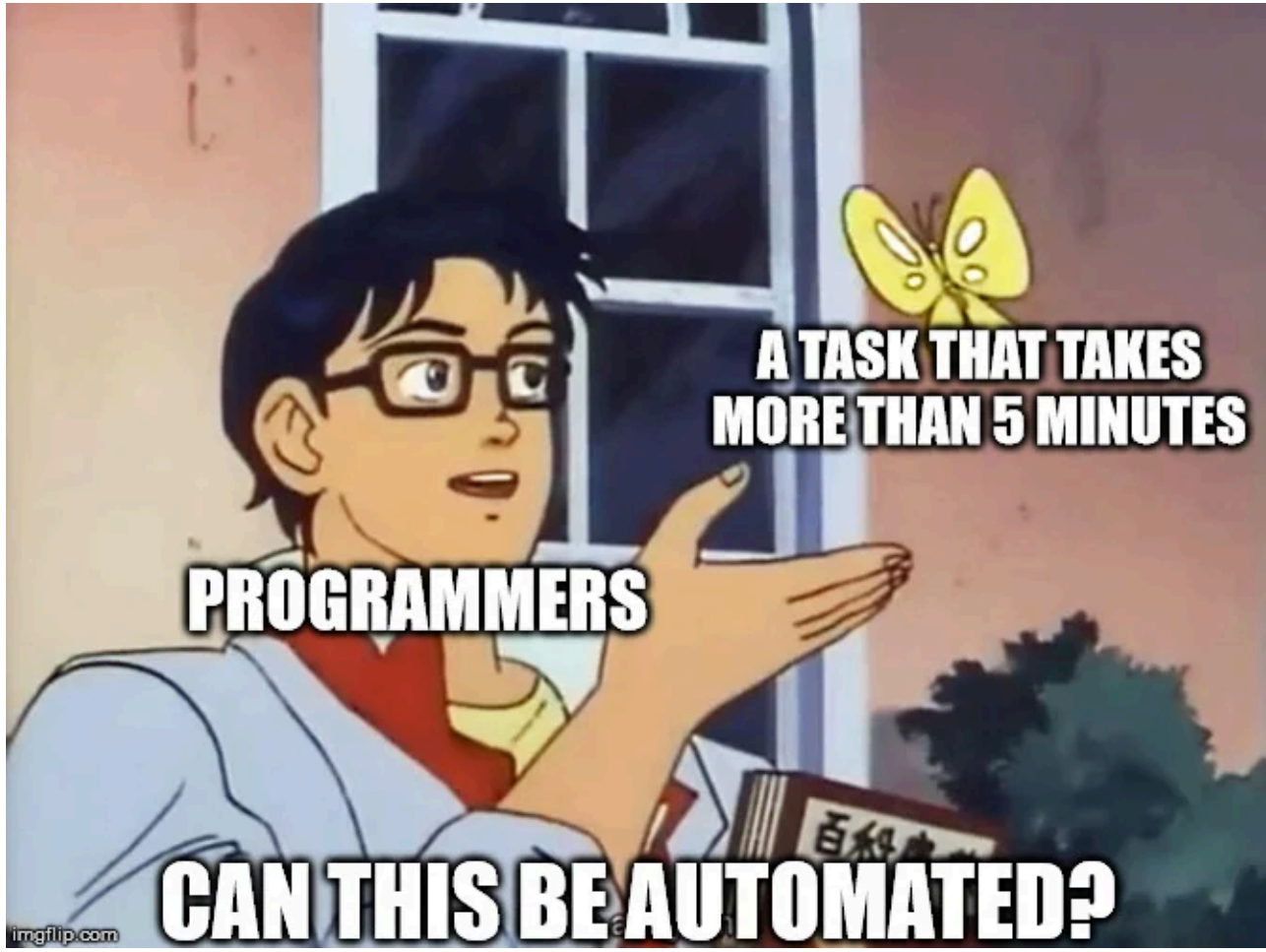
Baking Reliability into Everyday Python

Farid Nouri Neshat

🙋 Who is Farid?

