INTRODUÇÃO

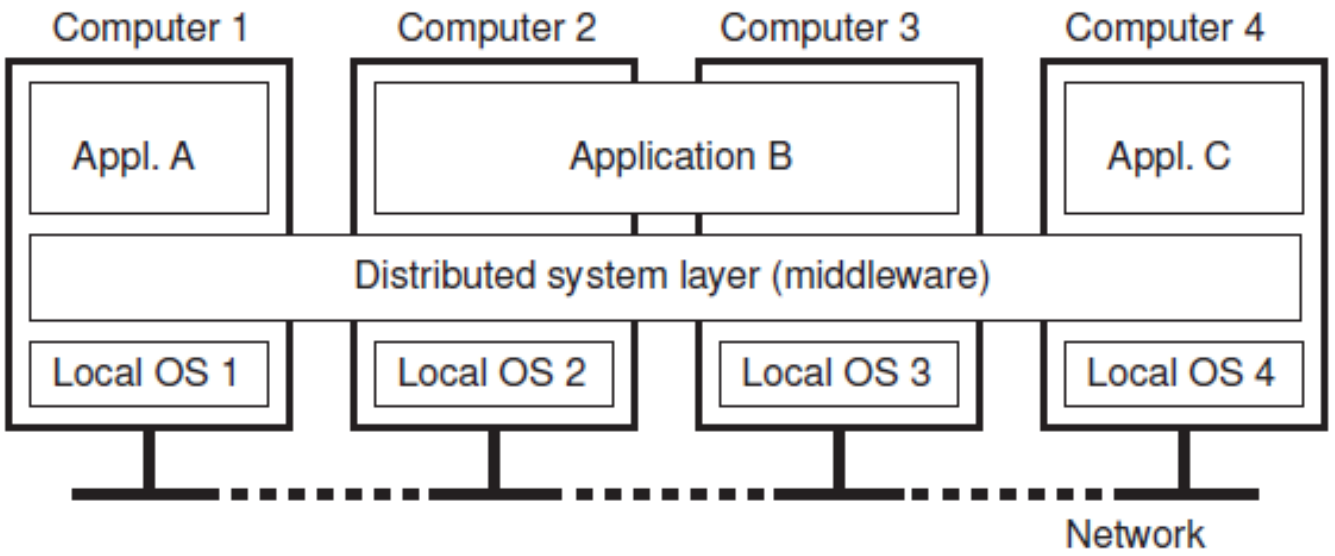
FABRÍCIO LUNA

Definições

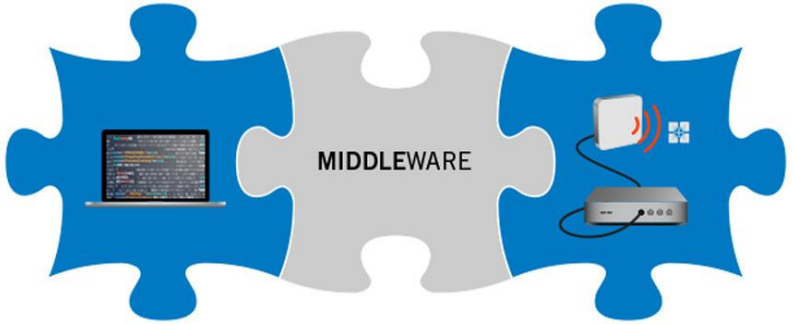
- "Um sistema constituído por um **conjunto de computadores** independentes vistos pelos utilizadores do sistema como sendo um sistema coerente e único." (Tanenbaum)
- "Um sistema no qual **componentes de hardware ou softwares** localizados em computadores em rede **comunicam e coordenam** as suas ações através da troca de mensagens" (Colouris)
- "Um sistema no qual a falha de um computador que nem sequer sabíamos existir, pode tornar o nosso computador não usável" (Lamport)

Definição (2)

