Computação Distribuída

Odorico Machado Mendizabal

Universidade Federal de Santa Catarina – UFSC
Departamento de Informática e Estatística – INE

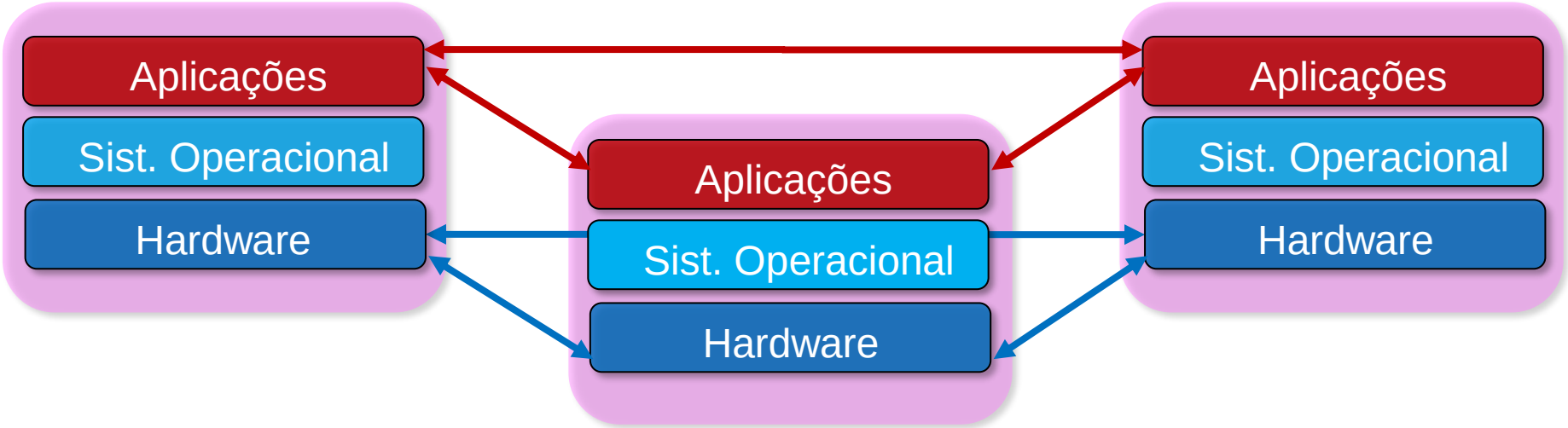
Middleware
Suporte a Sistemas Distribuídos

Suporte do SO aos Sistemas Distribuídos

Não faz o gerenciamento de recursos distribuídos:

• Aplicações são responsáveis pelo gerenciamento e utilização de recursos distribuídos
  - Controle de concorrência, segurança, etc.

• Comunicação feita através de protocolos de rede (ex. TCP, UDP)



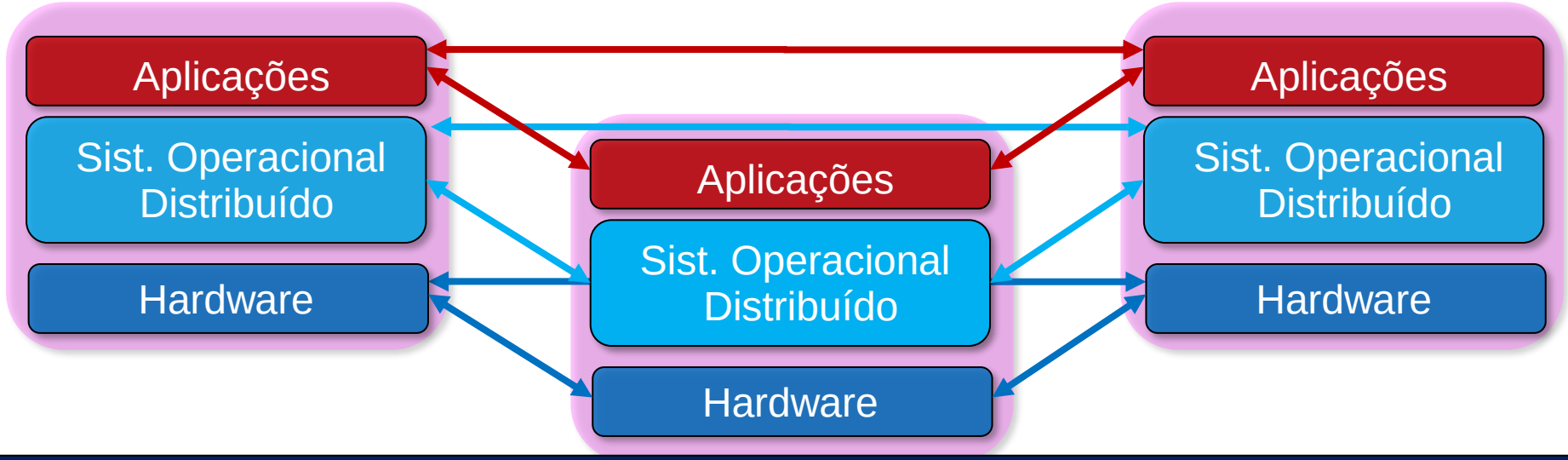
Suporte do SO aos Sistemas Distribuídos

Sistema Operacional Distribuído oferece serviços de gerenciamento de recursos distribuídos:

- Ex. servidor de arquivos distribuídos

• Oferecem normalmente transparência de localização, concorrência, segurança, persistência e replicação

• Ex.: Amoeba, Mach, Sprite, PLAN 9, Nanvix (arquiteturas *lightweight manycore*)