- ✨ Senior Vibe Coder
- 💻 Freelancer Cloud Engineer
- 🔧 Software Engineer at heart
- 🤖 Automates all things!
- 🎸 Horrible guitar player

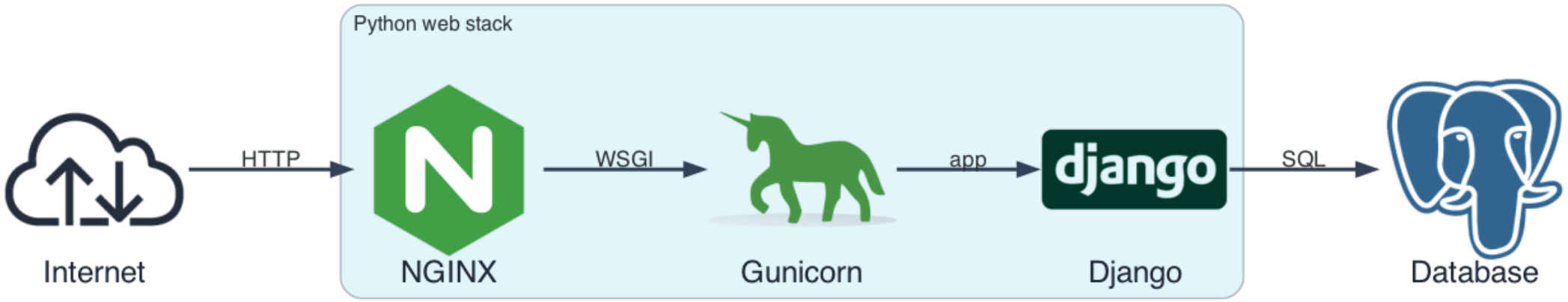


Complexity, usablity & fit for purpose

Everything fails!

- Your code has bugs!
- APIs will rate-limit you.
- The network will hiccup.
- Your users will send you garbage data.

Resource & Limits



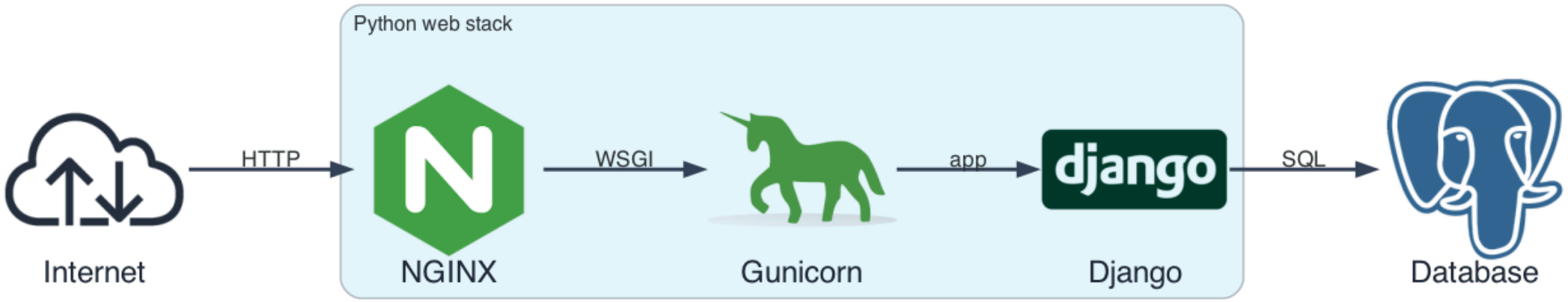
Resources

- Memory
- CPU
- Processes/Thread pools
- Connections
- Ports
- Open Files
- Bandwidth

