GBC074 – Sistemas Distribuídos

Processos

Baseado no material disponível pelo author em
https://www.distributed-systems.net

Threads

- Conceito de processadores virtuais (software) é fundamental em sistemas operacionais:
  - **Processador**: fornece conjunto de instruções e a capacidade de executá-las automaticamente
  - **Processo**: um processador de software (ou programa em execução) no qual uma ou mais threads podem ser
  - **Thread**: um processador de software mínimo no qual uma série de instruções é executada
- Processadores, processos e threads possuem diferentes definições de contexto

Contexto

- Contexto no:
  - **Processador**: coleção de valores nos registradores de um processador utilizada para execução de instruções (ex., stack pointer, addressing registers, program counter)
  - **Processo**: coleção mínima de valores nos registros e memória necessária para a execução de suas threads (i.e., contexto das threads, além dos valores de registro da MMU).
  - **Thread**: coleção de valores nos registros e memória necessária para para execução das instruções (i.e., contexto do processador + estado).

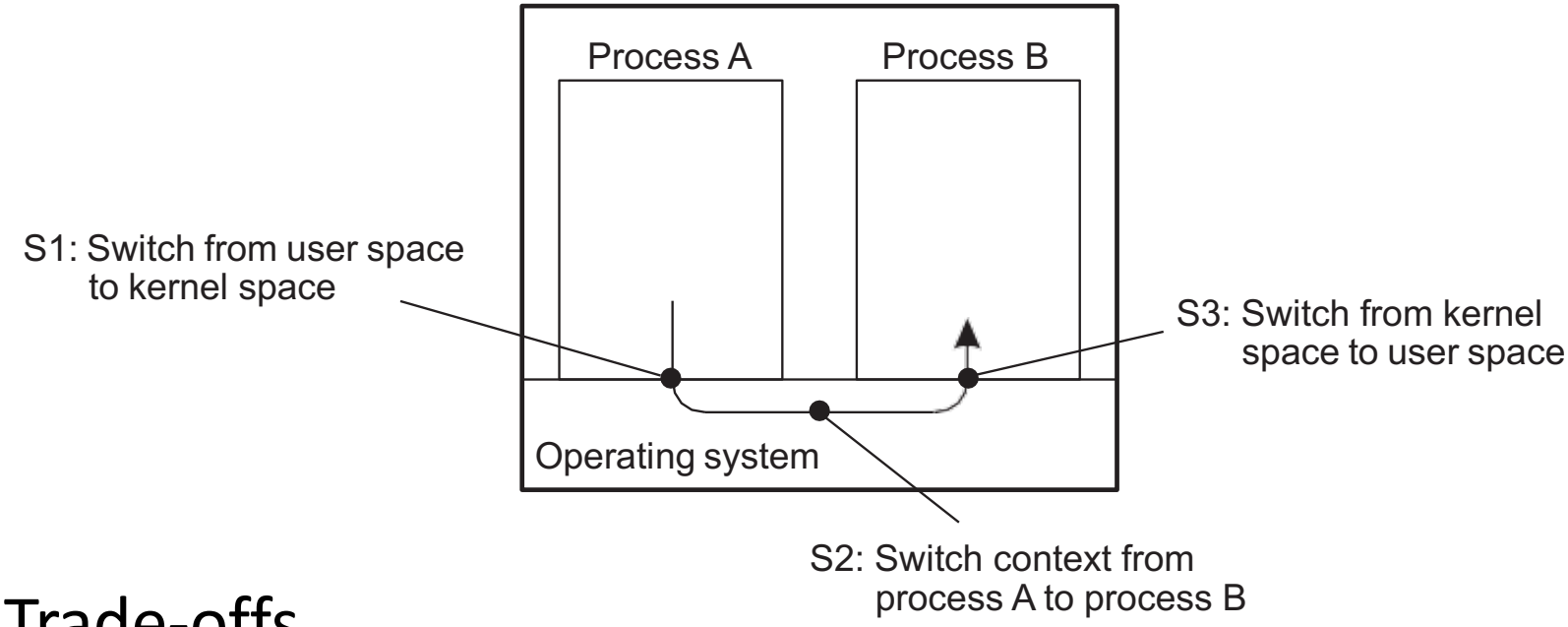
Mudança de contexto

- Threads:
  - Compartilham mesmo espaço de endereçamento
  - Mudança de contexto pode ser feita independentemente do sistema operacional
- Processos:
  - Mudança é geralmente mais cara
  - Envolve o sistema operacional
- Criação/Destruição de threads também é mais "barato" do que a de processos

