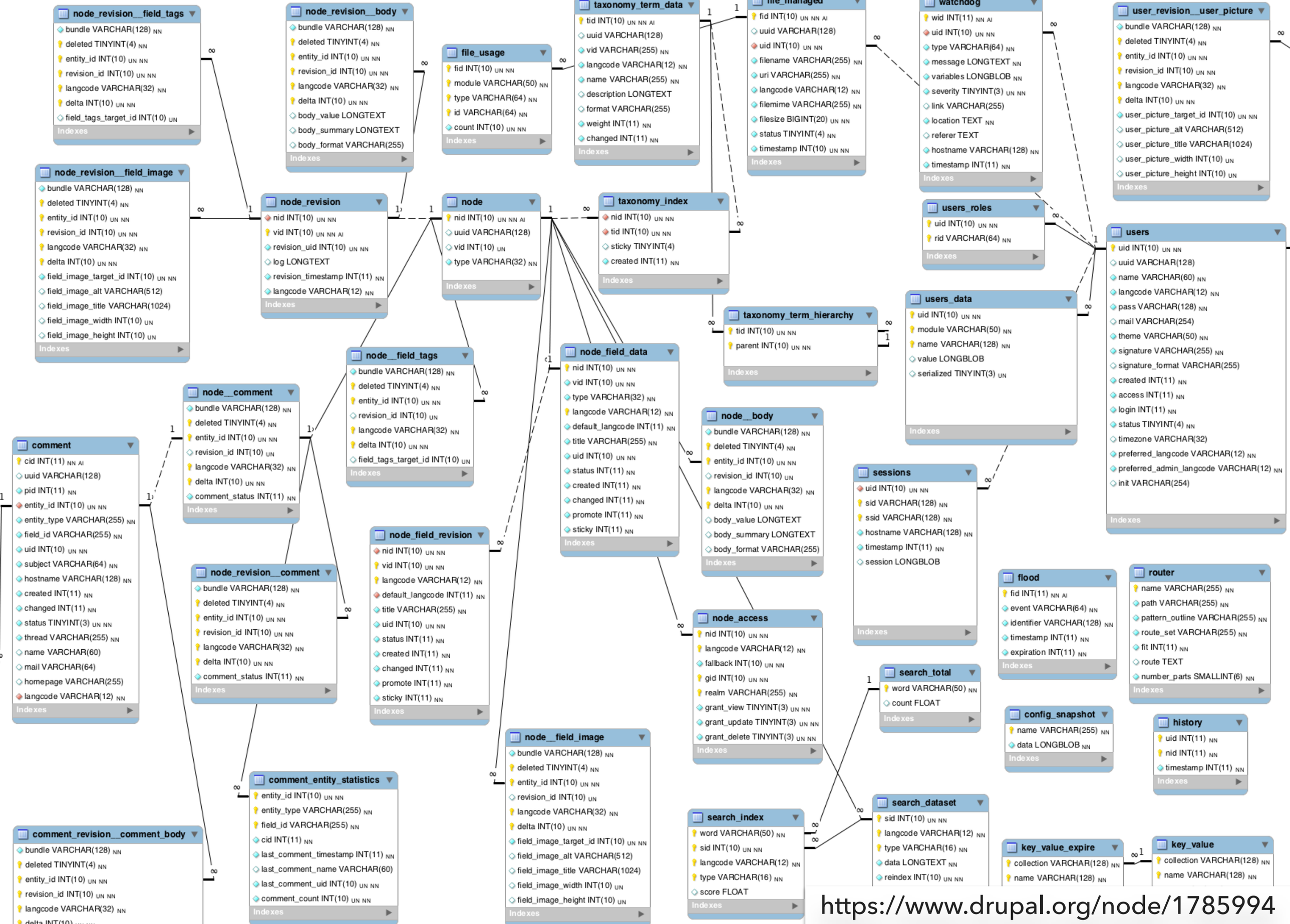
Question

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


node_revision__field_tags
bundle VARCHAR(128) NN
deleted TINYINT(4) NN
entity_id INT(10) UN NN
revision_id INT(10) UN NN
langcode VARCHAR(32) NN
delta INT(10) UN NN
field_tags_target_id INT(10) UN
Indexes
node_revision__body
bundle VARCHAR(128) NN
deleted TINYINT(4) NN
entity_id INT(10) UN NN
revision_id INT(10) UN NN
langcode VARCHAR(32) NN
delta INT(10) UN NN
body_value LONGTEXT
body_summary LONGTEXT
body_format VARCHAR(255)
Indexes
file_usage
fid INT(10) UN NN
module VARCHAR(50) NN
type VARCHAR(64) NN
id VARCHAR(64) NN
count INT(10) UN NN
Indexes
taxonomy_term_data
tid INT(10) UN NN AI
uuid VARCHAR(128)
vid VARCHAR(255) NN
langcode VARCHAR(12) NN
name VARCHAR(255) NN
description LONGTEXT
format VARCHAR(255)
weight INT(11) NN
changed INT(11) NN
Indexes
file_managed
fid INT(10) UN NN AI
uuid VARCHAR(128)
uid INT(10) UN NN
filename VARCHAR(255) NN
uri VARCHAR(255) NN
langcode VARCHAR(12) NN
filemime VARCHAR(255) NN
filesize BIGINT(20) UN NN
status TINYINT(4) NN
timestamp INT(10) UN NN
Indexes
watchdog
wid INT(11) NN AI
uid INT(10) UN NN
type VARCHAR(64) NN
message LONGTEXT NN
variables LONGBLOB NN
severity TINYINT(3) UN NN
link VARCHAR(255)
location TEXT NN
referer TEXT
hostname VARCHAR(128) NN
timestamp INT(11) NN
Indexes
user_revision__user_picture
bundle VARCHAR(128) NN
deleted TINYINT(4) NN
entity_id INT(10) UN NN
revision_id INT(10) UN NN
langcode VARCHAR(32) NN
delta INT(10) UN NN
user_picture_target_id INT(10) UN NN
user_picture_alt VARCHAR(512)
user_picture_title VARCHAR(1024)
user_picture_width INT(10) UN
user_picture_height INT(10) UN
Indexes
node_revision__field_image
bundle VARCHAR(128) NN
deleted TINYINT(4) NN
entity_id INT(10) UN NN
revision_id INT(10) UN NN
langcode VARCHAR(32) NN
delta INT(10) UN NN
field_image_target_id INT(10) UN NN
field_image_alt VARCHAR(512)
field_image_title VARCHAR(1024)
field_image_width INT(10) UN
field_image_height INT(10) UN
Indexes
node_revision
nid INT(10) UN NN
vid INT(10) UN NN AI
revision_uid INT(10) UN NN
log LONGTEXT
revision_timestamp INT(11) NN
langcode VARCHAR(12) NN
Indexes
node
nid INT(10) UN NN AI