Limits

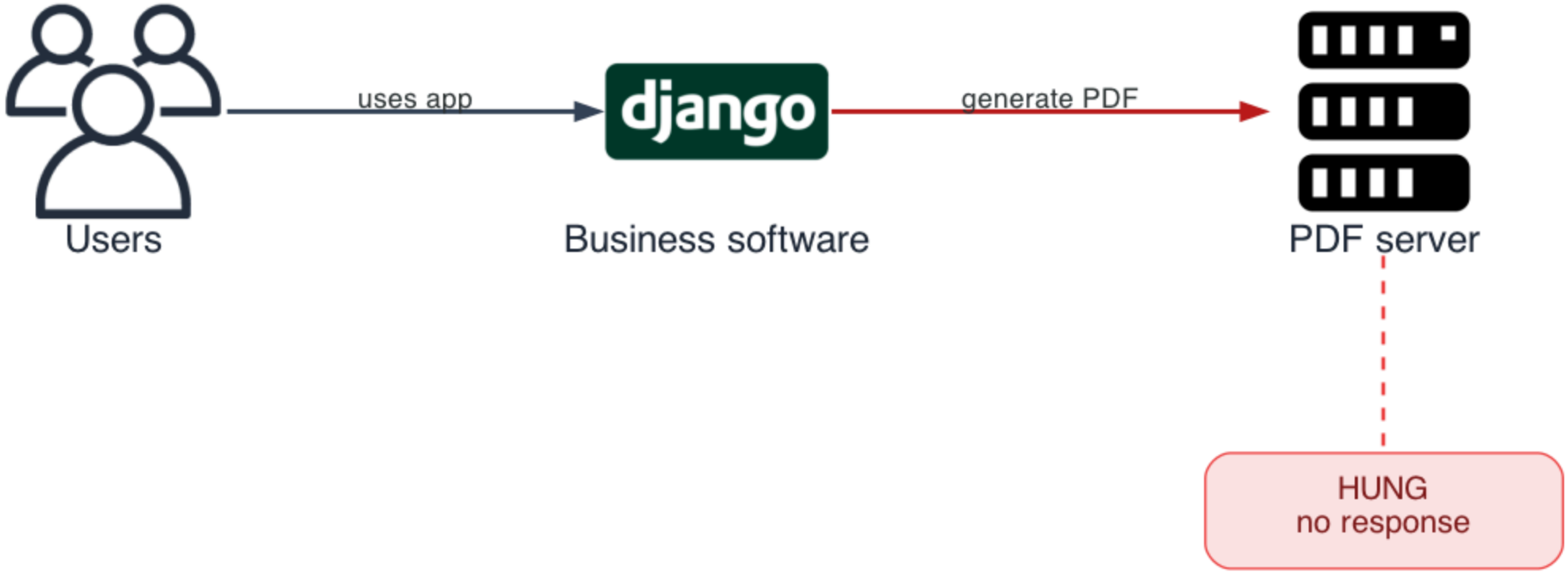
- Memory
- CPU
- Connection Limits (Max connections)
- Timeouts (Connect, Read, Write, DB Statement, user's patience)
- Queue Sizes (e.g., Listen queue, Gunicorn backlog)
- Pool Sizes (Processes, Threads, Connections)
- Memory Limits / OOM Controls
- File Descriptor Limits (`ulimit`)
- Rate Limiting
- Payload / Body Size Limits
- Ephemeral ports limit

