Victor Bona

Sistemas Distribuídos

Ementa

- Sobre o palestrante
- Introdução
- O que são sistemas distribuídos
- Propósitos e tipos de sistemas distribuídos
- Exemplos de sistemas distribuídos amplamente utilizados
- Por que usar sistemas distribuídos?
- Sistemas distribuídos em um contexto de micro serviços
- Sistemas distribuídos em um contexto de banco de dados
- Sistemas distribuídos em um contexto de sistemas especialistas
- Infraestrutura de Sistemas distribuídos
- Consenso
  - Oque é?
  - Tipos de garantia (linearidade, garantia de ordem...)
  - O problema de consenso
  - Soluções para garantir consenso
- Falhas Bizantinas
- Contratos
- Coordenação e orquestração dos sistemas
- Garantias transacionais
- Boas práticas ao trabalhar com sistemas distribuídos
- Hands-on (prática)
- Referências e sugestões bibliográficas
- Q&A

Sobre o palestrante

- Victor Bona
- Atua no mercado de TI a mais de 3 anos
- Ciência da computação @ FURB
- Engenheiro de software @ Amazon
- Certificações (AWS):
  - Solutions Architect – SAA-C03
  - Advanced Networking - ANS-C01

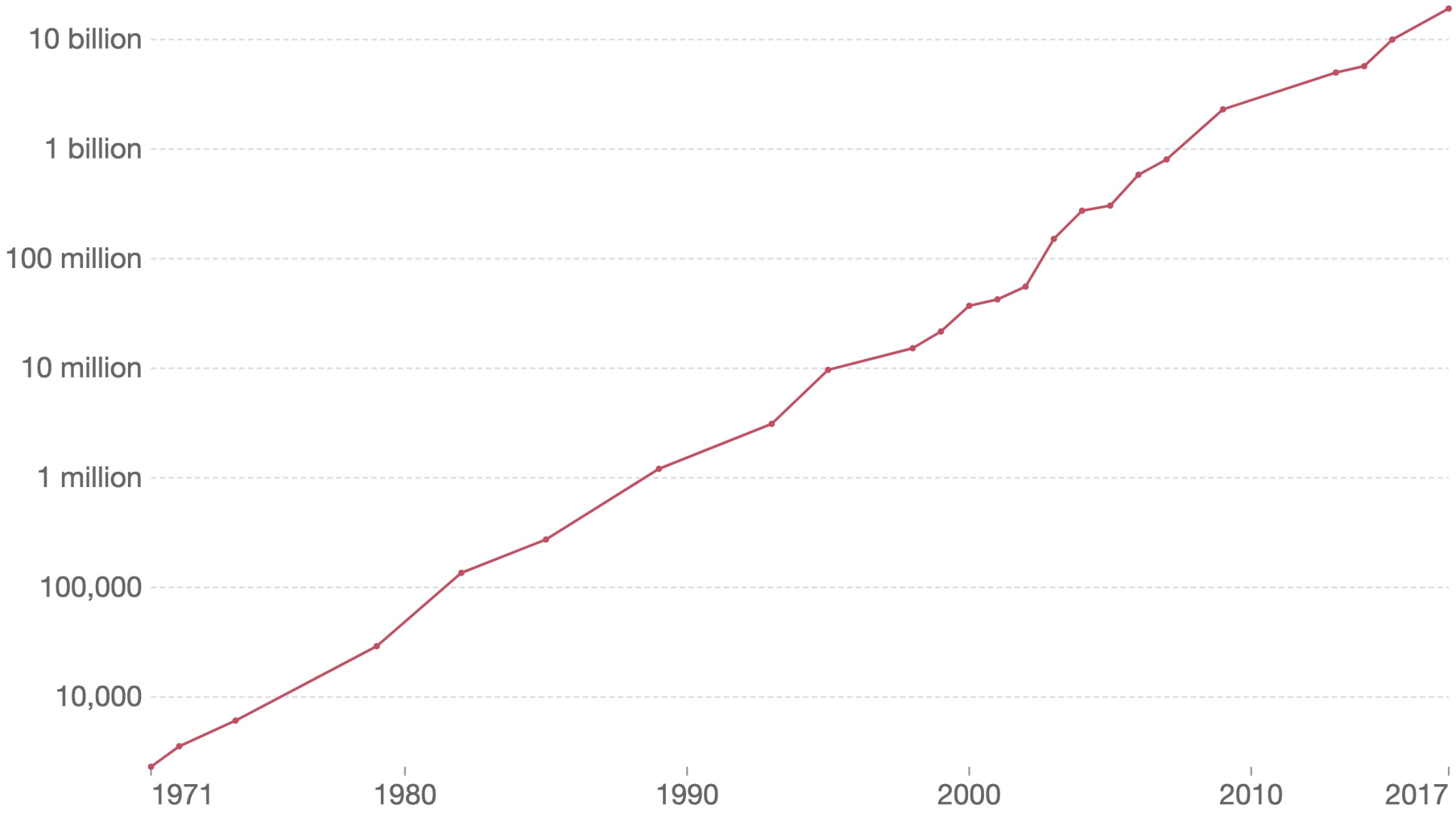
Introdução

- O poder computacional individual de um processador, é naturalmente limitado pelas leis da física.
- Lei de Moore.
- A demanda por processamento não pode ser suprida pelo poder de processamento individual.
- Uma única entidade que faz tudo, acaba fazendo tudo mal feito.

Moore's Law: The number of transistors per microprocessor

Our World in Data

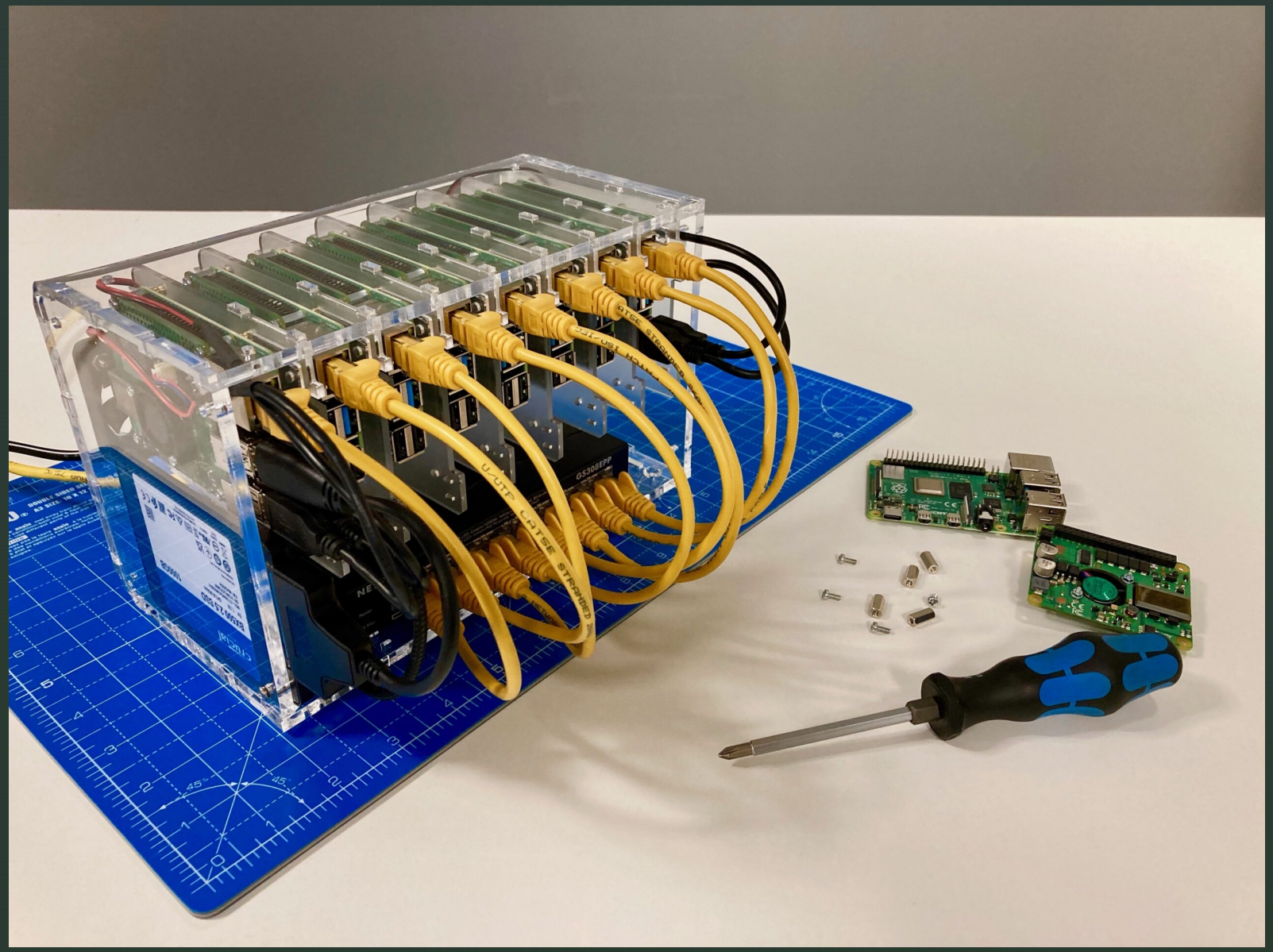
Number of transistors which fit into a microprocessor. The observation that the number of transistors on an integrated circuit doubles approximately every two years is called 'Moore's Law'.

