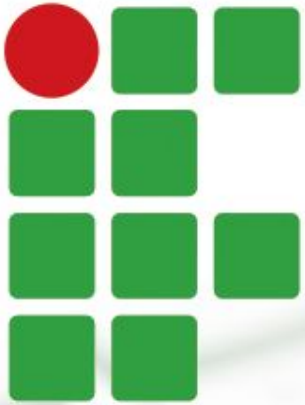


INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Sistemas Distribuídos

- RPC -

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Mensagens vs Serviços

- A **Comunicação Orientada a Mensagens** (*send* e *receive*) - como vimos na API Socket - implica que a troca de mensagens (e todo o contexto inerente) entre os processos (cliente-servidor) sejam inteiramente explícitas.

Mensagens vs Serviços

- Em 1984, Birrell e Nelson propuseram um novo modelo de comunicação, onde **processos** poderiam **invocar rotinas implementadas em máquinas remotas**.

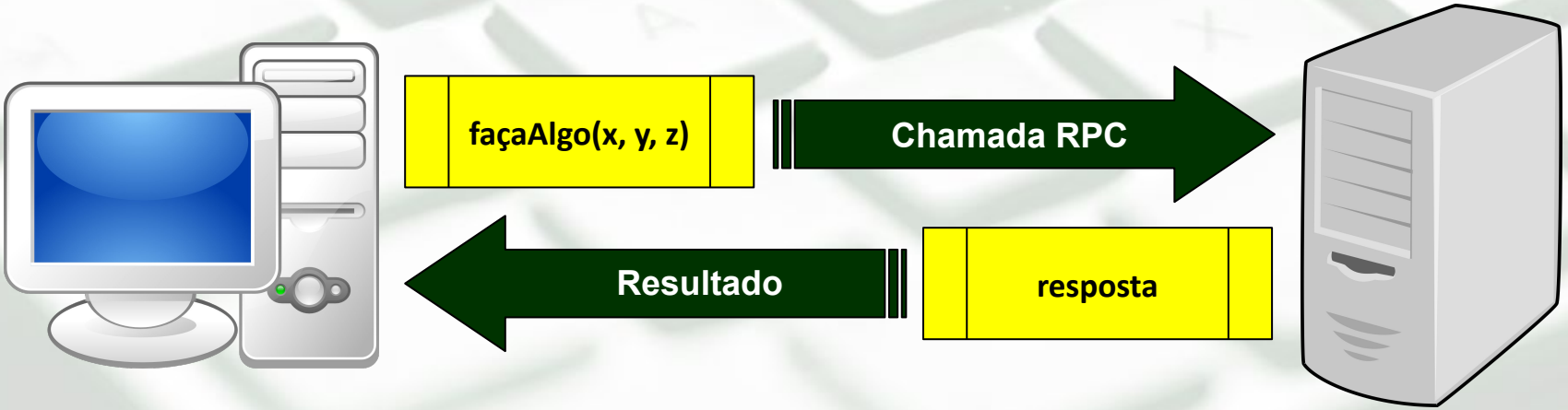
- **RPC => Chamada de Procedimento Remoto**

- Novo paradigma para a programação distribuída: **Arquitetura Orientada a Serviços** (SOA).

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Mensagens vs Serviços

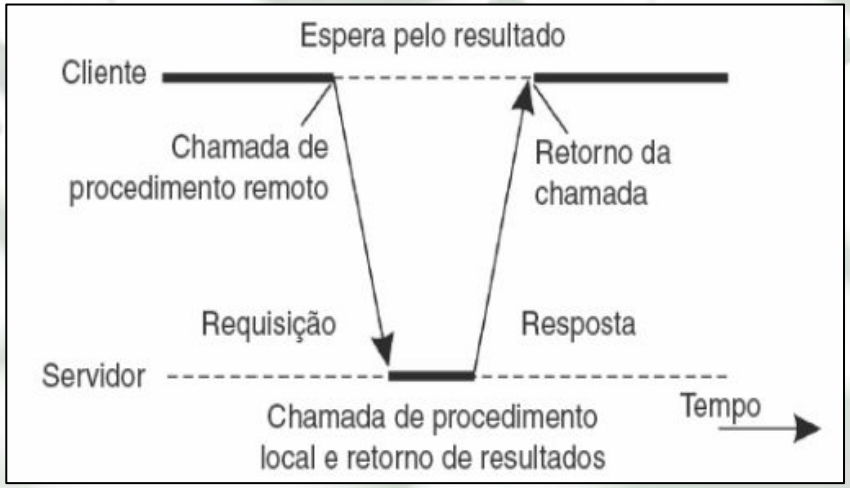
- Agora, o desenvolvedor pode apenas "*invocar uma função*" e não "*se preocupar*" com todos os detalhes da troca de mensagens entre os processos.
- A troca de mensagens (de fato), conexões, empacotamento, (etc…) são **transparentes para o programador**.



