

# SISTEMAS DISTRIBUÍDOS

## Capítulo 1

### Introdução

# NOTA PRÉVIA

A apresentação utiliza algumas das figuras do livro de base do curso

- G. Coulouris, J. Dollimore and T. Kindberg,
- Distributed Systems - Concepts and Design,
- Addison-Wesley, 5th Edition, 2011

# ORGANIZAÇÃO DO CAPÍTULO

Definição, exemplos

Características essenciais dos sistemas distribuídos

Desafios: heterogeneidade, abertura, segurança, escala, falhas, concorrência, transparência

# SISTEMAS DISTRIBUÍDOS ?

## Exemplos?

- Serviços web – redes sociais, e-commerce, jogo multi-utilizador, etc.
- Email
- Multibanco
- Etc.

## O que é importante?

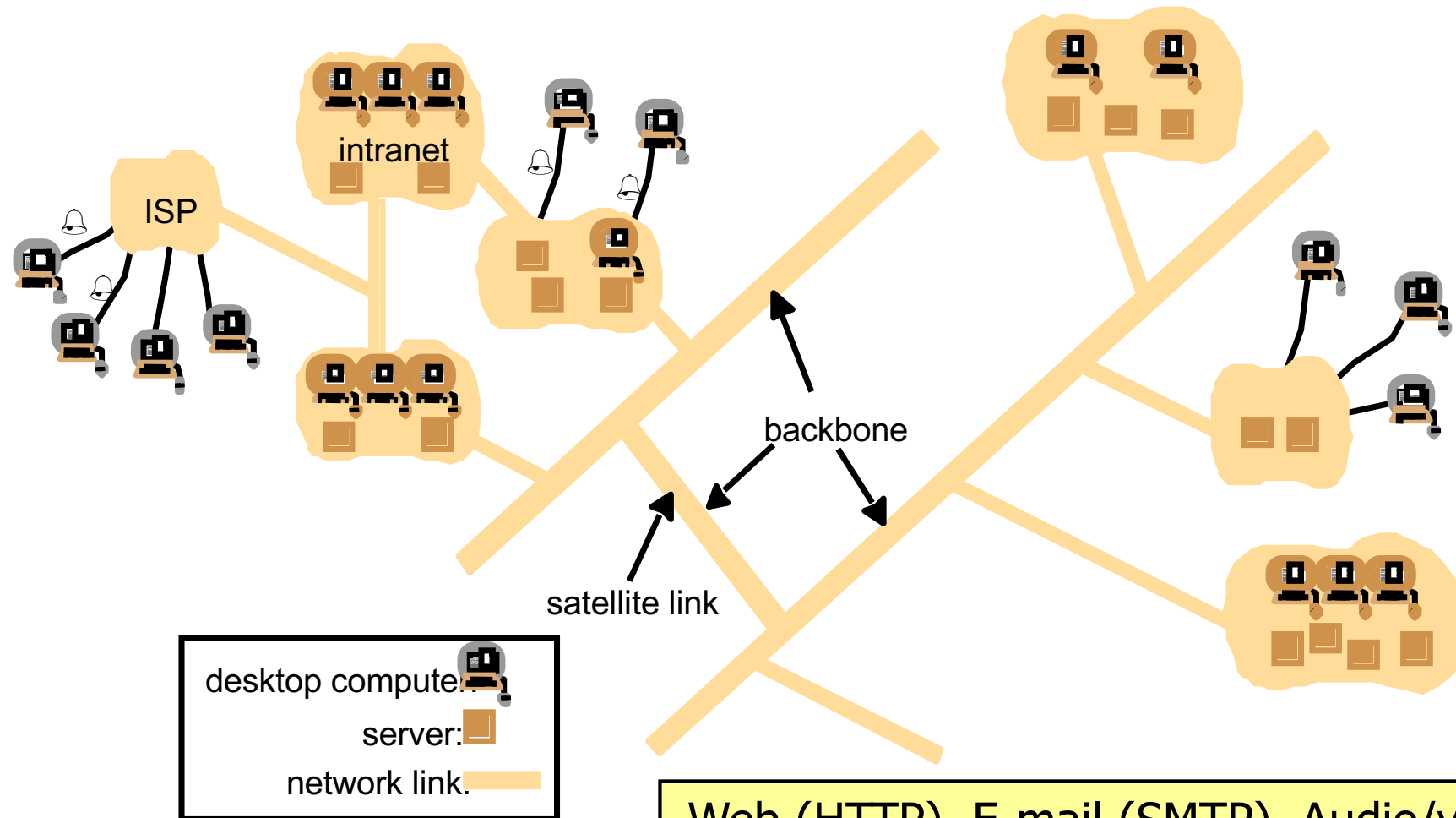
- Conjunto de nós / máquinas
- Utilização duma rede de comunicações para troca de mensagens

# O QUE É UM SISTEMA DISTRIBUÍDO ?

Um sistema distribuído é um conjunto de componentes hardware e software interligados através de uma infraestrutura de comunicações, que **cooperam e se coordenam entre si** apenas pela troca de mensagens, para execução de **aplicações distribuídas**

Assim, no âmbito desta cadeira, não estamos interessados nos sistemas que cooperam e se coordenam pela partilha de memória física comum ou de programas que executam em GPUs (essa temática é abordada inicialmente na disciplina de Concorrência e Paralelismo)