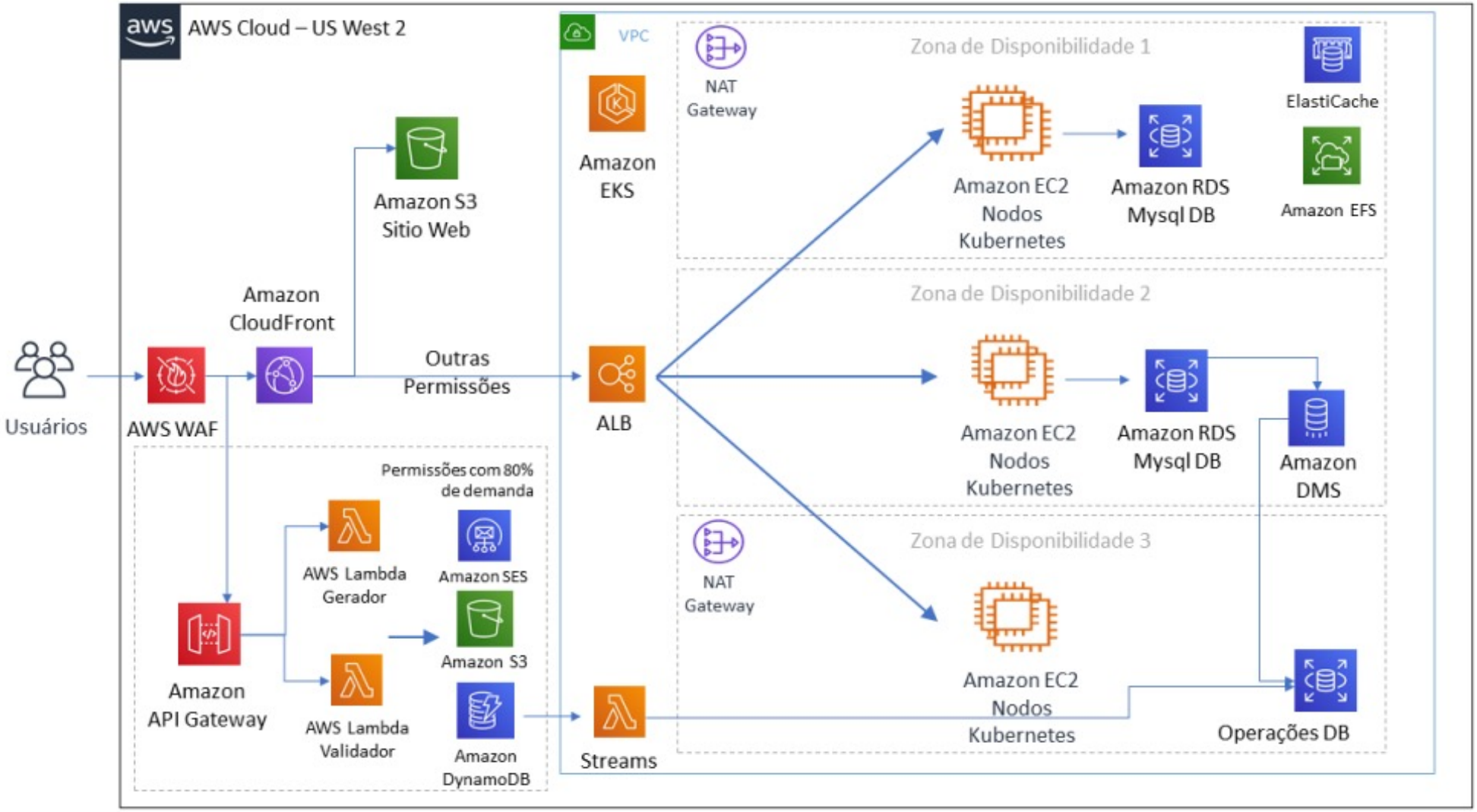


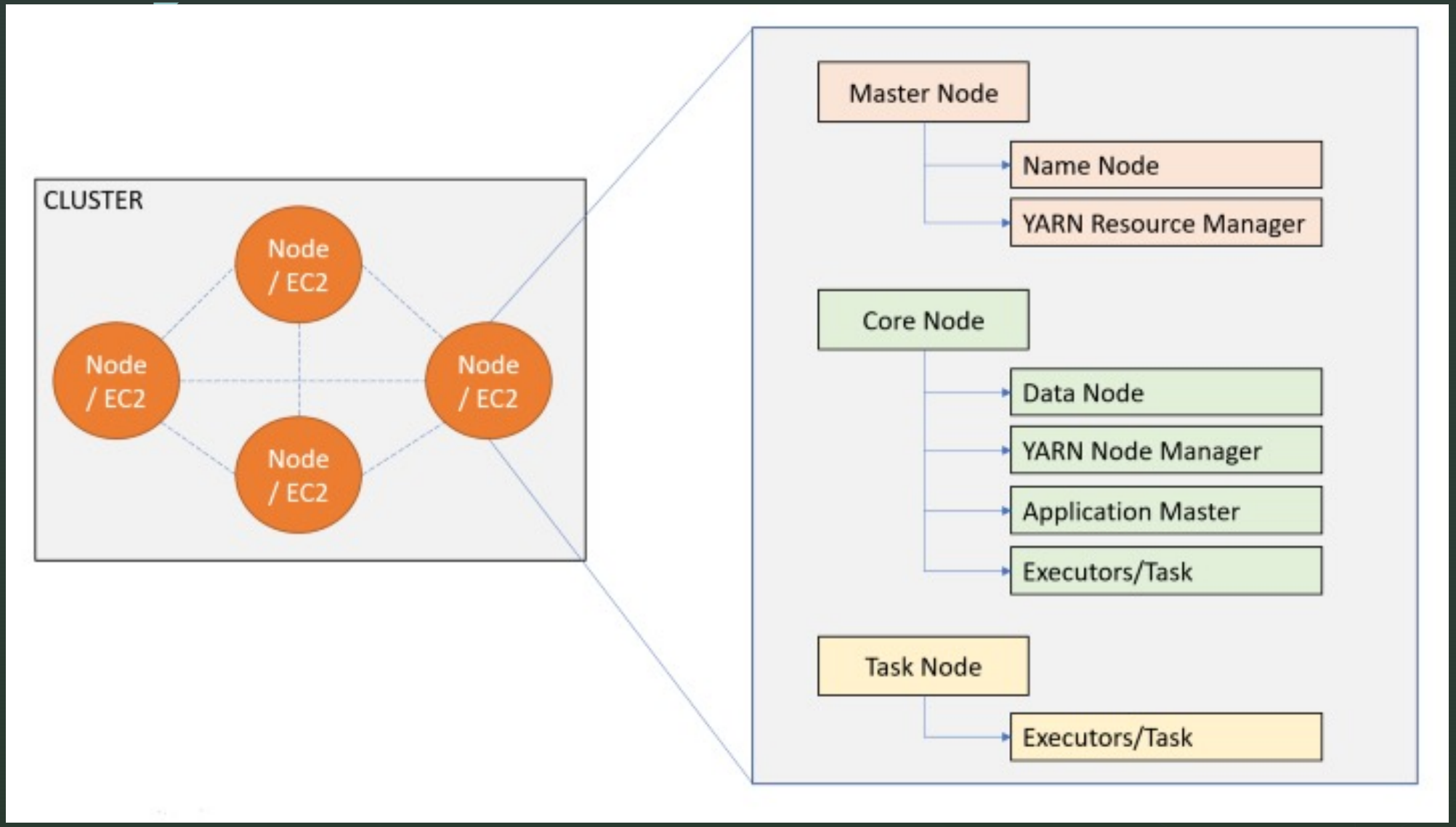
Source: Karl Rupp. 40 Years of Microprocessor Trend Data.

O que são Sistemas Distribuídos

- Caracteriza-se pela distribuição do processamento entre diversas unidades de computação, serviços e/ou plataformas.
- Divide contexto e escopo com computação distribuída em geral.
- Pode acontecer em contextos de hardware ou software.
- Melhora a granularidade e organização de sistemas complexos.
- Conjuntos computacionais são divididos em *Clusters*.




AWS Cloud – US West 2
VPC
Zona de Disponibilidade 1
Zona de Disponibilidade 2
Zona de Disponibilidade 3
Usuários
AWS WAF
Amazon CloudFront
Amazon S3 Sitio Web
Outras Permissões
Amazon EKS
ALB
NAT Gateway
NAT Gateway
Amazon EC2 Nodos Kubernetes
Amazon RDS Mysql DB
ElastiCache
Amazon EFS
Amazon EC2 Nodos Kubernetes
Amazon RDS Mysql DB
Amazon DMS
Amazon EC2 Nodos Kubernetes
Operações DB
Permissões com 80% de demanda
AWS Lambda Gerador
Amazon SES
Amazon S3
Amazon API Gateway
AWS Lambda Validador
Amazon DynamoDB
Streams


CLUSTER
Node / EC2
Node / EC2
Node / EC2
Node / EC2
Master Node
Name Node
YARN Resource Manager
Core Node
Data Node
YARN Node Manager
Application Master
Executors/Task
Task Node
Executors/Task

Propósitos e tipos de sistemas distribuídos

Tipos dos sistemas distribuídos

- Dado a natureza individual dos componentes em um SD, existem diversos tipos de sistemas que podemos encontrar e/ou arquitetar.
- *Single-leader.*
- *Multi-leader.*
- *Leaderless.*
- *Decentralized leaderless system.*
- *Managed application distributed process.*
- *Distributed pipeline.*
- *State machines.*

Normalmente aplicações especializadas organizadas em clusters

Normalmente sistemas compostos por diversas aplicações gerenciadas.