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

RPC

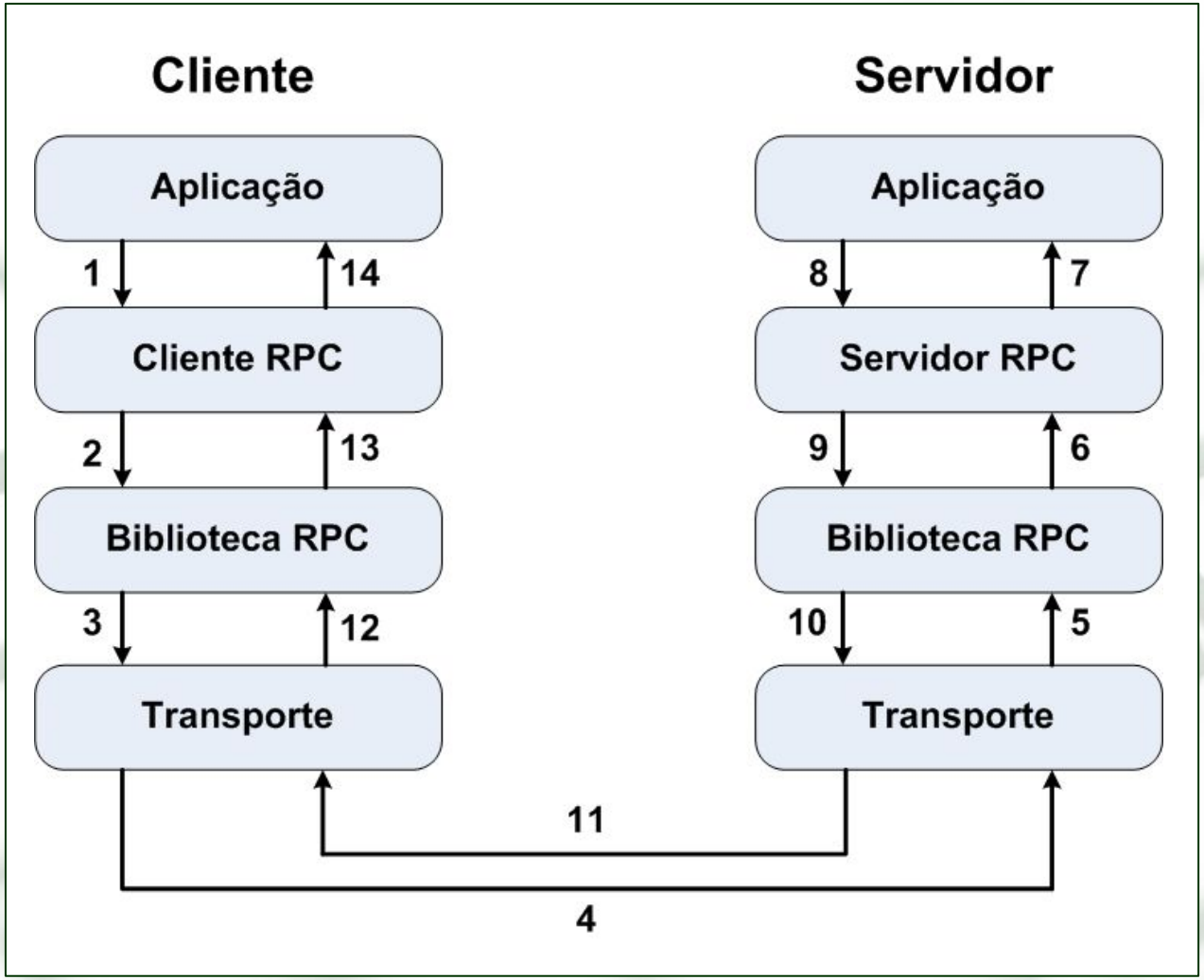
- A ideia que fundamenta uma RPC é fazer com que a execução de uma **rotina remota pareça o quanto possível a uma chamada de rotina local**.
- Ou seja, uma chamada RPC deve ser **transparente** no escopo local (não estar ciente que o processamento acontece remotamente).



RPC

- Embora a ideia pareça simples e elegante, existem problemas sutis a serem analisados…
  - Processos invocador e executor rodam em máquinas (arquiteturas/plataformas/codificações) distintas.
  - Como encapsular os parâmetros dos procedimentos e o resultado das chamadas em um formato inteligível para os dois lados dos processos?
  - Se houver demora ou falha na requisição-resposta?
- É necessário um ***middleware/biblioteca*** para auxiliar nessas questões…

Esquema RPC


Cliente
Servidor
Aplicação
Aplicação
1
14
8
7
Cliente RPC
Servidor RPC
2
13
9
6
Biblioteca RPC
Biblioteca RPC
3
12
10
5
Transporte
Transporte
11
4

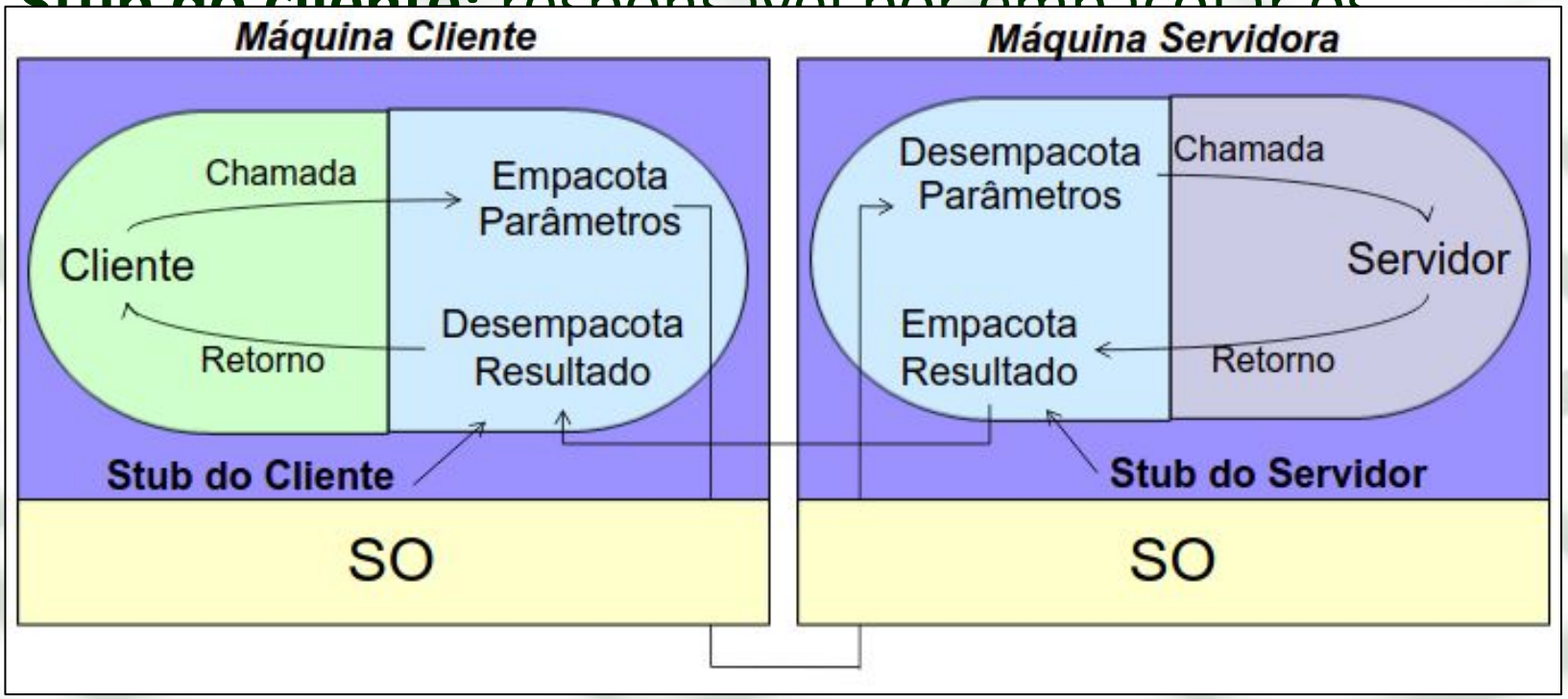
Stubs

- **A transparência é obtida por meio de stubs…**
- **Stub do cliente:** responsável por empacotar os parâmetros em uma mensagem e enviá-la para o processo servidor.
- **Stub do servidor:** responsável por desempacotar os parâmetros recebidos, executar o procedimento no servidor, empacotar e retornar o resultado para a máquina do cliente.

