Doutrina Operacional Sistemas Verificáveis

VERIFY SYSTEMS

Modelos de Produção Operacional

Build 2026.01
Princípio PROVA_>_CRENÇA
Restrição REALIDADE_VENCE

Renan Melo

Autor / Sistemas

Nota de Edição

Este documento descreve doutrina operacional para sistemas que operam sob falha parcial, latência, concorrência e verdade externa. Não é um manual de linguagem, framework ou stack.

O objetivo é transformar *execução* em *evidência* e transformar *estado* em *conhecimento operacional*.

Como Ler

- **AXIOMA**: afirmação estrutural (não depende de implementação).
- **ENGENHARIA**: mecanismo concreto (implementável / auditável).
- **ALERTA**: ponto onde sistemas normalmente mentem em produção.

Contents

Part I

FUNDAMENTOS OPERACIONAIS

Chapter 1

Introdução: O Problema da Produção

Software raramente falha onde foi projetado.

Durante o desenvolvimento, o sistema existe como descrição. Fluxos são lineares, estados são imediatos e causa e efeito parecem próximos.

Em produção, essa linearidade desaparece.

Eventos chegam fora de ordem. Dependências tornam-se instáveis. Processos reiniciam entre etapas. Efeitos externos ocorrem sem confirmação local.

O sistema continua executando mas deixa de saber.

O erro fundamental não é técnico. É epistemológico.

Sistemas falham não apenas porque executam incorretamente, mas porque assumem conhecimento que não possuem.

AXIOMA

Código correto não garante sistema correto.

O deploy não encerra o problema. Ele inicia o ambiente real onde o sistema passa a existir.

Produção é um ambiente hostil porque remove garantias implícitas:

- execução contínua não é garantida;
- ordem de eventos não é garantida;
- confirmação imediata não é garantida;
- observabilidade completa não é garantida.

Nesse ambiente, execução deixa de ser evidência.

Logs indicam intenção. Estados locais indicam interpretação. Somente evidência externa aproxima o sistema da realidade.

ENGENHARIA

Sistemas confiáveis separam explicitamente:

- decisão registrada,
- execução do efeito,
- verificação independente do resultado.

Quando essas etapas colapsam, o sistema perde a capacidade de explicar o próprio estado.

A maior parte da engenharia moderna ainda é orientada à implementação: como executar corretamente um fluxo.

Produção exige outra pergunta:

como continuar correto após interrupções inevitáveis?

Esse documento parte de uma mudança de perspectiva.

O problema não é impedir falhas.

O problema é construir sistemas que permanecem governáveis depois que elas acontecem.

ALERTA

Sistemas raramente quebram imediatamente. Eles entram em estados onde continuam executando, mas deixam de convergir para a realidade.

O objetivo deste livro não é ensinar arquitetura específica, linguagem ou framework.

O objetivo é redefinir o problema operacional:

- execução não produz verdade;
- tempo introduz divergência;
- conhecimento chega depois da ação;
- consistência precisa ser restaurada continuamente.

A partir deste ponto, produção deixa de ser vista como etapa final. Ela passa a ser o ambiente onde a verificação realmente começa.

INVARIANTES OPERACIONAIS

- Execução não é prova: sucesso só existe quando há evidência observável.
- Estado interno é interpretação e permanece revisável até a verificação externa.
- Ausência de confirmação não pode ser tratada como falha.
- Reinício é continuidade válida e deve ser previsto como comportamento normal.

Chapter 2

Axiomas de Sistemas Verificáveis

Todo sistema operacional maduro converge para o mesmo conjunto de limitações.

Essas limitações não surgem de tecnologia específica. Elas emergem da forma como sistemas interagem com o mundo.

Os axiomas abaixo não são recomendações. São propriedades observadas em sistemas que sobrevivem ao tempo.

AXIOMA

O sistema não sabe.

O sistema preserva estados. Conhecimento surge apenas quando esses estados permitem inferência confiável.

AXIOMA

Execução não é confirmação.

Uma operação pode ter sido executada sem que o efeito tenha ocorrido. Uma operação pode ter ocorrido sem que o sistema tenha observado.

Execução e realidade são eventos distintos.

AXIOMA

Estado local é hipótese.

Todo estado interno representa uma interpretação temporária. Ele permanece válido apenas até que evidência externa o contradiga.

AXIOMA

Falha é estado possível.

Falha não é exceção. É parte do espaço de estados do sistema.

Sistemas que não representam falha explicitamente apenas a deslocam para fora do modelo.

AXIOMA

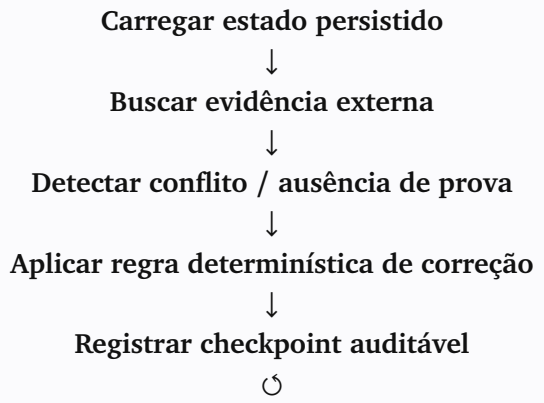
Consistência é restaurada, não mantida.

Sistemas distribuídos operam em divergência temporária. Convergência é resultado de reconciliação contínua.

AXIOMA

Recuperabilidade é superior à perfeição.

DIAGRAMA OPERACIONAL



Sistemas perfeitos falham ao primeiro evento inesperado. Sistemas recuperáveis continuam operando após erro parcial.

ENGENHARIA

Esses axiomas produzem uma consequência direta:
o sistema deve sempre ser capaz de reavaliar sua posição sem depender da continuidade da execução anterior.

Quando essa propriedade existe, reinicialização deixa de ser falha e passa a ser continuidade.

INVARIANTES OPERACIONAIS

- O sistema nunca sabe: ele apenas acumula estados que permitem inferência.
- Toda afirmação interna é provisória até ser confirmada ou contradita por evidência.
- Falha é estado possível e deve existir no modelo, não fora dele.
- Consistência não é mantida; é restaurada continuamente por reconciliação.
- Recuperabilidade tem prioridade sobre perfeição e linearidade.

Chapter 3

O Modelo Verify de Produção

À medida que sistemas crescem, fluxos lineares deixam de ser suficientes para explicar o comportamento real.

Arquiteturas diferentes convergem para o mesmo padrão operacional:

Evento → Execução → Estado Local → Evidência → Reconciliação → Estado Corrigido

Esse fluxo não descreve um pipeline. Ele descreve um ciclo.

Hipótese vs Verdade

O sistema inicia operando sob hipótese.

Um comando recebido representa intenção. A execução produz tentativa. O estado local registra interpretação.

A verdade surge apenas quando evidência externa é observada.

VERIFY LOOP

Decisão (registrada)
↓
Execução (tentativa)
↓
Estado Local (hipótese)
↓
Evidência Externa (verdade)
↓
Reconciliação (correção)
↓
Estado Corrigido
↺

ALERTA

Quando o sistema trata execução como evidência, ele perde a capacidade de distinguir sucesso de tentativa.

Latência de Conhecimento

Existe sempre um intervalo entre:

- executar um efeito;
- saber que o efeito ocorreu.

Essa distância é estrutural. Ela não pode ser eliminada por otimização.

Latência transforma sistemas determinísticos em sistemas epistemicamente incompletos.

Verdade Eventual Operacional

Consistência operacional não é instantânea.

Estados evoluem conforme novas informações chegam. O sistema deve permanecer capaz de corrigir decisões anteriores.

ENGENHARIA

Reconciliação não corrige erros. Ela corrige falta de conhecimento.

O Ciclo Infinito de Verificação

Verificação não encerra o fluxo. Ela reinicia o ciclo.

Após reconciliação:

- novos eventos são produzidos;
- novos estados são derivados;
- nova verificação torna-se necessária.

Sistemas verificáveis não convergem para certeza permanente.

Eles convergem para explicabilidade contínua.

A partir deste ponto, o sistema deixa de ser entendido como sequência de operações.

Ele passa a ser entendido como processo contínuo de aproximação da realidade.

INVARIANTES OPERACIONAIS

- Toda decisão deve ser registrada antes de qualquer efeito externo.
- Toda execução produz hipótese; a verdade só emerge após evidência externa.
- Nenhum estado final é alcançado sem prova suficiente do resultado.
- Reconciliação é o mecanismo que corrige falta de conhecimento, não bugs.
- Verificar é um ciclo: o sistema deve permanecer reavaliável ao longo do tempo.

Part II

ESTADOS OPERACIONAIS DO SISTEMA

Chapter 4

Sistemas Como Máquinas de Estado Operacional

Todo sistema em produção opera sob condições variáveis.

Durante o desenvolvimento, o sistema é normalmente descrito em termos de domínio: ordens, pagamentos, swaps, usuários, transações.