Propósitos dos sistemas distribuídos

- Devido aos diferentes caso de uso, existem diversos possíveis propósitos e usos de Sistemas distribuídos.
- Sistemas de Computação Distribuídos de Alto Desempenho.
- Sistemas de Informação Distribuídos.
- Sistemas de alta disponibilidade e resiliência.
- Sistemas Distribuídos Embutidos.
- Sistemas de baixo custo para computação não critica.
- Sistemas de governança descentralizada.

Exemplos de sistemas distribuídos amplamente utilizados

- Internet;
- Blockchain;
- Bancos de dados;
- Pipelines de processamento de dados;
- Serviços de mensageria e notificação;
- Aplicações em nuvem.
- Serviços de streaming.

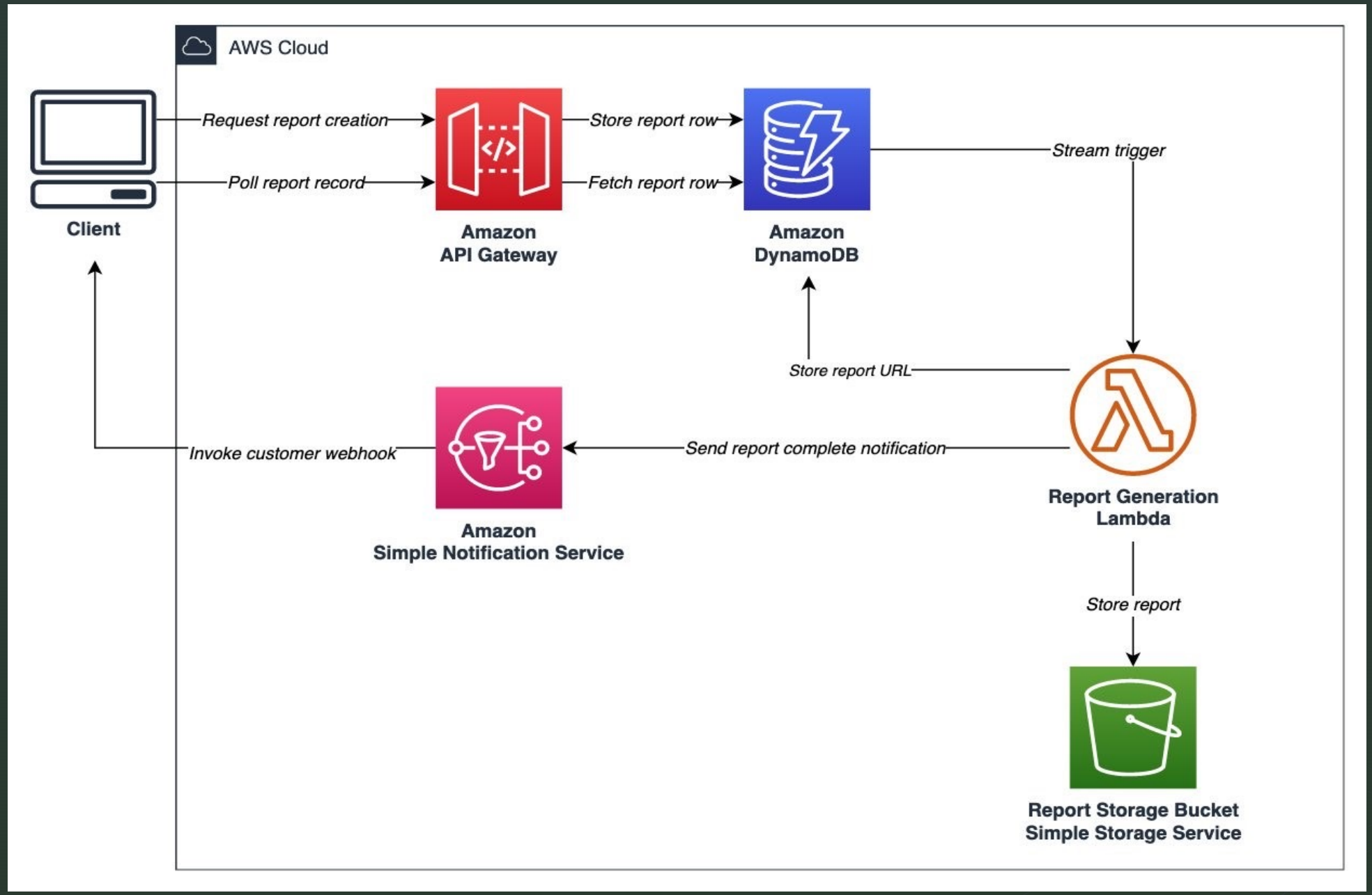
Por que usar sistemas distribuídos?

- Granularidade
- Escalabilidade
- Segurança
- Desempenho
- Resiliência
- Disponibilidade

Contextos de aplicação

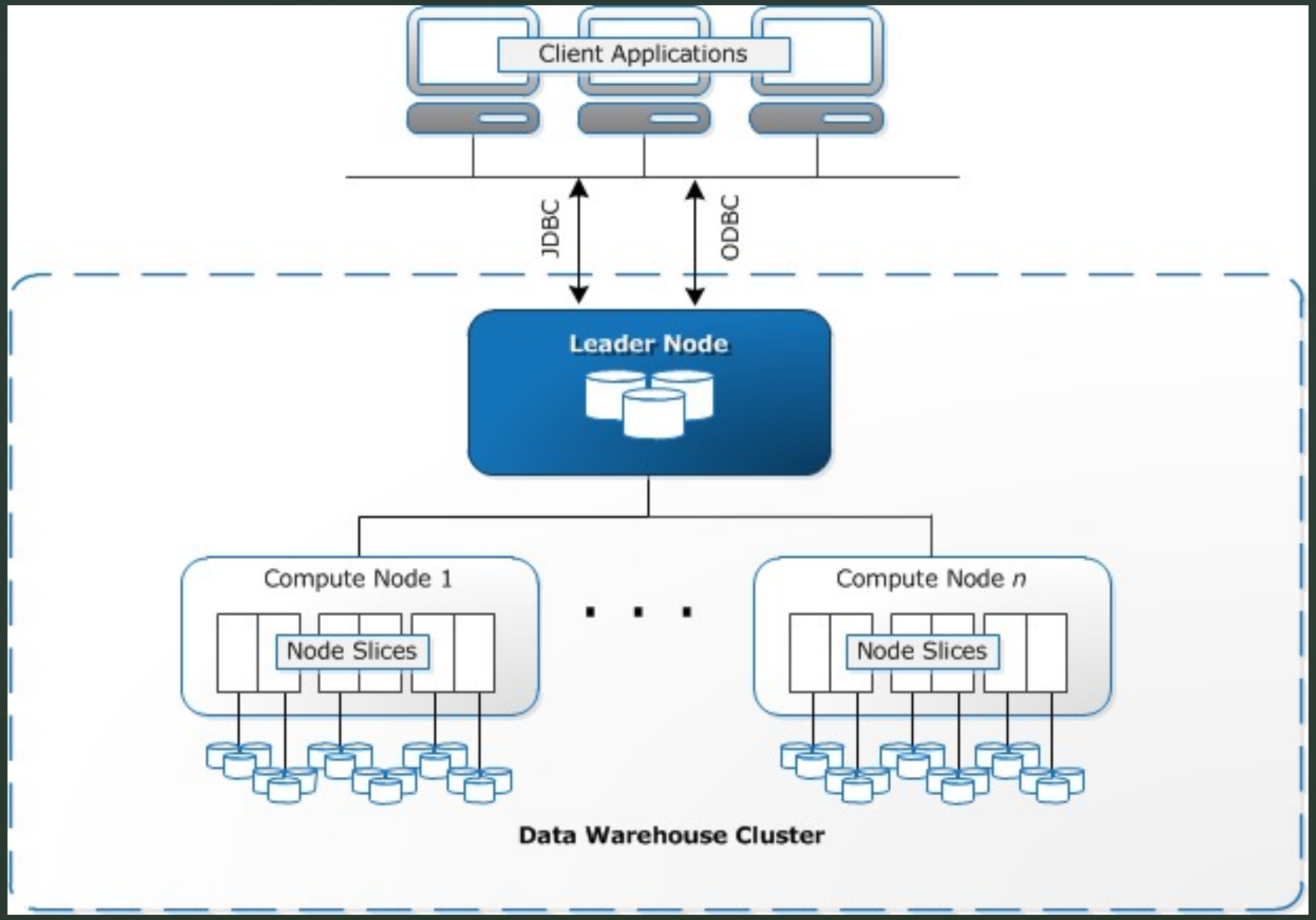
Sistemas distribuídos em um contexto de micro serviços

- Diferentes sistemas e aplicações comunicam-se entre si.
- Cada aplicação possui um escopo e responsabilidade própria.
- Pode ser integrado sincronamente ou assincronamente.
- Diferentes aplicações podem ser executadas em uma mesma maquina, e ainda assim compor um sistema distribuído.


