AWS re:Invent

NOV. 28 – DEC. 2, 2022 | LAS VEGAS, NV

API308

Are you integrating or building distributed applications?

Gregor Hohpe (any)

Director, Enterprise Strategy
AWS

Subtitle(s)

Two decades of integration: why everything changed and still much remains the same

Reflections on integration, distributed systems, coupling, events, abstractions, cloud, serverless, and automation

Aka “The Blue Box Talk”

Gregor Hohpe – Enterprise Strategist

As an AWS Enterprise Strategist, Gregor helps enterprise leaders rethink their IT strategy to get the most out of their cloud journey.

Prior to joining AWS, Gregor served as Smart Nation Fellow to the Singapore government, as technical director at Google Cloud, and as Chief Architect at Allianz SE.

O'REILLY
The Software Architect Elevator
Redefining the Architect's Role in the Digital Enterprise
Gregor Hohpe
CLOUD STRATEGY
A Decision-Based Approach to Successful Cloud Migration
Gregor Hohpe
An Architect Elevator Guide
With contributions by Michele Danieli, Tahir Hashmi, and Jean-Francois Landreau
The Addison-Wesley Signature Series
ENTERPRISE INTEGRATION PATTERNS
A Martin Fowler Signature Book
Designing, Building, and Deploying Messaging Solutions
Gregor Hohpe
Bobby Woolf
With Contributions by
Kyle Brown
Conrad F. D'Cruz
Martin Fowler
Sean Neville
Michael J. Rettig
Jonathan Simon
Forewords by John Crupi and Martin Fowler

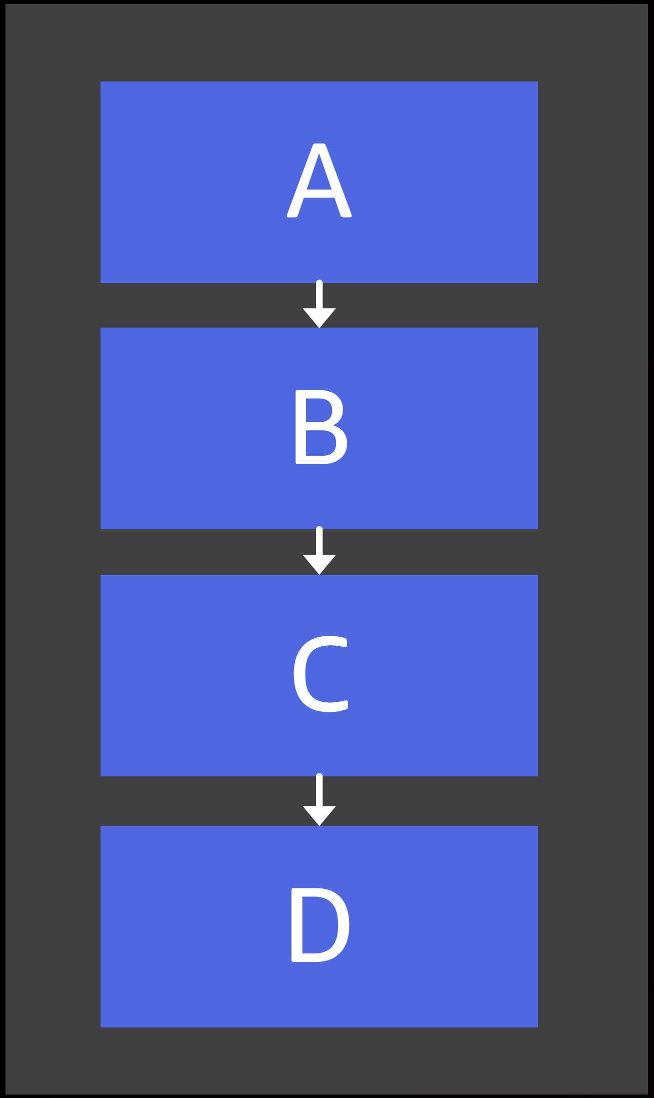
@ghohpe

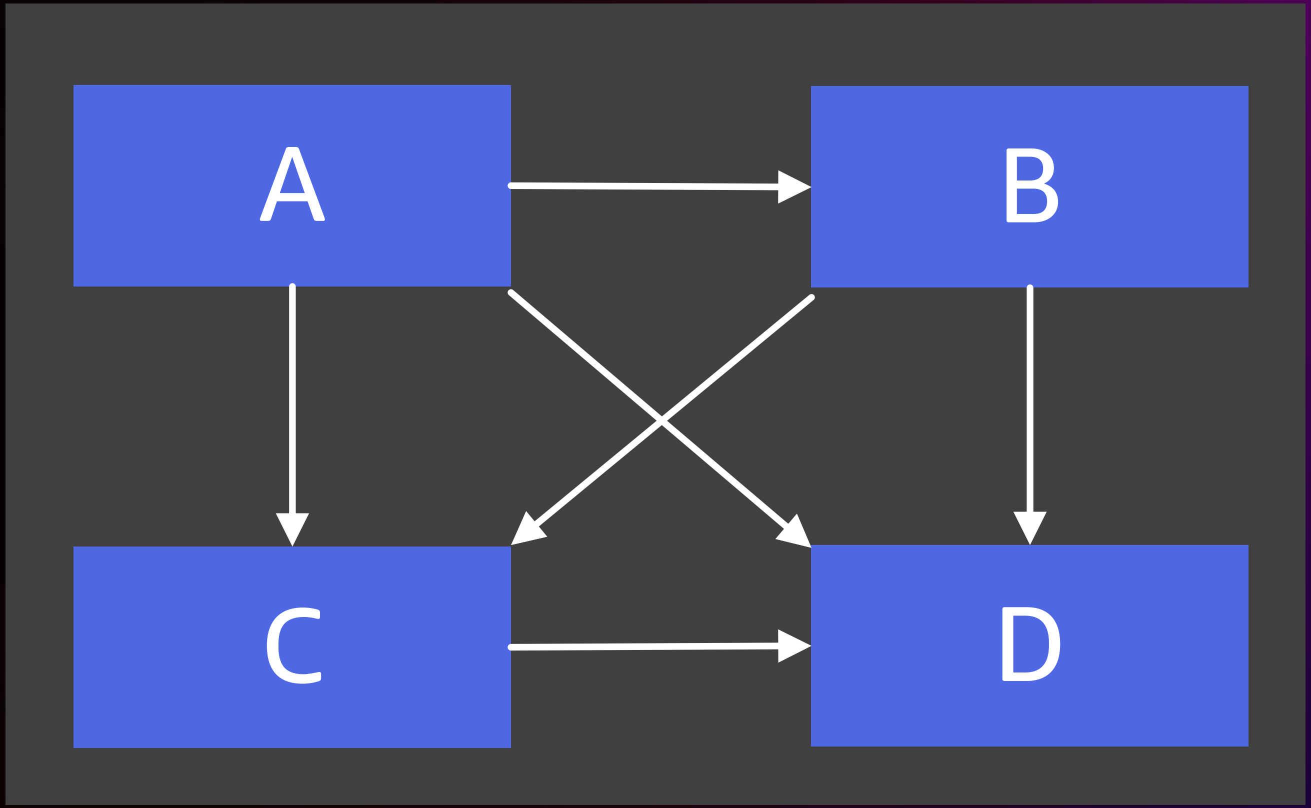
ArchitectElevator.com

www.linkedin.com/in/ghohpe/

Of boxes and lines

Two system designs





“Great architects are like great chefs: it’s not just about selecting ingredients; it’s how you put them together.”

Gregor

The Software Architect Elevator

Drawing a line

Connecting two systems. How hard can it be?

Pub-sub or point-to-point?

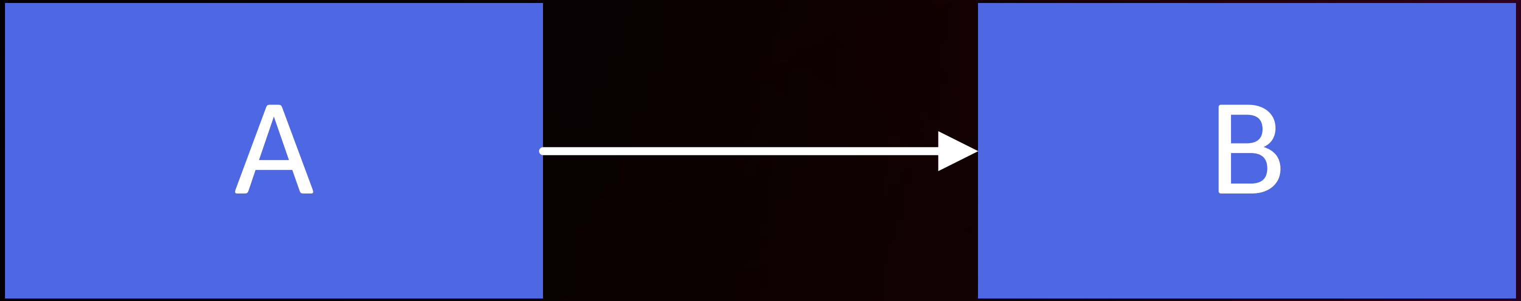
Data or control flow?

Polling?

Interaction model?

Sync/async?

Data format?



Distributed?

Systems or instances?

Conversation state?

Error handling?

Idempotency?

Partial failures?

Retries?

Back-off?

Architects see more dimensions

"Our event-driven architecture decouples teams so that they can add new components without side effects. And it needs to scale, so we made it asynchronous."

Your typical dev team

- Messaging	an interaction style
- Asynchrony	temporal interaction contract
- Publish-subscribe	message distribution, composition
- Events	specific message semantics
- Event-driven	the role of events in the application
- Distributed	a deployment choice

Your architect

Separate your architecture from your product choice

Most products combine several aspects

- Message-oriented	X	X	X	X
- Asynchronous	X	X	X	X
- Publish-subscribe	X	X		
- Events	X	X		
- Event-driven	?	?		
- Distributed	X	X	X	X

Note: for discussion purposes only. Not a product feature matrix.