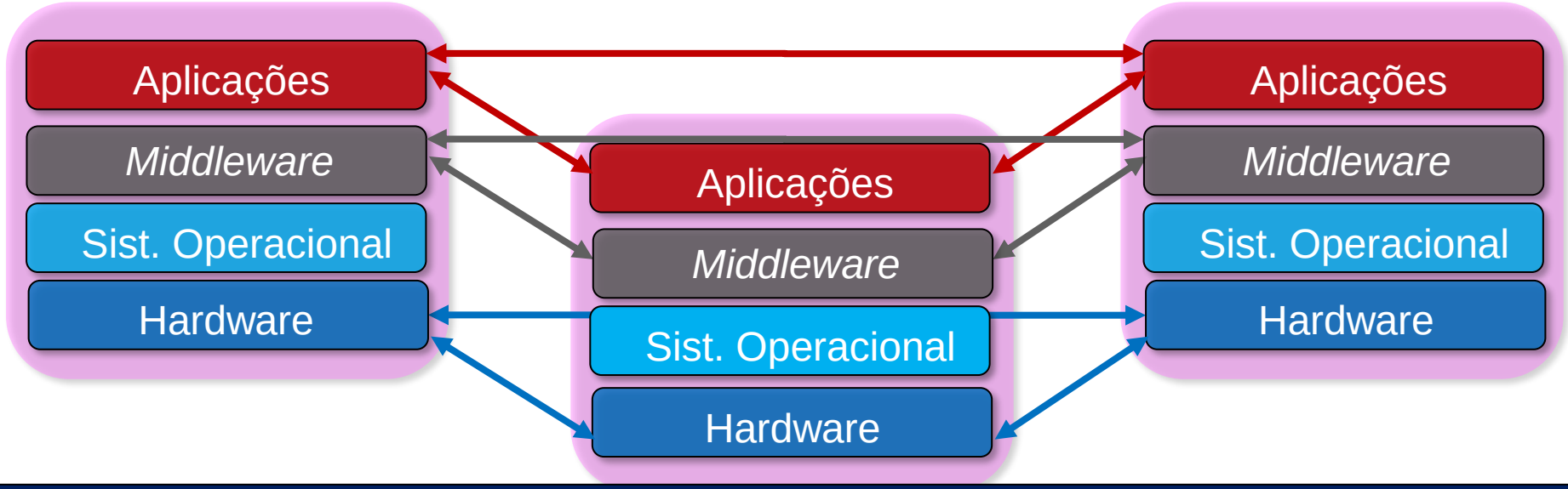


Suporte de *Middleware* para Sistemas Distribuídos

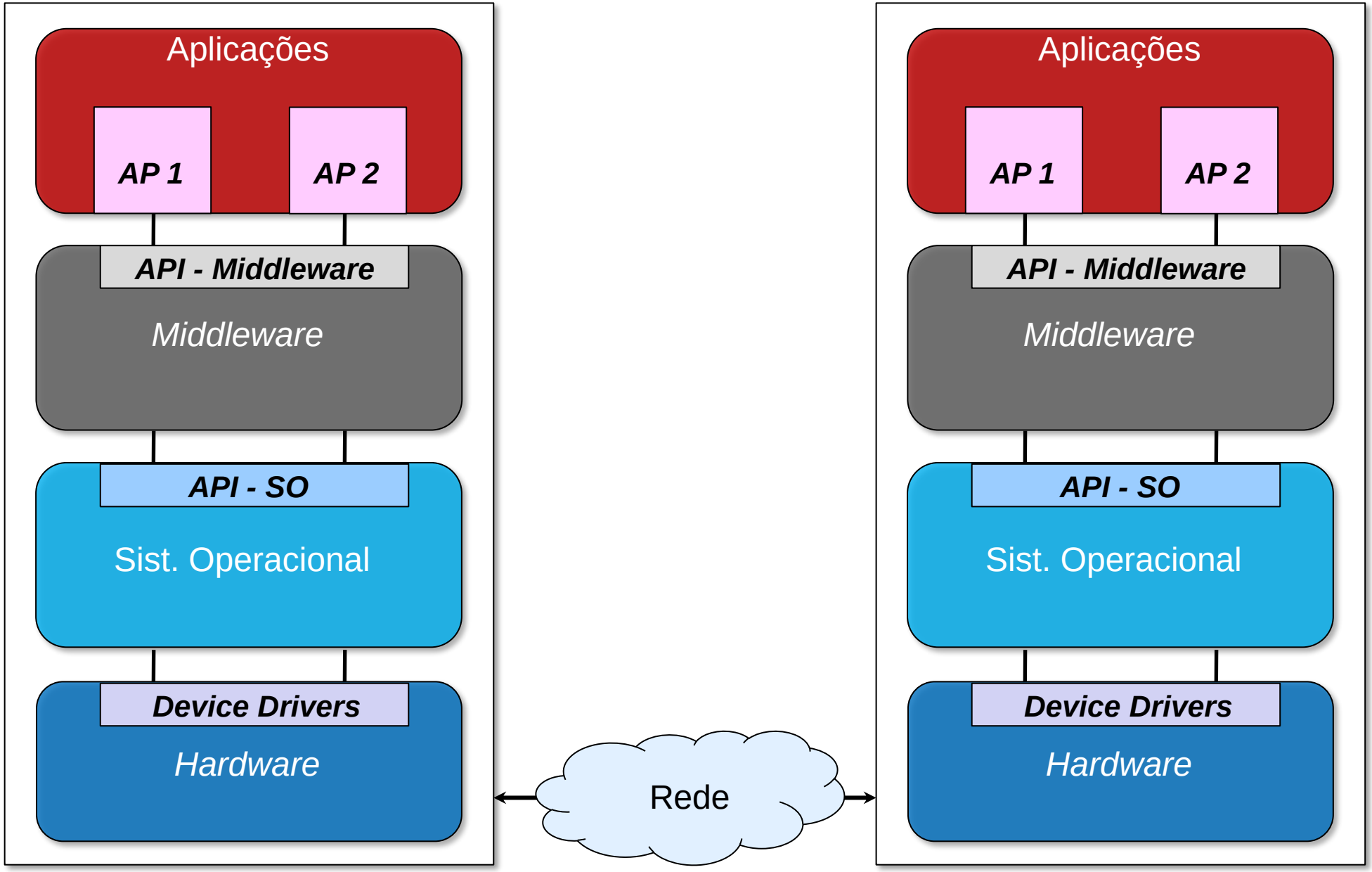
Middleware é uma camada de *software*, entre aplicações e plataforma, que oferece serviços especializados:

- Mascara heterogeneidade do ambiente (oferece transparência de acesso, localização, segurança, disponibilidade, persistência, concorrência, falhas)

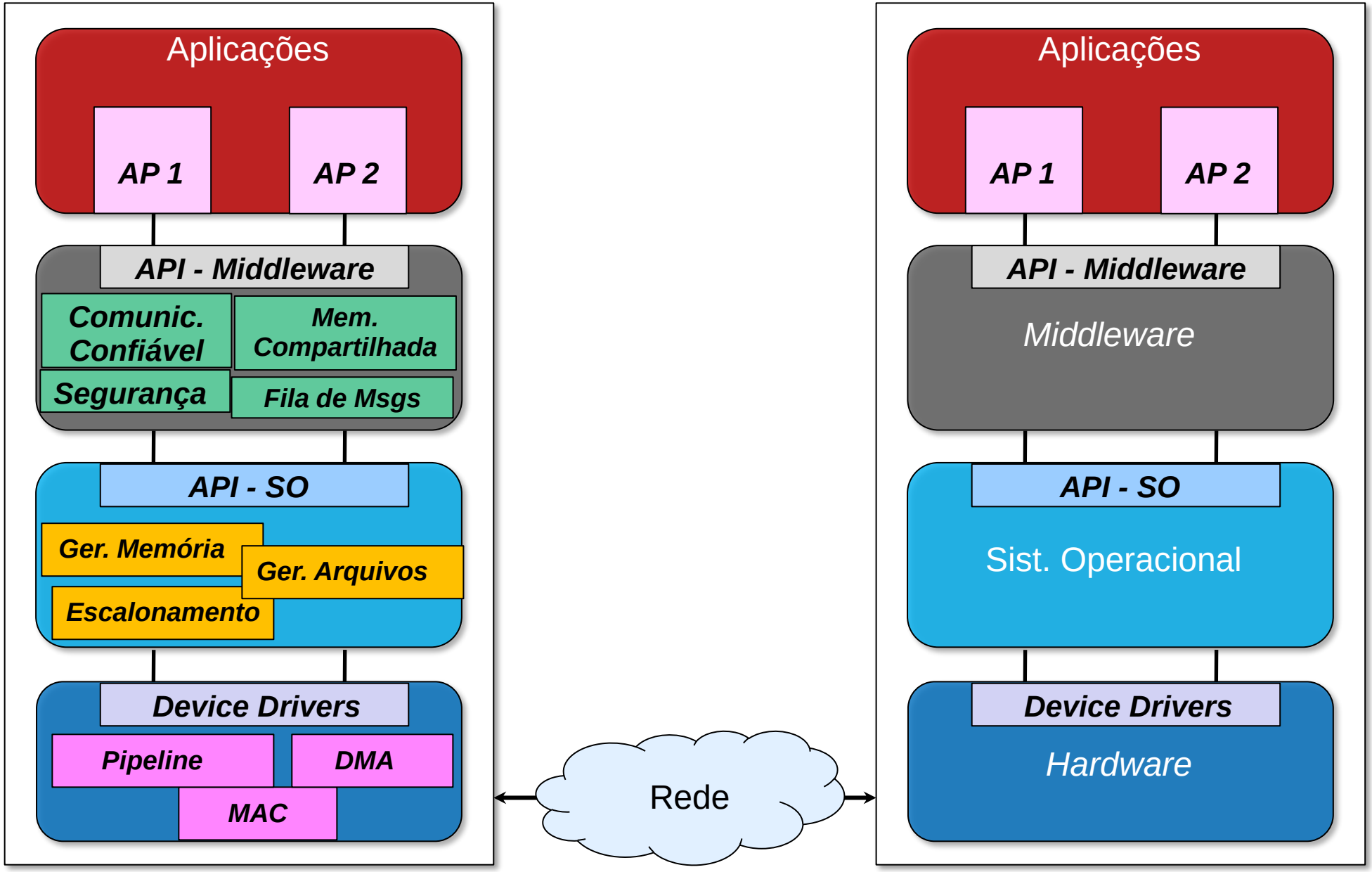
• Ex.: CORBA, Java-RMI, DCOM, *Enterprise Service Bus*



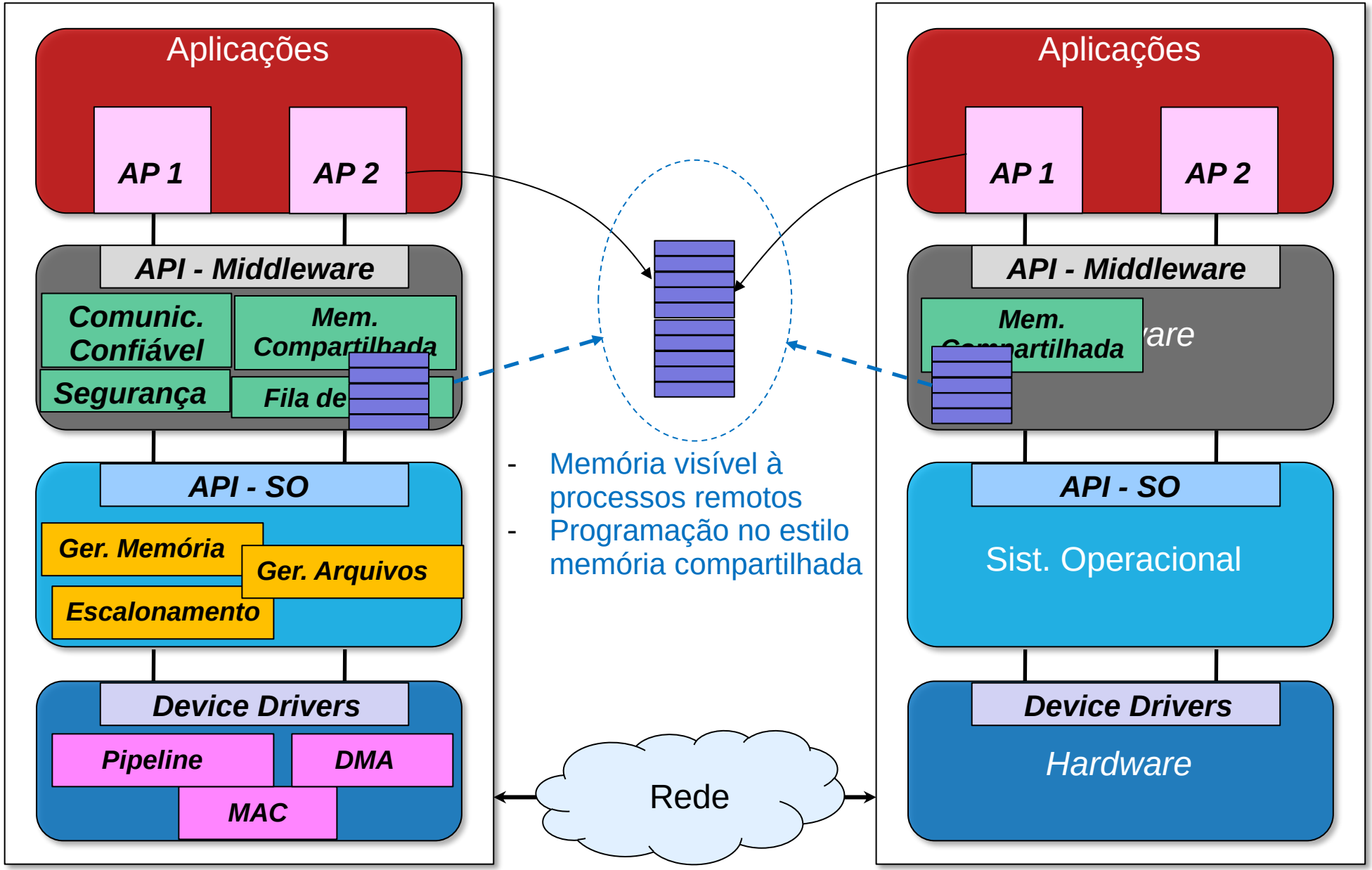
Suporte de *Middleware* para Sistemas Distribuídos