Em produção, essa descrição é insuficiente.

O sistema não existe apenas como domínio. Ele existe como comportamento sob condições operacionais.

Para tornar esse comportamento observável, é necessário distinguir três camadas diferentes de estado:

DIAGRAMA OPERACIONAL

NORMAL

↓ (sinais de instabilidade / risco subindo)

DEGRADADO

↓ (divergência detectada / atraso de prova)

RECONCILIANDO

↓ (verdade desconhecida / prova indisponível)

SAFE

Regra: quanto menor a prova, menor a permissão de executar efeitos irreversíveis.

- **Estado de domínio**: o que o sistema representa semanticamente.
- **Estado técnico**: o que os componentes internos estão executando.
- **Estado operacional**: em que condição o sistema está operando.

Essas camadas frequentemente divergem.

Um sistema pode possuir estados de domínio corretos e ainda assim estar operacionalmente degradado. Da mesma forma, um sistema pode estar tecnicamente saudável enquanto acumula divergência invisível.

AXIOMA

O sistema nunca está neutro. Ele sempre está operando sob um modo operacional.

A ausência explícita desse modelo produz ambiguidade.

Operadores passam a interpretar sinais isolados. Decisões tornam-se subjetivas. Intervenções ocorrem cedo demais ou tarde demais.

Modelar o estado operacional torna o comportamento do sistema previsível mesmo sob falha.

ENGENHARIA

O estado operacional não descreve o que o sistema faz. Ele descreve como o sistema deve se comportar enquanto faz.

Essa distinção permite que decisões automáticas sejam tomadas sem depender de interpretação humana contínua.

Sistemas maduros não perguntam apenas:

o fluxo funcionou?

Eles perguntam:

em que condição operacional estamos?

INVARIANTES OPERACIONAIS

- O sistema sempre opera em um modo operacional; neutro não existe.
- Estado de domínio, estado técnico e estado operacional podem divergir e devem ser tratados separadamente.
- Mudança de modo altera permissões e comportamento, não apenas alertas.
- O sistema deve expor estados intermediários para permanecer governável.

Chapter 5

Os Quatro Modos Universais

A observação de sistemas financeiros, distribuídos e orientados a eventos revela um padrão recorrente.

Independentemente da tecnologia, sistemas convergem para quatro modos operacionais fundamentais.

Esses modos não são estados de erro. São estados normais de operação ao longo do tempo.

NORMAL

O sistema executa automaticamente.

Dependências estão estáveis. Reconciliação ocorre continuamente. Divergência existe, mas permanece dentro de limites aceitáveis.

Características:

- execução automática habilitada;
- throughput máximo permitido;
- reconciliação em segundo plano;
- intervenção humana não necessária.

DEGRADED

O sistema detecta instabilidade externa ou interna.

O objetivo deixa de ser performance e passa a ser preservação de estado.

Características:

- dependências instáveis;
- aumento de latência ou retries;

- redução automática de throughput;
- bloqueio de operações irreversíveis.

ALERTA

Sistemas frágeis permanecem em NORMAL mesmo quando sinais de degradação já existem. Esse é o ponto onde incidentes começam.

RECONCILIATION MODE

Divergência foi detectada.

A prioridade deixa de ser executar novas operações e passa a ser restaurar consistência.

Características:

- reconciliação ativa com fontes externas;
- pausas seletivas de execução;
- revalidação de estados intermediários;
- redução de novas decisões.

SAFE MODE

O sistema não consegue afirmar seu próprio estado.

A continuidade automática torna-se risco.

Características:

- estado desconhecido;
- execução automática suspensa;
- intervenção humana requerida;
- preservação máxima de evidência.

FALHA CRÍTICA

SAFE MODE não é falha. É mecanismo de sobrevivência.

Transições Entre Modos

Transições não devem depender de decisão manual arbitrária.

Elas devem ser derivadas de sinais observáveis:

- aumento sustentado de divergência;
- perda de confirmação externa;
- falhas repetidas de reconciliação;
- perda de observabilidade confiável.

Quando esses critérios são explícitos, o sistema passa a reagir antes que o incidente se torne irreversível.

INVARIANTES OPERACIONAIS

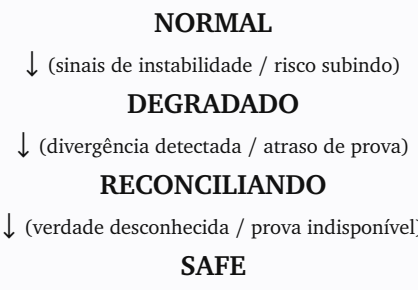
- Em SAFE, efeitos irreversíveis ficam bloqueados.
- Em SAFE, a preservação de evidência é máxima.
- Em SAFE, apenas operações reconciliáveis são permitidas.
- Toda ação manual exige registro auditável.
- A saída de SAFE exige checkpoint de reconciliação concluído.

Sinais Mensuráveis (Telemetria de Modo)

Defina sinais que o sistema mede continuamente. Exemplos implementáveis:

- **pending_external_confirmations**: contagem de operações com efeito externo sem confirmação observada.
- **reconciliation_lag_seconds**: tempo desde o último ciclo de reconciliação bem-sucedido.
- **truth_conflict_rate**: taxa de conflitos entre ledger interno e fonte externa por janela de tempo.
- **job_replay_rate**: proporção de reexecuções sobre jobs totais.
- **dependency_health_score**: score de saúde de dependências críticas (RPC, nodes, gateways).

DIAGRAMA OPERACIONAL



Regra: quanto menor a prova, menor a permissão de executar efeitos irreversíveis.

Regras de Transição (Exemplo Concreto)

As regras abaixo não são monitoramento. São condições que mudam comportamento.

ENGENHARIA

Exemplo de política de transição (ajuste N/T conforme domínio):

- **NORMAL → DEGRADED** se

$$pending\_external\_confirmations > N \text{ ou } dependency\_health\_score < S \text{ por } T$$

- **DEGRADED → RECONCILING** se

$$truth\_conflict\_rate > C \text{ ou } reconciliation\_lag\_seconds > L$$

- **RECONCILING → SAFE** se

$$unknown\_truth == true \text{ ou } reconciliation\_fails\_consecutive > K$$

- **SAFE → RECONCILING** quando

$$evidence\_channel\_restored == true \ \wedge \ operator\_ack == true$$

- **RECONCILING → NORMAL** quando

$$truth\_conflict\_rate \approx 0 \ \wedge \ pending\_external\_confirmations \leq n \text{ por } t$$

ALERTA

A transição não é alerta. É **mudança de contrato**. Modo operacional define o que é permitido executar.

INVARIANTES OPERACIONAIS

- Modo operacional define o que é permitido executar, não apenas o que é observado.
- Degradação reduz superfície de risco antes de ocorrer falha total.
- Reconciliação tem prioridade sobre novas decisões quando há divergência detectada.
- SAFE é acionado quando o sistema não consegue provar seu próprio estado.
- Transições de modo devem ser derivadas de sinais mensuráveis, não de julgamento arbitrário.

Chapter 6

Failure Budget como Estado

Failure budget é frequentemente tratado como métrica organizacional.

Em sistemas verificáveis, ele deve ser tratado como variável operacional.

Failure budget representa a quantidade de incerteza que o sistema pode absorver antes de alterar seu comportamento.

AXIOMA

Risco é uma variável do sistema.

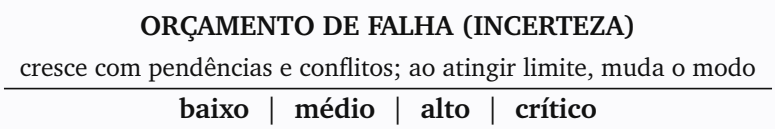
À medida que falhas parciais se acumulam:

- latência aumenta;
- retries aumentam;
- divergência cresce;
- confiança operacional diminui.

O erro comum é tratar esses sinais como eventos isolados.

Na prática, eles representam consumo progressivo do orçamento de falha.

DIAGRAMA OPERACIONAL



Failure Budget Operacional

Um failure budget operacional pode ser expresso como combinação de:

- taxa de reconciliação pendente;
- operações sem confirmação externa;
- divergência entre ledger interno e fonte de verdade;
- degradação de dependências críticas.

Quando limites são atingidos, o sistema deve alterar automaticamente seu modo operacional.

ENGENHARIA

Failure budget não define quando o sistema falha. Define quando o sistema deve mudar de comportamento.

Redução de Throughput

Reduzir throughput é frequentemente interpretado como perda de performance.

Operacionalmente, é mecanismo de estabilização.

Menos novas decisões significam menor expansão do espaço de estados enquanto o sistema converge.

Quando Parar Automaticamente

A parada automática torna-se correta quando:

- evidência externa não pode ser obtida;
- reconciliação não converge;
- o sistema não consegue distinguir sucesso de falha.

Nesse ponto, continuar executando aumenta o custo da recuperação futura.

Sistemas maduros aceitam interrupção temporária para preservar coerência global.