uuid VARCHAR(128)
vid INT(10) UN
type VARCHAR(32) NN
Indexes
taxonomy_index
nid INT(10) UN NN
tid INT(10) UN NN
sticky TINYINT(4)
created INT(11) NN
Indexes
users_roles
uid INT(10) UN NN
rid VARCHAR(64) NN
Indexes
users
uid INT(10) UN NN
uuid VARCHAR(128)
name VARCHAR(60) NN
langcode VARCHAR(12) NN
pass VARCHAR(128) NN
mail VARCHAR(254)
theme VARCHAR(50) NN
signature VARCHAR(255) NN
signature_format VARCHAR(255)
created INT(11) NN
access INT(11) NN
login INT(11) NN
status TINYINT(4) NN
timezone VARCHAR(32)
preferred_langcode VARCHAR(12) NN
preferred_admin_langcode VARCHAR(12) NN
init VARCHAR(254)
Indexes
taxonomy_term_hierarchy
tid INT(10) UN NN
parent INT(10) UN NN
Indexes
users_data
uid INT(10) UN NN
module VARCHAR(50) NN
name VARCHAR(128) NN
value LONGBLOB
serialized TINYINT(3) UN
Indexes
node__field_tags
bundle VARCHAR(128) NN
deleted TINYINT(4) NN
entity_id INT(10) UN NN
revision_id INT(10) UN
langcode VARCHAR(32) NN
delta INT(10) UN NN
field_tags_target_id INT(10) UN
Indexes
node_field_data
nid INT(10) UN NN
vid INT(10) UN NN
type VARCHAR(32) NN
langcode VARCHAR(12) NN
default_langcode INT(11) NN
title VARCHAR(255) NN
uid INT(10) UN NN
status INT(11) NN
created INT(11) NN
changed INT(11) NN
promote INT(11) NN
sticky INT(11) NN
Indexes
node__body
bundle VARCHAR(128) NN
deleted TINYINT(4) NN
entity_id INT(10) UN NN
revision_id INT(10) UN
langcode VARCHAR(32) NN
delta INT(10) UN NN
body_value LONGTEXT
body_summary LONGTEXT
body_format VARCHAR(255)
Indexes
node__comment
bundle VARCHAR(128) NN
deleted TINYINT(4) NN
entity_id INT(10) UN NN
revision_id INT(10) UN
langcode VARCHAR(32) NN
delta INT(10) UN NN
comment_status INT(11) NN
Indexes
comment
cid INT(11) NN AI
uuid VARCHAR(128)
pid INT(11) NN
entity_id INT(10) UN NN
entity_type VARCHAR(255) NN
field_id VARCHAR(255) NN
uid INT(10) UN NN
subject VARCHAR(64) NN
hostname VARCHAR(128) NN
created INT(11) NN
changed INT(11) NN
status TINYINT(3) UN NN
thread VARCHAR(255) NN
name VARCHAR(60)
mail VARCHAR(64)
homepage VARCHAR(255)
langcode VARCHAR(12) NN
Indexes
node_field_revision
nid INT(10) UN NN
vid INT(10) UN NN
langcode VARCHAR(12) NN
default_langcode INT(11) NN
title VARCHAR(255) NN
uid INT(10) UN NN
status INT(11) NN
