
# Pós-Graduação em Computação Distribuída e Ubíqua

## INF628 - Engenharia de Software para Sistemas Distribuídos
## Estilos Arquiteturais para Sistemas Distribuídos

Sandro S. Andrade
sandroandrade@ifba.edu.br

# Objetivos


- Apresentar os principais estilos arquiteturais utilizados no projeto de sistemas distribuídos modernos (além do MapReduce já visto)
- Realizar uma discussão sobre os atributos de qualidade induzidos por cada um destes estilos arquiteturais e seus cenário de uso mais comuns

# Objetivos


- Estilos a serem apresentados:
  - REST
  - CREST
  - Outras variações do REST:
    - A+REST, R+REST, REST+D, REST+E, ARRESTED
  - Tiles

# Arquiteturas para Sistemas Distribuídos


- Tentam proporcionar a "ilusão" da transparência de localização
- Na prática, entretanto, as seguintes hipóteses assumidas frequentemente se provam falsas:
  - A rede é confiável
  - A latência é zero
  - A largura de banda é infinita
  - A rede é segura
  - A topologia não muda
  - A rede possui um administrador
  - O custo de transporte é zero
  - A rede é homogênea

# Arquiteturas para Sistemas Distribuídos


- Tentar resolver esses problemas implica na tomada de decisões arquiteturais particulares:
  - Se a rede não é confiável então a arquitetura do sistema pode precisar ser dinamicamente adaptável
  - A presença de latência pode requerer que as aplicações trabalhem com base em valores localmente estimados de mensagens, baseados em dados previamente recebidos
  - Limitações e variabilidade na largura de banda pode requerer a inclusão de estratégias adaptativas para acomodar condições locais
  - A existência de mais de um domínio administrativo pode demandar a introdução de mecanismos de trustness
  - Heterogeneidade na rede pode demandar camadas de abstração ou o uso de padronizações

# Arquiteturas para Sistemas Distribuídos - REST


- REpresentational State Transfer (REST)
  - Como o REST foi desenvolvido ? O que motivou sua criação ? Quais influências arquiteturais prévias foram combinadas para produzir o REST ?
- Fatores motivadores:
  - Características da web como aplicação e propriedades da sua forma de implantação e utilização
  - A web é uma aplicação multi-usuário de hipermídias distribuídas
  - Visto que a informação deve ser trazida até o usuário todos os problemas de rede anteriormente citados estão presentes

# Arquiteturas para Sistemas Distribuídos - REST


- Fatores motivadores:
  - A web é uma aplicação multi-owner e heterogênea
  - O espaço de informações não está sob controle de uma única autoridade (aplicação descentralizada)
  - Nada deve ser assumido sobre a uniformidade ou qualidade das implementações
  - Visão prospectiva: novos tipos de informação ou processamento podem ser adicionados, demandando mecanismos para extensão facilitada
  - Qualquer estilo arquitetural para web deve ser capaz de suportar o constante crescimento de usuários e servidores

# Arquiteturas para Sistemas Distribuídos - REST


- Derivação do REST – heranças arquiteturais:
  - Separação em camadas: para melhorar a eficiência, possibilitar a evolução independente dos elementos do sistema e prover robustez
  - Replicação: para reutilizar informação e diminuir a latência e a contenção
  - Limited Commonality: para satisfazer a necessidade de operações extensíveis e universalmente compreendidas
  - Extensão dinâmica: através de mobile code → extensibilidade independente
  - Requisições ao servidor são sempre context-free → escalabilidade e robustez

# Arquiteturas para Sistemas Distribuídos - REST


- Separação em camadas:
  - Arquitetura Client-Server (CS):
    - Software para GUI (browser) evolui independentemente do software que gerencia os dados e responde às requisições (servidor)
    - Simplifica os componentes e possibilita sua otimização
  - Requisições independentemente processáveis:
    - Não há registro de sessões de interação com os clientes
    - O servidor pode desalocar qualquer recurso utilizado para atender a requisição, logo após o término do atendimento
    - Alto suporte a escalabilidade
    - Qualquer servidor que utilize o mesmo backend (banco de dados) pode tratar a requisição → balanceamento de carga
    - Intermediários para selecionar o servidor que atenderá a requisição, realizar processamento parcial e segurança

# Arquiteturas para Sistemas Distribuídos - REST


- Replicação:
  - Oportunidade para melhor desempenho e robustez
  - As camadas intermediárias abstraem a replicação dos clientes
  - Uma forma de replicação é caching de informação (proxies – próximo dos clientes; gateways – próximo do servidor)
  - Visto que as requisições são auto-contidas um único cache pode servir vários clientes
  - O desempenho é melhorado pois o cache pode já retornar os dados solicitados, sem envolver o servidor

# Arquiteturas para Sistemas Distribuídos - REST


- Limited Commonality:
  - As partes de uma aplicação distribuída se comunicam ou utilizando uma biblioteca em comum ou adotando padronizações de comunicação
  - Ao utilizar uma padronização, múltiplas implementações independentes podem existir e é uma solução superior em sistemas heterogêneos e abertos: incentiva-se a inovação, especializações locais e permite o uso de diversas plataformas de hardware

