



Aggregates

What If...

CSRs often experience a slow system and are sometimes unable to save changes to Contracts*

* because a back-end batch process is simultaneously bulk-updating RepairPOs (based on changes from Servicer Management context).

Concurrency is very
tricky to get right, and
can really negatively
impact performance

Question

If we update a Repair PO,
what else do we need to
save as part of that
transaction?

Transactional
boundaries are often
ad-hoc and create
sprawling dependencies

Question

How would we handle storing
RepairPOs on a different
database cluster from Contracts
and Claims?

Distribution choices
have a huge impact on
scaling, responsiveness
and overall
performance

What If...

The address changes for a ClientCompany and we need to update every Contract with this ClientCompany change?*

* and do it during business hours, without negatively impacting system responsiveness for the CSRs

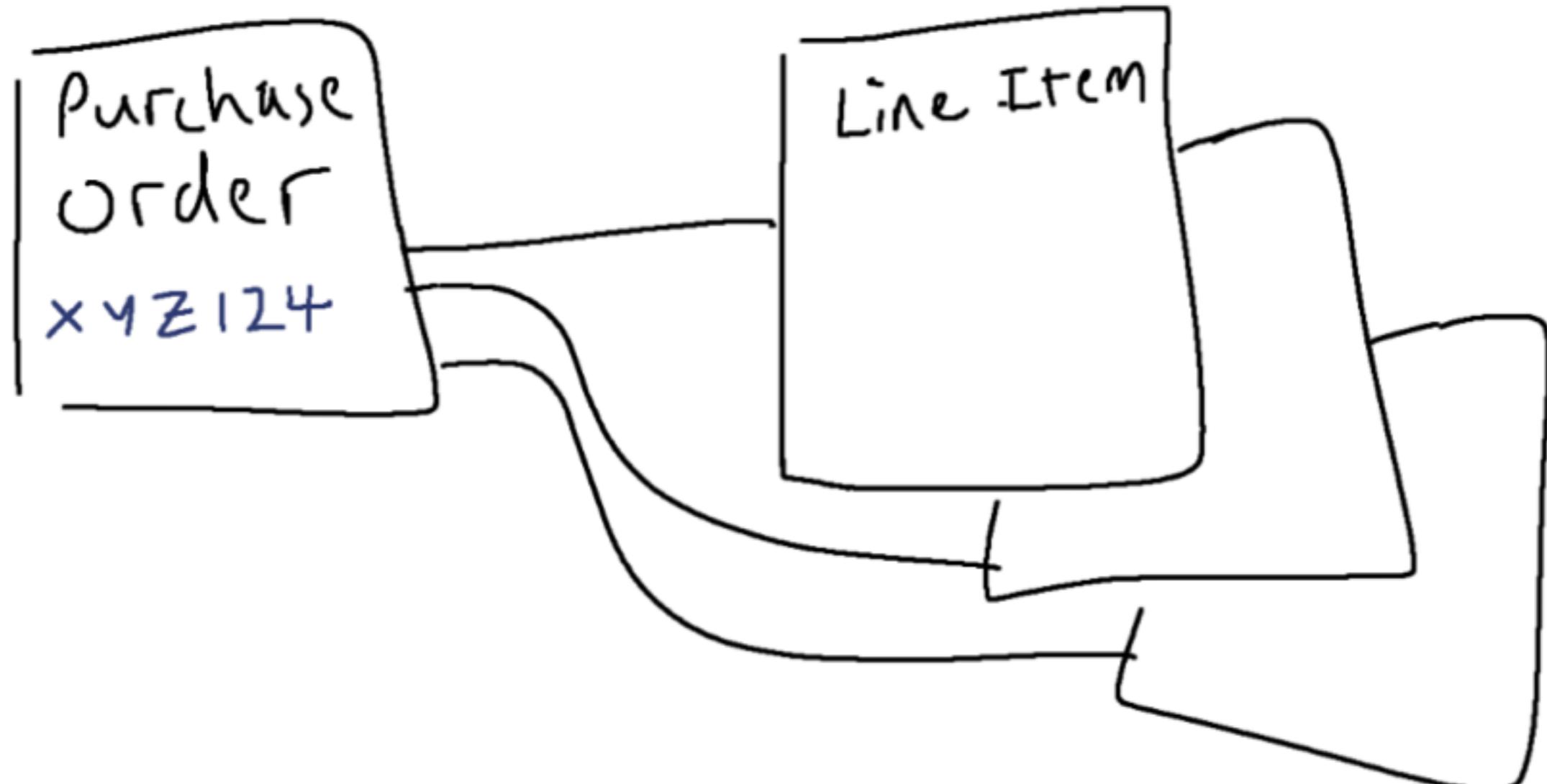
Building performant,
scalable, robust,
distributed systems
requires embracing
decoupling and latency

