gRPConf
INDIA 2026

Beyond Hello World: gRPC in Python for Real Systems

- PALAK JAIN
Sr Software Engineer, HSBC

- STUTI JAIN
SDE 2, Adobe

18TH NOVEMBER, 2026

gRPConf
INDIA 2026

PALAK JAIN



Sr. Software Developer at HSBC

Speaker at PyCon Korea, Indonesia, Panama

Love to learn about ancient Indian culture

palak-jain-980622145

STUTI JAIN



SDE 2 at Adobe

Speaker at PyCon Korea, Indonesia, Panama

Enjoy sparking ideas and connecting dots

stuti-jain-98630a197

SCENARIO 1



int64 user_id

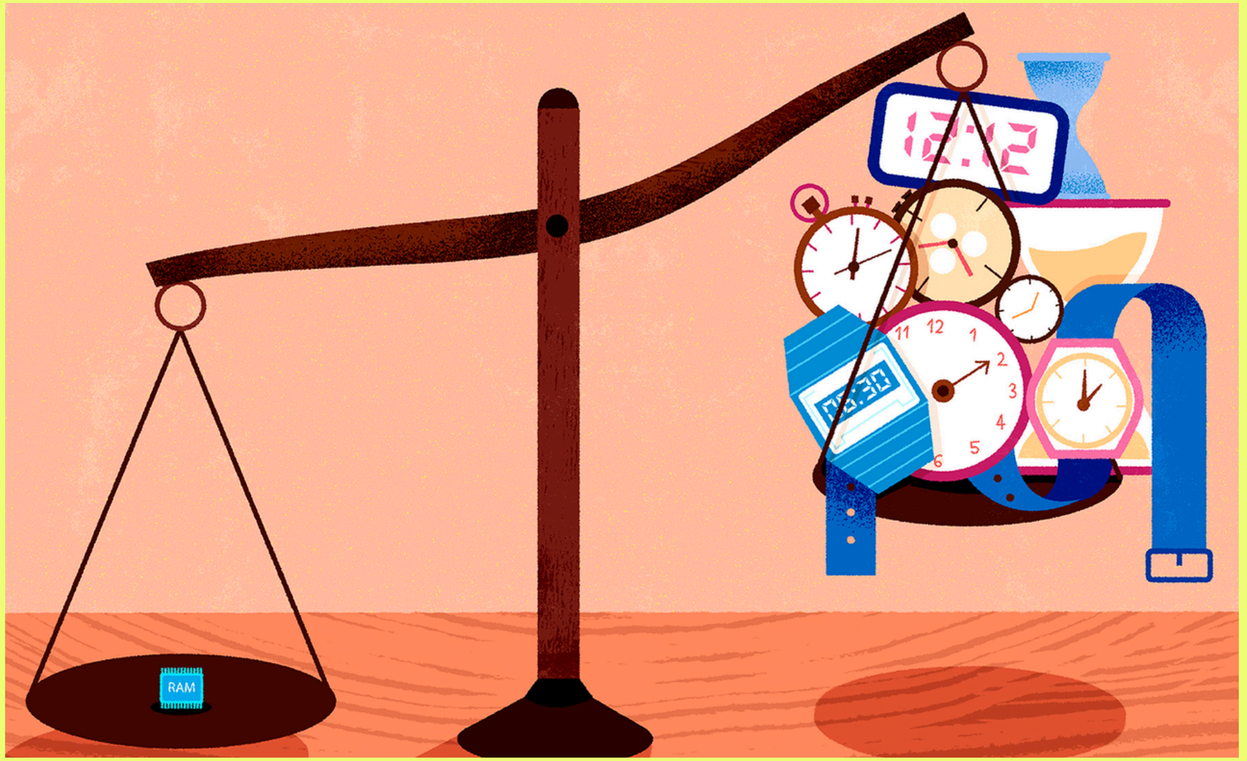


string user_id

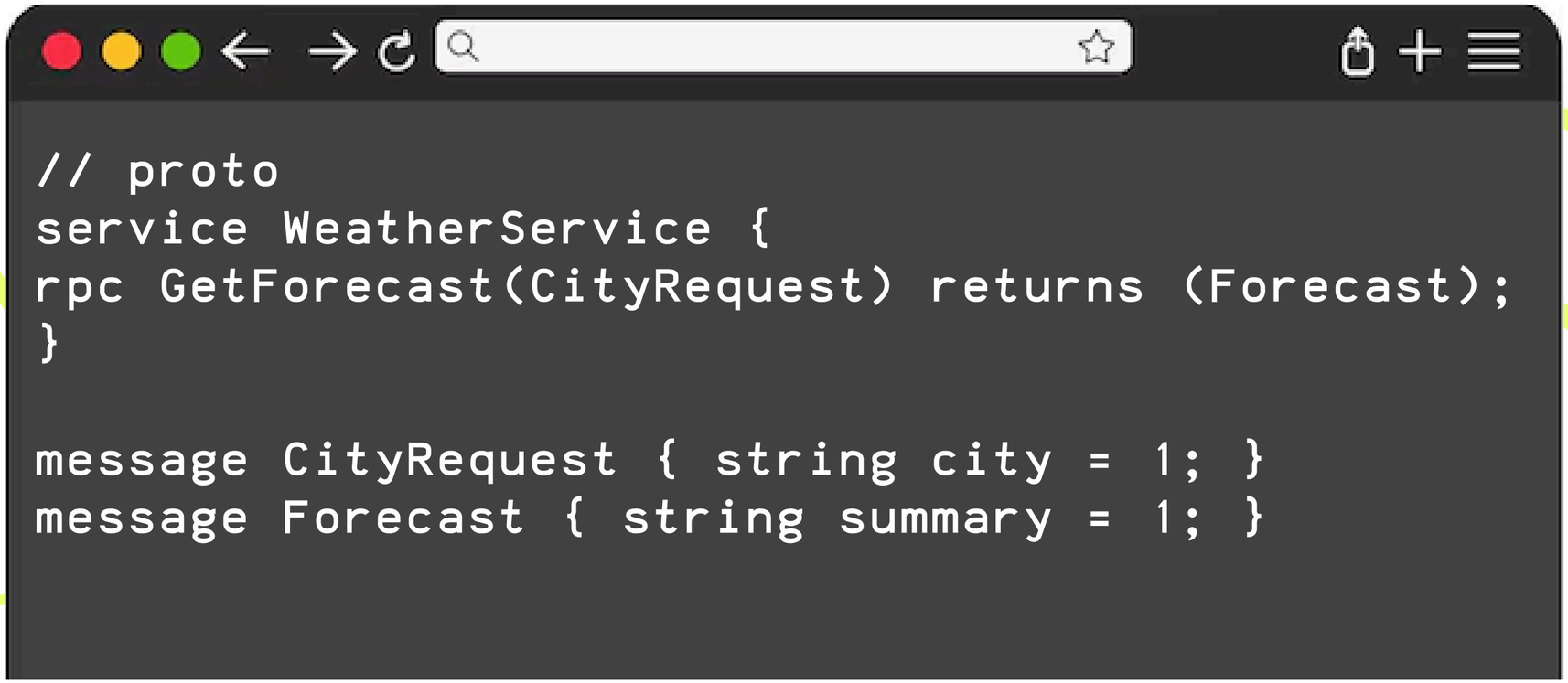
Output: Client v1 reads name field: ''

SCENARIO 2

```
Output: [CLIENT PID 56467] Received Token #438 | Buffer Count: 438
```



THE FUNCTION CALL ILLUSION



```
// proto
service WeatherService {
rpc GetForecast(CityRequest) returns (Forecast);
}

message CityRequest { string city = 1; }
message Forecast { string summary = 1; }
```

Looks local. Runs somewhere else.

Four Decisions Belong to Us

gRPC handles

- Serialization
- Framing and transport
- Generated interfaces
- Status codes

Your system handles

- Safe schema evolution
- Deadlines and retries
- Buffering and memory
- Process lifecycle

What happens when an old v1 client calls this new v2 server?



```
// v1 proto (OLD)

message UserProfile {
string name = 1;
}
```



```
// v2 proto (NEW)

message UserProfile {
int64 user_id = 1;
}
```

1) Reads the ID correctly
2) Crashes with an error
3) Status OK, but the value disappears

What happens when an old v1 client calls this new v2 server?



```
// v1 proto (OLD)

message UserProfile {
string name = 1;
}
```