- Computadores (processos) independentes
- Sistema único => **middleware**



Middleware

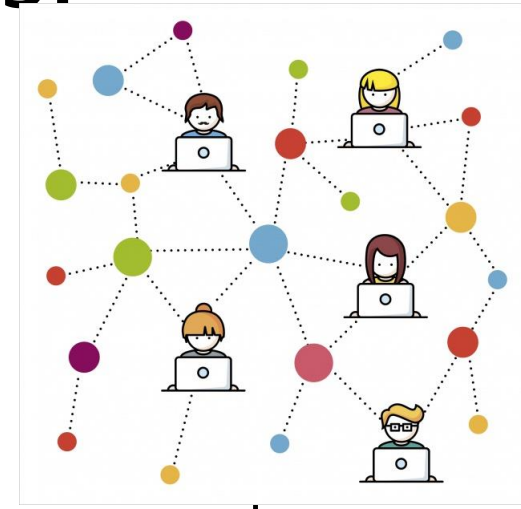


- O middleware é o software que se encontra **entre** o sistema operacional e os aplicativos nele executados.
- Funcionando de forma essencial como uma **camada oculta** de tradução, o middleware permite a **comunicação e o gerenciamento** de dados para aplicativos distribuídos
- Exemplos comuns de middleware incluem middleware de banco de dados, middleware de servidor de aplicativos, middleware orientado a mensagens, middleware de web e monitores de processamento de transações.

Definição (3)

- Podemos então caracterizar um Sistema Distribuído pelas seguintes propriedades:
  - Constituído por **múltiplos** computadores (processos)
  - **Ligados** por uma rede
    - não partilham memória
    - comunicam-se apenas por mensagens
  - Coordenam ações e cooperam entre si

Sistemas distribuídos não são redes!



- As redes preocupam-se com
  - O **envio** de mensagens de um ponto A para um ponto B
  - **Não** se preocupa com o que se faz com a mensagens
- A computação distribuída
  - Assume que existe alguma forma de enviar mensagem (o transporte de mensagem é assegurado pela rede)
  - Preocupa-se com as propriedades dessas mensagens
  - E como construir um sistema com o uso de mensagens

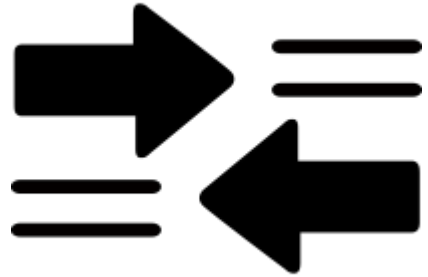
Dificuldades

- **Partilha**
  - Dados
  - Processamento
  - Consistência
- **Descoberta**
  - Como localizar recursos
  - Uma vez encontrados, como usá-los?
- **Modelos de programação**
  - Complexidade e dimensão dos sistemas
  - Não-determinismo no seu funcionamento
  - Torna os modelos de programação necessariamente complexos.

Falhas operacionais em sistemas

- **Sistema não distribuído**
  - Quando falha, esta é total
  - Quando ocorre uma falha, sabemos que ocorreu
  - Uma estratégia de recuperação: reiniciar
- **Sistema distribuído**
  - A falha pode ser parcial (apenas em alguns elementos)
  - A falha pode não ser conhecida
  - Requer uma estratégia para lidar com falhas

Exemplos de Sistemas Distribuídos



- A Internet e a Web
- Aplicações P2P: e.g. Napster, Gnutella, Freenet, Kazaa, Bittorrent, Skype (?)
- Computação voluntária: SETI@home, Folding@home
- Sistema SIBS (Gestão de caixas bancárias automáticas)
- Sistemas de gestão de inventários em cadeias de supermercados
- Sistemas de gestão de saúde
- …
- Arquiteturas: cliente-servidor e peer-to-peer

Calma, respira....

- O que é um Sistema Distribuído?
- O que é um Middleware
- Uma conexão de rede é um Sistema Distribuído?
- Quais as dificuldades dos Sistemas Distribuídos?
- Falhas operacionais?
- Exemplos?

- To be continued...

Razões a favor de sistemas distribuídos

- **Funcionalidade e capacidade distribuídas**
  - Clientes/servidores
  - Recolha de informação / processamento
- **Domínio da aplicação intrinsecamente distribuída**
  - Caixa de registro e sistema de inventário de uma cadeia de supermercado
  - Sistema de gestão de dados administrativos numa rede hospitalar

Razões a favor de sistemas distribuídos(2)

- **Desempenho: distribuição de carga, dados e processamento**
  - Permite distribuir tarefas de modo a otimizar o desempenho geral
  - Mais processadores, maior capacidade de processamento
  - Vantagem econômica: custo/performance
- **Expansibilidade (escalabilidade)**
  - Utilizadores (e processos), dispersão física e administração
- **Disponibilidade e tolerância a falhas**

