

GBC074 – Sistemas Distribuídos

Replicação
e
Tolerância a Falhas

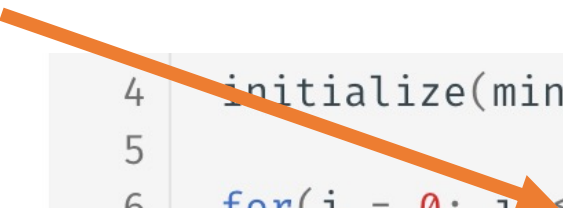
Introdução

- Classificação de problemas:
 - **Falta** (*defect, fault*, falha):
 - Erro no desenvolvimento do Sistema: *bug*, defeito de fabricação
 - Existe mesmo se for raramente ativada e mesmo se seus efeitos nunca forem percebidos

```
1  char minha_string[11];
2  int i;
3
4  initialize(minha_string);
5
6  for(i = 0; i ≤ 10; i++){
7      if (minha_string[i] == '.')
8          break;
9
10     minha_string[i] = 'a';
11 }
12
13 minha_string[i] = '\0';
```

Introdução

- Classificação de problemas:
 - **Erro** (*error*):
 - Manifestação da falta
 - No exemplo, seria quando a iteração passasse do ponto correto por causa do \leq
 - Pode passar despercebido, mas ainda assim é um erro



```
4 initialize(minha_string);
5
6 for(i = 0; i ≤ 10; i++){
7     if (minha_string[i] == '.')
8         break;
9
10    minha_string[i] = 'a';
11 }
12
13 minha_string[i] = '\0';
```

Introdução

- Classificação de problemas:
 - **Falha** (*failure*, defeito):
 - Erro percebido pelo usuário
 - No mesmo exemplo, isso seria um *stack overflow*
 - Falha pode afetar componentes dependentes
 - Esta cadeia pode levar cenários catastróficos
- Importante identificar causas (***root cause analysis***)
 - Forma de compartilhar conhecimento
 - Transparência
 - Evitar novas instâncias da mesma falha ou similares

Introdução

- Falha pode afetar componentes dependentes
- Esta cadeia pode levar cenários catastróficos
 - Exemplos:

” The Explosion of the Ariane 5

On June 4, 1996 an unmanned Ariane 5 rocket launched by the European Space Agency exploded just forty seconds after its lift-off [...] after a decade of development costing \$7B. The destroyed rocket and its cargo were valued at \$500M. [...] the failure was a software error [...] a 64 bit floating point number [...] was converted to a 16 bit signed integer. The number was larger than 32,767, the largest integer storeable in a 16 bit signed integer, and thus the conversion failed.

” Quote

The plane's electrical generators fall into a failsafe mode if kept continuously powered on for 248 days. The 787 has four such main generator-control units that, if powered on at the same time, could fail simultaneously and cause a complete electrical shutdown.

Segundo as "más línguas", o problema é que acontece um *overflow* em um contador de tempo

Classificação de faltas

- **Quebra (*crash*):**

- Componente para de funcionar, irreversivelmente
- Qualquer comunicação com o mesmo é interrompida
- Pode dar bons indicativos do defeito aos outros componentes.

- **Sistemas *fail-stop*:**

- Parada forçada quando percebem um falha
- Outros componentes sabem (são informados) da falha

- **Sistemas *fail-recover*:**

- Processo falho pode ser forçado a recuperar o estado em que estava logo antes do problema se manifestar
- Necessidade de capacidade de armazenar estado

Classificação de faltas

- **Omissão (*omission failure*):**
 - Componente deixa de executar alguma ação.
 - Exemplo:
 - Requisição recebida por um servidor não é processada
 - Disco não armazena os dados no meio magnético
 - Mensagem não é transmitida
 - Difícil de ser identificado

Classificação de faltas

- **Temporização**

- Violação de limites de tempo
- Exemplo:
 - Se o meio de comunicação se recusou a entregar uma mensagem que deveria ser entregue dentro de 3ms, houve falha de omissão.
 - Se a mensagem é retransmitida até que tenha sua entrega confirmada, mas a mesma é entregue com 5ms, temos uma falha de temporização

Classificação de faltas

- **Arbitrário (bizantino):**

- Qualquer comportamento pode acontecer
- Exemplos:
 - Mensagem pode ser modificada
 - Servidor pode reiniciar-se constantemente
 - Dados podem ser apagados
 - Acesso pode ser dado a quem não é devido
- Causadas por faltas no software / hardware, hackers e vírus

- **Hierarquia**

- Os tipos de faltas podem ser hierarquizados
 - Fail-stop \subset Quebra \subset Omissão \subset Temporização \subset Arbitrária
- Neste caso, uma quebra é apenas uma omissão por tempo infinito

Lidando com faltas

- **Prevenção**

- Por meio de técnicas bem estabelecidas de engenharia
- Uso de linguagens de programação fortemente tipadas
- Análise estática, especificação formal, teste e prova
 - [TLA+](#) e [Promela](#),

- **Remoção**

- Tradução de especificações formais para código é um passo complexo
- Testes e manutenção do sistema permitem a **remoção de faltas**

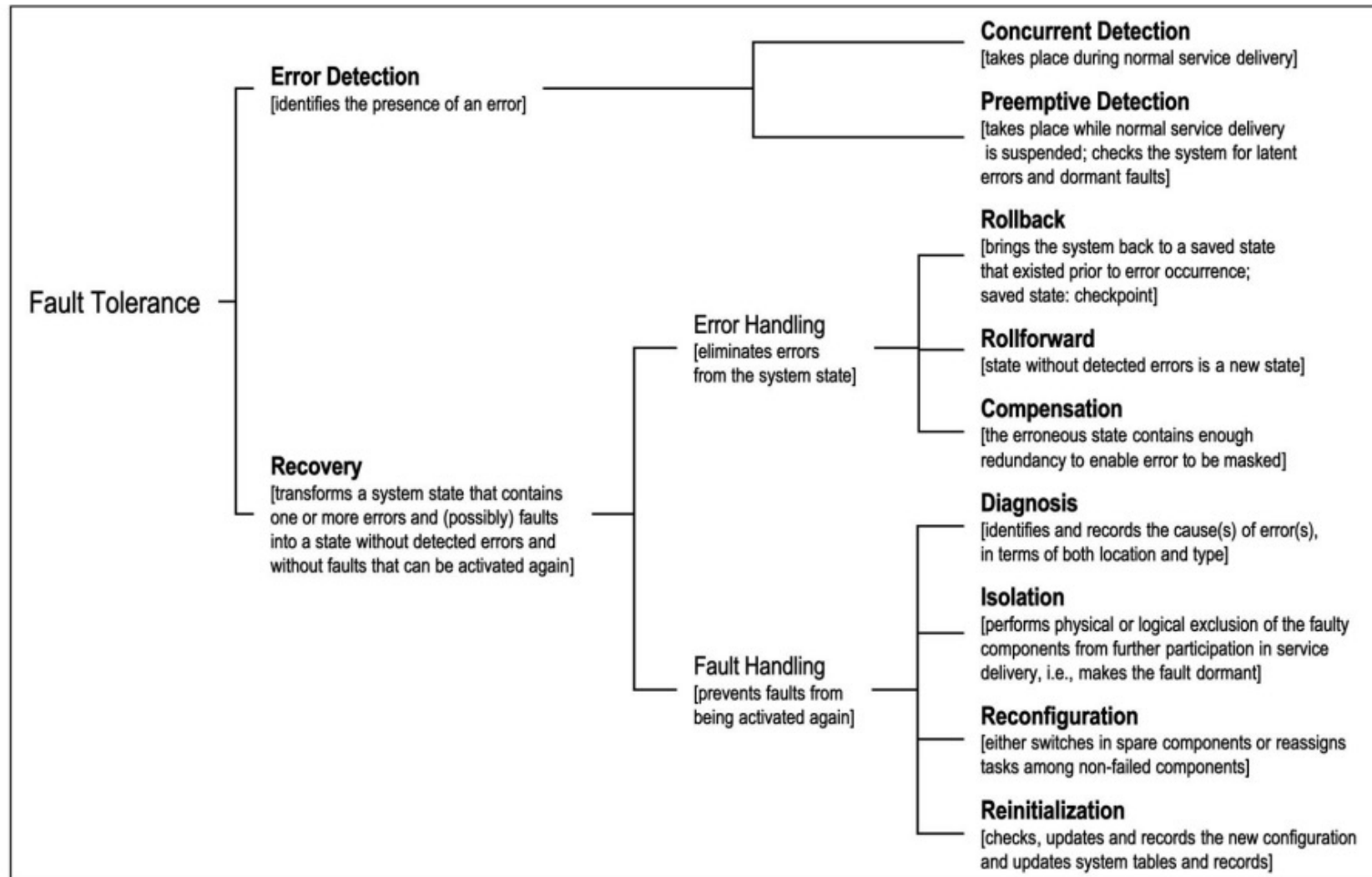
- **Tolerância**

- Mesmo se faltas ainda estiverem presentes, seus efeitos não devem ser percebidos como falhas
- Necessário detectar e se recuperar de erros
- Exemplo:
 - Sistema de arquivos que mantenha um *journal*, como o ext3

Lidando com falhas

- **Redundância como ferramenta de melhoria**
 - Para prevenir faltas, redundância de tempo para refinar projetos
 - Para remover faltas, redundância de tempo e recursos
Para lidar com erros, redundância de código
 - Remover os **pontos únicos de falha**
(SPOF, *Single Point of Failure*)
- No caso de um sistema distribuído:
 - Redundância = cópias ou réplicas
 - Quando um processo apresenta um defeito, outros podem continuar executando o serviço

Lidando com faltas

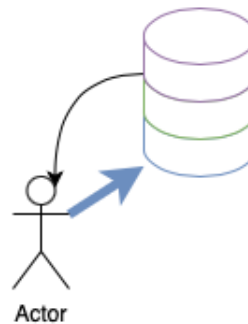
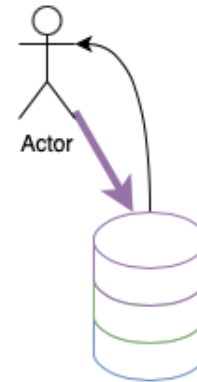


Replicação

- Ideia simples:
 - Criar cópias de um componente (serviço) para garantir disponibilidade
- Implementação complexa!
 - Diversas possibilidades
 - Diferentes níveis de “consistência” e corretude
 - Cada um com prós e contras

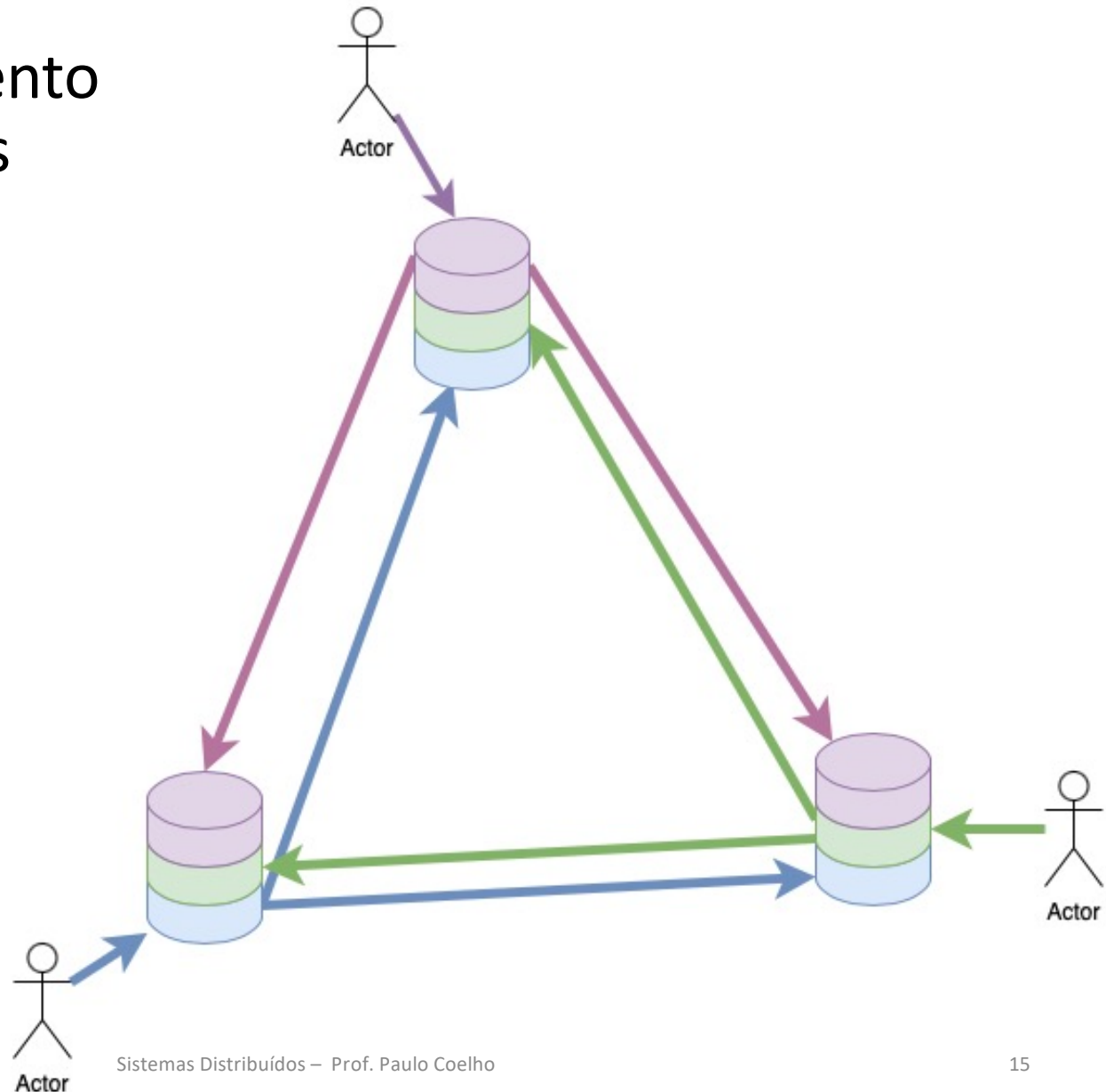
Replicação - Multi-escritores

- Clientes enviam modificações (escritas) e recuperações (leitura) para qualquer réplica



