Sistemas Distribuídos
Aula 7

Aula passada

- Limitação dos semáforos
- Monitores
- Variáveis de condição
- Semântica de *signal*

Aula de hoje

- Arquitetura de sistemas
- Arquitetura de sistemas distribuídos
- Cliente/servidor

Construindo Sistemas

- Como construir um sistema grande?
  - ex. g++, Linux, SIGA, Gmail?
- Modelar
- Projetar
- Implementar
- Testar

→ **Legal, mas como?**

- Diferentes técnicas/abordagens para cada etapa
  - ex. implementar em C++ ou Python?
  - cada abordagem tem vantagens/desvantagens
  - não há *bala de prata!*

Projetando Sistemas

- Como projetar (*design*) sistemas?
  - como organizar um grande sistema?

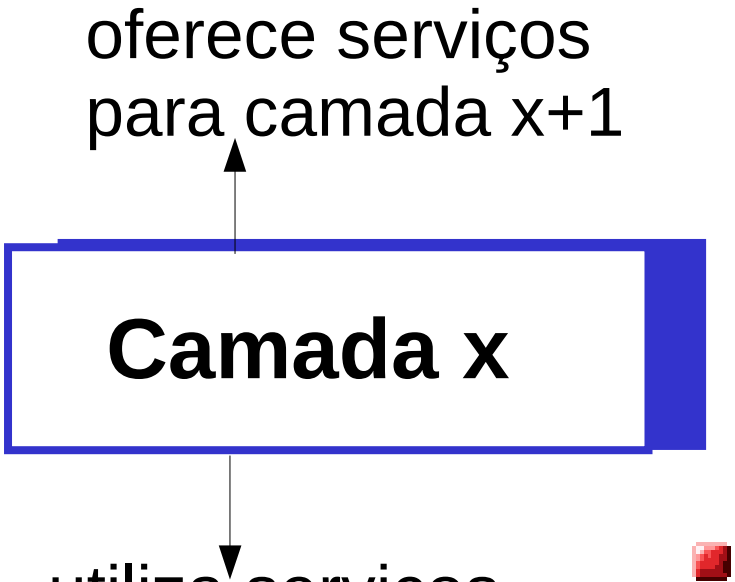
- Adotar uma *arquitetura de sistemas*
  - forma de organizar um sistema grande
  - dividir em componentes/módulos de software
  - módulos com interfaces bem definidas
- Diferentes arquiteturas desenvolvidas ao longo do tempo, com prática e teoria
  - como dividir, como conectar, como trocar → define uma arquitetura de sistema

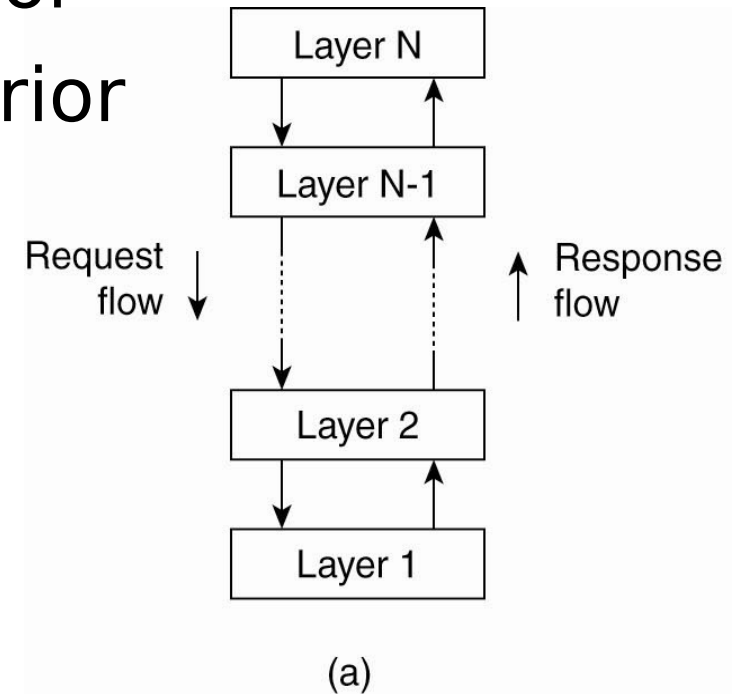
Arquitetura de Sistemas

- Quatro grandes abordagens
  - baseada em camadas (mais antiga)
  - baseada em objetos
  - baseada em eventos
  - centrada em dados (mais nova)
- Cada qual com vantagens/desvantagens
- Não necessariamente ortogonais
  - grandes sistemas reais misturam as abordagens