Modeling Boundaries

Transaction
Distribution
Concurrency

Aggregate

A cluster of domain objects
that can be treated as a
single unit

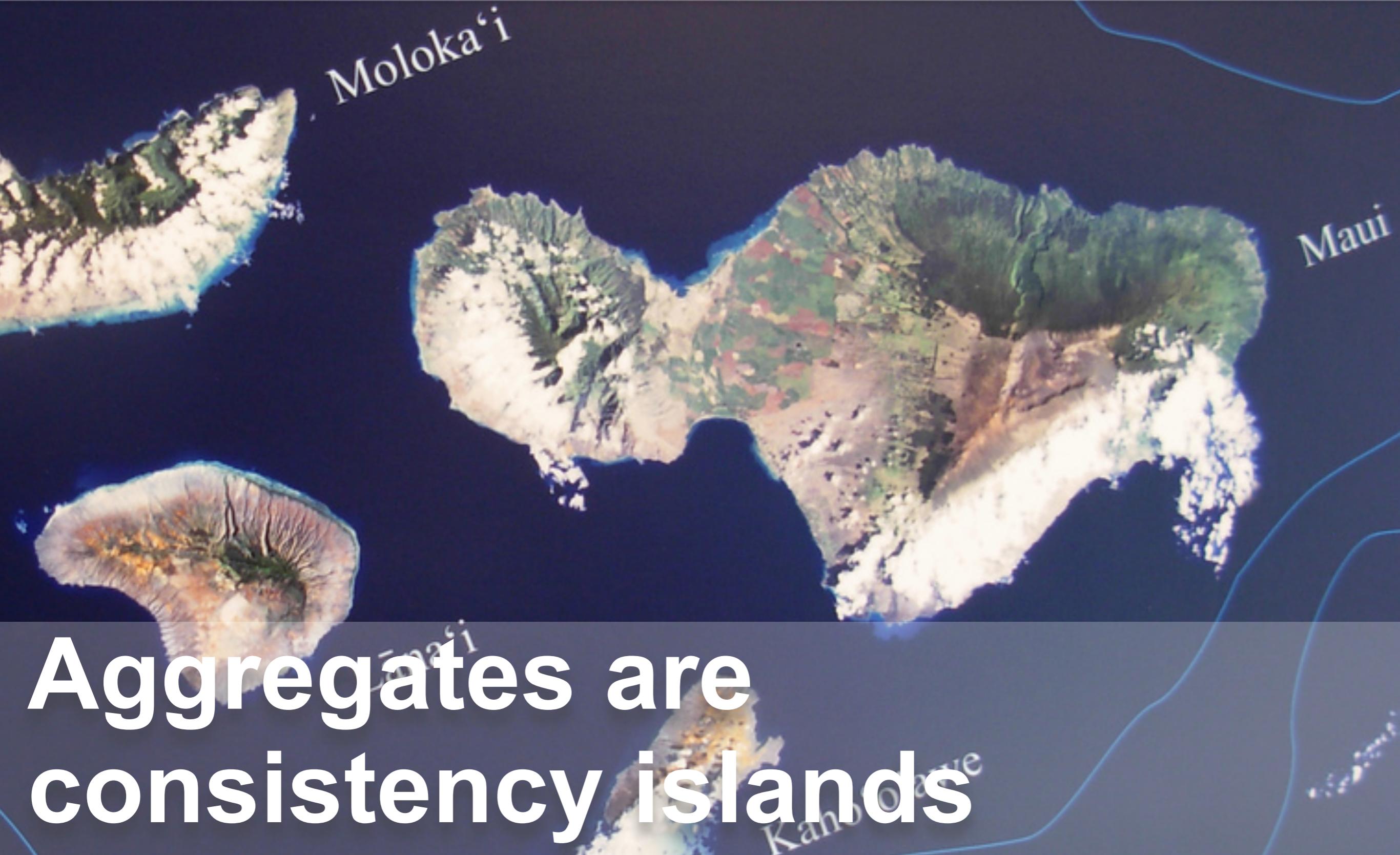


Aggregates are always internally consistent.

- invariants apply at every transaction commit

Aggregates are always internally consistent.

Invariants apply at every transaction commit



Aggregates are
consistency islands



5 Guidelines for Aggregate Design

Guideline #1

Focus on identifying true
invariants to clarify
aggregate boundaries

Sample Invariants

Contract must always have valid:

- ClientCompany
- Product
- TermsAndConditions

Repair PO must always have at least one LineItem

Guideline #2

Design small,
cohesive
aggregates

Avoid large, ad-hoc, sprawling
transactional, distribution and
concurrency boundaries

Guideline #3

Reference other
aggregates by
identity

Aggregate Root

An entity that serves as the identity holder, namesake, and gatekeeper for the aggregate.

Use the delete rule of thumb to assist you in identifying the root of an aggregate.

Use the Factory pattern to construct more complex aggregates to ensure invariants are enforced correctly.

What Needs to Change...

To separate RepairPurchaseOrders, Claims and Contracts into separate aggregates?

Guideline #4

Only hold references to
aggregate roots
(i.e. limit interconnectedness)

New Invariants

- *A pending Contract cannot have any Claims.*
 - *Only one open Claim at at time per active Contract.*
 - *Sum of costs of all Claims for a Contract must never exceed the LimitOfLiability.*
-
- What needs to change in your design to make Claims part of the Contract aggregates again?
 - How well does your design handle relating each RepairPO to its associated Claim, especially when there are multiple Claims for a Contract?

Guideline #5

Use eventual
consistency across
aggregate boundaries

Aggregates are “eventually consistent” with each other.

- asynchronous updates propagate through system

Propagating Changes

For CustomerReimbursed, walk through how we might
we ensure the:

- Contract is terminated?
- Associated Claims are all closed?

For LimitOfLiabilityExceeded:



Aggregate
Persistance

Aggregates are the basic element of transfer of data storage

- you request to load or save *whole aggregates*.

Repositories are services
that provide access to
stored aggregates in terms
of the ubiquitous language