AWS Cloud
Client
Request report creation
Poll report record
Amazon
API Gateway
Store report row
Fetch report row
Amazon
DynamoDB
Stream trigger
Store report URL
Report Generation
Lambda
Send report complete notification
Invoke customer webhook
Amazon
Simple Notification Service
Store report
Report Storage Bucket
Simple Storage Service

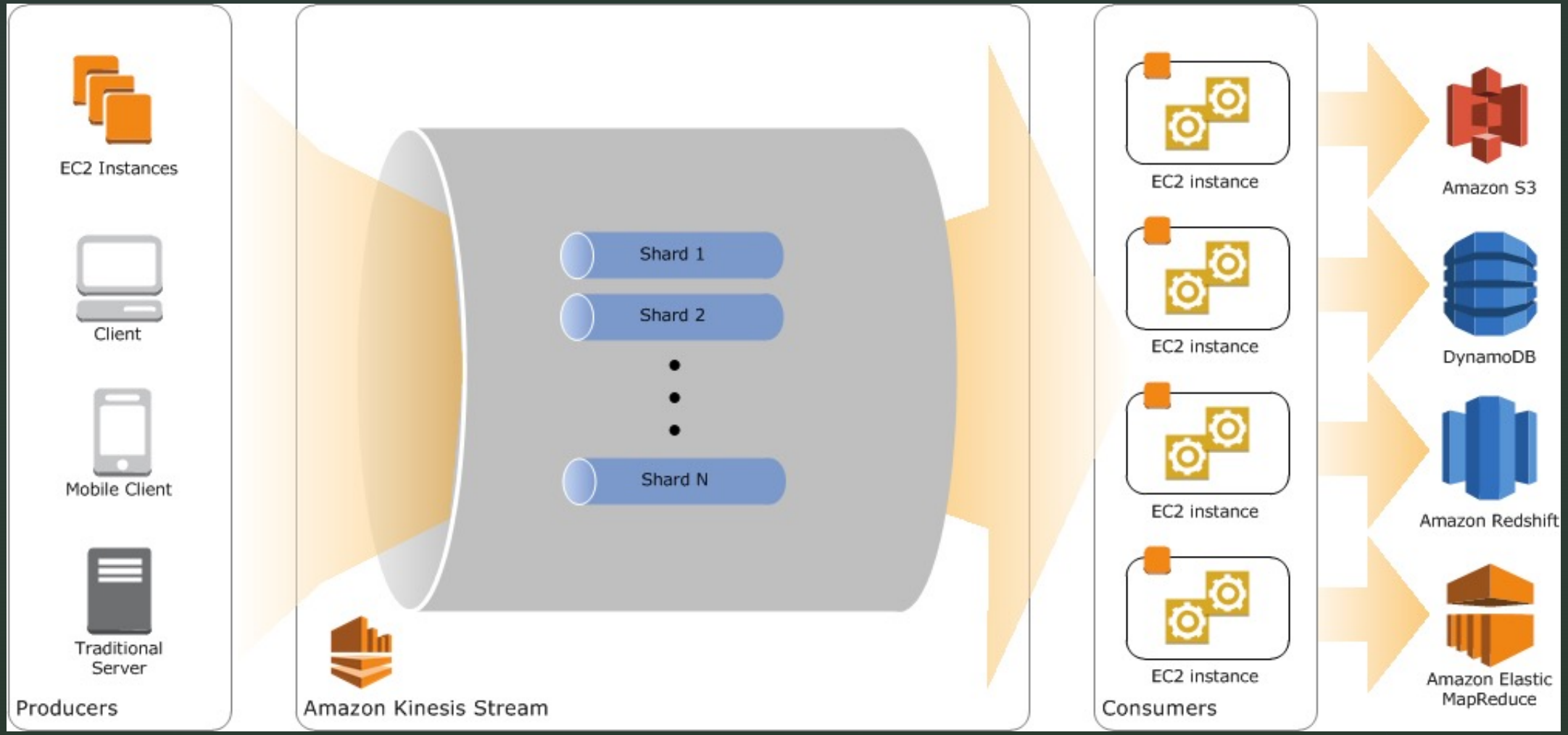
Sistemas distribuídos em um contexto de banco de dados

- Aumento de resiliência, confiabilidade e disponibilidade.
- Demanda implementação de resolução conflitos de leitura e/ou escrita.
- Normalmente executado em um contexto de nós com um ou mais mestres e diversos nós servos.
- Extremamente comum, principalmente em bancos de dados NoSQL.


Client Applications
JDBC
ODBC
Leader Node
Compute Node 1
Node Slices
Compute Node n
Node Slices
Data Warehouse Cluster

Sistemas distribuídos em um contexto de sistemas especialistas

- Sistemas nichados e auto contidos.
- Performance otimizada apenas para um caso de uso especifico.
- Alta disponibilidade.
- Normalmente caro.
- Pode demandar hardware especializado.


EC2 Instances
Client
Mobile Client
Traditional Server
Producers
Shard 1
Shard 2
Shard N
Amazon Kinesis Stream
EC2 instance
EC2 instance
EC2 instance
EC2 instance
Consumers
Amazon S3
DynamoDB
Amazon Redshift
Amazon Elastic MapReduce

Infraestrutura de Sistemas distribuídos

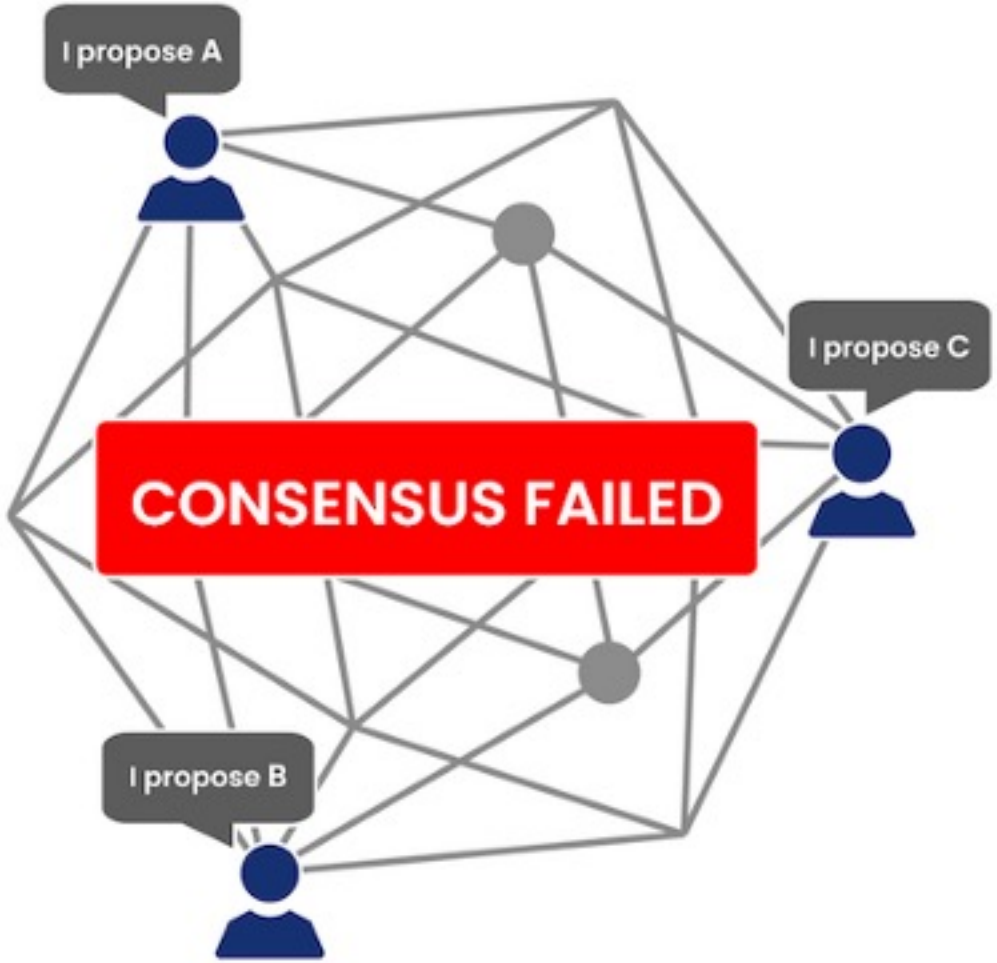
- Com a popularização dos serviços de computação em nuvem, a maioria dos SD utiliza serviços gerenciados ou maquinas virtuais gerenciadas na nuvem.
- A arquitetura mais comum consiste em um cluster de maquinas. Um conjunto de maquinas trabalhando em paralelo para resolver um mesmo ou diversos problemas correlacionados.
- O custo varia muito do tamanho do sistema e proposito da aplicação.

“A consensus means that everyone agrees to say collectively what no one believes individually.”
Abba Eban

Consenso em Sistemas Distribuidos

Oque é consenso no contexto de Sistemas distribuídos?

- Garantia de uma única verdade para dada entidade (dado) em todas as partes do sistema.
- Sistemas centralizados não possuem necessidade de consenso.
- Falta de consenso pode causar *resultados cinza,* resultados diferentes dependendo do momento e do observador.
- Consenso absoluto é impossível de se atingir sem assumir riscos.
- Consenso pode ser adquirido de diversas formas.
- Afeta diretamente o funcionamento de todo o sistema.
- Falhas de consenso podem causar problemas na consistência dos dados.
- Dificilmente afeta sistemas pequenos.