Suporte de *Middleware* para Sistemas Distribuídos



Suporte de *Middleware* para Sistemas Distribuídos



Middleware – Breve Histórico

• O termo surgiu inicialmente nos anos 80, associado a um *software* de gerenciamento de conexões de rede

• Nos anos 90 o termo se difundiu, com utilização e criação de paradigmas e serviços que tornavam mais fácil o gerenciamento e implementação de sistemas distribuídos

• Exemplos de *middleware*:

- Cronus, Cloud – Objetos Distribuídos
- RPC – Chamadas a Procedimentos Remotos
- QuO – Qualidade de Serviço para Objetos Distribuídos
- CORBA – Plataforma para objetos distribuídos com qualidade de serviço
- Linda, Jini – Implementação de um Espaço de Tuplas
- JGroups, Ensemble, Isis – Plataformas para comunicação em grupo
- MOM (*Message Oriented Middleware*) – Soluções baseadas em filas de mensagens

Middleware – Breve Histórico

• O termo surgiu inicialmente nos anos 80, associado a um *software* de gerenciamento de conexões de rede

• Nos anos 90 o termo se difundiu, com utilização e criação de paradigmas e serviços que tornavam mais fácil o gerenciamento e implementação de sistemas distribuídos

• Exemplos de *middleware*:

- Cronus, Cloud – Objetos
- RPC – Chamadas a Proc
- QuO – Qualidade de Serv
- CORBA – Plataforma para objetos distribuídos com qualidade de serviço
- Linda, Jini – Implementação de um Espaço de Tuplas
- JGroups, Ensemble, Isis – Plataformas para comunicação em grupo
- MOM (*Message Oriented Middleware*) – Soluções baseadas em filas de mensagens

Você Sabia?

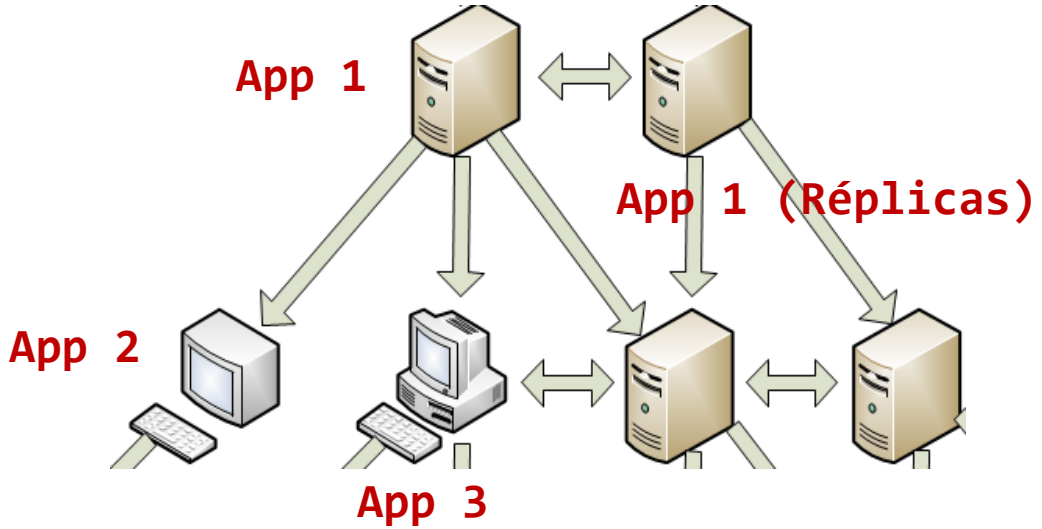
CURIOSIDADE:
- Apesar do termo *middleware* ser popular em meados da década de 1990, ele foi provavelmente mencionado pela primeira vez em um relatório sobre uma conferência de engenharia de software da OTAN, editado por Peter Naur e Brian Randell em outubro de 1968

Middleware – Características

- Convergência de conceitos e paradigmas
  - Muitas vezes o *middleware* não implementa exclusivamente um único paradigma
  - Ex. 1: Objetos distribuídos e RPC em um mesmo *middleware*
  - Ex. 2: Objetos distribuídos, serviços de eventos e filas de mensagens (MOM) em um mesmo *middleware* (implementações CORBA)

- Muito usado para integração de sistemas legados
  - Ex. 1: Aplicação Servidor em C integrada a clientes Java
  - Ex. 2: Integrar aplicação de *mainframe* com outras aplicações distribuídas

(canary releases)




```
App 1: C++, Linux, PC

App 2: Cobol, Guardian, Tandem

App 3: Java, Windows Server, PC
```

Middleware – Características

- É comum implementação em modelo em camadas ou orientando a serviços
  - Serviços de camadas mais baixas podem ser acessados por aplicação ou por serviços de camadas mais altas
  - Não confundir camadas de rede: *Middleware* normalmente implementado na camada de aplicação

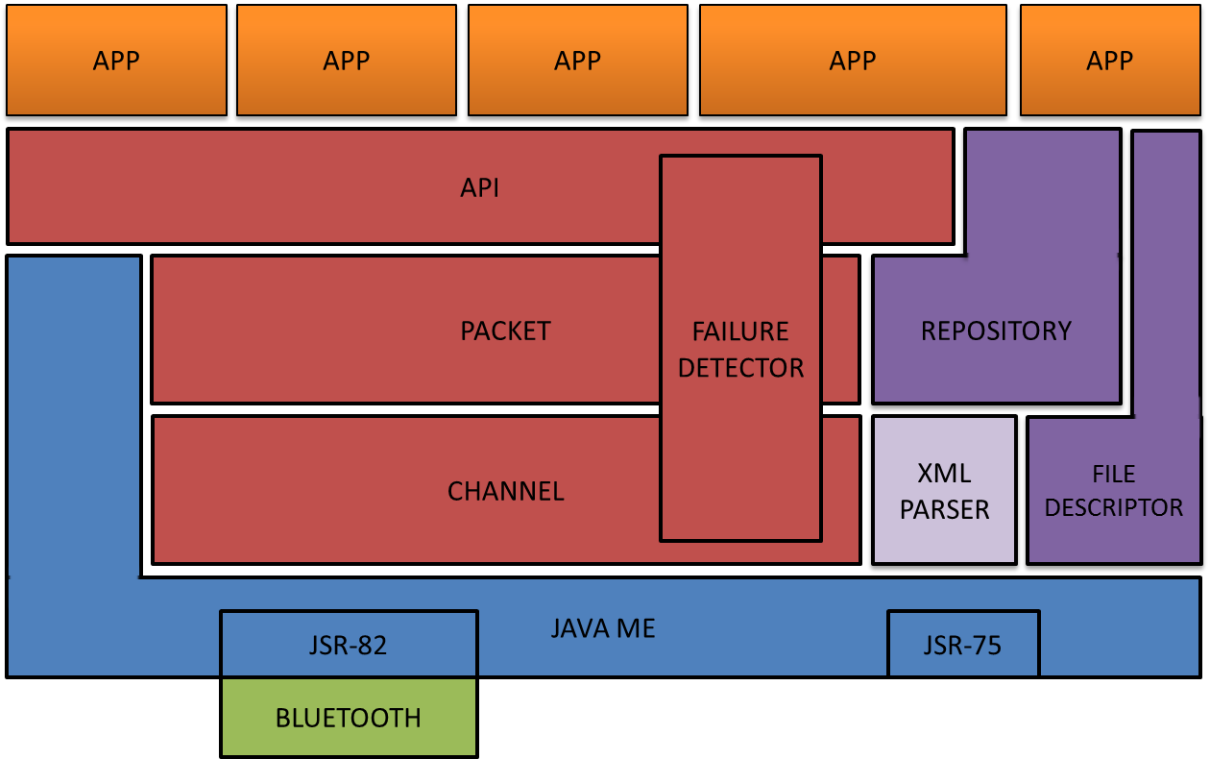


Figura extraída de BlueGroups (Salvá, Senna, Prisco, Mendizabal)

