

# INVOCACÃO REMOTA E SERVIÇO DE MIDDLEWARE

RPC, RMI

SERVIÇO DE NOMES

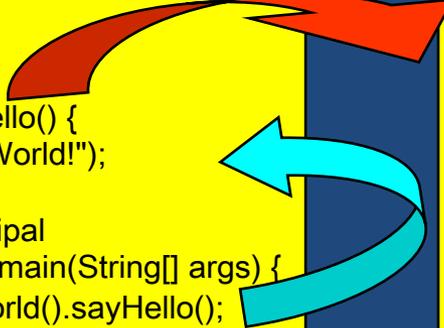
```
// quase Java....
class HelloWorld {
    // método local
    public void sayHello() {
        print("Hello World!");
    }
    // programa principal
    public static void main(String[] args) {
        new HelloWorld().sayHello();
    }
}
```

```
public class HelloWorld {
    public HelloWorld ( String name ) {
        Naming.rebind( name, this );
    }
    // método remoto
    public String sayHello () {
        return "Hello World!";
    }
}
public class HelloWorldServer {
    public static void main(String[] args) {
        HelloWorld object = new HelloWorld( "Hello" );
    }
}
```



```
public class HelloWorldClient {

    public static void main(String[] args) {
        // conexão
        HelloWorld hello_server = (HelloWorld)Naming.lookup("rmi://hostB/Hello");
        // chamada remota
        print( hello_server.sayHello() );
    }
}
```



# Camadas

Aplicações e serviços

RMI e RPC

Protocolo Request-Reply  
Marshalling e eXternal Data Representation

TCP e UDP (sockets)

Middleware  
(camada de  
sessão/  
apresentação  
[OSI])

# Transmissão de dados

Dados em programas são **estruturados**

Enquanto isso, mensagens carregam informação **sequencial**:

Linearização/Restauração de dados

**Heterogeneidade** na representação de dados em computadores

Uso de um **formato externo comum**

Inclusão de uma identificação de arquitetura na mensagem

# Marshalling/Unmarshalling

## Marshalling:

Linearização de uma coleção de itens de dados estruturados

Tradução dos dados em formato externo (ex: **XDR** – eXternal Data Representation)

## Unmarshalling:

Tradução do formato externo para o local

Restauração dos itens de dados de acordo com sua estrutura

# Chamada de Procedimentos Remotos

## Remote Procedure Call (RPC)

Ideal: programar um sistema distribuído como se fosse centralizado

RPC objetiva permitir chamada de procedimento remoto como se fosse local, ocultando entrada/saída de mensagens

RPC: considerações em função da distribuição

evitar passagem de endereço e variáveis globais

novos tipos de erros

Ex.: tratamento de exceções

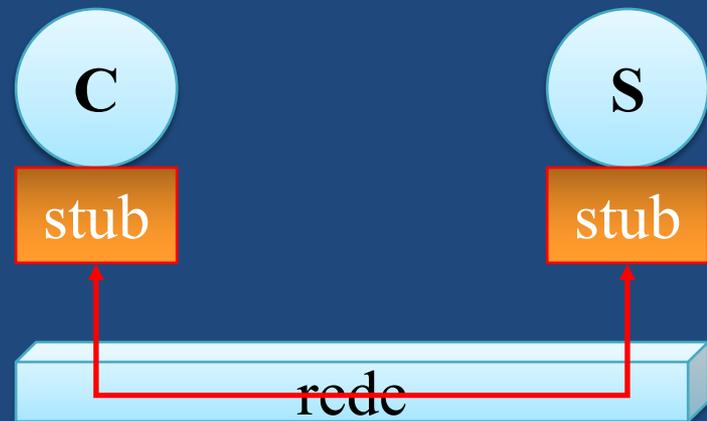
Se atraso de comunicação ➡ timeout

# RPC: processamento de interface

**Objetivo:** integração dos mecanismos de RPC com os programas cliente e servidor escritos em uma linguagem de programação convencional