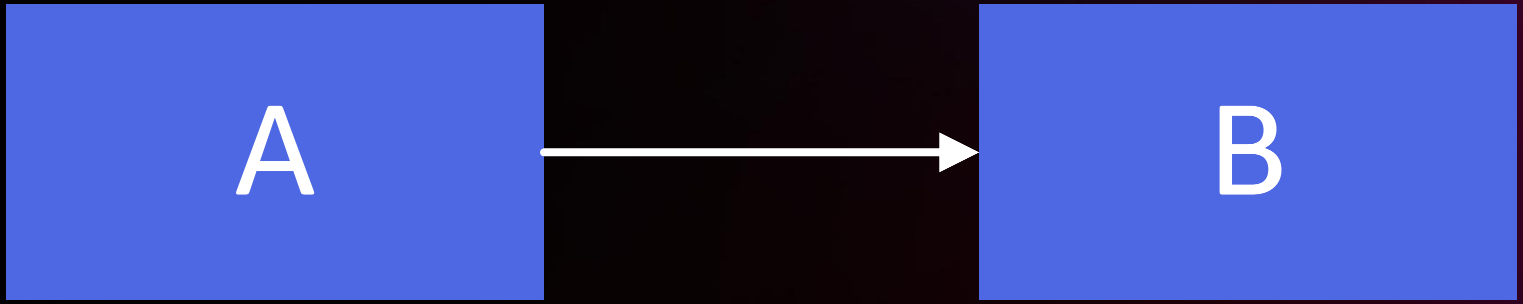
Connections and coupling

> “How do you make two systems loosely coupled? Don't connect them!”

David Orchard

BEA

Coupling – Integration’s magic word



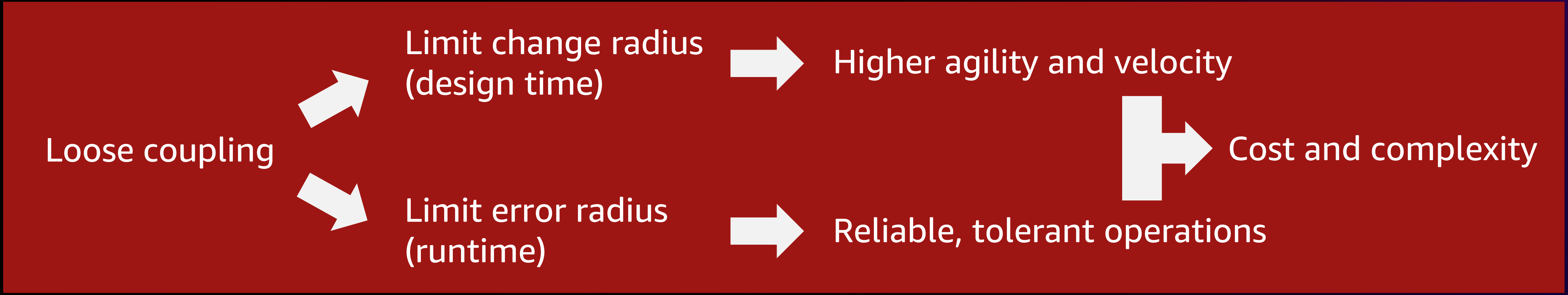
Coupling is a measure of independent variability between connected systems

Decoupling has a cost, both at design and runtime

Coupling is multi-dimensional and not binary

Buzzword slaying

- What is it (in plain terms)?	- A measure of dependency
- What benefit does it bring?	- Limits change and error radius
- When is it most valuable?	- Frequent change but limited control
- How is it achieved?	- Asynchrony, common data formats, …
- What has to be in place?	- Tooling, messaging infrastructure
- What downsides does it have?	- Overhead, complexity, tool dependence



Loose coupling

Limit change radius (design time)

Higher agility and velocity

Cost and complexity

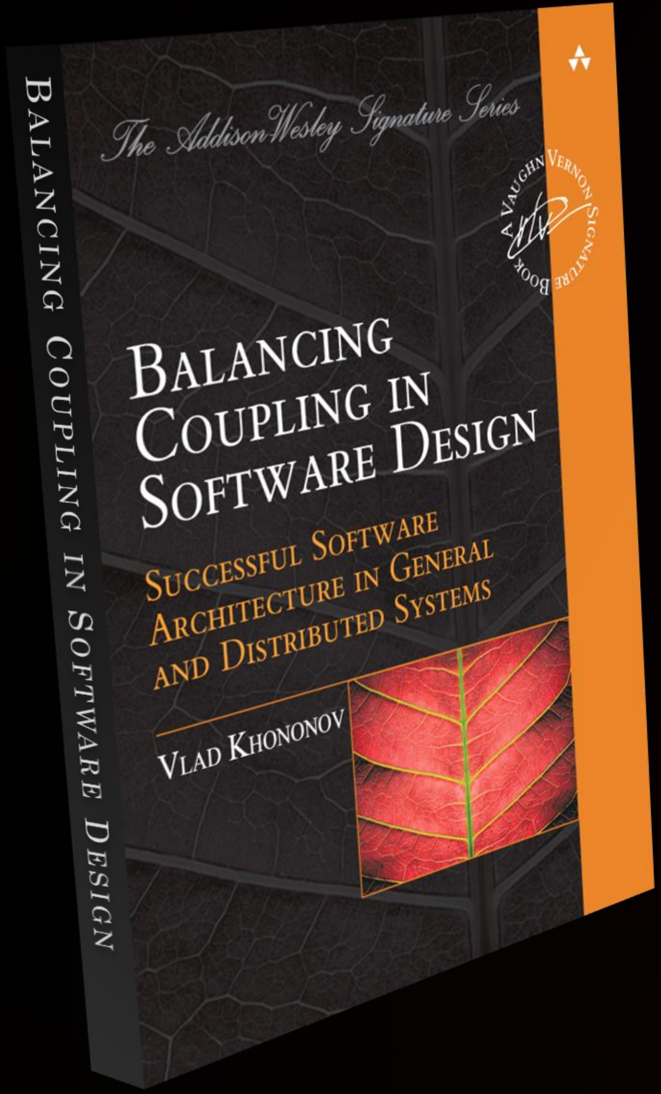
Limit error radius (runtime)

Reliable, tolerant operations

The many facets of coupling

Technology dependency:	Java vs. C++
Location dependency:	IP addresses, DNS
Data format dependency:	Binary, XML, JSON, ProtoBuf, Avro
Data type dependency:	int16, int32, string, UTF-8, null, empty
Semantic dependency:	Name, Middlename, ZIP
Temporal dependency:	sync, async
Interaction style dependency:	messaging, RPC, query style (GraphQL)
Conversation dependency:	pagination, caching, retries

Balancing coupling


The Addison-Wesley Signature Series
A Vaughn Vernon Signature Book
Balancing Coupling in Software Design
Successful Software Architecture in General and Distributed Systems
Vlad Khononov
Balancing Coupling in Software Design

- Strength
  - Content
  - Common
  - External
  - …

- Distance
  - Methods
  - Classes
  - Components
  - Services
  - Systems

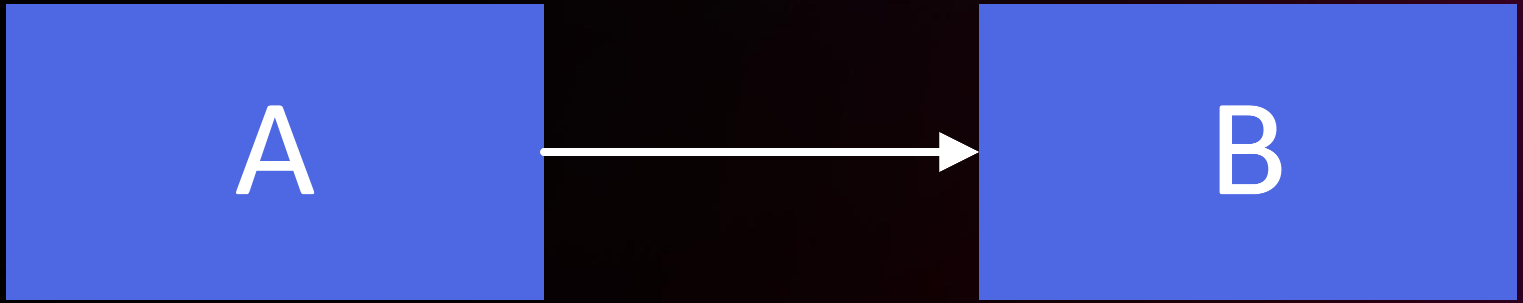
- Volatility
  - Semantic
  - Functional
  - Development
  - Operational
  - Accidental

> “The appropriate level of (design-time) coupling depends on the level of control you have over the endpoints.”

Me, after two decades of struggling with it

Integration vs. distributed systems

Integration? Distributed system?