Middleware – Utilização

- Implementação de aplicações com uso de funções de *middleware*
  - Não necessita aprender nova linguagem de programação
  - Para acessar funcionalidades do *middleware*:
    i) Através de uma biblioteca (C, C++, Java, etc..)
    ii) Linguagem de definição externa (IDL – *Interface Definition Language*)
    iii) Recursos nativos da plataforma (ex. Java RMI, que já está incluso na JVM)

Estrutura e implementação de *Middleware*

Middleware – Organização

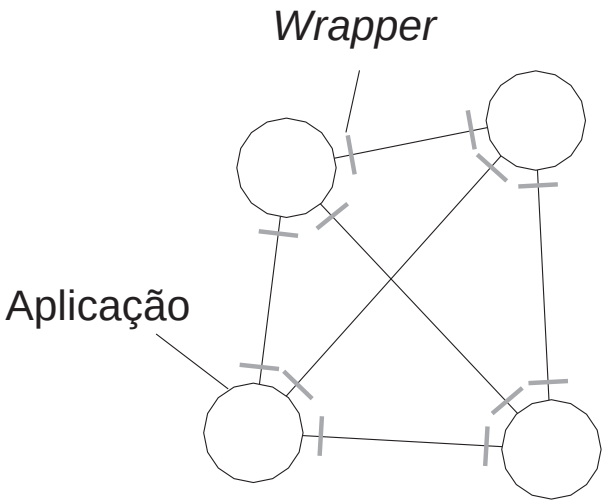
- Problema: As interfaces oferecidas por um componente legado provavelmente não são adequadas para todas as aplicações

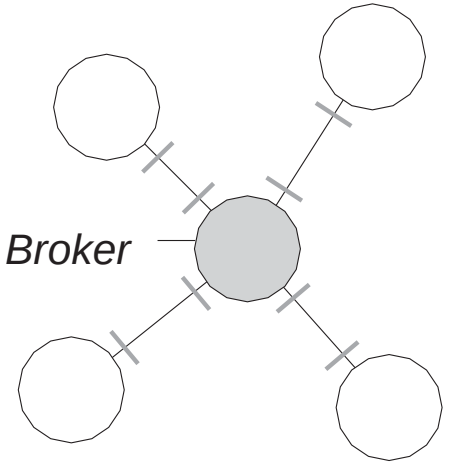
- Solução: Um *wrapper* (empacotador) ou adaptador oferece uma interface aceitável para uma aplicação cliente. Suas funções são transformadas naquelas disponíveis no componente

- Diferentes padrões de projeto podem ser adotados
  - Mais comuns: *wrappers* e adaptadores / interceptadores

Middleware – Organização (*wrappers*)

- Duas abordagens: 1:1 ou usando um *broker (agente)*





Complexidade com *N* aplicações

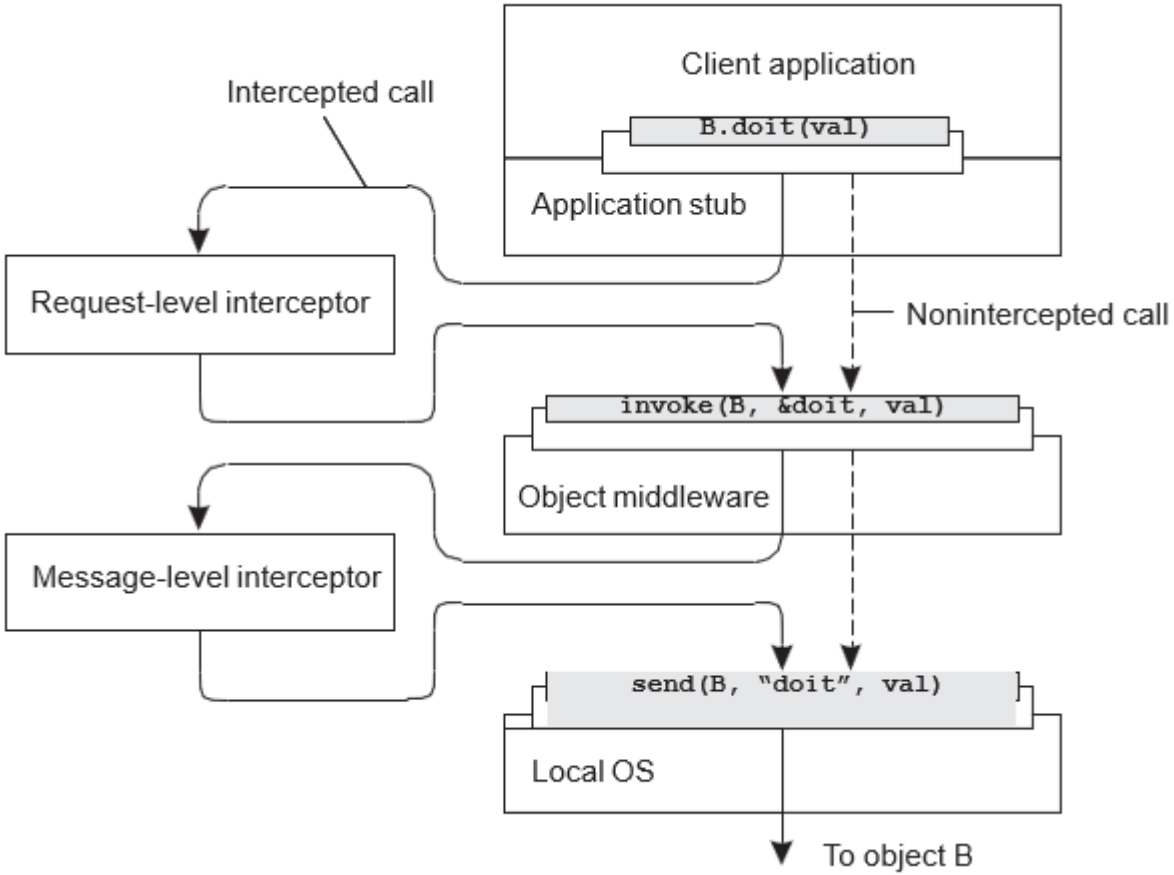
- 1-para-1: requer $N \times (N - 1) = O(N^2)$ *wrappers*
- *broker*: requer $2N = O(N)$ *wrappers*

Middleware – Organização – Interceptadores

Problema: Middleware oferece soluções boas para a maioria das aplicações

⇒

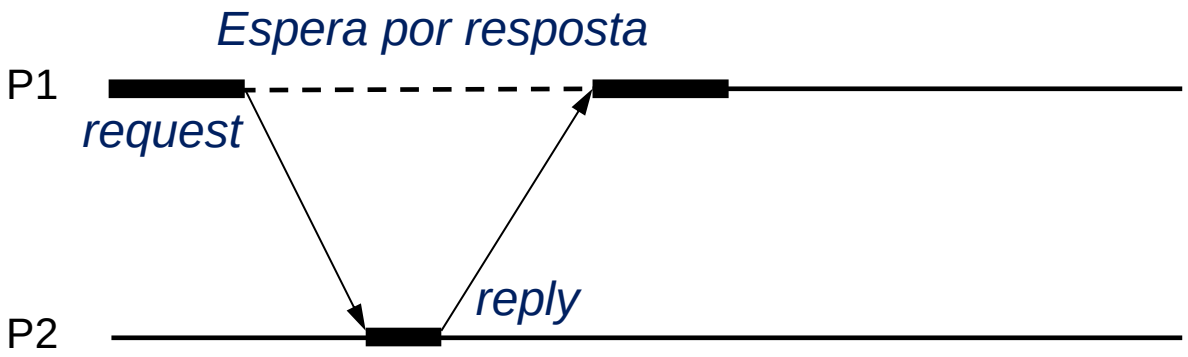
Pode ser necessário adaptar o comportamento para aplicações específicas