Resource & Limits

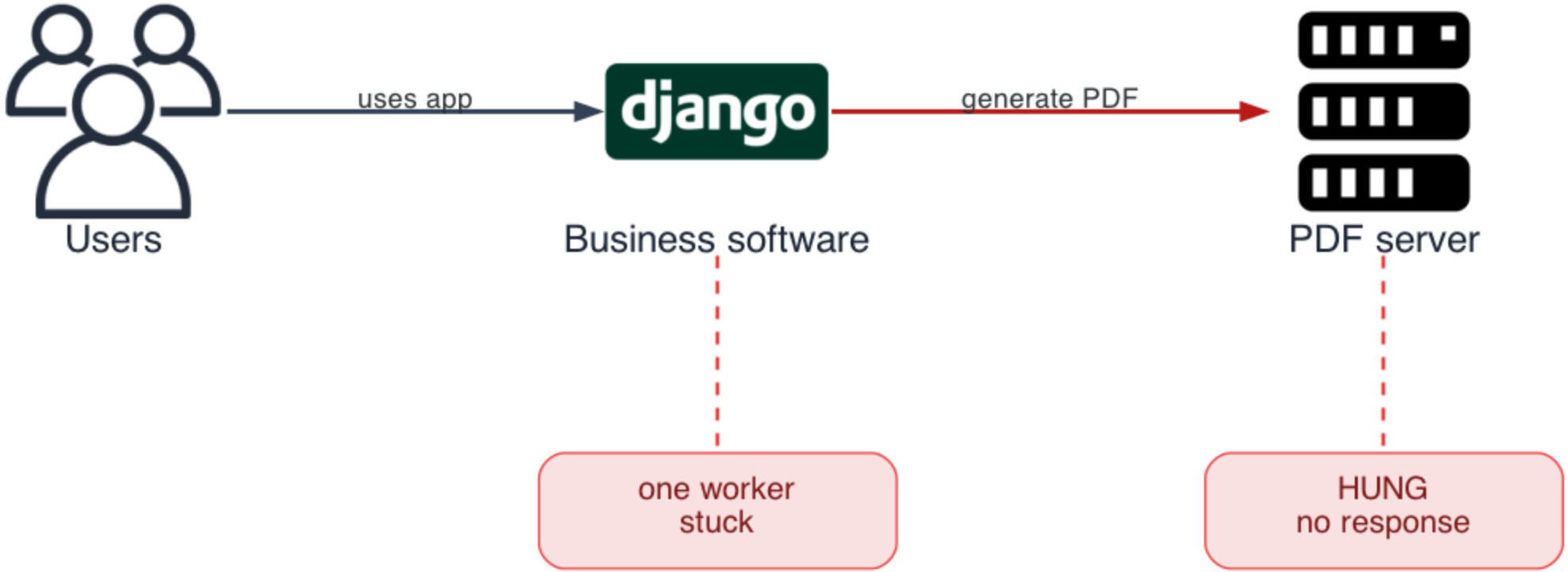


Shady PDF server!

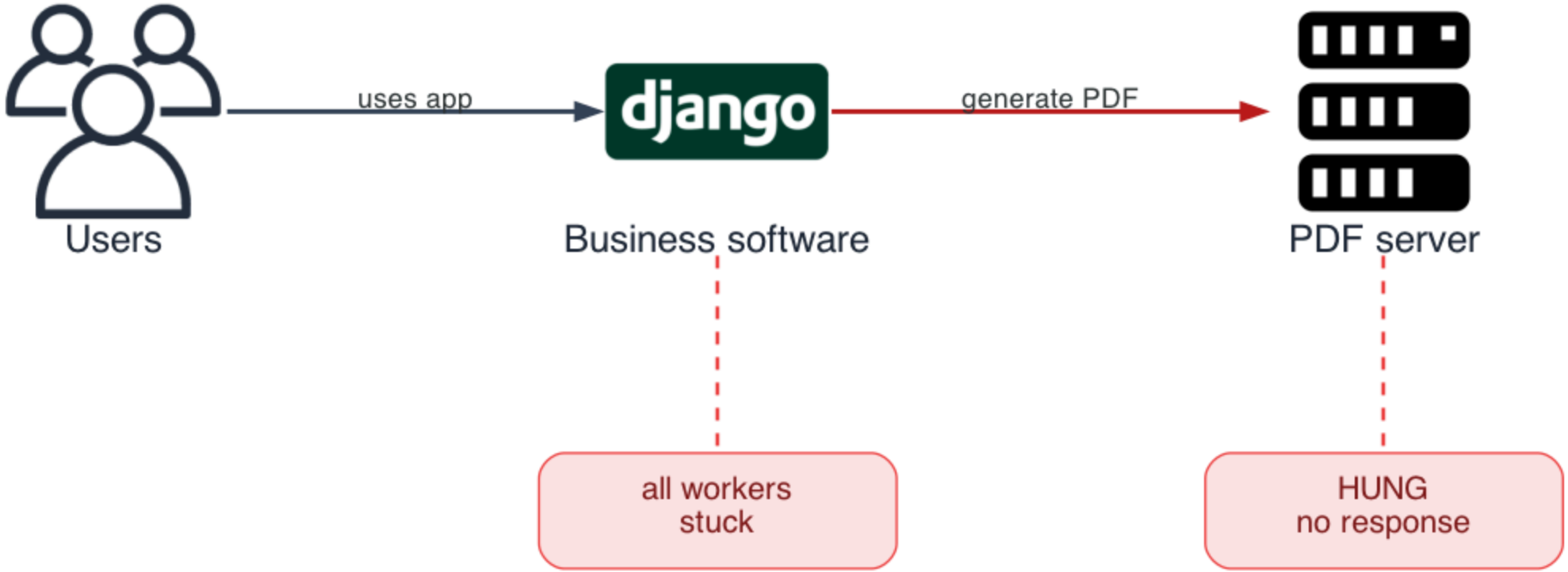
Shady PDF server!



Shady PDF server!



Shady PDF server!