Architecture trade-offs very much depend on the context: organization, timeline, and level of control

Spanning teams, time, and control

Approach	Level of control	Delivery lifecycle	Team
Migration	Low	One time	One off
Data synchronization/ traditional integration	Low	Long	Dedicated
Enterprise service bus	Some	Slower than component development	Likely dedicated
Distributed cloud applications	High	Same as component development	Embedded

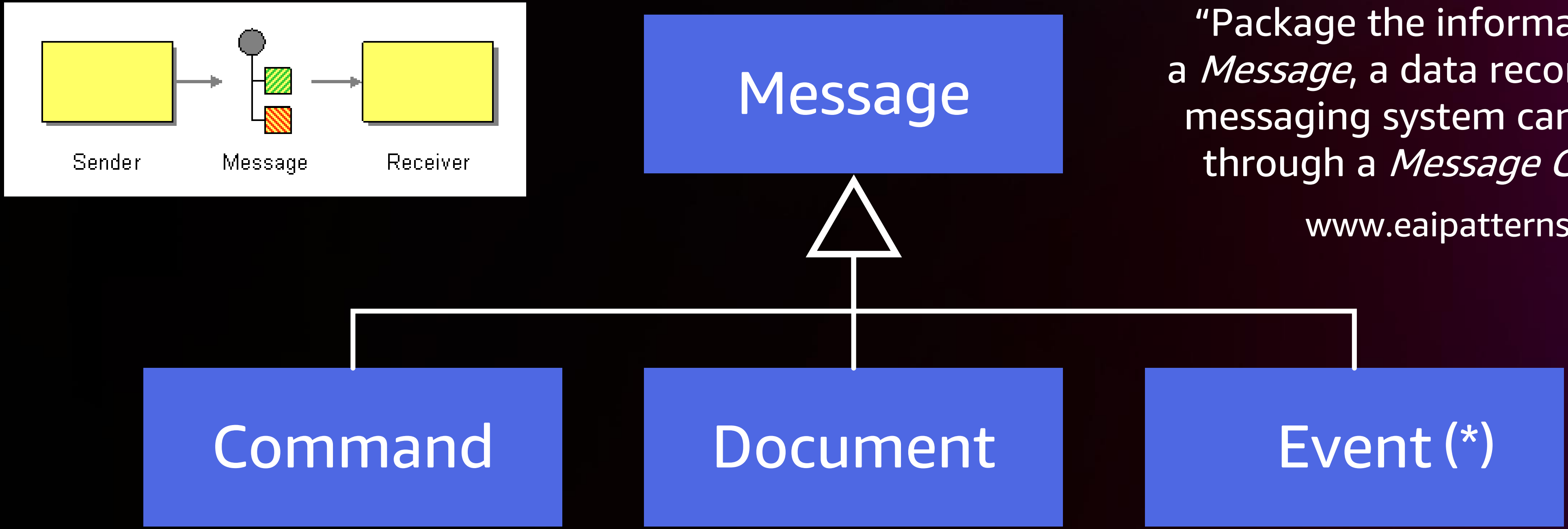
Spanning teams, time, and control

Approach	Level of control	Delivery lifecycle	Team	Tool (indicative)
Migration	Low	One time	One off	Amazon AppFlow
Data synchronization/ traditional integration	Low	Long	Dedicated	Amazon AppFlow
Enterprise service bus	Some	Slower than component development	Likely dedicated	Amazon MQ
Distributed cloud applications	High	Same as component development	Embedded	Amazon EventBridge, AWS Lambda Destinations

> “Integration differs from building distributed systems by lifecycle, team, and level of control.”

Messages and events: Time for some semantics

Constructing messages



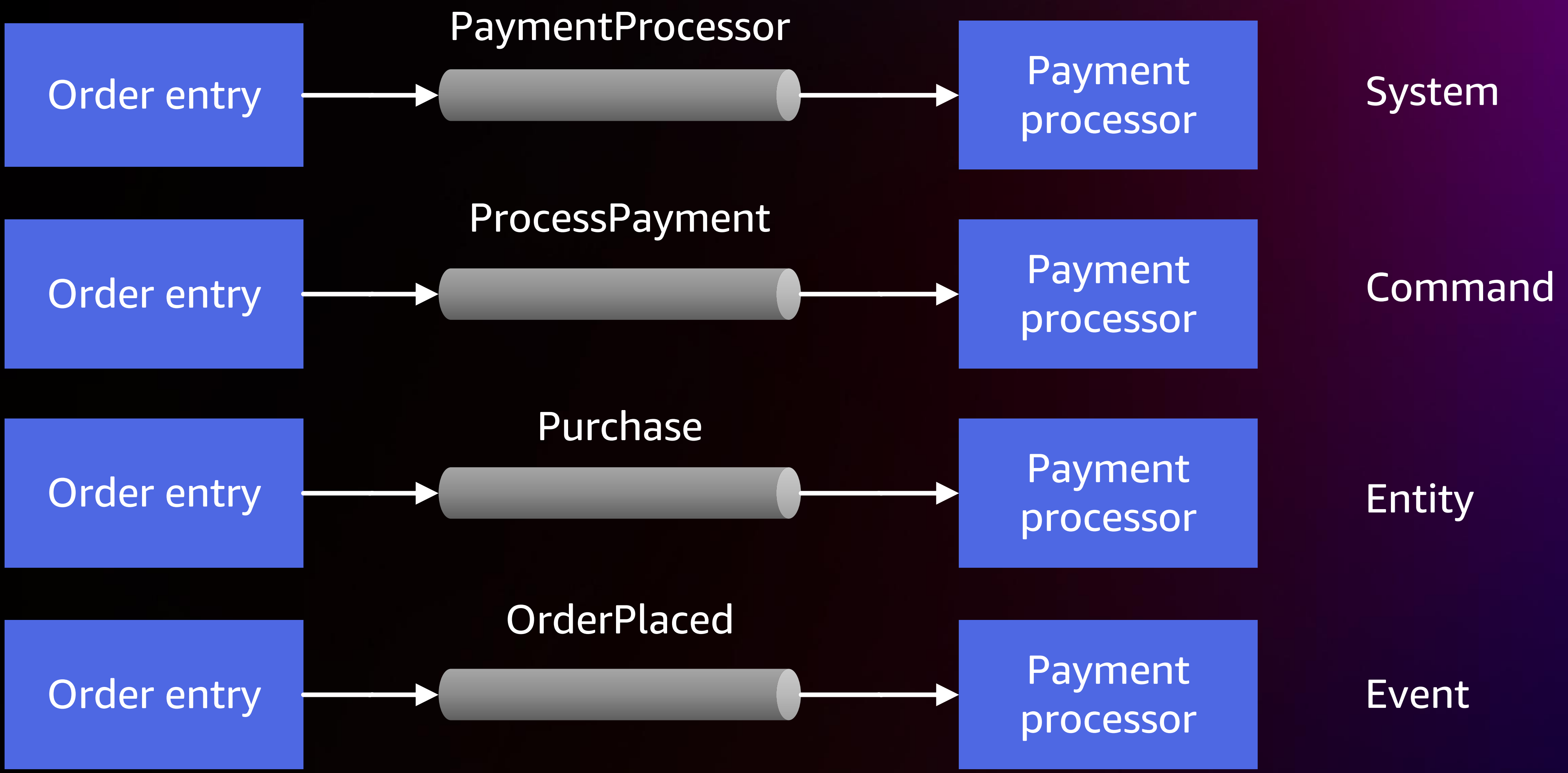
“Package the information into a *Message*, a data record that the messaging system can transmit through a *Message Channel*."

www.eaipatterns.com

* Martin Fowler: Beware of events used as a passive-aggressive commands: “When the source system expects the recipient to carry out an action, [it] ought to use a command message.”

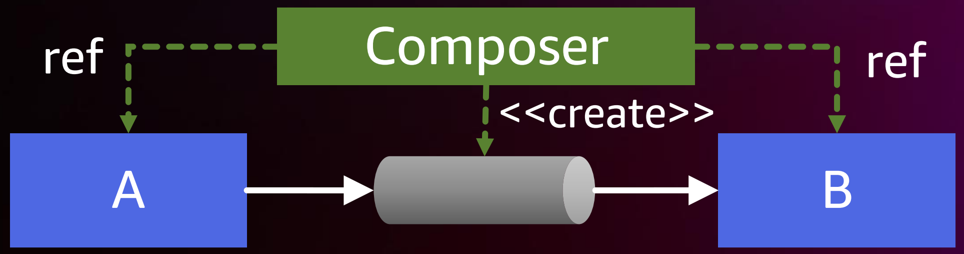
https://martinfowler.com/articles/201701-event-driven.html

Message channel semantics

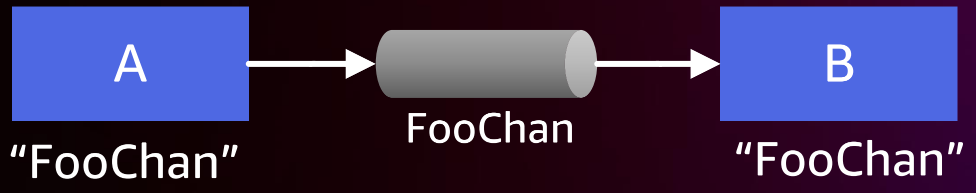


Matching producer and consumer

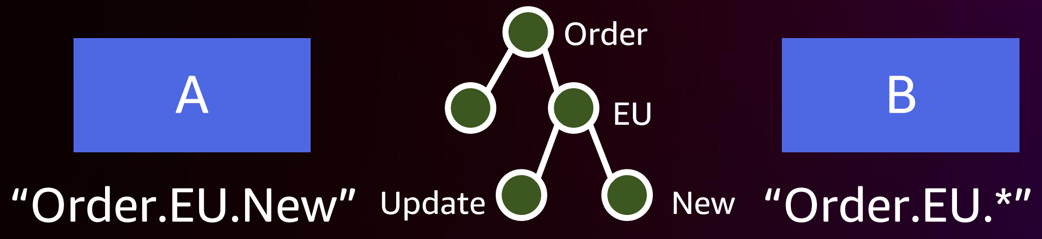
Channel instance
(explicit composition)



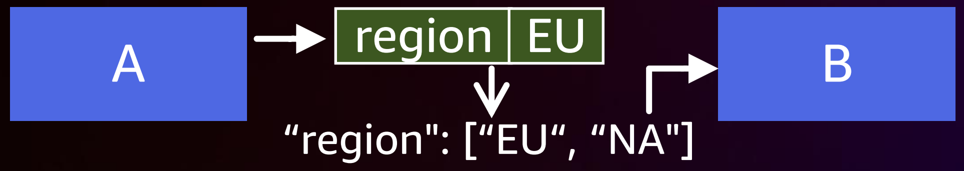
Channel name



Topic hierarchy



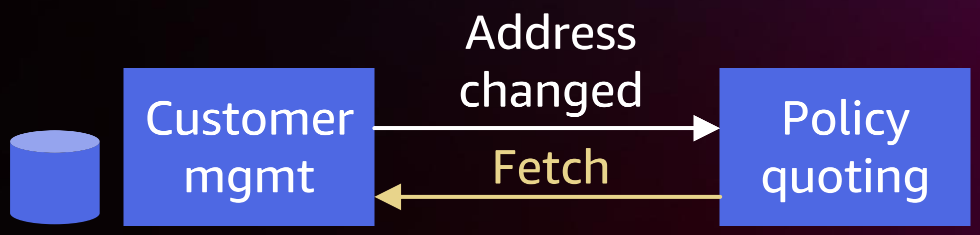
Content
(existence, value)



Event based, event driven, or event sourced?