Threads - motivação

- Evitar bloqueios desnecessários:
  - Processo de thread única bloqueia quando executa I/O e perde a “vez” no processador
  - Processo multi-thread terá apenas a thread em espera, processador ainda pode ser utilizado por outras threads
- Explorar paralelismo:
  - Threads podem ser agendadas para execução em paralelo em sistemas multiprocessadores e/ou multicore
- Evitar troca de contexto:
  - Estruturar grande aplicações como conjunto de threads (em oposição a conjunto de processos)

Troca de contexto: comparação



- Trade-offs
  - Threads usam o mesmo espaço de endereçamento
  - Nenhum suporte do S.O./HW para proteger acesso por uma thread à memória de outra
  - Mudança de contexto de threads tende a ser mais rápida

Threads e S.O.

- *User-space vs. kernel-space*

- Solução no **User-space**:
  - Operações manipuladas no contexto de um único processo
  - Implementações tendem a ser extremamente eficientes
  - Operações no kernel são feitas em nome do processo ⇒ se o kernel bloqueia a thread, todo o processo é bloqueado.
  - Threads e eventos externos:
    - Threads que bloqueiam por eventos ⇒
      se o kernel não distingue threads,
      como sinalizar os eventos para a thread correta?

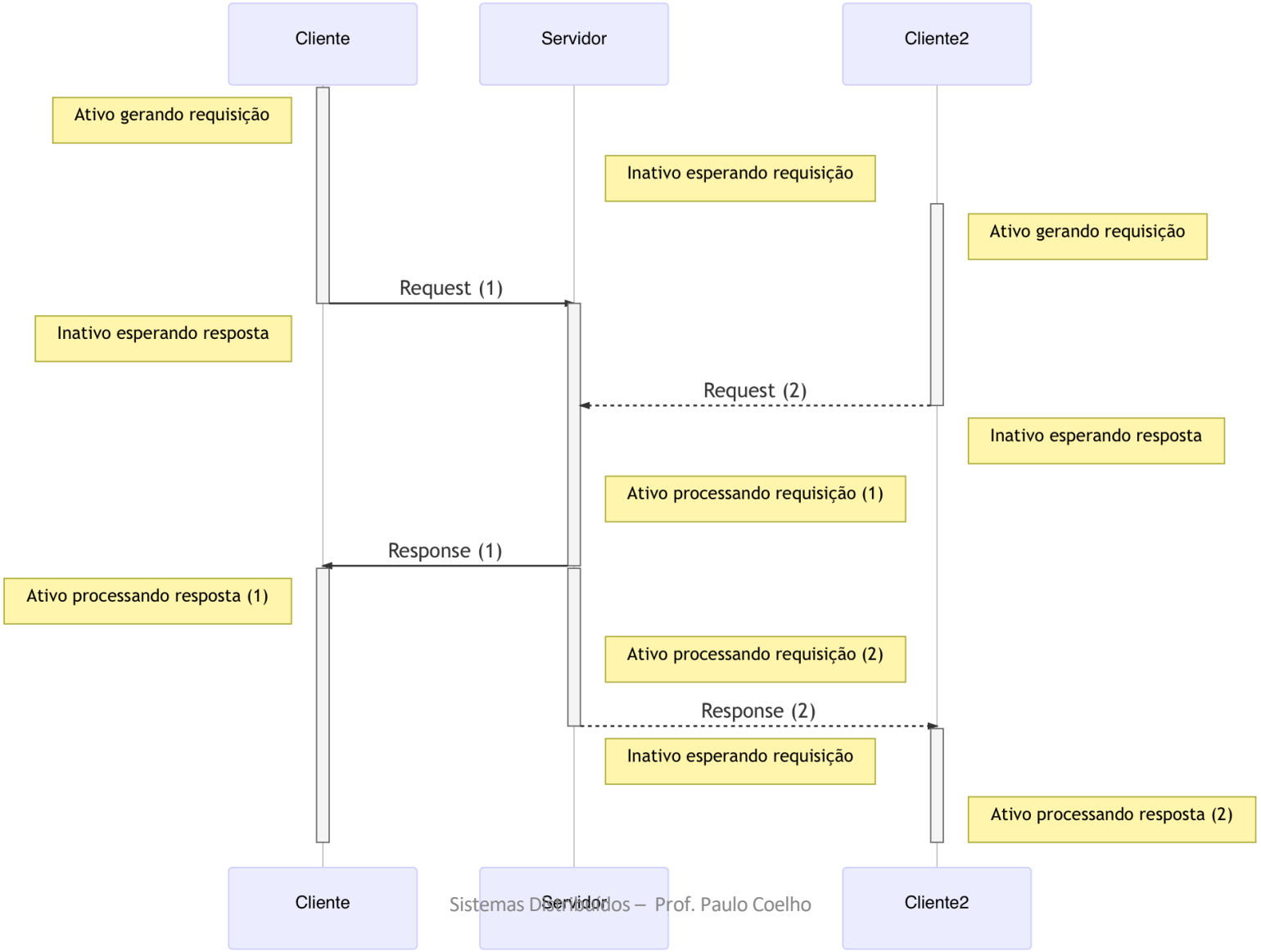
Threads e S.O.