Objetivos

- **Acessibilidade** – tornar acessíveis recursos eventualmente dispersos fisicamente
- **Transparência** (da distribuição)
  - Capacidade de esconder dos utilizadores a distribuição física de recursos
  - A visão deverá ser de um sistema único e consistente
- **Aberto**
  - Capacidade do sistema ser implementado de diferentes formas
- **Expansível**

Transparência da distribuição

Transparência	Descrição
Acesso	**Esconder diferenças** na representação dos dados e como acessar recursos
Localização	**Esconder** a **localização** dos recursos
Migração	Poder **mudar** um **recurso** sem afetar o modo como é acessado
Recolocação	Poder **mudar** a **localização** de recursos durante o seu uso
Replicação	**Esconder** a **existência** de múltiplas réplicas de um recurso (associado à transparência de localização)
Concorrência	**Esconder** a **coordenação** necessária para acessar a recursos partilhados e assegurar consistência
Falhas	**Esconder falhas** e possíveis recuperações de recurso

Transparência é um objetivo de SDs, mas é difícil alcançá-lo

Grau de transparência da distribuição

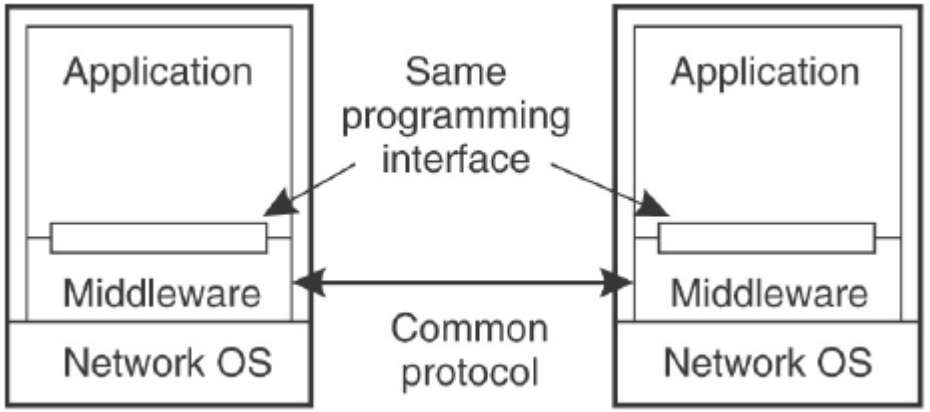
- Observação: pretender transparência completa pode ser pedir demais;
- Nem sempre há interesse em esconder a **localização** dos recursos
  - Ex: Partilhar uma impressora
- Por vezes é impossível esconder completamente falhas no sistema:
  - Como distinguir um computador muito lento de uma falha,
  - Como determinar se um servidor efetuou de fato uma operação antes de um crash?
- Pode ter custos elevados no desempenho do sistema
  - Manter um cache Web atualizada com a cópia original
  - Transferir para disco as operações de escrita para assegurar tolerância a falhas

Abertura de um sistema distribuído

- Deve ser capaz de interagir com outros sistemas abertos independentemente do ambiente. Para isso precisam de:
  - Ter **interfaces** bem definidas e públicas
  - Suportar **portabilidade** das aplicações
  - Serem facilmente **interoperáveis** (transparente)
- Deve ser independente da **heterogeneidade** do seu ambiente de execução:
  - Plataformas (Hardware+Software)
  - Linguagens de programação

Middleware e abertura

- Num SD baseado num **middleware aberto**, os protocolos em cada camada do middleware têm de ser os mesmos, assim como as interfaces disponibilizadas às aplicações.