created INT(11) NN
changed INT(11) NN
promote INT(11) NN
sticky INT(11) NN
Indexes
node_revision__comment
bundle VARCHAR(128) NN
deleted TINYINT(4) NN
entity_id INT(10) UN NN
revision_id INT(10) UN NN
langcode VARCHAR(32) NN
delta INT(10) UN NN
comment_status INT(11) NN
Indexes
node_access
nid INT(10) UN NN
langcode VARCHAR(12) NN
fallback INT(10) UN NN
gid INT(10) UN NN
realm VARCHAR(255) NN
grant_view TINYINT(3) UN NN
grant_update TINYINT(3) UN NN
grant_delete TINYINT(3) UN NN
Indexes
sessions
uid INT(10) UN NN
sid VARCHAR(128) NN
ssid VARCHAR(128) NN
hostname VARCHAR(128) NN
timestamp INT(11) NN
session LONGBLOB
Indexes
search_total
word VARCHAR(50) NN
count FLOAT
Indexes
flood
fid INT(11) NN AI
event VARCHAR(64) NN
identifier VARCHAR(128) NN
timestamp INT(11) NN
expiration INT(11) NN
Indexes
router
name VARCHAR(255) NN
path VARCHAR(255) NN
pattern_outline VARCHAR(255) NN
route_set VARCHAR(255) NN
fit INT(11) NN
route TEXT
number_parts SMALLINT(6) NN
Indexes
config_snapshot
name VARCHAR(255) NN
data LONGBLOB NN
Indexes
history
uid INT(11) NN
nid INT(11) NN
timestamp INT(11) NN
Indexes
node__field_image
bundle VARCHAR(128) NN
deleted TINYINT(4) NN
entity_id INT(10) UN NN
revision_id INT(10) UN
langcode VARCHAR(32) NN
delta INT(10) UN NN
field_image_target_id INT(10) UN NN
field_image_alt VARCHAR(512)
field_image_title VARCHAR(1024)
field_image_width INT(10) UN
field_image_height INT(10) UN
Indexes
comment_entity_statistics
entity_id INT(10) UN NN
entity_type VARCHAR(255) NN
field_id VARCHAR(255) NN
cid INT(11) NN
last_comment_timestamp INT(11) NN
last_comment_name VARCHAR(60)
last_comment_uid INT(10) UN NN
comment_count INT(10) UN NN
Indexes
search_index
word VARCHAR(50) NN
sid INT(10) UN NN
langcode VARCHAR(12) NN
type VARCHAR(16) NN
score FLOAT
Indexes
search_dataset
sid INT(10) UN NN
langcode VARCHAR(12) NN
type VARCHAR(16) NN
data LONGTEXT NN
reindex INT(10) UN NN
key_value_expire
collection VARCHAR(128) NN
name VARCHAR(128) NN
key_value
collection VARCHAR(128) NN
name VARCHAR(128) NN
comment_revision__comment_body
bundle VARCHAR(128) NN
deleted TINYINT(4) NN
entity_id INT(10) UN NN
revision_id INT(10) UN NN
langcode VARCHAR(32) NN
https://www.drupal.org/node/1785994