- *User-space vs. kernel-space*

- Solução no **kernel-space**:
  - Kernel implementa a gestão de threads.
  - Todas as operações são *system calls*
  - Operações que bloqueiam não são mais problema
  - Manipulação de eventos também é mais direta
  - Perda na eficiência devido à mudança de contexto para cada operação sobre a thread
- Conclusão ("só que não"):
  - Misturar implementações no user- e kernel-space (LWP)
  - Complexidade desmotivou adoção

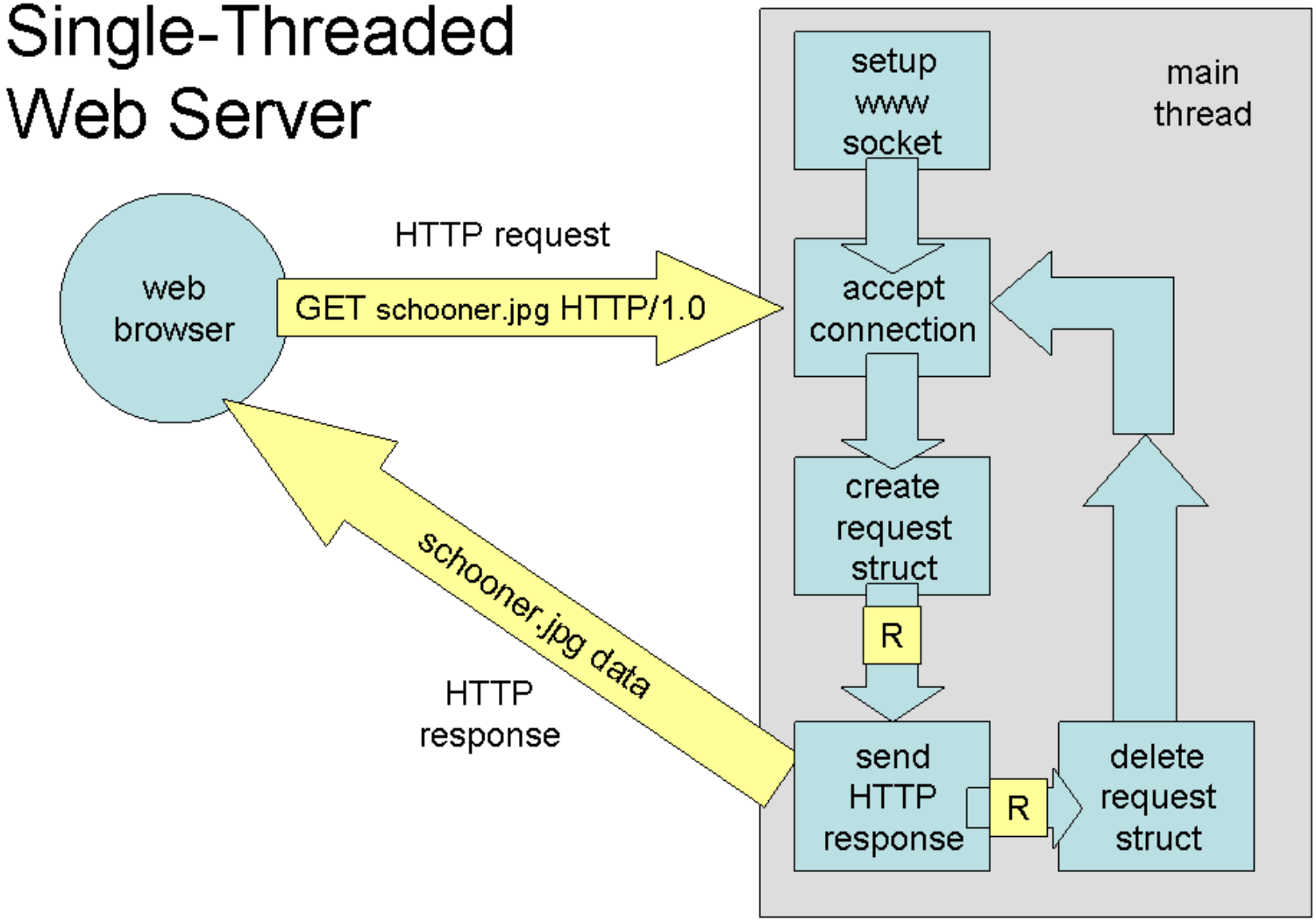