# EXEMPLO: SERVIÇOS NA INTERNET

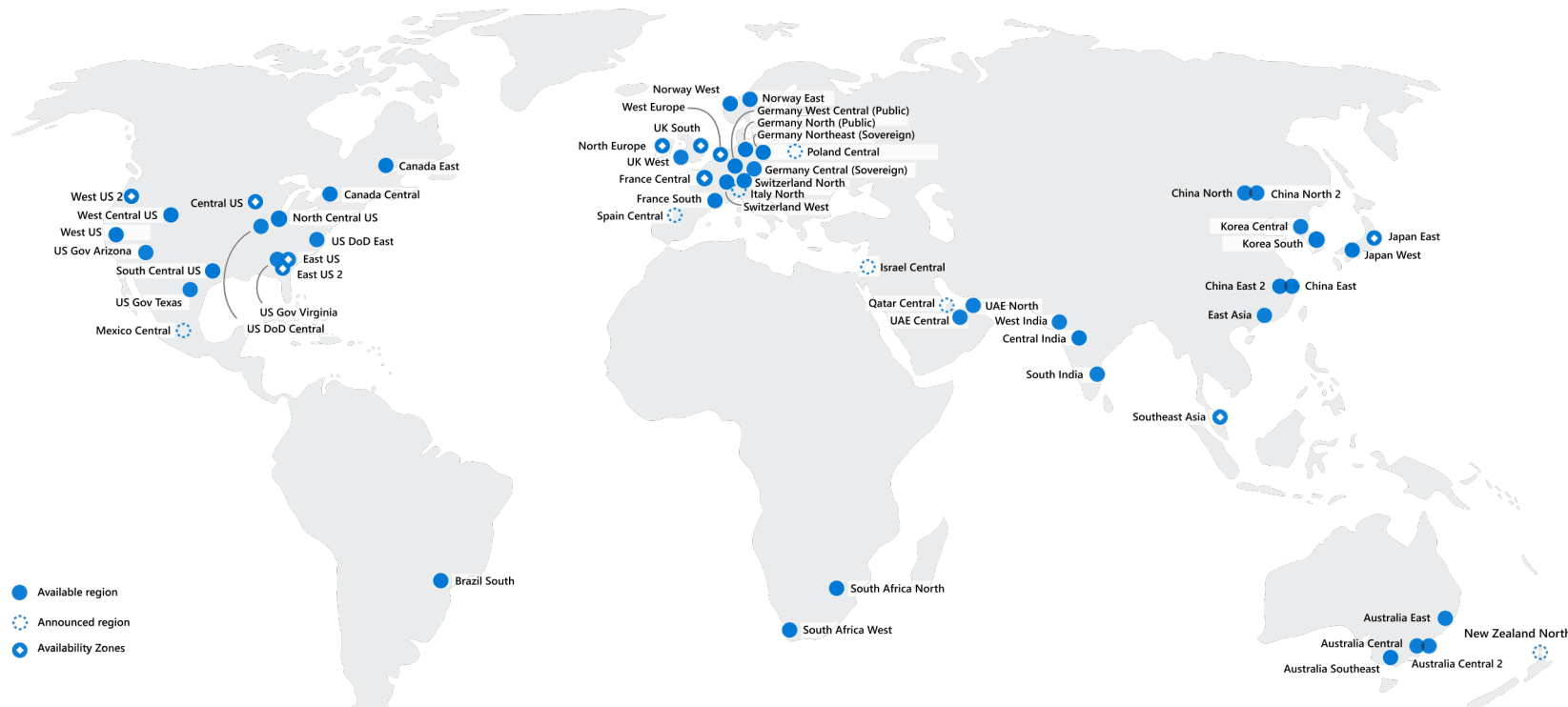


Web (HTTP), E-mail (SMTP), Audio/video-conf (e.g. Zoom, Skype), streaming (e.g. Netflix), multi-player games (e.g. Pokemon Go) etc.

# EXEMPLO: CLOUD COMPUTING

Diferentes tipos de serviço disponibilizados através duma infraestrutura distribuída globalmente.

- IaaS (Infrastructure as a Service) : e.g. AWS EC2 (máquinas virtuais)
- PaaS (Platform as a Service) : e.g. Google App Engine (app service)
- SaaS (Software as a Service) : e.g. Google Docs

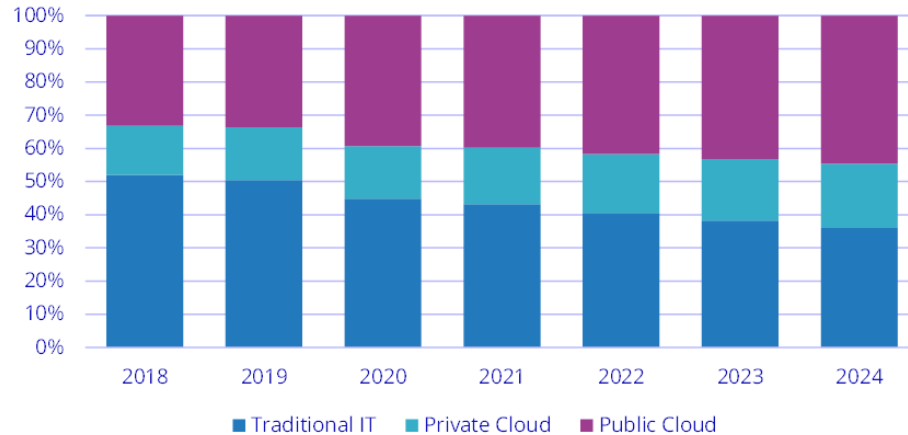


Azure locations: <https://azure.microsoft.com/en-us/global-infrastructure/geographies/> (accessed: Sep, 2020)

# CLOUD COMPUTING: ALGUMAS TENDÊNCIAS

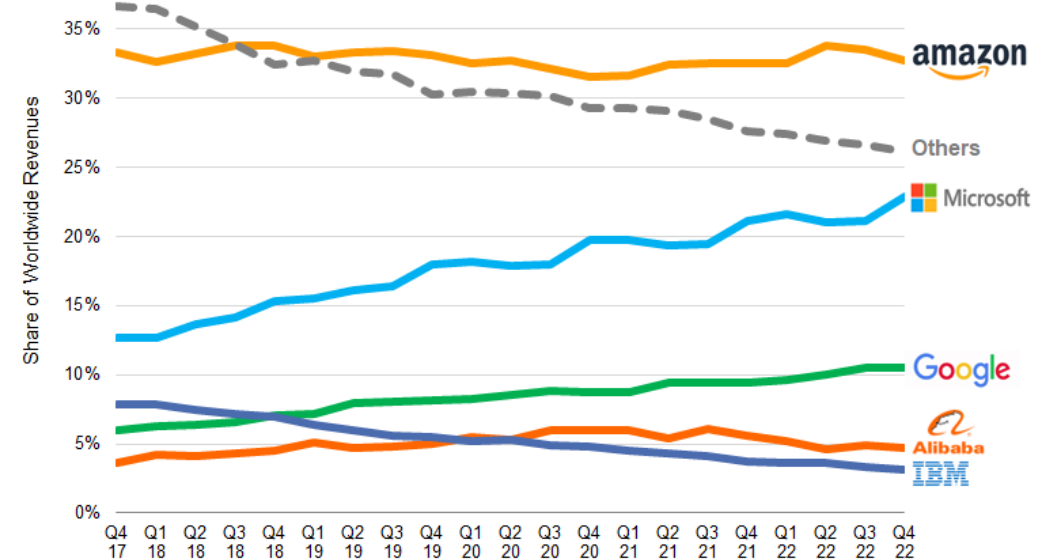


Worldwide Cloud IT Infrastructure Market Forecast by Deployment Type, 2018- 2024 (shares based on Value)