Arquitetura em Camadas

- Componentes organizados “verticalmente”
  - facilita organização e modificação
- Camada implementa um serviço (*função*)
  - utiliza serviço da camada inferior
  - oferece serviço a camada superior



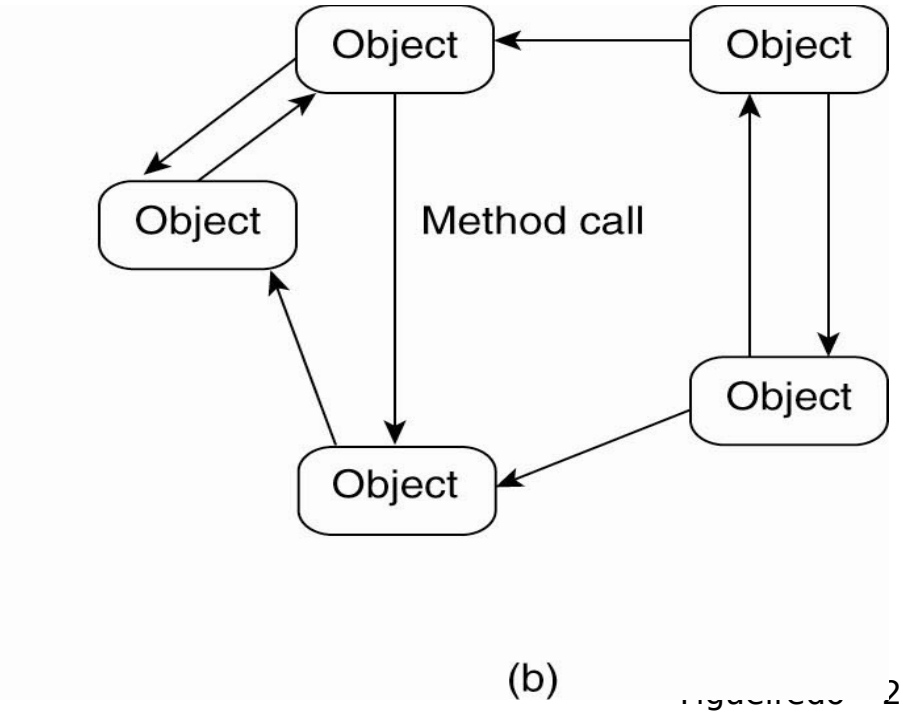


- Mais clássica, muitos exemplos
  - TCP/IP, SO, etc

Arquitetura em Objetos

- Componentes organizados como um grafo
  - maior flexibilidade, maior eficiência
- Objetos implementam serviços (*funções*)
  - utilizam serviços de outros objetos
  - chamam objetos necessários

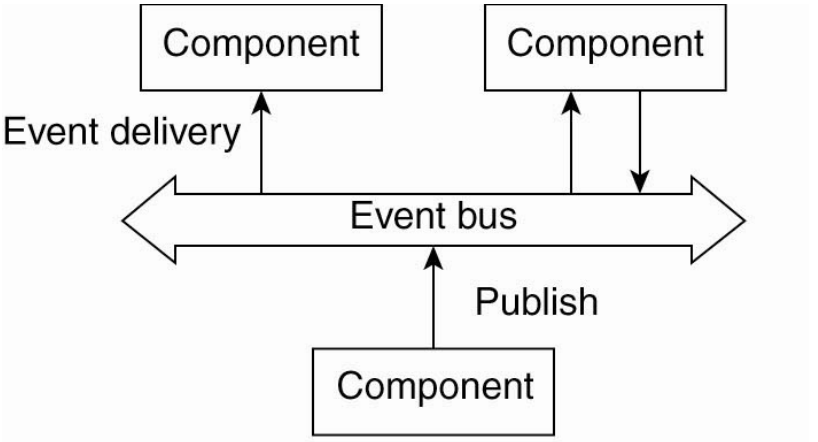
- Exemplos
  - alguns SGBDs (bando de dados)



(b)