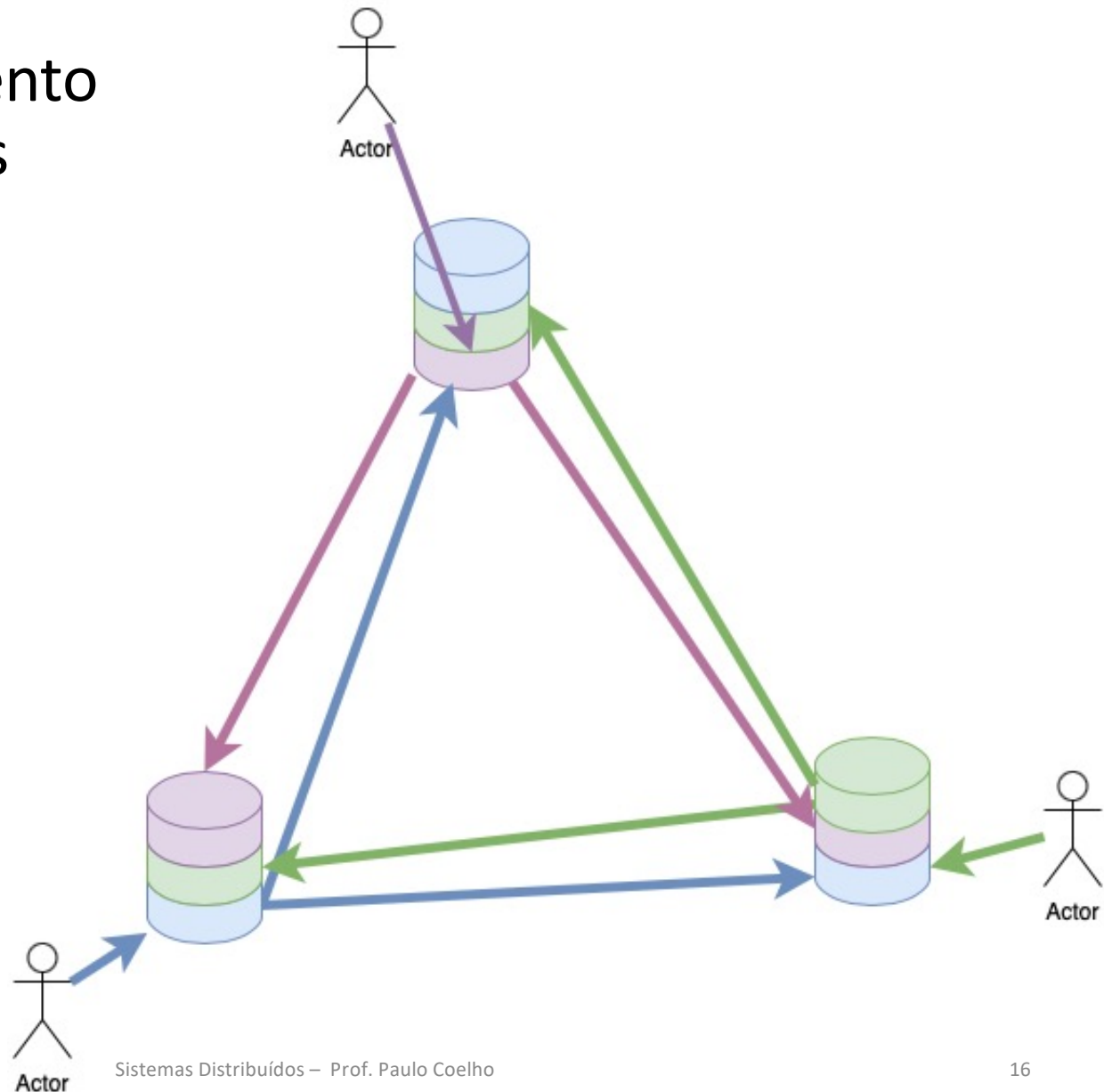
Replicação - Multi-escritores

- Encaminhamento de mensagens
 - UDP
 - TCP
 - Ordem?



Replicação - Multi-escritores

- Encaminhamento de mensagens
 - UDP
 - TCP
 - Ordem?



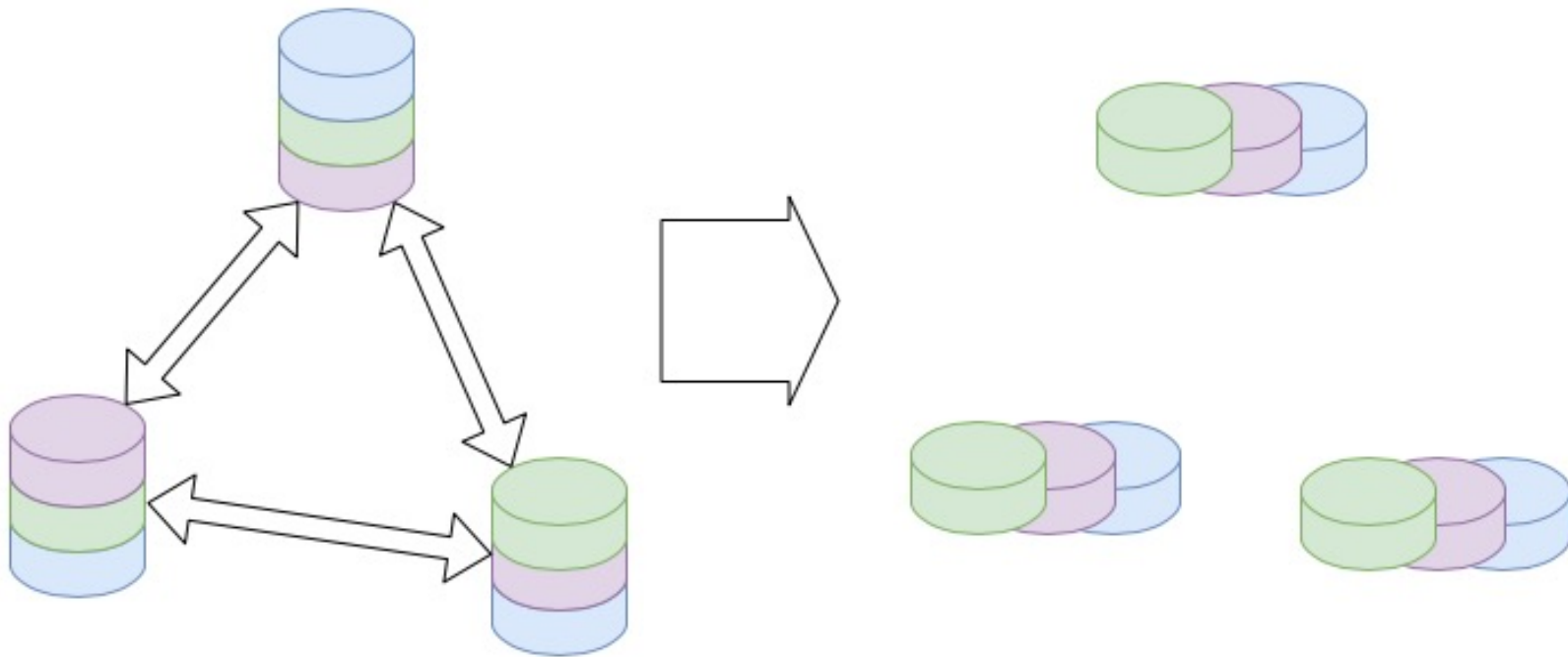
Replicação - Multi-escritores

- Anti-entropia
 - Combinação de [gossiping](#) e usando [relógios vetoriais](#) para concordar nas versões de dados conflitantes
 - Em algum momento replicas estarão consistentes: ***eventual consistency***
 - Exemplos: Cassandra, Redis e Dynamo

Replicação - Multi-escritores

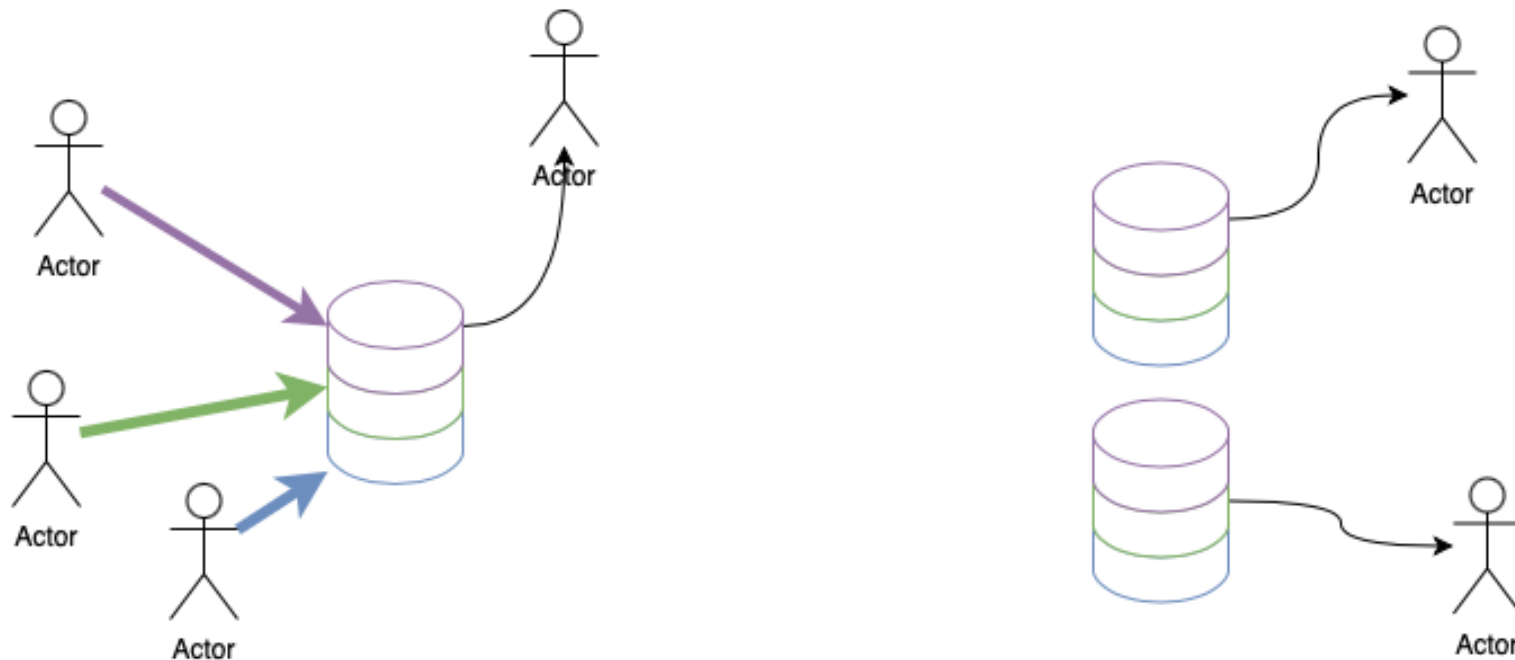
- Anti-entropia

- Estado alcançado não necessariamente faz sentido
 - Pode corresponder, por exemplo, a uma ordenação errada dos comandos emitidos por um dado cliente
- Técnica não pode ser aplicada em todas as situações



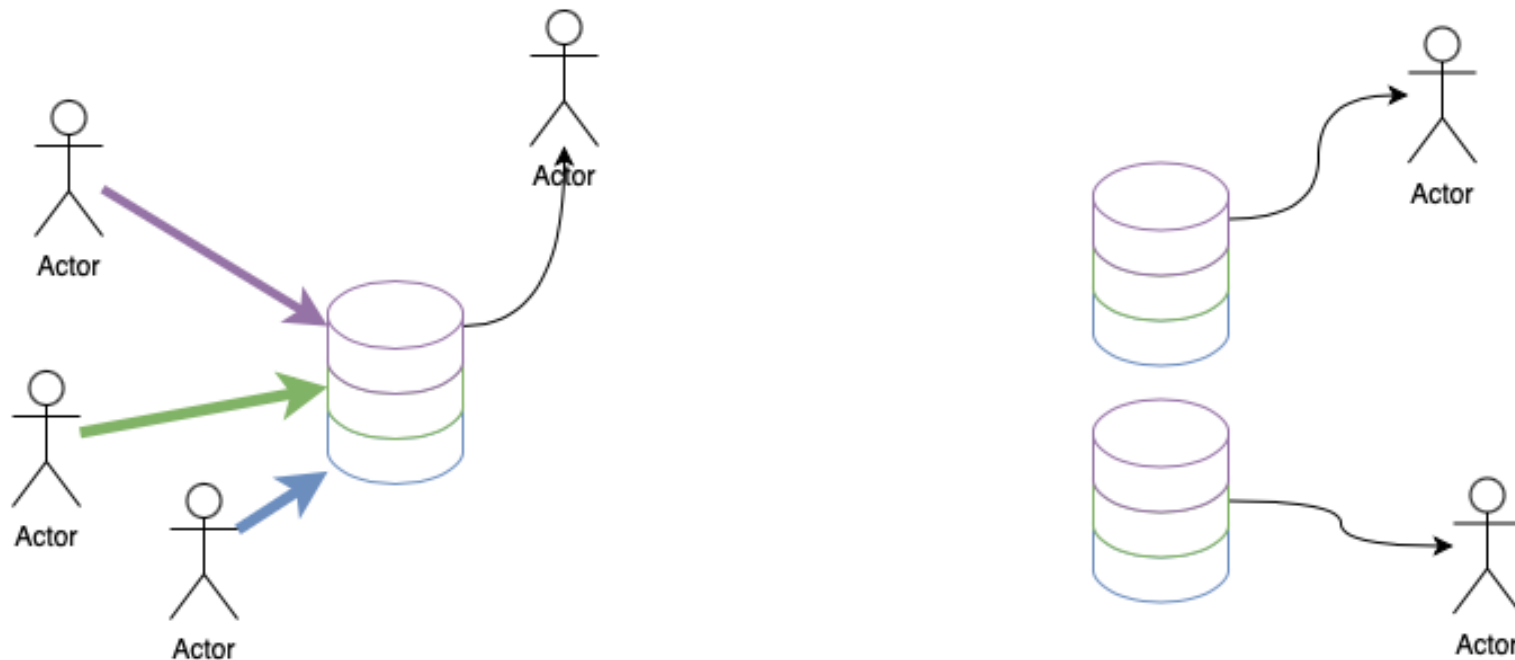
Replicação

- Único-escriptor:
 - Operações de escrita direcionadas para uma única réplica
 - Operações de leitura podem ser direcionadas à mesma réplica ou a quaisquer das replicas
 - Depende da **consistência** dos clientes