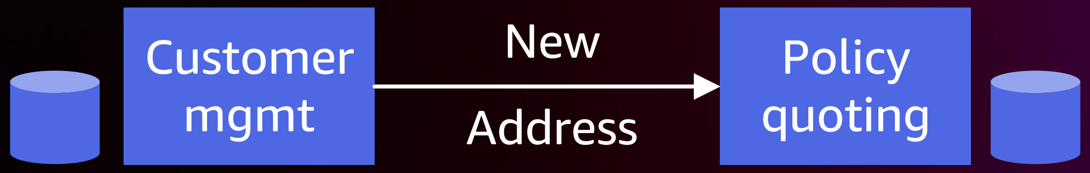
Event Notification

- Inverse dependencies
- Change as first class concern



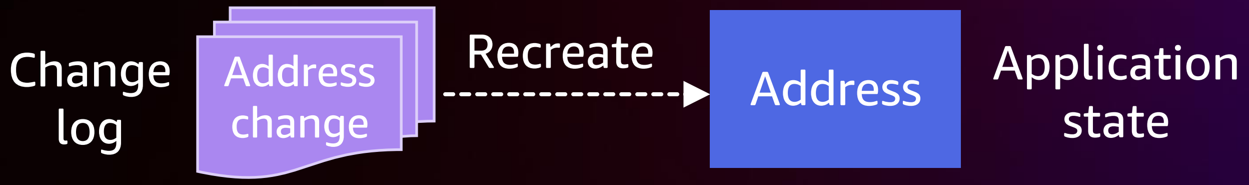
Event-carried state transfer

- Less chatty
- Resilience but inconsistency



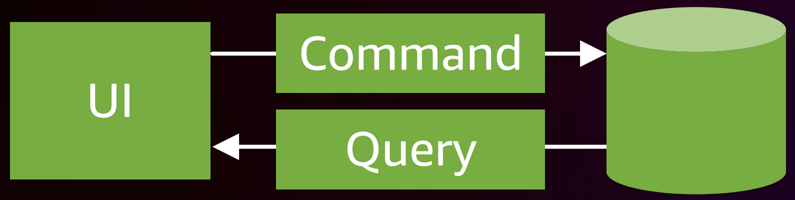
Event sourcing

- Ability to change the past
- Analog to version control, ledger



CQRS – Cmd/query segregation

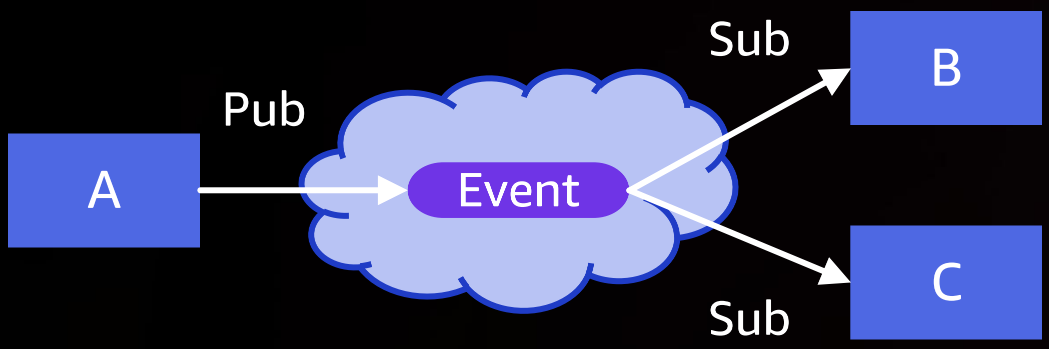
- Two models for updates vs reads
- Encapsulate complex validations

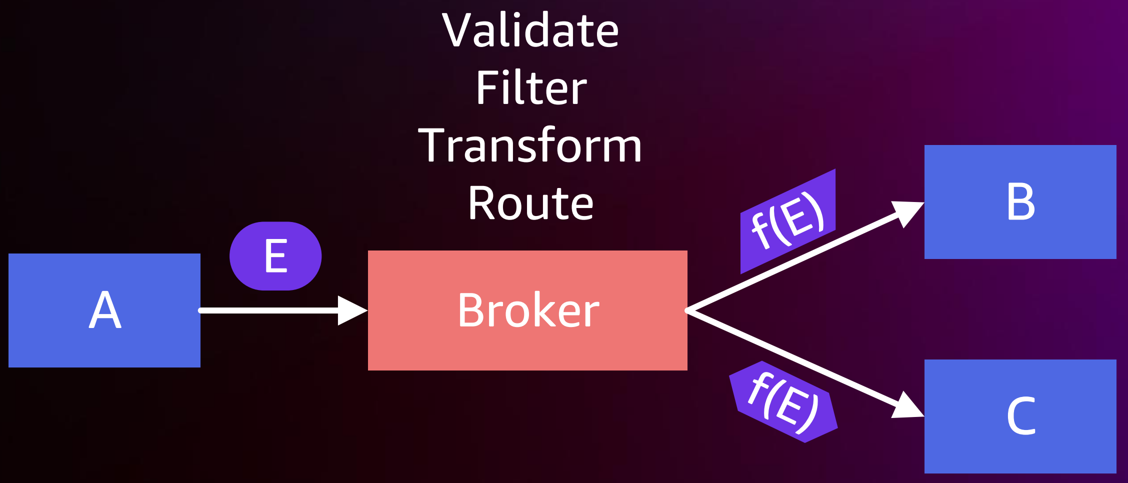


https://martinfowler.com/articles/201701-event-driven.html

Message coordination: Orchestration, choreography, brokerage

Event routing





Event cloud

- Fully decentralized, “purist”
- All responsibilities in endpoints
- Coupling may be hidden in endpoints
- Historically considered more scalable

Event broker

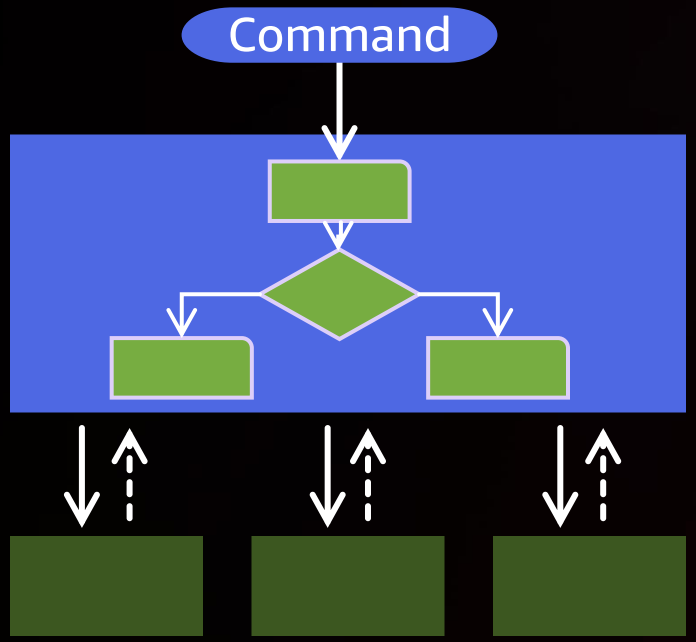
- Centralized element, “pragmatic”
- Structures event cloud
- Absorbs schema differences
- Scalability generally no longer an issue

Thanks to a Pipes-and-Filters architecture a producer doesn’t know if it’s talking to a broker or a directly to consumer

