Irina Scurtu

Embracing gRPC in .NET

https://irina.codes

@irina_scurtu

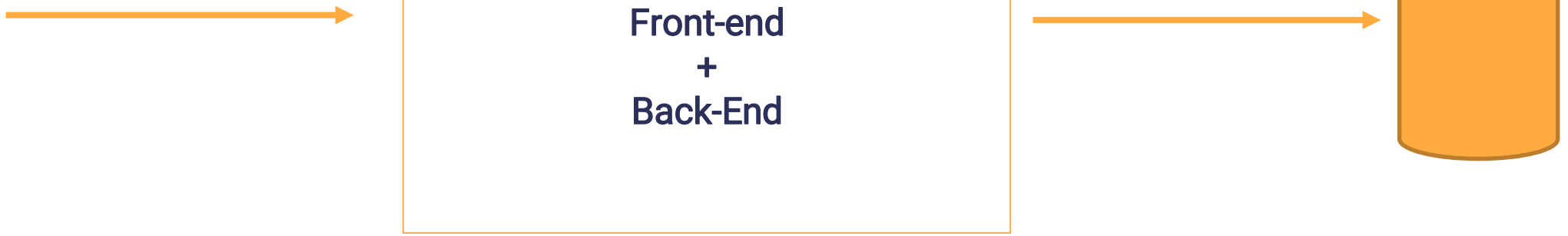
Irina Scurtu

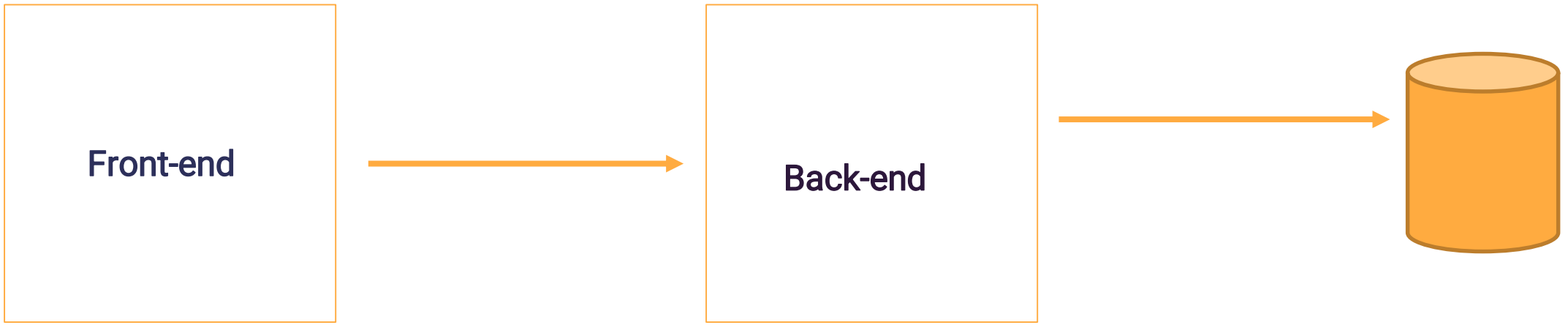

@irina_scurtu

- **Romania Based**
- Software Architect @Endava
- Organizer of DotNetIasi user group
- I teach .NET
- Blog: https://Irina.codes