Suporte de *Middleware* à Comunicação

Modelos Síncronos e Assíncronos

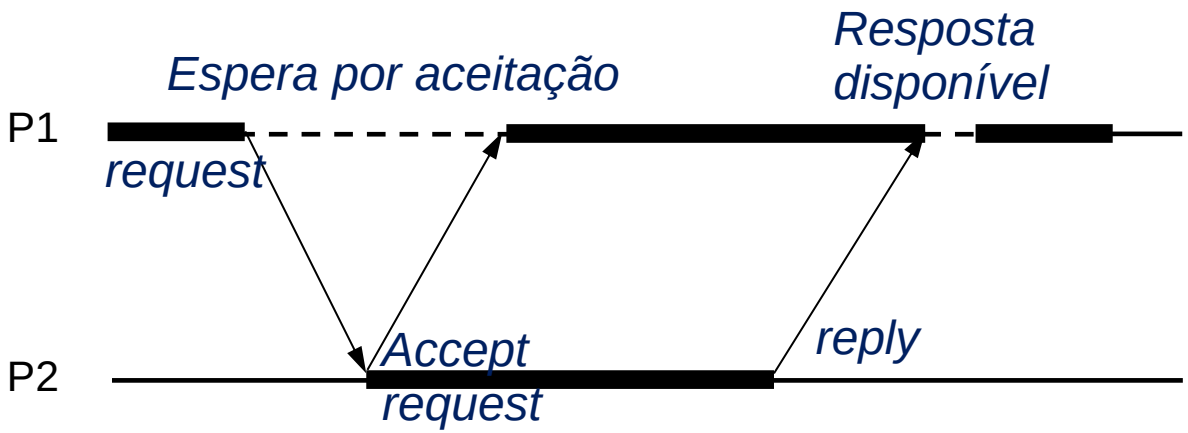
Modelo Síncrono



Comunicação Síncrona

operações *send* e *receive* causam bloqueio

Modelo Assíncrono

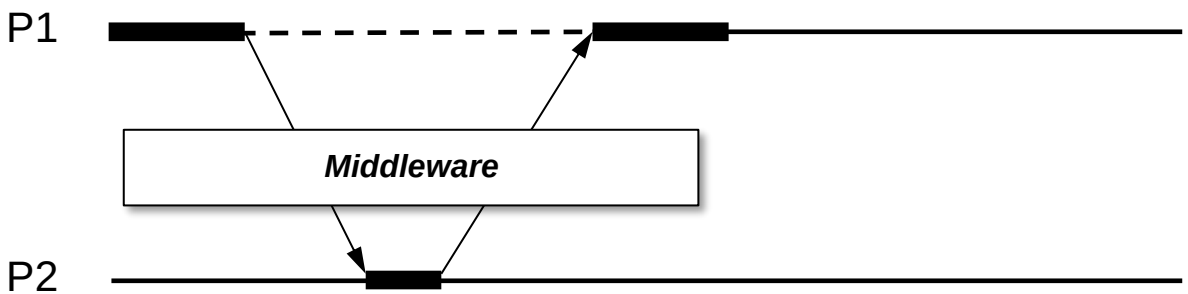


Comunicação Assíncrona

send não causa bloqueio, *receive* pode causar bloqueio ou não (*receive* não bloqueante – mensagens são armazenadas em um *buffer* em *background*)

Modelos Síncronos e Assíncronos com Garantias

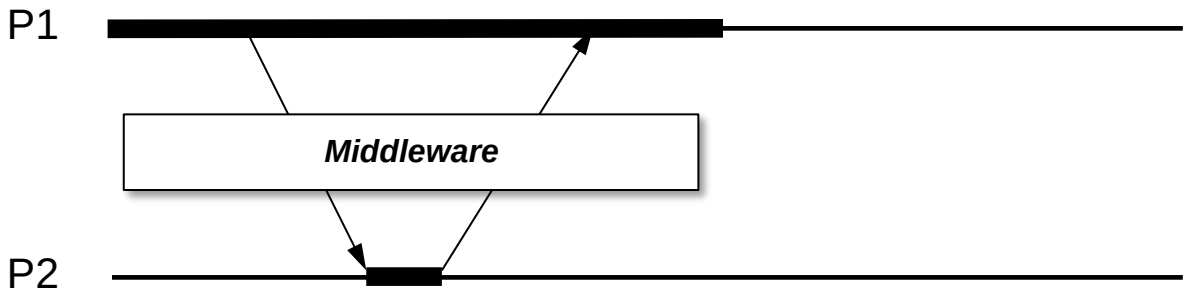
Modelo Síncrono



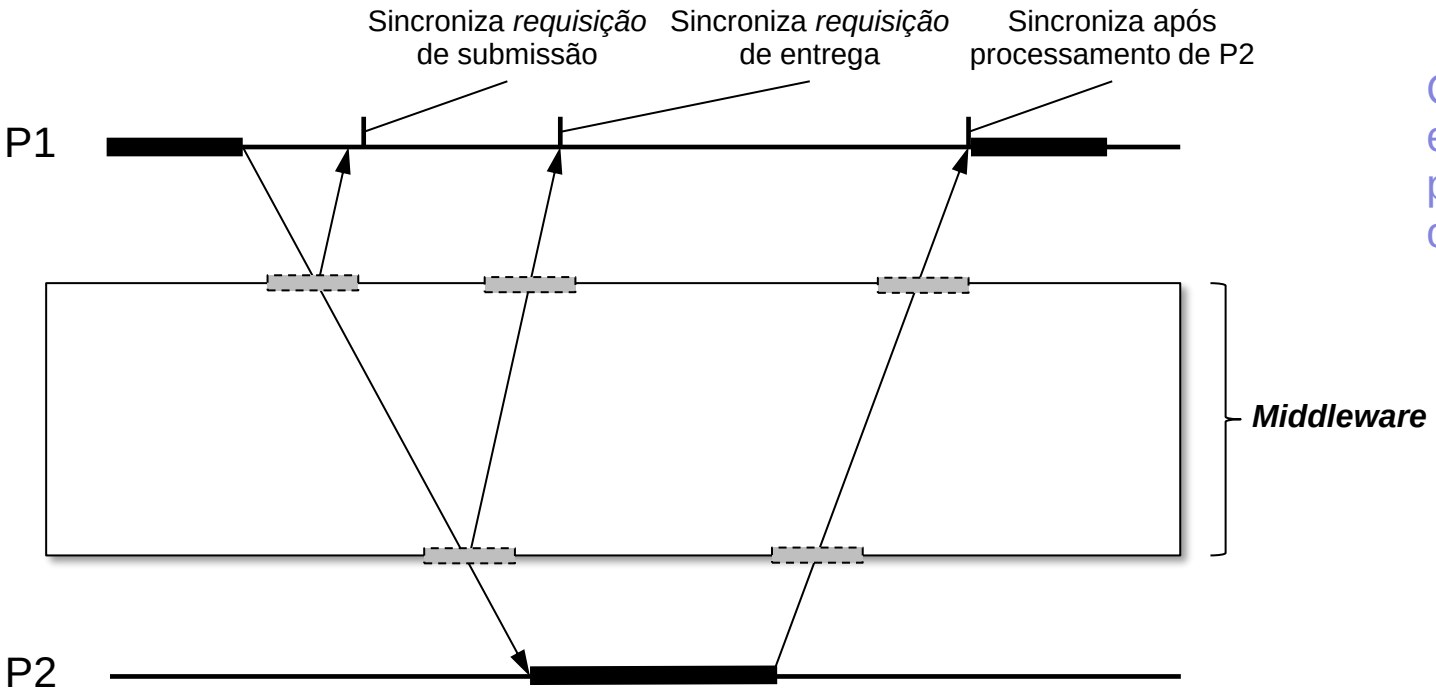
Serviços de *Middleware* podem implementar/oferecer certas garantias nos canais de comunicação:

- Segurança (ex. criptografia)
- Confiança (garantia de entrega)
- Etc.

Modelo Assíncrono



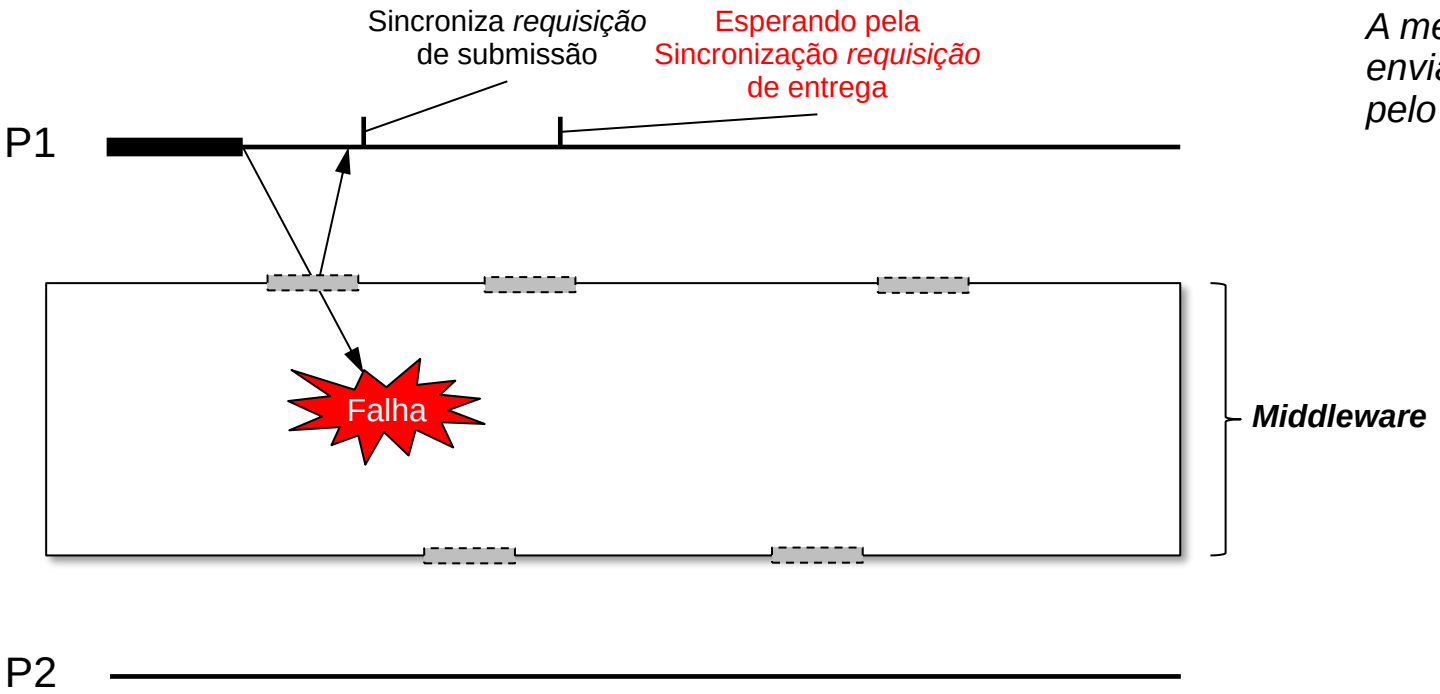
Modelos Síncronos – Transiente vs. Persistente