*Encapsulate actual storage
and query technology
inside repositories*

Pramodkumar J Sadalage & Martin Fowler

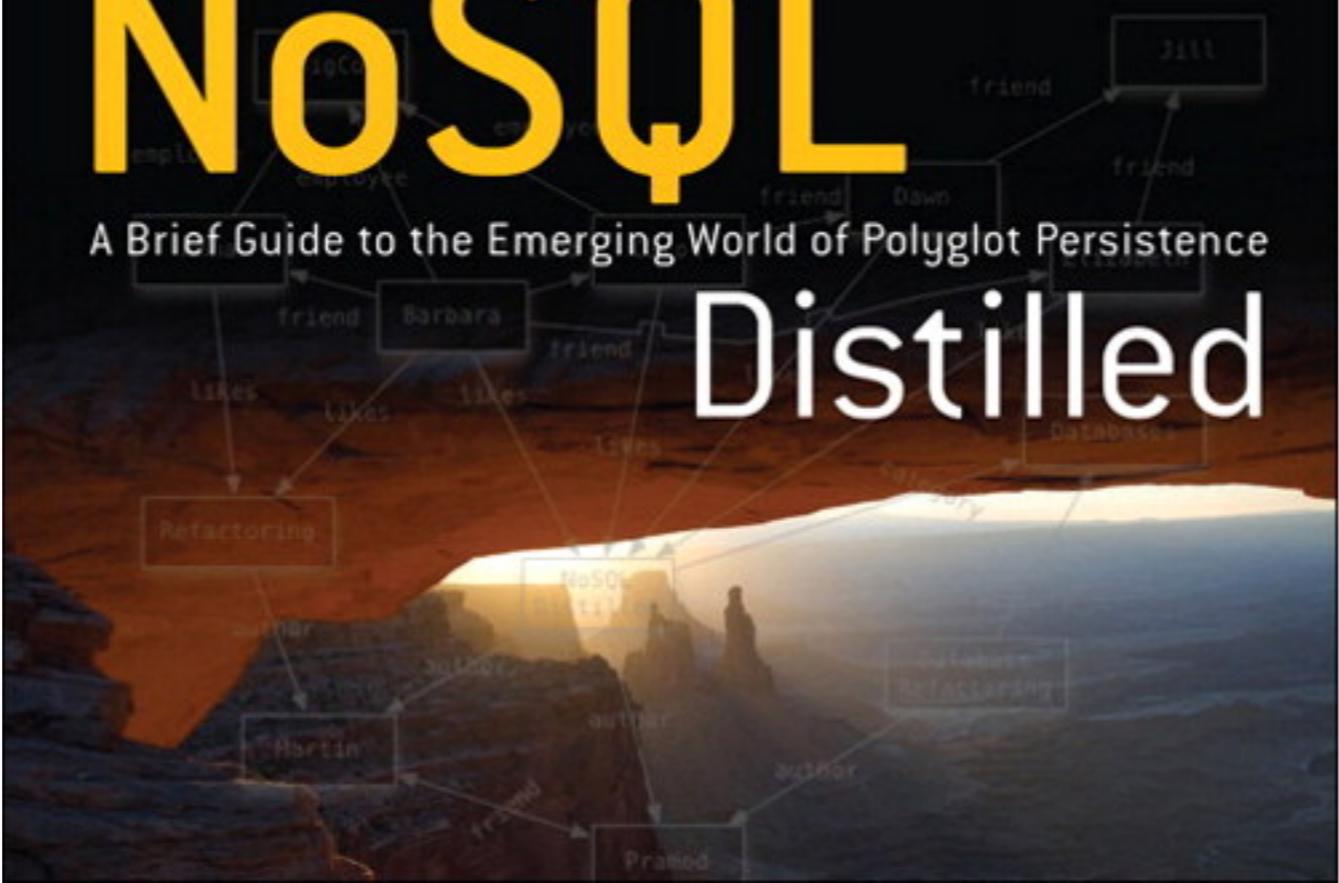
```
( "firstname": "Pramod",
  "citiesvisited": [ "Chicago", "London", "Pune", "Bangalore" ],
  "addresses": [
    { "state": "AK",
      "city": "DILLINGHAM",
      "type": "R"
    },
    { "state": "MH",
      "city": "PUNE",
      "type": "R"
    }
  ],
  "lastcity": "Chicago"
)
```