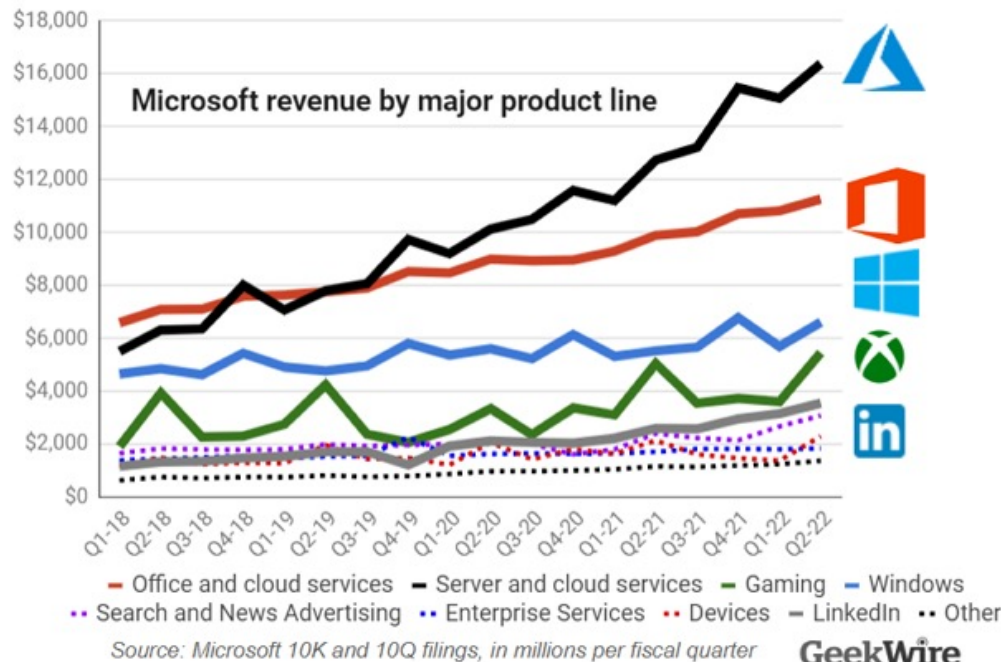


Source: IDC 2021

Cloud Provider Market Share Trend  
(IaaS, PaaS, Hosted Private Cloud)