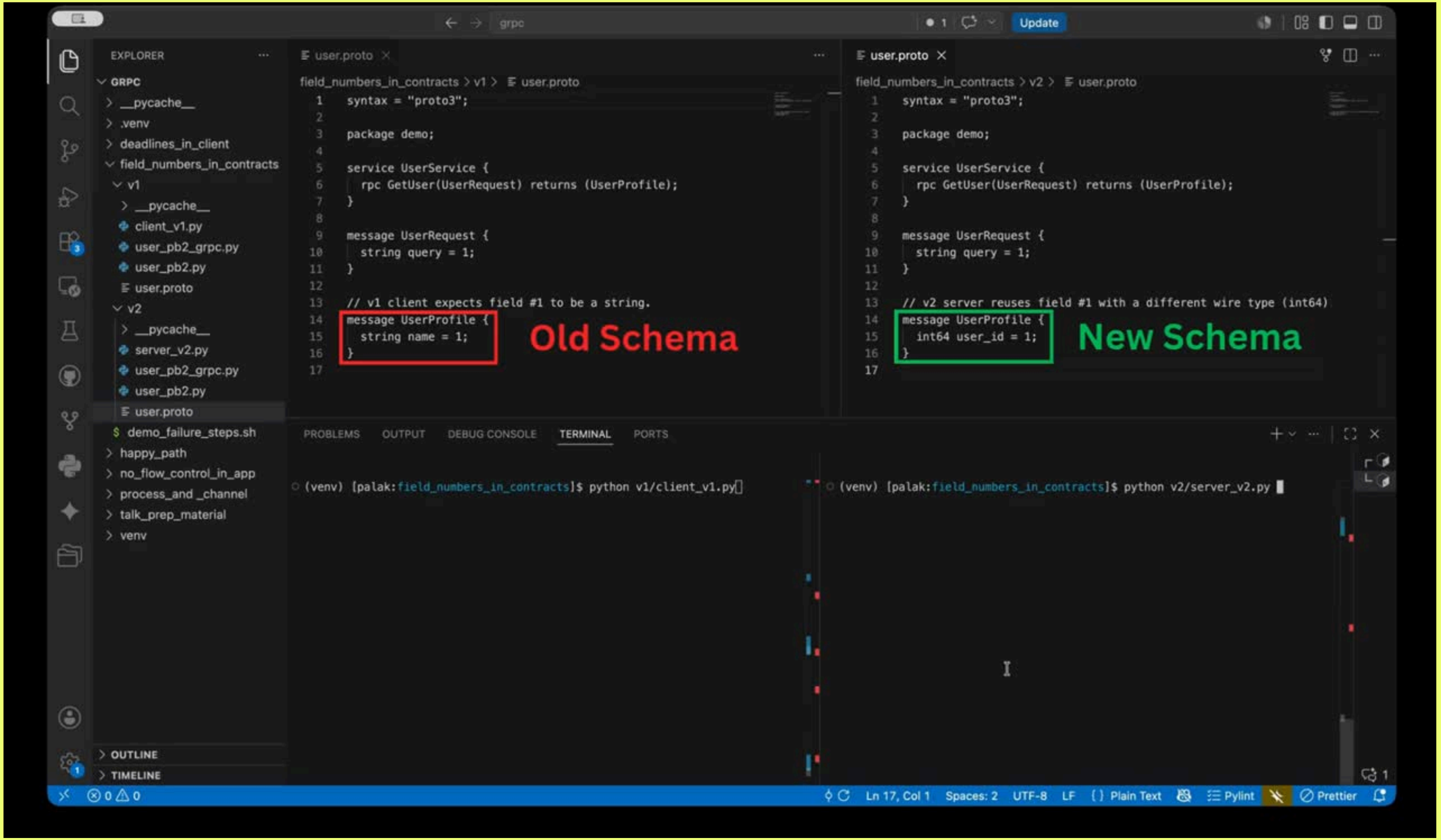
```
// v2 proto (NEW)

message UserProfile {
int64 user_id = 1;
}
```

1) Reads the ID correctly
2) Crashes with an error
3) Status OK, but the value disappears

gRPConf INDIA 2026

Live Demo: Silent Schema Break



RULE #1

Field Numbers Are Forever



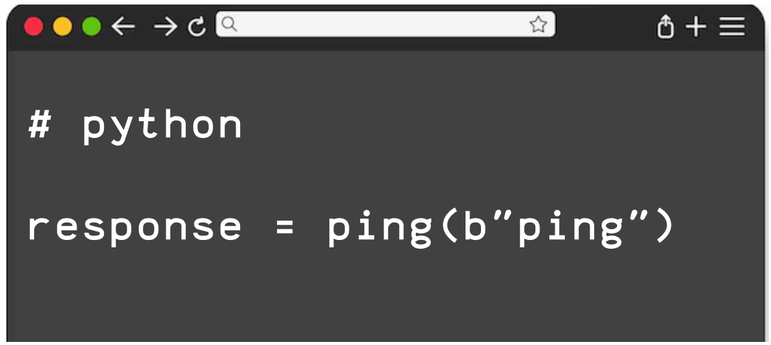
```
// proto

message UserProfile {
reserved 1;
reserved "name";
int64 user_id = 2;
}
```

Policy: Never reuse a field number.