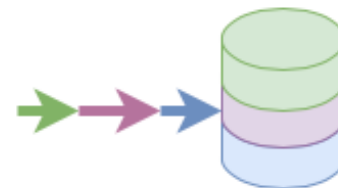
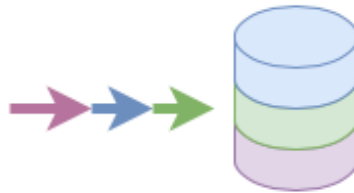
Replicação - Único-escriptor

- Primário/Cópias (*Primary/backup*)
 - **Problema até agora:** ordenação das operações
 - Cada processo pode receber as mensagens em qualquer ordem
 - Cada uma tem uma fila de mensagens independente



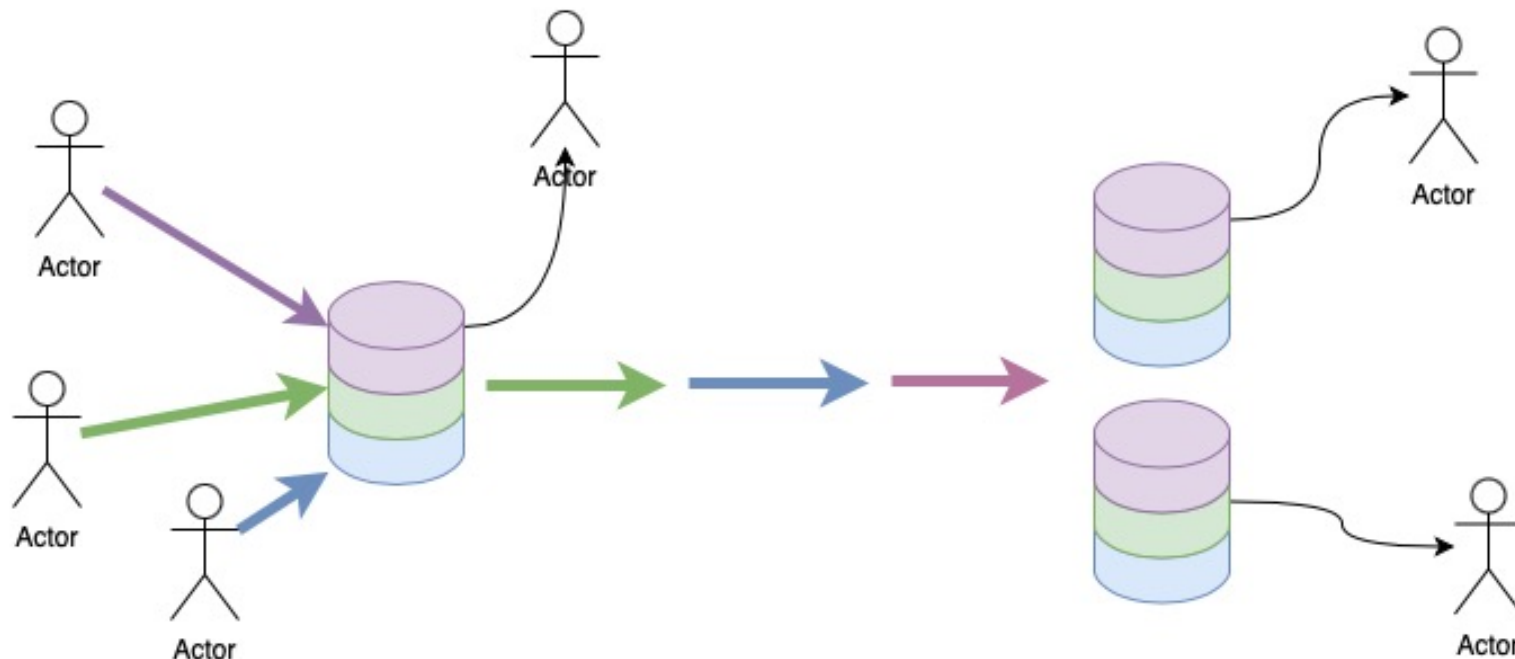
Replicação - Único-escriptor

- Primário/Cópias (*Primary/backup*)
 - **Problema até agora:** ordenação das operações
 - Cada processo pode receber as mensagens em qualquer ordem
 - Cada uma tem uma fila de mensagens independente
 - Réplicas podem chegar a estados distintos



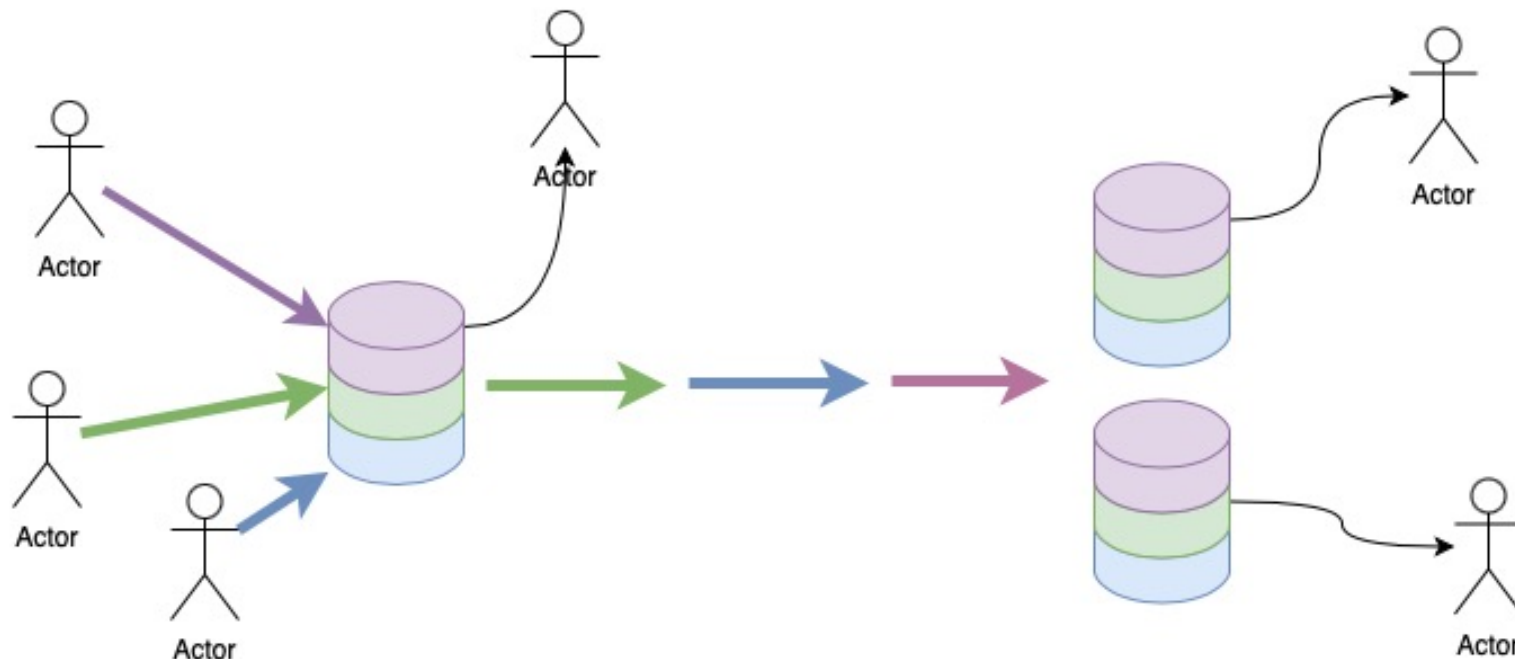
Replicação - Único-escriptor

- Primário/Cópias (*Primary/backup*)
 - E se tivéssemos uma fila única?
 - Apenas o primário é responsável por lidar com clients
 - Primário informa cópias das modificações de estado
 - Ordenação de operações faz sentido:
 - Corresponde à ordem de entrega das mensagens



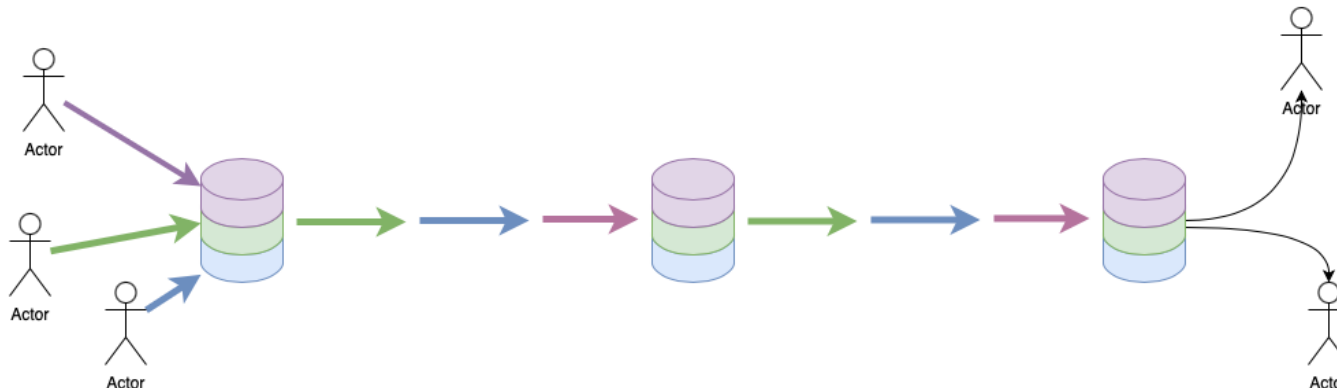
Replicação - Único-escriptor

- Primário/Cópias (*Primary/backup*)
 - **Operações x Estado**
 - O que o primário deve passar para as cópias?
 - **Cópias desatualizadas**
 - Operações de leituras podem ser feitas nas cópias?



Replicação - Único-escriptor

- Replicação em cadeia (*Chain replication*)
 - generalização de primário/
 - processos se organizam em uma sequência
 - **Atualizações** sempre direcionadas ao primário (**cabeça**)
 - **Leituras**
 - Se necessidade de dados escritos mais recentemente, também direcionadas à **cabeça**
 - Caso contrário, podem ser direcionadas aos processos na **cauda**
 - Diminui a carga de trabalho na cabeça
 - Quanto mais relaxado for a exigência de "frescor" dos dados, mais para o fim da cauda a requisição pode ser enviada

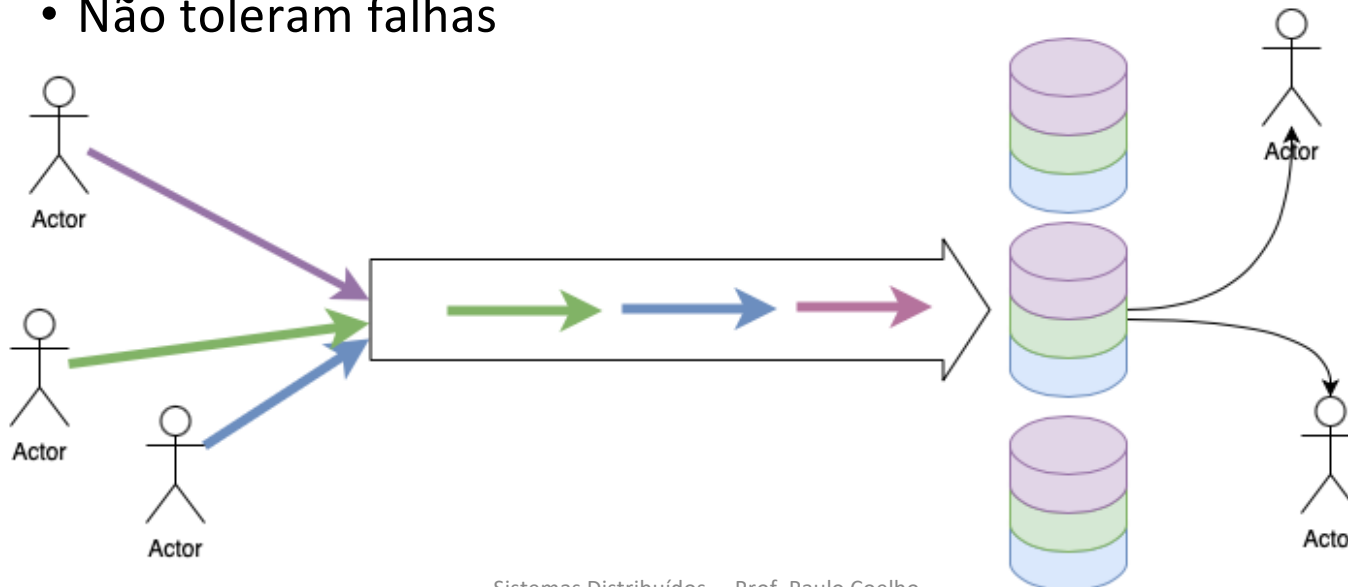


Replicação - Único-escriptor

- Identificação de falhas é crítico para bordagens baseadas em um primário
- Primeiro desafio está em identificar a falha
 - Tarefa não trivial ou impossível em algumas situações
- Identificação perfeita de falhas:
 - E as operações que já foram entregues para o primário mas que ainda não foram propagadas para as réplicas?
 - Se esta situação for inaceitável, replicação ativa pode ser usada

Replicação ativa

- Todas as **cópias executam todos os comandos**
- Todas aptas a continuar a executar o serviço a qualquer instante
- Exemplo - Replicação de máquinas de estados
 - Utiliza primitivas de comunicação em grupo
 - Primitivas vistas anteriormente não são funcionais
 - Não toleram falhas



Dependabilidade

- Sistema ter a propriedade de se poder depender do mesmo
- Componente C **depende de um componente C'** se a corretude do comportamento de C depende da corretude do componente C'
- Propriedade por ser dividida em outras propriedades mais "simples"
 - Disponibilidade (*Availability*) - Prontidão para uso
 - Confiabilidade/Fiabilidade (*Reliability*) - Continuidade do serviço
 - Segurança (*Safety*) - Tolerância a catástrofes
 - Integridade (*Integrity*) - Tolerância a modificações
 - Manutenibilidade (*Maintainability*) - Facilidade de reparo

Dependabilidade

- **Disponibilidade**

The term 'availability' means ensuring timely and reliable access to and use of information

- Na prática:

- percentagem de tempo que o sistema está disponível para uso

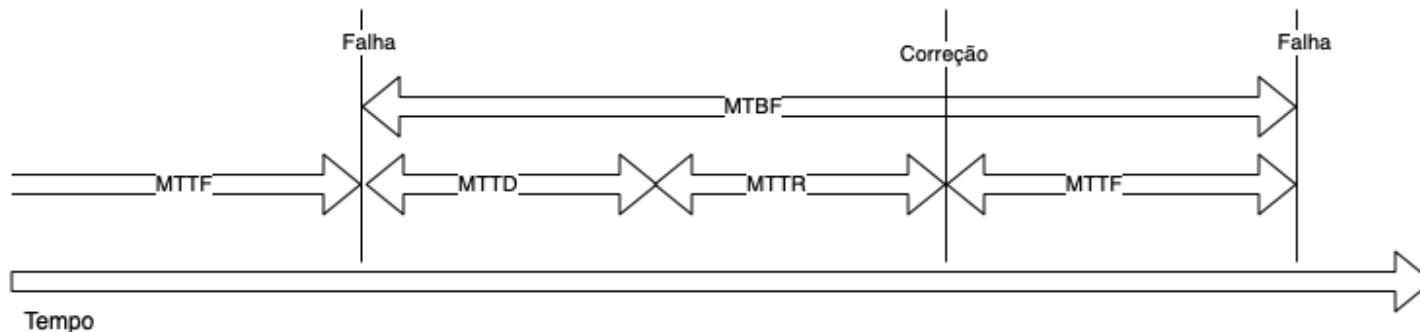
$$\text{Disponibilidade} = \frac{\text{Tempo de disponibilidade acordado} - \text{Tempo de indisponibilidade}}{\text{Tempo de disponibilidade acordado}}$$

Disponibilidade (%)	"noves"	downtime anual	downtime mensal	downtime semanal
90	1	36,5 dias	72 horas	16,8 horas
99	2	3,65 dias	7,2 horas	1,68 horas
99,9	3	8,76 horas	43,8 minutos	10.1 minutos
99,99	4	52,56 minutos	4,38 minutos	1,01 minutos
99,999	5	5,26 minutos	25,9 segundos	6,06 segundos
99,9999	6	31,5 segundos	2,59 segundos	0,605 segundos

Dependabilidade

- **Confiabilidade:** Mede a prontidão para uso por um período, definida por quatro métricas:

- Tempo médio para falha, MTTF (*mean time to failure*)
 - Expectativa de quanto tempo resta até que o sistema apresente o próximo defeito (relevante);
- Tempo médio para diagnóstico, MTDD (*mean time to diagnose*)
 - Expectativa de quanto tempo leva perceber que o sistema apresentou um defeito e iniciar sua correção
- Tempo médio para reparo, MTTR (*mean time to repair*)
 - Expectativa de quanto tempo leva para corrigir o sistema e retorná-lo ao estado funcional, uma vez que o defeito foi percebido
- Tempo médio entre falhas, MTBF (*mean time between failures*)
 - Expectativa de quanto tempo transcorre entre falhas



Dependabilidade

- **Segurança**

- Ausência de consequências catastróficas para usuários e/ou ambiente

- **Integridade**

- Ausência de alterações impróprias no Sistema
- Exemplo:
 - Capacidade de impedir acesso não-autorizado
 - possibilidade de corrupção de dados e capacidade de se recuperar

Dependabilidade

- **Manutenabilidade**
- Medida da expectativa do quão rápido um sistema é reparado uma vez identificado sua falha
- Uma manutenabilidade de 95% para 1 hora implica que com 95% de probabilidade chance o sistema voltará a estar funcional dentro de 1 hora.
- Intimamente ligada ao tempo médio de reparo (MTTR)

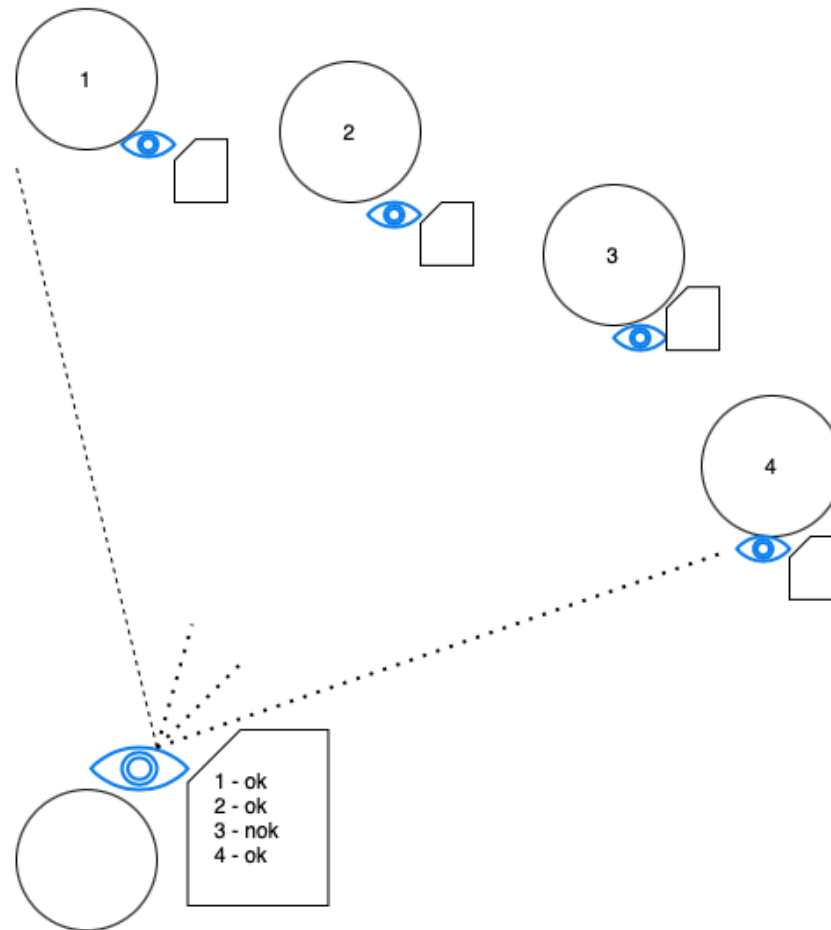
Detectores (não confiáveis) de falha

- Necessidade de perceber quando problemas acontecem
 - Exemplo, componente não funcional pode necessitar manutenção:
 - Reinicialização, troca de disco ou uma fonte, etc
- **Detectores de Falhas:**
 - Introduzidos por Chandra e Toueg
 - Forma de encapsular a percepção do estado funcional dos outros processos

Detectores (não confiáveis) de falha

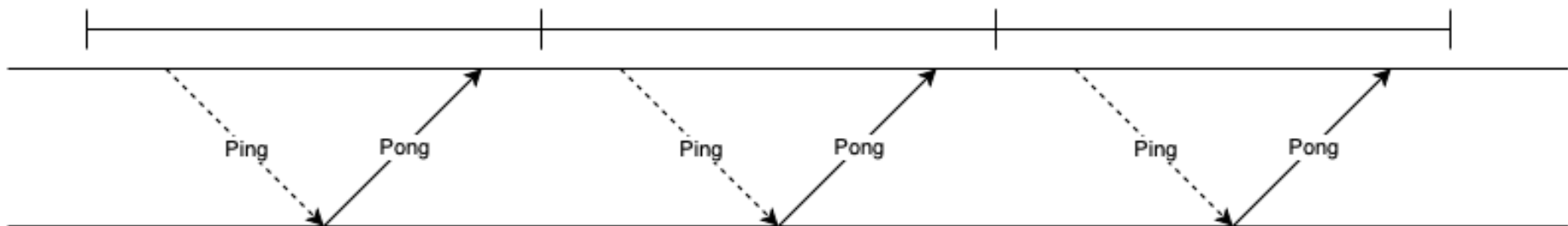
- **Detectores de Falhas**

- Podem ser vistos como **oráculos distribuídos**



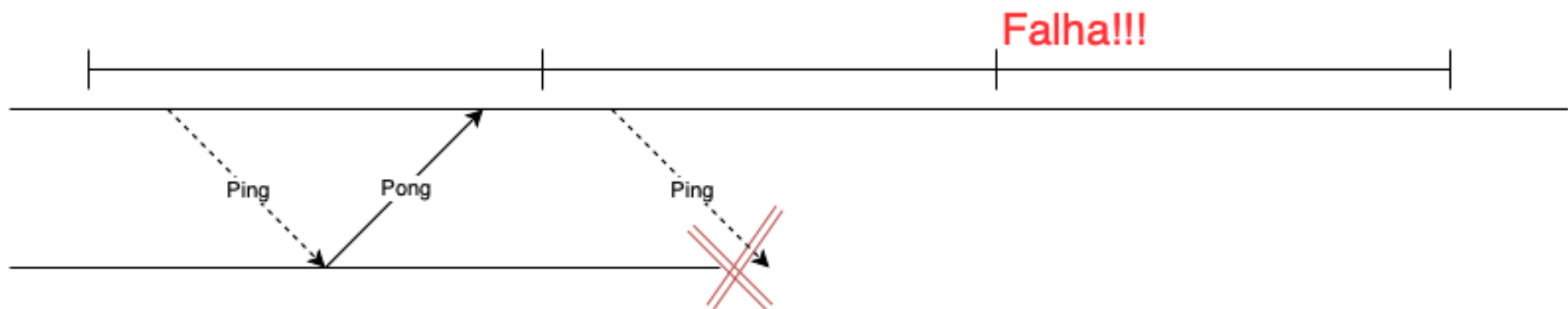
Detectores (não confiáveis) de falha

- Normalmente implementados por meio de trocas de mensagens de *heartbeat*
- Mensagens são esperadas em momentos específicos:
 - Recebimento **sugere** que remetente continua funcional



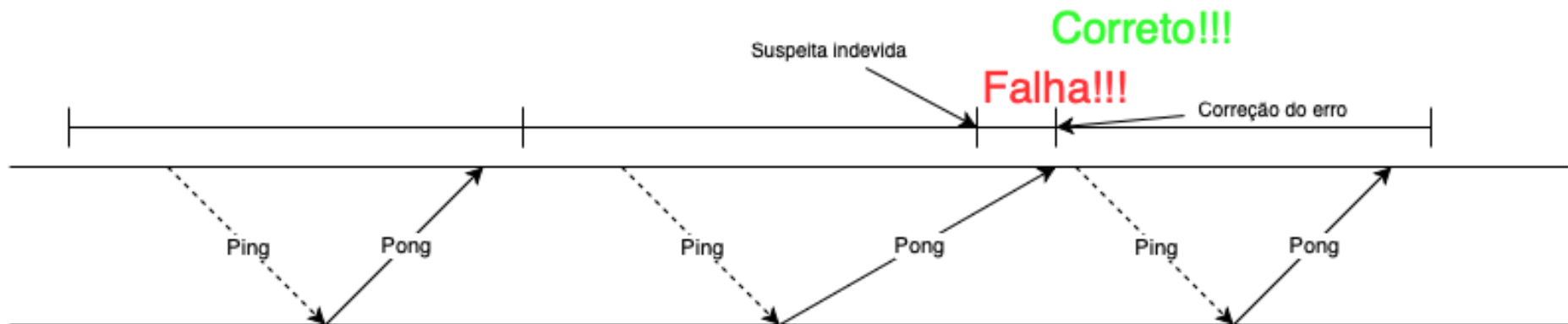
Detectores (não confiáveis) de falha

- Normalmente implementados por meio de trocas de mensagens de *heartbeat*
- Mensagens são esperadas em momentos específicos:
 - Recebimento **sugere** que remetente continua funcional



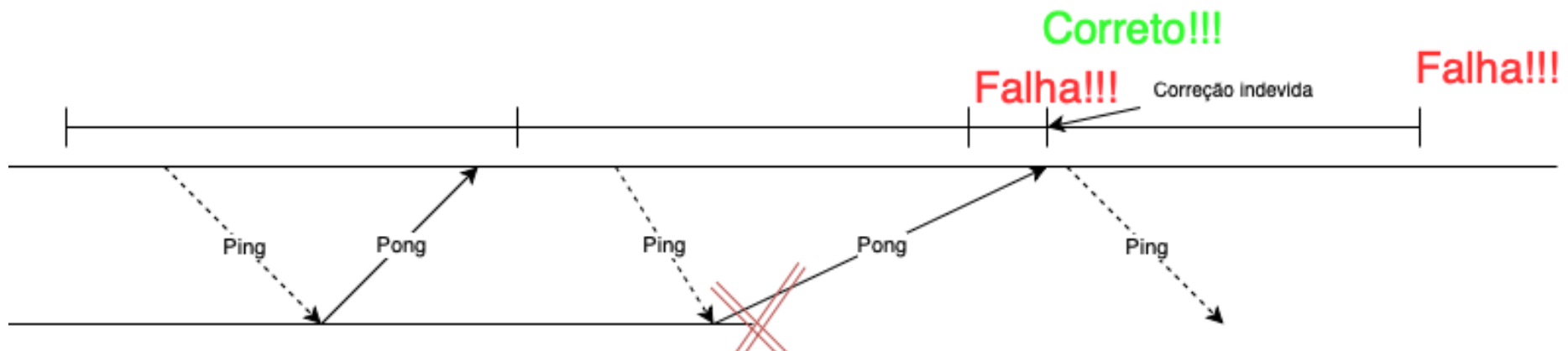
Detectores (não confiáveis) de falha

- Normalmente implementados por meio de trocas de mensagens de *heartbeat*
- Mensagens são esperadas em momentos específicos:
 - Recebimento **sugere** que remetente continua funcional



Detectores (não confiáveis) de falha

- Normalmente implementados por meio de trocas de mensagens de *heartbeat*
- Mensagens são esperadas em momentos específicos:
 - Recebimento **sugere** que remetente continua funcional



Detectores (não confiáveis) de falha

Propriedades

- Completude (*completeness*)
 - Capacidade de suspeitar de um processo defeituoso
- Acurácia (*accuracy*)
 - Capacidade de não suspeitar de um processo correto
- **Processo correto:**
 - não falha durante execução do protocolo