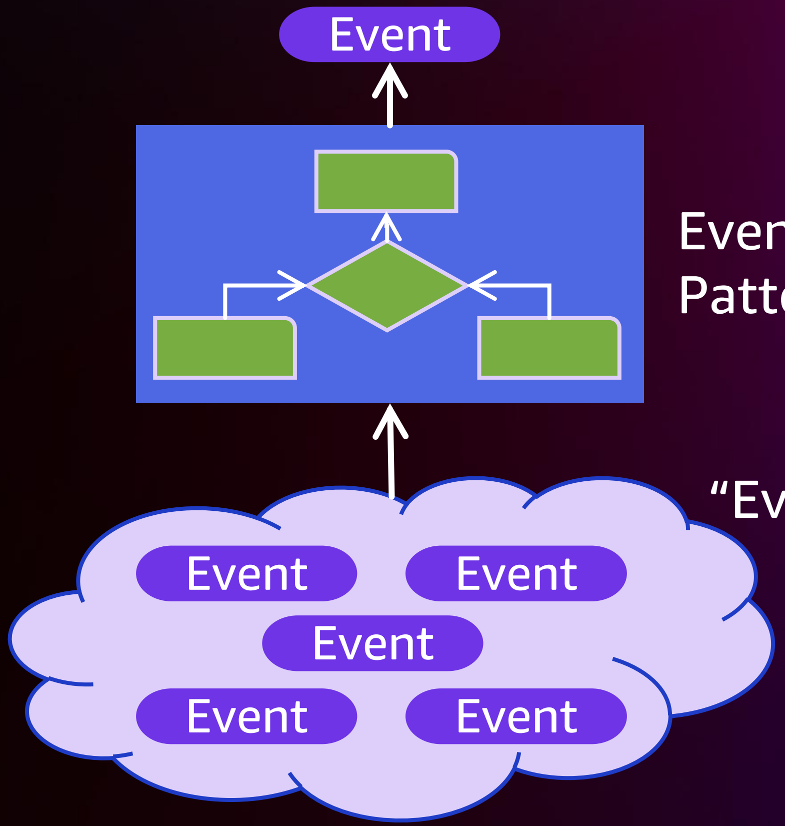
Orchestration vs. event processing

Control
central state

Passive
responders



Orchestration



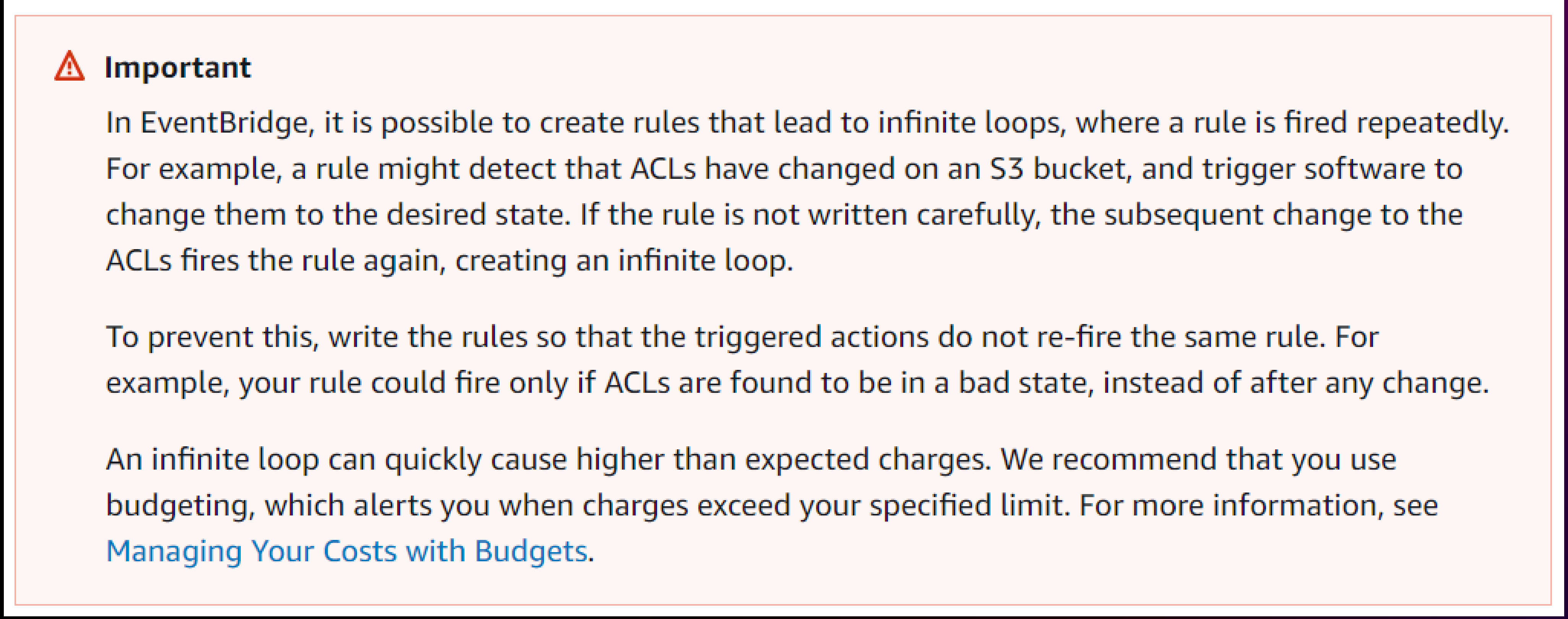
Event correlation
Pattern detection

“Event cloud”

Eventing
(Complex Event Processing)

What could possibly go wrong?

Event-based systems are dynamic in nature



⚠ **Important**

In EventBridge, it is possible to create rules that lead to infinite loops, where a rule is fired repeatedly. For example, a rule might detect that ACLs have changed on an S3 bucket, and trigger software to change them to the desired state. If the rule is not written carefully, the subsequent change to the ACLs fires the rule again, creating an infinite loop.

To prevent this, write the rules so that the triggered actions do not re-fire the same rule. For example, your rule could fire only if ACLs are found to be in a bad state, instead of after any change.

An infinite loop can quickly cause higher than expected charges. We recommend that you use budgeting, which alerts you when charges exceed your specified limit. For more information, see Managing Your Costs with Budgets.

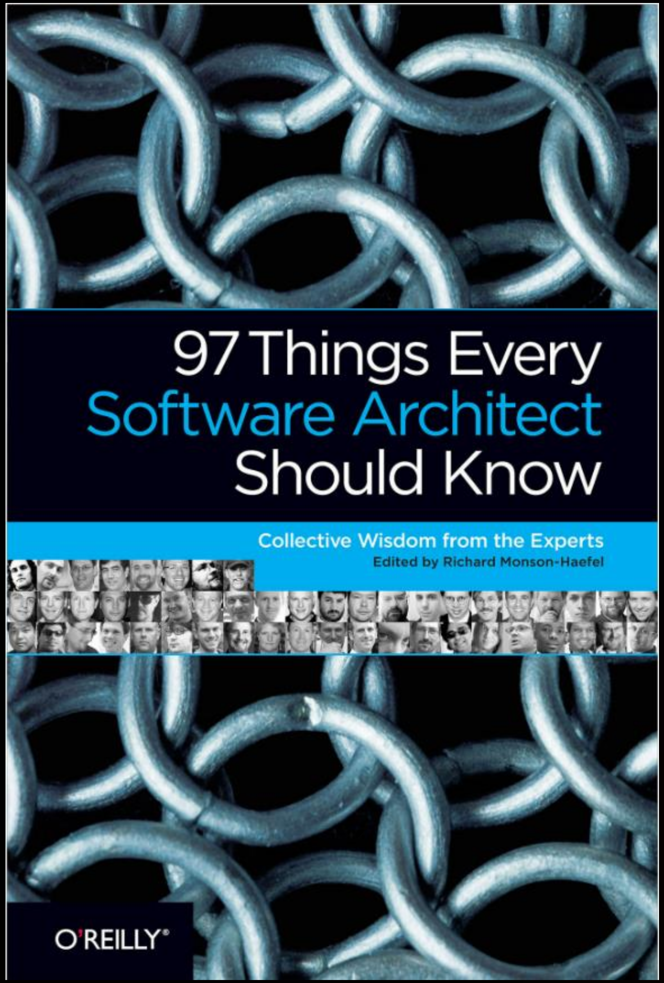
https://docs.aws.amazon.com/eventbridge/latest/userguide/eb-event-patterns.html

“A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.”