Escala em sistemas distribuídos

- A expansividade de um sistema distribuído
  - Envolve pelo menos 3 componentes
    - **Dimensão**: número de utilizadores e/ou processos
    - **Dispersão** geográfica: distância máxima entre nós (nós de nó, não eu e você)
    - **Administração**: número de domínios administrativos
  - A grande maioria dos sistemas apenas olha à expansibilidade relacionada com a dimensão;

Técnicas para assegurar escalabilidade

- **Minorar latência** de comunicação: enquanto espera por uma resposta, faz outra coisa:
  - Usar o máximo possível de comunicação assíncrona
  - Ter um *handler(tratador)* separado para respostas em espera
  - Dificuldade: nem todas as aplicações se encaixam nesse modelo.
- **Distribuição:** dividir dados e computação por múltiplas máquinas:
  - Deslocar computação para clientes (Java applets)

Técnicas para assegurar escalabilidade(2)

- **Replicação**: Tornar disponível em diferentes máquinas cópias de dados:
  - Replicação em servidores de ficheiros e bases de dados
  - Cópias de Web sites (mirrors)
  - Web caches (em browsers e proxies)
  - Caching de ficheiros (no servidor e no cliente)

Escalabilidade: dificuldades

- A **replicação e caching**
  - Reduz latência da comunicação
  - Distribui processamento
- Mas pode originar **problemas de consistência**
  - Modificar uma cópia a torna diferente das demais
- Manter a informação consistente requer **sincronização global** em cada modificação
  - Assegurar sincronização global impede problemas em grande escala
- **Observações**
  - Se pudermos tolerar alguma inconsistência, reduzimos a necessidade de uma sincronização global
  - A tolerância de cópias não consistentes depende do tipo de aplicação

Falácias no desenvolvimento de sistemas distribuídos

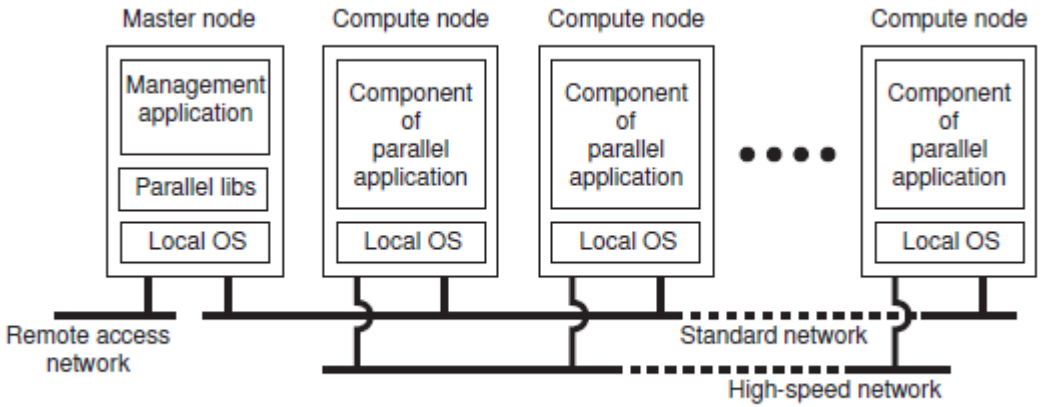
- Premissas que nem sempre se verificam:
  - A rede é confiável
  - A rede é segura
  - A rede é homogénea
  - A topologia não muda
  - A latência é zero
  - A largura de banda é infinita
  - O transporte de mensagens tem custo zero
  - Não existe administrador

Tipos de sistemas distribuídos

- Sistemas distribuídos de computação
- Sistemas distribuídos de informação
- Sistemas distribuídos ubíquos

Sistemas distribuídos de computação

- Configurados para computação de elevado desempenho (HPC):
- **Computação em** *clusters*
  - Nós homogêneos (mesmo OS e hardware quase idênticos)
  - Um nó de administração
- **Computação em** *Grid*
  - Nós heterogêneos
  - Dispersos através de várias organizações e WANs



Sistemas distribuídos de informação

- Inúmeros SDs em uso são sistemas de informação tradicionais que integram sistemas antigos. Ex: Sistemas transacionais