# Arquiteturas para Sistemas Distribuídos - REST

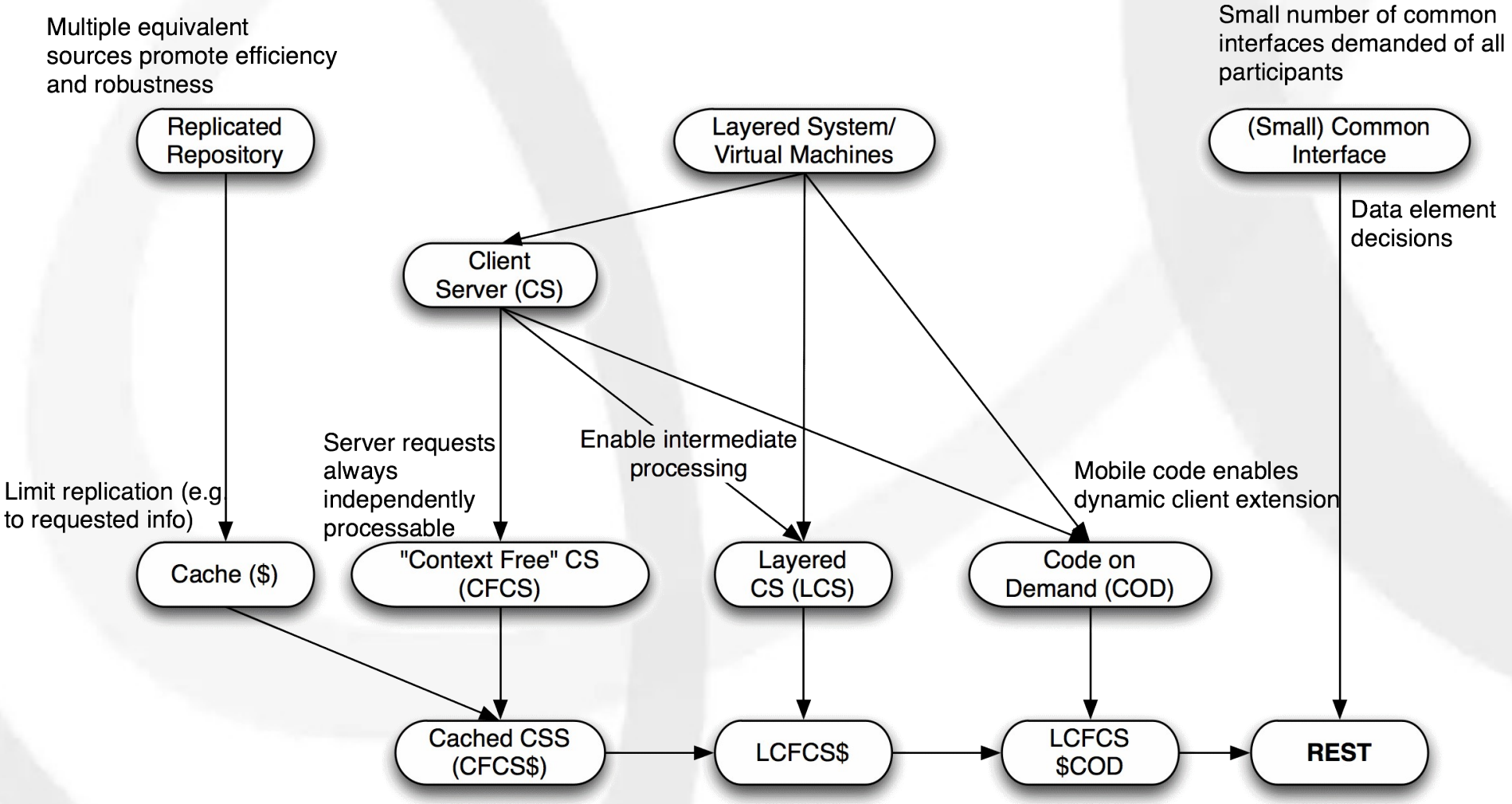

- Limited Commonality:
  - Que tipo de padronização de comunicação deve ser utilizada:
    - Feature-Rich Style
    - Limited Commonality: i) especifica-se como a informação é identificada e representada sob a forma de meta-dados e ii) especifica-se poucos serviços básicos que toda implementação deve suportar
  - É eficiente para a transferência de dados hipermídia grandes (objetivo da web) mas não é ótimo para outras formas de interação

# Arquiteturas para Sistemas Distribuídos - REST


- Extensão dinâmica:
  - Ao permitir que clientes recebam dados arbitrários, descritos por meta-dados, suas funcionalidades podem ser estendidas dinamicamente
  - Exemplos de dados recebidos: script, applet, etc
  - O REST incorpora o estilo arquitetural mobile code (variação do code-on-demand)

# Arquiteturas para Sistemas Distribuídos - REST

- Derivação do REST:



# Arquiteturas para Sistemas Distribuídos - REST


- Derivação do REST:

| |
|---|
| **Resumo**: *client-server* restrito, com foco na comunicação de elementos de dados |
| **Componentes**: *origin server* (apache, IIS, etc); *gateway* (squid, CGI, etc); *proxy*; *user agent* (Safari, Internet Explorer, *search bots*, etc) |
| **Conectores**: *client-side interface* (libwww, etc); *server-side interface* (Apache API, etc); *tunnel* (SOCKS, SSL após HTTP CONNECT) |
| **Elementos de Dados**: *resource*; *resource identifier* (URL); *representation*; *representation meta-data* (MIME); *resource meta-data* (*source link*, *alternates*); *control data (if-modified-since*, *cache-control)* |
| **Topologia**: *multi-client / multi-server* com *proxies* intermediários |
| **Restrições Impostas**: os seis princípios do REST: |
| **P1**: um *resource* é uma abstração de uma informação, identificado por uma URL |
| **P2**: um *resource representation* é uma sequência de *bytes* e meta-dados associados |
| **P3**: todas as interações são *context-free* |
| **P4**: componentes executam somente um conjunto pequeno de métodos bem definidos |
| **P5**: incentivo a operações idem-potentes e uso meta-dados para suportar *cache* e reuso |
| **P6**: a presença de intermediários (de filtragem, de redirecionamento, etc) é desejada |

# Arquiteturas para Sistemas Distribuídos - REST

- Derivação do REST (cont.):

| |
|---|
| **Qualidades Induzidas**: produção de aplicações em rede abertas, extensíveis e altamente escaláveis; redução da latência na rede; facilitação da implementação independente e eficiente de componentes |
| **Usos Típicos**: *World Wide Web* (hipermídia distribuída) |
| **Precauções**: diversos *sites web* e livros caracterizam ou exemplificam os princípios do REST de forma incorreta ou incompleta |
| **Relacionamento com Linguagens de Programação e Ambientes**: os aspectos dinâmicos e de *code-on-demand* da *web* favorecem linguagens tais como *JavaScript*, *Java*, *Scheme*, *Ruby* e Python |

# Arquiteturas para Sistemas Distribuídos - REST


- Restrições do REST:

| Domain Property | REST-imposed Constraint | REST-induced Benefit/Property |
|---|---|---|
| A user is interested in some hypermedia document stored externally | User Agent represents User Origin Server has hypermedia docs | User Agent initiates pull-based request from an Origin Server<br>Requests from User Agent have a clearly associated response from an Origin Server |
| Hypermedia documents can have many formats | **Metadata** describing **representation** presented with document | User Agent can render documents appropriately based on metadata |
| Many independent hypermedia origin servers | Define a set of common operations with well-defined semantics (**Extensible methods**) | User Agent can talk to any Origin Server |
| A document may have multiple valid depictions with differing metadata | Distinction between abstract **resource** & transferred **representation**<br>Metadata can be sent by user agent that indicates preferences (**Internal transformation**) | User Agent can request resource and receive an appropriate representation based on presented metadata<br>One-to-many relationship between a resource and representation |