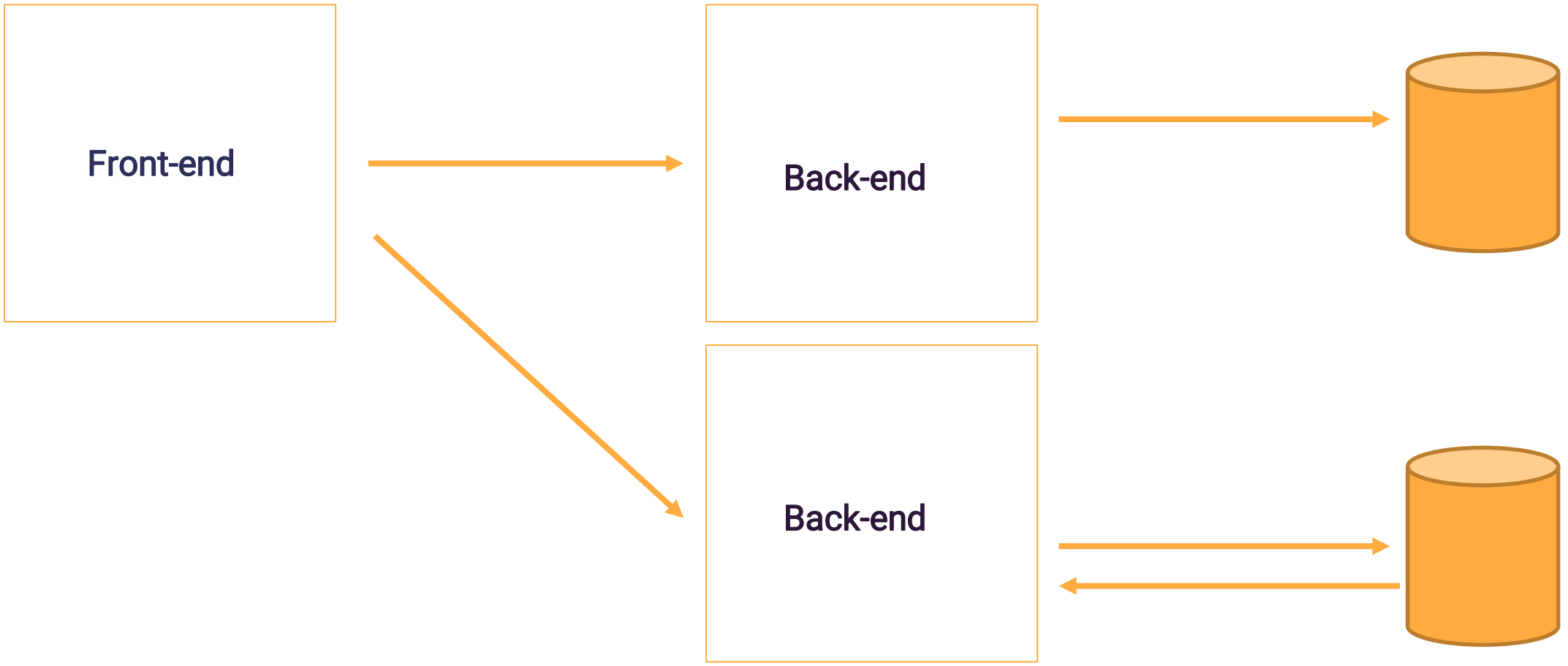
The Monolith

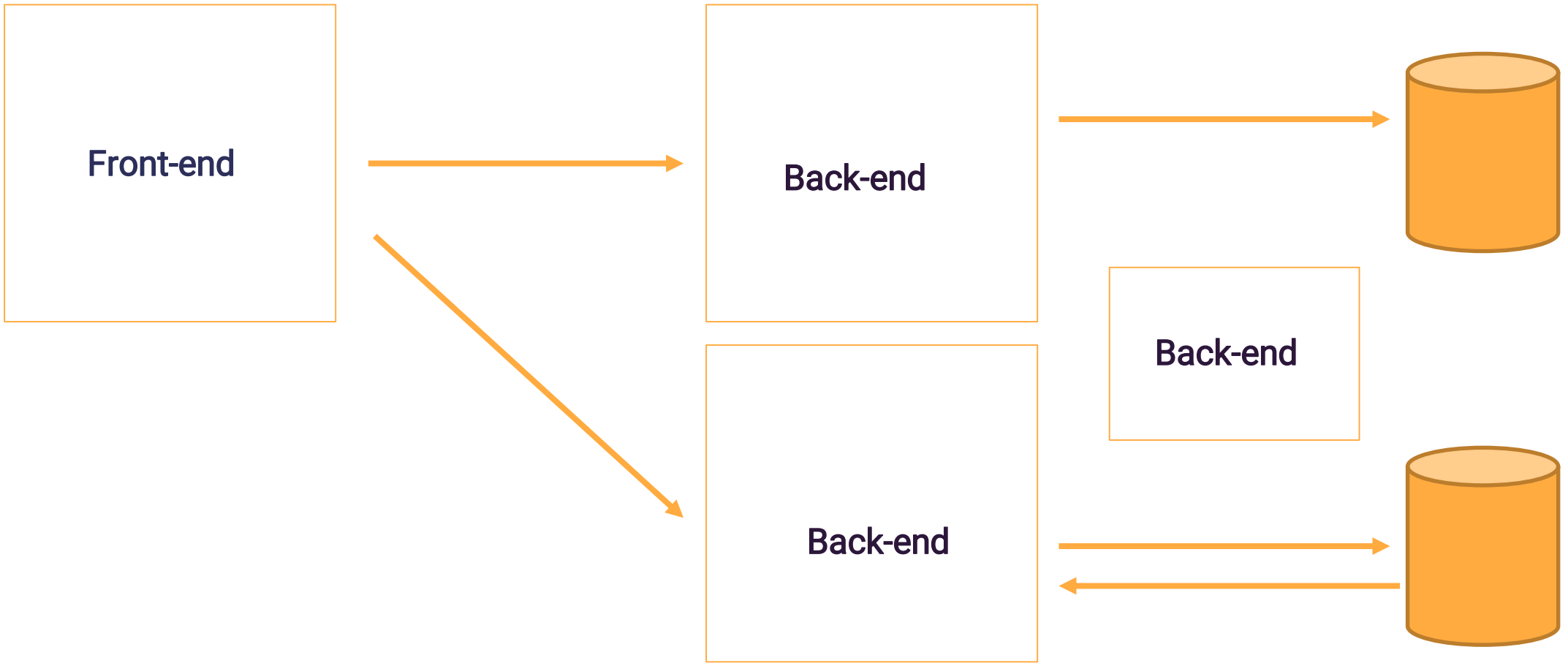
Easy life

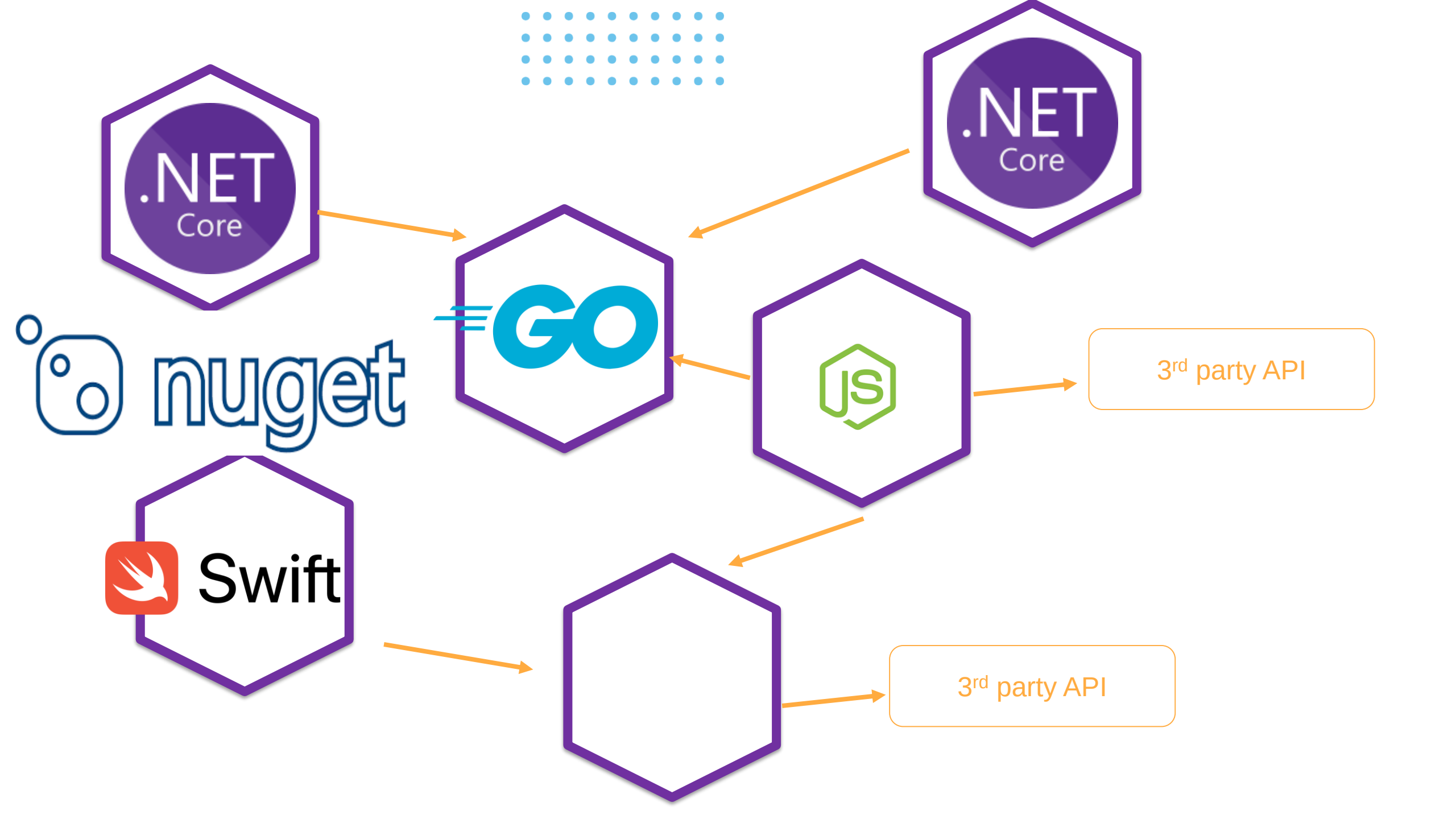

Front-end
+
Back-End


Front-end
Back-end

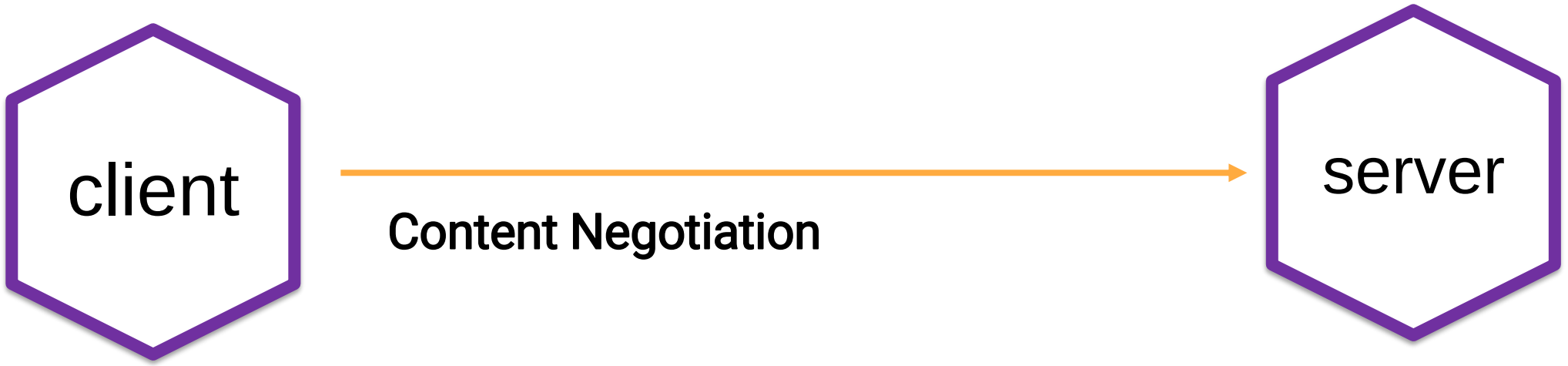
But the future... is distributed


Front-end
Back-end
Back-end


Front-end
Back-end
Back-end
Back-end


.NET Core
.NET Core
nuget
GO
JS
3rd party API
Swift