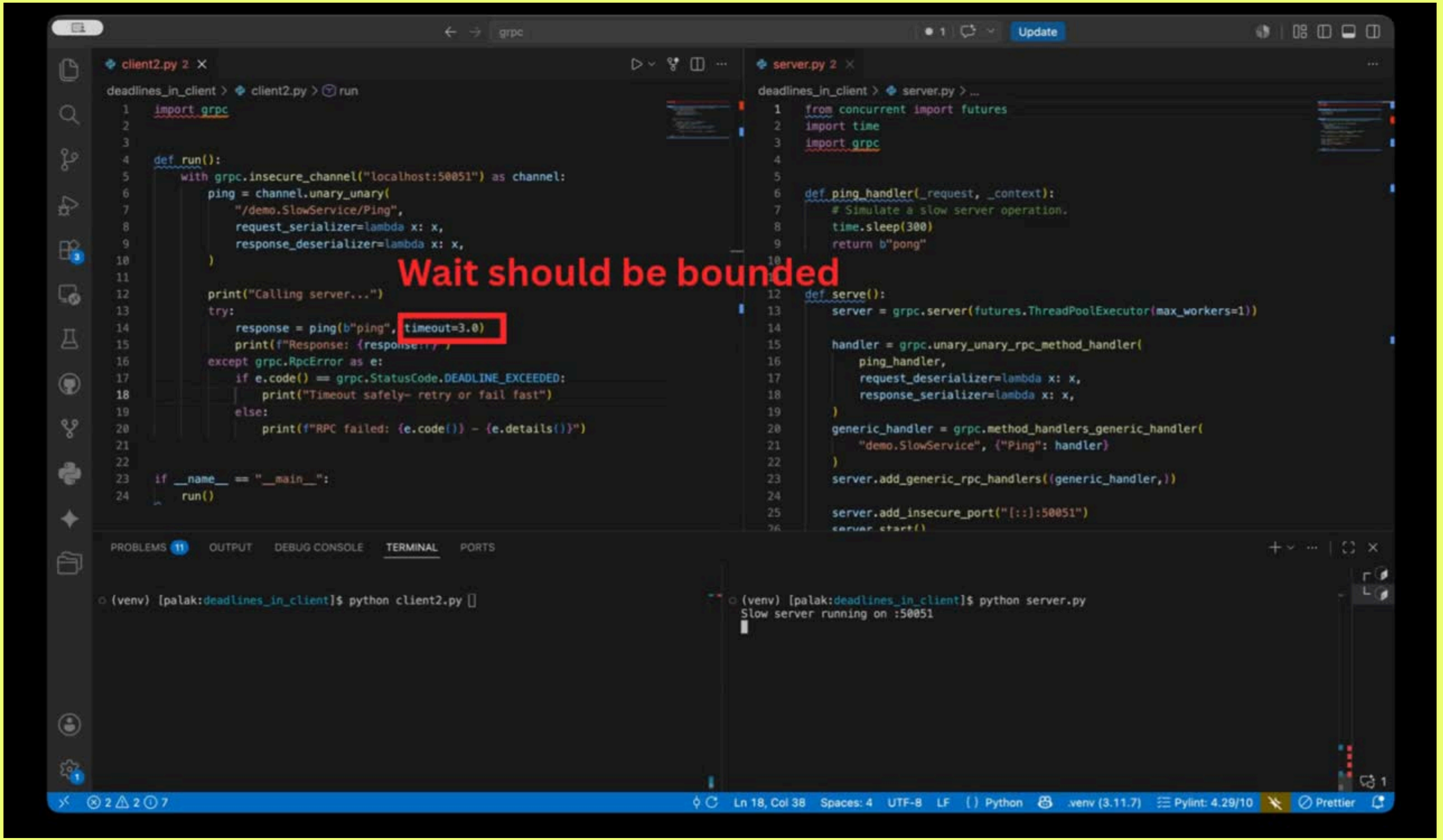
How Long Would You Wait?