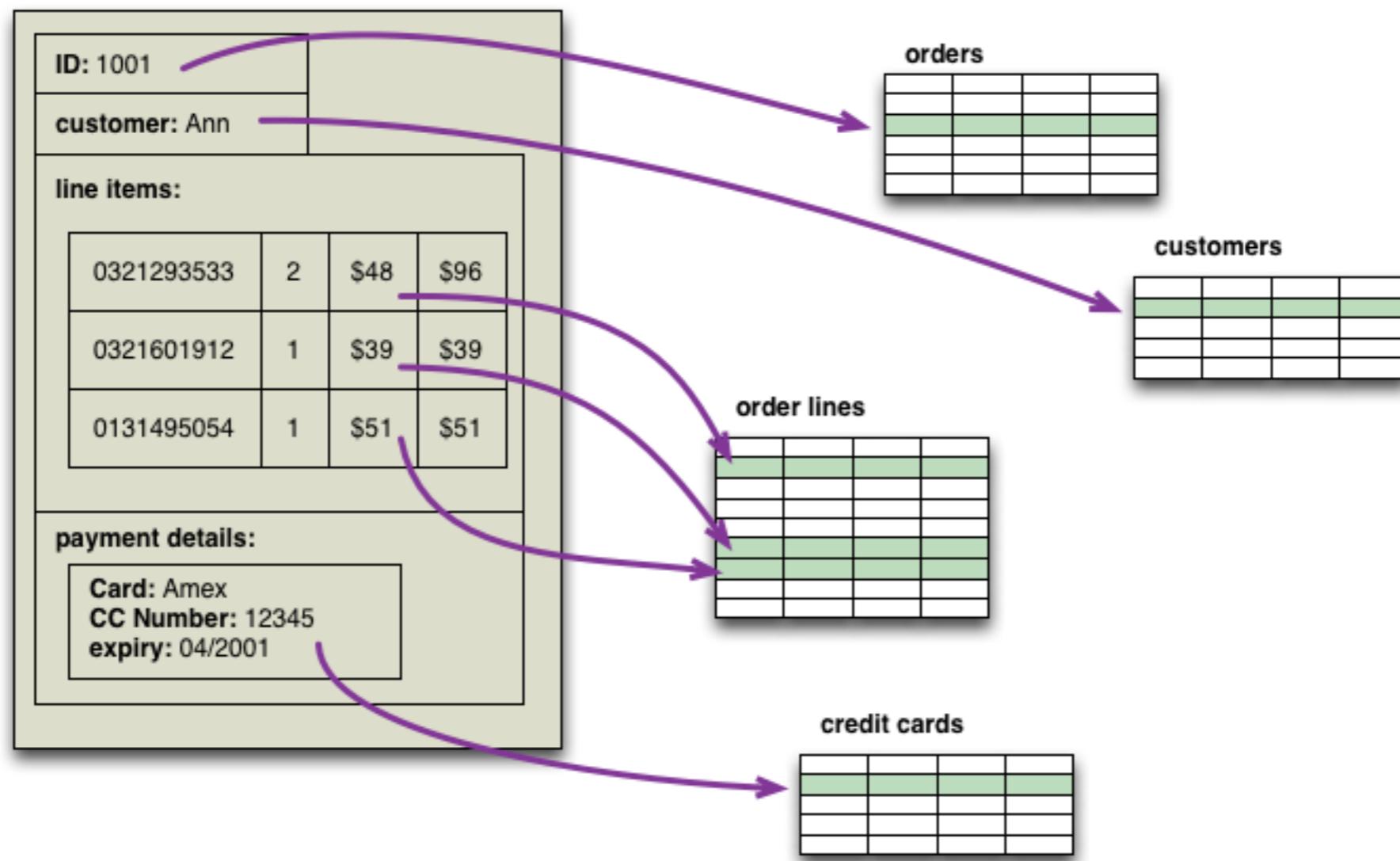


NoSQL

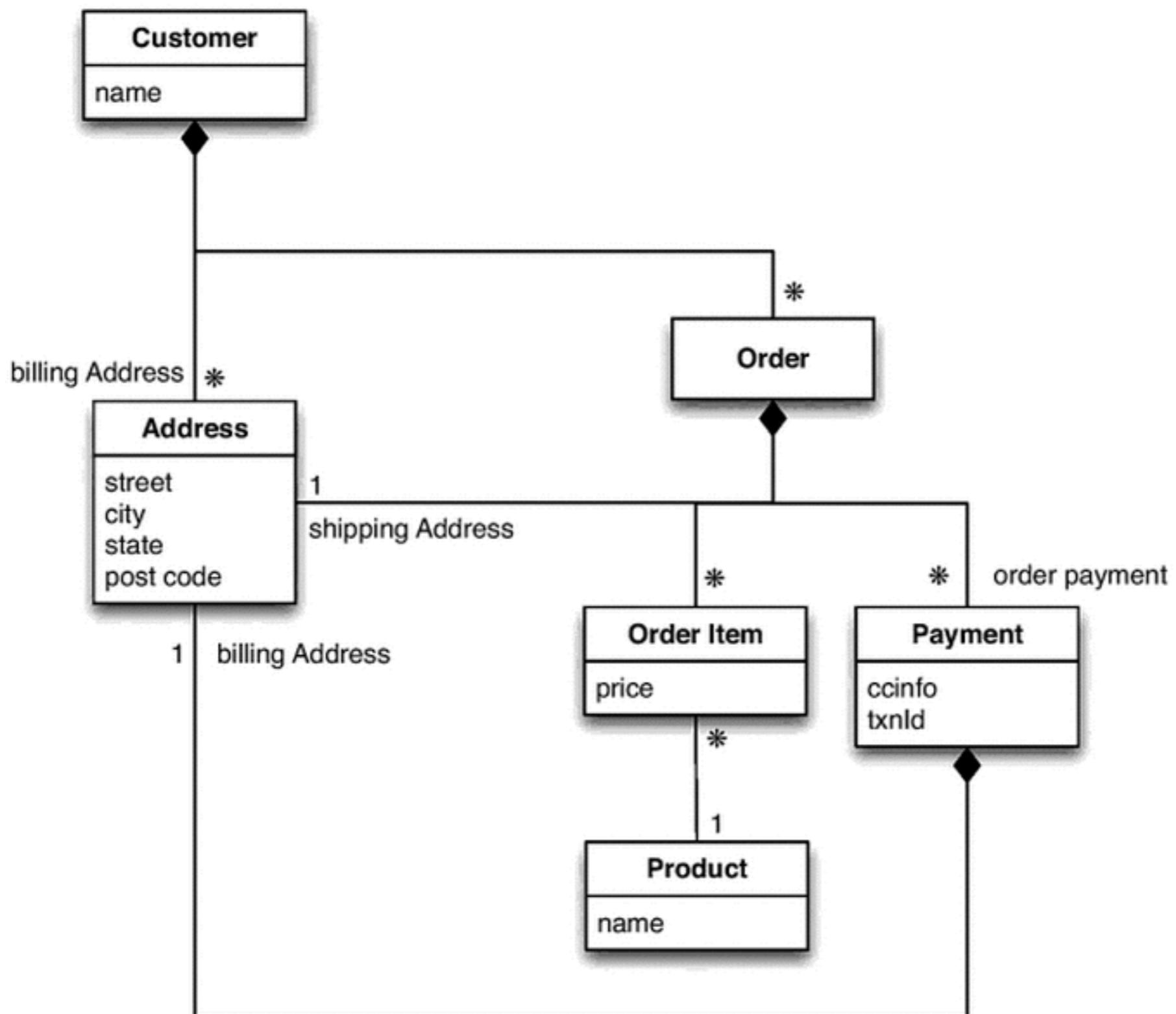
A Brief Guide to the Emerging World of Polyglot Persistence