Source: Synergy Research Group

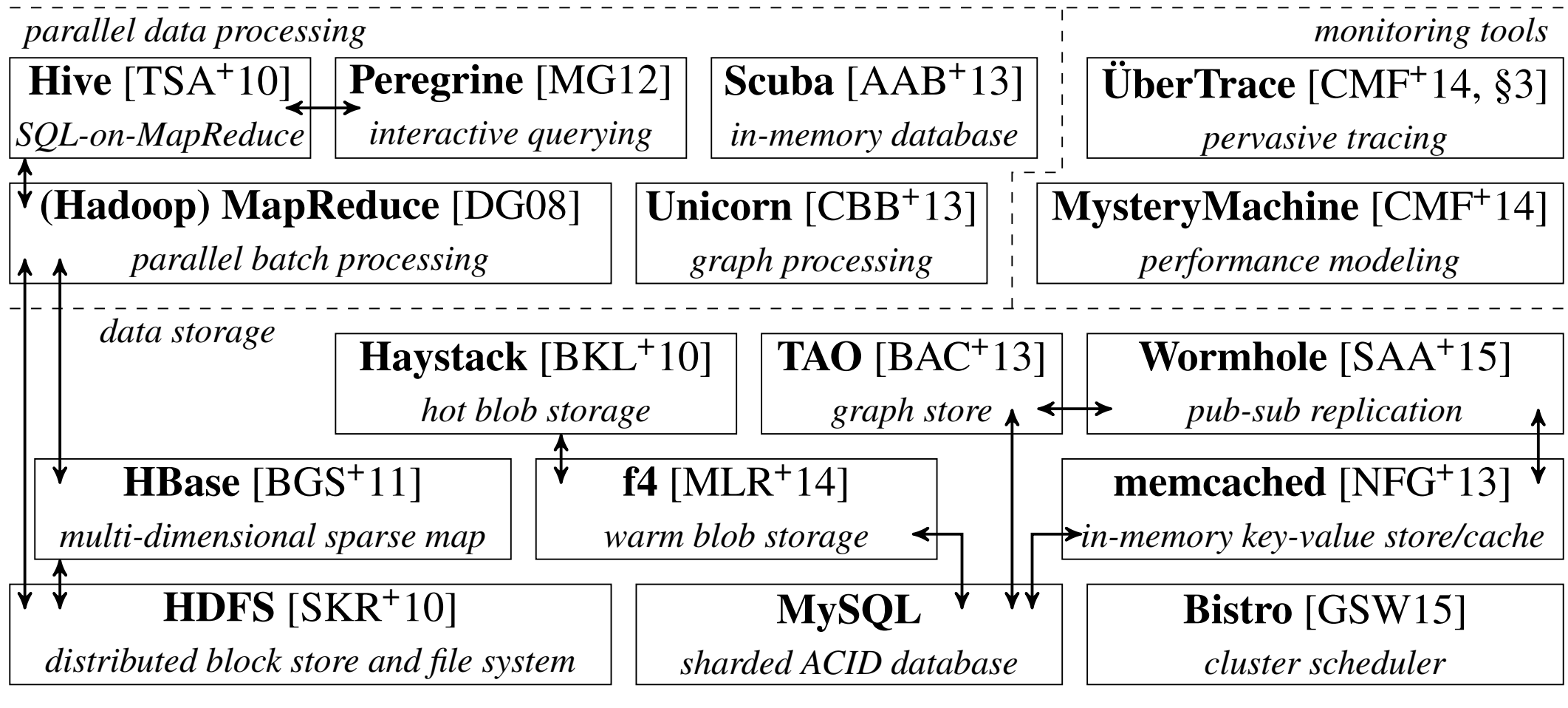


Source: Microsoft 10K and 10Q filings, in millions per fiscal quarter

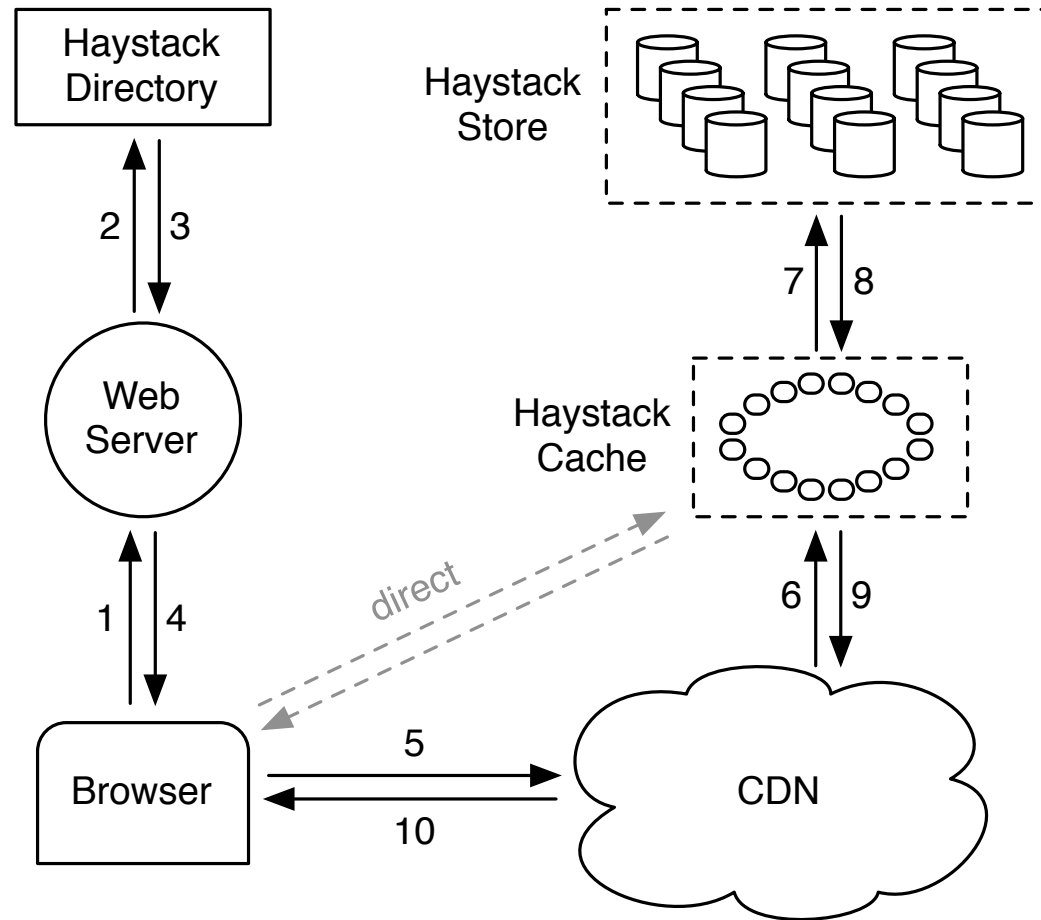
GeekWire



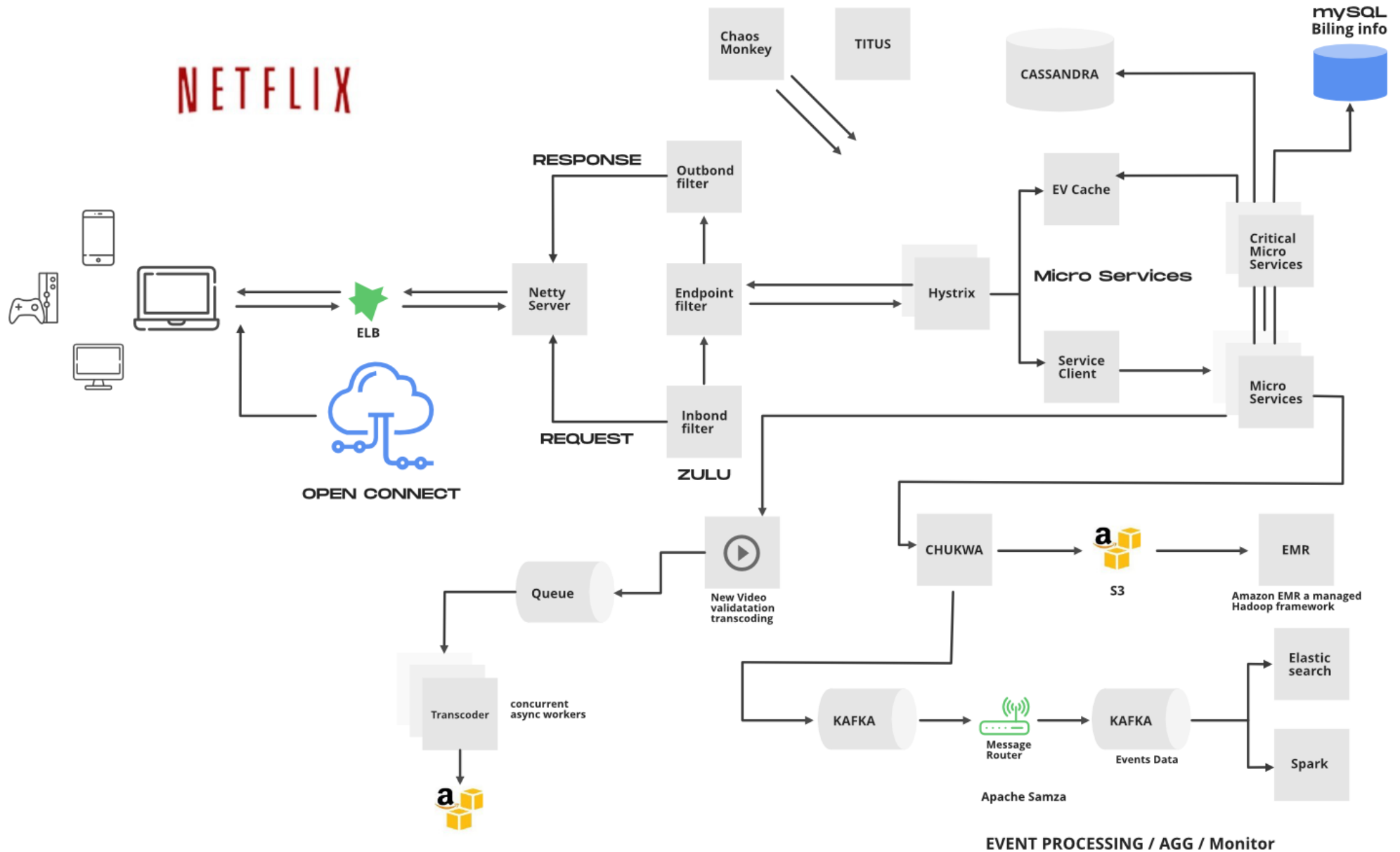
# EXEMPLO: FACEBOOK / META



# EXEMPLO: FACEBOOK HAYSTACK

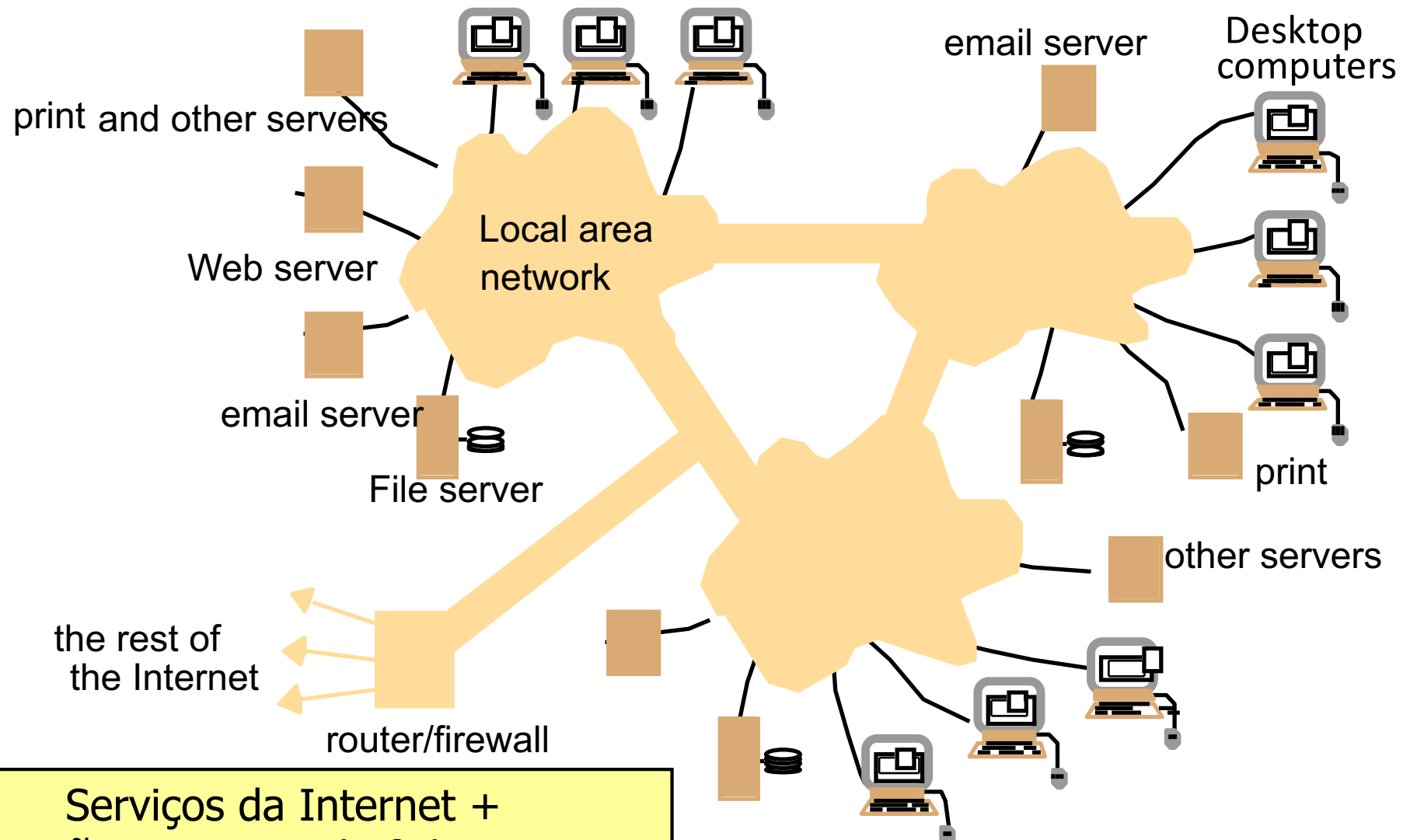


# NETFLIX



# EXEMPLO: SERVIÇOS EM INTRANETS

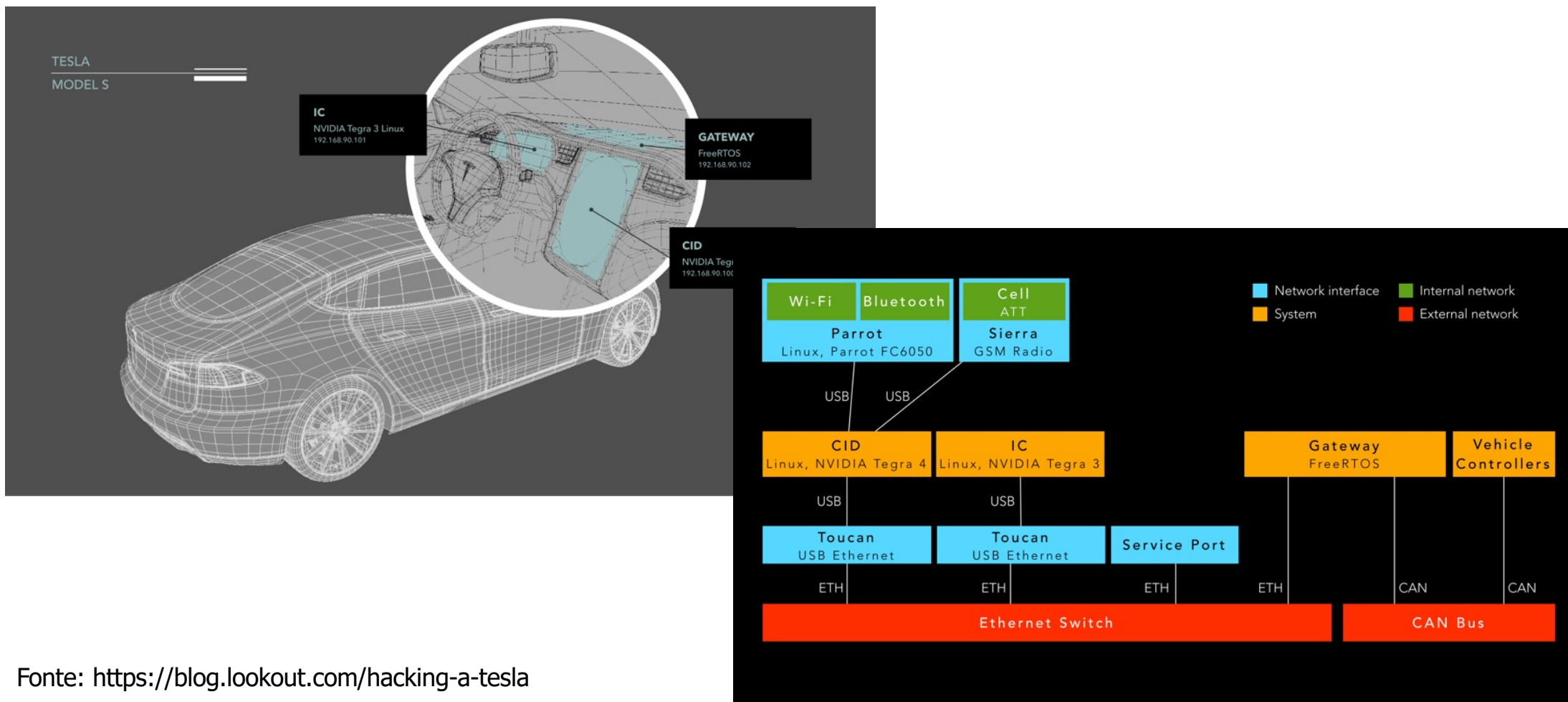
Intranets: redes isoladas fisicamente, redes isoladas logicamente (private virtual network), ligação à Internet através de firewalls, etc.



Serviços da Internet +  
impressão; sistemas de ficheiros; etc.

## OUTROS EXEMPLOS: CARROS

Dispositivos ou máquinas especiais controlados através de conjuntos de **computadores embebidos** (por exemplo: um avião ou um carro, uma fábrica)



# OUTROS EXEMPLOS

**Sistemas de controlo** de processos industriais em fábricas (por exemplo, linhas de montagem)

**Clusters de computadores** interligados através de redes de alta velocidade para cálculo paralelo

# AULA 2

# ... NA AULA 1

## O que é um sistema distribuído

- Um sistema distribuído é um conjunto de componentes hardware e software interligados através de uma infra-estrutura de comunicações, que **cooperam e se coordenam entre si** apenas pela troca de mensagens, para execução de **aplicações distribuídas**