client call -> waiting... -> waiting... -> ?



gRPConf INDIA 2026

Live Demo: Bound the Wait



RULE #2

Bound the Wait

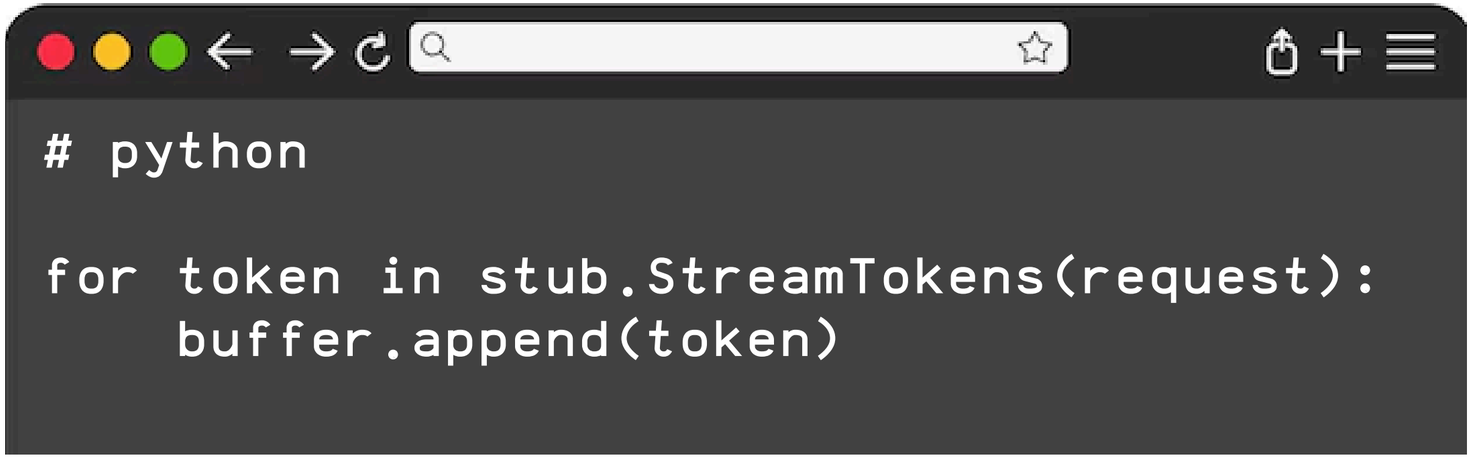
A deadline returns control. It does not make retry safe.