3rd party API

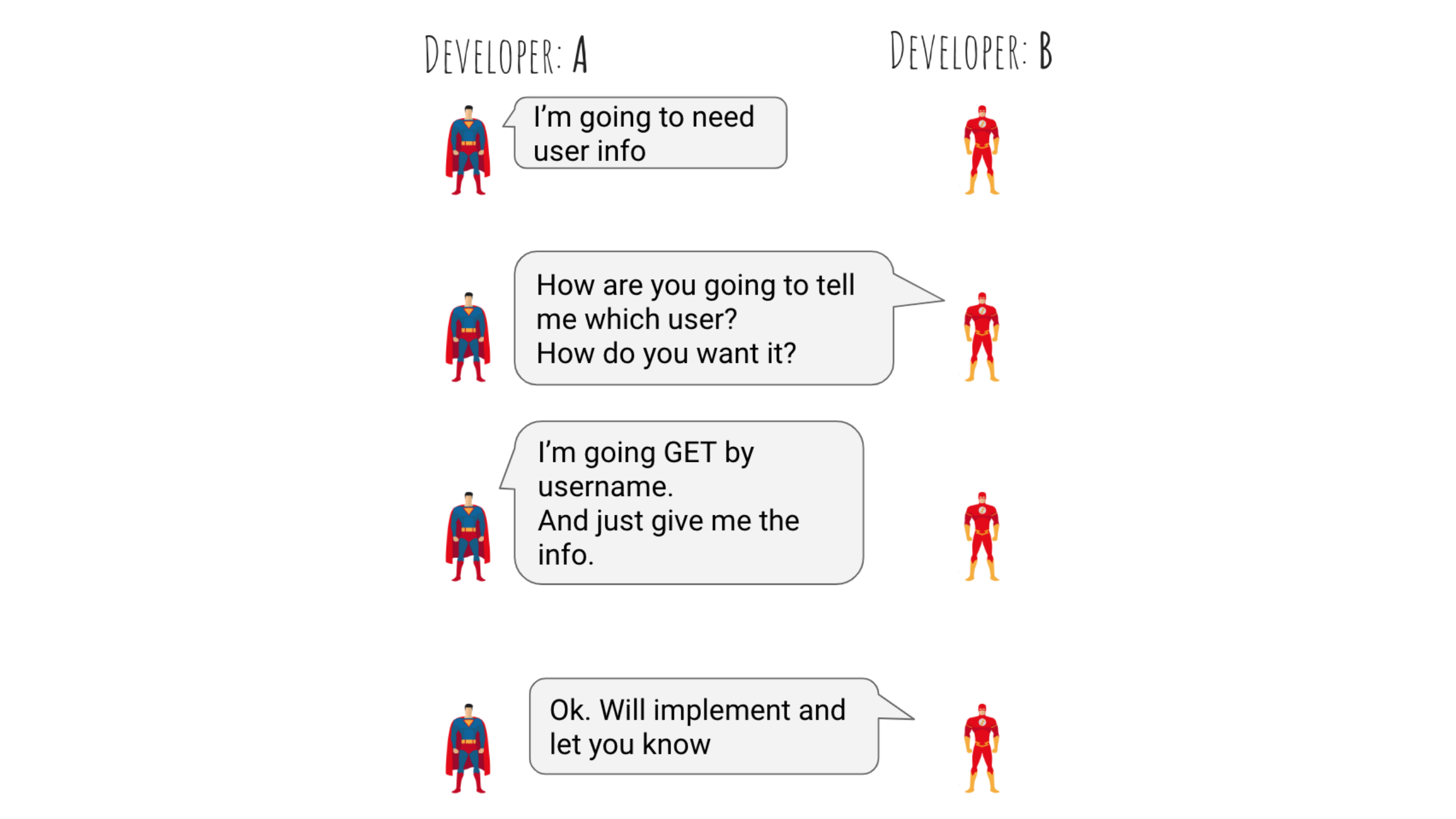
We have REST for everything!



- Format of the request
- What is payload?
- Error handling, retries
- Authentication?
- What happens when the model changes?

Why?!

What is wrong with REST?

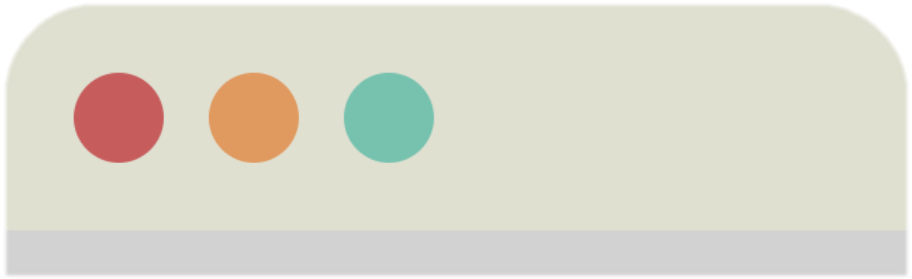

DEVELOPER: A
DEVELOPER: B
I'm going to need user info
How are you going to tell me which user? How do you want it?
I'm going GET by username. And just give me the info.
Ok. Will implement and let you know

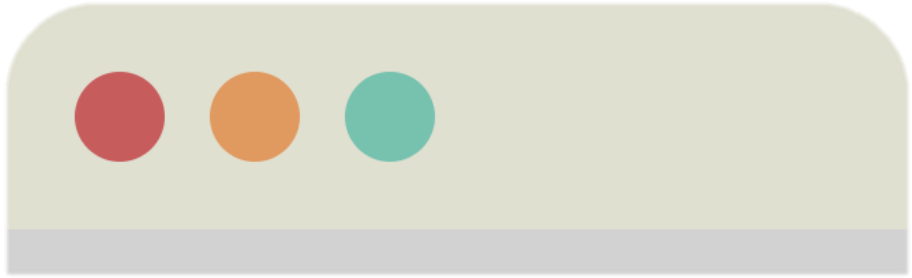
Remote Procedure Calls

RPC

```
var order = salesBoundedContext.CreateOrder(orderRequest);
var paymentStatus = billingBoundedContext.ProcessPaymentFor(order);
if (paymentStatus.IsSuccessful)
{
    shippingBoundedContext.ArrangeShippingFor(order);
}
```