Uso de threads: cliente/servidor

- Sem threads:



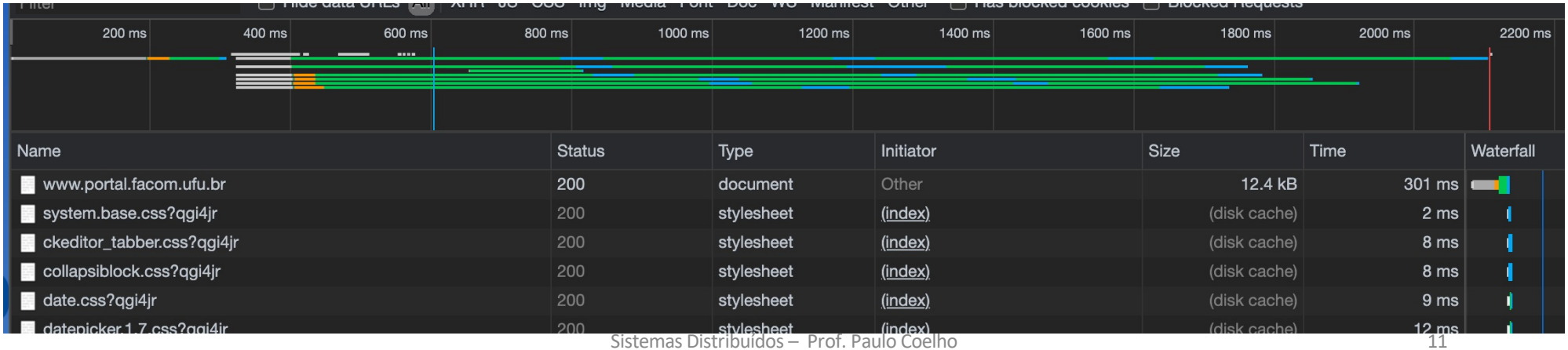
Uso de threads: cliente/servidor

- Sem threads:



Uso de threads: cliente/servidor

- *Multithreaded web client*
  - Esconde latências da rede
  - Web browser lê página HTML e percebe que mais arquivos necessitam ser descarregados
  - Cada arquivo é baixado por uma thread diferente
  - Cada thread faz uma requisição HTTP (bloqueante)
  - Assim que arquivos são recebidos, são exibidos