Failure budget transforma essa decisão em mecanismo, não em julgamento humano.

INVARIANTES OPERACIONAIS

- Risco é variável operacional e deve alterar comportamento automaticamente.
- O orçamento de falha é consumido por incerteza acumulada, não apenas por erros.
- Ao degradar, o sistema deve reduzir throughput para conter expansão do espaço de estados.
- Quando a prova não é obtida, parar é preferível a ampliar divergência.

Part III

VERDADE OPERACIONAL

Chapter 7

Hierarquia de Verdade (Source of Truth)

Em sistemas reais, múltiplas representações do mesmo estado coexistem.

O ledger interno possui uma versão. O node possui outra. Sistemas externos possuem outra. Logs contam uma narrativa adicional.

Essas representações inevitavelmente divergem.

O erro fundamental não é a divergência. O erro é não definir qual representação prevalece quando há conflito.

AXIOMA

Nem toda informação possui o mesmo peso operacional.

Para que sistemas permaneçam verificáveis, é necessário estabelecer uma hierarquia explícita de verdade.

Uma hierarquia típica em sistemas financeiros e distribuídos é:

1. Settlement externo (blockchain, clearing, rede de liquidação)
2. Estado observado do node
3. Ledger interno
4. Execução de jobs
5. Logs e métricas

Cada camada representa uma proximidade diferente da realidade externa.

DIAGRAMA OPERACIONAL

Verdade mais forte (vence conflitos)
Liquidação externa (settlement)
↓
Estado observado no node / rede
↓
Ledger interno
↓
Execução de jobs
↓
Logs e métricas (narrativa / sinais)
Verdade mais fraca

Resolução de Conflitos

Quando duas fontes divergem, o sistema não deve tentar reconciliar arbitrariamente.

Ele deve ascender na hierarquia.

Exemplo:

- ledger indica falha;
- blockchain indica confirmação;

o estado interno deve ser corrigido.

ALERTA

Logs nunca são fonte de verdade. Eles são apenas registro da interpretação do sistema em um momento específico.

Quando Esperar vs Quando Reverter

Nem toda divergência exige ação imediata.

O sistema deve distinguir entre:

- ausência temporária de conhecimento;
- evidência contraditória.

No primeiro caso, o comportamento correto é aguardar. No segundo, é corrigir.

ENGENHARIA

Reconciliação eficaz depende mais de saber quando não agir do que de agir rapidamente.

A hierarquia de verdade transforma decisões operacionais em regras explícitas, reduzindo dependência de julgamento humano.

INVARIANTES OPERACIONAIS

- Conflitos entre fontes devem ser resolvidos por hierarquia explícita de verdade.
- A fonte mais próxima do settlement prevalece sobre interpretações internas.
- Logs e métricas não são fonte de verdade; são narrativa e sinais.
- Ausência de evidência é estado de espera; evidência contraditória exige correção.

Chapter 8

Reconciliação como Processo Primário

Sistemas tradicionais tratam reconciliação como tarefa secundária.

Ela é executada periodicamente, após a execução principal.

Sistemas verificáveis invertem essa relação.

AXIOMA

Reconciliação é o processo primário. Execução é apenas geração de hipóteses.

Toda execução cria possibilidade de divergência:

- mensagens podem ser perdidas;
- eventos podem chegar fora de ordem;
- confirmações podem atrasar;
- processos podem reiniciar entre etapas.

Reconciliação existe para restaurar alinhamento entre:

- estado interno;
- evidência externa;
- histórico observável.

DIAGRAMA OPERACIONAL

Carregar estado persistido
↓
Buscar evidência externa
↓
Detectar conflito / ausência de prova
↓
Aplicar regra determinística de correção
↓
Registrar checkpoint auditável
↺

Reconciliação Contínua vs Periódica

Reconciliação periódica assume estabilidade entre execuções.

Esse modelo falha em sistemas de alto valor ou alta concorrência.

Reconciliação contínua assume divergência constante.

O sistema permanece permanentemente em processo de verificação.

Reconciliação Determinística

Reconciliação não deve depender de interpretação humana.

Dado o mesmo estado e a mesma evidência externa, o resultado deve ser sempre o mesmo.

Isso exige:

- estados explícitos;
- transições auditáveis;
- regras de correção determinísticas.

ENGENHARIA

Reconciliação não conserta o sistema. Ela remove suposições incorretas acumuladas ao longo do tempo.

Detecção de Divergência

Divergência ocorre quando:

- estado interno não pode ser comprovado externamente;

- evidência externa contradiz estado interno;
- estados intermediários permanecem indefinidos por tempo excessivo.

A detecção precoce reduz o custo de correção.

Exemplos Operacionais

Em sistemas financeiros:

- swaps parcialmente executados;
- canais LN fora de sincronização;
- withdrawals sem confirmação observada;

o erro não é a falha inicial.

O erro é permitir que divergência permaneça invisível.

INVARIANTES OPERACIONAIS

- Reconciliação é processo primário; execução é geração de hipóteses.
- Reconciliação deve ser determinística: mesmo estado + mesma evidência mesmo resultado.
- Divergência deve ser detectável por tempo em estados intermediários e por conflitos de prova.
- A reconciliação deve sobreviver a reinícios sem depender da memória do processo.

Chapter 9

Idempotência e Reexecução Segura

Tempo remove a garantia de execução única.

Retries são inevitáveis. Processos reiniciam. Mensagens são reenviadas.

A pergunta correta deixa de ser:

como evitar execução duplicada?

e passa a ser:

como tornar repetição segura?

AXIOMA

Reexecução é inevitável. Efeitos duplicados não são.

Efeitos Irreversíveis

Operações que produzem efeitos externos não podem depender da memória da execução.

Exemplos:

- transferências financeiras;
- broadcast de transações;
- liquidação entre redes independentes.

Se o sistema cair após executar o efeito, a única forma de continuidade é reavaliar a partir do estado persistido.

Chaves de Idempotência

Idempotência surge da identidade da operação.

Uma operação precisa responder à pergunta:

o que torna esta execução única?

Sem identidade explícita:

- retries tornam-se novas operações;
- reconciliação torna-se ambígua;
- efeitos externos podem duplicar.

Replay Seguro

Replay seguro permite que o sistema:

- execute novamente sem produzir novo efeito;
- recuperar após falha parcial;
- reconstruir estado após reinicialização.

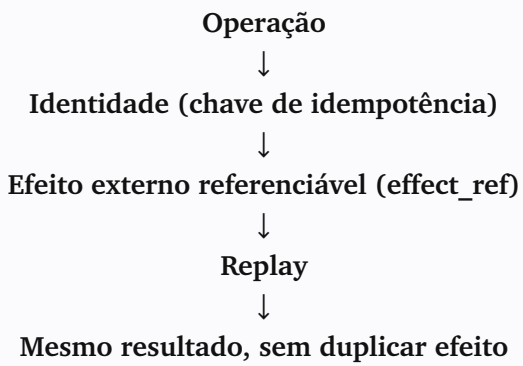
ENGENHARIA

Idempotência não é otimização. É requisito para operar sob tempo e falha parcial.

Sistemas verificáveis não tentam impedir repetição.

Eles garantem que repetição não altere o resultado final.

DIAGRAMA OPERACIONAL



INVARIANTES OPERACIONAIS

- Reexecução é inevitável; efeitos duplicados não são aceitáveis.
- Toda operação com efeito externo deve ter identidade e chave de idempotência.
- A repetição deve ser segura sem depender do estado em memória do worker.
- O sistema deve conseguir reconstruir sua posição a partir do estado persistido.

Part IV

SISTEMAS SOB FALHA

Chapter 10

Taxonomia de Falhas em Produção

Falhas em produção raramente são eventos isolados.

Elas emergem da interação entre componentes corretos operando sob condições inesperadas.

A tentativa de eliminar falhas leva a sistemas frágeis. A classificação de falhas produz sistemas governáveis.

AXIOMA

Falha não é exceção. É parte do espaço operacional do sistema.

Sem taxonomia, toda falha parece única. Toda resposta torna-se improvisação.

Com taxonomia, falhas tornam-se reconhecíveis.

Falha de Rede

A rede introduz incerteza fundamental:

- mensagens podem atrasar;
- respostas podem se perder;
- confirmações podem chegar após timeout.

O erro comum é interpretar ausência de resposta como evidência de falha.

Em sistemas distribuídos, ausência de resposta significa apenas ausência de conhecimento.

Falha de Ordenação

Eventos raramente chegam na ordem em que ocorreram.

Filas, retries e concorrência removem sequencialidade.

Estados impossíveis surgem quando o sistema assume ordem perfeita.

ALERTA

Ordem observada não é ordem real.

Falha de Confirmação

Execução ocorreu, mas evidência não foi registrada.

Ou evidência existe, mas não foi observada.

Esse tipo de falha é particularmente perigoso porque o sistema continua operando enquanto diverge silenciosamente.

Falha Humana

Intervenções humanas ocorrem sob informação incompleta.