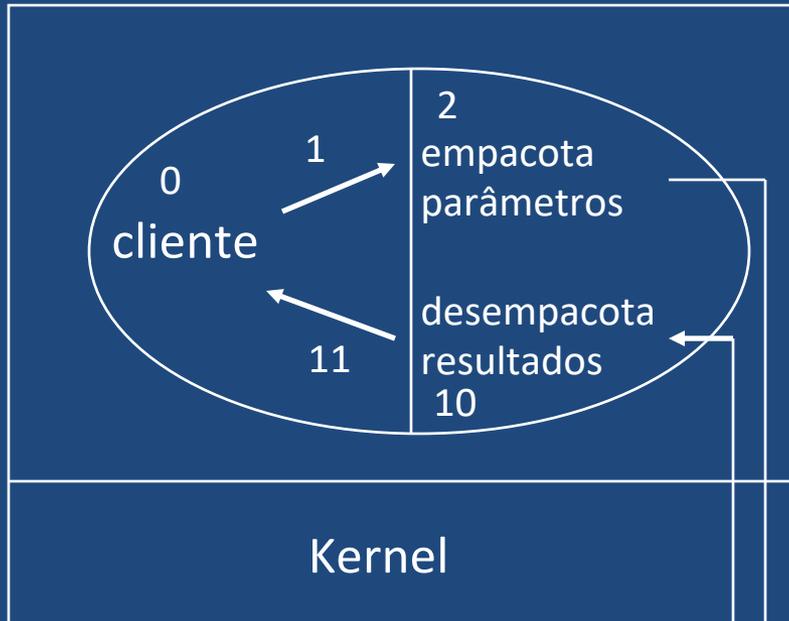
**Stubs (no cliente e no servidor): transparência de acesso**

tratamento de algumas exceções no local  
marshalling  
unmarshalling

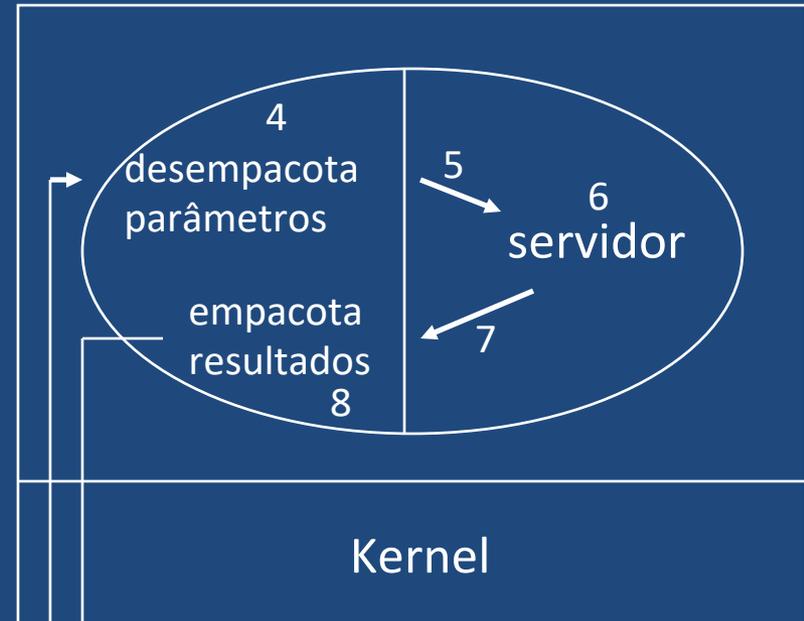


# Chamadas e mensagens em RPC

Máquina do Cliente



Máquina do Servidor



3

9

transporte de mensagens  
via rede

# Funções dos Stubs

## Client stub

1. intercepta a chamada
2. empacota os parâmetros (marshalling)
3. envia mensagem de request ao servidor (através do núcleo)

## Server stub

4. recebe a mensagem de request (através do núcleo)
5. desempacota os parâmetros (unmarshalling)
6. chama o procedimento, passando os parâmetros
7. empacota o resultado
8. envia mensagem de reply ao cliente (através do núcleo)