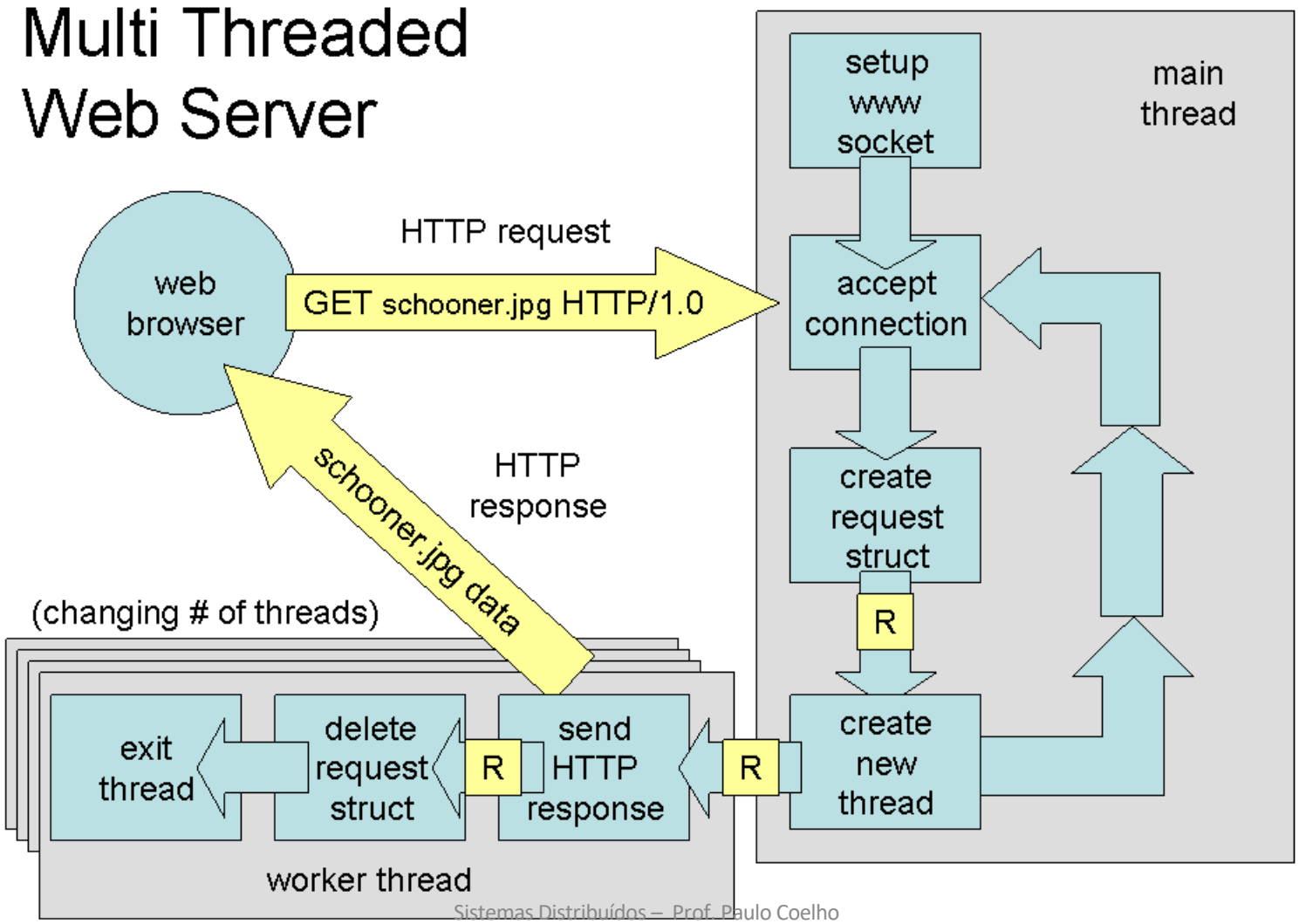


Uso de threads: cliente/servidor

- *Multithreaded web server*
  - Melhor performance
  - Iniciar thread é mais barato
  - Servidor single-threaded impede escalar em sistema multiprocessador
  - Para os clientes: esconde latência respondendo a requisições em paralelo
  - Melhor estruturado:
    - Servidores tem alta demanda de I/O
    - Chamadas bloqueantes simples simplificam a estrutura
    - Programas tendem a ser menores e mais fáceis de entender:
      - Fluxo de controle simplificado