Correções manuais frequentemente introduzem novos estados inconsistentes quando não seguem regras operacionais explícitas.

Falha de Observabilidade

O sistema pode continuar correto e ainda assim tornar-se impossível de operar.

Quando estados não são observáveis:

- operadores intervêm cedo demais;
- automações tomam decisões prematuras;
- reconciliação torna-se impossível.

ENGENHARIA

O objetivo da taxonomia não é evitar falhas. É permitir resposta previsível quando elas ocorrem.

INVARIANTES OPERACIONAIS

- Reexecução é inevitável; efeitos duplicados não são aceitáveis.
- Toda operação com efeito externo deve ter identidade e chave de idempotência.
- A repetição deve ser segura sem depender do estado em memória do worker.
- O sistema deve conseguir reconstruir sua posição a partir do estado persistido.

Chapter 11

Sistemas Degradados

A maioria dos incidentes graves não começa com falha total.

Eles começam com degradação progressiva.

Latência aumenta. Retries se acumulam. Dependências tornam-se intermitentes.

O sistema continua executando, mas sua capacidade de convergir diminui.

AXIOMA

Degradação controlada é mecanismo de sobrevivência.

Sistemas maduros não tentam manter comportamento normal sob condições anormais.

Eles mudam de comportamento.

Graceful Degradation

Degradação graciosa significa reduzir o escopo de risco enquanto mantém continuidade.

Exemplos:

- suspender operações irreversíveis;
- limitar criação de novos estados;
- priorizar reconciliação sobre execução.

Redução de Superfície de Risco

Cada nova operação expande o espaço de estados possível.

Durante degradação, o objetivo é impedir expansão adicional.

Menos decisões novas significam menos divergência futura.

Pausas Seletivas

Parar completamente é raramente necessário.

Sistemas podem:

- permitir depósitos, mas bloquear saques;
- permitir leitura, mas suspender escrita;
- continuar reconciliação enquanto pausam execução.

Essa separação exige arquitetura desacoplada entre decisão e efeito.

ALERTA

O erro comum é tratar degradação como falha temporária. Na prática, ela é estado operacional legítimo.

INVARIANTES OPERACIONAIS

- Degradação é estado operacional legítimo, não quase falha.
- Em degradação, o objetivo é preservar evidência e conter risco, não otimizar performance.
- Operações irreversíveis devem ser bloqueadas primeiro.
- Pausas devem ser seletivas sempre que possível (ler/reconciliar continua).

Chapter 12

Safe Mode e Parada Segura

Existe um ponto onde continuar executando aumenta o custo da recuperação.

Esse ponto define a entrada em SAFE MODE.

AXIOMA

Quando o sistema não consegue afirmar seu próprio estado, parar é comportamento correto.

SAFE MODE não representa falha técnica. Representa perda de certeza operacional.

SAFE MODE como Mecanismo

SAFE MODE existe para impedir que o sistema amplifique erro quando o estado real não pode ser inferido.

O sistema não entra em SAFE porque deu problema. Ele entra em SAFE porque perdeu capacidade de provar.

INVARIANTES OPERACIONAIS

- safe_mode ⇒write_effects == disabled
- safe_mode ⇒evidence_preservation == max
- safe_mode ⇒only_reconcilable_ops == allowed
- manual_action ⇒audit_record == required
- exit_safe ⇒reconciliation_checkpoint == complete

Critérios Automáticos (Mensuráveis)

SAFE MODE deve ser derivado de sinais observáveis. Exemplos:

- **unknown_truth**: true quando o sistema não consegue afirmar qual fonte venceu.
- **truth_conflicts_unresolved**: conflitos persistentes por mais de `T` minutos.
- **stuck_intermediate_states**: operações em estado intermediário por mais de `TTL`.
- **evidence_channel_down**: perda sustentada de acesso às fontes externas de evidência (RPC/node).
- **double-spend / replay suspicion**: evidência de duplicidade potencial de efeitos externos.

ENGENHARIA

Entrada em SAFE MODE (exemplo determinístico):

- Se `unknown_truth == true` por `T`
- ou se `truth_conflicts_unresolved > 0` e `reconciliation_fails_consecutive > K`
- ou se `stuck_intermediate_states > M` por `TTL`

então: `mode := SAFE` e `irreversible_ops := disabled`.

O que é Bloqueado em SAFE

SAFE MODE não é parar tudo. É congelar o que é irreversível e preservar evidência.

- Bloquear: broadcast/settlement, withdrawals, commits externos.
- Permitir: leitura, inspeção, reconciliação, replay idempotente.
- Prioridade: reduzir ambiguidade e restaurar prova.

Recuperação (Saída de SAFE)

A saída de SAFE só ocorre quando o sistema consegue provar novamente seu estado.

ENGENHARIA

Saída de SAFE (contrato):

- `evidence_channel_restored == true`
- `reconciliation_checkpoint == complete`
- `pending_external_confirmations ≤n`
- `truth_conflict_rate ≈0` por `t`
- `operator_ack == true` (para estados ambíguos)

FALHA CRÍTICA

SAFE MODE é sucesso operacional: o sistema escolheu sobreviver em vez de parecer funcionando.

Recuperação Após Pausa

Recuperação não começa retomando execução.

Ela começa reconstruindo conhecimento:

- verificar evidência externa;
- reconciliar estados pendentes;
- eliminar ambiguidades.

Somente após convergência o sistema retorna ao modo NORMAL.

FALHA CRÍTICA

Continuar executando sob incerteza não é resiliência. É amplificação de erro.

INVARIANTES OPERACIONAIS

- SAFE significa perda de prova suficiente; continuar executando vira risco.
- Em SAFE, efeitos irreversíveis ficam bloqueados e a preservação de evidência é máxima.
- Intervenção manual deve ser auditável e nunca deve alterar a evidência externa.
- Saída de SAFE exige checkpoint de reconciliação concluído e prova restaurada.
- Parar sob incerteza é comportamento correto para reduzir custo de recuperação.

Part V

INTERFACE HUMANO ⟷ SISTEMA

Chapter 13

Operador Como Parte do Sistema

Automação não elimina decisões humanas. Ela apenas desloca onde elas ocorrem.

Sistemas em produção inevitavelmente atingem estados onde decisões automáticas deixam de ser seguras. Nesse ponto, o operador torna-se parte do sistema operacional.

AXIOMA

O operador não existe fora do sistema. Ele é um componente do sistema.

Ignorar essa realidade produz dois extremos igualmente perigosos:

- automação excessiva, que continua executando sob incerteza;
- intervenção manual arbitrária, que quebra invariantes do domínio.

O objetivo não é remover o humano do processo. É tornar decisões humanas verificáveis.

Automação Incompleta

Todo sistema possui limites onde automação deve parar:

- quando evidência é insuficiente;
- quando múltiplas interpretações são possíveis;
- quando o custo de erro excede o custo de pausa.

Nesse ponto, a decisão deve ser transferida ao operador.

Decisões Humanas Verificáveis

Uma decisão humana torna-se operacionalmente segura quando:

- o estado anterior é preservado;

- a ação executada é registrada;
- o motivo da decisão é auditável;
- o sistema pode continuar após a intervenção.

ENGENHARIA

Intervenção humana não deve alterar a realidade. Deve apenas alterar a interpretação do sistema sobre a realidade.

Essa distinção impede que correções manuais criem novos estados impossíveis.

INVARIANTES OPERACIONAIS

- O operador é componente do sistema: decisões humanas precisam ser modeladas.
- Automação deve parar quando há múltiplas interpretações possíveis com risco alto.
- Decisão humana segura exige rastreabilidade: ação, motivo, contexto e estado anterior preservados.
- Intervenção humana ajusta interpretação do sistema, não a realidade externa.

Chapter 14

Intervenção Manual Segura

Intervenção manual é inevitável em sistemas long-lived.

O problema não é intervir. O problema é intervir sem limites explícitos.

AXIOMA

Nem todo estado pode ser corrigido manualmente.

A arquitetura deve definir claramente três categorias:

Estados Corrigíveis

Estados internos derivados podem ser ajustados manualmente:

- marcação de reconciliação concluída;
- reprocessamento de jobs;
- atualização de estados intermediários.

Essas alterações não modificam evidência externa.

Estados Não Corrigíveis

Evidência externa nunca deve ser alterada manualmente:

- transações confirmadas;
- liquidações externas;
- eventos já observados na fonte de verdade.

Alterar esses elementos destrói a capacidade de reconciliação futura.

FALHA CRÍTICA

Quando evidência é alterada manualmente, o sistema perde a capacidade de verificar.

Comandos Operacionais Seguros

Intervenções devem ocorrer através de comandos explícitos, nunca por alteração direta de dados.

Exemplos:

- `reconcile(txid)`
- `replay(job_id)`
- `mark_as_observed(event)`

Comandos preservam intenção e rastreabilidade.

Alterações diretas removem contexto.