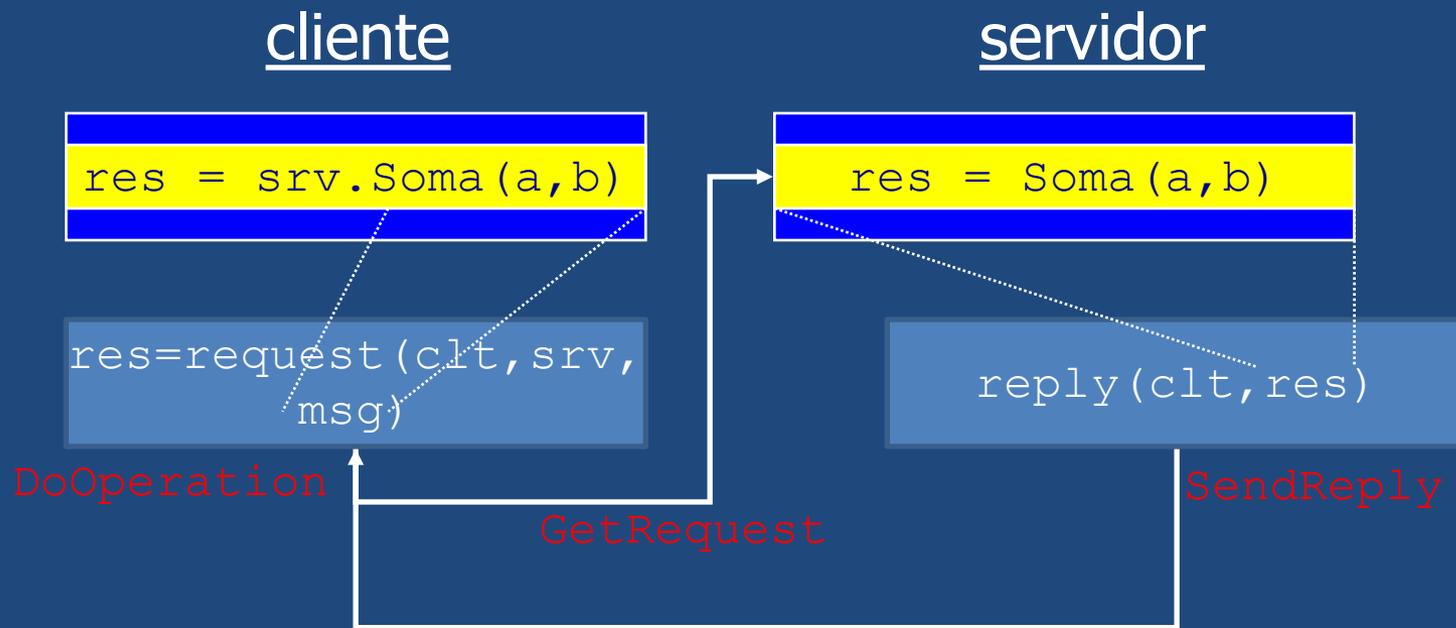
## Client stub

9. recebe a mensagem de reply (através do núcleo)
10. desempacota o resultado
11. passa o resultado para o cliente