Detectores (não confiáveis) de falha

- **Classificação:** captura combinações de eventos apresentadas
- Completude Forte
 - A partir de algum instante, **todo processo falho** é suspeito permanentemente por **todos os processos corretos**
- Completude Fraca
 - A partir de algum instante, **todo processo falho** é suspeito permanentemente por **algum processo correto**
- Precisão Forte
 - **Todos os processos são suspeitos** somente **após** terem falhado
- Precisão Fraca
 - **Algum processo correto nunca** é suspeito de ter falhado
- Precisão Eventual Forte
 - A partir de algum instante, **todos os processos são suspeitos** somente **após** terem falhado
- Precisão Eventual Fraca
 - A partir de algum instante, **algum processo correto nunca** é suspeito

Detectores (não confiáveis) de falha

- Detector ideal
 - Completude Forte e Precisão Forte
 - Conhecido como **P** ou **Perfeito**
 - Só podem ser implementados em sistemas síncronos
 - ausência de uma mensagem significa que mensagem não será entregue
- Em ambientes **parcialmente síncronos**
 - É possível implementar detectores não confiáveis
 - Se os processos dispõem de **temporizadores** precisos
 - Detector pode considerar um **limite de tempo** para tentar determinar processos encontram-se defeituosos ou não, como nas figuras apresentadas acima.
 - Detectores podem voltar atrás em suas suspeitas
 - Informação provida já pode ser suficiente para que se resolva diversos problemas em computação distribuída

Detectores (não confiáveis) de falha

- Detector mais fraco com o qual se pode resolver o problema do consenso distribuído
 - Completude Fraca e Precisão Eventual Fraca ($\Diamond W$ - *Eventual Weak*)
- Transformando $\Diamond W$ em $\Diamond S$ (Completude Forte e Precisão Eventual Fraca)
 - Basta propagar as suspeitas de um processo para o outro
 - Eventualmente, todos os processos falhos serão suspeitos

Detectores (não confiáveis) de falha

- Eleitor de líderes Ω
 - Permite que os processos elejam, de forma não confiável, um dentre o conjunto de processos como seu líder
- Implementação com $\Diamond W$ (e portanto de $\Diamond S$)
 - Basta utilizar o $\Diamond S$ para recuperar a lista de processos não suspeitos atualmente e, destes, escolher um
 - Se $\Diamond S$ corrigir a informação e passar suspeitar do processo, escolha outro, em um rodízio
 - Em algum momento $\Diamond S$ deixará de suspeitar de algum processo correto
 - Em algum momento posterior este processo será escolhido pelo rodízio e será a partir então permanentemente o líder

Detectores (não confiáveis) de falha

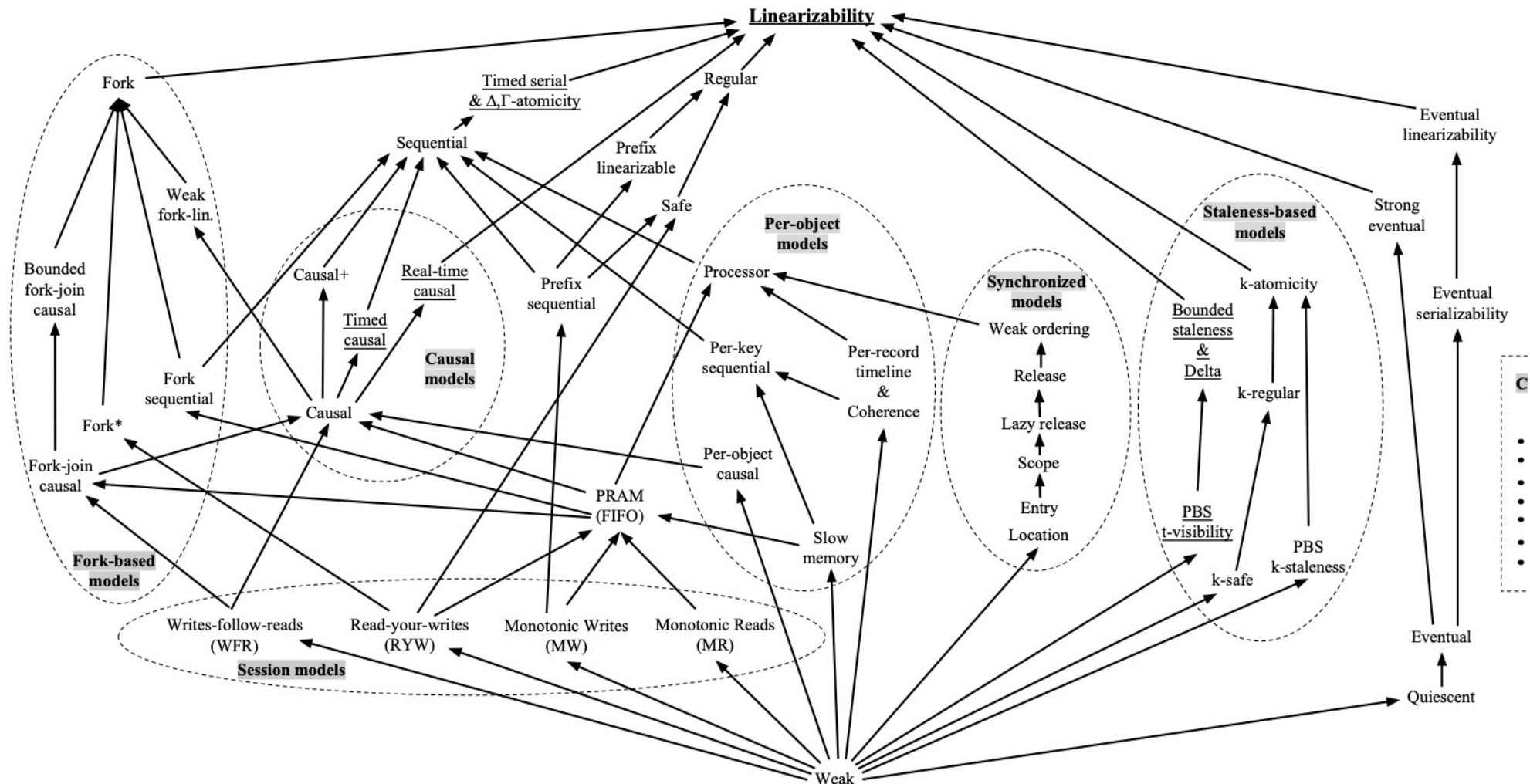
- Como se implementa detector $\Diamond W$?
 - **Pré-condição:** sistemas com **limite superior** de tempo para a transmissão de mensagens, **mesmo que este limite seja desconhecido**
 - Feito esquema de *pings*, esperando-se por um tempo t qualquer desde o envio do *ping* e até a resposta do *pong*
 - Se percebe que suspeitou indevidamente de outro processo, então dobra o tempo t
 - Em algum momento t será maior que o limite superior e detector não mais incorrerá em erros
- Protocolos que utilizam detectores $\Diamond S/\Diamond W$ ou o eleitor de líderes Ω são escritos de forma que o progresso só é garantido quando o detector para de cometer enganos
- Se o limite superior não existir
 - Protocolo não pode garantir terminação
 - Mas nenhum **resultado errado nunca é alcançado**
- Na prática
 - Sistemas assíncronos passam por períodos síncronos que permitem que os protocolos progridam e terminem... mas sem garantias

Consistência

- Modelos
 - Surgiram em dois mundos diferentes:
 - **Sistemas distribuídos**: diz respeito à execução de conjuntos de operações individuais sobre objetos simples, como uma variável
 - **Bancos de dados**: se refere às garantias dadas por transações
 - Apesar do ponto de partida disjunto, os dois pontos de vista podem ser unificados

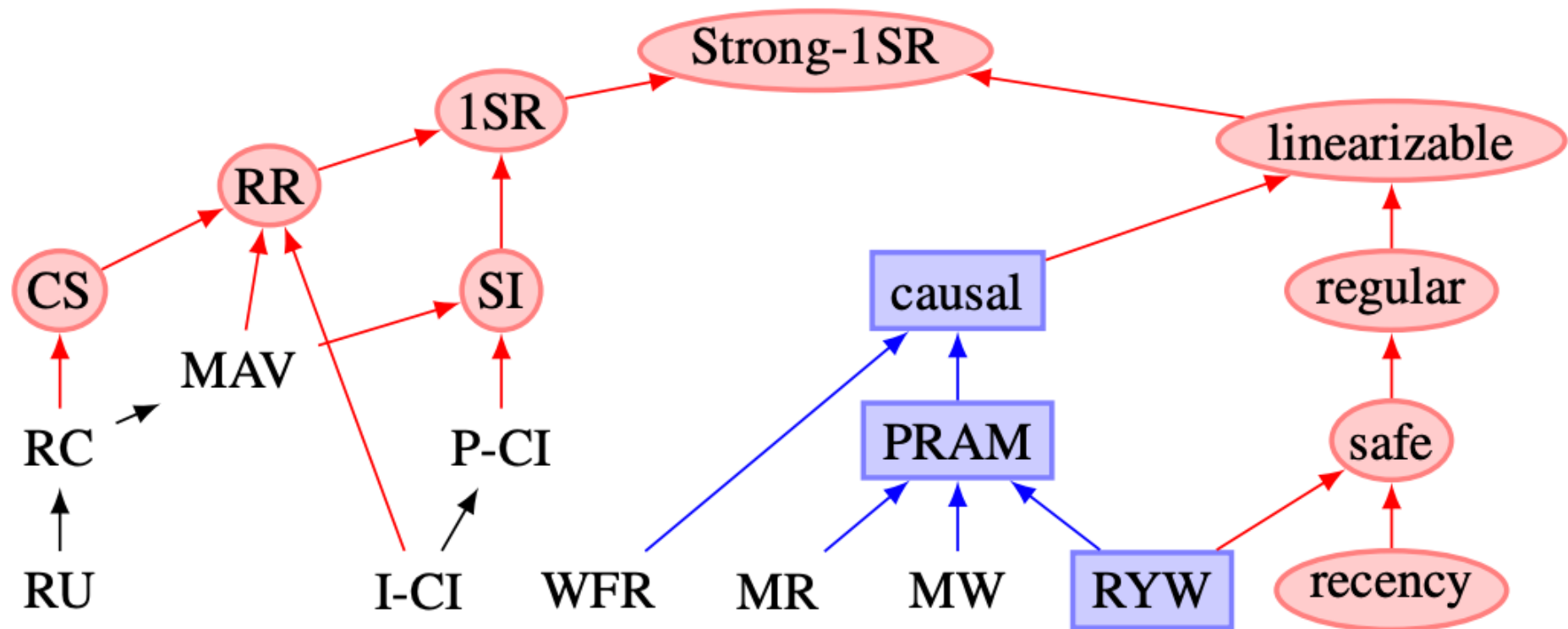
Consistência

• Modelos – Sistemas Distribuídos



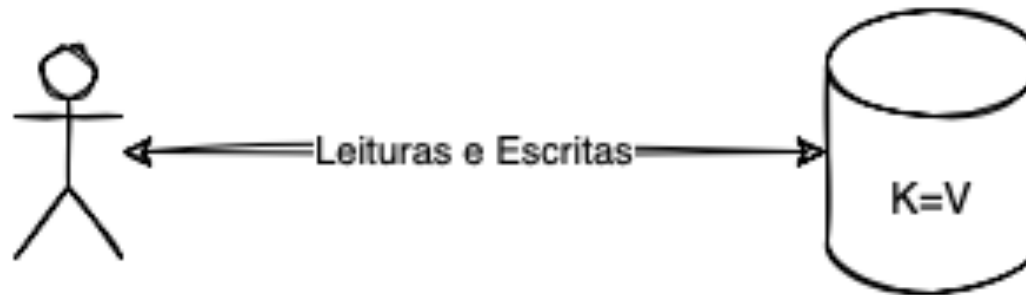
Consistência

- Modelos – Bancos de dados



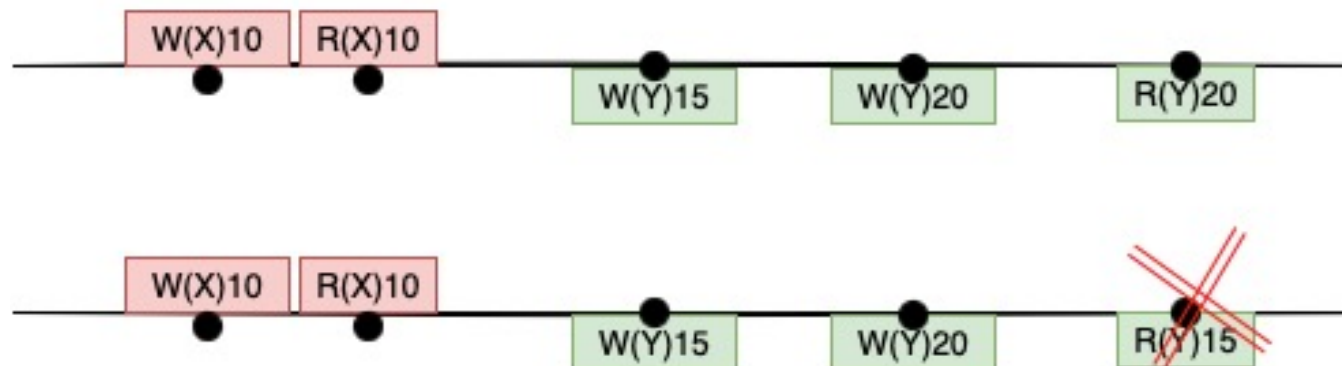
Consistência

- Modelos – simplificação
 - Considere um banco de dados para falar tanto sobre modelos para operações simples quanto para transações
 - Tabela do tipo chave/valor.
 - Podem ser associadas a comandos:
 - **X recebe 'João' e qual o valor de Y**
 - Várias partes:
 - **"{'Endereço':'Av 1, número 2', 'Profissão':'Computeiro'}"**
 - Simplificação poderosa (bancos NOSQL)



Consistência

- **Expectativa x Realidade**
- Processos *esperam* um certo comportamento quanto ao funcionamento deste banco
 - Exemplo, ao escrever um dados no banco, o cliente geralmente espera que as escritas aconteçam na ordem em que as disparou e que, ao ler, lhe seja retornado o "último" valor escrito