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

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

Aggregates

Modeling Boundaries

Transaction
Distribution
Concurrency

Aggregate

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

Purchase order

ABC432

Line Item

Purchase order

XYZ124

Line Item

5 Guidelines for Aggregate Design

Guideline #1

Design small, cohesive aggregates

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

Guideline #2

Identify 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

What Needs to Change…

To separate RepairPurchaseOrders, Claims and Contracts into separate aggregates?

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.

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 a time per active Contract.*

–*Sum of costs of all Claims for a Contract must <u>never</u> 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

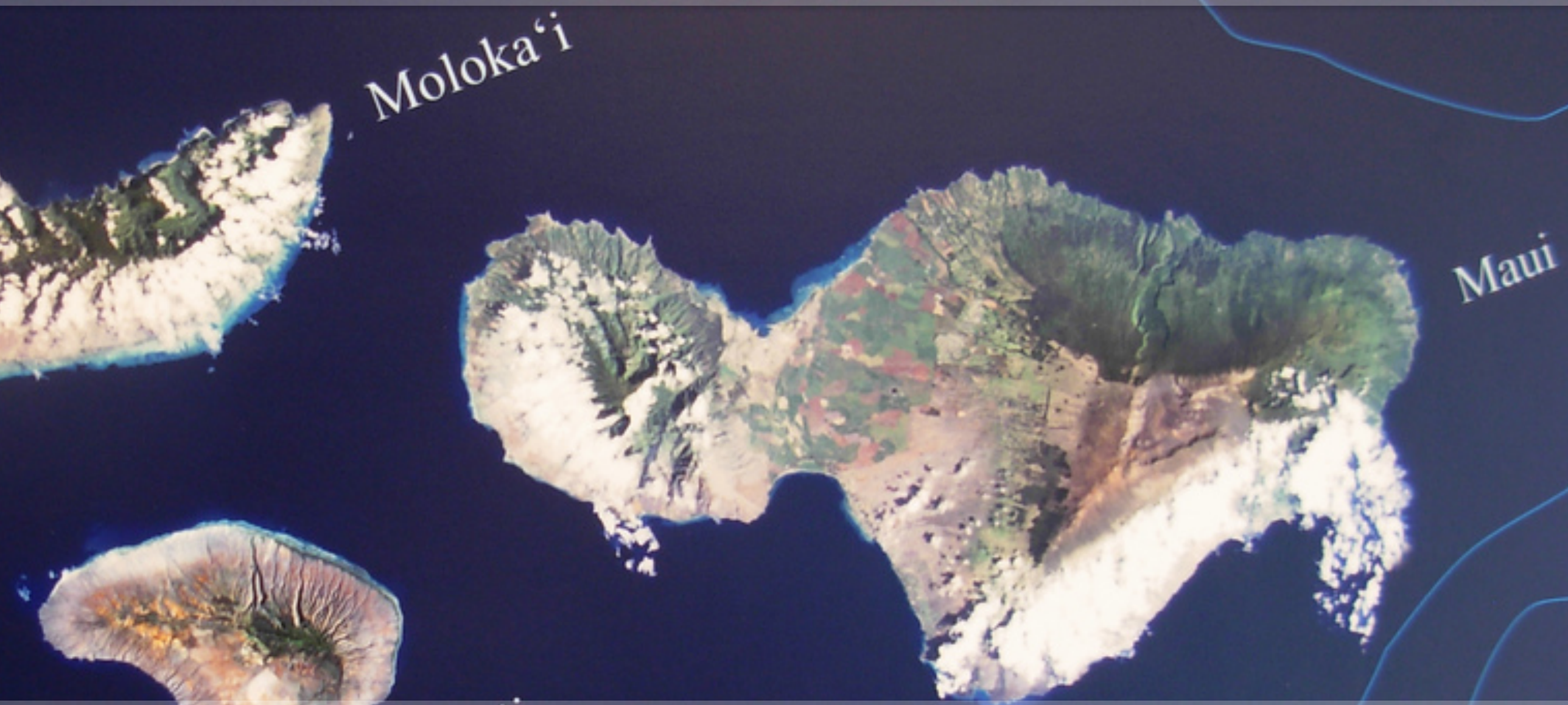
Use eventual consistency across aggregate boundaries

Aggregates are “eventually consistent” with each other.

- asynchronous updates propagate through system

Aggregates are always internally consistent.

Invariants apply at every transaction commit



Aggregates are consistency islands

EventStorming

EventStorm how you think the Limit of Liability scenario <u>should</u> work for Repair warranties. Assume Contracts, Claims and Repair POs are different aggregates.