Os avisos de submissão, entrega e processamento podem ser implementados, ou não, pelo *Middleware*

Comunicação Transiente: Mensagem é descartada quando não pode ser entregue ao destinatário

Modelos Síncronos – Transiente vs. Persistente

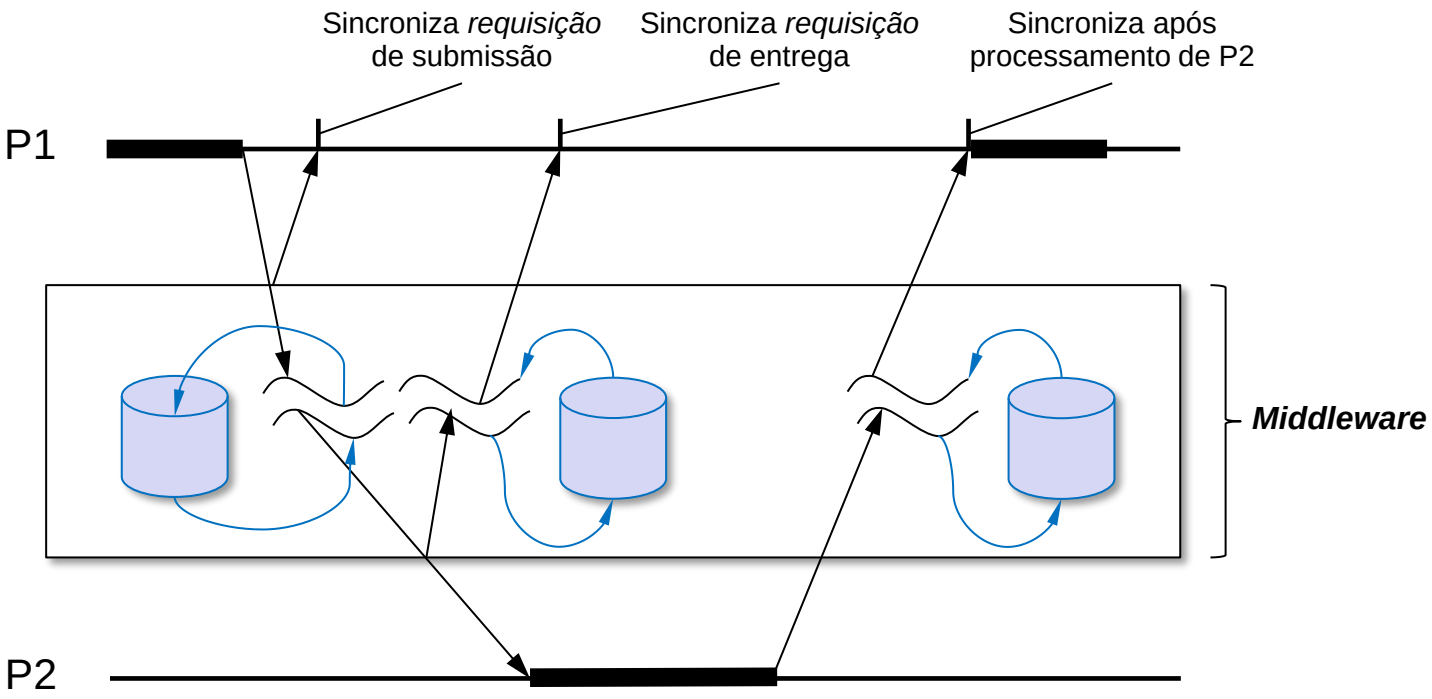


A mensagem originalmente enviada não será recuperada pelo Middleware

Comunicação Transiente: Mensagem é descartada quando não pode ser entregue ao destinatário

Alguns motivos pelo qual a falha pode ocorrer:
- Serviço de *Middleware* indisponível
- Problemas de comunicação com serviço de *Middleware*
- *Buffer Overflow* (Escassez da memória usada para armazenamento de mensagens)

Modelos Síncronos – Transiente vs. Persistente



Comunicação Persistente: Mensagem é armazenada por tanto tempo quanto necessário, até que seja entregue

Mesmo que o serviço de *Middleware* torne-se indisponível, quando este se recuperar, ele pode retransmitir as mensagens que estão armazenadas (pendentes)

Algumas Observações

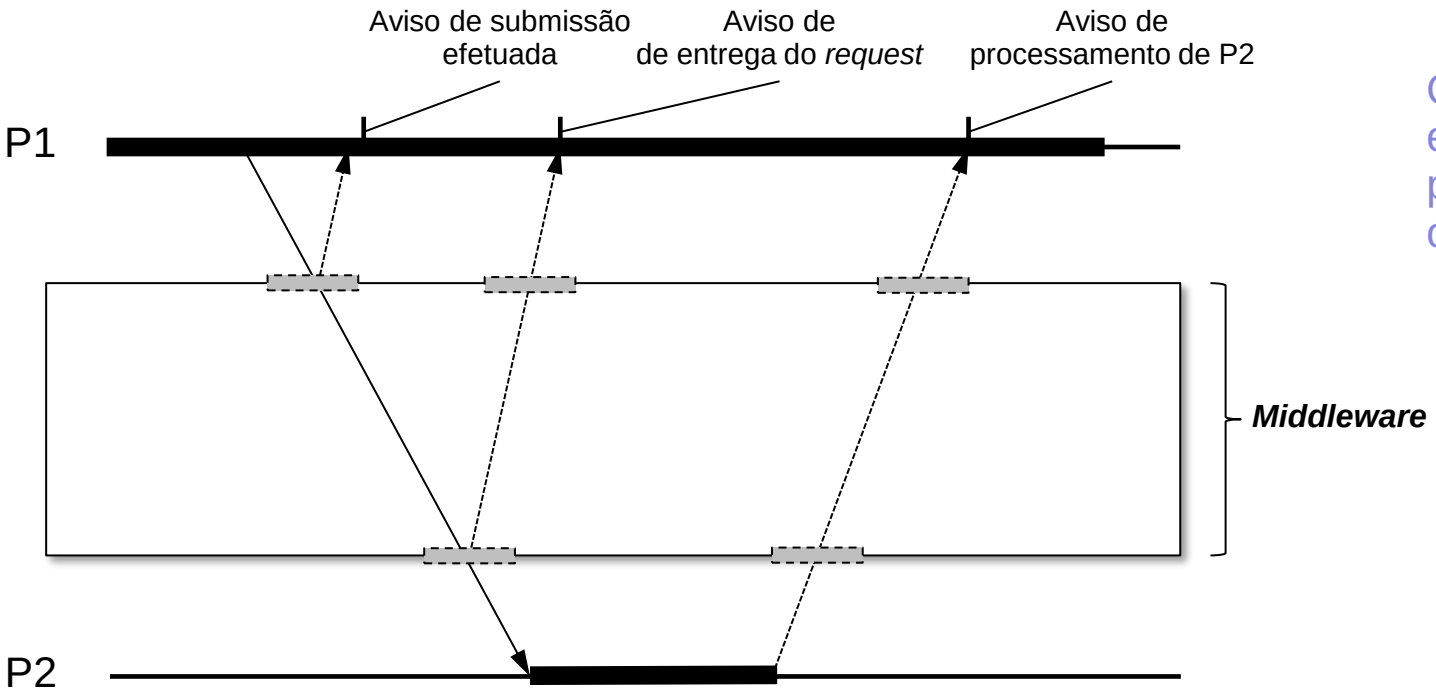
Aplicações cliente/servidor normalmente baseiam-se no modelo de **comunicação síncrono transiente**:

- Cliente e servidor precisam estar ativos durante comunicação
- Cliente envia mensagem e fica bloqueado até receber um retorno (não pode executar outra atividade enquanto espera)
- Servidor responsável por atender e responder requisições
  - *Comportamento reativo, espera pelo recebimento de mensagens e responde*

Desvantagens

- Cliente não pode prosseguir processando enquanto espera por respostas
- Falhas devem ser recuperadas imediatamente, pois o cliente está ocioso (bloqueado) esperando
- Esta abordagem pode não ser adequada (ex. email)