Deadline -> Cancellation -> Retry decision

Only safe +
transient +
within budget

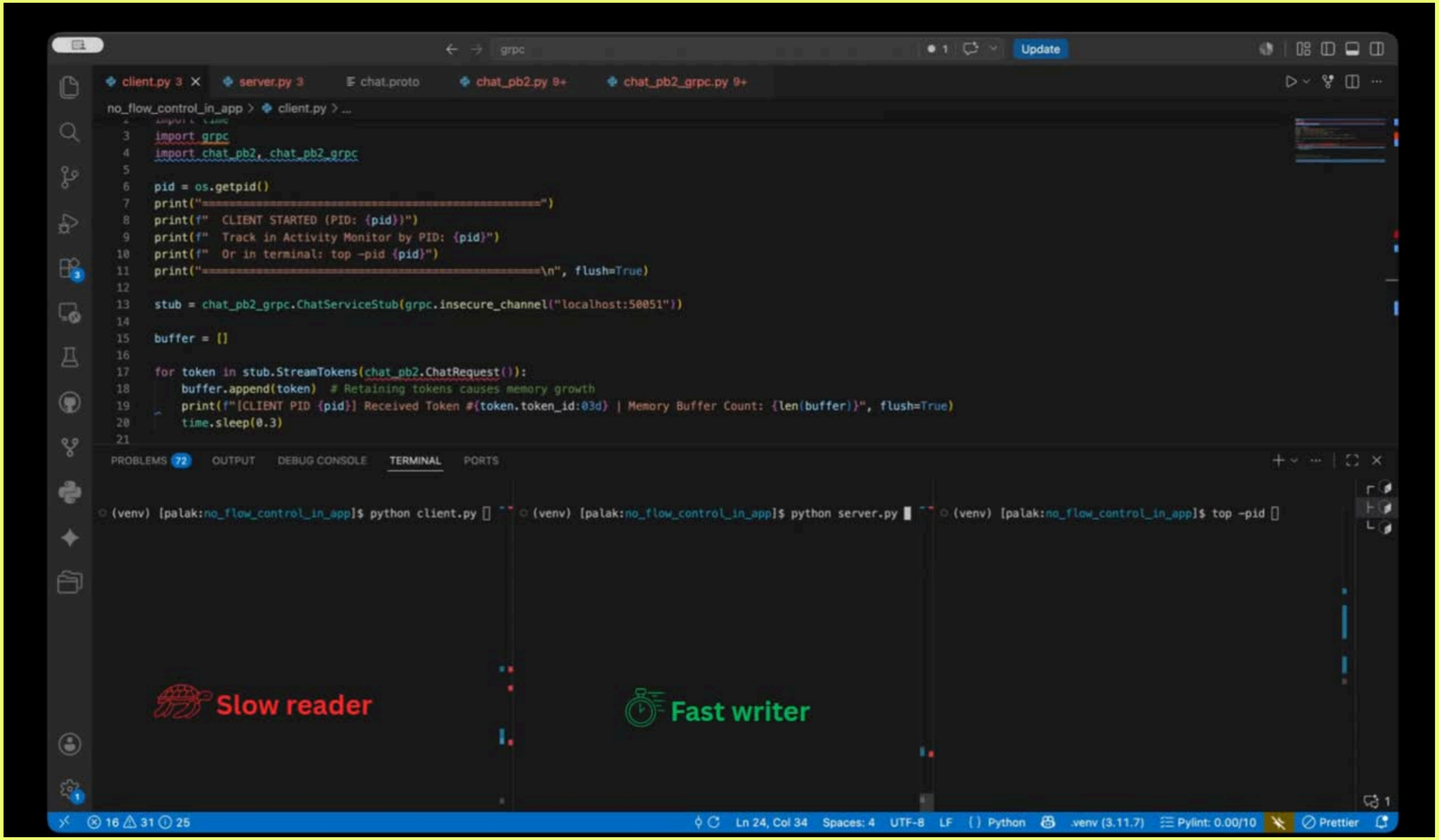
Who Owns This Memory?

server -> HTTP/2 flow control -> Python iterator -> buffer = []



gRPConf INDIA 2026

Live Demo: The Unbounded Buffer



RULE #3

Every Buffer Needs a Bound

- Process messages immediately
- Use bounded queue with backpressure

A gRPC channel is opened and then the process forks. What happens after fork?

```
# client.py

channel =
grpc.insecure_channel(localhost:50051)
stub =
example_pb2_grpc.SimpleServiceStub(channel)
stub.DoSomething(example_pb2.Empty())
```

```
# server.py

class
MyService(pb2_grpc.SimpleServiceServicer):
    def DomSomething(self, request, context):
        pid = os.fork()
```

1) Works fine
2) Hangs forever
3) Random errors

A gRPC channel is opened and then the process forks. What happens after fork?

```
# client.py

channel =
grpc.insecure_channel(localhost:50051)
stub =
example_pb2_grpc.SimpleServiceStub(channel)
stub.DoSomething(example_pb2.Empty())
```

```
# server.py

class
MyService(pb2_grpc.SimpleServiceServicer):
    def DomSomething(self, request, context):
        pid = os.fork()
```