# Arquiteturas para Sistemas Distribuídos - REST


- Restrições do REST:

| | | |
|---|---|---|
| Hypermedia documents are usually organized hierarchically + uniquely identified servers | Resources explicitly requested by name | User Agent can 'bookmark' a location and return to it later |
| | Origin Server controls own namespace | Origin Server can replace backend and persist identical namespace |
| Origin Server may not be able to receive inbound connections from the world<br>User Agent may not be able to make outbound connections to the world | Gateway node (Origin Servers) | Even if direct paths are not available between two nodes, indirect paths may be available through REST intermediaries |
| | **Proxy** node (User Agents) | |
| | No assumption of persistent connection or routing; Hop-by-hop only<br>Any **state** must be explicitly transferred in each message | Gateway and Proxy nodes treat routing of each message independently (packet-switched)<br>Duplicate copies of Origin Servers may be deployed |
| Common hypermedia operations do not change the content + Documents may change over time<br><br>REST nodes may need to handle large amounts of traffic or otherwise optimize network bandwidth | Idempotent methods | Ability to reuse a representation |
| | **Cacheability** components introduced | Each node can independently have a local cache of documents; cache can re-serve representations |
| | Expiration control data can be presented with a representation | Mechanism to locally expire cached content |
| | Control data presented in requests to indicate current cached version | Mechanism to cheaply re-validate 'stale' content in the cache |

# Arquiteturas para Sistemas Distribuídos - REST


- O REST é certamente um estilo poderoso para aplicações baseadas em rede que apresentam problemas de latência e agências (limites de autoridade)
- É também um guia para que outros arquitetos criem outros estilos especializados
- Análise de trade-offs, como o número de operações na especificação da interface, mostra que a construção de tais estilos pode não ser trivial
- A combinação de restrições obtidas de estilos arquiteturais simples pode ser uma ferramenta poderosa e específica

# Arquiteturas para Sistemas Distribuídos - CREST

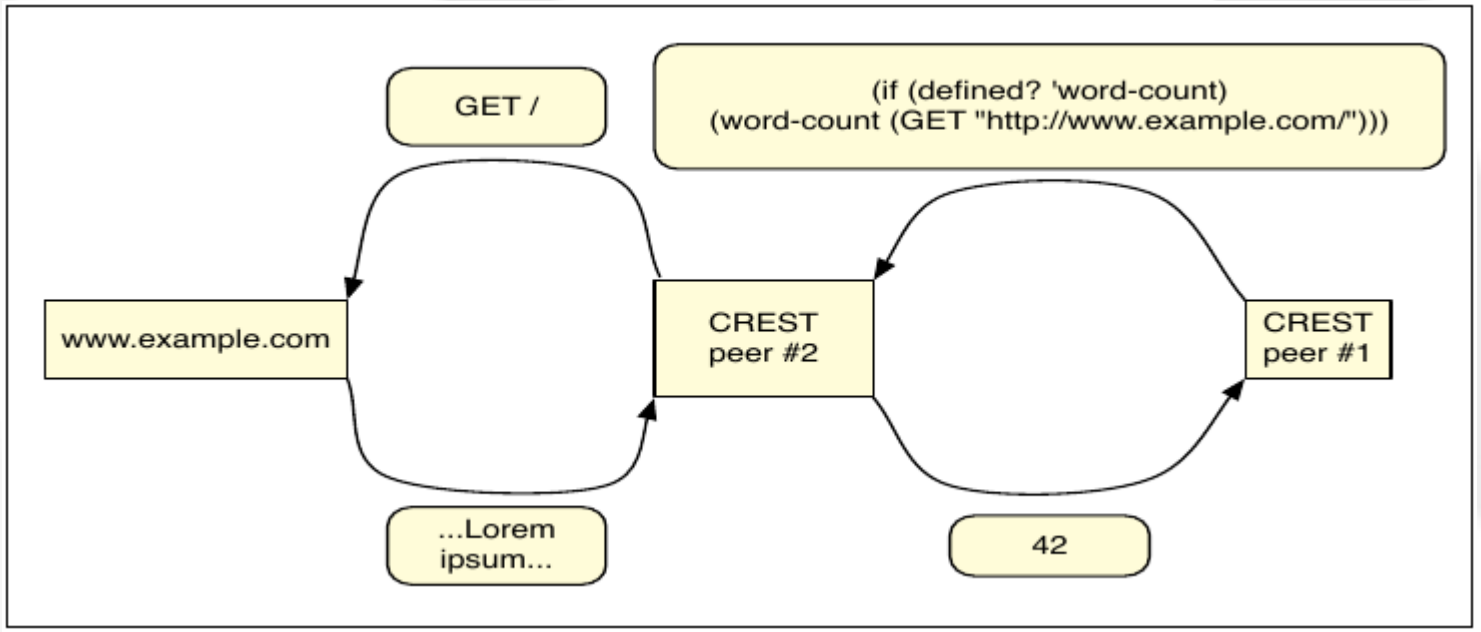

- Computational REST:
  - Surgiu da constatação empírica da dissonância de certas aplicações web em relação ao estilo arquitetural REST
  - Tais aplicações são centradas em computações em vez de em representações
  - A idéia do CREST é criar uma computational-centric web, elevando os códigos móveis a entidades de primeira-classe

# Arquiteturas para Sistemas Distribuídos - CREST


- Computational REST:
  - Uma URL passa a denotar um recurso computacional:
    - Clientes emitem requisições sob a forma de programas, servidores executam estes programas e enviam os valores de retorno
    - O retorno pode ser um valor primitivo, uma lista, um programa, uma "continuação" de execução de programa, etc
  - Dois mecanismos fundamentais:
    - Remote: solicita a execução de um programa e a obtenção do valor de retorno
    - Spawn: instala uma computação do usuário, a ser executada periodicamente a longo-prazo

# Arquiteturas para Sistemas Distribuídos - CREST

- Exemplo de execução CREST para contagem de palavras de uma página HTML:



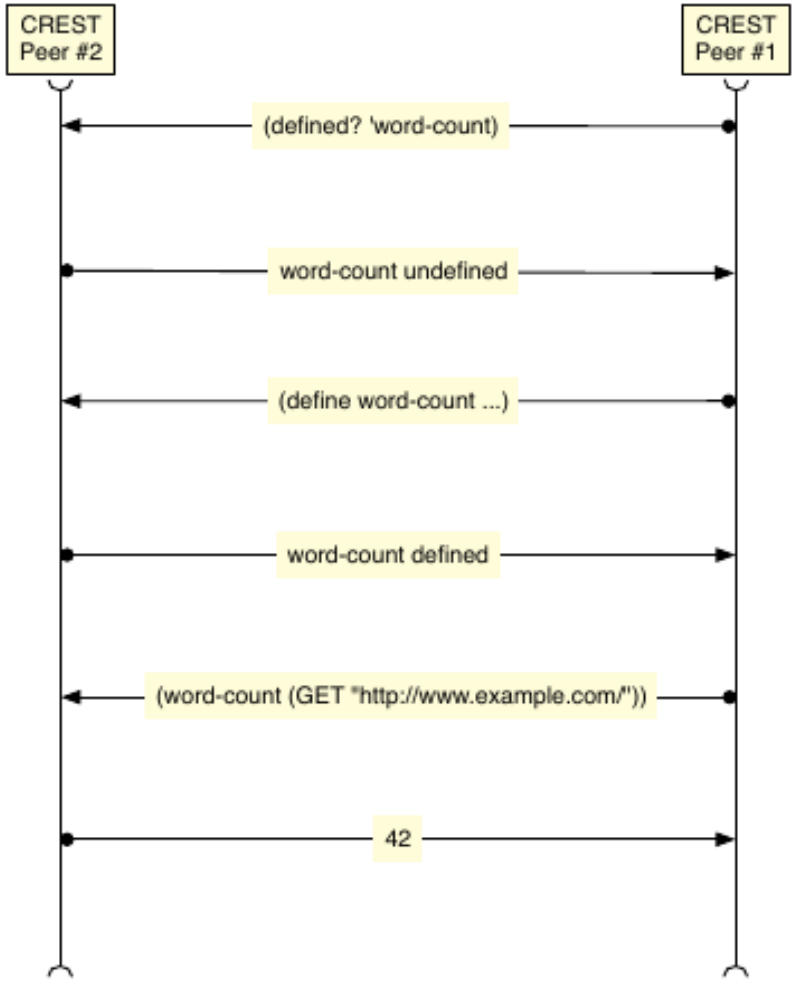
# Arquiteturas para Sistemas Distribuídos - CREST