Modelos Assíncronos – Transiente vs. Persistente

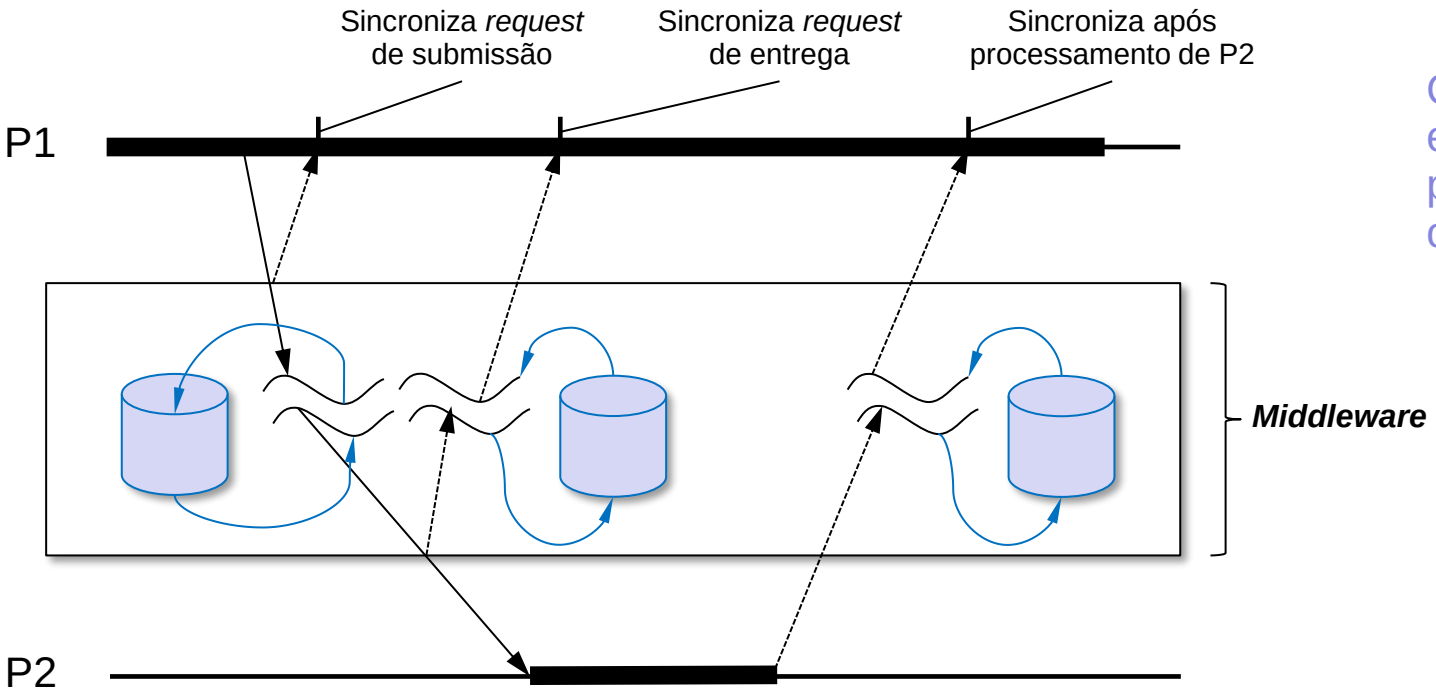


Os avisos de submissão, entrega e processamento podem ser implementados, ou não, pelo *Middleware*

Comunicação Transiente: Mensagem é descartada quando não pode ser entregue ao destinatário

Observe que P1 não fica bloqueado, independentemente de receber ou não resposta de P2.

Modelos Assíncronos – Transiente vs. Persistente



Os avisos de submissão, entrega e processamento podem ser implementados, ou não, pelo *Middleware*

Comunicação Persistente: Mensagem é armazenada por tanto tempo quanto necessário, até que seja entregue

Mesmo que o serviço de *Middleware* torne-se indisponível, quando este se recuperar, ele pode retransmitir as mensagens que estão armazenadas (pendentes)

A grande diferença é que P1 não bloqueia ao enviar mensagens para P2.

Algumas Observações

- Remetente não precisa esperar por avisos de recebimento ou retorno de mensagens para prosseguir seu processamento

- Processos tornam-se mais autônomos (modelos diferentes do cliente/servidor)

- No modelo assíncrono com persistência, não há necessidade dos processos estarem executando a todo instante.
    - Mensagens podem ser entregues posteriormente, sem que requisitante bloqueie

- Modelos de *middleware* orientados à mensagens implementam em alto nível o modelo de **comunicação assíncrono persistente**

Middleware – Outras funcionalidades

Além de sincronia e persistência, funcionalidades diversas podem ser oferecidas pelo Middleware

• Integração – intermediação e adaptação para permitir acesso a funcionalidades com APIs incompatíveis

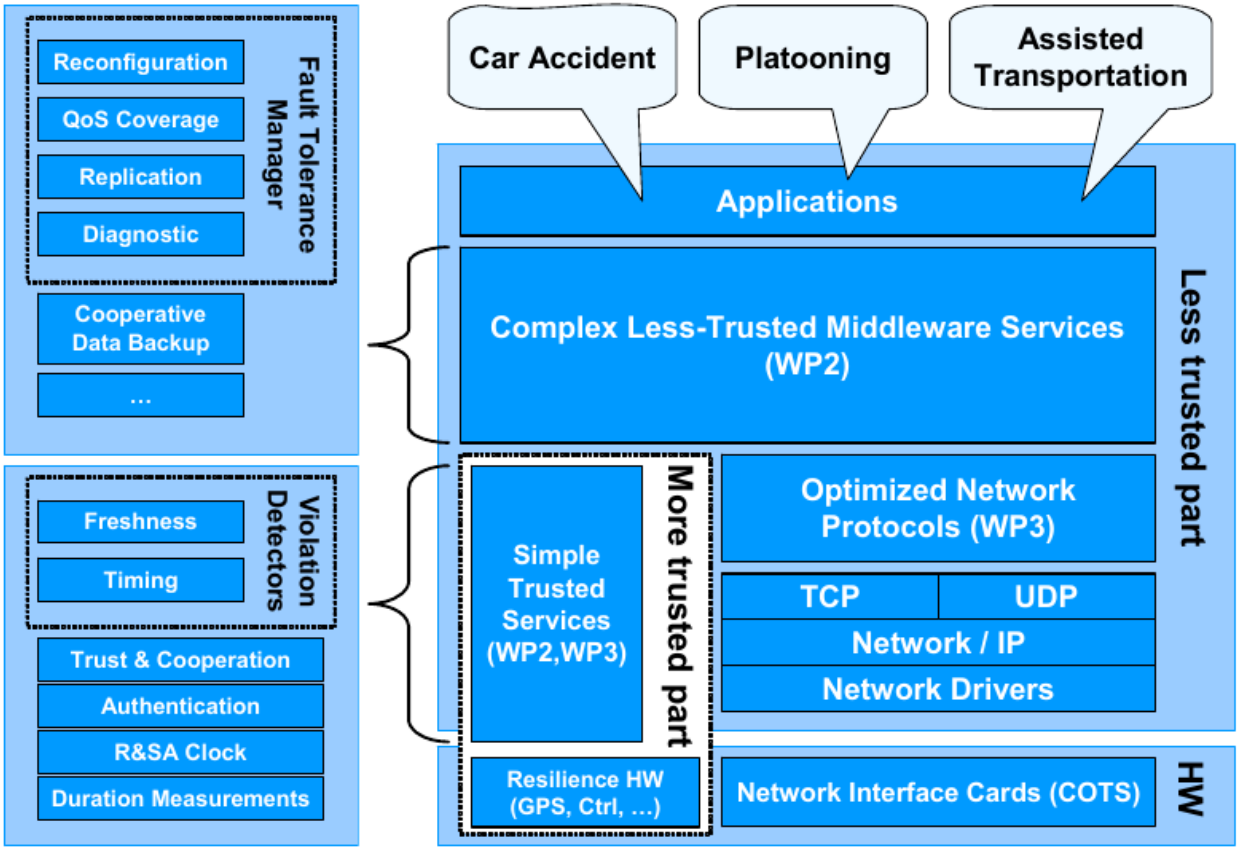
• QoS (Qualidade de Serviço) – Adaptação com base em SLAs (*Service Level Agreement*)

• Confiabilidade:
  - Tolerância a Falhas – Manter serviços altamente disponíveis (uso de redundância e replicação)
  - *Trustworthness* – garantias de autenticidade, integridade e não-repúdio de dados

• Atualidade de dados/mensagens (*timeliness*) – Garantia de que dados ou mensagens respeitam requisitos temporais