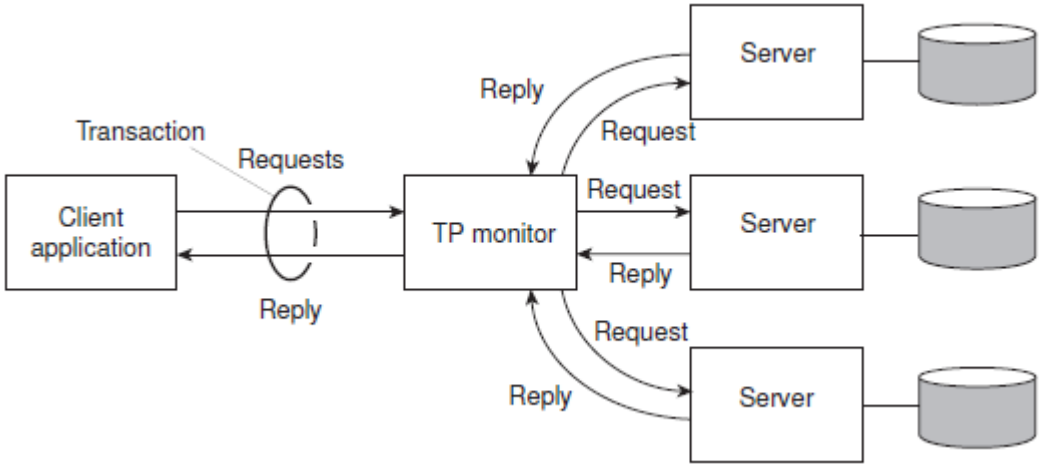
```
BEGIN_TRANSACTION(server, transaction);
READ(transaction, file-1, data);
WRITE(transaction, file-2, data);
newData := MODIFIED(data);
IF WRONG(newData) THEN
        ABORT_TRANSACTION(transaction);
ELSE
        WRITE(transaction, file-2, newData);
        END_TRANSACTION(transaction);
END IF;
```

Sistemas distribuídos de informação(2)

- Essencial: todas as operações READ e WRITE são executadas e o seu efeito é tornado permanente com a execução de END_TRANSACTION
- Uma transação constitui em operação **atômica.**

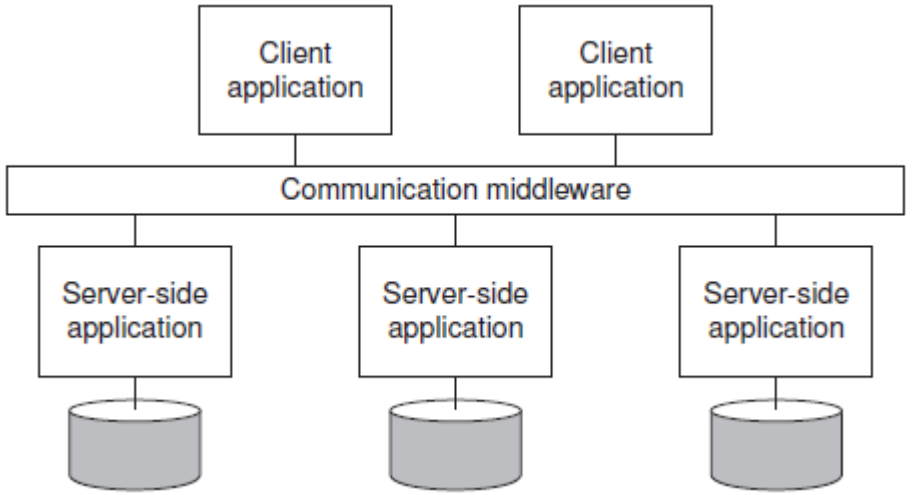
Sistemas distribuídos de informação: transações

- Transação é um conjunto de operações sobre o estado de um objeto (Ex: Base de dados) que satisfaz as propriedades (ACID):
  - **Atomicidade**: as operações ou sucedem todas, ou falham todas. A falha de uma transação falha, não afeta o estado do objeto.
  - **Consistência**: estabelece sempre transições de estado válidas
  - **Isolamento**: as transações não interferem umas com as outras.
  - **Durabilidade**: depois da sua execução, o seu efeito é permanente.



Integração de aplicações empresariais

- Usar um monitor para coordenar a execução de uma transação faz sentido!
- Mas, em muitos casos, precisamos separar a aplicação da base de dados sobre a qual intervém.
- Solução: usar mecanismos de comunicação entre aplicações:
  - Invocação remota de procedimentos (RCP)
  - Middlware orientado por mensagens (MOM)