I propose A
I propose C
CONSENSUS FAILED
I propose B

Exemplos de uso em Sistema Distribuídos

- Eleição de nós lideres em clusters de maquinas.
- Estado e promoção de dados.
- Exclusão mutua – Acesso exclusivo de um nó especifico a uma sessão critica.
- Transações distribuídas – Sistemas de conclusão ou exclusão de uma transação.
- Ordem de execução - Decide qual a próxima transação a ser executada.
- Garantia de consistência – Tenta garantir que todas as partes de um sistema possuem a mesma versão de um determinado dado.
- Priorização de transação – Garantia de que todas as transações serão priorizadas corretamente.

Tipos de garantia de consenso

Linearização

- Consenso por observação.
- Método que tenta forçar o sistema a possuir apenas uma versão de um determinado dado, garantindo propagação da consistência após leitura.
- Se uma leitura com novo valor é feita, o valor desta leitura passa a valer para todas as outras.
- Confiável, entretanto suscetível a problemas de consistência.

Sistema não linearizado

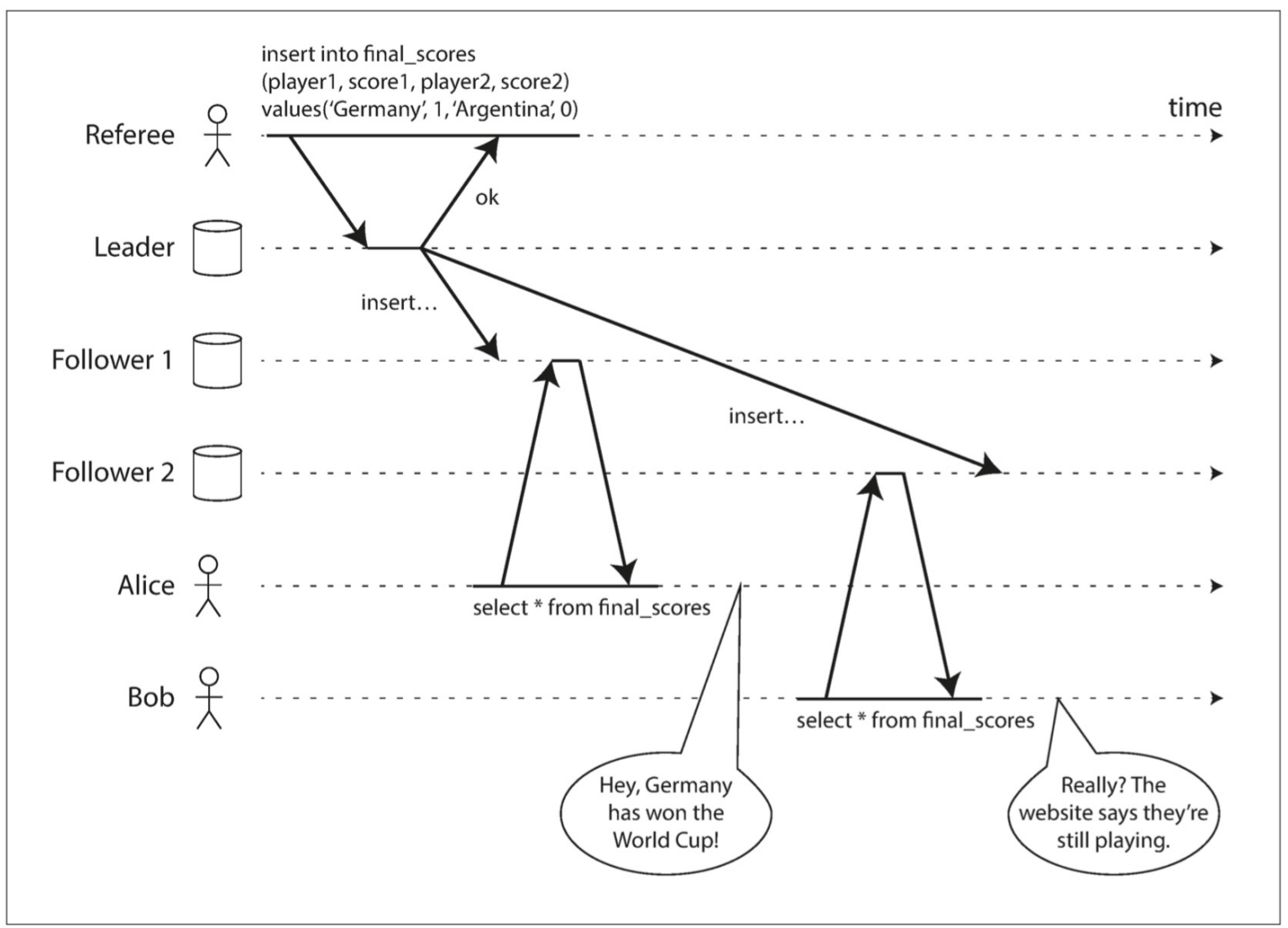


Figure 9-1. This system is not linearizable, causing football fans to be confused.

Sistema linearizado

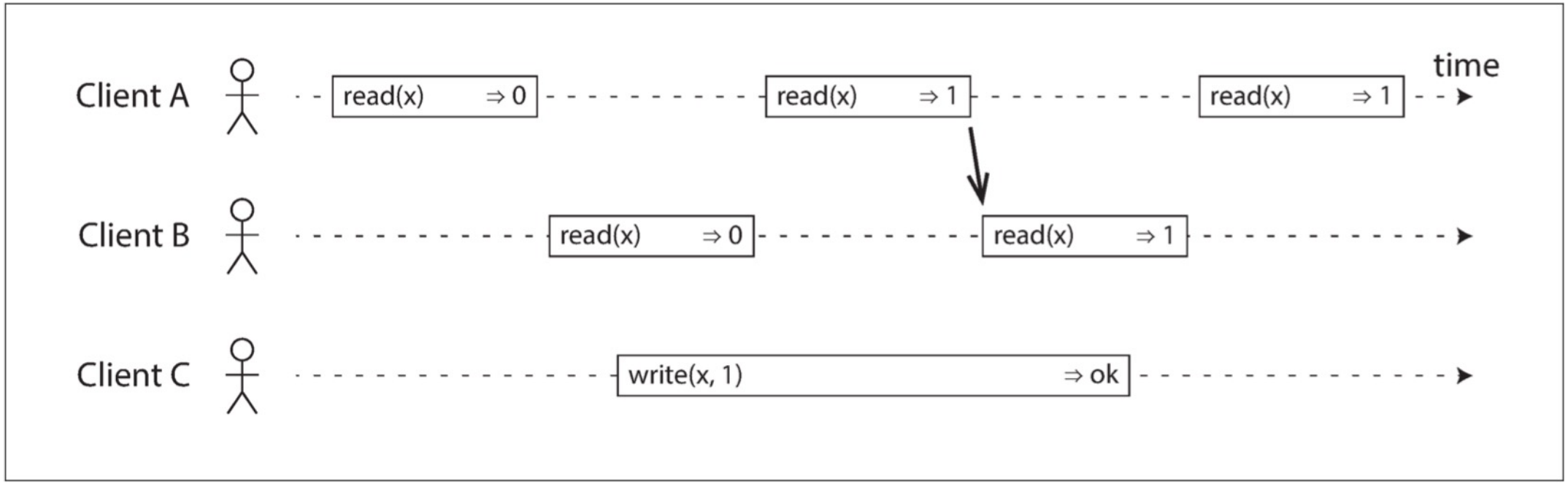


Figure 9-3. After any one read has returned the new value, all following reads (on the same or other clients) must also return the new value.

Two-phase commit

- Procura garantir o consenso ao criar uma transação coordenada, preparando todos as partes do sistema antes de realizar escritas e/ou leituras.
- Normalmente, nós ou sistemas lideres orquestram a transação.
- Cria um ponto único de falha da transação no líder.
- Impacta a performance da aplicação.
- Demanda *lock* de recursos.
- Comumente utilizado em bancos de dados.
- Excelente para sistemas assíncronos que não dependem muito de performance.