Consistência

- **Expectativa x Realidade**

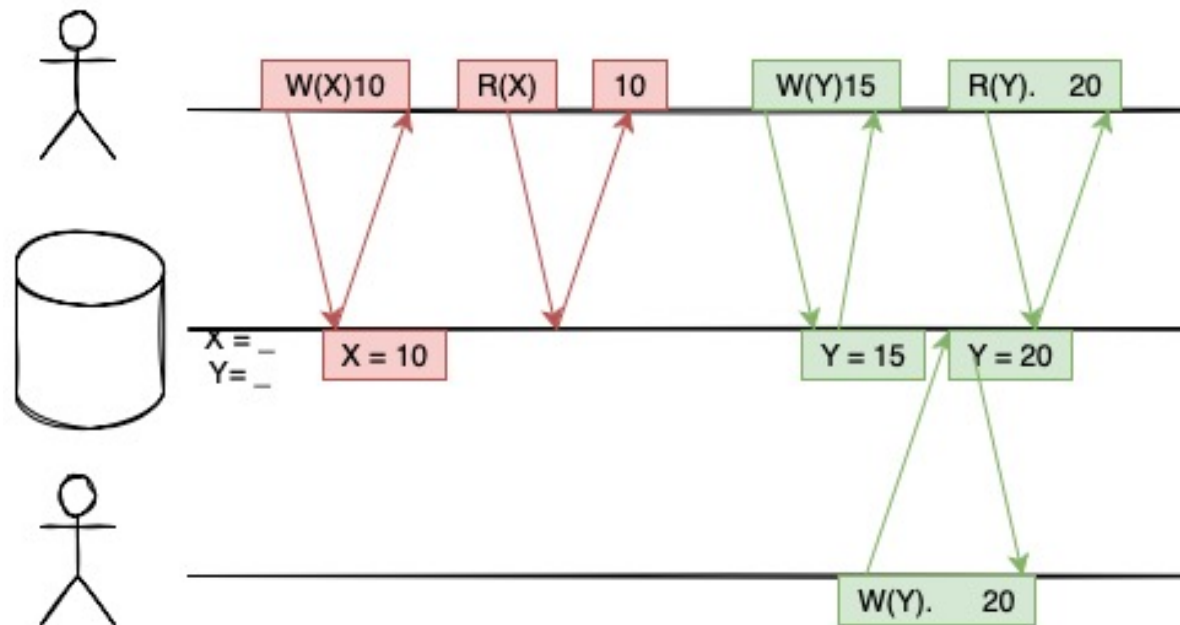
- Esta expectativa é o que denominamos um **modelo de consistência**

- **Níveis de Consistência**

- **Forte:** leituras sempre retornam versão mais recente do dado sendo lido
 - Propagação instantânea ou *locks* dos dados sendo manipulados enquanto a propagação acontece
- **Fraca:** leituras retornam algum dado escrito anteriormente
 - Qualquer coisa vale
- **Eventual:** se não houver novas escritas, a partir de algum momento leituras retornam a versão mais recente do dado
 - Propagação acontece no segundo plano

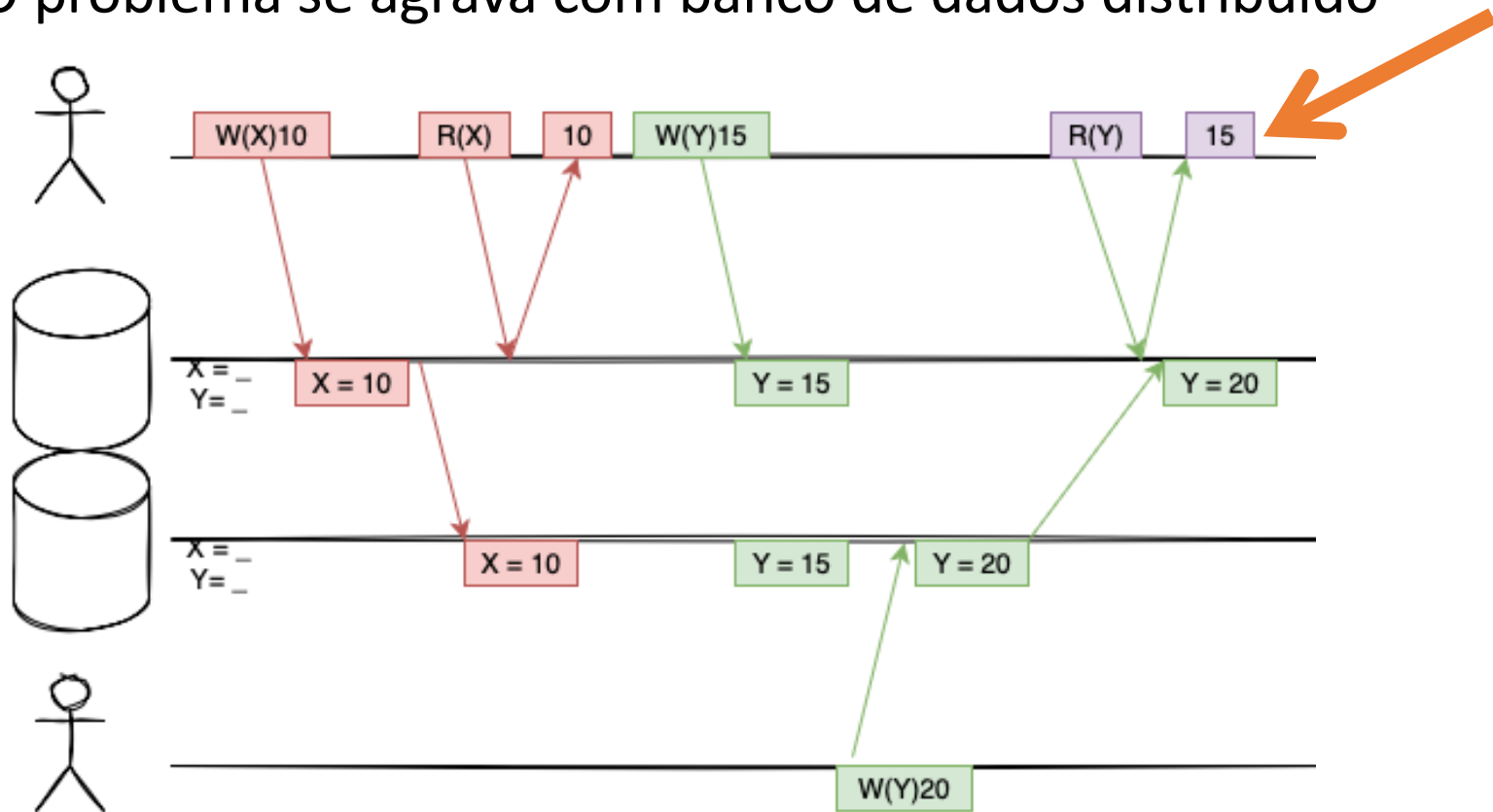
Consistência

- Implementação de modelo de consistência
- Dificuldades:
 - Operações não são atômicas
 - Processos em máquinas distintas
 - Tempo de propagação para os comandos e respostas



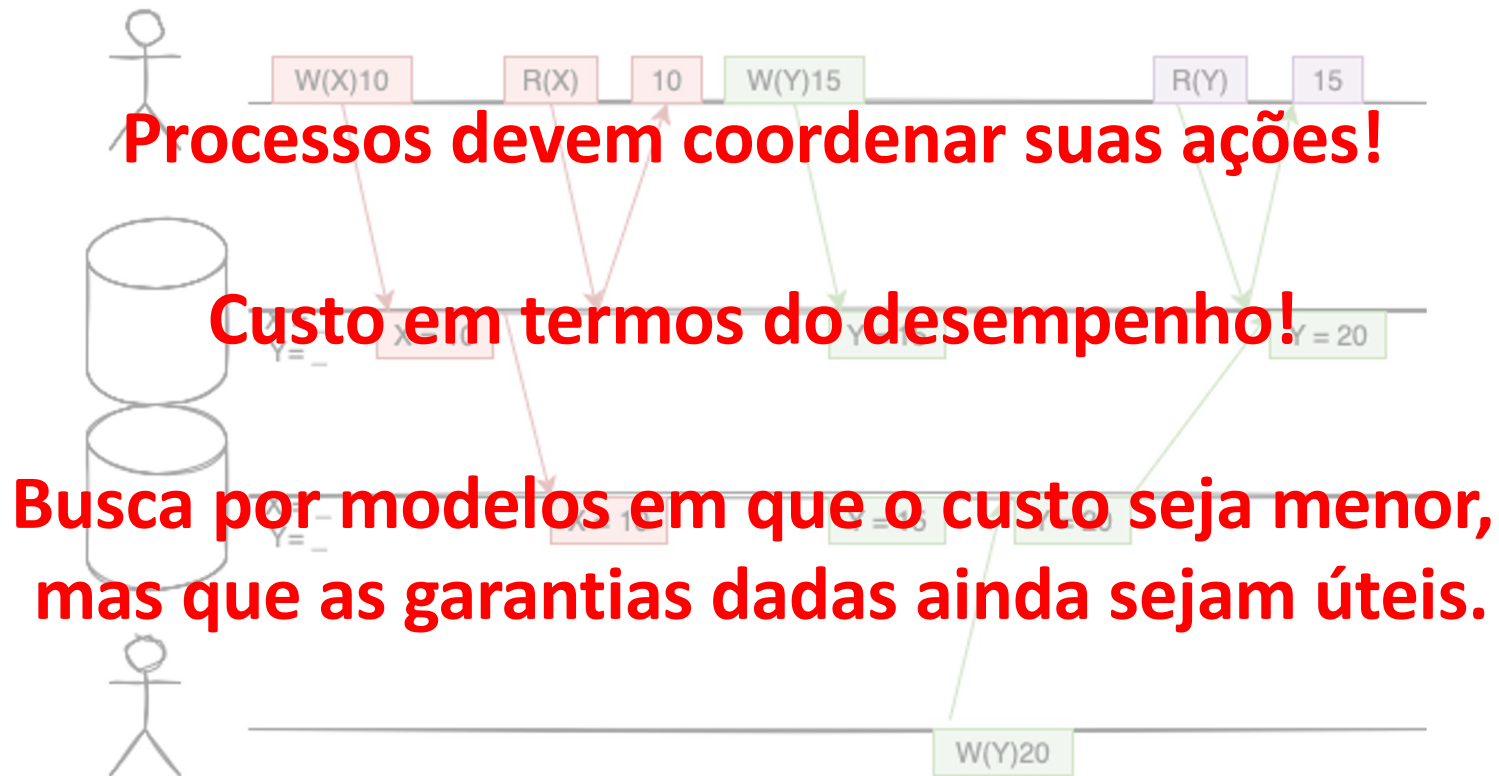
Consistência

- Implementação de modelo de consistência
- Dificuldades:
 - O problema se agrava com banco de dados distribuído



Consistência

- Implementação de modelo de consistência
- Dificuldades:
 - O problema se agrava com banco de dados distribuído

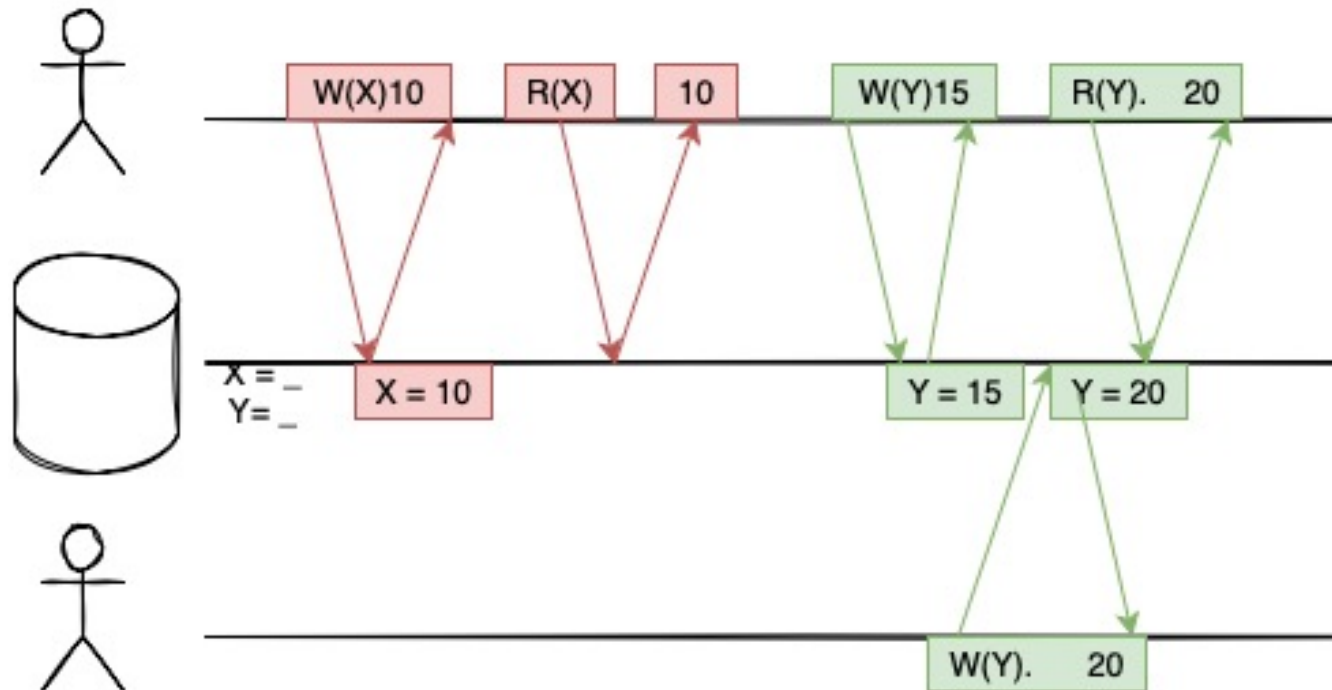


Consistência - Modelos

- Os modelos pode ser classificados sob 2 pontos de vistas:
- Centrados nos dados
 - Foco na relação entre operações nos objetos (dados)
 - São
 - Linearizabilidade, cons. sequencial, cons. causal, cons. FIFO, cons. de entrada
- Centrados nos clientes
 - Foco na percepção do cliente sobre o consistência do sistema
 - São
 - Leituras Monotônicas, Escritas Monotônicas, Leia suas Escritas e Escritas seguem Leituras