INVARIANTES OPERACIONAIS

- Intervenção manual deve ocorrer por comandos explícitos, não por edição direta de dados.
- Evidência externa é imutável operacionalmente: nunca deve ser corrigida manualmente.
- Toda ação manual deve produzir registro auditável e reversibilidade lógica quando aplicável.
- A intervenção deve reduzir ambiguidade e habilitar reconciliação determinística.

Chapter 15

Postmortems Verify-Oriented

Após um incidente, a tendência natural é reconstruir uma narrativa linear.

Essa narrativa raramente corresponde ao que ocorreu.

AXIOMA

Incidentes não são histórias. São estados que deixaram de convergir.

Postmortems tradicionais procuram causas. Postmortems verify-oriented procuram suposições incorretas.

A pergunta central não é:

quem errou?

mas:

qual hipótese o sistema assumiu sem evidência?

Incidentes Como Remoção de Ilusões

A maioria dos incidentes revela:

- execução tratada como confirmação;
- estados intermediários invisíveis;
- dependência de ordem implícita;
- ausência de reconciliação.

O incidente não cria o problema. Ele apenas o torna visível.

Análise Baseada em Estado

Uma análise verify-oriented reconstrói:

- quais estados existiam;
- quais transições ocorreram;
- qual evidência estava disponível em cada momento.

Essa abordagem evita erro retrospectivo, onde decisões passadas são julgadas com conhecimento futuro.

ENGENHARIA

O objetivo do postmortem não é evitar falhas futuras. É reduzir o espaço de suposições não verificadas.

Sistemas maduros acumulam memória operacional.

Cada incidente reduz a quantidade de comportamento implícito.

INVARIANTES OPERACIONAIS

- Incidente é divergência que deixou de convergir, não narrativa linear.
- A causa operacional é uma hipótese assumida sem evidência suficiente.
- A análise deve reconstruir estados e evidências disponíveis no tempo, não julgar com conhecimento futuro.
- Todo postmortem deve reduzir o espaço de suposições não verificadas do sistema.

Part VI

ARQUITETURA VERIFY EM ESCALA

Chapter 16

Sistemas Long-Lived

A maioria dos sistemas é projetada como se tivesse início e fim bem definidos.

Deploy. Execução. Encerramento.

Sistemas reais não possuem esse ciclo.

Eles continuam existindo enquanto executam.

AXIOMA

Tempo é dimensão arquitetural.

À medida que o tempo passa:

- dependências mudam;
- versões divergem;
- dados acumulam inconsistências;
- suposições originais deixam de ser verdadeiras.

Esse fenômeno é inevitável.

É conhecido como drift operacional.

Drift Inevitável

Drift não indica erro. Indica evolução sob condições reais.

Exemplos comuns:

- estados antigos que não correspondem mais ao modelo atual;
- integrações que mudaram comportamento;
- dados persistidos sob regras anteriores.

Sistemas frágeis tentam eliminar drift. Sistemas verificáveis aprendem a operar apesar dele.

Evolução Segura

Evoluir um sistema long-lived exige preservar três propriedades:

- estados antigos permanecem interpretáveis;
- evidência histórica não é destruída;
- reconciliação continua possível após mudanças.

ENGENHARIA

Arquitetura verify não assume que o sistema será reescrito. Ela assume continuidade indefinida.

Isso altera decisões fundamentais:

- migrações tornam-se transições de estado;
- versões coexistem temporariamente;
- compatibilidade histórica torna-se requisito.

INVARIANTES OPERACIONAIS

- Tempo é dimensão arquitetural: drift é inevitável e deve ser operável.
- Estados antigos precisam permanecer interpretáveis após evolução do sistema.
- Evidência histórica não deve ser destruída; ela é a base da verificação.
- Mudanças devem preservar a possibilidade de reconciliação contínua.

Chapter 17

Observabilidade como Evidência

Observabilidade é frequentemente tratada como ferramenta de debugging.

Em sistemas verificáveis, ela possui função mais fundamental.

AXIOMA

Observabilidade produz evidência operacional.

Sem observabilidade adequada, o sistema continua executando, mas perde capacidade de explicar o próprio comportamento.

Métricas vs Prova

Métricas indicam tendência. Elas não confirmam eventos.

Exemplo:

- aumento de latência indica problema possível;
- não confirma qual operação falhou.

Métricas são sinais. Não são evidência.

Logs vs Estado

Logs registram narrativa. Estado representa posição atual.

Logs podem afirmar que algo ocorreu. Somente estado verificável permite confirmar.

ALERTA

Logs descrevem o que o sistema acreditou que aconteceu. Não o que necessariamente aconteceu.

Tracing como Reconstrução Histórica

Tracing permite reconstruir causalidade.

Ele responde perguntas que logs isolados não conseguem:

- qual decisão originou o efeito;
- quais estados intermediários existiram;
- onde o conhecimento foi perdido.

Tracing transforma execução em história auditável.

Observabilidade Operacional

Observabilidade madura expõe:

- estados intermediários;
- divergência acumulada;
- tempo em estados não confirmados;
- reconciliações pendentes.

Quando esses elementos são visíveis, incidentes deixam de surgir como surpresa.

INVARIANTES OPERACIONAIS

- Observabilidade deve produzir evidência operacional, não apenas métricas de tendência.
- Logs registram crença do sistema; não constituem prova por si só.
- Estados intermediários devem ser observáveis para evitar mentiras operacionais.
- Tracing deve permitir reconstruir causalidade: decisão execução evidência correção.

Chapter 18

Arquiteturas que Sobrevivem ao Tempo

A maioria das arquiteturas é avaliada por desempenho inicial.

Poucas são avaliadas por sobrevivência ao longo dos anos.

AXIOMA

Arquiteturas sobrevivem quando permanecem explicáveis.

Complexidade inevitavelmente cresce.

Novos fluxos são adicionados. Novas integrações surgem. Estados intermediários aumentam.

Sem mecanismos de verificação, o sistema continua funcionando, mas deixa de ser compreensível.

Desacoplamento Operacional

Desacoplamento não é apenas separação técnica.

É separação entre:

- decisão;
- execução;
- verificação.

Quando essas etapas são independentes:

- falhas tornam-se locais;
- reconciliação torna-se possível;
- evolução não quebra continuidade.

Sistemas Auditáveis

Auditabilidade não é requisito regulatório. É requisito operacional.

Um sistema auditável consegue responder:

- por que este estado existe;
- qual evidência o originou;
- qual transição o produziu.

Sem essa propriedade, recuperação após falha torna-se interpretação humana.

Explicabilidade Sistêmica

Explicabilidade significa que o sistema consegue justificar seu estado atual sem depender da memória de quem o construiu.

Isso exige:

- estados explícitos;
- transições registradas;
- evidência preservada.

ENGENHARIA

Sistemas que sobrevivem ao tempo não são os mais eficientes. São os que continuam compreensíveis.

INVARIANTES OPERACIONAIS

- Arquitetura sobrevive quando permanece explicável sob falha e mudança.
- Decisão, execução e verificação devem ser desacopladas para permitir recuperação.
- Todo estado deve ser justificável por transições registradas e evidência preservada.
- Evolução segura exige que o sistema continue auditável após mudanças.

Part VII

VERIFY SYSTEMS NA PRÁTICA

Chapter 19

Casos Reais de Falha Operacional

Os exemplos anteriores descrevem o modelo Verify de forma estrutural.

Nesta seção, o objetivo é demonstrar como falhas reais emergem em produção, mesmo quando todos os componentes estão funcionando corretamente.

Os casos abaixo não representam erros excepcionais. Eles representam situações inevitáveis em sistemas distribuídos.

Cada caso é descrito em termos de:

- estado antes da falha;
- hipótese do sistema;
- evidência real observável;
- divergência resultante;
- reconciliação determinística.

O objetivo não é evitar o evento inicial, mas preservar a capacidade de convergir após ele.

19.1 Pagamento LN confirmado com timeout local

Estado Inicial

O sistema inicia um pagamento Lightning Network.

- invoice válida;
- liquidez disponível;
- pagamento enviado ao node LN;
- worker aguardando confirmação.

Hipótese do Sistema

O worker atinge timeout antes de receber confirmação.

Estado interno:

- pagamento considerado falho;
- operação elegível para retry.

Evidência Real

O pagamento foi concluído na rede LN.

- preimage revelado;
- invoice liquidada;
- efeito externo irreversível ocorreu.

Divergência

O sistema acredita que o pagamento falhou, enquanto o settlement externo indica sucesso.

Sem reconciliação:

- retry gera novo pagamento;
- duplicação de efeito financeiro.

Reconciliação Verify

Procedimento correto:

1. consultar estado do pagamento via payment_hash;
2. observar evidência externa (preimage);

3. atualizar ledger interno para refletir settlement real;

4. marcar operação como concluída.

INVARIANTES OPERACIONAIS

- execution_timeout $\not\Rightarrow$ payment_failure
- external_settlement > local_timeout
- retry_allowed $\Rightarrow$ proof_absent

19.2 Broadcast de Transação + Queda do Worker

Estado Inicial

Um swap executa broadcast de transação on-chain.