Exemplo de escrita de duas fases em diversos nós

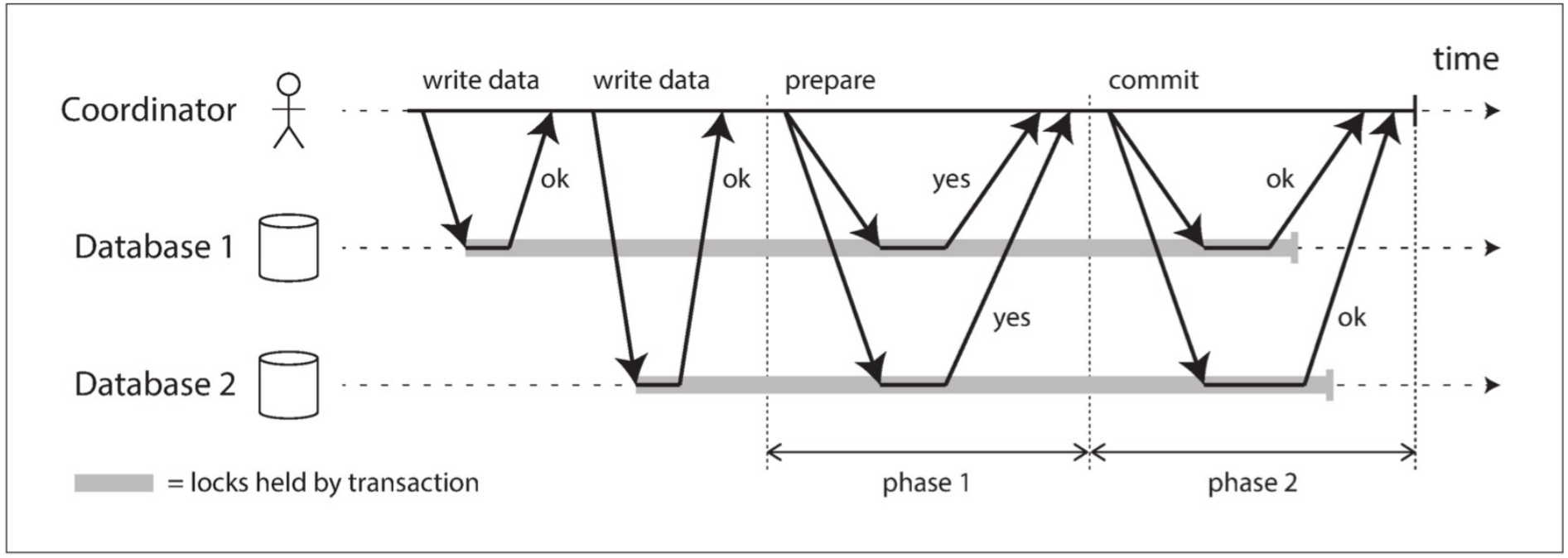
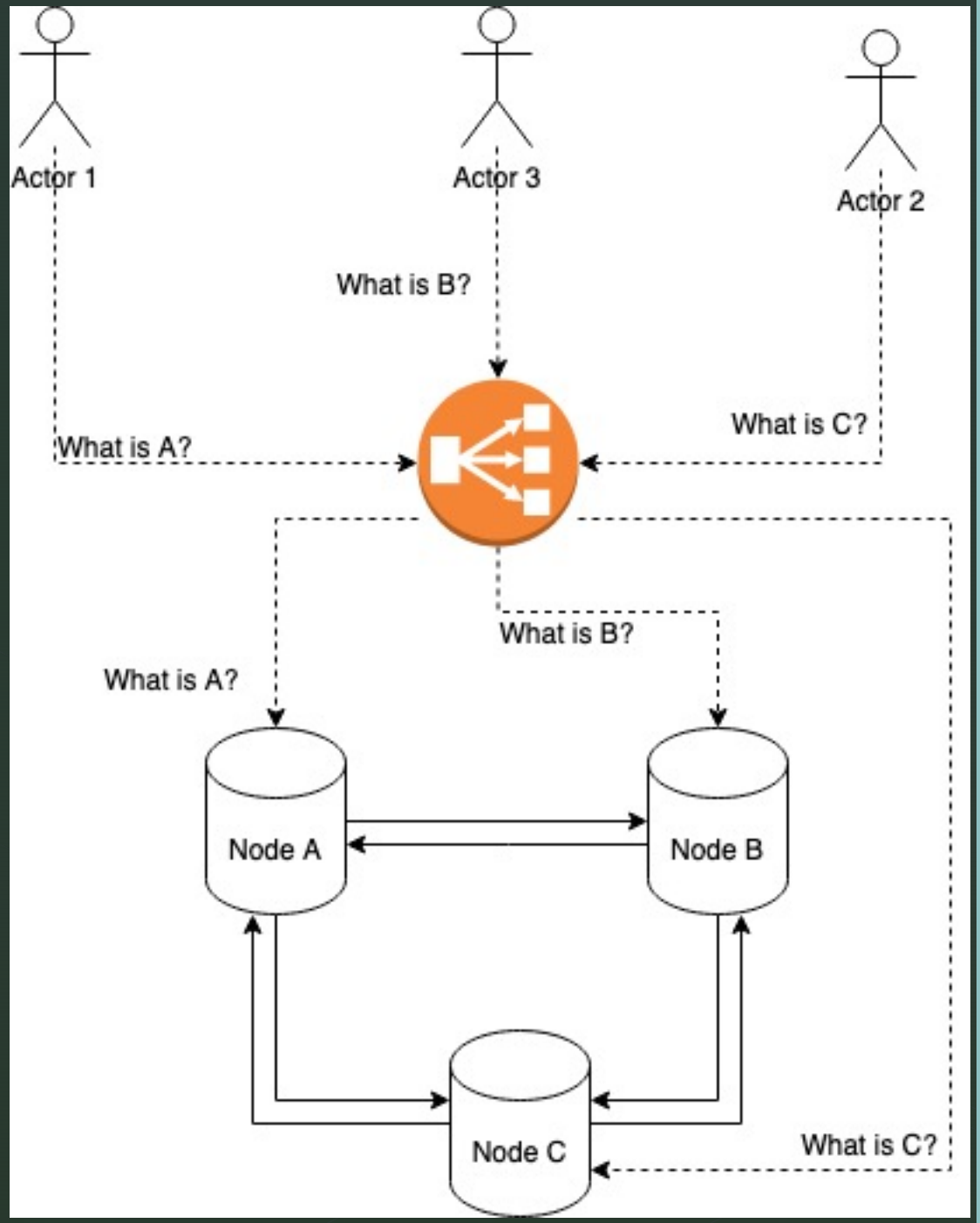


Figure 9-9. A successful execution of two-phase commit (2PC).

Garantia de consenso por exclusividade

- Garante o consenso entre as diversas partes da aplicação ao delegar a responsabilidade de escrita e leitura para partes especificas.
- Suscetível a falhas.
- Todas as partes do sistema podem possuir um mesmo dado, mas apenas uma parte é considerada a fonte da verdade para o dado.
- Escala mal em situações de stress.

Exemplo


Actor 1
Actor 3
Actor 2
What is B?
What is A?
What is C?
What is B?
What is A?
Node A
Node B
Node C
What is C?

Consenso por ordenação

- Garante o consenso entre as partes usando um atributo para determinar a ordem das alterações que definem o estado de um dado ou do sistema.
- Demanda que todos os sistemas possuam a mesma heurística de ordenação.
- Diretamente afetado pela natureza do atributo utilizado para determinar a ordem.
- Ordenação por tempo é consideravelmente suscetível a problemas de rede e sincronização de relógios (**RELÓGIOS NÃO SÃO CONFIÁVEIS**).

Entre outros exemplos na literatura.

Engenheiros são livres para definir seus sistemas de consenso em suas aplicações.

O problema do consenso

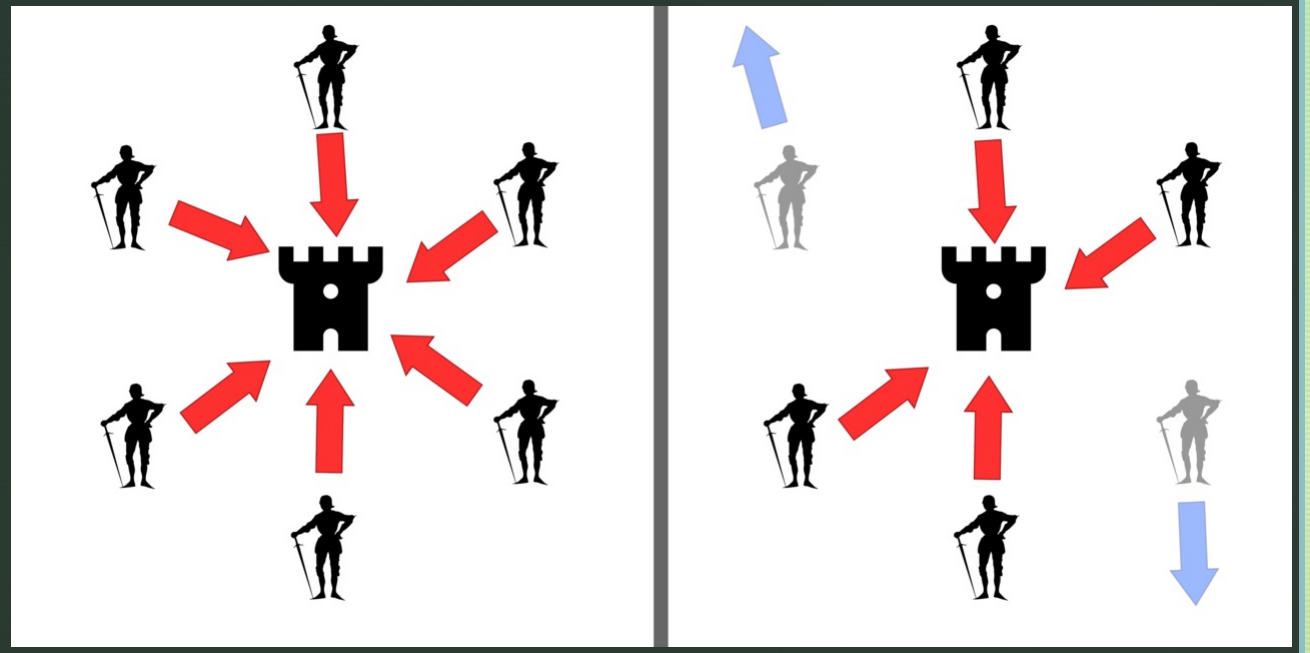
- É impossível garantir consenso entre as partes de um sistema 100% do tempo.
- Consenso entre aplicações demanda suposições, expondo o sistema a riscos e falhas.
- Os mecanismos de consenso centralizados geram um ponto único de possível falha sistêmica.