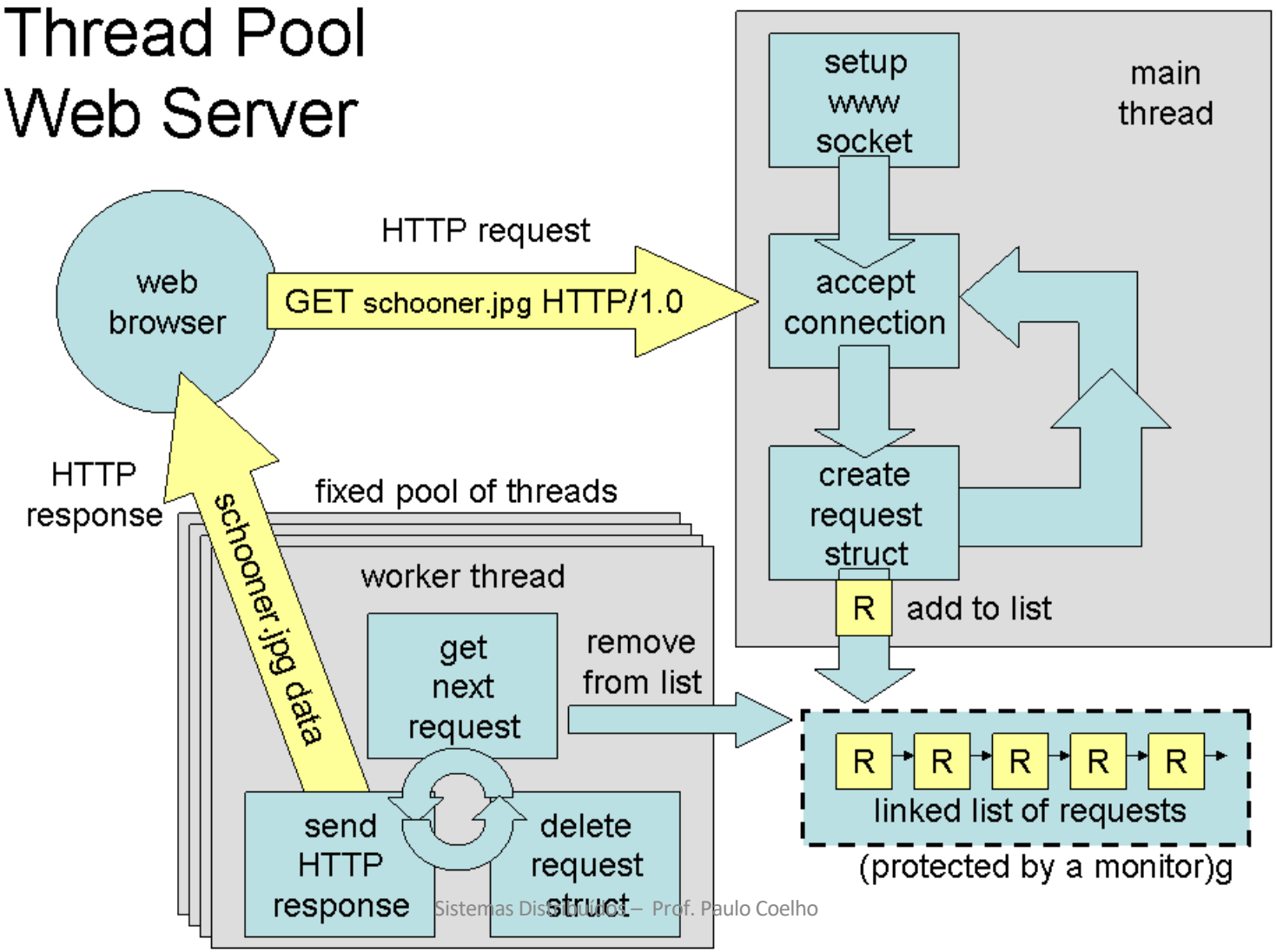
Uso de threads: cliente/servidor

- *Multithreaded web server*

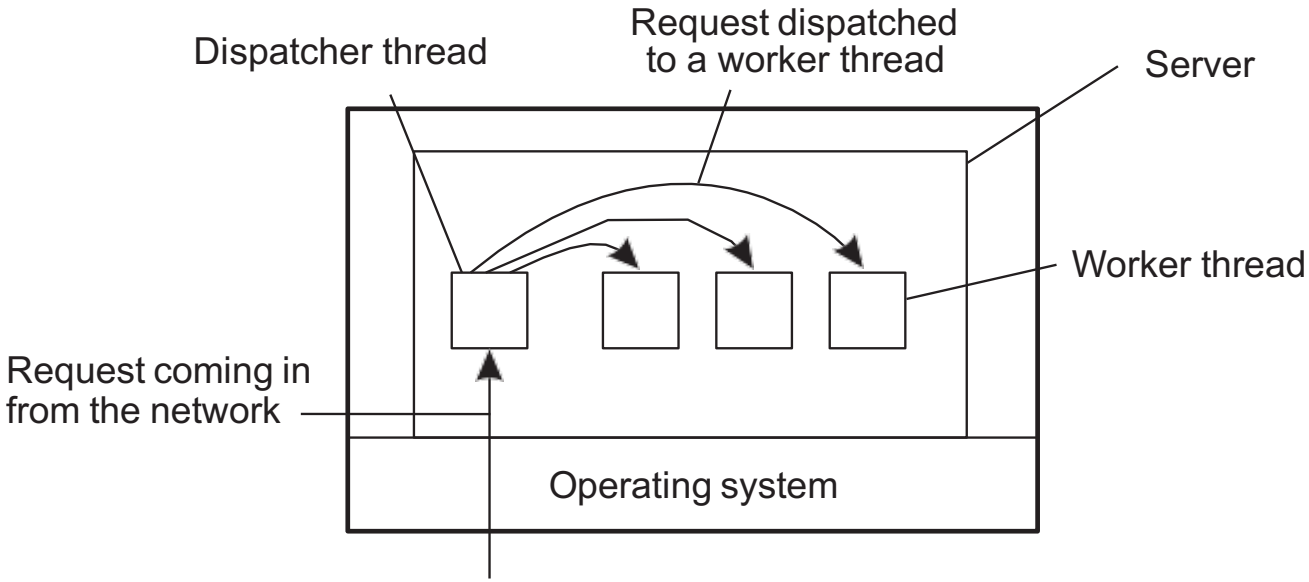


Uso de threads: cliente/servidor

- *Multithreaded web server*



Uso de threads: dispatcher/worker



Model	Characteristics
Multithreading	Paralelismo, system calls bloqueantes
Processo single-threaded	Sem paralelismo, system calls bloqueantes
Finite-state machine	Paralelismo, system calls não-bloqueantes

Servidores e o estado

- **Sem estado**
  - Nunca mantem informação acurada sobre estado do cliente após atender requisição