Arquitetura em Eventos

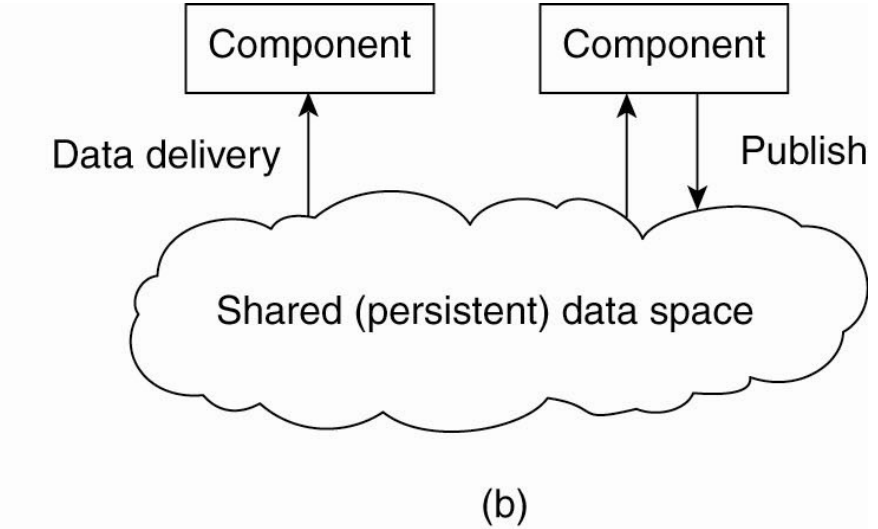
- Componentes organizados em um “barramento “
  - maior simplicidade
- Componentes enviam/recebem eventos
  - eventos tem identidade e informação, processados apenas onde necessário
  - utilizam serviço de outros componentes
- Exemplos
  - Ethernet é um barramento de verdade!
  - *Publish/Subscribe systems*



(a)

Arquitetura Centrada em Dados

- Componentes organizados em estrela
  - maior simplicidade
- Centro da estrela é um grande repositório de dados compartilhado
  - comunicação indireta, através repositório
  - repositório é outro sistema
- Exemplos
  - Google File System
  - Hadoop (Map/Reduce)
  - Content Centric Networks



Sistemas Distribuídos

- Qual é mesmo a definição?

  A collection of independent computers that appears to its users as a single coherent system.

- Como projetar sistemas distribuídos?
  - o que vimos era para sistemas não necessariamente distribuídos
- Novamente, adotar uma *arquitetura de sistemas distribuído*
  - forma de organizar um sistema grande
  - dividir em componentes/módulos de software
  - módulos com interfaces bem definidas

Aquitetura de Sistemas Distribuído

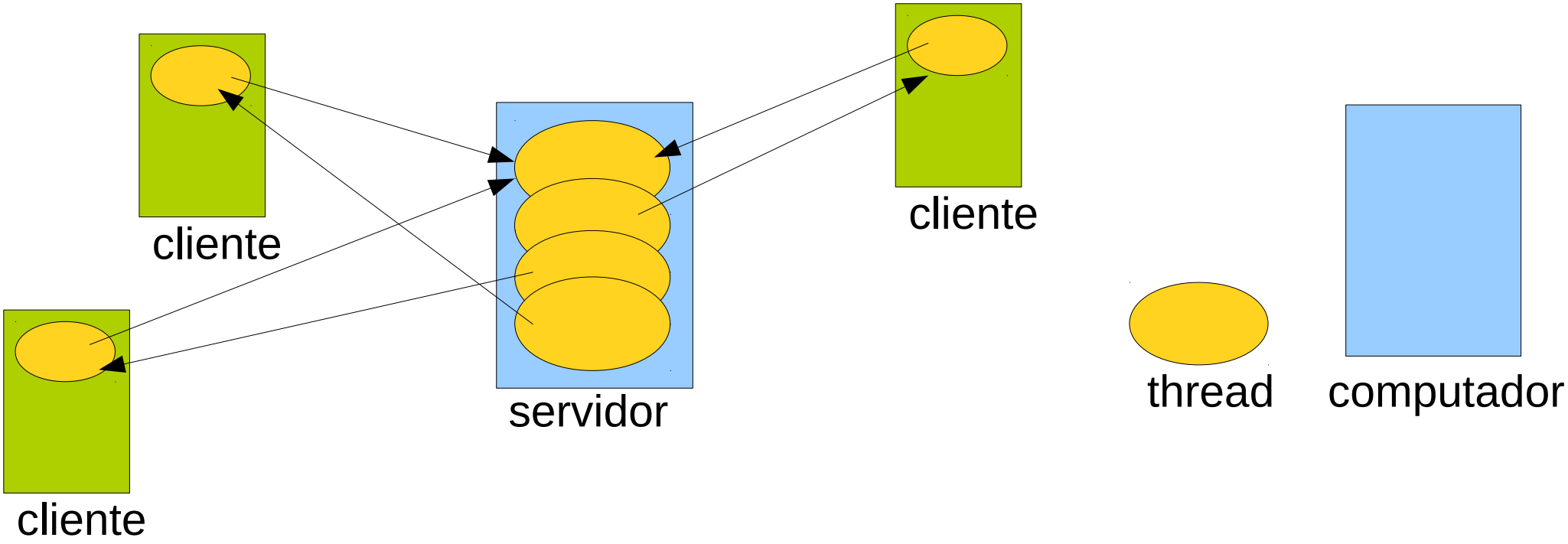
- Duas grandes abordagens
  - cliente/servidor: clássica e mais usada
  - P2P: mais nova e mais difícil
- Cada qual com vantagens/desvantagens
- Não necessariamente ortogonais
  - grandes sistemas reais misturam as abordagens