RPC





Network

```
var paymentStatus =
billingBoundedContext.ProcessPaymentFor(order)
```

```
public Status
ProcessPaymentFor(order){
…….
….
….
….

}
```

RPC

- Makes code look local
- Is prone to errors

RPC

- Make network communication transparent

geee RPC

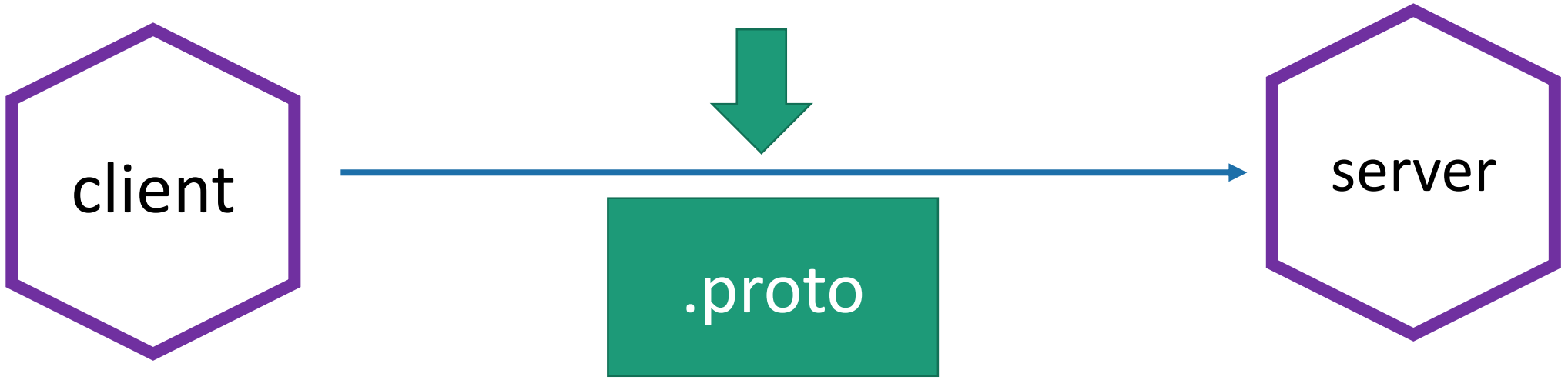
gRPC history

- 2001 google Stubby
- 2005 gRPC open sourced
- 2016 gRPC v1.0
- 2019 Sept gRPC with .NET core

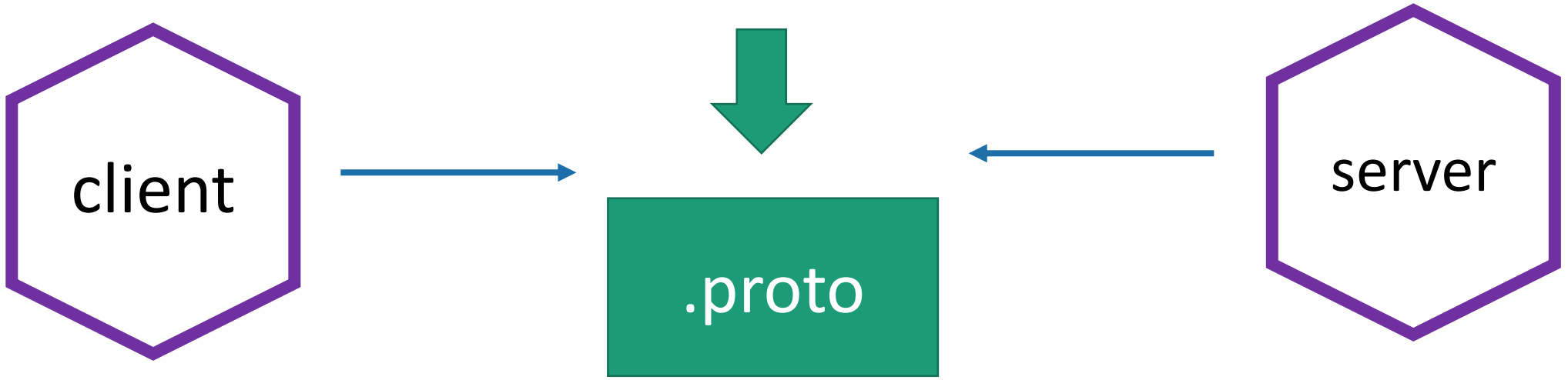
gRPC

- No-code references
- Contract based
- Uses HTTP/2 => faster
- Efficient ProtoBuf serialization => smaller payload
- Available in many languages
- Code generation