Leslie Lamport

Microsoft Research

Don’t control but observe

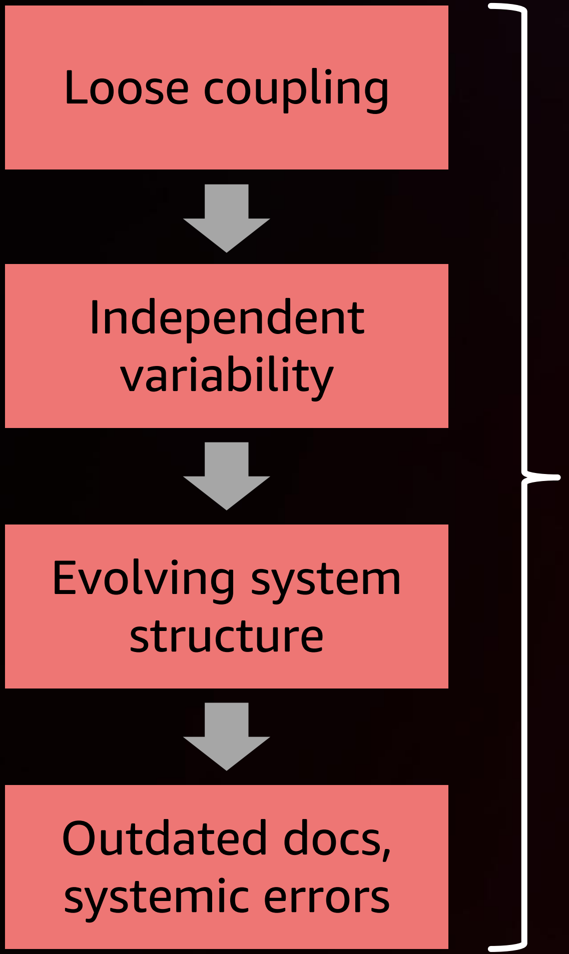


97 Things Every Software Architect Should Know

Collective Wisdom from the Experts

Edited by Richard Monson-Haefel

O'REILLY®

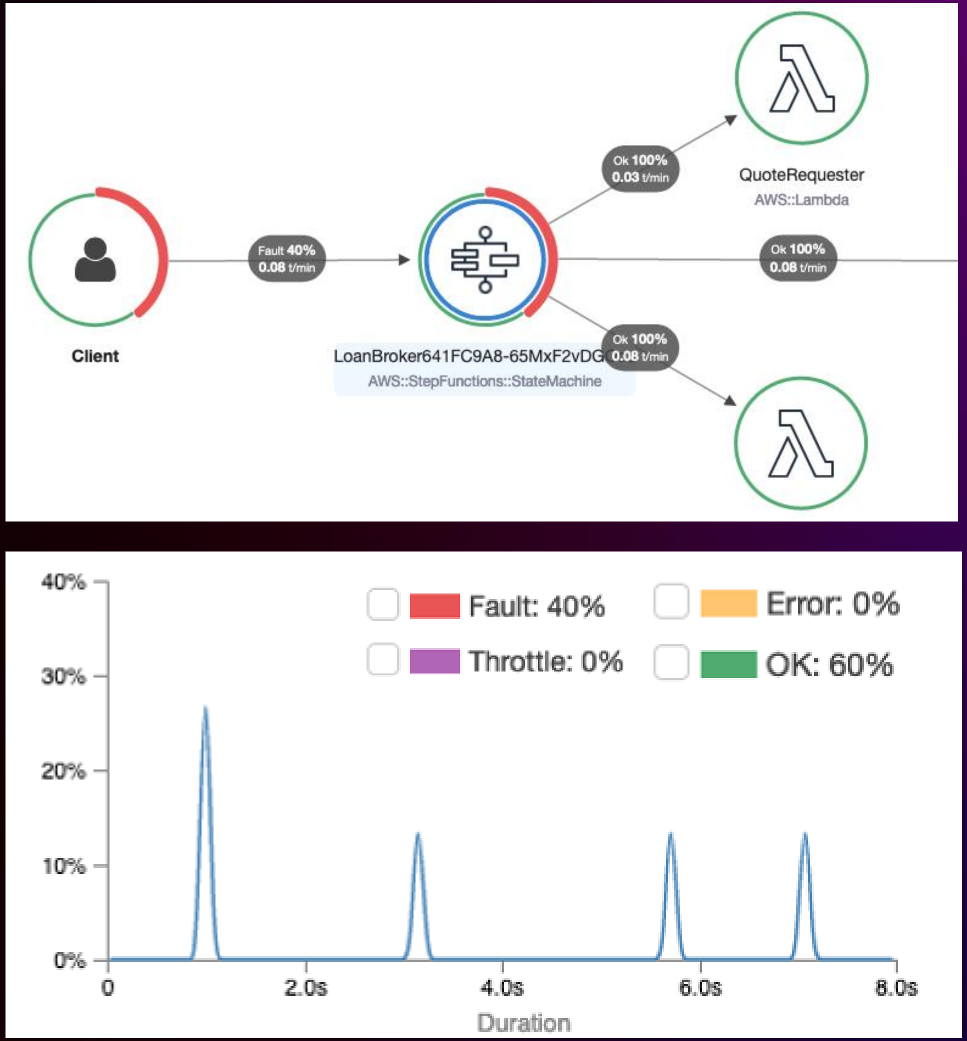


Loose coupling

Independent variability

Evolving system structure

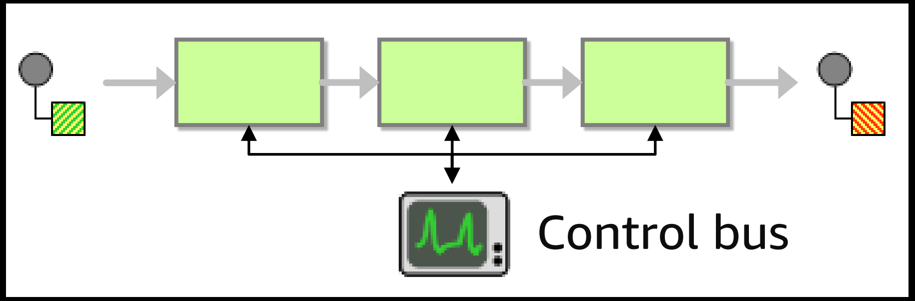
Outdated docs, systemic errors

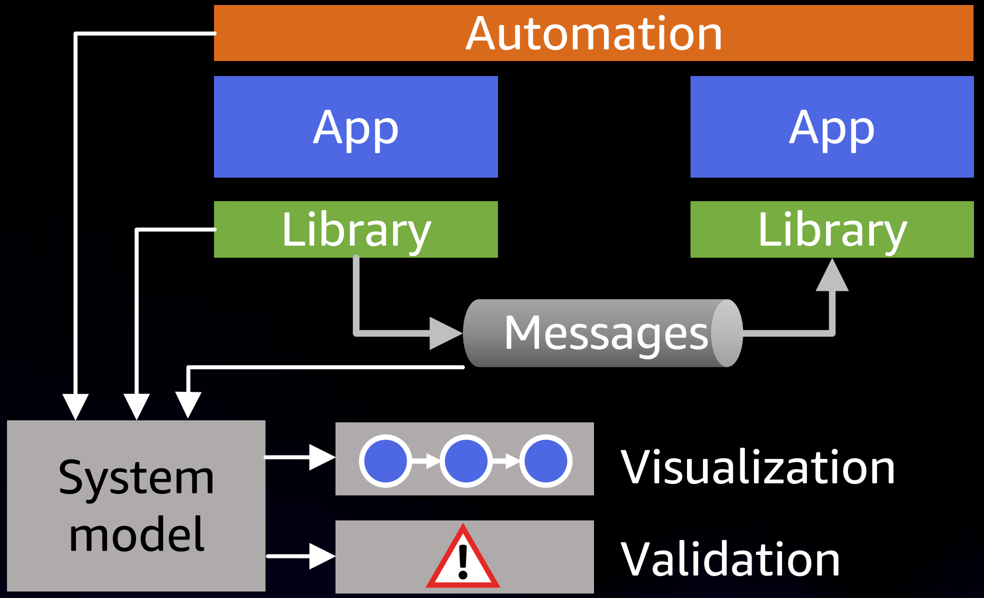


AWS X-Ray – distributed tracing

Shifting control to the runtime – Control bus

- Building a system model
  - At connect (open/sub) time
  - Based on message flow
- Static validation
  - Missing subscriptions
  - Loops
- Dynamic validation
  - Infinite retries
  - Poison messages
  - Surging queues





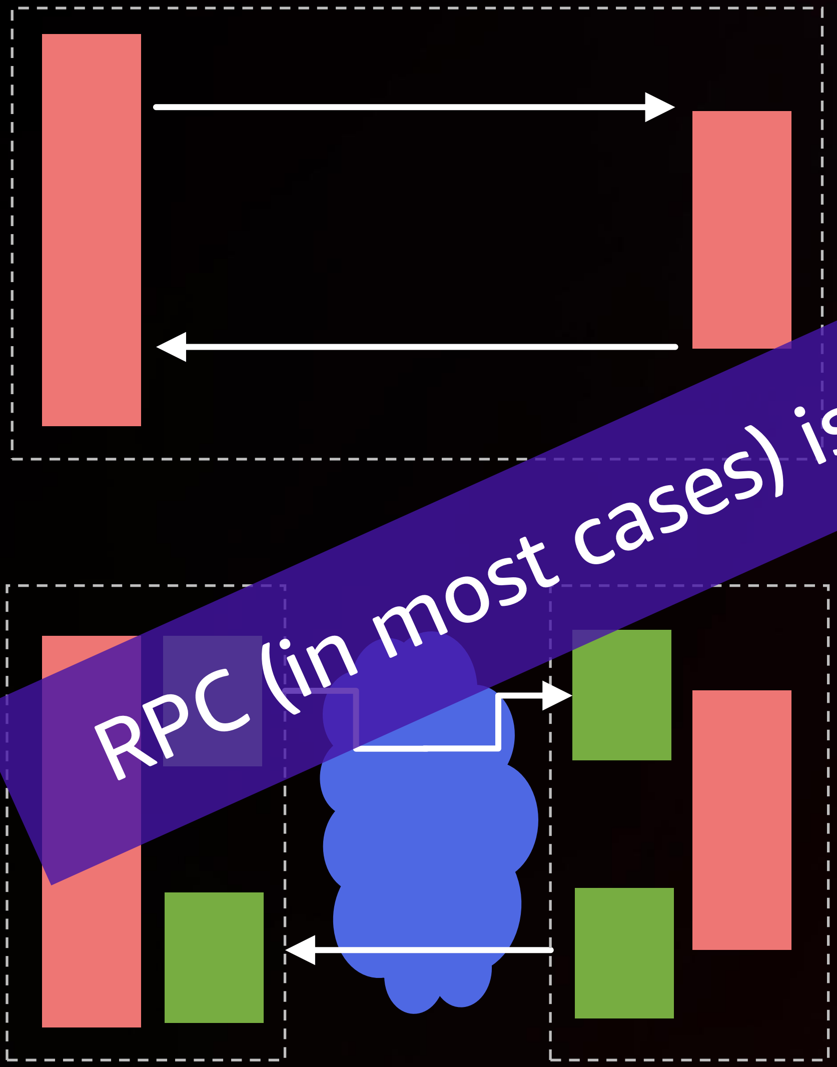
https://www.enterpriseintegrationpatterns.com/ramblings/48_validation.html

Abstractions vs illusions

RPC – Remote procedure call

Local call

Remote call



RPC (in most cases) is a dangerous illusion.

1. Single process
2. Negligible latency
3. Call stack built-in
4. Same language and data types
5. No partial failure

Fallacies of distributed computing:

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

Failure (and physics) don’t respect abstractions

Abstraction:

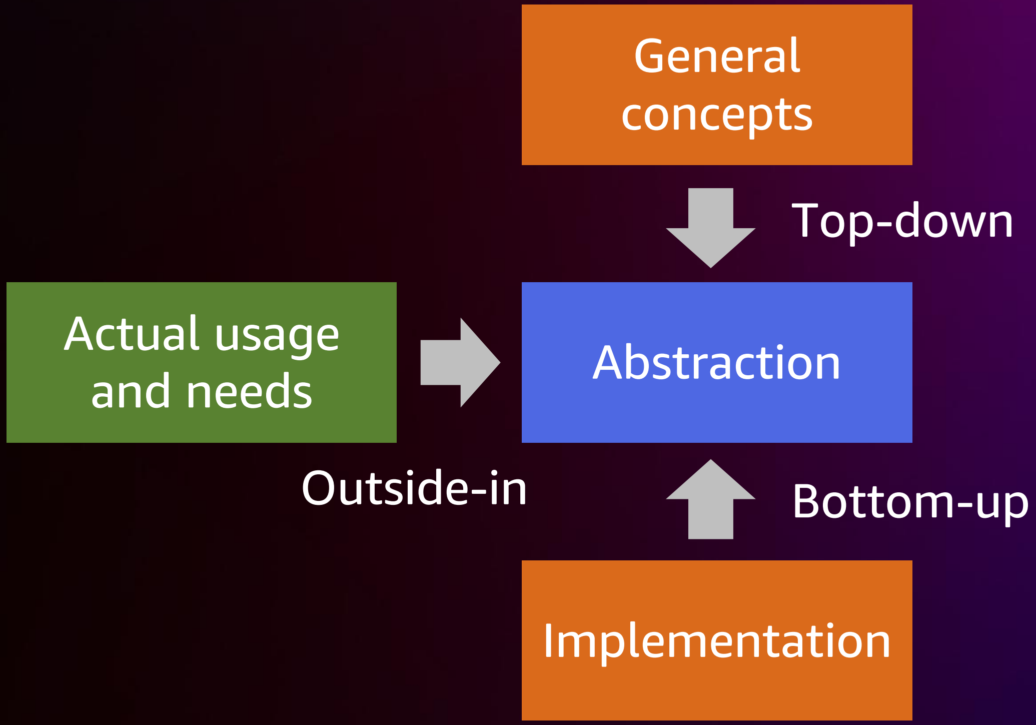
Removing or generalizing details or attributes to focus attention on details of greater importance