Definições não são cartesianas!

Cliente/Servidor

- Componentes separados em máquinas que fazem papel de *cliente* e *servidor*
- **Servidor:** oferece um serviço
  - aguarda conexão de um cliente
  - recebe e processa pedido do cliente
  - retorna resultado
- **Cliente:** demanda um serviço
  - se conecta ao servidor
  - envia pedido
  - aguarda resposta
- Papel cliente/servidor bem distintos

Exemplo Cliente/Servidor



- Web: HTTP e todos os aplicativos sobre HTTP (Web, YouTube, Facebook, Netflix, etc)
- Email
- DNS

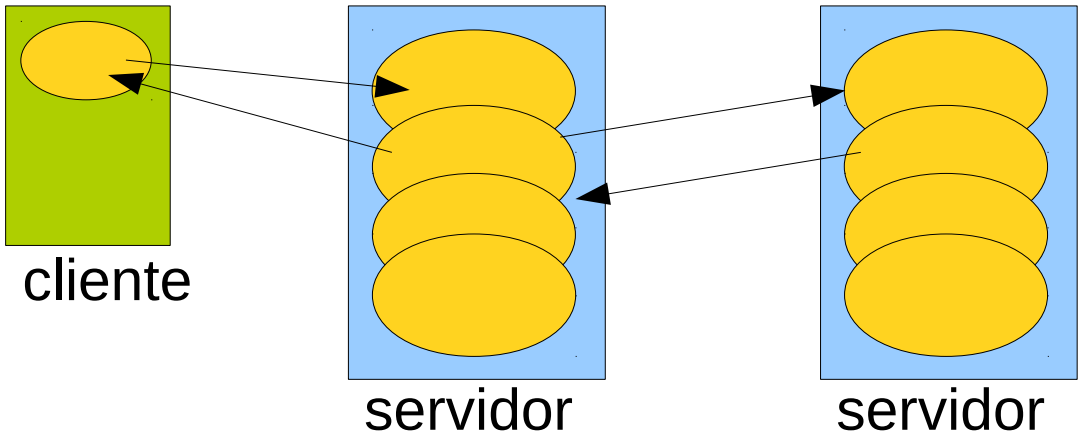
Aspectos Centrais a C/S

- Quais são os aspectos centrais da arquitetura cliente/servidor?

- Como encontrar o servidor?
  - thread do cliente precisa “conectar”
- Como organizar pedidos e respostas
  - um ou múltiplos pedidos por conexão?
- Como dimensionar um servidor
  - atender quantos clientes? Demanda variável?
  - escalabilidade?
- Ponto único de falha? Como ter redundância?
  - múltiplos servidores idênticos

Servidor que é Cliente

- Um servidor pode também fazer papel de cliente?
- *Claro! Servidor é meu, e projeto como quiser*



- Servidor contacta outro servidor (sendo cliente) para exercer sua função
- Sistema de dois níveis (*two-tier system*)
- Exemplo: servidor web e banco de dados