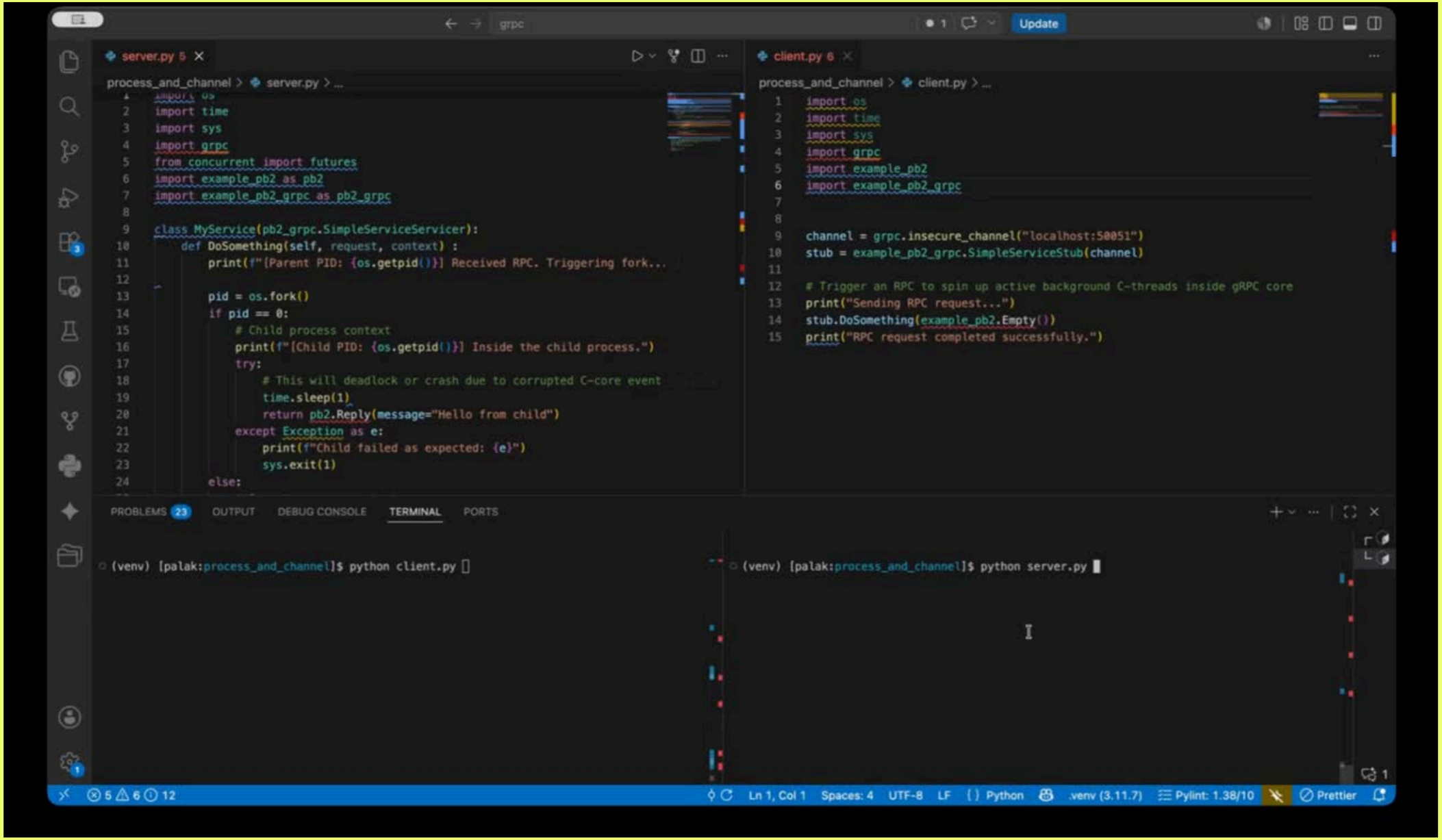
1) Works fine

2) Hangs forever

3) Random errors

gRPConf INDIA 2026

Live Demo: Fork Incompatibility



RULE #4

A Channel Belongs to Its Process

Forking should be a precursor to creating channels- be it server side or client side.

✅ **fork/spawn workers -> initialize gRPC**

❌ initialize gRPC threads -> os.fork()

gRPConf INDIA 2026

The gRPC Success Trap Revisited

1. `name field: ''` : **No `RpcError` was thrown, but application data was silently lost**

   → Rule 1: Field numbers are forever.

2. `Buffer growth` : **Stream completed without errors, but client RAM bloated**

   → Rule 3: Bound your application buffers.

And to prevent failure in latency and execution:

→ Rule 2: Bound your waits

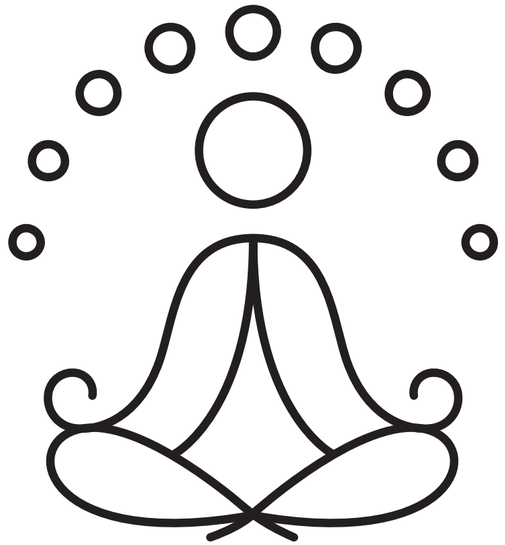
→ Rule 4: Keep channels process-local

Mechanisms Need Policies

gRPC gives you mechanisms.
Production reliability comes from policies.