Illusion:

Removing or generalizing important details, which cause the user to be misled

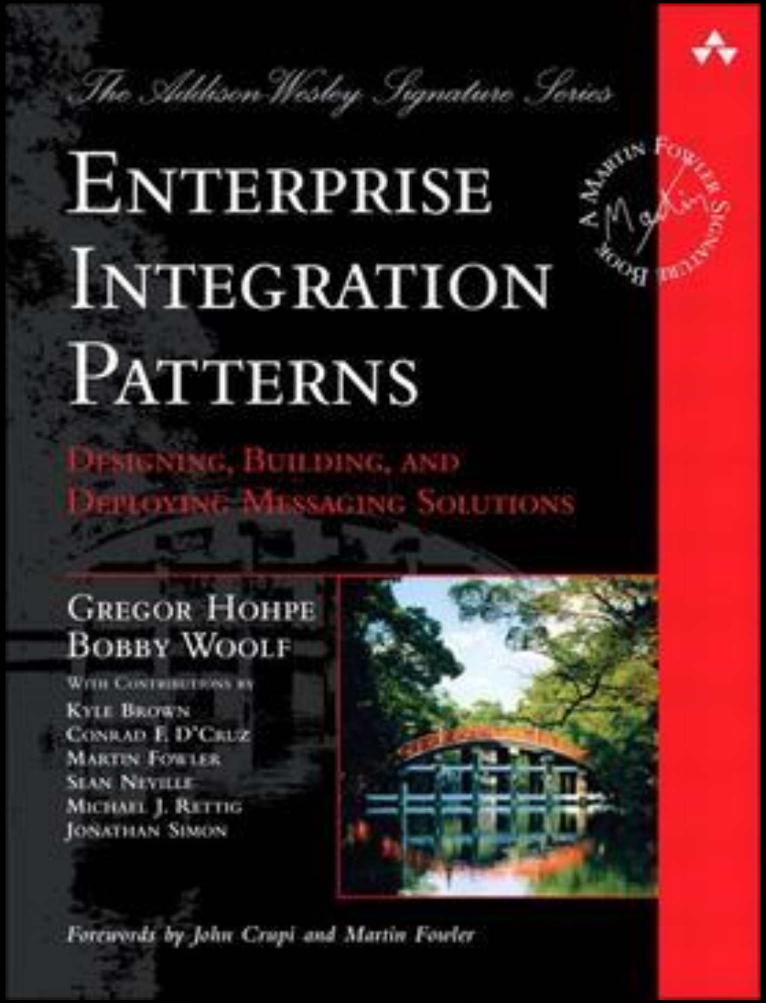
Law of leaky abstractions (Joel Spolsky):

All non-trivial abstractions, to some degree, are leaky



https://architectelevator.com/architecture/failure-doesnt-respect-abstraction
https://architectelevator.com/cloud/abstractions-difficult
https://www.joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions/

Enterprise Integration Patterns (2003)



The Addison-Wesley Signature Series

ENTERPRISE INTEGRATION PATTERNS

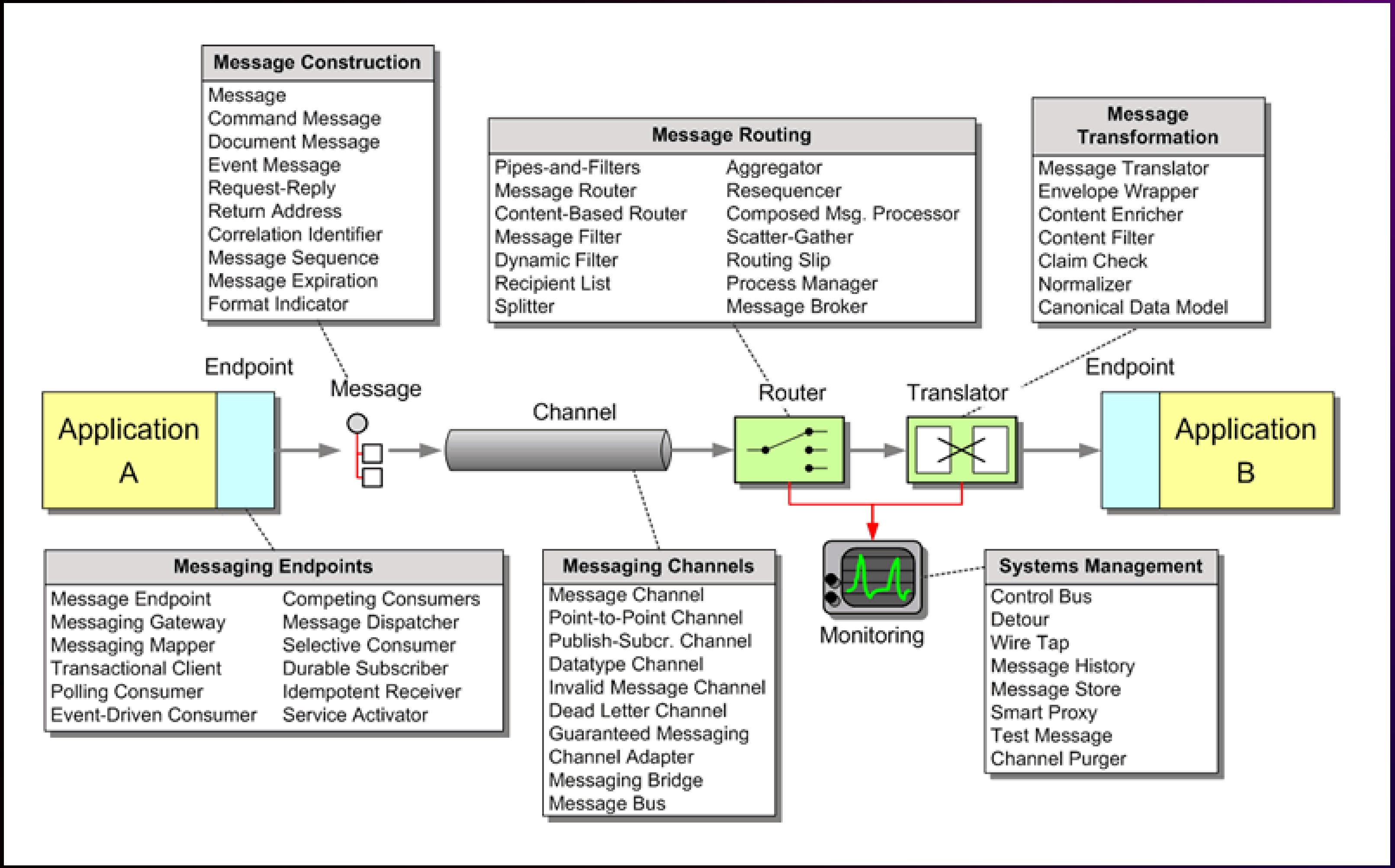
DESIGNING, BUILDING, AND DEPLOYING MESSAGING SOLUTIONS

GREGOR HOHPE
BOBBY WOOLF

WITH CONTRIBUTIONS BY
KYLE BROWN
CONRAD F. D'CRUZ
MARTIN FOWLER
SEAN NEVILLE
MICHAEL J. RETTIG
JONATHAN SIMON

Forewords by John Crupi and Martin Fowler

A MARTIN FOWLER SIGNATURE BOOK



Message Construction
Message
Command Message
Document Message
Event Message
Request-Reply
Return Address
Correlation Identifier
Message Sequence
Message Expiration
Format Indicator

Message Routing
Pipes-and-Filters
Message Router
Content-Based Router
Message Filter
Dynamic Filter
Recipient List
Splitter
Aggregator
Resequencer
Composed Msg. Processor
Scatter-Gather
Routing Slip
Process Manager
Message Broker

Message Transformation
Message Translator
Envelope Wrapper
Content Enricher
Content Filter
Claim Check
Normalizer
Canonical Data Model

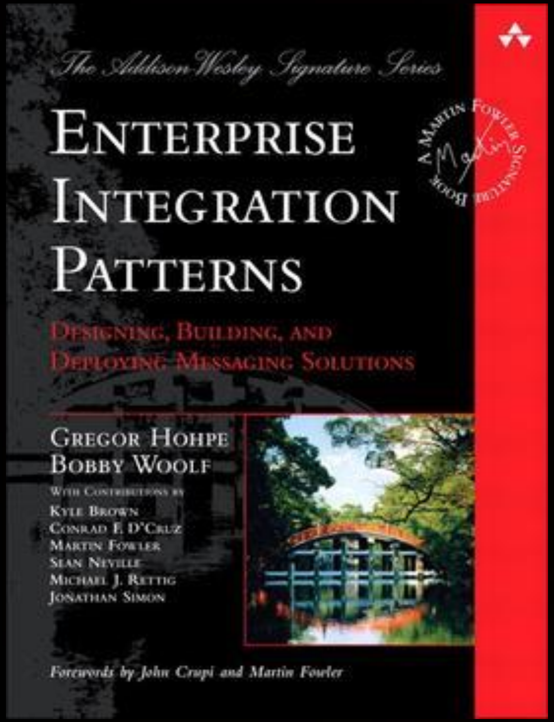
Messaging Endpoints
Message Endpoint
Messaging Gateway
Messaging Mapper
Transactional Client
Polling Consumer
Event-Driven Consumer
Competing Consumers
Message Dispatcher
Selective Consumer
Durable Subscriber
Idempotent Receiver
Service Activator

Messaging Channels
Message Channel
Point-to-Point Channel
Publish-Subscr. Channel
Datatype Channel
Invalid Message Channel
Dead Letter Channel
Guaranteed Messaging
Channel Adapter
Messaging Bridge
Message Bus

Systems Management
Control Bus
Detour
Wire Tap
Message History
Message Store
Smart Proxy
Test Message
Channel Purger

Abstractions, cloud style

The integration patterns loan broker (2003–2022)