gRPC



gRPC



*.PROTO

```
 syntax = "proto3";
 option csharp_namespace = "MyFirstGrpc";
 package Fibonacci;



// The service definition.
service Fibo {
 rpc ComputeFibonacci(RequestedNumber) returns (FibonacciResult){}
}

//the request message format
 message RequestedNumber {
  int32 number = 1;
 }



 //the response message format
 message FibonacciResult {
   int32 result = 1;
 }
```

```
 syntax = "proto3";
 option csharp_namespace = "MyFirstGrpc";
 package Fibonacci;



// The service definition.
service Fibo {
 rpc ComputeFibonacci(RequestedNumber) returns (FibonacciResult){}
}

//the request message format
 message RequestedNumber {
  int32 number = 1;
 }


 //the response message format
 message FibonacciResult {
   int32 result = 1;
 }
```

```
 syntax = "proto3";
 option csharp_namespace = "MyFirstGrpc";
 package Fibonacci;



// The service definition.
service Fibo {
 rpc ComputeFibonacci(RequestedNumber) returns (FibonacciResult){}
}

//the request message format
 message RequestedNumber {
  int32 number = 1;
 }


 //the response message format
 message FibonacciResult {
   int32 result = 1;
 }
```

```
 syntax = "proto3";
 option csharp_namespace = "MyFirstGrpc";
 package Fibonacci;



// The service definition.
service Fibo {
 rpc ComputeFibonacci(RequestedNumber) returns (FibonacciResult){}
}

//the request message format
 message RequestedNumber {
  int32 number = 1;
 }


 //the response message format
 message FibonacciResult {
   int32 result = 1;
 }
```

```
 syntax = "proto3";
 option csharp_namespace = "MyFirstGrpc";
 package Fibonacci;



// The service definition.
service Fibo {
 rpc ComputeFibonacci(RequestedNumber) returns (FibonacciResult){}
}

//the request message format
 message RequestedNumber {
  int32 number = 1;
 }


 //the response message format
 message FibonacciResult {
   int32 result = 1;
 }
```

```
 syntax = "proto3";
 option csharp_namespace = "MyFirstGrpc";
 package Fibonacci;



// The service definition.
service Fibo {
 rpc ComputeFibonacci(RequestedNumber) returns (FibonacciResult){}
}

//the request message format
 message RequestedNumber {
  int32 number = 1;
 }


 //the response message format
 message FibonacciResult {
   int32 result = 1;
 }
```

gRPC

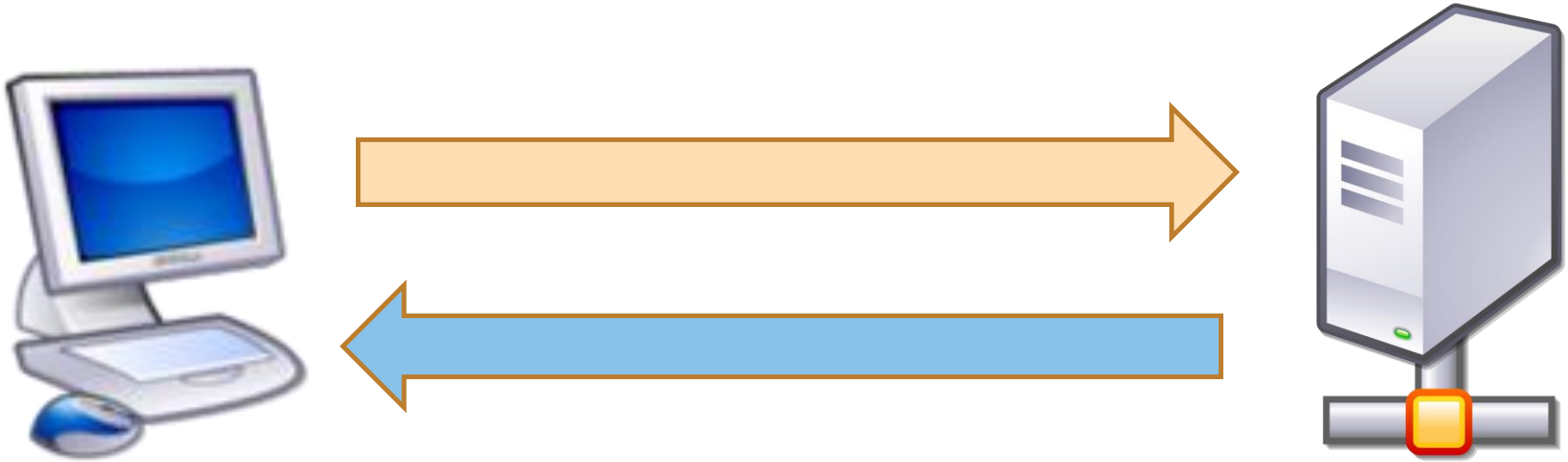
types

Modes/Method types

- Unary
- Server Streaming
- Client Streaming
- Bi-directional streaming

Unary

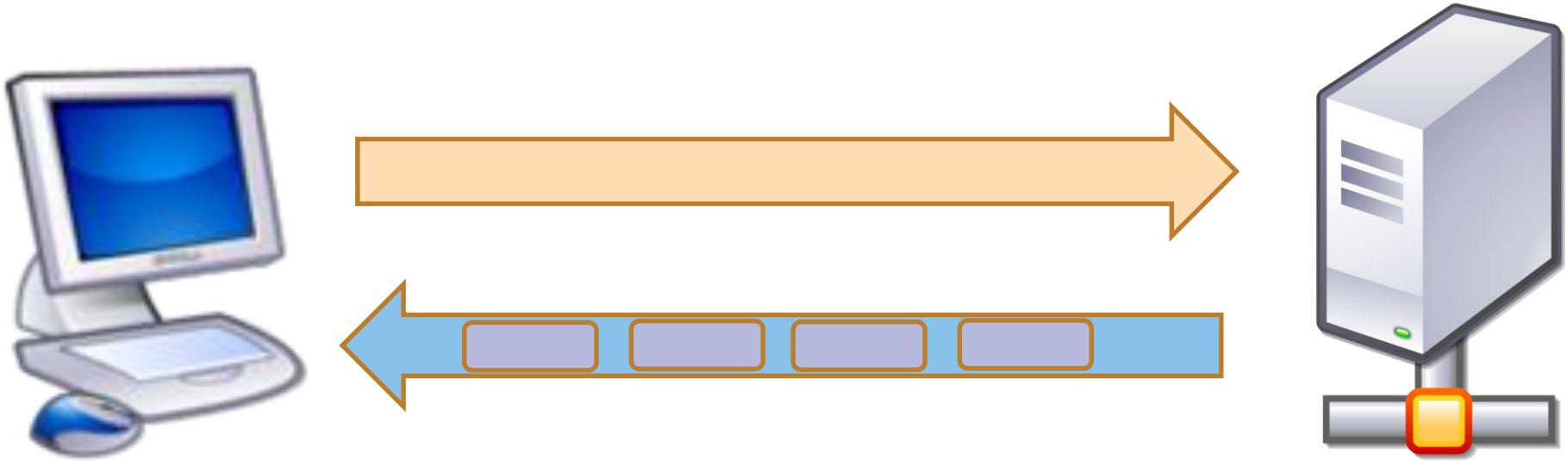
Unary



```
service Fibo {
   rpc ComputeFibonacci(RequestedNumber) returns (stream FibonacciResult){}
}
```