Soluções para garantir consenso

- Escolha a metodologia de consenso cuidadosamente para seu caso de uso.
- Determine se sua aplicação considera escrita ou leitura mais importante.
- Não confie na rede e nem em relógios de sistema.
- Assuma e documente todos os riscos que foram aceitos em sua arquitetura.
- Documente e prepare processos manuais para solucionar problemas previstos.
- Garanta a observabilidade do estado dos dados na sua aplicação através de métricas e logs.
- Tenha em mente que se você fez e aceitou uma suposição, ela provavelmente irá se provar verdadeira ou falsa eventualmente, para bom ou para mal.

Falhas Bizantinas

- Falhas bizantinas acontecem quando um sistema possui partes não confiáveis que parecem confiáveis a observadores externos.
- Resulta em dados inconsistentes ou incompletos, causados pelas partes não cofiáveis.
- Difícil de identificar.
- Prejuízo principalmente na confiança de seu usuário no sistema.



Contratos

- Estabelece um padrão de comunicação entre as partes de um sistema distribuido.
- Define a linguagem e formato de grande parte dos dados de um sistema.
- Contratos bem formados evitam problemas de integração entre as partes.
- Bons contratos são retro compativeis.
- Um contrato deve estabelecer uma interface de comunicação, mas não necessariamente forçar um meio de transporte. Isso deve ser responsabilidade de um conector abstraído do restante do contexto.

Coordenação dos sistemas

- A coordenação pode ser feita por um líder entre diversos nós do sistema, ou caso se trate de um sistema de aplicações especialistas, um coordenador externo pode ser necessário.
- Coordenadores são responsáveis por garantir e orquestrar o trabalho de todas as partes do sistema.
- O coordenador pode representar um ponto único de falha.
- Normalmente não é necessário para sistema distribuídos organizados em formato de pipeline.

Garantias transacionais

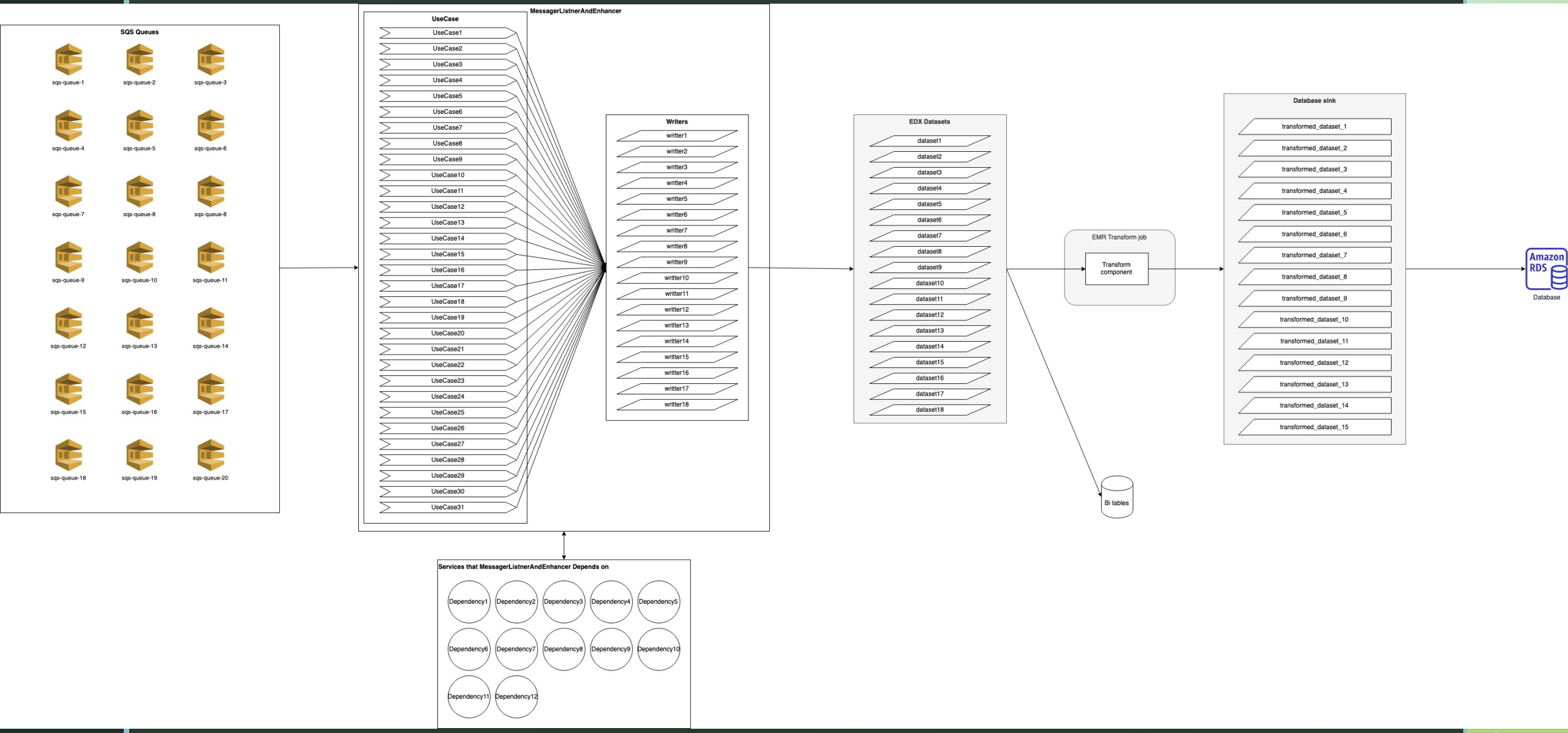
- O status de cada transação feita para o sistema deve ser propagado para todo o sistema.
- Caso uma transação falhe no meio do caminho, todas as partes relevantes precisam ser notificadas para processar de acordo.
- **Sempre** procure utilizar uma chave de impotência comum em todos os componentes do sistema para evitar reprocessamento e possíveis resultados não determinísticos.

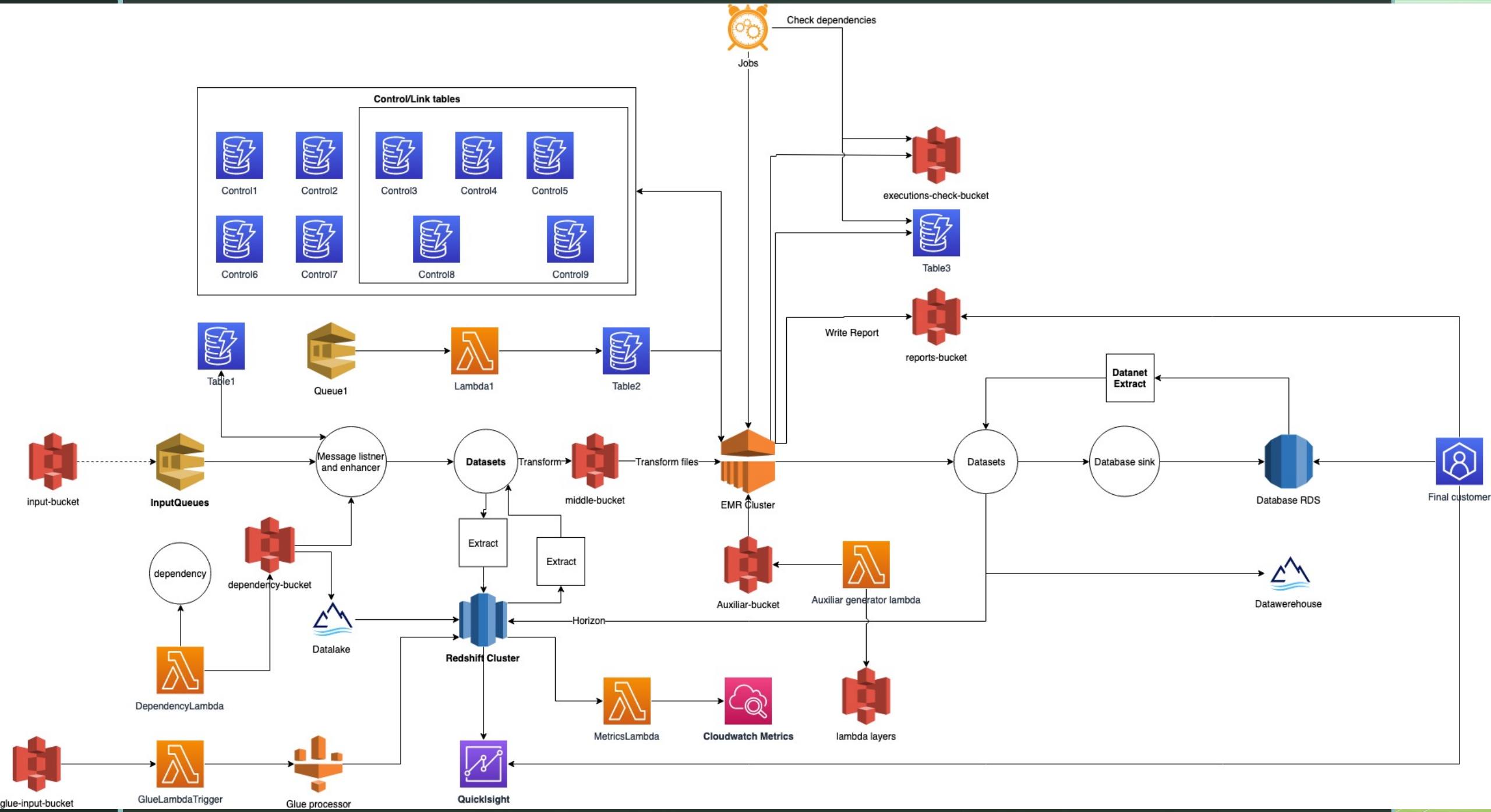
Boas práticas ao trabalhar com sistemas distribuídos