Include any domain events necessary, including

- Contract is terminated
- Associated open Claims are closed
- Repair POs are finalized
- Customer is notified
- Finance is notified of relevant financials

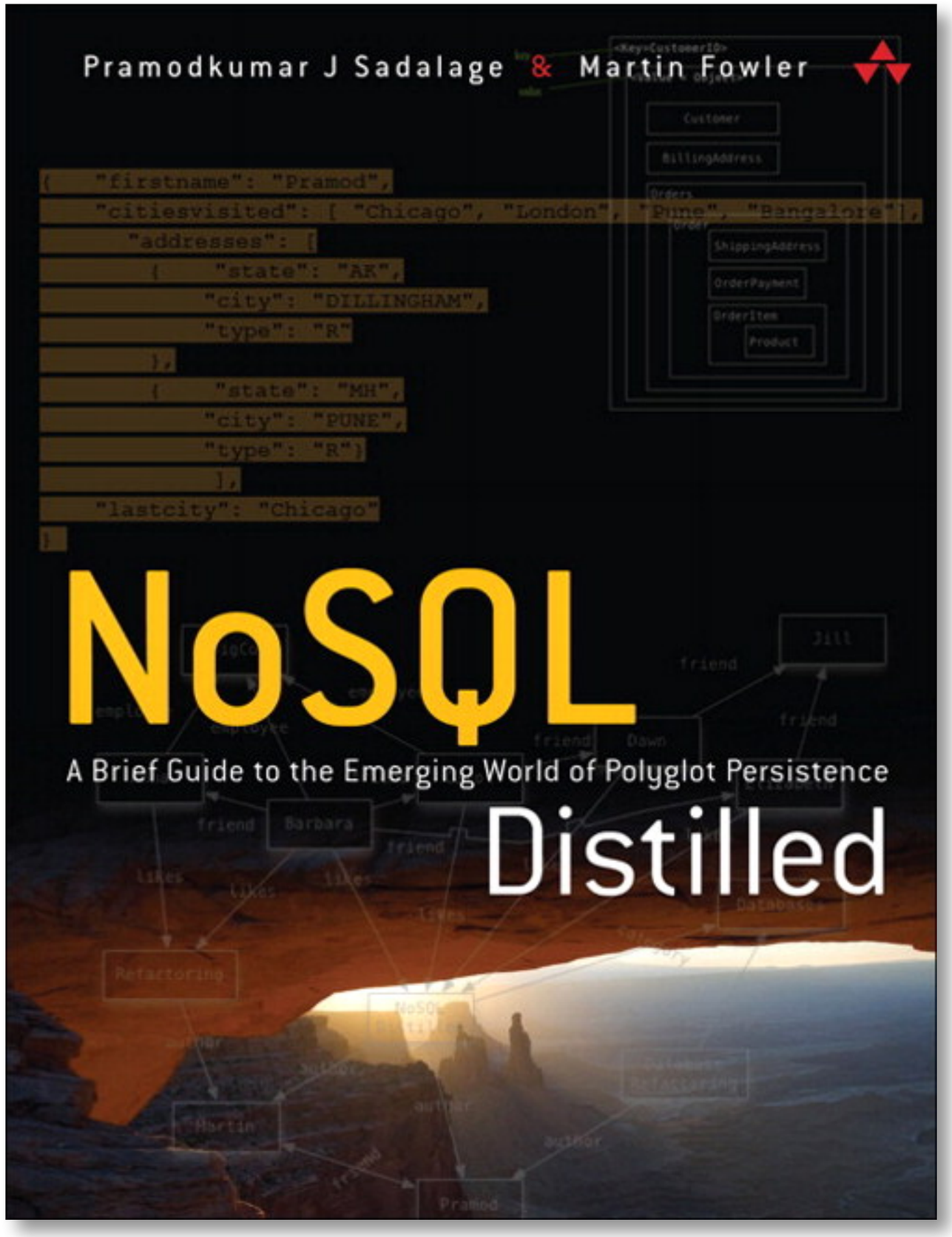
Aggregate Persistance

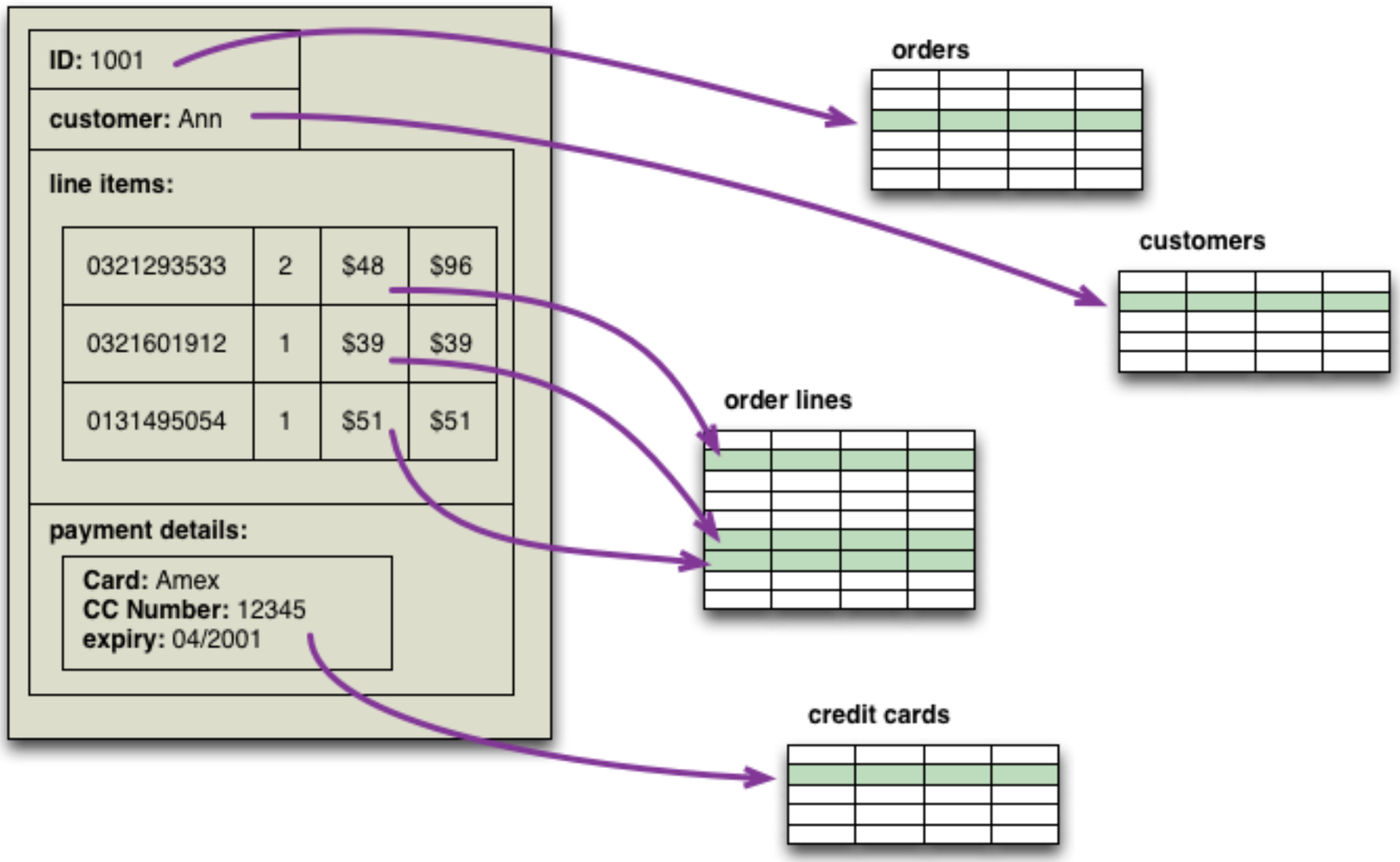
Aggregates are the basic element of transfer of data storage