- transação construída;
- broadcast realizado;
- txid retornado pelo node.

Hipótese do Sistema

O processo falha antes de persistir o estado final.

Após reinicialização:

- swap aparece como não executado;
- sistema considera repetir operação.

Evidência Real

A transação já existe na mempool ou blockchain.

O efeito externo ocorreu independentemente do estado interno.

Divergência

Ausência de registro interno é interpretada como ausência de execução.

Retry produz:

- nova transação;
- possível duplicação de saída.

Reconciliação Verify

Procedimento correto:

1. localizar transação por padrões determinísticos (outputs/endereço);
2. confirmar existência externa;
3. registrar effect_ref no ledger;
4. avançar estado para SETTLED.

INVARIANTES OPERACIONAIS

- missing_local_state $\not\Rightarrow$ missing_external_effect
- restart == reconciliation_required
- state_advance $\Rightarrow$ external_proof

19.3 Node Desincronizado

Estado Inicial

Sistema consulta estado da blockchain através de node local.

- node aparentemente saudável;
- RPC responde normalmente;
- altura do bloco atrasada.

Hipótese do Sistema

Consulta retorna que a transação não existe.

Estado interno:

- operação considerada não confirmada;
- reconciliação continua aguardando.

Evidência Real

A transação já foi confirmada na rede.

Outros nodes e exploradores indicam confirmação.

Divergência

O sistema utiliza fonte de verdade desatualizada.

Node state diverge do settlement real.

Reconciliação Verify

Procedimento correto:

1. detectar inconsistência entre fontes;
2. ascender na hierarquia de verdade;
3. priorizar settlement externo;
4. corrigir estado interno.

INVARIANTES OPERACIONAIS

- node_state $\not=$ settlement_truth
- truth_conflict $\Longrightarrow$ hierarchy_ascend
- single_source_trust == false

19.4 RPC Retornando Estado Obsoleto

Estado Inicial

Sistema consulta RPC para verificar estado recente.

- cache intermediário ativo;
- RPC retorna estado antigo;
- execução continua normalmente.

Hipótese do Sistema

O sistema assume que ausência de atualização indica ausência de evento.

Evidência Real

O evento ocorreu, mas ainda não é visível na fonte consultada.

Divergência

Execução avança baseada em conhecimento incompleto.

Estados intermediários tornam-se inconsistentes.

Reconciliação Verify

Procedimento correto:

1. reconhecer ausência de evidência como estado neutro;

2. aguardar nova observação ou consultar fonte alternativa;

3. evitar avanço irreversível sem prova.

INVARIANTES OPERACIONAIS

- absence_of_evidence $\neq$ evidence_of_absence
- stale_data $\Rightarrow$ wait_or_revalidate
- irreversible_ops $\Rightarrow$ proof_required

Chapter 20

Anti-Patterns Verify

Sistemas raramente falham por ausência de tecnologia.

Eles falham por assumir propriedades que não existem em produção.

Os anti-patterns abaixo representam violações recorrentes do modelo Verify. Eles aparecem mesmo em sistemas tecnicamente corretos.

O objetivo desta seção não é criticar implementações, mas tornar falhas reconhecíveis antes que se tornem incidentes.

20.1 Logs como Fonte de Verdade

Sintoma

O sistema considera uma operação concluída porque um log afirma que foi executada.

Por que acontece

Logs registram intenção e fluxo de execução. Eles não confirmam efeitos externos.

Consequência Operacional

- estados internos divergem silenciosamente;
- reconciliação torna-se ambígua;
- incidentes só aparecem tardiamente.

Correção Verify

Estado só avança mediante evidência externa verificável.

INVARIANTES OPERACIONAIS

- $\log \not\Rightarrow \text{proof}$
- $\text{execution\_recorded} \not\Rightarrow \text{effect\_confirmed}$

20.2 Retry sem Idempotência

Sintoma

Retries geram múltiplos efeitos externos para a mesma intenção.

Por que acontece

Execução é tratada como evento único. Falhas parciais não são modeladas.

Consequência Operacional

- duplicação de pagamentos;
- inconsistência financeira;
- reconciliação manual necessária.

Correção Verify

Toda operação com efeito externo possui identidade única e replay seguro.

INVARIANTES OPERACIONAIS

- retry == reexecution
- $\text{same\_intent} \Rightarrow \text{same\_effect}$

20.3 Execução Linear em Sistema Assíncrono

Sintoma

O sistema assume que eventos ocorrem na ordem em que foram emitidos.

Por que acontece

Modelos mentais síncronos são aplicados a ambientes distribuídos.

Consequência Operacional

- estados impossíveis;
- confirmações fora de ordem;
- correções destrutivas.

Correção Verify

Estados devem ser derivados de evidência, não da ordem de execução observada.

INVARIANTES OPERACIONAIS

- observed_order $\not=$ real_order
- state_advance $\Longrightarrow$ proof

20.4 Dependência da Memória do Worker

Sintoma

Após reinicialização, o sistema perde capacidade de continuar corretamente.

Por que acontece

Estado crítico existe apenas em memória durante execução.

Consequência Operacional

- duplicação de operações;
- estados órfãos;
- necessidade de intervenção manual.

Correção Verify

Toda informação necessária para continuidade deve estar persistida.

INVARIANTES OPERACIONAIS

- restart == valid_continuation
- memory $\not=$ source_of_truth

20.5 Confirmação Antes da Prova

Sintoma

O sistema marca sucesso antes de observar evidência externa.

Por que acontece

Otimização prematura ou tentativa de reduzir latência percebida.

Consequência Operacional

- estados finais incorretos;

- correções tardias complexas;
- perda de confiança operacional.

Correção Verify

Estados finais exigem prova externa suficiente.

INVARIANTES OPERACIONAIS

- execution $\not\Rightarrow$ success
- final_state $\Rightarrow$ external_proof

20.6 Reconciliação como Tarefa Secundária

Sintoma

Reconciliação ocorre apenas após incidentes.

Por que acontece

Execução é tratada como fluxo principal.

Consequência Operacional

- divergência acumulada;
- incidentes maiores;
- recuperação custosa.

Correção Verify

Reconciliação é processo contínuo e primário.

INVARIANTES OPERACIONAIS

- execution == hypothesis
- reconciliation == convergence

Chapter 21

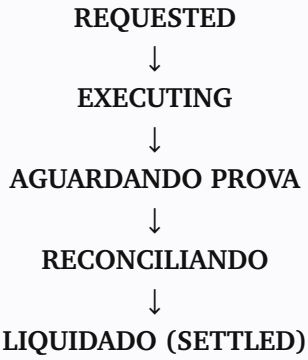
Exemplo Completo: Sistema de Swap

Swaps entre redes independentes expõem todas as propriedades Verify:

- múltiplas fontes de verdade (ledger interno, node(s), settlement externo);
- efeitos externos irreversíveis;
- confirmação assíncrona;
- falha parcial inevitável;
- necessidade de reconciliação determinística.

O objetivo aqui é mostrar o sistema como **máquina de estado operacional**.

DIAGRAMA OPERACIONAL



Máquina de Estados do Swap

Estados mínimos (operacionais):

- `REQUESTED`: intenção registrada (decisão persistida).
- `EXECUTING`: tentativa de produzir efeito externo (broadcast / HTLC / payment).
- `AWAITING_PROOF`: efeito pode ter ocorrido; o sistema aguarda evidência.
- `RECONCILING`: divergência detectada; prioridade é provar e corrigir.
- `SETTLED`: evidência externa confirmada e estado interno alinhado.
- `FAILED_SAFE`: estado não provável; SAFE MODE / intervenção humana.

O que Persistir (para sobreviver ao restart)

Para que reinicialização seja continuidade, persista o mínimo necessário:

- **decision_record**: intenção + parâmetros + `idempotency_key`
- **effect_ref**: referência do efeito externo (txid, payment_hash, htlc id, etc.)
- **expected_proof**: qual evidência valida sucesso (confirmação on-chain, preimage, etc.)
- **TTL / deadlines**: limites de espera operacional
- **audit_trail**: transições + motivo (suficiente para reconstrução)

ENGENHARIA

Transições determinísticas (forma reduzida):

- `REQUESTED -> EXECUTING` quando `preconditions == true`
- `EXECUTING -> AWAITING_PROOF` ao emitir efeito externo + persistir `effect_ref`
- `AWAITING_PROOF -> SETTLED` quando evidência externa confirma `effect_ref`
- `AWAITING_PROOF -> RECONCILING` se `TTL` excede ou conflito detectado
- `RECONCILING -> SETTLED` quando correção determinística converge
- `RECONCILING -> FAILED_SAFE` se `unknown_truth == true` ou `K` falhas consecutivas

Timeline de Falha (Caso Realista)

Cenário: o sistema transmite a transação (efeito externo), mas cai antes de consolidar o estado interno.