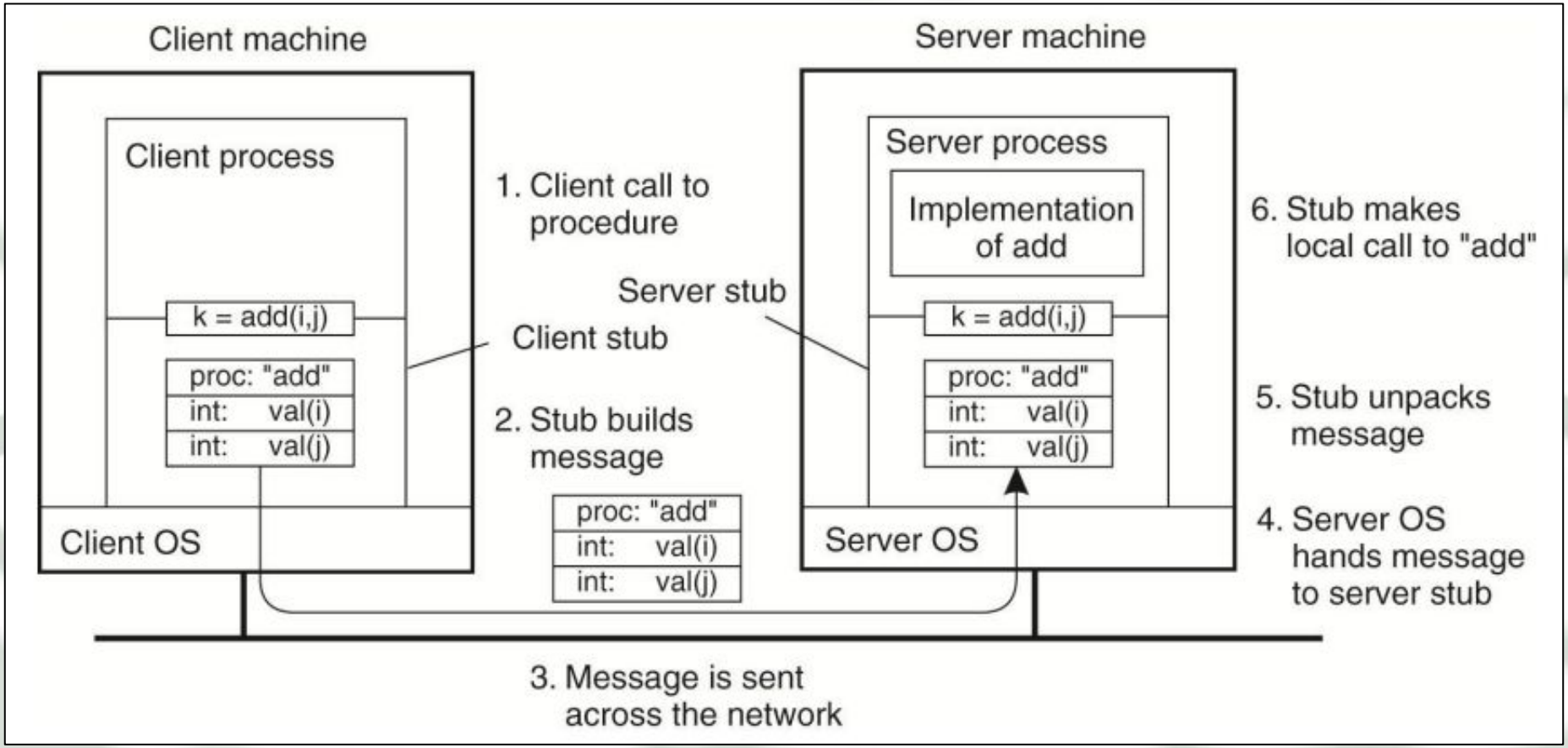
INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Stubs

- **A transparência é obtida por meio de stubs…**
- **Stub do cliente:** responsável por empacotar os
-