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

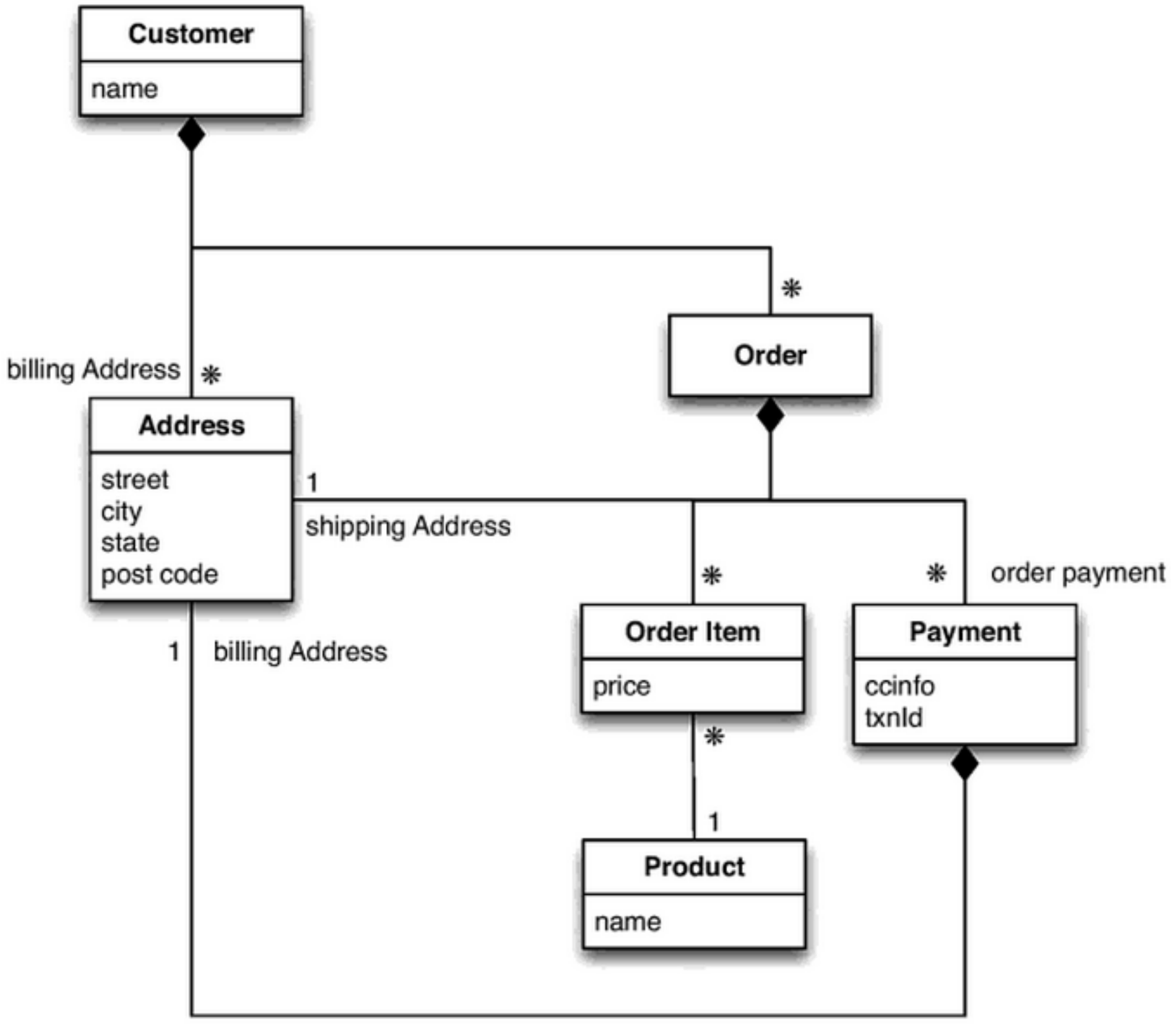
Repositories are services that provide access to stored aggregates in terms of the ubiquitous language