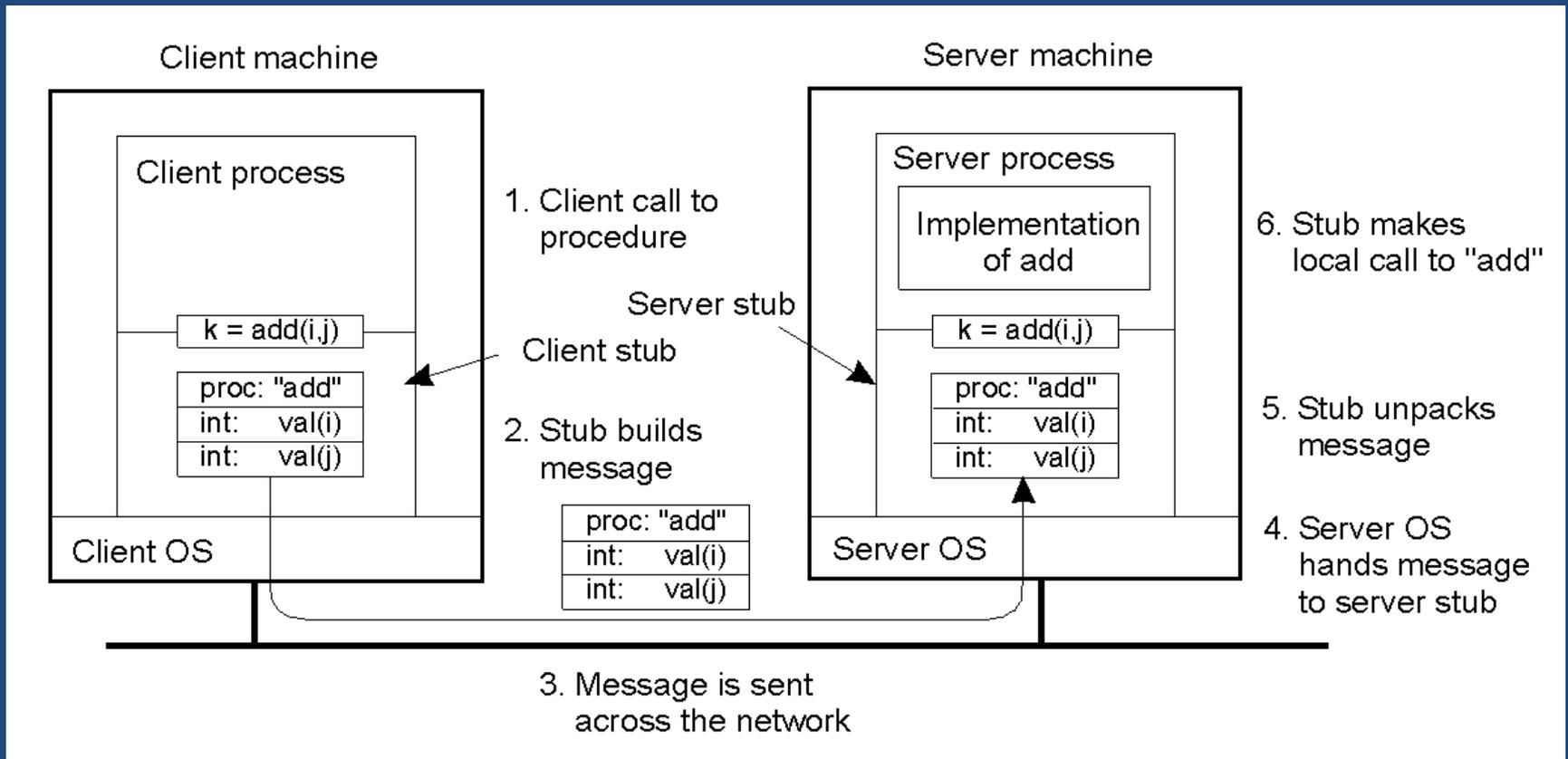
# RPC: tratamento da comunicação

Módulo de comunicação usa protocolo **pedido-resposta** / **request-reply** para troca de mensagens entre cliente e servidor

Mensagem de aplicação encapsulada em um request/reply



# RPC: Passagem de Parâmetros



RPC: ligação

(antes da comunicação...)