1. **T0** `REQUESTED`: swap criado. `idempotency_key` persistida.
2. **T1** `EXECUTING`: worker constrói e transmite `txid=abc...`
3. **T2** Falha: processo reinicia antes de registrar `effect_ref` corretamente em ledger interno.
4. **T3** Após restart, sistema vê swap incompleto. Sem Verify, executaria novamente.
5. **T4** Verify: sistema entra em `RECONCILING`, porque `unknown_truth==true`.

O ponto crítico é **T3**: ausência de confirmação não é evidência de falha.

ALERTA

Sem reconciliação, o retry vira uma nova operação. Com Verify, retry vira reexecução segura guiada por prova.

Reconciliação (Provar e Corrigir)

Reconciliação é um procedimento determinístico:

1. Recarregar `decision_record` pelo `idempotency_key`.
2. Descobrir `effect_ref` por:
   - busca no node por padrões (outputs/addresses), ou
   - consulta por `payment_hash / htlc id`, ou
   - re-derivação determinística do efeito esperado.
3. Consultar evidência externa: mempool/chain/LN proof.

4. Aplicar regra de correção: `ledger := external_truth`.
5. Registrar checkpoint: `reconciliation_checkpoint := complete`.

ENGENHARIA

Regra de correção (exemplo simples):

- Se `external_proof(effect_ref) == confirmed` então `state := SETTLED`
- Se `external_proof == not_found` e `TTL not exceeded` então `state := AWAITING_PROOF`
- Se `TTL exceeded` e `proof ambiguous` então `state := FAILED_SAFE`

Intervenção Humana (somente para ambiguidade)

Intervenção ocorre quando:

- múltiplos `effect_ref` possíveis,
- evidência externa incompleta,
- dependência externa instável impede prova.

O operador não conserta o settlement. Ele resolve ambiguidade e libera reconciliação.

INVARIANTES OPERACIONAIS

- manual_decision ⇒audit_trail.appended == true
- manual_action ⇒evidence_unchanged == true
- operator_ack ⇒system_reconciles == true

FALHA CRÍTICA

O objetivo do operador é restaurar prova, não forçar sucesso.

INVARIANTES OPERACIONAIS

- Nenhum avanço para final ocorre sem prova externa suficiente do resultado.
- Toda tentativa deve ter identidade e idempotência para impedir duplicação de efeito.
- Estados intermediários precisam de TTL e gatilhos de reconciliação determinística.
- Na ambiguidade, o sistema reduz permissões e pode exigir decisão humana auditável.
- Reinício deve retomar pela evidência e pelo estado persistido, nunca pela memória do worker.

Chapter 22

VERIFY Metrics

Sistemas verificáveis não operam apenas por lógica. Eles operam por sinais observáveis.

VERIFY Metrics transformam propriedades epistemológicas em variáveis mensuráveis.

O objetivo não é medir performance. É medir proximidade entre estado interno e realidade externa.

Métricas Verify respondem à pergunta:

o sistema sabe o suficiente para continuar executando?

22.1 Princípios das Métricas Verify

Métricas tradicionais medem:

- latência;
- throughput;
- uso de recursos;
- taxa de erro.

Métricas Verify medem:

- atraso entre execução e prova;
- duração da incerteza;
- capacidade de convergência.

AXIOMA

O risco operacional cresce quando o tempo entre execução e prova aumenta.

Uma métrica Verify é válida quando altera comportamento do sistema, não apenas dashboards.

INVARIANTES OPERACIONAIS

- Métricas Verify medem conhecimento operacional, não performance.
- Toda métrica deve poder alterar modo operacional automaticamente.
- Aumento de incerteza deve reduzir permissões do sistema.

22.2 Reconciliation Lag

Definição

Tempo desde o último ciclo de reconciliação bem-sucedido.

$$reconciliation\_lag = now - last\_successful\_reconciliation$$

O que mede

A distância temporal entre estado interno e verificação externa.

Risco

Quanto maior o lag:

- maior o número de hipóteses não verificadas;
- maior o custo de correção futura;
- maior a probabilidade de divergência acumulada.

Uso Operacional

- aumento progressivo move sistema para DEGRADED;
- limite superior ativa RECONCILIATION MODE.

INVARIANTES OPERACIONAIS

- reconciliation_lag $\uparrow \Rightarrow operational\_risk \uparrow$
- no_reconciliation $\Rightarrow$no_trustable_state

22.3 Unknown Truth Duration

Definição

Tempo em que o sistema permanece incapaz de afirmar qual estado é verdadeiro.

$$unknown\_truth\_duration = now - first\_moment(unknown\_truth == true)$$

O que mede

Duração da ambiguidade operacional.

Exemplos:

- efeito externo possivelmente executado;
- confirmação inconclusiva;
- fontes de verdade conflitantes.

Risco

Ambiguidade prolongada expande o espaço de estados possíveis.

Cada nova execução aumenta custo de reconciliação.

Uso Operacional

- acima de limiar bloquear operações irreversíveis;
- acima de limite crítico SAFE MODE.

INVARIANTES OPERACIONAIS

- unknown_truth ⇒irreversible_ops == disabled
- ambiguity_time ↑⇒ $execution\_permission \downarrow$

22.4 Proof Latency

Definição

Tempo entre execução de um efeito e observação da evidência correspondente.

$$proof\_latency = time(execution) - time(external\_proof)$$

O que mede

A latência epistemológica do sistema.

Não mede rede ou performance, mas o tempo necessário para transformar hipótese em conhecimento.

Risco

Alta proof latency implica:

- estados intermediários prolongados;
- aumento de retries;
- maior probabilidade de decisões prematuras.

Uso Operacional

- define TTL de estados intermediários;
- ajusta limites de retry;
- alimenta failure budget.

INVARIANTES OPERACIONAIS

- execution ⇒proof_expected
- proof_latency > threshold ⇒reconciliation

22.5 Truth Conflict Rate

Definição

Taxa de divergência entre estado interno e evidência externa.

$$truth\_conflict\_rate = conflicts/reconciliation\_window$$

O que mede

Frequência com que hipóteses internas são corrigidas.

Interpretação

- baixo e estável sistema saudável;
- crescente perda de qualidade da evidência ou dependência;
- alto execução deve desacelerar.

INVARIANTES OPERACIONAIS

- conflict_rate ↑⇒ $throughput \downarrow$
- persistent_conflicts ⇒degraded_mode

22.6 VERIFY Metrics como Sistema de Controle

Quando combinadas, essas métricas formam um mecanismo de controle:

- reconciliation lag mede atraso de verificação;
- unknown truth duration mede ambiguidade;
- proof latency mede tempo até conhecimento;
- conflict rate mede divergência estrutural.

ENGENHARIA

Modo operacional pode ser derivado diretamente:

- NORMAL métricas estáveis
- DEGRADED lag ou latency crescentes
- RECONCILING conflitos detectados
- SAFE unknown truth prolongado

Nesse ponto, o sistema deixa de depender de interpretação humana contínua.

Ele passa a reagir à própria incerteza.

INVARIANTES OPERACIONAIS

- Métricas Verify existem para limitar execução sob incerteza crescente.
- Aumento de ambiguidade deve reduzir automaticamente permissões do sistema.
- Modo operacional deve ser derivável das métricas observadas.
- O sistema deve medir o tempo necessário para saber, não apenas o tempo para executar.

Chapter 23

VERIFY Control Plane

Sistemas tradicionais misturam execução, observação e decisão no mesmo fluxo.

Enquanto o sistema opera sob condições ideais, essa fusão parece eficiente. Sob falha parcial, ela torna o sistema incapaz de distinguir entre o que aconteceu e o que ele acredita que aconteceu.

O modelo Verify separa explicitamente três responsabilidades:

- Execution Plane
- Verify Plane
- Control Plane

Essa separação não é organizacional. É epistemológica.

Cada plano possui acesso a informações diferentes e responsabilidades distintas.

AXIOMA

Execução produz efeitos. Verificação produz conhecimento. Controle produz comportamento.

23.1 Execution Plane

O Execution Plane é responsável por produzir efeitos.

Ele executa:

- broadcasts;
- pagamentos;
- swaps;
- escritas externas;

- transições de estado iniciadas por decisão.

Sua função é tentativa.

Ele não determina sucesso.

ENGENHARIA

Execution Plane assume que pode falhar, ser interrompido ou reiniciado a qualquer momento.

Características:

- opera sob hipótese;
- não possui autoridade sobre estado final;
- pode repetir execução de forma idempotente.

INVARIANTES OPERACIONAIS

- execution $\neq$confirmation
- execution_plane $\not\Rightarrow$final_state
- execution_failure $\neq$operation_failure

23.2 Verify Plane

O Verify Plane observa o mundo externo.

Ele transforma execução em conhecimento.

Responsabilidades:

- observar evidência externa;
- comparar estado interno com realidade;
- detectar divergência;
- iniciar reconciliação.

O Verify Plane não executa efeitos irreversíveis.