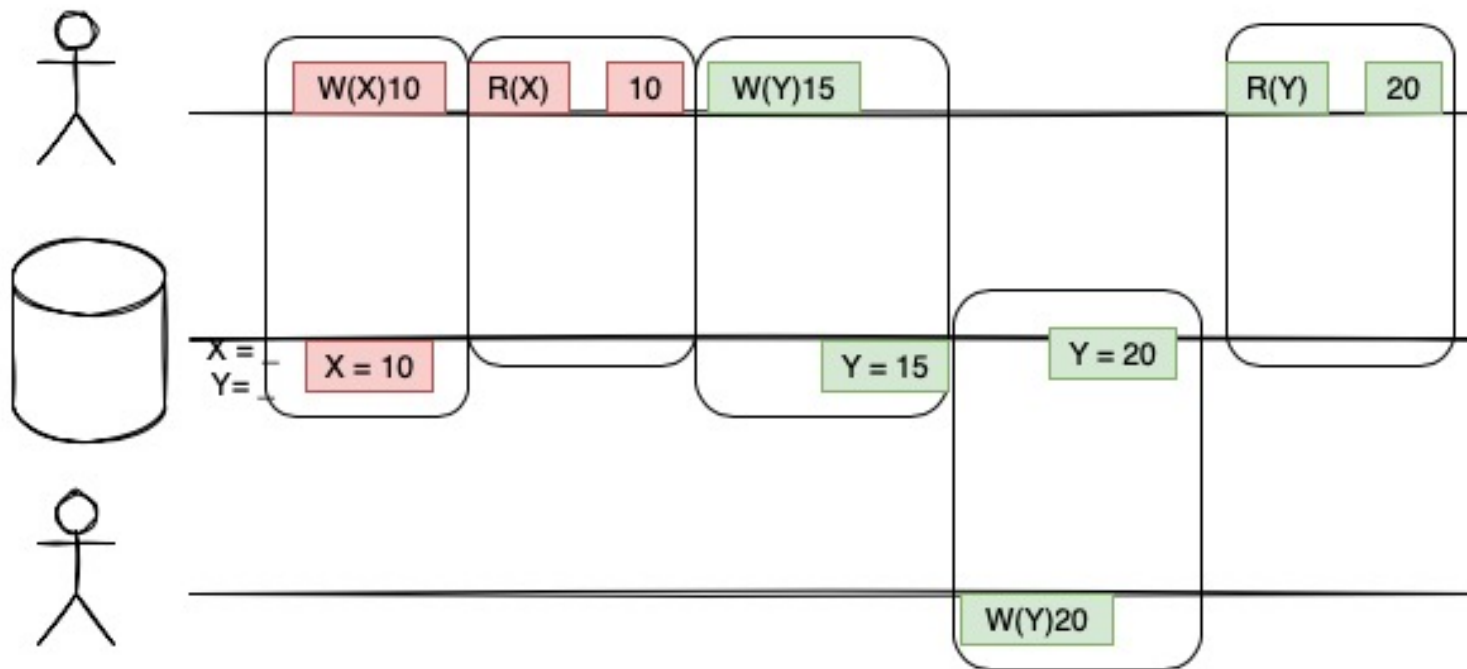
Modelos centrados nos dados

- **Linearizabilidade** (*linearizability*)
 - Modelo mais forte de consistência,
 - Operações aparentam executar atomicamente
 - Respeitam a ordem temporal das operações
 - Mesmo comportamento de um sistema centralizado



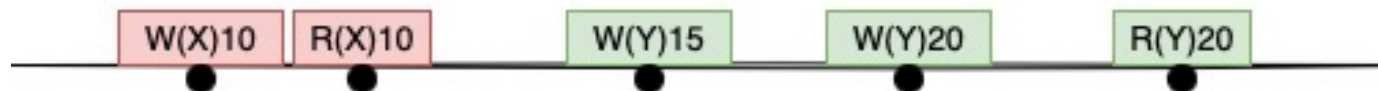
Modelos centrados nos dados

- Linearizabilidade (*linearizability*)
- Execução é equivalente à seguinte, onde não há concorrência entre operações:



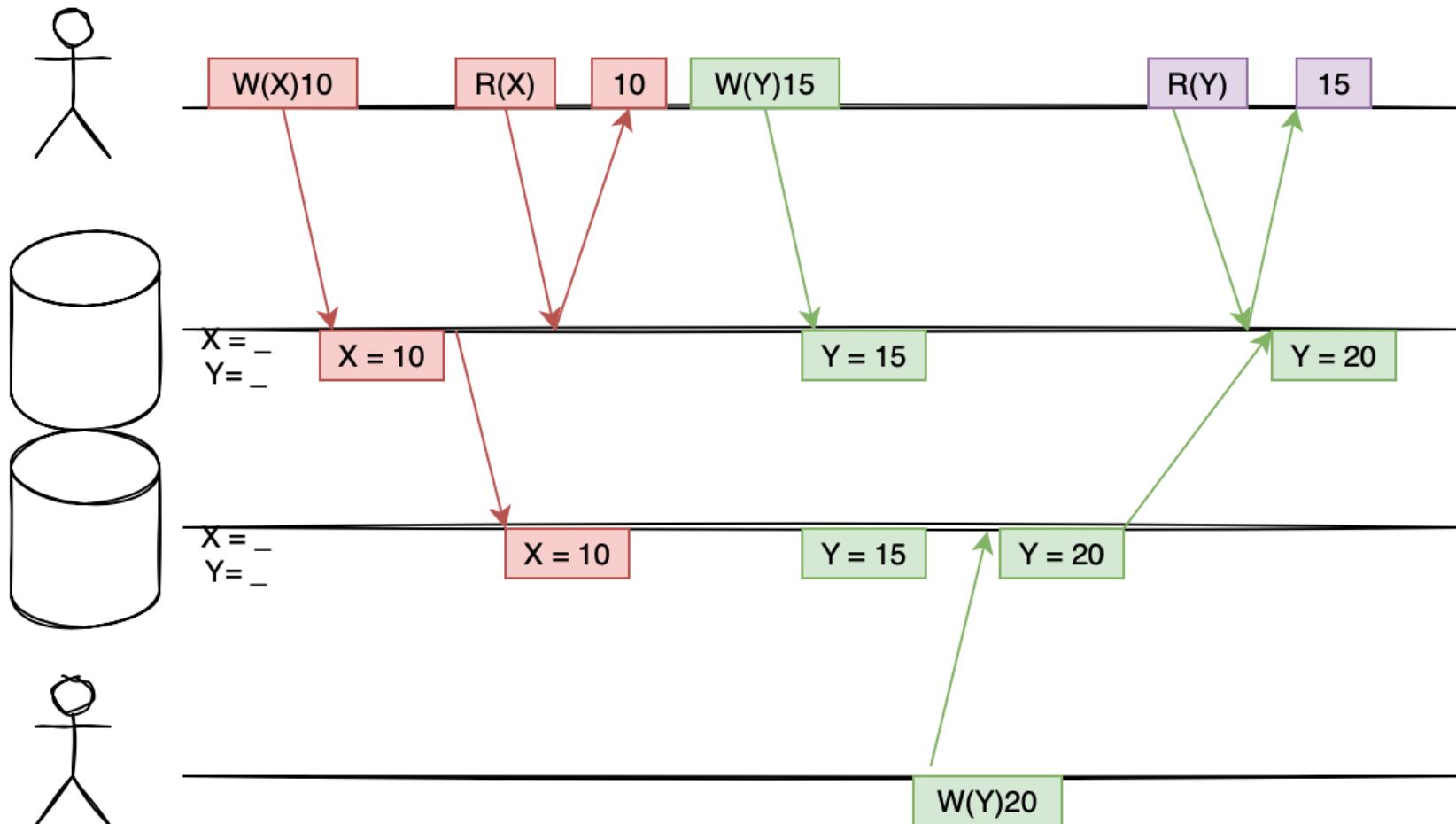
Modelos centrados nos dados

- Linearizabilidade (*linearizability*)
- Execução é equivalente à seguinte, onde não há concorrência entre operações:



Modelos centrados nos dados

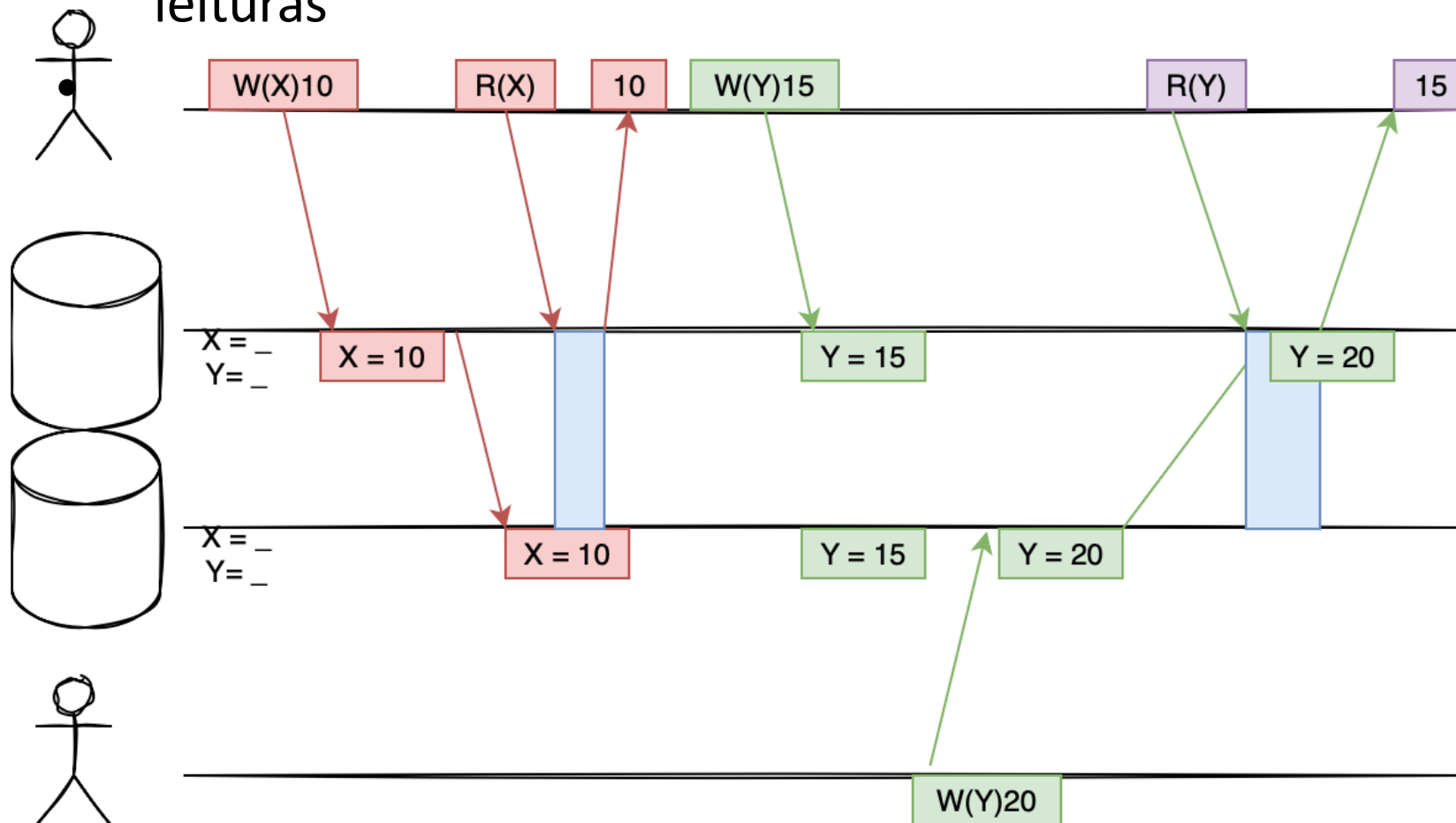
- **Linearizabilidade** (*linearizability*)
 - Revisitando o problema anterior: execução **inválida**



Modelos centrados nos dados

- **Linearizabilidade** (*linearizability*)

- Réplicas precisam se "sincronizar" antes de executarem leituras



Modelos centrados nos dados

- **Linearizabilidade** (*linearizability*)
 - Mais exemplos: qual(is) é(são) linearizável(is)?

P1: $W(x)a$

P2: $R(x)a$
(a)

P1: $W(x)a$

P2: $R(x)NIL$ $R(x)a$
(b)

- Notação:
 - $R(x)a$: A leitura de x retorna a
 - $W(x)a$: Atualização do valor de x para a

Modelos centrados nos dados

- Linearizabilidade (*linearizability*)
 - Mais exemplos: qual(is) é(são) linearizável(is)?