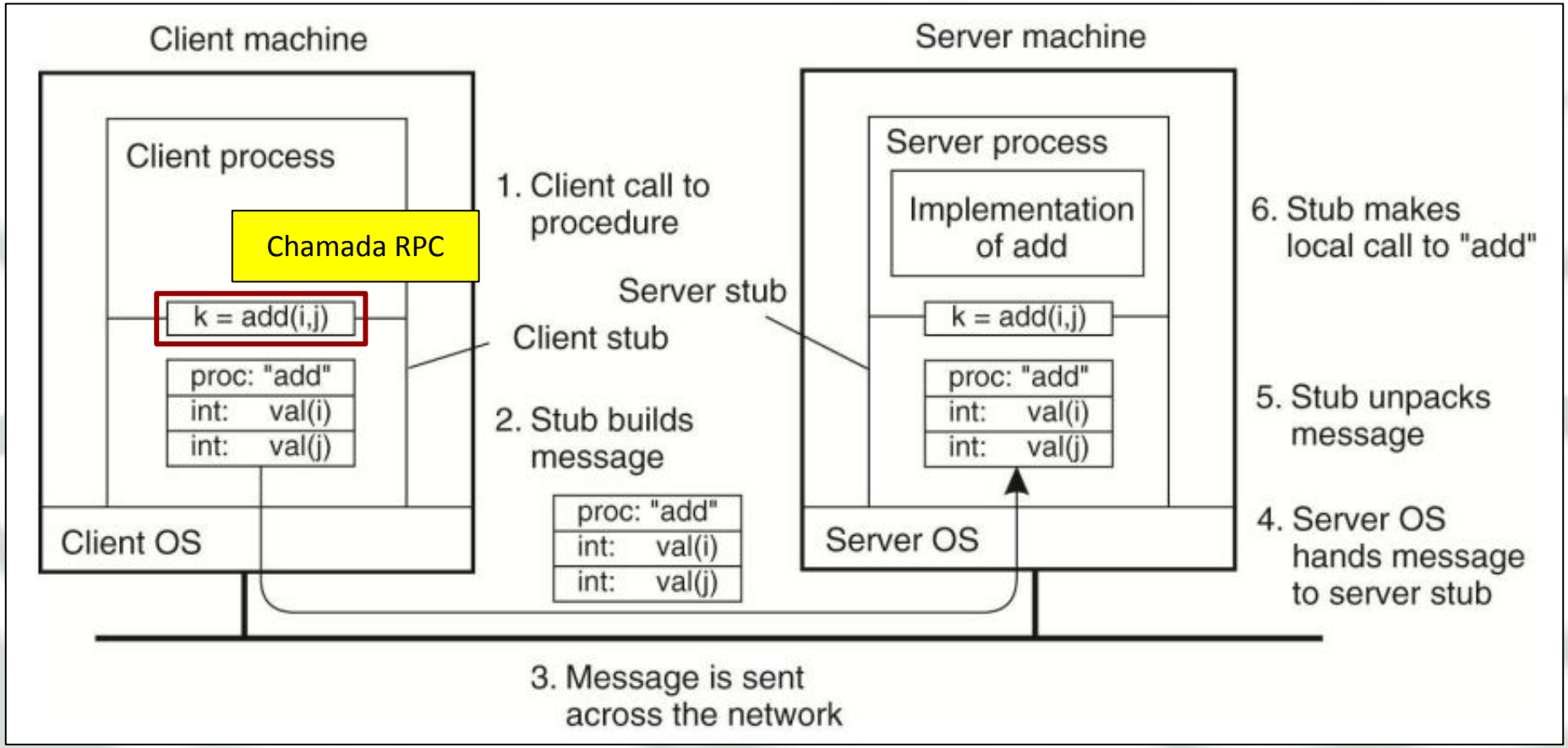


Stubs


Client machine
Server machine
Client process
Server process
Implementation of add
k = add(i,j)
k = add(i,j)
proc: "add"
int: val(i)
int: val(j)
proc: "add"
int: val(i)
int: val(j)
1. Client call to procedure
Server stub
Client stub
2. Stub builds message
proc: "add"
int: val(i)
int: val(j)
6. Stub makes local call to "add"
5. Stub unpacks message
4. Server OS hands message to server stub
Client OS
Server OS
3. Message is sent across the network

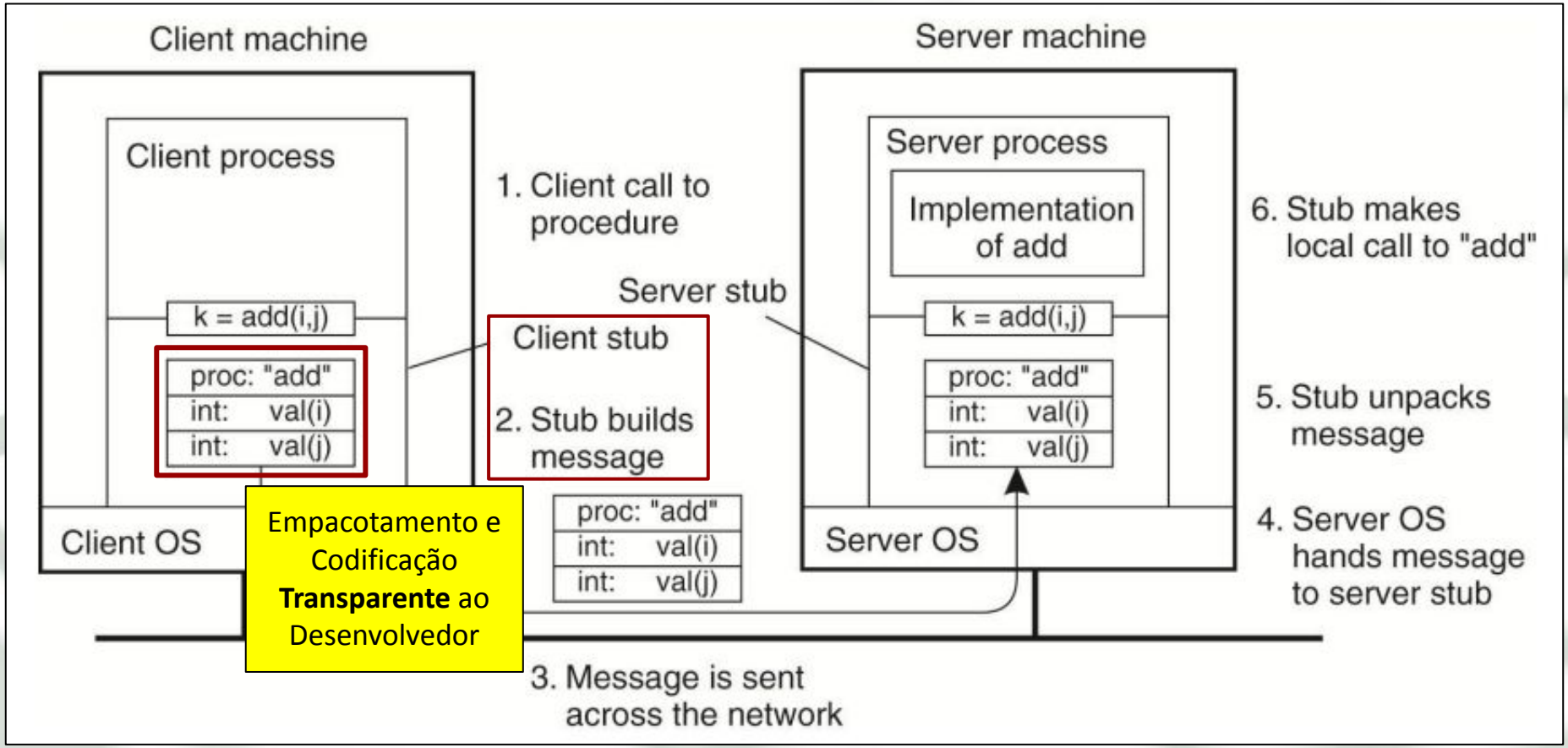
INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Stubs