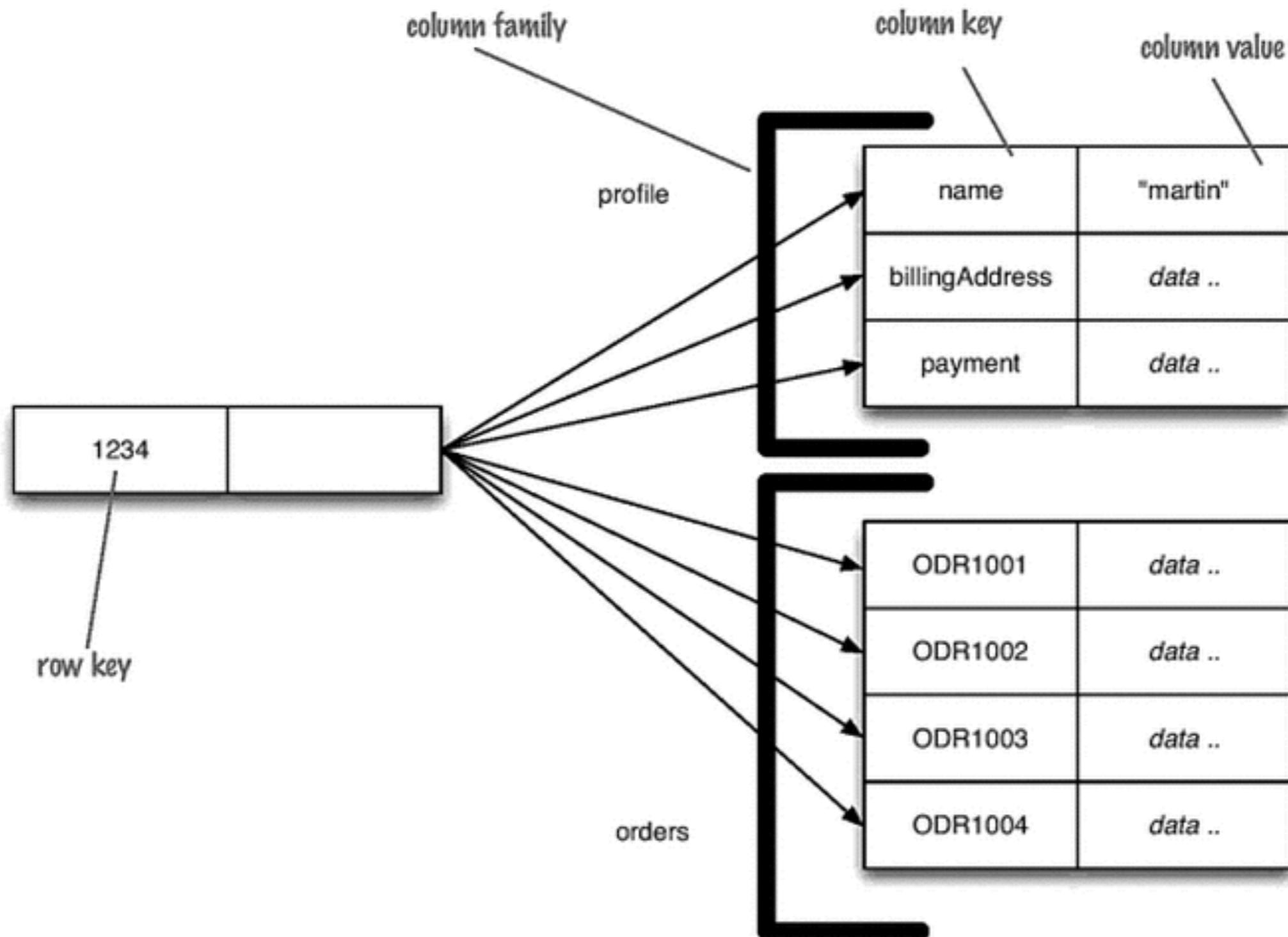
Distilled





```
{  
    orderid: '1001',  
    customerid: '232322',  
    customername: 'Ann Rogers',  
    shippingaddress: {  
        street: '1343 Broadway Street',  
        city: 'Denver',  
        zipcode: '80210',  
        state: 'CO'  
    }  
    lineitems: [  
        {id: '032193533', count: '2', cost: '$48', amount: '$96'},  
        {id: '0321601912', count: '1', cost: '$39', amount: '$39'},  
        {id: '013495054', count: '1', cost: '$51', amount: '$51'}  
    ]  
    paymentdetails : {  
        card: 'Amex',  
        ccnumber: '12345',  
        expiry: '04/2001'  
    }  
}
```





An aggregate is a collection of data that we interact with as a unit.

Aggregates form the boundaries for ACID operations with the database.



Key-value, document, and column-family databases can all be seen as forms of *aggregate-oriented database*.



Aggregates make it easier
for the database to
manage data storage over
clusters.

Aggregate-oriented databases work best

*when most data interaction is done with
the same aggregate;*

aggregate-ignorant databases are better

*when interactions use data organized in
many different formations.*