- Axiomas do CREST:

| |
|---|
| CA1. A resource is a locus of computations, named by an URL. |
| CA2. The representation of a computation is an expression plus metadata to describe the expression. |
| CA3. All computations are context-free. |
| CA4. Only a few primitive operations are always available, but additional per-resource and per-computation operations are also encouraged. |
| CA5. The presence of intermediaries is promoted. |

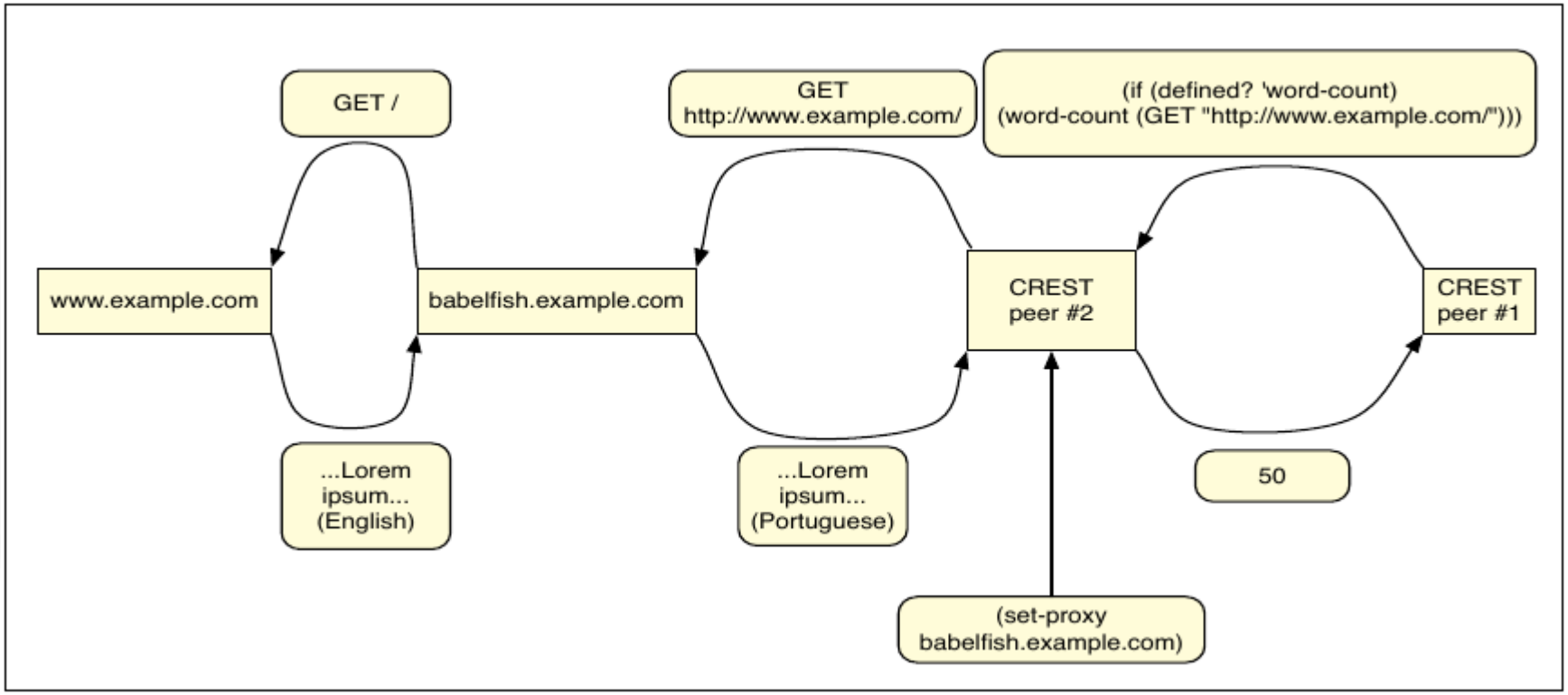
# Arquiteturas para Sistemas Distribuídos - CREST

- Exemplo de adaptação dinâmica de protocolo no CREST:



# Arquiteturas para Sistemas Distribuídos - CREST

- Uso de intermediários:

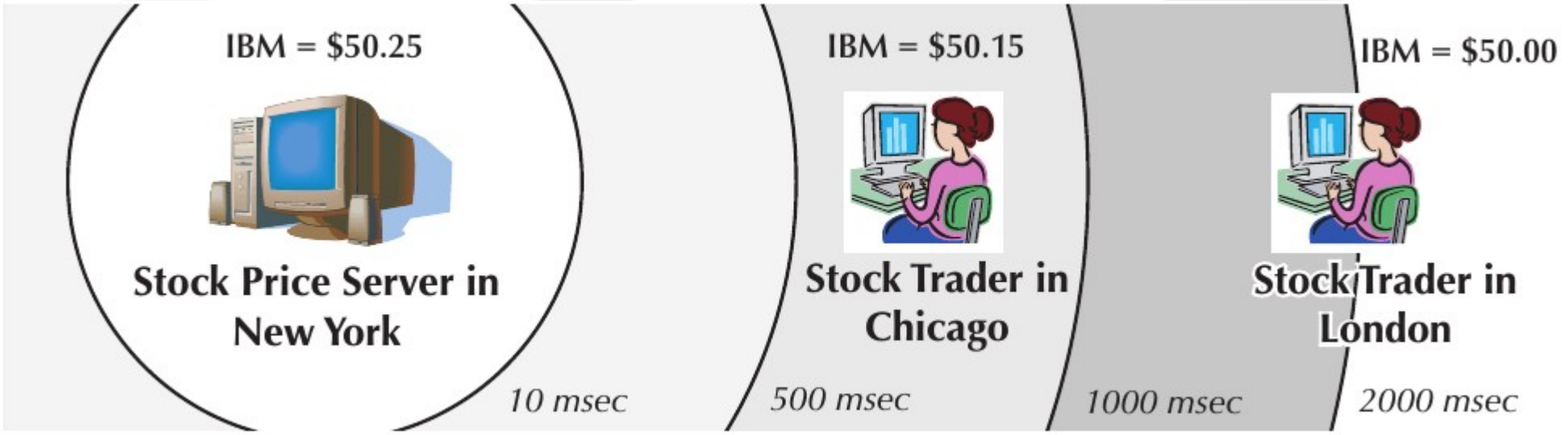


# Arquiteturas para Sistemas Distribuídos – Variações REST


- "Extending the REpresentational State Transfer (REST) Architectural Style for Decentralized Systems", Khare & Taylor, 2004
- Motivação:
  - Múltiplas agências administrativas independentes
  - Maior latência em função da dispersão geográfica
  - Foco em serviços distribuídos descentralizados
  - Desafios: uncertainty e disagreement
  - Solução arquitetural:
    - Restrições na configuração de componentes e conectores para induzir propriedades desejadas

# Arquiteturas para Sistemas Distribuídos – Variações REST

- Latência e 'now horizon':



# Arquiteturas para Sistemas Distribuídos – Variações REST


- Agência:
  - Conceito socialmente construído
  - Refere-se aos interesses divergentes das organizações que efetivamente possuem e operam os computadores que executam os softwares
  - Diferença entre um acesso local:
    - Eu acredito que X=5 neste momento
  - E um acesso remoto a outra agência:
    - Alguém afirma que X era 5 naquele momento

# Arquiteturas para Sistemas Distribuídos – Variações REST

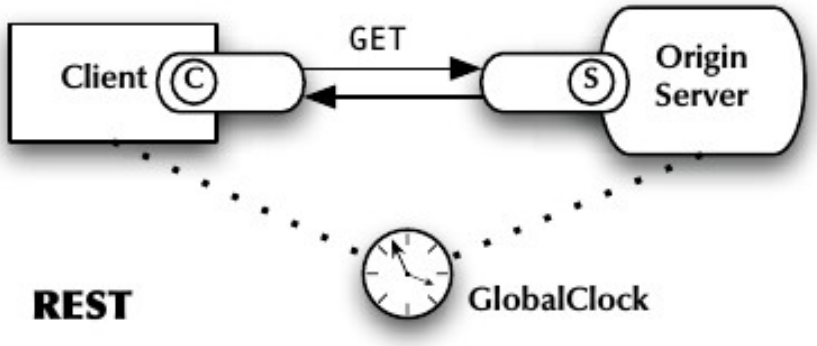

- Impossibilidade FLP (Fisher, Lynch e Paterson)
- Acordo simultâneo:
  - Todos os processos decidem sobre um novo valor "ao mesmo tempo", antes que outro valor seja proposto
  - Tipos de variáveis:
    - Centralizada: requer acordo simultâneo entre o líder e todos os seguidores
    - Distribuída: determinada pela aplicação de uma função compartilhada de decisão em todas as propostas
    - Estimada: acordo simultâneo somente em uma fração do tempo
    - Descentralizada: aplicação de uma função privada em variáveis de outros participantes "trusted"

# Arquiteturas para Sistemas Distribuídos – Variações REST


- Novos estilos propostos:
  - Baseados em consenso:
    - A+REST
    - R+REST
    - REST+D
  - Livres de consenso:
    - REST+E
    - ARRESTED

# Arquiteturas para Sistemas Distribuídos – Variações REST

- Nova visão do REST:



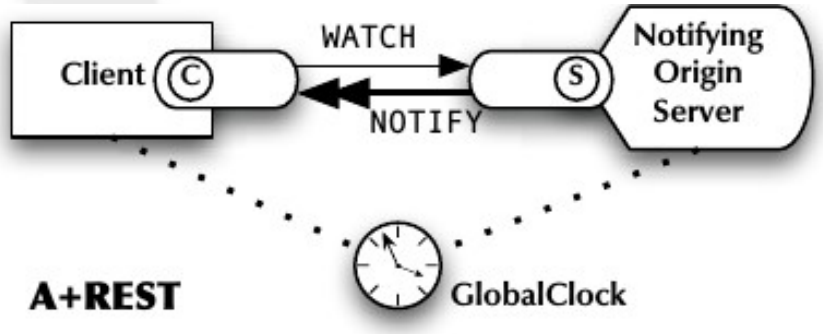

| | |
|---|---|
| Goal | Refer to a centralized resource. |
| New Elements | GLOBALCLOCK makes explicit how clients, servers, and caches are synchronized. |
| New Constraints | ORIGINSERVER must always specify a consistent expiry deadline if the resource is ever to be updated. |
| Induced Property | *Consensus:* Ensures that local resource proxies ***could*** agree with leader. [REST+Polling would guarantee it.] |

# Arquiteturas para Sistemas Distribuídos – Variações REST


- A-REST: Asynchronous REST
- Objetivo:
  - Obter acordo simultâneo o mais rápido possível
  - Baseados em notificações via broadcast
  - REST resource reprojetado para ser uma fonte de notificação de eventos
- É um desafio de implementação em redes públicas