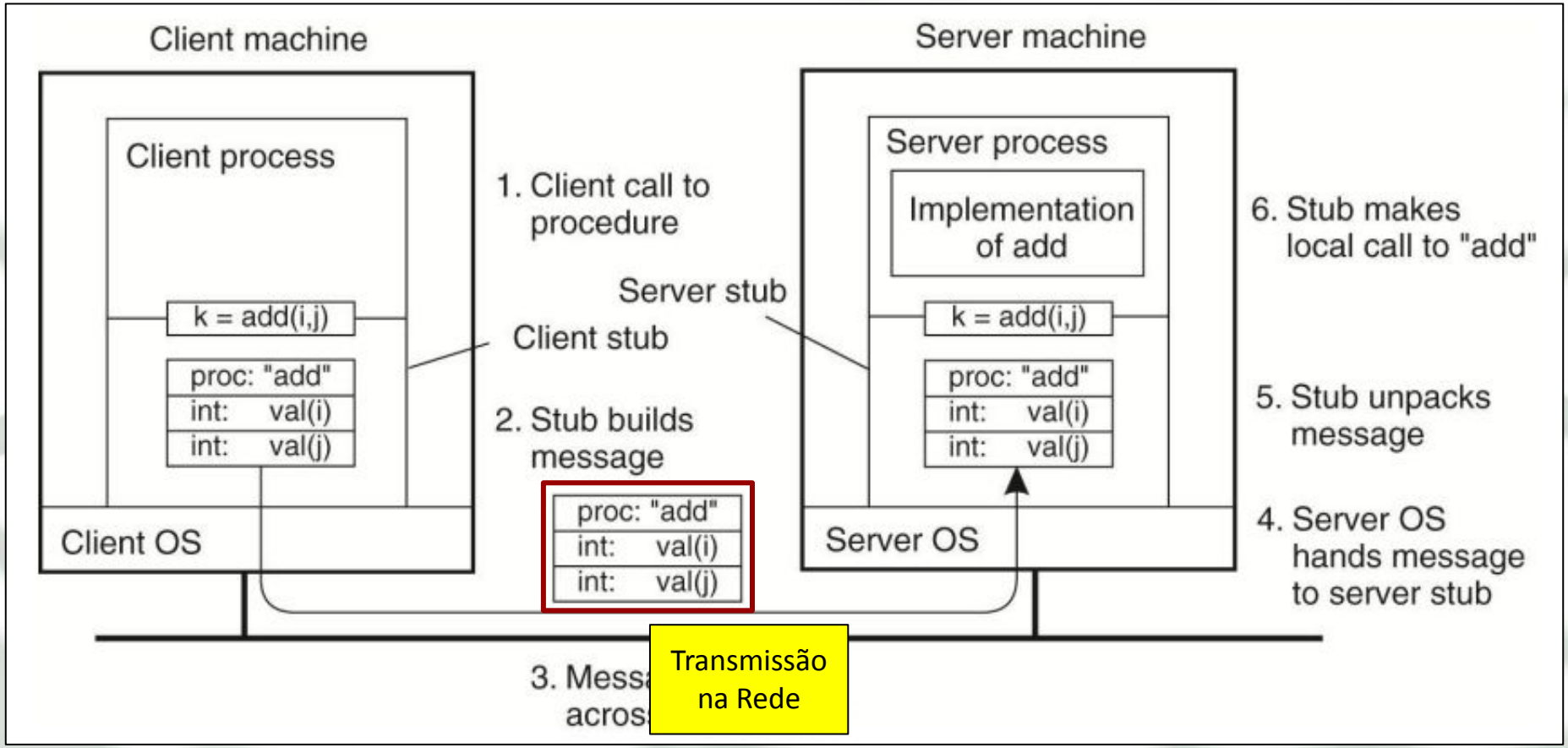
INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Stubs



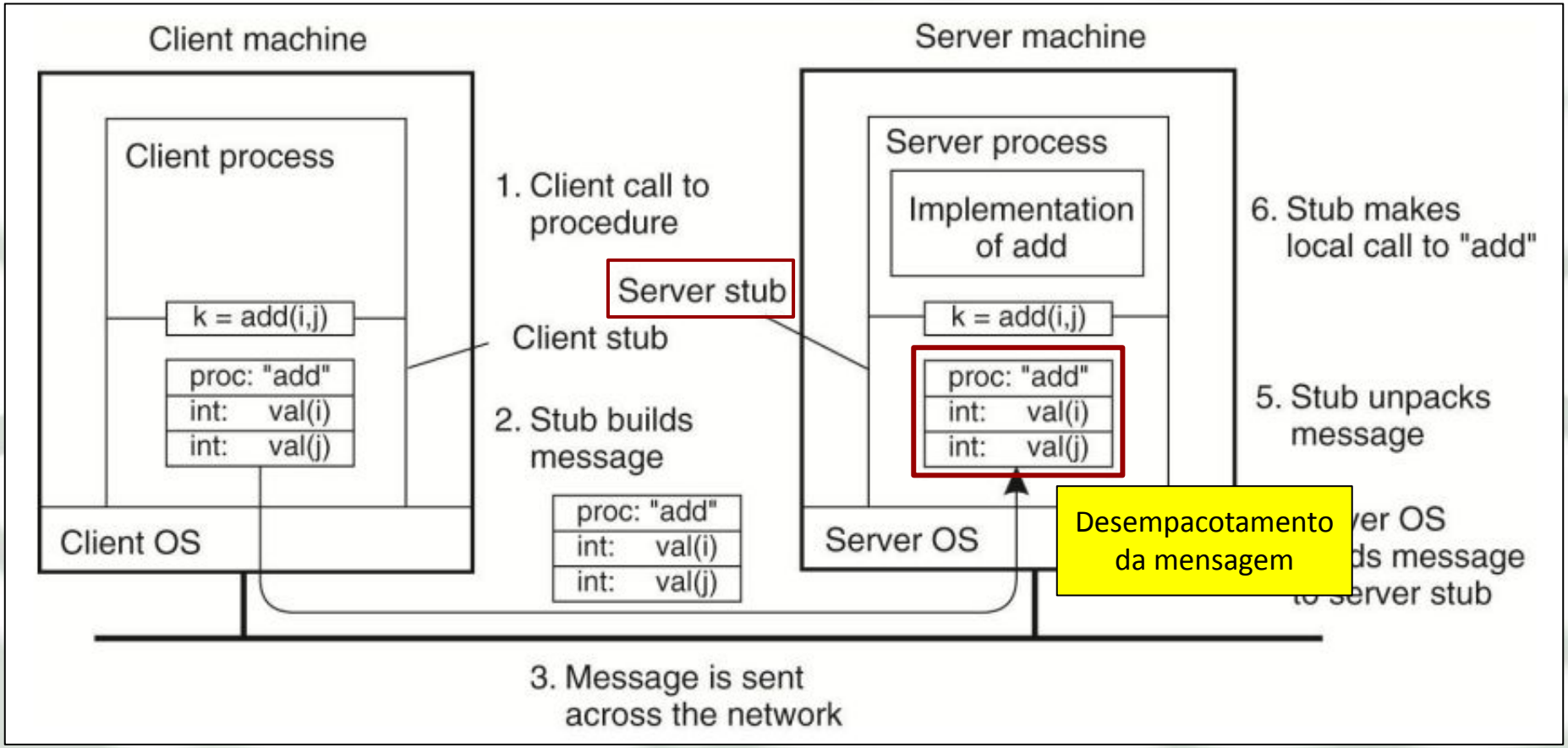
INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Stubs