O mecanismo possui um binder para resolução de nomes, permitindo

Ligação dinâmica

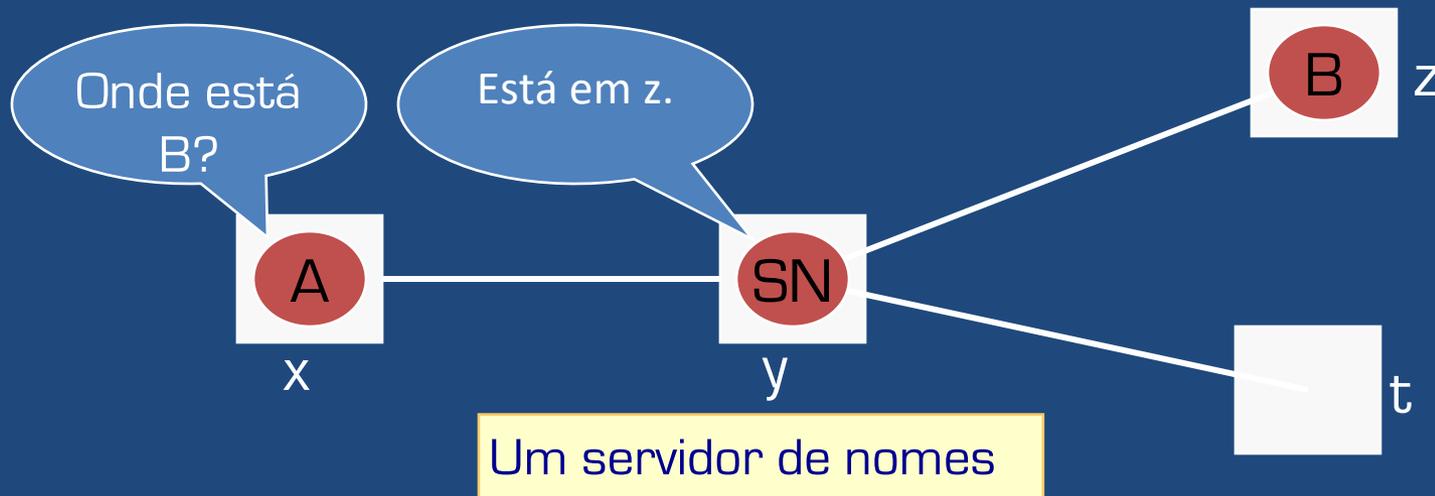
Transparência de localização

# Ligação Dinâmica e Nomes

Nomes podem ser a solução, mas na realidade os endereços físicos é que são necessários...

Daí, faz-se necessário mapear nomes em endereços... como **serviço**

Geralmente, algum agente intermediário tem este papel específico de resolvidor de nomes

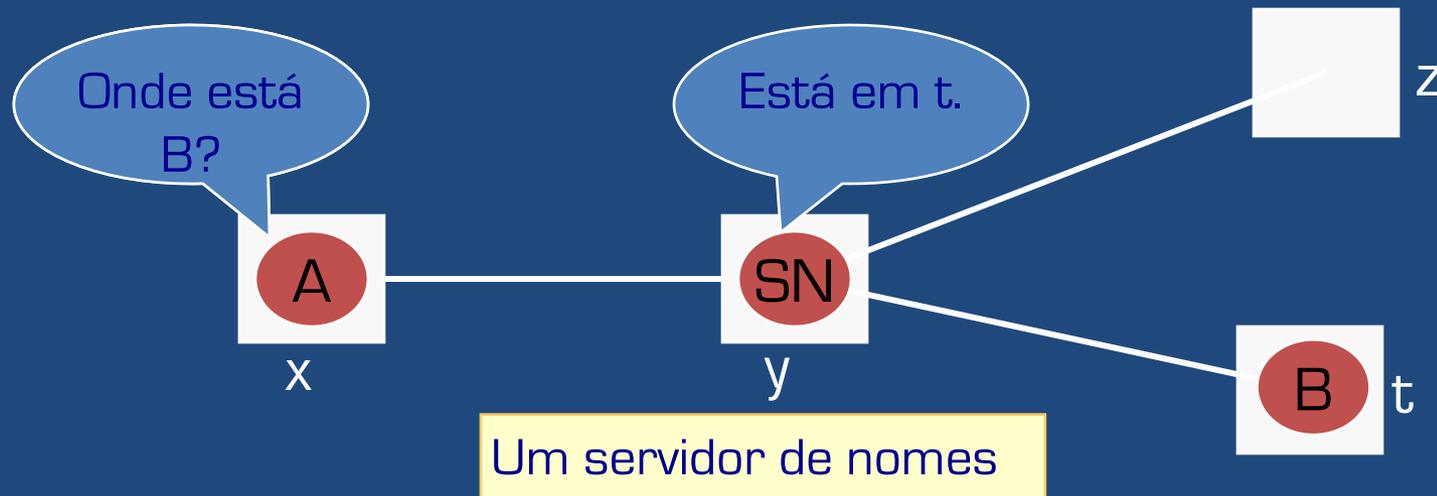


# Ligação Dinâmica e Nomes

Nomes podem ser a solução, mas na realidade os endereços físicos é que são necessários...

Daí, faz-se necessário mapear nomes em endereços... como **serviço**

Geralmente, algum agente intermediário tem este papel específico de resolvidor de nomes



# Servidores de Nomes

Os servidores de nomes, com este papel de mapeamento de nomes para endereços, contribuem para implementar a