## Exemplos de sistemas distribuídos

- Serviços na Internet, cloud computing
- Sistemas de controlo (e.g. fábricas)
- Sistemas embebidos e de tempo-real (e.g. carros, aviões)



# NESTA AULA

Características essenciais dos sistemas distribuídos

Desafios na conceção de sistemas distribuídos

# MOTIVAÇÕES DOS SISTEMAS DISTRIBUÍDOS

Acesso generalizado sem restrições de localização

- Acessibilidade ubíqua (suporte para utilizadores fixos, móveis)

Partilha dos recursos distribuídos pelos diferentes utilizadores

- Exemplos: impressores, ficheiros

Distribuição da carga – melhoria do desempenho

Tolerância a falhas – melhoria da disponibilidade

Flexibilidade e adaptabilidade

- Decomposição de um sistema complexo num conjunto de sistemas mais simples

# CARACTERÍSTICAS FUNDAMENTAIS

Componentes do sistema **executam de forma concorrente** – paralelismo real

- Necessidade de coordenação entre os vários componentes

**Falhas independentes** das componentes e das comunicações

- Impossível determinar se existe uma falha dum componente ou do sistema de comunicações
- Necessidade de tratar as falhas

**Ausência de relógio global** – existem limites para a precisão da sincronização dos relógios locais

- Impossível usar relógios locais para ordenar globalmente todos os eventos

# IMPLICAÇÕES

Nenhum componente tem uma visão exacta instantânea do estado global de todo o sistema

Os componentes têm uma **visão parcial do estado global** do sistema

- Os componentes do sistemas estão distribuídos e só podem cooperar através da troca de mensagens, as quais levam um tempo não nulo a propagarem-se

Na presença de falhas, o estado global pode tornar-se incoerente, i.e., as visões parciais do estado global podem tornar-se incoerentes

- Por exemplo, réplicas de um objecto podem ficar incoerentes

# DESAFIOS

Heterogeneidade

Abertura

Transparência

Segurança

Escala

Tratamento das falhas

# HETEROGENEIDADE

Hardware: smartphones, tablets, portáteis, servidores, clusters, ...

- Diferentes características dos processadores, da memória, da representação dos dados, dos códigos de caracteres,...

Redes de interligação e protocolos de transporte: Redes móveis (5G, 4G, 3G, GSM), WLANs, wired LANs,..., TCP/IP, ....

Sistema de operação: Windows, MacOS, iOS, Android,...