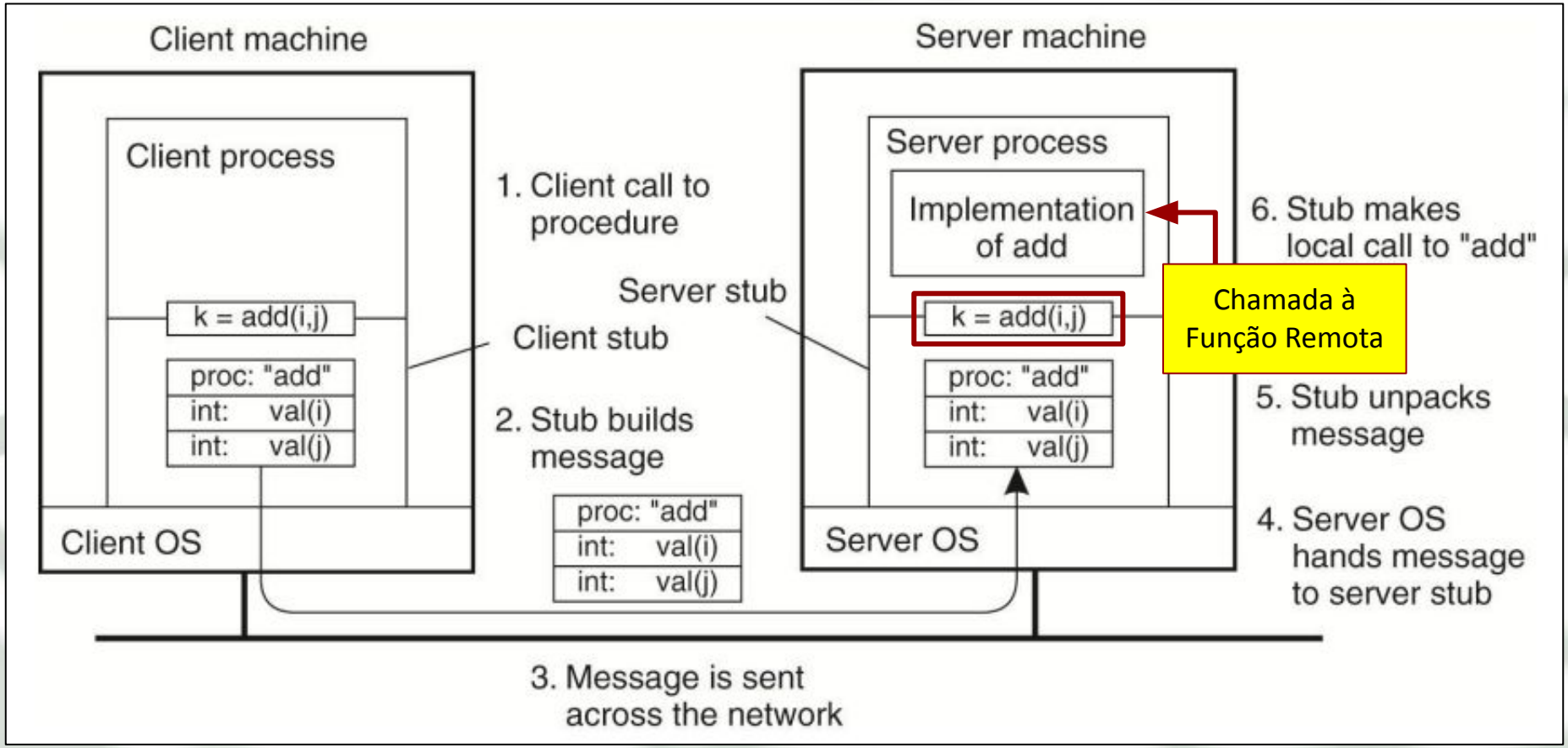
INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Stubs



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Stubs



IDLs

- Geralmente, o servidor RPC disponibiliza uma **Interface** com chamadas disponíveis (parâmetros necessários e retornos esperados) para utilização dos clientes.

- IDL = *Interface Definition Language*

- Exemplo: RPC Google Cloud - DataStore
  - *https://cloud.google.com/datastore/docs/reference/data/rpc*

Algumas Implementações

- CORBA (RPC multiplataforma)
- DCOM (Microsoft/Windows)
- Java RMI (*Remote Method Invocation*)
  - RPC + Orientação a Objeto
- gRPC (Google)
- RPyC (Python)
- Arquitetura de *Web Services*:
  - RPC sobre HTTP
  - ➔ SOAP - *Simple Object Access Protocol*
  - ➔ REST - *Representational State Transfer*