The Addison-Wesley Signature Series

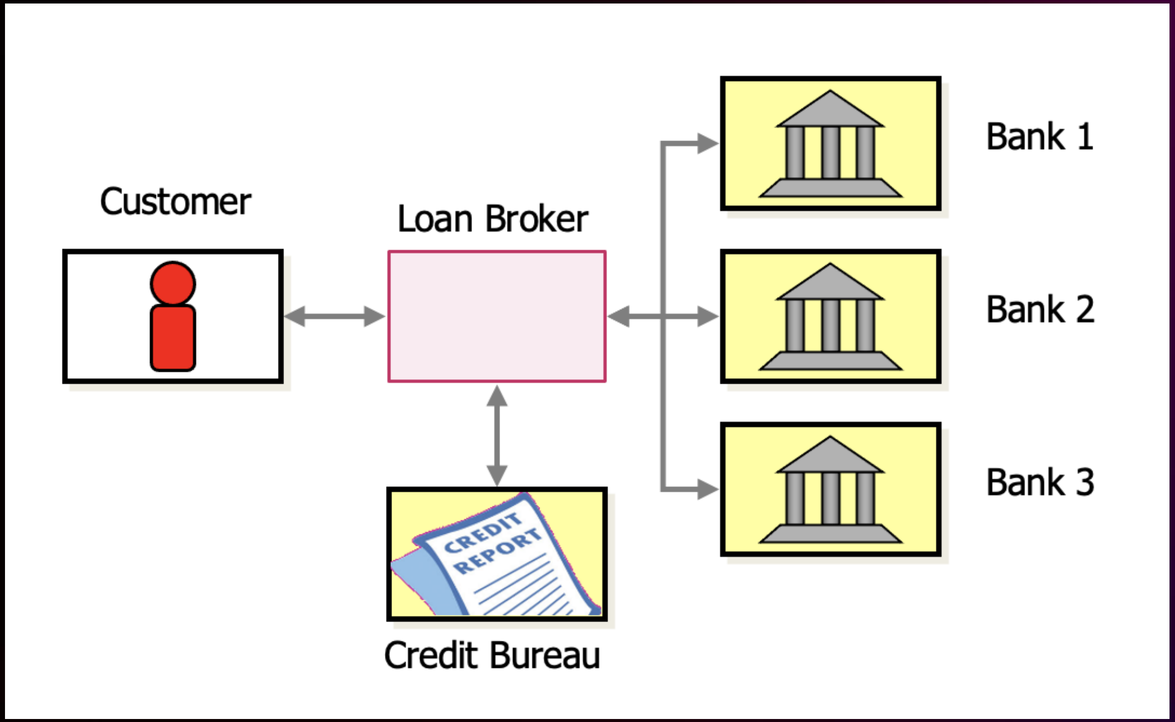
Enterprise Integration Patterns

Designing, Building, and Deploying Messaging Solutions

Gregor Hohpe
Bobby Woolf

With Contributions by
Kyle Brown
Conrad F. D'Cruz
Martin Fowler
Sean Neville
Michael J. Rettig
Jonathan Simon

Forewords by John Crupi and Martin Fowler



Customer

Loan Broker

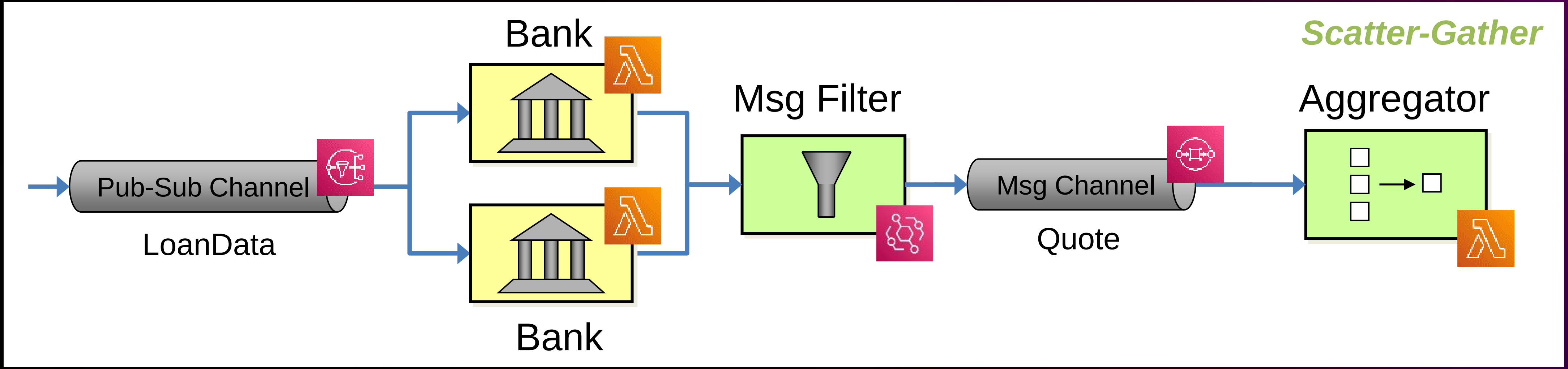
Credit Bureau

Bank 1

Bank 2

Bank 3

A cloud-native, serverless implementation



Business domain constructs

- Bank
- Loan broker
- LoanQuote

Integration/event patterns

- Message filter
- Content filter
- Aggregator
- Publish-subscribe

AWS CloudFormation resources

- Lambda function
- Amazon SQS/Amazon SNS queues & topics
- AWS Step Functions tasks
- Amazon EventBridge rules

Serverless composition with AWS CDK

Business domain constructs

Config

```
const bankRecipientPawnshop = this.createBank(
  'BankRecipientPawnshop', { BANK_ID: 'PawnShop', BASE_RATE: '5',
  MAX_LOAN_AMOUNT: '500000', MIN_CREDIT_SCORE: '400‘ }, mortgageQuotesBus);
```

Composition

Integration/ event patterns

```
nonEmptyQuotesOnly = MessageFilter.fieldExists(this, 'nonEmptyQuotes’, ‘bankId’);
payloadOnly = ContentFilter.payloadFilter(this, 'PayloadContentFilter’);

new MessageContentFilter(this, 'FilterMortgageQuotes’,
    { sourceEventBus: mortgageQuotesEventBus, targetQueue: mortgageQuotesQueue,
      messageFilter: nonEmptyQuotesOnly, contentFilter: payloadOnly });
```

Serverless composition with AWS CDK

```
new ChoreographyBuilder(this)
    .fromQueue(mortgageQuotesQueue)
    .scatterGather([ bankPawnshop, bankUniversal, bankPremium ])
    .messageFilterIfFieldExists("bankId")
    .contentFilter("$.detail.responsePayload")
    .aggregate({
       condition: AggregatorCondition.MIN_COUNT,
       threshold: 2,
       aggregation: AggregatorStrategy.APPEND,
     })
    .toQueue(loanQuoteQueue);
```

Describe composition and intent instead of provisioning resources

Code the lines, not just the boxes

> “CDK automation and abstraction allows you to code your application topology using design patterns as vocabulary.”

Time for a recap – Two decades on one slide

- Lines are at least as interesting as the boxes.
- Don’t let the products choices define your architecture. You’re the chef!
- Coupling has many facets. The right level of design-time coupling depends on the level of control you have.
- Integration vs. distributed systems isn’t a technical distinction but about lifecycle, org structures, and level of control.
- Events are messages. They invert dependencies from producer to consumer.
- Not all “Event Architectures” are created the same.
- The runtime – don’t control but observe.
- Build abstractions, not illusions!
- Code your cloud abstractions – with AWS CDK and patterns.

Want more?

Architect Elevator Blog

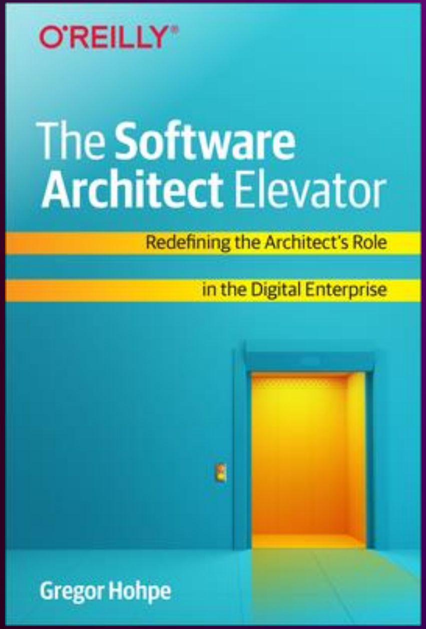
ArchitectElevator.com

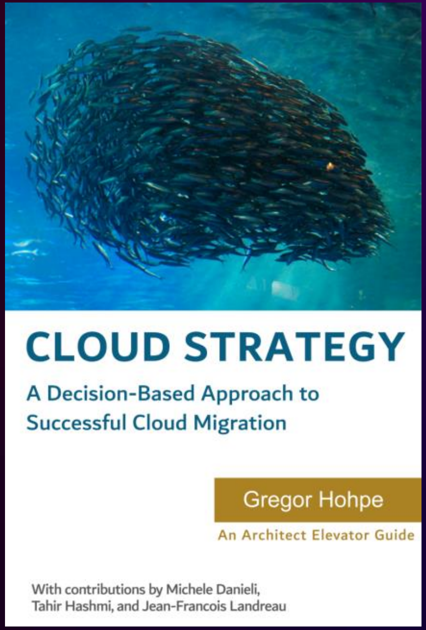
- Multi-cloud: From Buzzword to Decision Model
- Concerned about Serverless Lock-in? Consider Patterns!
- Good abstractions are obvious but difficult to find, even in the cloud.

EnterpriseIntegrationPatterns.com

- Loan Broker on AWS Serverless
- Integration patterns with CDK

linkedin.com/in/ghohpe/


O'REILLY
The Software Architect Elevator
Redefining the Architect's Role in the Digital Enterprise
Gregor Hohpe


CLOUD STRATEGY
A Decision-Based Approach to Successful Cloud Migration
Gregor Hohpe
An Architect Elevator Guide
With contributions by Michele Danieli, Tahir Hashmi, and Jean-Francois Landreau

Thank you!

Gregor Hohpe

@ghohpe

ArchitectElevator.com

linkedin.com/in/ghohpe/

Please complete the session survey in the **mobile app**