P1: $W(x)a$
P2: $R(x)a$
(a)

Válida

P1: $W(x)a$
P2: $R(x)NIL$ $R(x)a$
(b)

Inválida

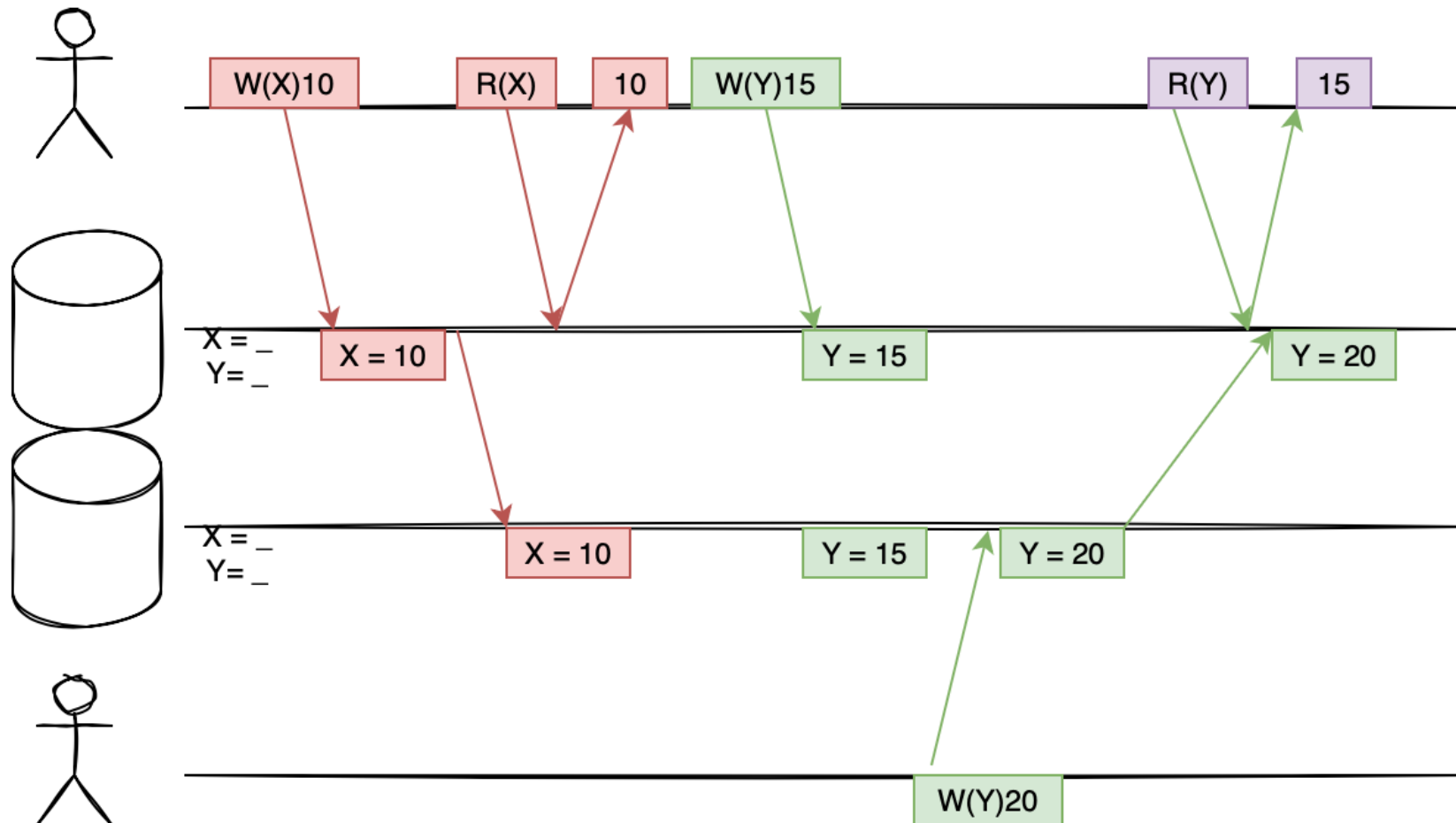
- Notação:
 - $R(x)a$: A leitura de x retorna a
 - $W(x)a$: Atualização do valor de x para a

Modelos centrados nos dados

- **Consistência Sequencial** (*sequential consistency*)
 - A grande dificuldade da linearizabilidade vem da exigência das operações respeitarem uma ordem baseada no **tempo** de execução
 - Exige sincronização entre os nós onde o dado é armazenado, **a cada operação**, o que pode se muito custoso
 - A Consistência Sequencial abre mão da obrigação de fidelidade ao tempo de execução

Modelos centrados nos dados

- **Consistência Sequencial** (*sequential consistency*)
 - Revisitando o problema anterior: execução **válida**!



Modelos centrados nos dados

- **Consistência Sequencial** (*sequential consistency*)
 - Mais exemplos:

P1:	W(x)a		
P2:		W(x)b	
P3:		R(x)b	R(x)a
P4:		R(x)b	R(x)a

(a)

P1:	W(x)a		
P2:		W(x)b	
P3:		R(x)b	R(x)a
P4:		R(x)a	R(x)b

(b)

Modelos centrados nos dados

- **Consistência Sequencial** (*sequential consistency*)
 - Mais exemplos:

P1:	W(x)a		
P2:	W(x)b		
P3:		R(x)b	R(x)a
P4:		R(x)b	R(x)a

(a)

Válida

P1:	W(x)a		
P2:	W(x)b		
P3:		R(x)b	R(x)a
P4:		R(x)a	R(x)b

(b)

Inválida

Modelos centrados nos dados

- **Consistência Causal** (*causal consistency*)
 - Escritas com *potencial* relação causal são vistas por todos os processos na mesma ordem
 - Escritas concorrentes (não causalmente relacionadas) podem se vistas em ordens diferentes por processos diferentes

P1:	W(x)a		W(x)c	
P2:		R(x)a	W(x)b	
P3:		R(x)a		R(x)c
P4:		R(x)a		R(x)b
			R(x)b	R(x)c

- **$W(x)b$** depende de **$R(x)a$** que depende de **$W(x)b$**
- **$W(x)c$** e **$W(x)b$** são concorrentes

Modelos centrados nos dados

- **Consistência Causal** (*causal consistency*)
 - Mais exemplos:

P1:	W(x)a		
P2:	R(x)a	W(x)b	
P3:		R(x)b	R(x)a
P4:		R(x)a	R(x)b

(a)

P1:	W(x)a		
P2:		W(x)b	
P3:		R(x)b	R(x)a
P4:		R(x)a	R(x)b

(b)

Modelos centrados nos dados

- **Consistência Causal** (*causal consistency*)
 - Mais exemplos:

P1:	W(x)a		
P2:	R(x)a	W(x)b	
P3:		R(x)b	R(x)a
P4:		R(x)a	R(x)b

(a)

Inválida

P1:	W(x)a		
P2:		W(x)a	
P3:		R(x)b	R(x)a
P4:		R(x)a	R(x)b

(b)

Válida

Modelos centrados nos dados

- **Consistência FIFO** (*FIFO consistency*)

- **Escritas de um processo** são vistas por todos os outros processos na ordem em que foram feitas
- Escritas de diferentes processos podem ser vistas em ordens diferentes

P1: $W(x)a$

P2: $R(x)a$ $W(x)b$ $W(x)c$

P3: $R(x)b$ $R(x)a$ $R(x)c$

P4: $R(x)a$ $R(x)b$ $R(x)c$

- $W(x)b$ precede $W(x)c$
- $W(x)a$ não tem nenhuma relação com demais escritas

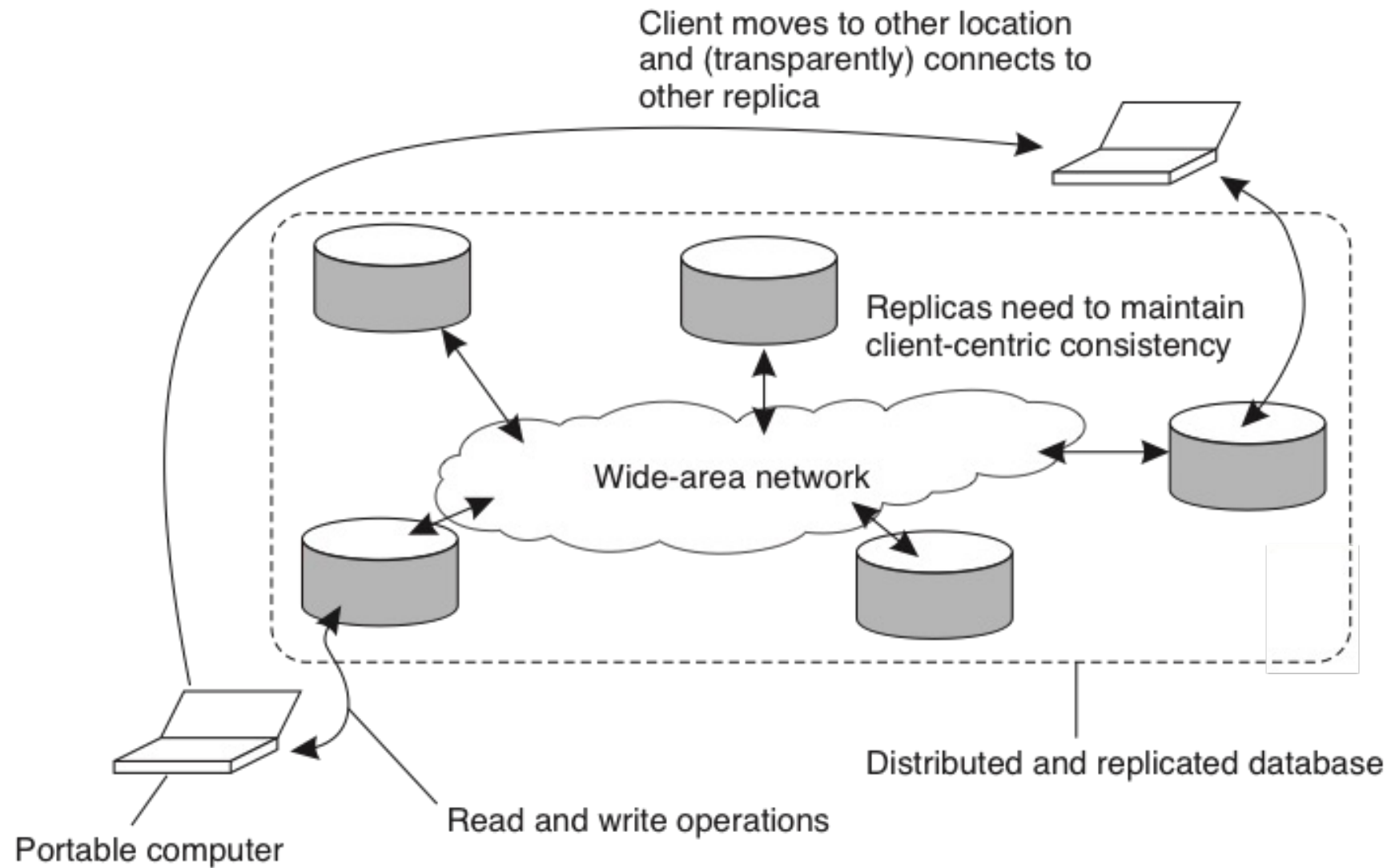
Modelos centrados nos dados

- **Consistência de Entrada**
- Efeitos de operações individuais em um grupo não são visíveis.
- Variáveis de sincronização
 - Acesso às variáveis de sincronização da datastore é sequencialmente consistente.
 - Acesso à variável de sincronização não é permitido até que todas as escritas das anteriores tenham sido executadas em todos os lugares.
 - Acesso aos dados não é permitido até que todas as variáveis de sincronização tenham sido liberadas.

Modelos centrados nos clientes

- Objetivo
 - Evitar sincronização global
 - Se para os clientes parecer consistente, tudo bem
- Consistência **Eventual**
 - Se nenhuma escrita ocorrer em período considerável de tempo, os clientes gradualmente se sincronizarão e ficarão consistentes
 - Se clientes sempre acessarem as mesmas réplicas, terão impressão de consistência
- Garantias são do ponto de vista de **um** cliente.
 - Leituras monotônicas
 - Escrita monotônicas
 - Leia suas escritas
 - Escritas seguem leituras

Modelos centrados nos clientes



Modelos centrados nos clientes

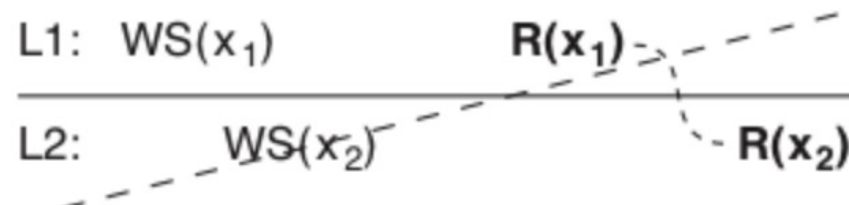
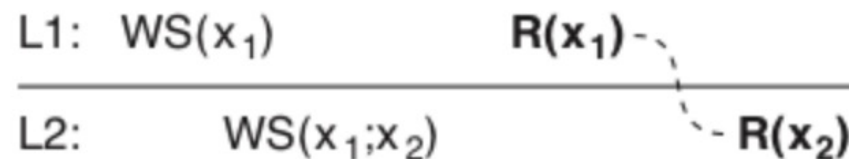
- **Leituras Monotônicas**

- **Garantia**

- Se um processo lê o valor de um item x , qualquer leitura sucessiva de x retornará o mesmo valor ou um mais recente

- **Exemplo:**

- Toda vez que se conecta a um servidor de e-mail, seu cliente lê novas mensagens, caso haja.
 - O cliente nunca esquece uma mensagem, mesmo que ainda não esteja no servidor conectado por último.



Modelos centrados nos clientes

- **Escritas Monotônicas**

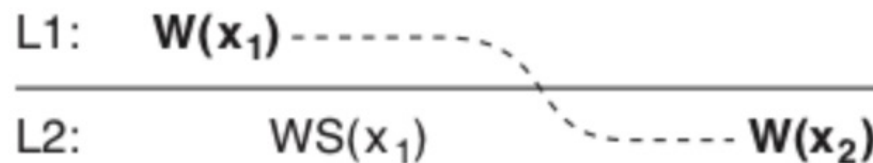
- Garantia

- Se um processo escreve em item x , então esta operação deve terminar antes que qualquer escrita sucessiva em x possa ser executada pelo mesmo processo

- Exemplo:

- Sistema de arquivos na rede:

- escrita do conteúdo de um arquivo, em certa posição, só pode ser feita se escritas anteriores já estão registradas no arquivo



Modelos centrados nos clientes

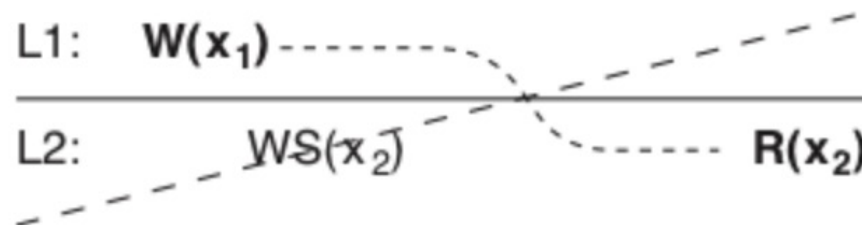
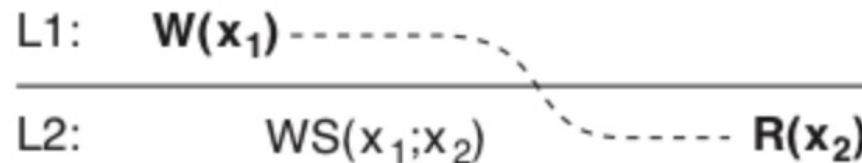
- Leia suas escritas

- Garantia

- Se um processo escreve em item x , então leituras sucessivas no mesmo item pelo **mesmo** processo devem refletir tal escrita

- Exemplo:

- Atualizar código fonte de uma página e exigir que o navegador carregue a nova versão



Modelos centrados nos clientes

- **Escritas seguem leituras**

- Garantia

- Se um processo lê um item x , então escritas sucessivas no mesmo item só podem ser completadas se o mesmo reflete o valor lido anteriormente

- Exemplo:

- Só é permitido enviar uma resposta a uma mensagem se a mensagem em si é vista, independentemente do cliente ter se movimentado