Middleware – Exemplos de estudos de caso

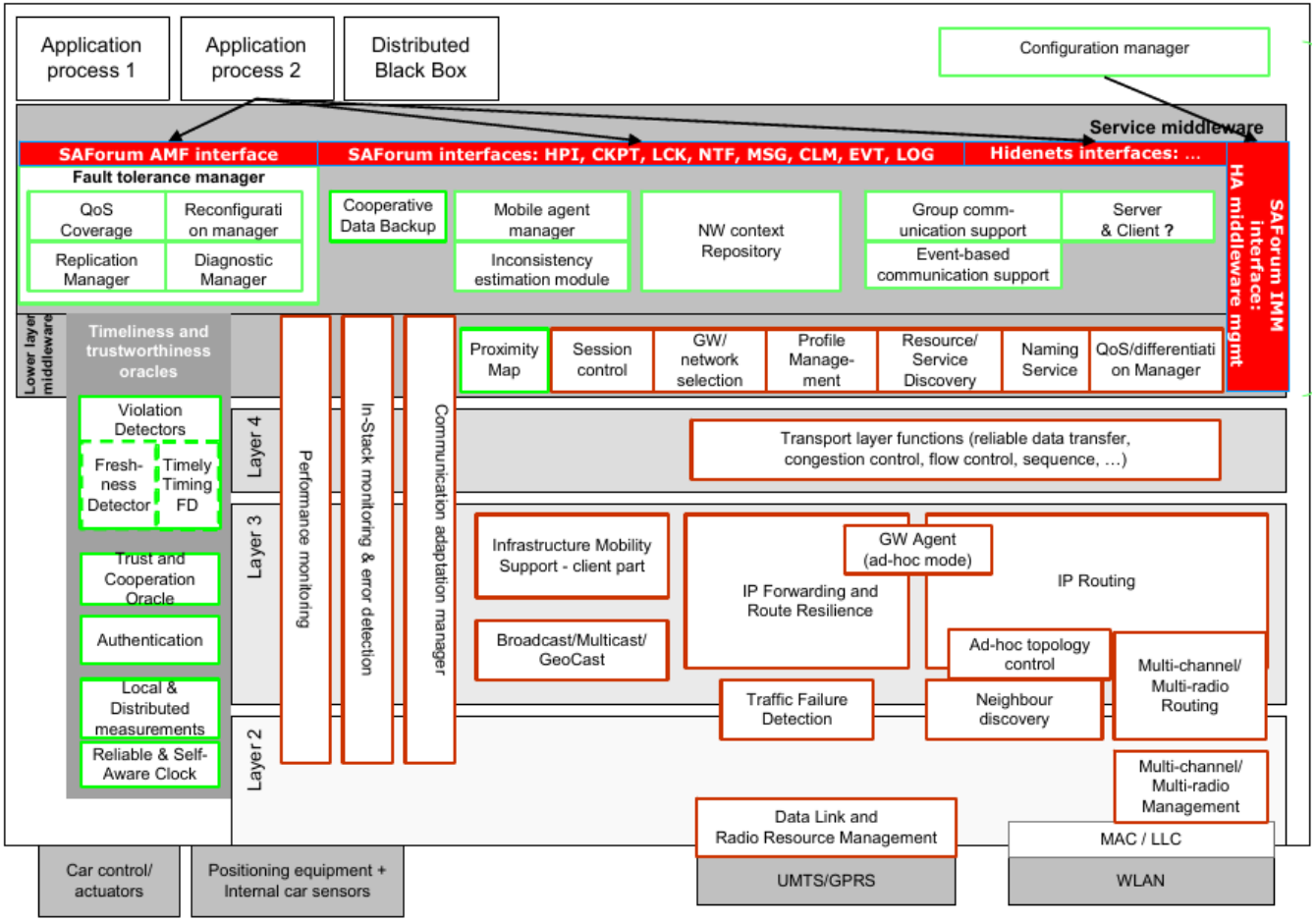
- Projeto HIDENETS (2004-2007)
  - Middleware para comunicação segura e confiável entre veículos



Figuras extraídas de IST-FP6-STREP-26979 / HIDENETS: Resilient Architecture (A. Casimiro, A. Bondavalli, A. Ceccarelli, A. Daidone, L. Falai, P. Frejek, F. Di Giandomenico, G. Huszerl, M.-O. Killijian, A. Kövi, E.V. Matthiesen, O. Mendizabal, H. Moniz, T. Renier, M. Roy)

Middleware – Exemplos de estudos de caso

- Projeto HIDENETS (2004-2007)
  - Middleware para comunicação segura e confiável entre veículos



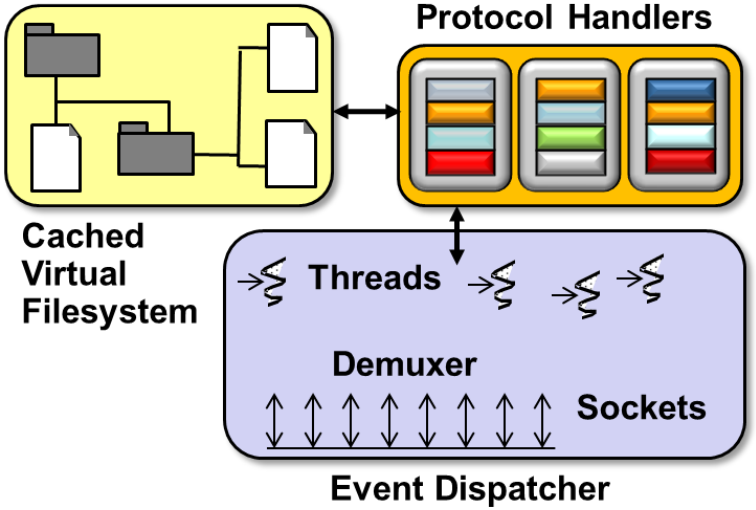
Figuras extraídas de IST-FP6-STREP-26979 / HIDENETS: Resilient Architecture (A. Casimiro, A. Bondavalli, A. Ceccarelli, A. Daidone, L. Falai, P. Frejek, F. Di Giandomenico, G. Huszerl, M.-O. Killijian, A. Kövi, E.V. Matthiesen, O. Mendizabal, H. Moniz, T. Renier, M. Roy)

Middleware – Exemplos de estudos de caso

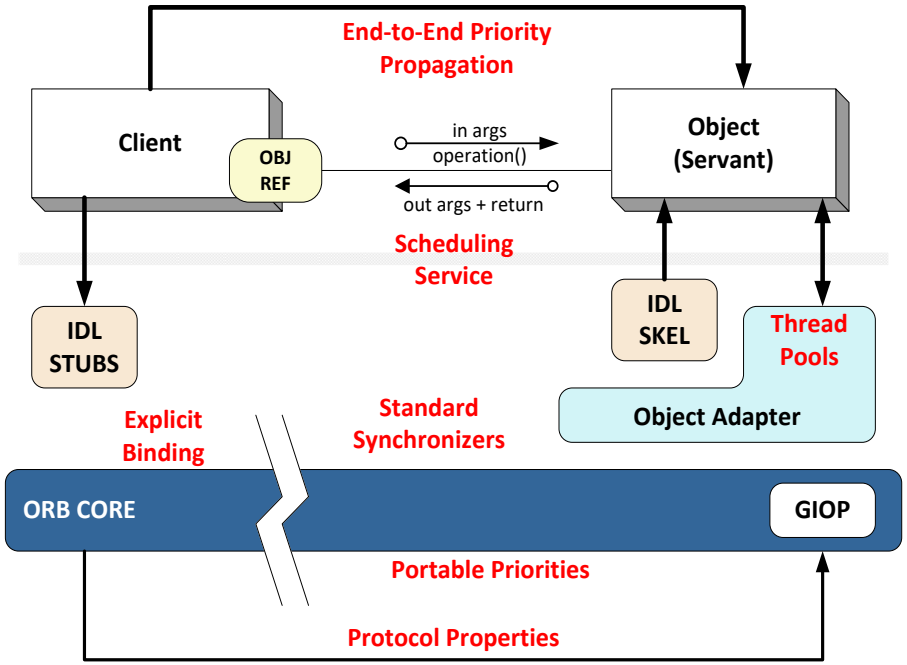
Exemplos extraídos de *slides* do prof. Douglas Schmidt

ACE, TAO, & JAWS (disponíveis com Código fonte aberto)

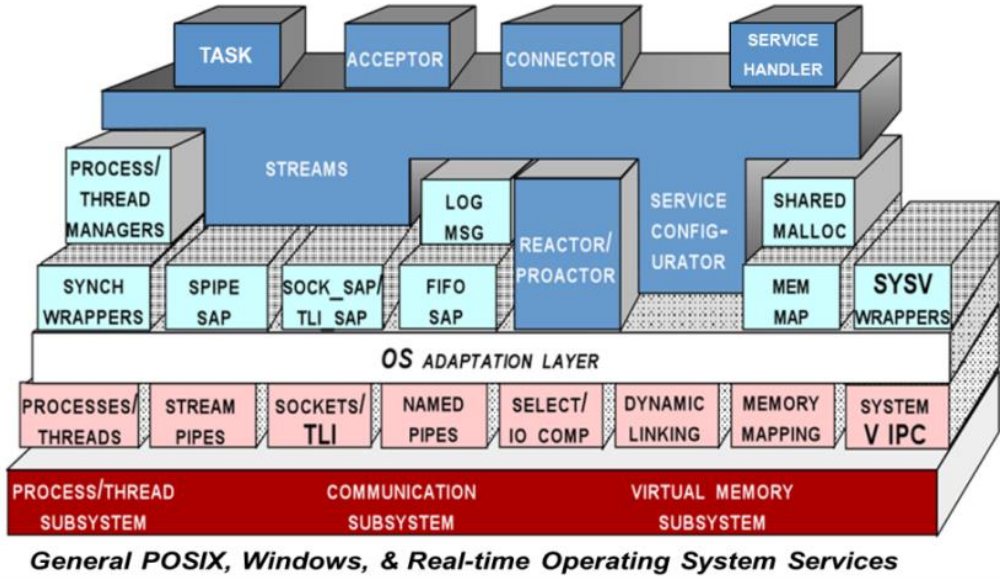
www.dre.vanderbilt.edu/JAWS



www.dre.vanderbilt.edu/TAO



www.dre.vanderbilt.edu/ACE



Referências

Parte destes slides são baseadas em material de aula dos livros:

- *Coulouris, George; Dollimore, Jean; Kindberg, Tim; Blair, Gordon. Sistemas Distribuídos: Conceitos e Projetos. Bookman; 5ª edição. 2013. ISBN: 8582600534*

- *Tanenbaum, Andrew S.; Van Steen, Maarten. Sistemas Distribuídos: Princípios e Paradigmas. 2007. Pearson Universidades; 2ª edição. ISBN: 8576051427*