- Diferentes interfaces para as mesmas funcionalidades

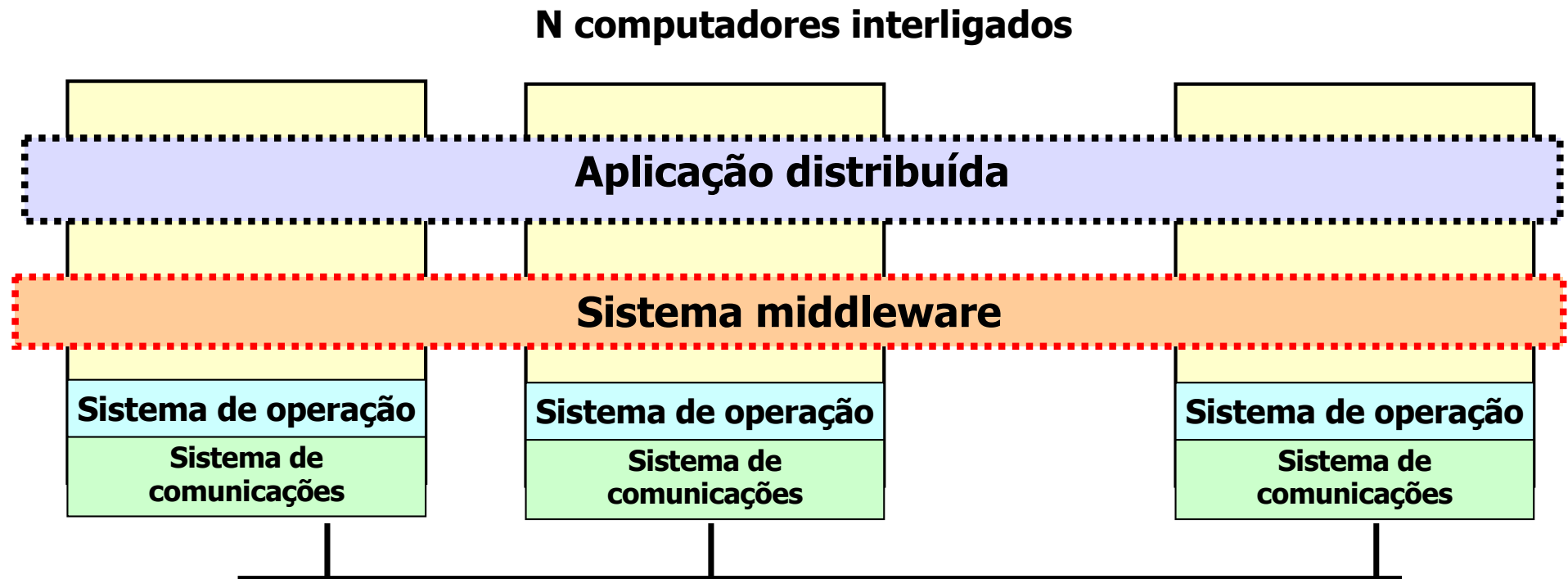
Linguagens de programação...

Lidar com esta heterogeneidade é muito complexo

As seguintes soluções podem ajudar:

- Sistemas de Middleware
- Máquinas Virtuais
- Containers

# MIDDLEWARE



## Sistemas operativos

Interfaces heterogénea

Serviços básicos

- Sistema *middleware*
  - Interface homogénea
  - Serviços mais complexos (invocação remota: Web-services; message-queue: \*MQ, etc)
  - Verdadeira interoperabilidade requer idênticos interface e protocolos

# MÁQUINA VIRTUAL

Objetivo: permitir executar os mesmos programas em máquinas com diferentes características

Máquina virtual aplicação: virtualiza ambiente de execução independentemente do sistema de operação

- E.g.: programas escritos numa única linguagem (JavaVM) ou em múltiplas linguagens (Microsoft CLR)

Máquina virtual sistema: virtualiza máquina física

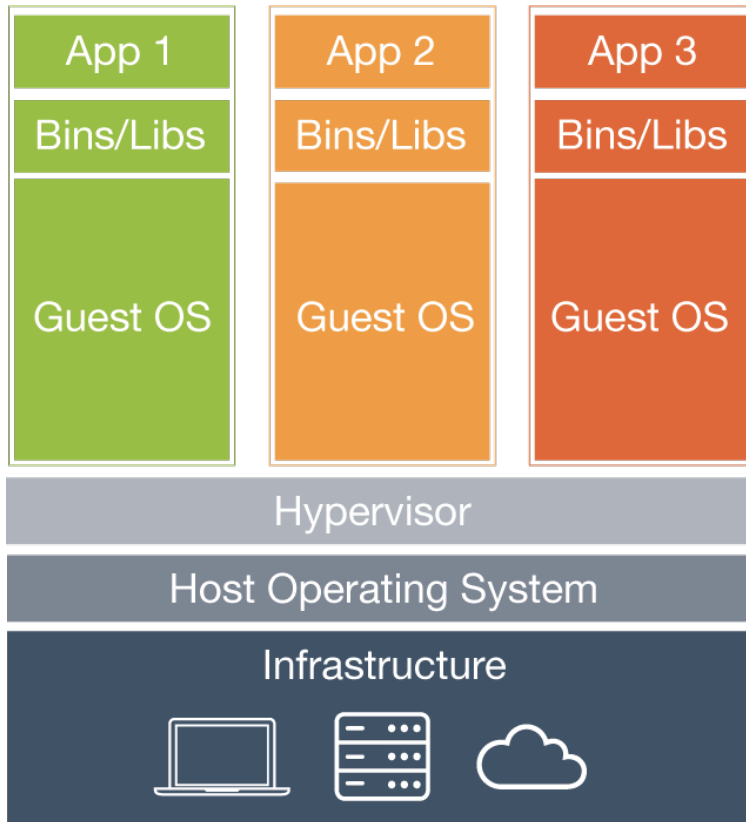
- E.g.: VmWare, VirtualBox, etc.



# CONTAINER

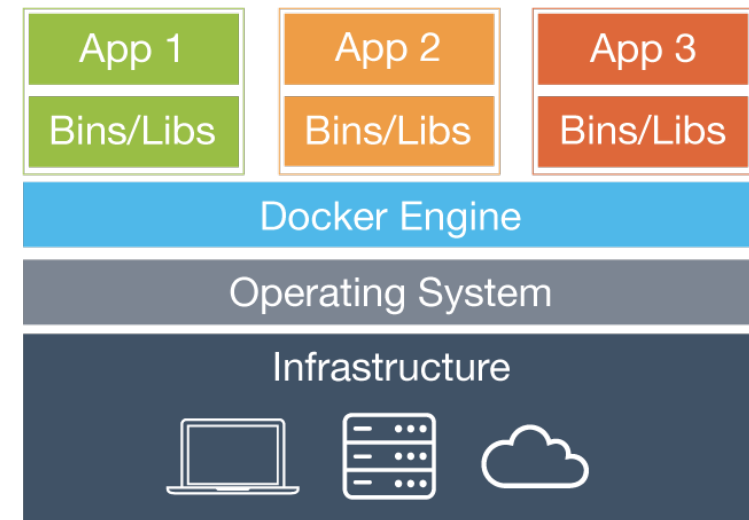
Objetivo: permitir executar aplicações, incluindo todas as suas dependências

## Máquina Virtual



Images from [docker.com](https://www.docker.com)

## Container



# CONTAINER (CONT.)

A container uses the underlying OS of the system (or VM) where it is running...