IN TIME
OCTOBER 28

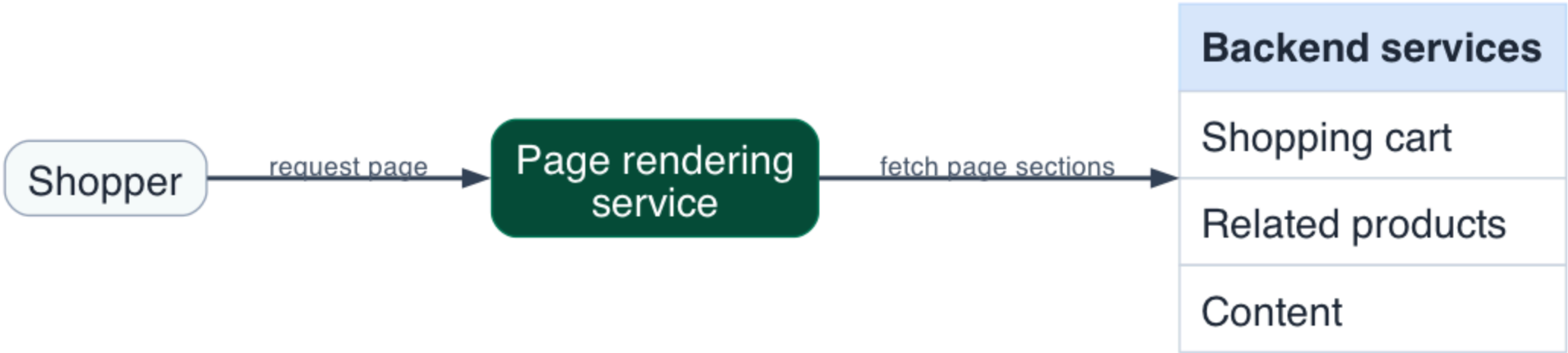
Always bet on timeouts!

```
import requests

response = requests.get(
    'https://api.pdf-server.internal/generate',
    timeout=5  # fail after 5 seconds!
)
```

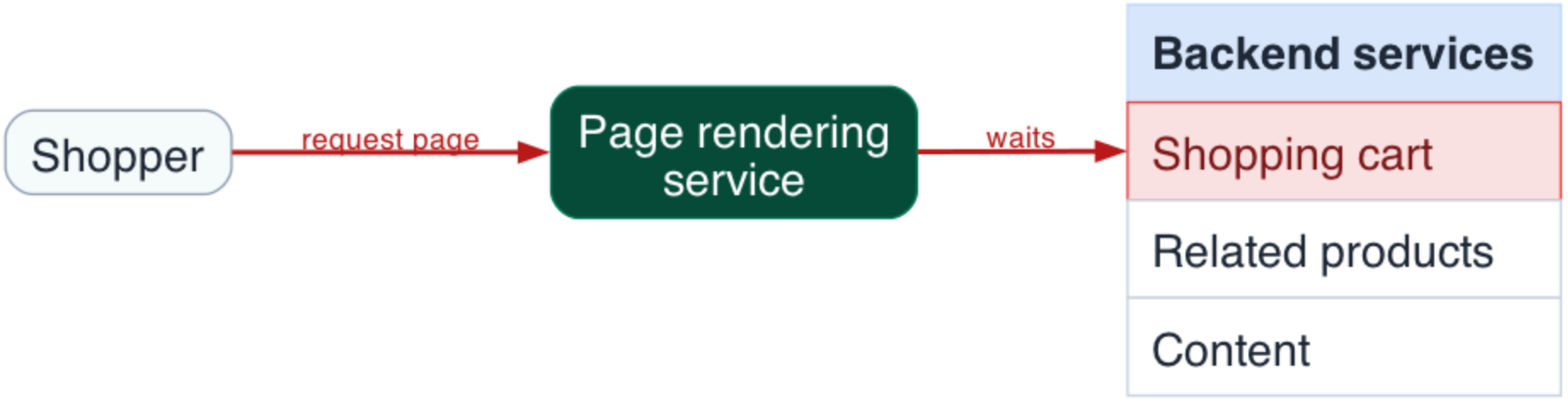
Retries & Idempotency

```
import requests
from tenacity import retry, stop_after_attempt, wait_exponential

@retry(
    stop=stop_after_attempt(3),
    wait=wait_exponential(multiplier=1.1, min=2, max=10)
)
def generate_pdf(payload: dict, idempotency_key: str):
    response = requests.post(
        'https://api.pdf-server.internal/generate',
        json=payload,
        headers={'X-Idempotency-Key': idempotency_key},
        timeout=5  # Timeout triggers a retry!
    )
    response.raise_for_status()
    return response
```

Ecommerce page render!