# Arquiteturas para Sistemas Distribuídos – Variações REST

- A-REST: Asynchronous REST



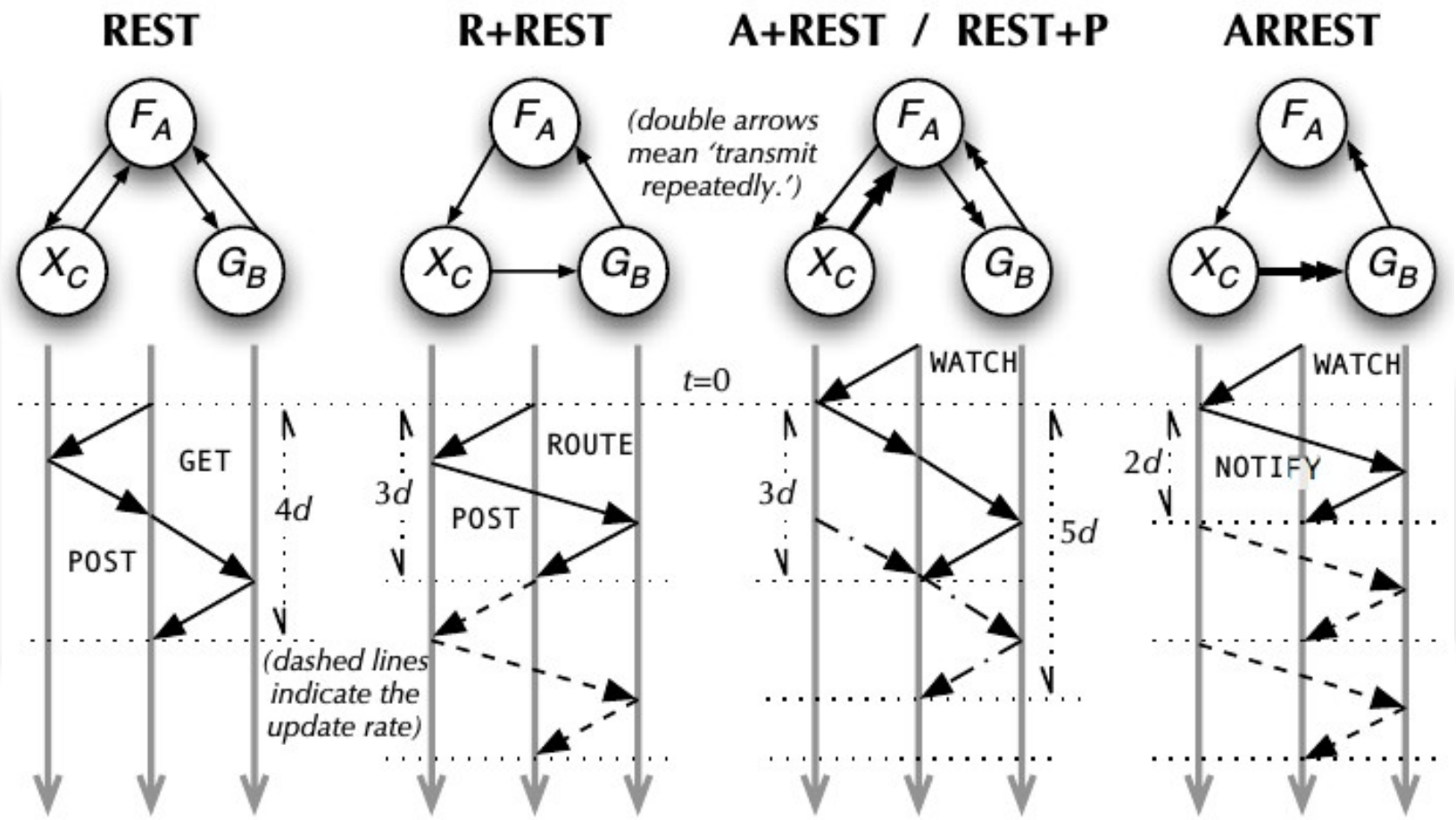

| | |
|---|---|
| Goal | Refer to a changing centralized resource. |
| New Elements | NotifyingOriginServer that can send multiple responses to a WATCH request. |
| New Constraints | Every resource update must lead to transmission of a new representation to all watchers. |
| Induced Property | *Simultaneous Agreement:* Ensures that local resource proxies ***will*** agree with leader's value, even if it is being updated at ½ Hz. |

# Arquiteturas para Sistemas Distribuídos – Variações REST


- R-REST: Routed REST
- Objetivo:
  - Composição de serviços eliminando relacionamentos de confiança não justificados, ao mesmo tempo em que minimiza a latência total

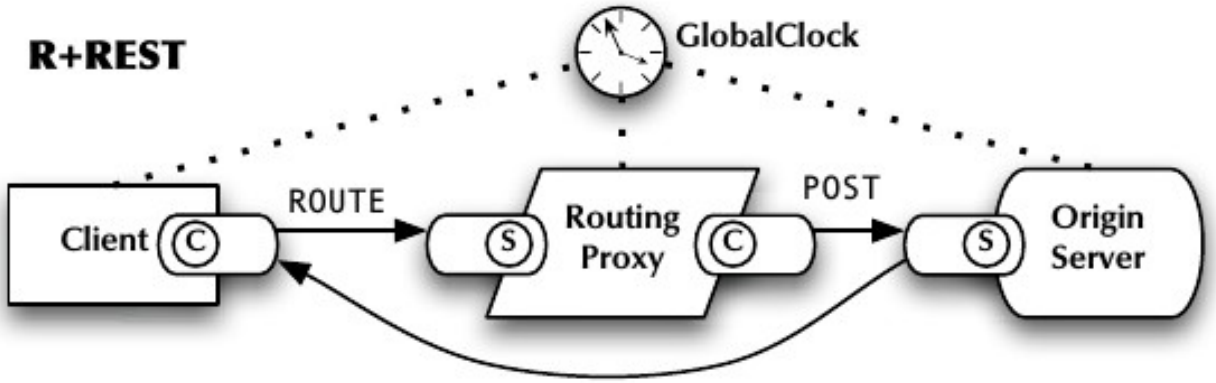
# Arquiteturas para Sistemas Distribuídos – Variações REST

- Diferentes estratégias para F(G(X)):



# Arquiteturas para Sistemas Distribuídos – Variações REST

- R-REST: Routed REST



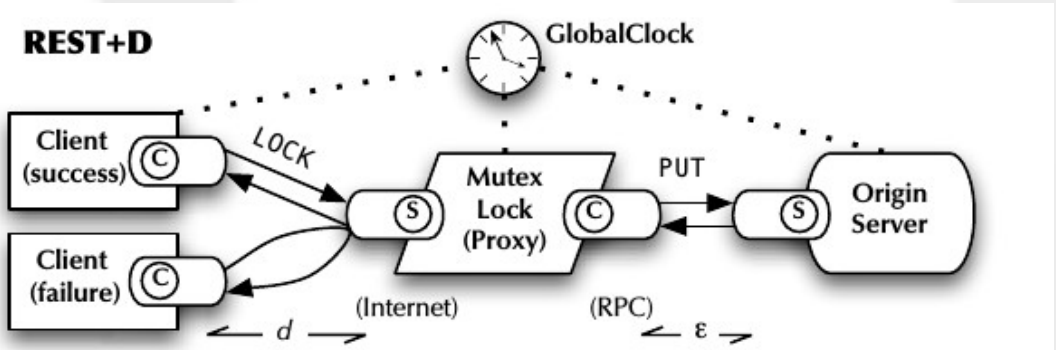

| | |
|---|---|
| Goal | Compose services provided by multiple agencies. |
| New Elements | ROUTING PROXY Component that permits clients to control relaying. |
| New Constraints | Every representation transfer must be justified by a corresponding edge in the web of trust. |
| Induced Property | *Multilateral Extensibility:* Can compose trusted invocations without requiring mutual trust. |

# Arquiteturas para Sistemas Distribuídos – Variações REST


- REST-D: REST com Delegação
- Objetivo:
  - Garantir durabilidade e consistência na execução simultânea de duas operações de write