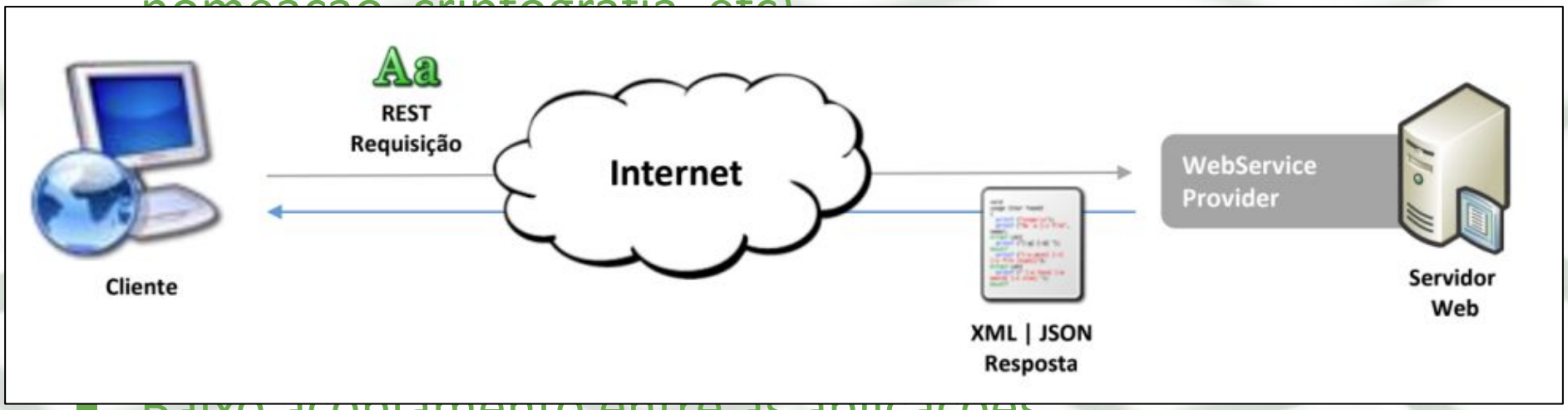
Web Services

- Utiliza a infraestrutura HTTP para encapsulamento e transporte das mensagens (aproveitamento o esquema de nomeação, criptografia, etc).
- Porta já disponível (p.ex. 80 ou 443) para tratamento das requisições (torna o gerenciamento do ambiente de segurança mais simplificado).
- Interoperabilidade com diversas plataformas e linguagens.
- Baixo acoplamento entre as aplicações.
- XML e JSON como principais de padrões para empacotamento dos dados.

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Web Services

- Utiliza a infraestrutura HTTP para encapsulamento e transporte das mensagens (aproveitamento o esquema de nomeação, criptografia, etc).



- Baixo acoplamento entre as aplicações.
- XML e JSON como principais de padrões para empacotamento dos dados.

RPyC

- *"RPyC is a transparent python library for symmetrical remote procedure calls, clustering and distributed-computing"*

- https://rpyc.readthedocs.io

RPyC

- ***Features***:
  - Transparente: Acesso a objetos remotos como se fossem locais.
  - Simétrico: Clientes e Servidores podem invocar e servir requisições simultaneamente.
  - Suporte a chamadas Síncronas (bloqueantes) e Assíncronas (não-bloqueantes).
  - Multi plataforma e multi arquitetura.
  - Baixo *Overhead* e simples configuração.
  - Segurança: Integrável a protocolos SSH, TLS/SSL.

RPyC Server

```
import rpyc
from rpyc.utils.server import ThreadPoolServer

class MyService(rpyc.Service):
    #
    # ... you service's implementation
    #
    …

if __name__ == "__main__":
  try:
    server = ThreadPoolServer(MyService, port=5001)
    server.start()
  except Exception as e:
    print(e)
```

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

RPyC Server

```
import rpyc
from rpyc.utils.server import ThreadPoolServer

class MyService(rpyc.Service):
    #
    # ... you service's implementation
    #
    …

if __name__ == "__main__":
  try:
    server = ThreadPoolServer(MyService, port=5001)
    server.start()
  except Exception as e:
    print(e)
```

Já trata a concorrência de acessos, oferecendo aos clientes os serviços expostos pela Classe **MyService**.

RPyC Server

```
import rpyc
from rpyc.utils.server import ThreadPoolServer

class Calculadora(rpyc.Service):
    def exposed_soma(self, op1, op2):
        return op1 + op2

    def exposed_produto(self, op1, op2):
        return op1 * op2

if __name__ == "__main__":
  try:
    server = ThreadPoolServer(Calculadora, port=5001)
    server.start()
  except Exception as e:
    print(e)
```

RPyC Server

```
import rpyc
from rpyc.utils.server import ThreadPoolServer

class Calculadora(rpyc.Service):
    def exposed_soma(self, op1, op2):
        return op1 + op2

    def exposed_produto(self, op1, op2):
        return op1 * op2

if __name__ == "__main__":
  try:
    server = ThreadPoolServer(Calculadora, port=5001)
    server.start()
  except Exception as e:
    print(e)
```

O prefixo "**exposed_**" nos métodos indica que esse serviço pode ser invocado pelos clientes RPyC.

RPyC Server

```
import rpyc
from rpyc.utils.server import ThreadPoolServer

class Calculadora(rpyc.Service):
    def exposed_soma(self, op1, op2):
        return op1 + op2

    def exposed_produto(self, op1, op2):
        return op1 * op2

if __name__ == "__main__":
  try:
    server = ThreadPoolServer(Calculadora, port=5001)
    server.start()
  except Exception as e:
    print(e)
```

O prefixo "**exposed_**" nos métodos indica que esse serviço pode ser invocado pelos clientes RPyC.

Variáveis e **Classes** (*Objetos Remotos*) também podem ser expostos…
>>> Necessário o prefixo **exposed_**

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

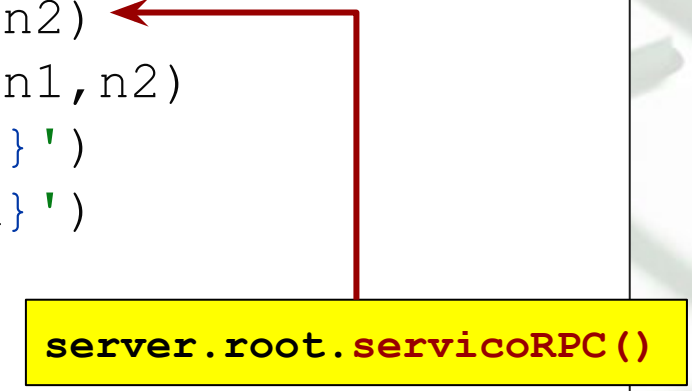
RPyC Client

```
import rpyc

n1 = int(input('Informe um valor numérico: '))
n2 = int(input('Informe outro valor numérico: '))

try:
   server = rpyc.connect('localhost',5001)
   resultSoma = server.root.soma(n1,n2)
   resultProd = server.root.produto(n1,n2)
   print(f'{n1} + {n2} = {resultSoma}')
   print(f'{n1} * {n2} = {resultProd}')

except Exception as e:
   print ('Falha: '+str(e))
```