Ecommerce page render!



Circuit Breakers & Degradation

If it's already broken, stop hitting it.

```
from circuitbreaker import circuit
import requests

@circuit(failure_threshold=5, recovery_timeout=30)
def get_recommendations(user_id):
    # If this fails 5 times, it stops trying for 30 seconds
    return requests.get(f'http://recommendations/{user_id}', timeout=2)

def render_homepage(user_id):
    try:
        recs = get_recommendations(user_id)
    except CircuitBreakerError:
        recs = get_static_fallback_products() # Graceful degradation!
```

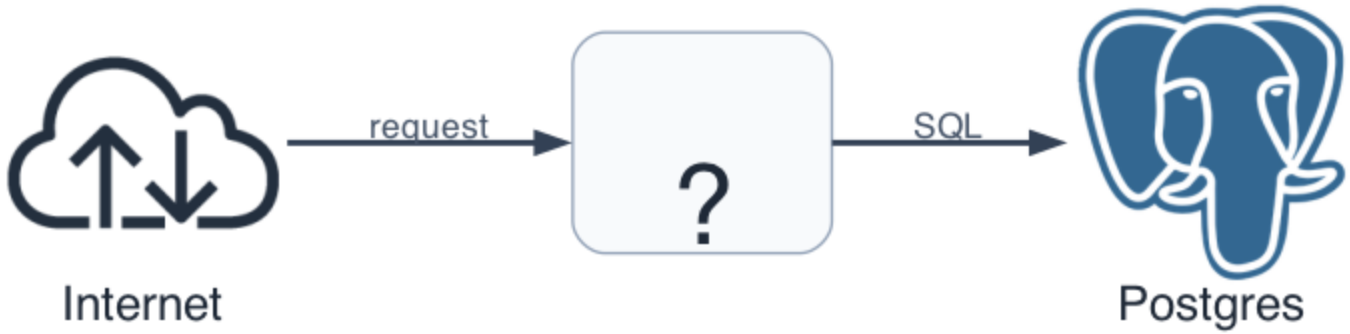
Memory Monster

```
def copy_api_response_to_s3(bucket, key):
    result = api.get_all_pages():
    with open(f'./tmp_file', 'wb') as fout:
        csv_writer = csv.DictWriter(fout, fieldnames)
        csv_writer.writerows(result)
    s3.copy('./tmp_file', f's3://{bucket}/{key}')
```

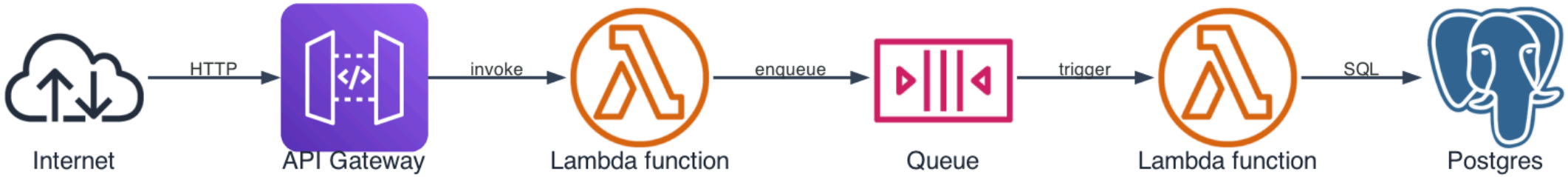
Constant Memory Under Load

```
from smart_open import open

def copy_api_response_to_s3(bucket, key):
    with open(f's3://{bucket}/{key}', 'wb') as fout:
        for page in api.pages():
            csv_writer = csv.DictWriter(fout, fieldnames)
            csv_writer.write(page)
```

How do ingest millions of datapoints per day without losing any?



Queue-based processing



Common themes & Takeaways

- Code in isolation is easy, culprit shows up in inter-process communication!
- Be aware of the limits and ensure they are aligned
- Set timeout at every stage!
- Don't retry too fast, too slow or predictable
- Stream data one chunk at time, maintain backpressure by queuing

Link for slides, books, packages & refrences.