- The OS is shared among all containers running

.. But containers are isolated from each other with assigned exclusive resources, including CPU, memory, network, disk, etc.

This makes a container act as an independent machine, with its own IP.

# ABERTURA

A abertura de um sistema determina o modo como pode ser estendido e re-implementado

## Sistemas abertos

- Interfaces e modelo (incluindo protocolos de comunicação) conhecidos
- Evolução controlada por organismos de normalização independentes ou consórcios industriais
- Permite a interoperação de componentes com diferentes implementações

## Sistemas proprietários

- Podem ser modificados pelo seu “dono”

Vantagens e desvantagens de cada aproximação?

# ABERTURA E OPEN-SOURCE

Uma aplicação de código aberto (open source) é uma aplicação da qual o código fonte está disponível.

A utilização da aplicação é regida por uma licença que é tipicamente distribuída com o código. Existem vários tipos de licença:

***Public domain*** – todos os direitos são transferidos.

**Permissive licence** (e.g. MIT, Apache) – oferece direito de utilização (incluindo relicenciar código derivado).

**Copyleft (e.g. GPL)** – oferece direito de utilização, proíbe tornar código proprietário (pode obrigar que todo o código que usa a aplicação tenha a mesma licença).

**Noncommercial licence** – oferece direito de utilização para utilização não comercial.

**Proprietary licence** – oferece direito de utilização em troca dum pagamento (utilização normal do copyright).

# TRANSPARÊNCIA

A transparência (da distribuição) é a propriedade relativa a esconder ao utilizador e ao programador das aplicações a separação física dos elementos que compõem um sistema distribuído

- Simplicidade e flexibilidade são objectivos

Em certas circunstâncias, a transparência total é indesejável

- Que fazer em caso de falha?

# SEGURANÇA

Necessidade de proteger os recursos e informação gerida num sistema distribuído

- Recursos têm valor para os seus utilizadores

Segurança tem três componentes:

- Confidencialidade: indivíduos não autorizados não podem obter informação
- Integridade: dados não podem ser alterados ou corrompidos
- Disponibilidade: acesso aos dados deve continuar disponível

Aspetos envolvidos

- Autenticação dos parceiros
- Canais seguros
- Prevenção de ataques de “negação de serviço” (denial of service attacks)

# SEGURANÇA - RGPD

**O Regulamento Geral sobre a Proteção de Dados (RGPD)** (UE) 2016/679 regula a privacidade e proteção de dados pessoais, aplicável a todos os indivíduos na UE e Espaço Económico Europeu (EEU).

É aplicável a todas as empresas que operem no Espaço Económico Europeu, independentemente do seu país de origem.

Pretende garantir que:

- Não se podem guardar e disponibilizar dados sem consentimento explícito;
- Os dados guardados não podem ser utilizados sem que o proprietário tenha dado consentimento explícito;
- O proprietário tem o direito de revogar as permissões em qualquer momento.

# ESCALA

A **escala de um sistema distribuído** é o âmbito que o mesmo abrange assim como o número de componentes.

A escala de um sistema tem várias facetas:

- recursos e utilizadores
- âmbito geográfico (rede local, país, mundo, ...)
- âmbito administrativo (uma organização, inter-organizações)

Um sistema capaz de escalar (escalável) é um sistema que continua eficaz quando há um aumento significativo do número de recursos e utilizadores

- i.e., em que não é necessário alterar a implementação dos componentes e da forma de interacção dos mesmos



# COMO LIDAR COM A ESCALA ?

Para reduzir o número de pedidos tratados por cada componente

- Divisão de um componente em partes e sua distribuição
- Replicação e caching (problema da consistência entre réplicas e caches)

Para reduzir o tempo de acesso de clientes distribuídos geograficamente?

- Geo-replicação – replicar as aplicações (e dados) em diferentes locais geográficos
- Replicação na edge – replicar as aplicações ou dados na periferia da rede (e.g. CDN)

# COMO LIDAR COM A ESCALA ?

Para reduzir dependências entre componentes

- Meios de comunicação assíncronos

Para simplificar o sistema

- Uniformidade de acesso aos recursos e dos mecanismos de cooperação, sincronização, etc.
- Meios de designação universais (independentes da localização e dos recursos)

# AVARIAS, ERROS E FALHAS

Os componentes de um sistema podem **falhar**, i.e., comportar-se de forma não prevista e não de acordo com a especificação devido a **erros** (por exemplo a presença de ruído num canal de comunicação ou um erro de software) ou **avarias** (mecanismo que entra em mau funcionamento)

Num sistema distribuído, as **falhas são geralmente parciais** (num componente do sistema) **e independentes**

- Um componente em falha pode induzir uma mudança de estado incorrecta noutro componente, levando eventualmente o sistema a falhas, i.e., a ter um comportamento não de acordo com a sua especificação.

# COMO LIDAR COM AS FALHAS

## Detectar falhas

- Possível: e.g.: mensagens corrompidas através de checksums
- Pode ser impossível: Falha (crash) num computador remoto
  - Desafio: Funcionar através da suspeição das falhas

## Mascarar falhas (após a sua detecção)

- Exemplos: retransmissão de mensagens, redundância

## Tolerar falhas

- Definição do comportamento na presença de falhas
  - Parar até falhas serem resolvidas; recorrer a componentes redundantes para continuar a funcionar

## Recuperação de falhas

- Mesmo num sistema que tolere falhas é necessário recuperar os componentes falhados. Porquê?
- Problema: recuperar estado do serviço

# PARA SABER MAIS

G. Coulouris, J. Dollimore and T. Kindberg,  
Distributed Systems – Concepts and Design,  
Addison-Wesley, 5th Edition, 2011

Capítulo 1.