  - Não grava se algum arquivo foi aberto/acessado (fecha após acesso)
  - Não promete invalidar cache
  - Não mantém listagem de clientes
- Consequências
  - Clientes e servidores completamente independentes
  - Inconsistências reduzidas
  - Perda de performance (não pode antecipar comportamento do cliente)

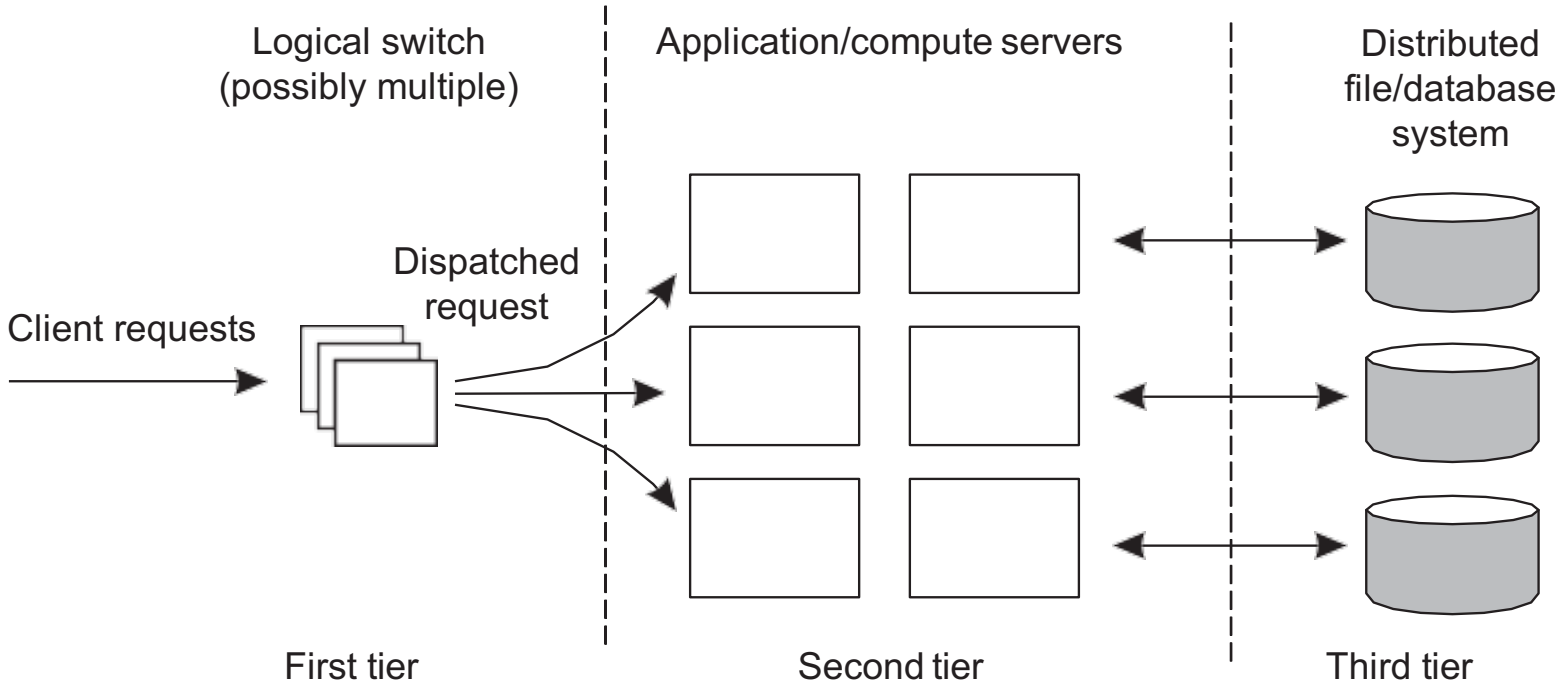
Servidores e o estado

- **Com estado**
  - Mantém estado dos clientes
  - Grava arquivo abertos (pode fazer prefetching)
  - Conhece cache do cliente e permite que mantenha cópias locais de dados compartilhados