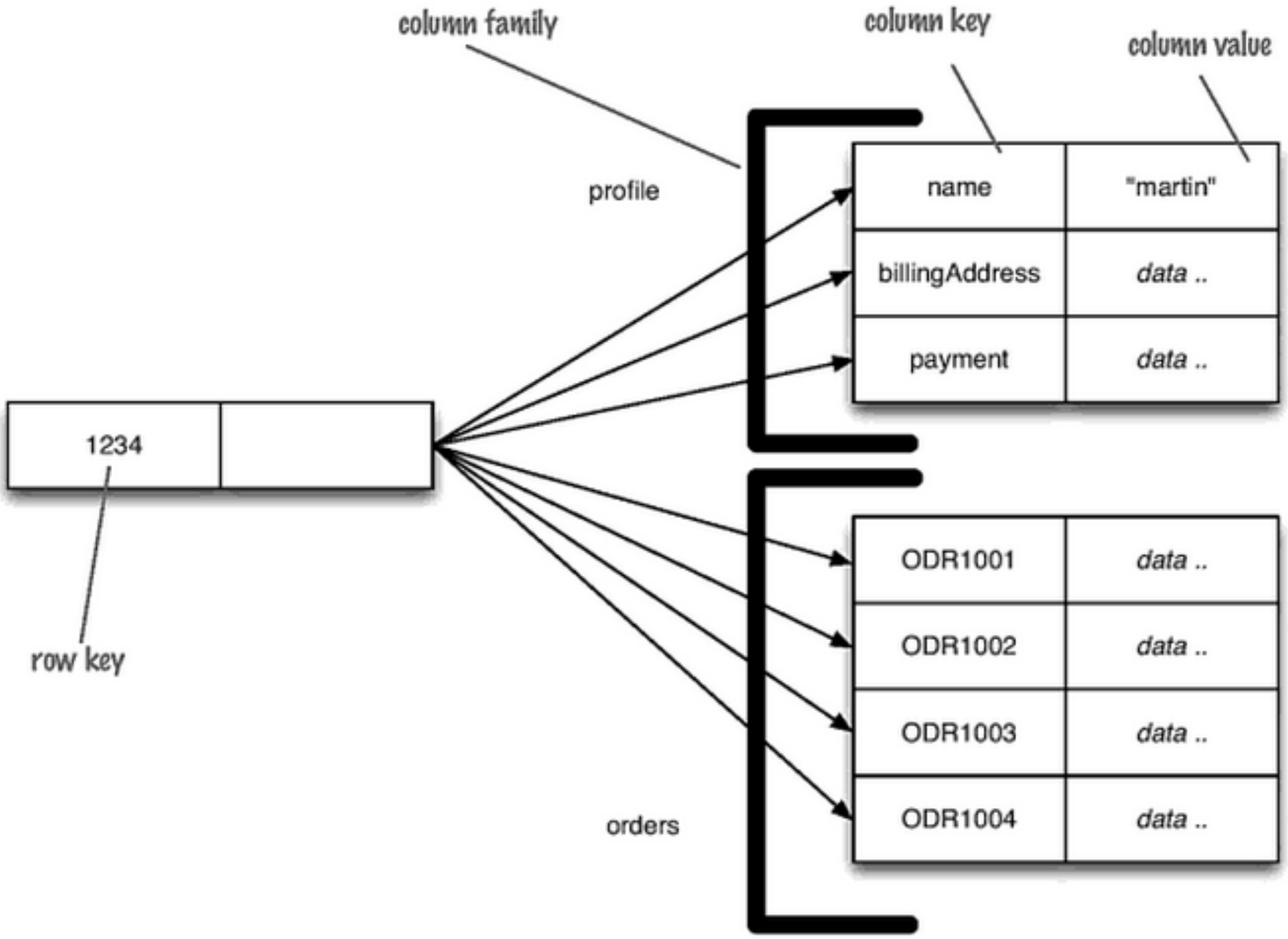
Encapsulate actual storage and query technology inside repositories


Pramodkumar J Sadalage & Martin Fowler
{ "firstname": "Pramod",
"citiesvisited": ["Chicago", "London", "Pune", "Bangalore"],
"addresses": [
{ "state": "AR",
"city": "DILLINGHAM",
"type": "R"
},
{ "state": "MH",
"city": "PUNE",
"type": "R"}
],
"lastcity": "Chicago"
}
NoSQL
A Brief Guide to the Emerging World of Polyglot Persistence
Distilled


ID: 1001
customer: Ann
line items:
0321293533
2
$48
$96
0321601912
1
$39
$39
0131495054
1
$51
$51
payment details:
Card: Amex
CC Number: 12345
expiry: 04/2001
orders
customers
order lines
credit cards

```
{
    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'
    }
}
```

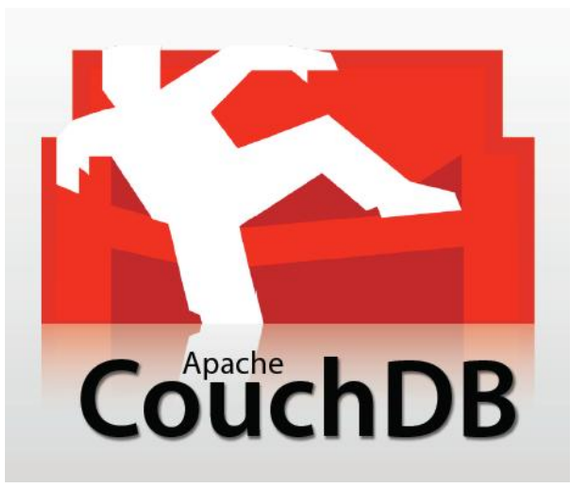

Customer
name
billing Address
*
*
Order
Address
street
city
state
post code
1
shipping Address
1
billing Address
*
*
order payment
Order Item
price
Payment
ccinfo
txnId
*
1
Product
name


column family
column key
column value
profile
name
"martin"
billingAddress
data ..
payment
data ..
1234
row key
ODR1001
data ..
ODR1002
data ..
ODR1003
data ..
ODR1004
data ..
orders

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.