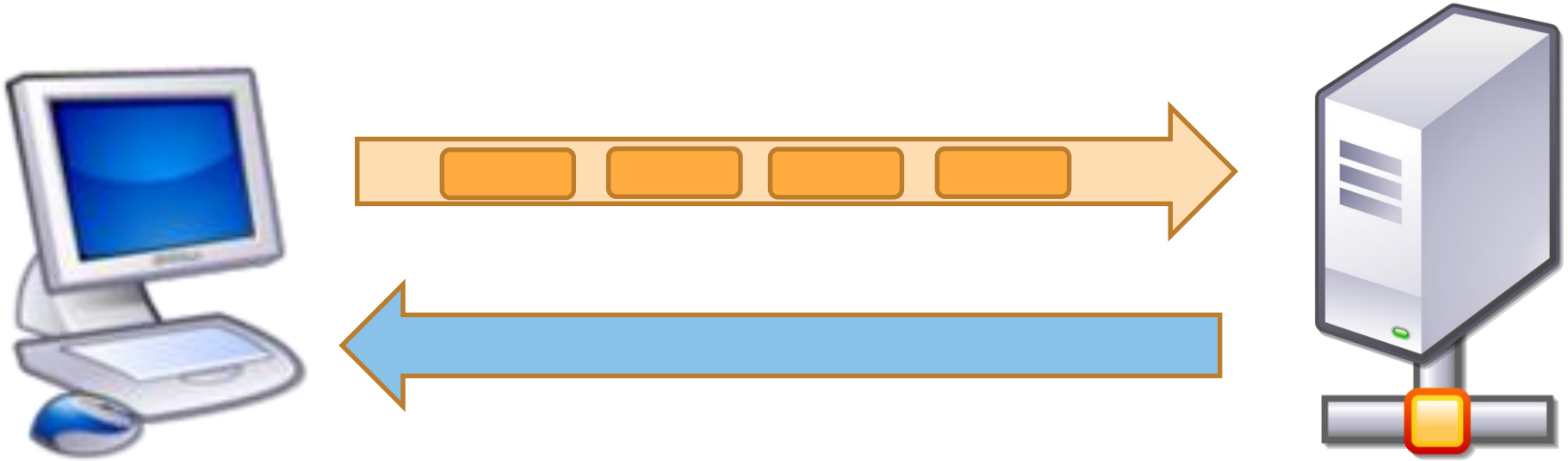
Server streaming

Server Streaming



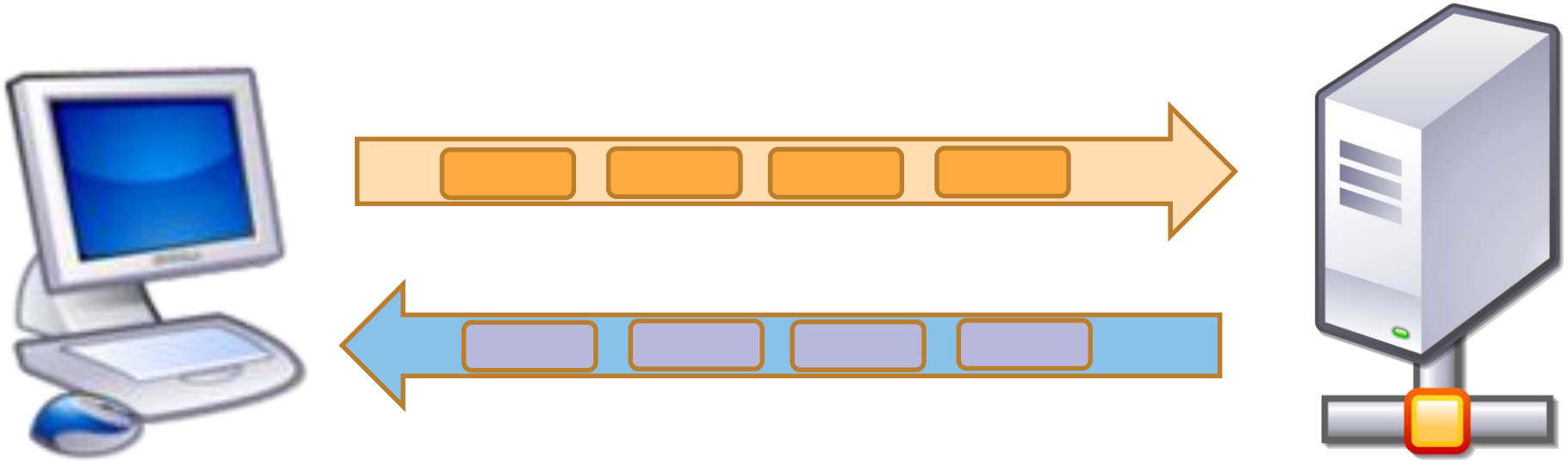
Client streaming

Client Streaming



Bi-directional streaming

Bi-directional Streaming



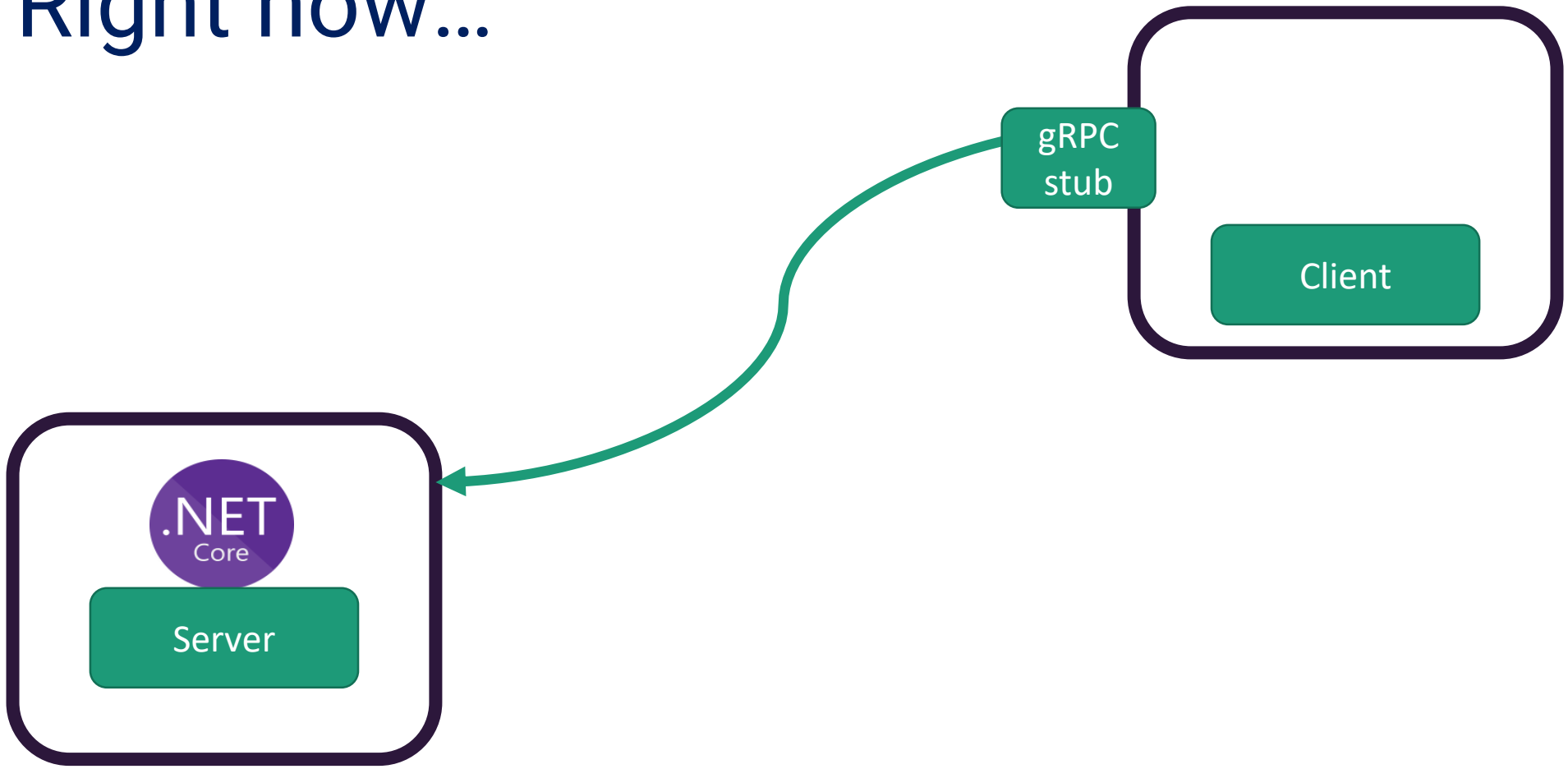
```
service Fibo {
 rpc ComputeFibonacci(stream RequestedNumber)
 returns (stream FibonacciResult){}
}
```

Strengths

In comparison

REST	RPC
Resource focused	Action based
Embrace HTTP semantics	Embrace programming semantics
Loose coupling?!	Tighter coupling
Text based	Binary based

Right now…



Performance

REST

10,072 Bytes

gRPC

1,916 Bytes

80.98% SMALLER

Polyglot environments

Lightweight

Point-to-point communication

- Allows polyglot environments
- Contract-based
- Smaller payloads
- HTTP/2 is
- Supports different streaming types

Downsides

- Temporal coupling
- You might forget that there is a network involved
- Not human readable
- You'll need better testing
- Focus on CI/CD

Summary

- Great choice for microservice communication
- Polyglot environments
- 4 modes/methods
- Decouples code

Future of Web APIs

- No more hunting documentation
- No misinterpretation of HTTP status codes
- No more data-parse errors

Distributed systems are all about tradeoffs

Next Steps

- gRPC-Web
- HTTP 2
- https://developers.google.com/protocol-buffers/docs/proto3

Thanks for listening!

@irina_scurtu

https://Irina.codes

Please rate this session using

Whova web portal

Event Sponsors

KMD
An NEC Company

Demant