- Observação
  - A performance pode ser melhorada
  - Clientes com cópias locais

Servidores em "3 camadas"

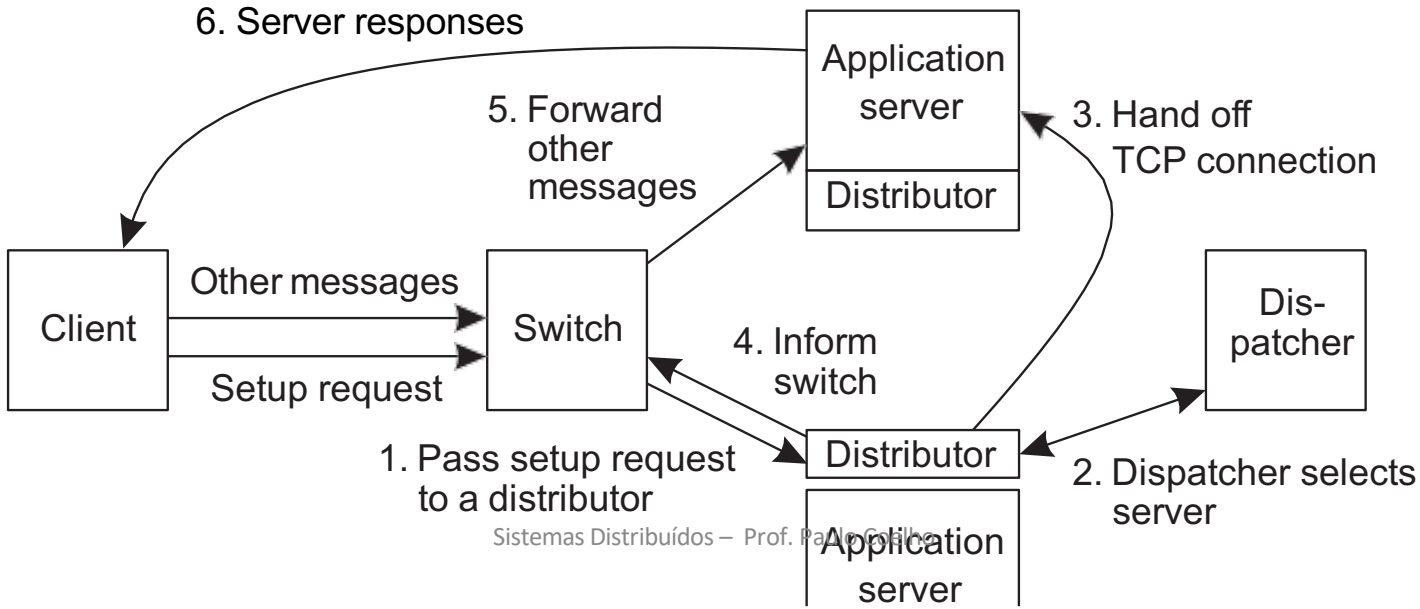
- Cenário típico:



- Primeira camada geralmente é responsável por enviar requisições ao servidor apropriado

Server clusters: mesma rede

- The front end pode ficar sobrecarregado
  - Comutação na camada de transporte:
    - Front end passa a requisição TCP para um dos servidores
    - Realiza algumas medições de performance para decisão
  - Distribuição baseada no conteúdo:
    - Front end lê conteúdo da requisição e seleciona melhor servidor
  - Combinação das duas soluções:



Servidores espalhados na Internet

- Observação
  - Possibilidade de problemas administrativos
  - Resolvido com uso de data centers de único provedor
- Despacho da requisição:
  caso localidade seja importante
  - Abordagem típica – uso de DNS:
    - Cliente procura serviço através do DNS: IP do cliente é parte da requisição
    - Servidor DNS mantém lista das réplicas para o serviço, retornando endereço do servidor mais local
- Transparência para o cliente
  - DNS resolver age no lugar do cliente
  - Pode estar longe do cliente