# Arquiteturas para Sistemas Distribuídos – Variações REST


- REST-D: REST com Delegação




| | |
|---|---|
| Goal | Refer to a pairwise distributed read/write resource reliably. |
| New Elements | MUTEXLOCK Component ensures only one client at a time has write access to the origin server. |
| New Constraints | Lock must be acquired before attempting write.<br>Current state of the resource must be read before attempting write. |
| Induced Property | *ACID (Pairwise) Simultaneous Agreement:*<br>Clients can modify centralized resources within $3d$ — but only in the absence of contention. |

# Arquiteturas para Sistemas Distribuídos – Variações REST

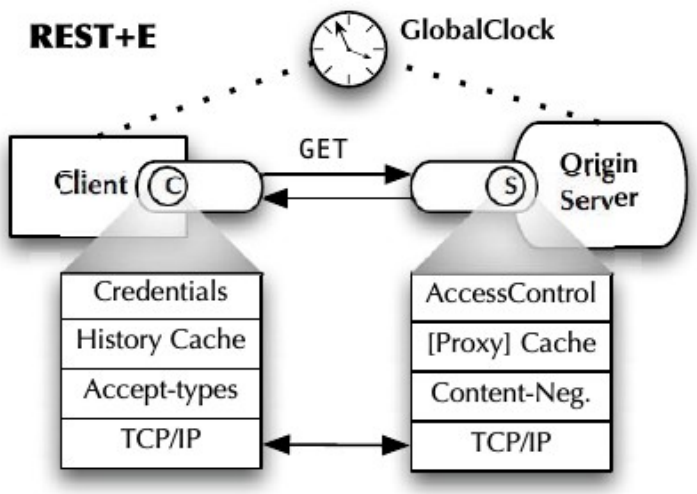

- Estilos livres de consenso:
  - Não-limitados ao 'now horizon'
    - Informações podem estar obsoletas
  - Não-limitados ao uso de agências confiáveis
    - Não existem mais 'fatos' e sim 'opiniões'

# Arquiteturas para Sistemas Distribuídos – Variações REST


- REST+E: REST with Estimates
  - Melhora a precisão dos valores através da extensão do REST para suportar a ausência do limite de transmissão de mensagens
- Unindo as estratégias obtem-se estilos adequados a diferentes cenários:
  - Centralizado: ARREST
  - Distribuído: ARREST+D
  - Estimado: ARREST+E
  - Descentralizado: ARRESTED

# Arquiteturas para Sistemas Distribuídos – Variações REST

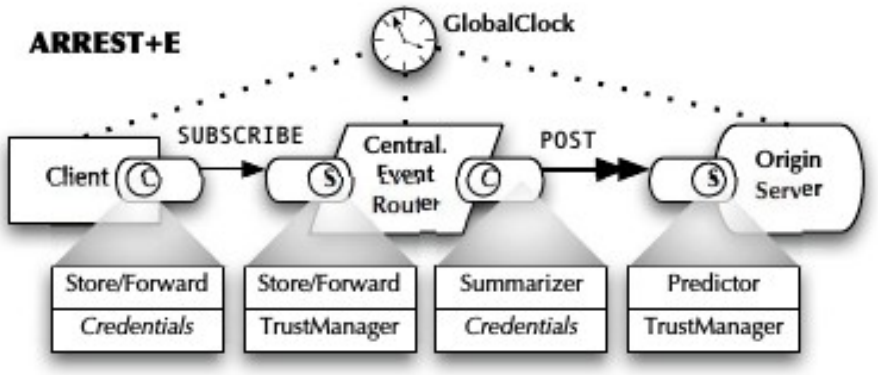

- REST+E: REST with Estimates




| | |
|---|---|
| Goal | Refer to a read-only centralized resource beyond its now horizon |
| [Old] Elements | [TCP/IP], [CACHE], [ACCESSCONTROL], [CONTENTNEGOTIATION] |
| New Constraints | Inertia assumes that the most recent representation is still valid, until cache revalidation fails. |
| Induced Property | *Approximate agreement:* The local proxy should be in simultaneous agreement *P%* of the time |

# Arquiteturas para Sistemas Distribuídos – Variações REST

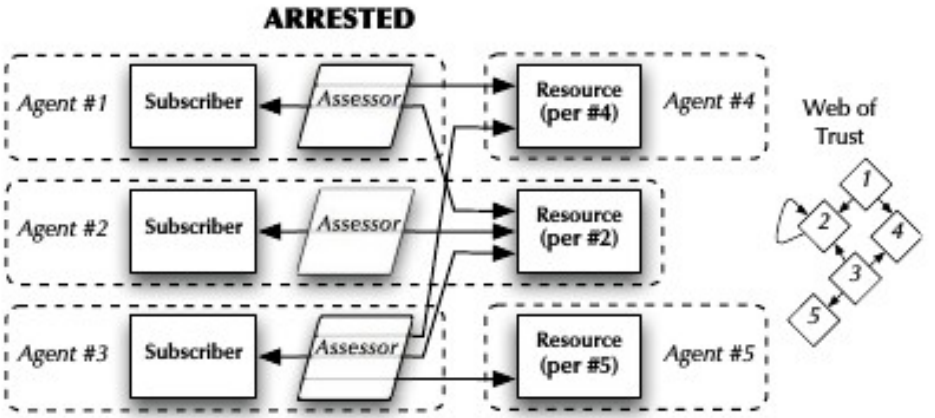
- ARREST+E:




| Goal | New Elements |
|---|---|
| Refer to a read/write resource connected by a faulty network beyond its now horizon. | STOREANDFORWARD Connector that adds end-to-end retransmission & acknowledgement. |
| | PREDICTOR Connector for encapsulating Turing-complete prediction functions. |
| | TRUSTMANAGER Connector that drops notifications from untrusted sources. |
| | SUMMARIZER Connector to resample queued events at lower frequency, reduce size. |
| Decentralize control of a shared resource across disjoint 'now horizons' | ASSESSOR Component that manages the risk of inter-agency disagreement over the 'true' value using a panel of opinions. |

# Arquiteturas para Sistemas Distribuídos – Variações REST


- ARRESTED:




| New Constraints | Induced Property |
|---|---|
| End-to-end retransmission of messages and acknowledgements. | *Best-Effort data transfer*: Cope with message loss. |
| Predict probable current state from past data (where possible). | *Approximate estimates*: Cope with message delay. |
| Ensure that all reachable endpoints are also trusted. | *Self-Centered*: Cope with dynamic participation. |
| Prohibit transmission of already-expired data. Eliminate buffering. | *Efficient data transfer*: Cope with net congestion. |
| Eliminate reliable references to remote resources; only contingent assessments remain. | *Consensus-freedom:* avoid presuming feasibility of consensus. |