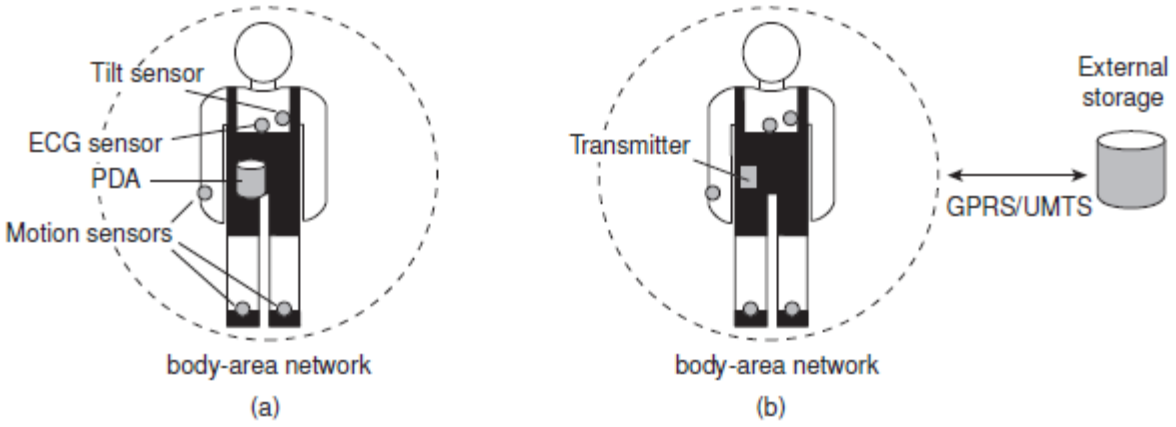


Sistemas Distribuídos Ubíquos

- A próxima geração de sistemas distribuídos terá
  - Nós pequenos e móveis
  - Integrados num sistema maior
- Alguns requisitos:
  - **Mudança de contexto**: as alterações devem ser imediatamente reconhecidas
  - **Composição ad-hoc:** cada nó poderá ser usado de diferentes formas por diferentes utilizadores
  - **Partilha é o default:** os nós ligam-se e desligam-se, fornecendo partilha de serviços. Requer simplicidade.
  - Observação: ubiquidade e transparência sempre casam bem.

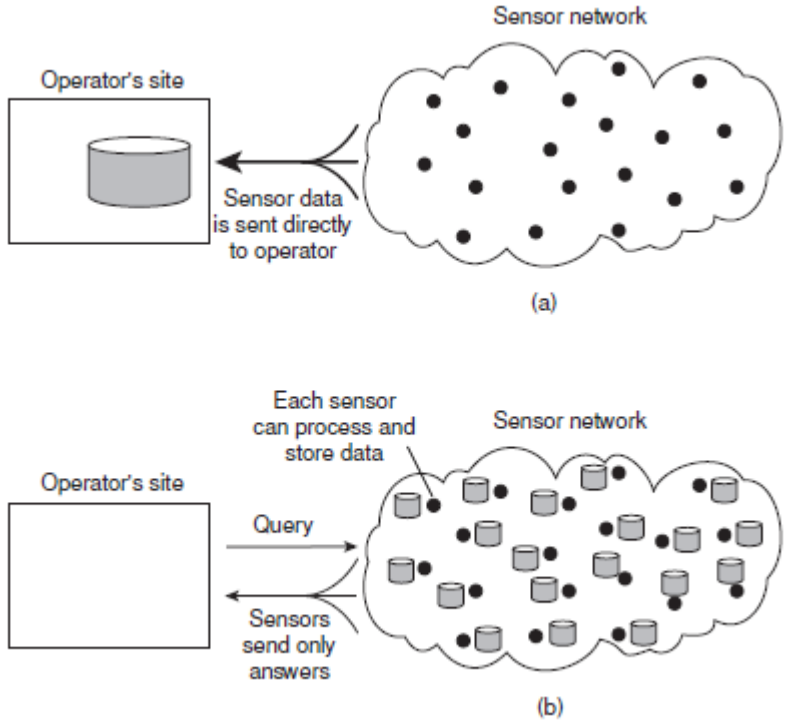
SD ubíquos: exemplos

- Sistemas eletrônicos de saúde: periféricos próximos da pessoa
  - Onde e como guardar os dados que estão sendo vigiados?
  - Como evitar perder dados cruciais?
  - O que é preciso para propagar alertas?
  - Como assegurar a segurança?
  - Como podem os médicos dar parecer online?



SD ubíquos: exemplos

- **Rede de sensores:** os nós aos quais os sensores estão ligados são:
  - Muitos (10-1000)
  - Simples (infraestrutura quase sem memória, CPU, comunicação)
  - Operados a bateria



Fim =)