gRPConf
INDIA 2026

THANK YOU

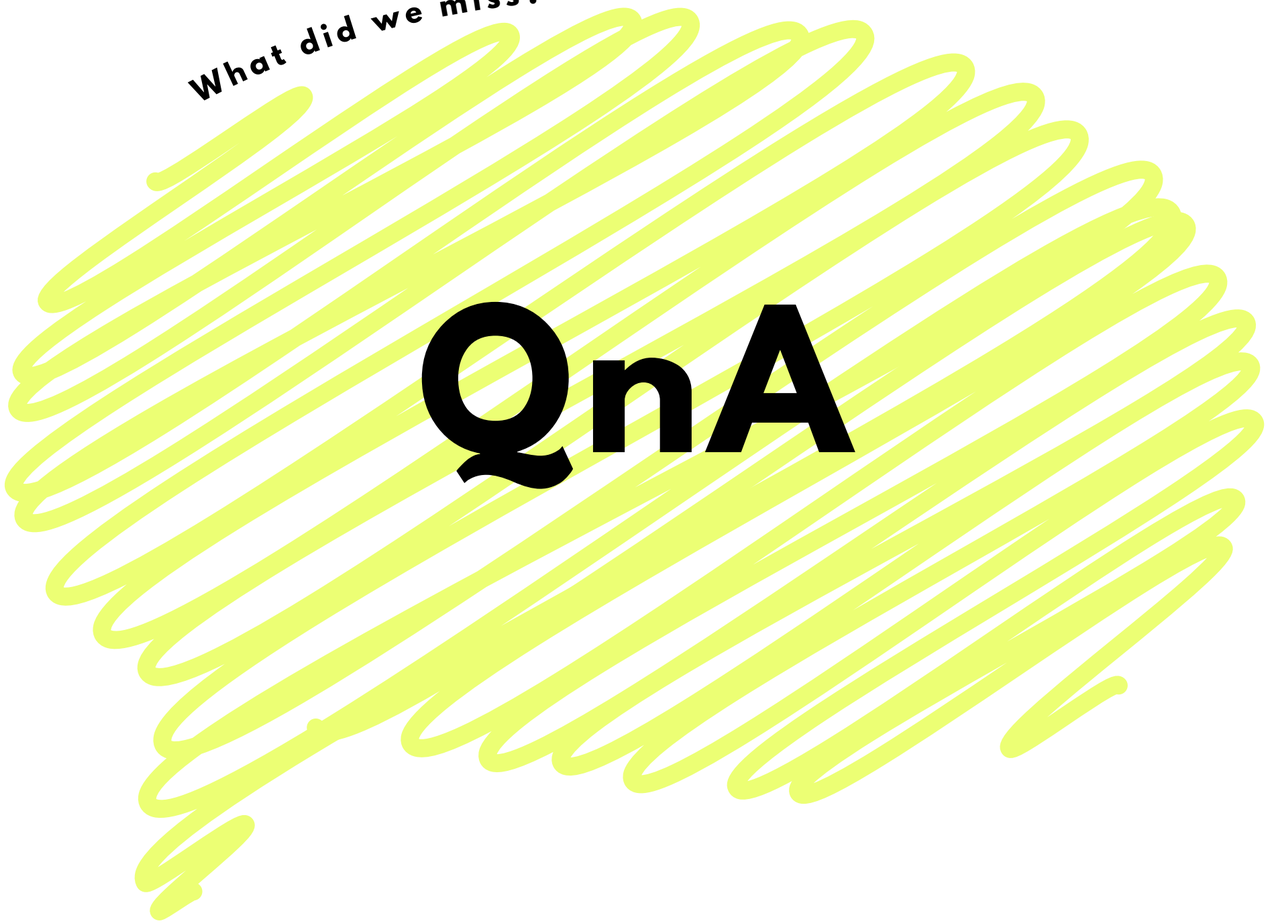


PALAK JAIN
STUTI JAIN

18TH NOVEMBER, 2026

gRPConf
INDIA 2026

What did we miss?



18TH NOVEMBER, 2026