Outros Métodos...

```
import rpyc
from rpyc.utils.server import ThreadPoolServer

class Calculadora(rpyc.Service):
    def on_connect(self, conn):
        print(f'Requisição Solicitada por
                {conn._channel.stream.sock.getpeername()}')

    def on_disconnect(self, conn):
        print('Conexão Encerrada')

if __name__ == "__main__":
  try:
    server = ThreadPoolServer(Calculadora, port=5001)
    server.start()
  except Exception as e:
    print(e)
```

Cliente RPyC com Dicionários

```
import json
import rpyc

def enviarPessoa(server):
  pessoa = {}
  pessoa['nome'] = 'Adriano'
  pessoa['idade'] = 35
  pessoa['altura'] = 1.79
  result = server.root.cadastrarPessoa(json.dumps(pessoa))
  print('Dados enviados para o Server.')
  print(result)

try:
  server = rpyc.connect('localhost',5001)
  enviarPessoa(server)
except Exception as e:
   print ('erro: '+str(e))
```

Server RPyC com Dicionários

```
import json
import rpyc
from rpyc.utils.server import ThreadPoolServer

class MyService(rpyc.Service):
    def exposed cadastrarPessoa(self, pessoa):
        dados = json.loads(pessoa)
        print(dados['nome'])
        print(dados['idade'])
        print(dados['altura'])
        return 'Dados Recebidos e Cadastrados.'

try:
    server = ThreadPoolServer(MyService, port=5001)
    server.start()
except Exception as e:
    print(e)
```

Server RPyC com Dicionários

Contudo, em um cenário realístico, a conexão e tempo de transferência de dados entre cliente-servidor irá oscilar na rede de comunicação...

```
class MyService(rpyc.Service):
    def exposed_cadastrarPessoa(self, pessoa):
        # simula oscilações na rede
        time.sleep(random.randint(2,10))
        dados = json.loads(pessoa)
        print(dados['nome'])
        print(dados['idade'])
        print(dados['altura'])
        return 'Dados Recebidos e Cadastrados.'
try:
    server = ThreadPoolServer(MyService, port=5001)
    server.start()
except Exception as e:
    print(e)
```

O lado cliente deve estar preparado para essas oscilações

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Cliente RPyC Assíncrono

```
def enviarPessoa(server):
  pessoa = {}
  pessoa['nome'] = 'Adriano'
  pessoa['idade'] = 35
  pessoa['altura'] = 1.79
  func assync = rpyc.async_(server.root.cadastrarPessoa)
  result = func_assync(json.dumps(pessoa))
  print('Dados enviados para o Server.')
  print(result.value)

try:
  server = rpyc.connect('localhost',5001)
  enviarPessoa(server)
except Exception as e:
   print ('erro: '+str(e))
```

"**rpyc.async_()**" transforma um serviço RPC em uma função assíncrona no cliente.

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Cliente RPyC Assíncrono

```
def enviarPessoa(server):
  pessoa = {}
  pessoa['nome'] = 'Adriano'
  pessoa['idade'] = 35
  pessoa['altura'] = 1.79
  func assync = rpyc.async_(server.root.cadastrarPessoa)
  result = func_assync(json.dumps(pessoa))
  print('Dados enviados para o Server.')
  print(result.value)

try:
  server = rpyc.connect('localhost',5001)
  enviarPessoa(server)
except Exception as e:
   print ('erro: '+str(e))
```

"**rpyc.async_()**" transforma um serviço RPC em uma função assíncrona no cliente.

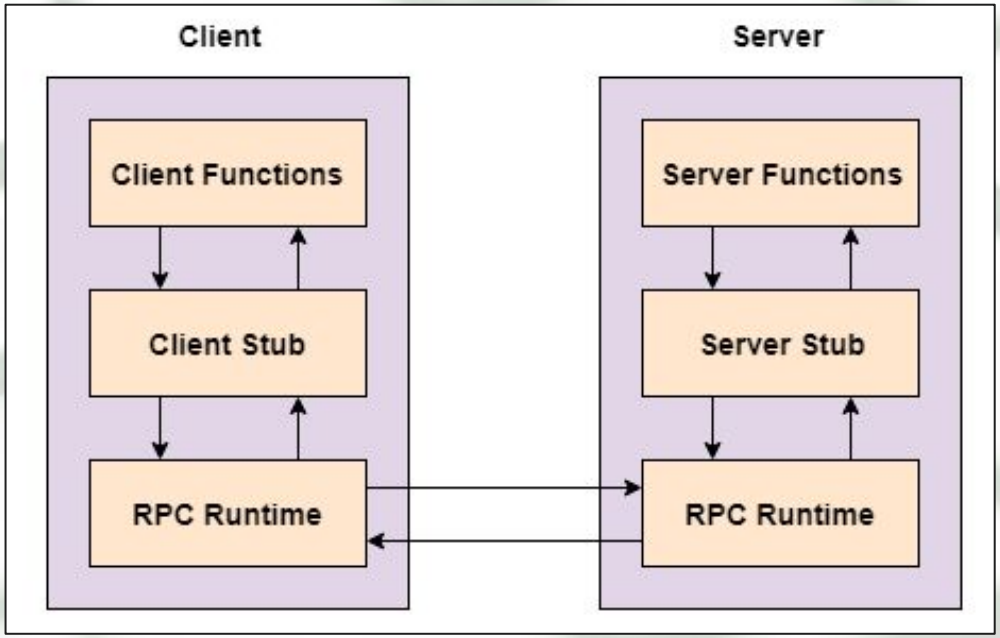
Essa função pode então ser invocada **concorrentemente**, e o resultado da chamada RPC é obtido (quando estiver disponível) pelo objeto **result.value**

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Laboratório #08.01

- Vimos que o RPyC é **simétrico**, ou seja, permite que serviços RPC sejam expostos tanto do lado do servidor quanto também do cliente…

Pesquise e implemente uma aplicação que demonstra o funcionamento desse recurso, usando dicionários na troca de mensagens.



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Laboratório #08.02

- Implemente uma simulação do ambiente de vendas através de máquinas de cartão de crédito…

Referências

- VAN STEEN, Maarten; TANENBAUM, Andrew S. Distributed systems. Leiden, The Netherlands: Maarten van Steen, 2017.

- MENDES, Eduardo. Lives de Python. YouTube Channel. https://github.com/dunossauro/live-de-python

- GUEDES, Dorgival. Notas de aula, UFMG. YouTube Channel: https://www.youtube.com/channel/UCJQHsVoqmkygpOXtGfKECFw