Ele observa e corrige interpretação.

ENGENHARIA

Verify Plane responde à pergunta:

o que pode ser provado que aconteceu?

Características:

- orientado a evidência;
- determinístico;
- independente da execução original.

INVARIANTES OPERACIONAIS

- proof > execution_result
- verify_plane ⇒state_correction
- same_evidence ⇒same_result

23.3 Control Plane

O Control Plane decide como o sistema deve se comportar.

Ele não executa operações nem observa diretamente o mundo externo.

Ele consome:

- métricas Verify;
- sinais de divergência;
- estados operacionais;
- resultados de reconciliação.

Sua função é alterar permissões do sistema.

ENGENHARIA

Control Plane responde à pergunta:
o sistema sabe o suficiente para continuar executando?

Responsabilidades:

- definir modo operacional;
- reduzir throughput;
- bloquear operações irreversíveis;
- ativar SAFE MODE quando necessário.

INVARIANTES OPERACIONAIS

- uncertainty ↑⇒ $permissions \downarrow$
- control_plane ⇒behavioral_change
- mode ⇒execution_constraints

23.4 Interação Entre os Planos

O fluxo Verify completo pode ser descrito como:

$$\text{Decision} \rightarrow \text{Execution Plane} \rightarrow \text{Verify Plane} \rightarrow \text{Control Plane}$$

- Execution produz hipótese;
- Verify produz evidência;
- Control ajusta comportamento.

Nenhum plano possui autoridade completa sozinho.

ALERTA

Quando Execution Plane decide estado final, o sistema perde verificabilidade.
Quando Control Plane não reage à incerteza, o sistema amplifica erro.

23.5 Separação Lógica vs Separação Física

Os três planos não precisam existir como serviços independentes.

Eles podem coexistir:

- no mesmo processo;
- em workers diferentes;
- ou distribuídos em múltiplos sistemas.

O requisito é apenas que suas responsabilidades não sejam confundidas.

Misturar planos remove a capacidade do sistema de distinguir tentativa de verdade.

23.6 Resultado Arquitetural

Quando os três planos existem explicitamente:

- execução pode falhar sem corromper estado;
- reinicialização torna-se continuidade;
- reconciliação torna-se determinística;
- comportamento adapta-se à incerteza.

O sistema deixa de depender de execução perfeita.

Ele passa a depender de verificação contínua.

INVARIANTES OPERACIONAIS

- Execution produz efeitos, mas não determina verdade.
- Verify transforma evidência em conhecimento operacional.
- Control altera comportamento conforme nível de certeza do sistema.
- Separação entre planos preserva capacidade de convergir após falha parcial.

Chapter 24

Checklist Verify para Produção

O modelo Verify não é apenas descritivo. Ele pode ser aplicado como critério operacional explícito antes do deploy e após incidentes.

Este checklist não valida implementação. Ele valida se o sistema permanece governável sob falha parcial.

O objetivo não é impedir erro. É garantir capacidade de convergência após erro.

Antes do Deploy Validação Operacional

Um fluxo está pronto para produção apenas quando cada uma das condições abaixo pode ser respondida positivamente.

Execution Plane

- Existe identidade única para cada operação com efeito externo?
- A operação pode ser reexecutada sem duplicar efeito?
- A execução pode ser interrompida sem perda de continuidade?

Verify Plane

- Existe evidência externa clara de sucesso?
- Existe evidência externa clara de falha?
- O sistema distingue ausência de prova de prova negativa?

Control Plane

- Divergência altera comportamento automaticamente?
- O sistema reduz permissões sob incerteza crescente?

- SAFE MODE pode ser acionado sem intervenção manual?

Observabilidade

- Estados intermediários são visíveis?
- Reconciliações pendentes são mensuráveis?
- Tempo até prova (proof latency) é observável?

Recuperação

- O sistema pode reiniciar sem depender da memória do worker?
- Reconciliação funciona após restart?
- O estado atual pode ser reconstruído apenas por evidência persistida?

ALERTA

Se qualquer resposta depender de execução contínua, o sistema ainda não está pronto para produção.

Após Incidente Análise Verify

Postmortems verify-oriented não procuram culpados. Eles procuram hipóteses não verificadas.

Perguntas fundamentais:

- Qual decisão avançou sem evidência suficiente?
- Qual estado intermediário não era observável?
- Onde execução foi tratada como confirmação?
- Qual métrica Verify deveria ter alterado o modo operacional?

O objetivo da análise é reduzir o espaço de suposições implícitas.

Cada incidente deve remover uma crença não verificável do sistema.

Critérios de Maturidade Verify

Um sistema atinge maturidade operacional quando:

- pode ser reiniciado sem perda de coerência;
- pode reexecutar operações com segurança;
- pode explicar qualquer estado existente;

- pode convergir após falha parcial sem intervenção humana imediata.

Nesse estágio, confiabilidade deixa de depender de execução perfeita.

Ela passa a depender de verificação contínua.

INVARIANTES OPERACIONAIS

- Cada fluxo deve declarar explicitamente sua evidência externa de sucesso e falha.
- Reexecução deve ser segura por construção (idempotência + persistência mínima).
- Divergência deve ser detectável e capaz de alterar comportamento automaticamente.
- Reconciliação deve sobreviver a reinícios e dependências instáveis.
- Produção só é válida quando continuidade não depende da execução anterior.

EPÍLOGO SISTEMAS QUE NÃO MENTEM

Produção não é ambiente de confirmação. É ambiente de descoberta.

Sistemas não se tornam confiáveis porque evitam erro. Eles se tornam confiáveis porque continuam corretos após erro.

Humildade operacional substitui confiança.

O sistema não assume. O sistema verifica.

Perfeição é frágil. Sobrevivência é estável.

Sistemas que não mentem não prometem certeza. Eles preservam evidência suficiente para que a verdade possa emergir.

DOUTRINA VERIFY

Ao longo deste documento, sistemas foram descritos sob uma premissa simples:

o sistema não sabe.

Isso não é limitação técnica. É condição estrutural de qualquer sistema que opera no mundo real.

Eventos ocorrem fora de ordem. Confirmações chegam tarde. Estados divergem. Execução continua enquanto o conhecimento ainda é incompleto.

A engenharia tradicional tenta eliminar essa incerteza. Sistemas verificáveis partem do princípio oposto.

Eles assumem que a incerteza permanece.

Da Execução à Evidência

Execução produz efeitos. Efeitos não produzem verdade.

Verdade emerge apenas quando efeitos podem ser demonstrados.

Essa mudança altera a arquitetura fundamental do sistema:

- estados deixam de representar progresso e passam a representar evidência;
- reconciliação deixa de ser exceção e torna-se processo primário;
- falha deixa de ser interrupção e torna-se estado transitório.

O sistema deixa de perguntar:

o que deveria ter acontecido?

e passa a perguntar:

o que pode ser demonstrado que aconteceu?

Humildade Operacional

Grande parte da fragilidade em software nasce de excesso de confiança.

Confiança em execução contínua. Confiança em ordem perfeita. Confiança em confirmação imediata.

Sistemas verificáveis substituem confiança por humildade operacional.

Eles aceitam que:

- conhecimento chega depois da ação;
- consistência é temporária;
- correção precisa ser restaurada continuamente.

Essa humildade não reduz capacidade. Ela aumenta sobrevivência.

Sistemas Que Sobrevivem

Sistemas não falham porque são imperfeitos.

Eles falham porque assumem perfeição.

Ao longo do tempo, todos os sistemas encontram:

- falhas parciais,
- estados inesperados,
- operadores sob pressão,
- decisões tomadas com informação incompleta.

Arquiteturas verify não tentam impedir esses momentos.

Elas garantem que o sistema permaneça explicável depois deles.

AXIOMA

Confiabilidade não é ausência de erro. É capacidade de continuar correto após erro.

O Limite da Verificação

Nem toda incerteza pode ser eliminada.

Sempre existirá um ponto onde o sistema precisa agir antes de possuir certeza completa.

O objetivo não é remover esse limite, mas torná-lo explícito.

Quando limites são explícitos:

- decisões tornam-se conscientes;
- risco torna-se mensurável;
- falhas tornam-se recuperáveis.

Depois da Verificação

O modelo Verify não termina em arquitetura.

Ele termina em comportamento.

Sistemas maduros:

- não avançam sem evidência;
- não escondem estados intermediários;
- não confundem tentativa com resultado;
- não dependem da memória da execução.

Eles preservam informação suficiente para que a verdade possa ser reconstruída.

Esse é o ponto final.

O sistema não precisa estar sempre certo.

Ele precisa apenas nunca perder a capacidade de descobrir quando esteve errado.

VERIFY PRINCIPLES

Execução não é confirmação.

Estado local é hipótese.

Evidência supera intenção.

Reconciliação é contínua.

Incerteza reduz permissão.

Reinício é continuidade.

Sistemas devem permanecer explicáveis.

Sobrevivência é superior à perfeição.

Dont Trust. Verify.