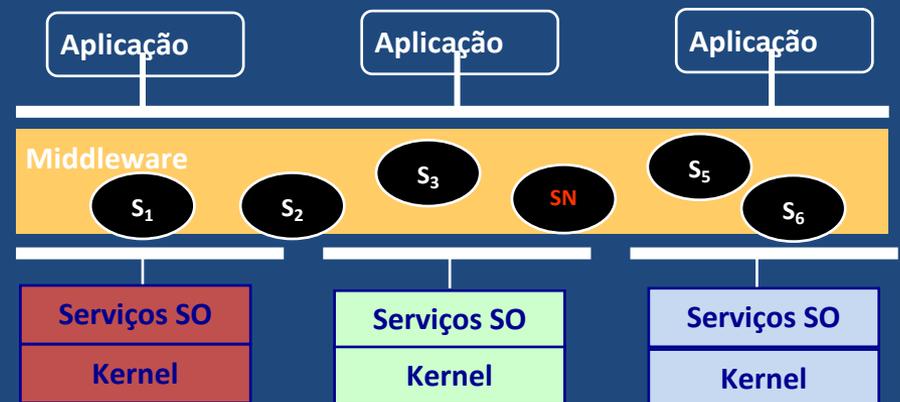
Transparência de localização

Os servidores de nomes são serviços executados no ambiente de suporte a aplicações distribuídas

Exs:

Rmiregistry: JavaRMI

tnameserv: CORBA (JDK)



# Exemplo: Servidor

```
import java.rmi.Naming;
```

1

Importa Serviço de Nomes do  
Middleware Java RMI (*Remote Method Invocation*);

```
public class CalculatorServer {
```

```
    public CalculatorServer() {
```

```
        try {
```

```
            Calculator c = new CalculatorImpl();
```

2

Cria referência do serviço  
a oferecer;

```
            Naming.rebind("rmi://localhost:1099/CalculatorService", c);
```

```
        } catch (Exception e) {
```

```
            System.out.println("Trouble: " + e);
```

```
        }
```

```
    }
```

```
    public static void main(String args[]) {
```

```
        new CalculatorServer();
```

```
    }
```

```
}
```

Registra (exporta)  
referência "c" com  
nome "...CalculatorService"  
no servidor de nomes.

# Exemplo: Cliente

```
import java.rmi.Naming;
```

```
...  
public class CalculatorClient {  
    public static void main(String[] args) {  
        try {  
            Calculator c = (Calculator) Naming.lookup( "rmi://localhost  
                                                    /CalculatorService");  
            System.out.println( c.sub(4, 3) );  
            System.out.println( c.add(4, 5) );  
            System.out.println( c.mul(3, 6) );  
            System.out.println( c.div(9, 3) );  
        }  
        catch ...  
    }  
}
```

Procura (importa) no servidor de nomes o endereço do servidor usando o seu nome.

3

4

Obtido o endereço, o cliente está ligado (conectado) ao servidor e passa a chamar as operações do serviço.

# Objetivos de serviços de nomes

Ser **escalar**

Ter um **longo tempo de vida**

Ser altamente **disponível**

Ter **isolamento de falhas**

Tolerar **desconfiança**

# Serviço de *Trading*

em vez de apenas fornecer uma maneira para os clientes procurarem servidores disponíveis **por nome**, um serviço de *trading* permite aos servidores o registro de seus recursos e os clientes os encontram com base em suas **propriedades**, especificadas por meio de uma linguagem de restrição simples

# *Naming vs Trading*

## Serviço de Nomes

- Localiza componentes por nomes
- Semelhantes a “páginas brancas”

## Serviço de *Trading*

- Localiza componentes por características do serviço
- Semelhantes a “páginas amarelas”

# COMPUTAÇÃO DISTRIBUÍDA

CONCLUSÕES

# Características marcantes

Concorrência de componentes – possibilidade de  
paralelismo

Compartilhamento de recursos

Componentes falham de forma independente – falha  
parcial

# Próximas Datas

19/11, ter	Aula prática: JavaRMI e OpenMP – <b>Lab G2</b>
21/11, qui	<b>Revisão 1o. EE – D002</b>
26/11, ter	<b>Exercício</b> de RPC/RMI – <b>Lab G2</b>
28/11, qui	<b>3o. EE:</b> Sist. Arquivos, E/S, Sist. Distribuídos – D002 <a href="https://www.cin.ufpe.br/~cagf/if677/2019-2/slides/">https://www.cin.ufpe.br/~cagf/if677/2019-2/slides/</a>
05/12, qui	<b>Revisão de Avaliações/Notas</b> – <b>Sala C-125</b>
10/12, ter	<b>PROVA FINAL</b>