- Pense com cuidado em seu caso de uso, você realmente precisa de um sistema distribuído?
- Planeje com cuidado, grandes mudanças após a conclusão do projeto são complicadas e custosas.
- Define contratos fortes e auto contidos, não permita furos em contratos por conveniência.
- **Evite integrar as partes do sistema distribuído utilizado datasets. São difíceis de manter, atualizar e é uma opção mais cara de integração.**
- Prefira aplicações gerenciadas em vez de configurar novas por conta própria, são mais rápidas e mais confiáveis de se utilizar.
- Define metas e SLAs para disponibilidade e benchmarks, monitore métricas e se mantenha dentro de suas SLAs sempre que possível.
- Trabalhe para garantir uma boa observabilidade do seu sistema, isso facilitará o debug, profiling e compreensão do funcionamento do sistema como um todo.
- Define fortemente a responsabilidade de cada parte do sistema e evite ao máximo romper o escopo dessas responsabilidades.

Um sistema processador de mensagens

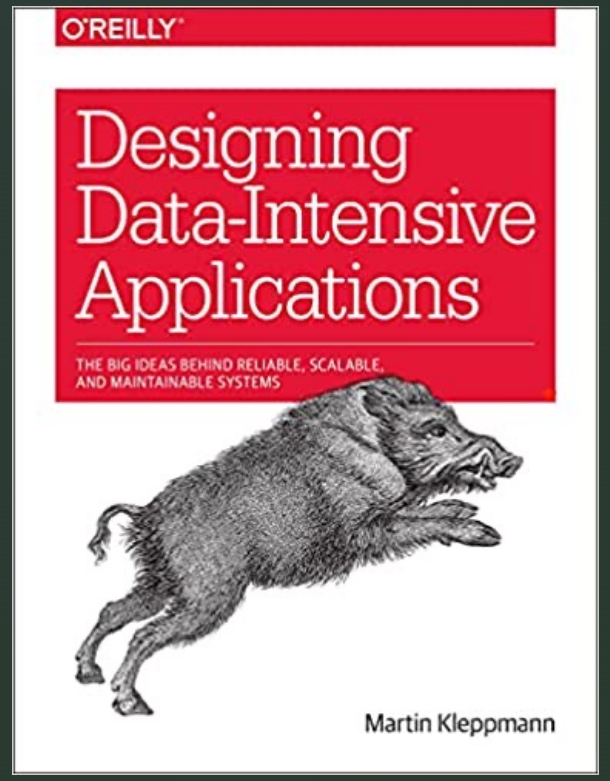
A realidade corporativa em escala

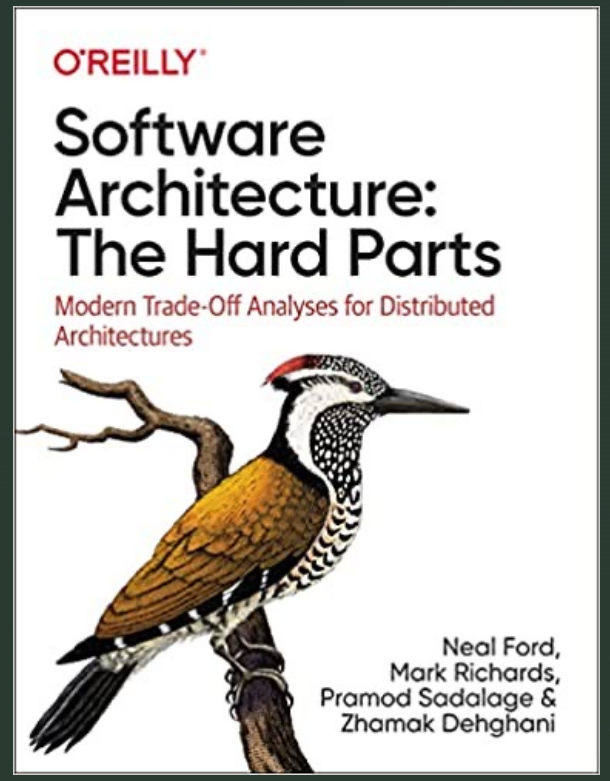

SQS Queues
sqs-queue-1
sqs-queue-2
sqs-queue-3
sqs-queue-4
sqs-queue-5
sqs-queue-6
sqs-queue-7
sqs-queue-8
sqs-queue-8
sqs-queue-9
sqs-queue-10
sqs-queue-11
sqs-queue-12
sqs-queue-13
sqs-queue-14
sqs-queue-15
sqs-queue-16
sqs-queue-17
sqs-queue-18
sqs-queue-19
sqs-queue-20
MessagerListnerAndEnhancer
UseCase
UseCase1
UseCase2
UseCase3
UseCase4
UseCase5
UseCase6
UseCase7
UseCase8
UseCase9
UseCase10
UseCase11
UseCase12
UseCase13
UseCase14
UseCase15
UseCase16
UseCase17
UseCase18
UseCase19
UseCase20
UseCase21
UseCase22
UseCase23
UseCase24
UseCase25
UseCase26
UseCase27
UseCase28
UseCase29
UseCase30
UseCase31
Writers
writter1
writter2
writter3
writter4
writter5
writter6
writter7
writter8
writter9
writter10
writter11
writter12
writter13
writter14
writter15
writter16
writter17
writter18
Services that MessagerListnerAndEnhancer Depends on
Dependency1
Dependency2
Dependency3
Dependency4
Dependency5
Dependency6
Dependency7
Dependency8
Dependency9
Dependency10
Dependency11
Dependency12
EDX Datasets
dataset1
dataset2
dataset3
dataset4
dataset5
dataset6
dataset7
dataset8
dataset9
dataset10
dataset11
dataset12
dataset13
dataset14
dataset15
dataset16
dataset17
dataset18
EMR Transform job
Transform component
Bi tables
Database sink
transformed_dataset_1
transformed_dataset_2
transformed_dataset_3
transformed_dataset_4
transformed_dataset_5
transformed_dataset_6
transformed_dataset_7
transformed_dataset_8
transformed_dataset_9
transformed_dataset_10
transformed_dataset_11
transformed_dataset_12
transformed_dataset_13
transformed_dataset_14
transformed_dataset_15
Amazon RDS
Database

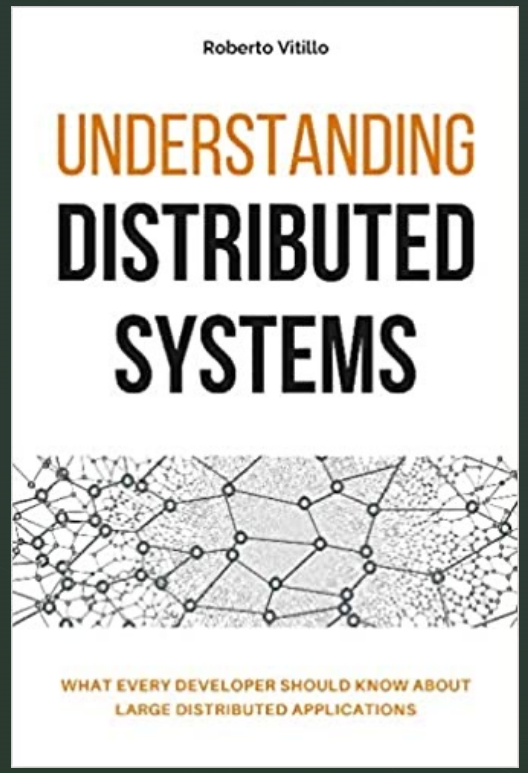

Jobs
Check dependencies
Control/Link tables
Control1
Control2
Control3
Control4
Control5
Control6
Control7
Control8
Control9
executions-check-bucket
Table3
Write Report
reports-bucket
Table1
Queue1
Lambda1
Table2
Datanet Extract
input-bucket
InputQueues
Message listner and enhancer
Datasets
Transform
middle-bucket
Transform files
EMR Cluster
Datasets
Database sink
Database RDS
Final customer
dependency
dependency-bucket
Extract
Extract
Auxiliar-bucket
Auxiliar generator lambda
Datawerehouse
Datalake
Horizon
Redshift Cluster
DependencyLambda
MetricsLambda
Cloudwatch Metrics
lambda layers
glue-input-bucket
GlueLambdaTrigger
Glue processor
Quicksight

Hands-on

Sugestões bibliográficas







Q&A