# Arquiteturas para Sistemas Distribuídos – Tiles


- "An Architectural Style for Solving Computationally Intensive Problems on Large Networks", Brun & Medvidovic, 2007
- Objetivo:
  - Propor um estilo arquitetural para sistemas distribuídos de larga escala, baseado no estudo matemático sobre crescimento de cristais
  - Propriedades:
    - Discrição: nós na rede não conseguem "aprender" o algoritmo sendo executado ou dados utilizados
    - Tolerância a falhas
    - Escalabilidade na comunicação entre os nós

# Arquiteturas para Sistemas Distribuídos – Tiles

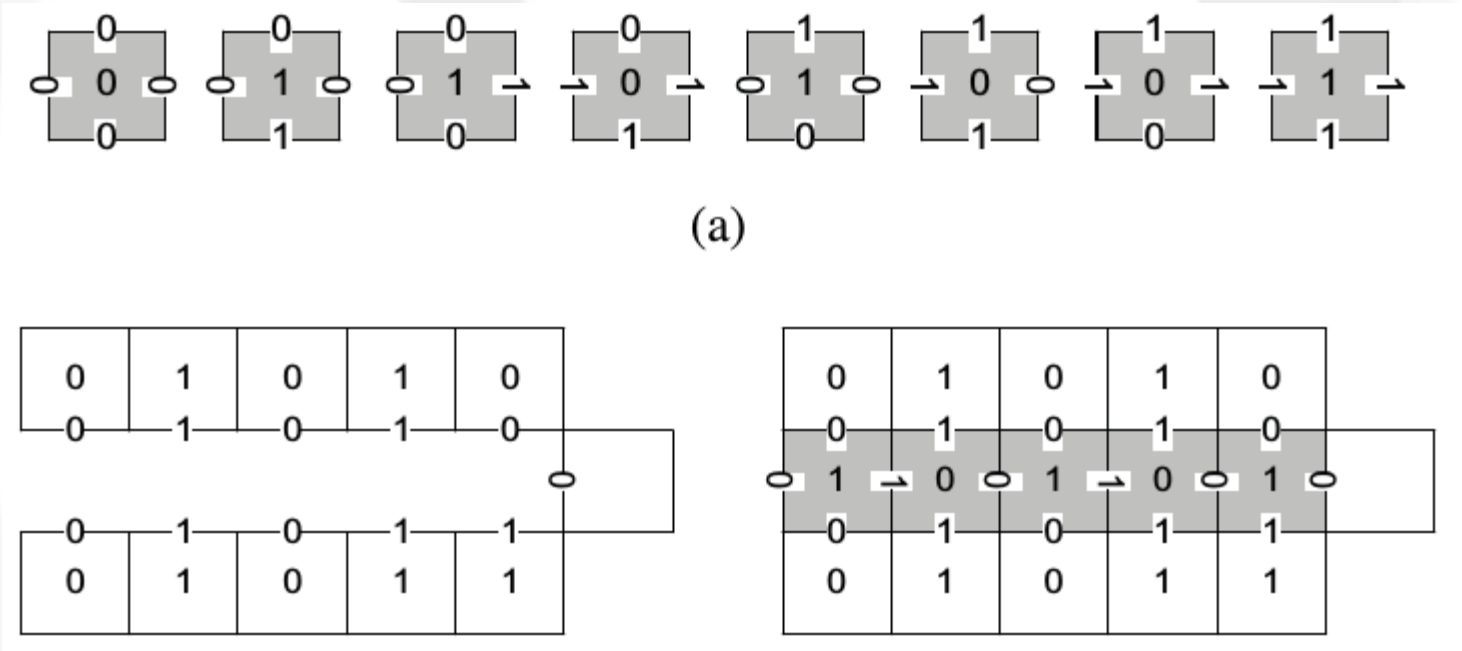

- Exemplo de uso:
  - Uma agência de espionagem deseja quebrar um código RSA enviado pelo inimigo
  - A agência deseja utilizar uma rede em larga escala como a Internet para fatorar a chave pública do inimigo
  - Ninguém deve poder conhecer os fatores da chave ou a chave sendo fatorada
  - Usa-se a Internet para fatorar de forma não-determinística ou por força bruta
- Baseia-se no Tile Assemble Model
  - É turing universal

# Arquiteturas para Sistemas Distribuídos – Tiles


- Tile Assembly Model:
  - Componentes são 'ladrilhos' quadrados com rótulos especiais em cada um dos seus quatro lados
  - Tiles podem permanecer unidos sob certas condições quando há um casamento dos rótulos dos lados confinados
  - É possível codificar dados de entradas como tiles
  - É possível projetar conjuntos de tiles para computar funções

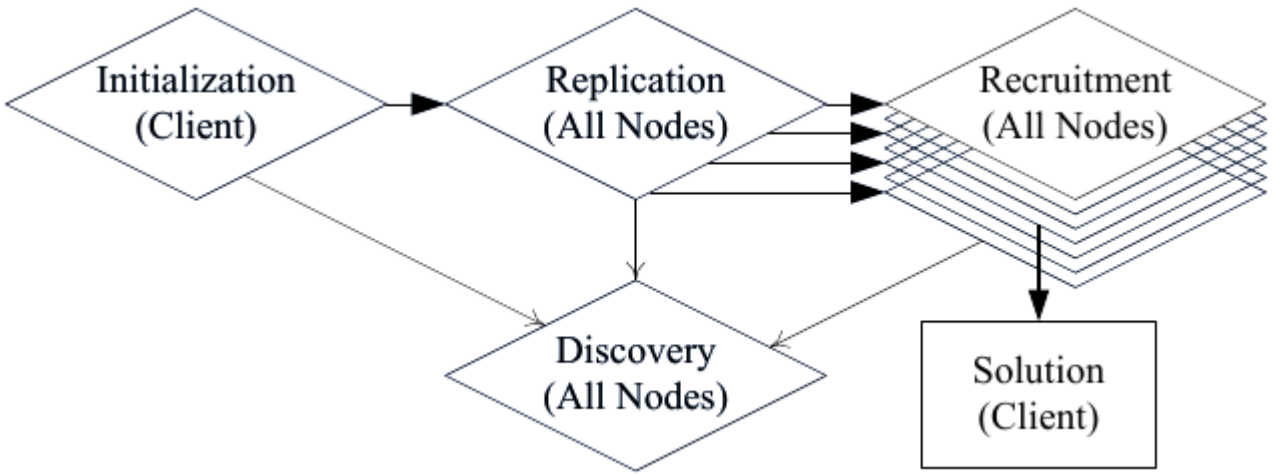
# Arquiteturas para Sistemas Distribuídos – Tiles

- Exemplo: tile system para adição



# Arquiteturas para Sistemas Distribuídos – Tiles

- Overview da operação no Estilo Arquitetural Tiles




# Pós-Graduação em Computação Distribuída e Ubíqua

## INF628 - Engenharia de Software para Sistemas Distribuídos
## Estilos Arquiteturais para Sistemas Distribuídos

Sandro S. Andrade
sandroandrade@ifba.edu.br