Database Management Systems

- NoSQL

Doug Shook

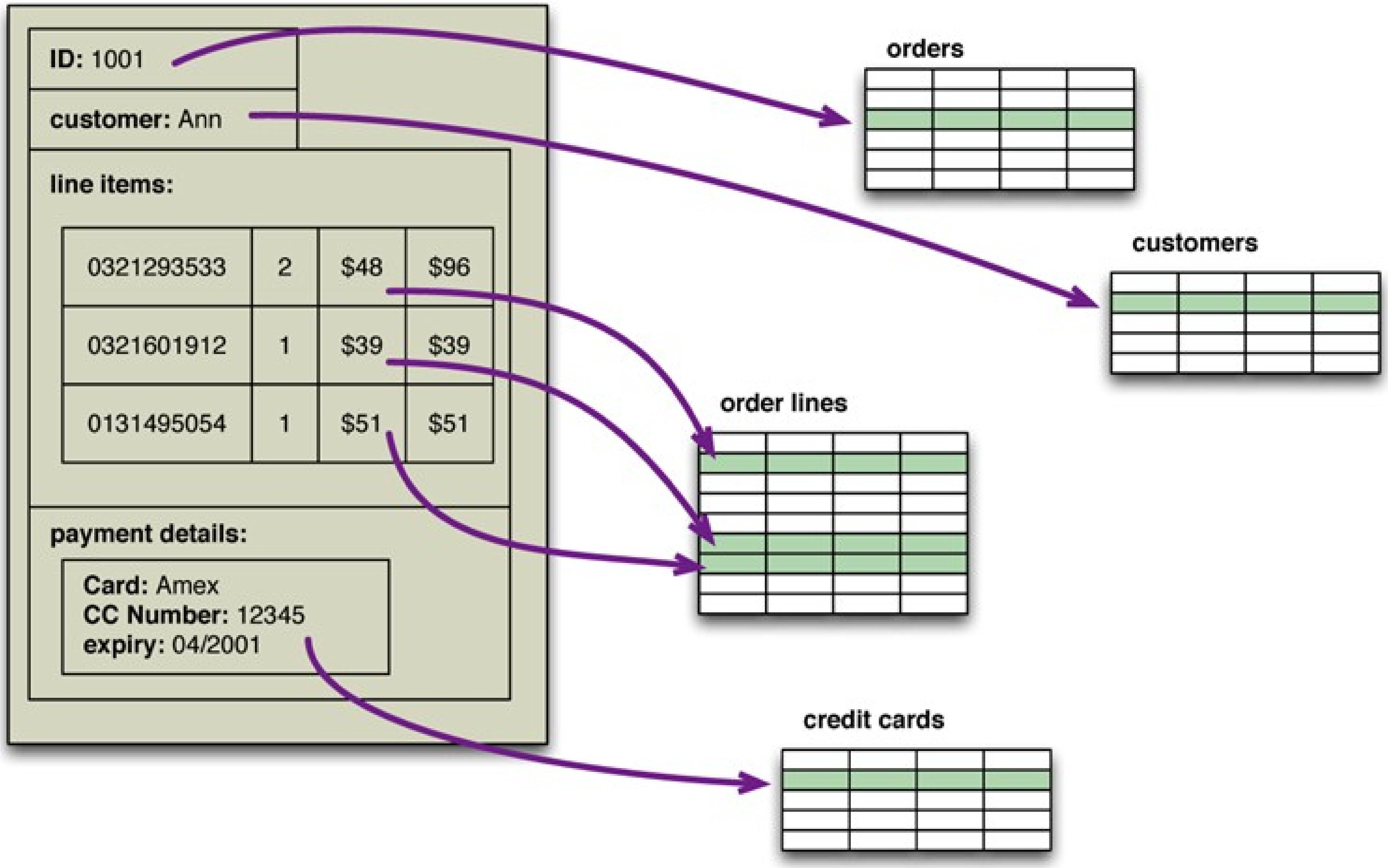
Why NoSQL?

- Better question: why relational databases?

- Sadalage, Fowler, “NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence” 2012

Problems with the Relational Model

- Impedance Mismatch



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

Application vs. Integration

- What are you using the database for?
  - How does this effect...
    - Complexity?
    - Performance?
    - Development?

The effect of scale

- Consider how the internet looks now compared to 15 years ago....
  - How did we come to meet the increase in demand?

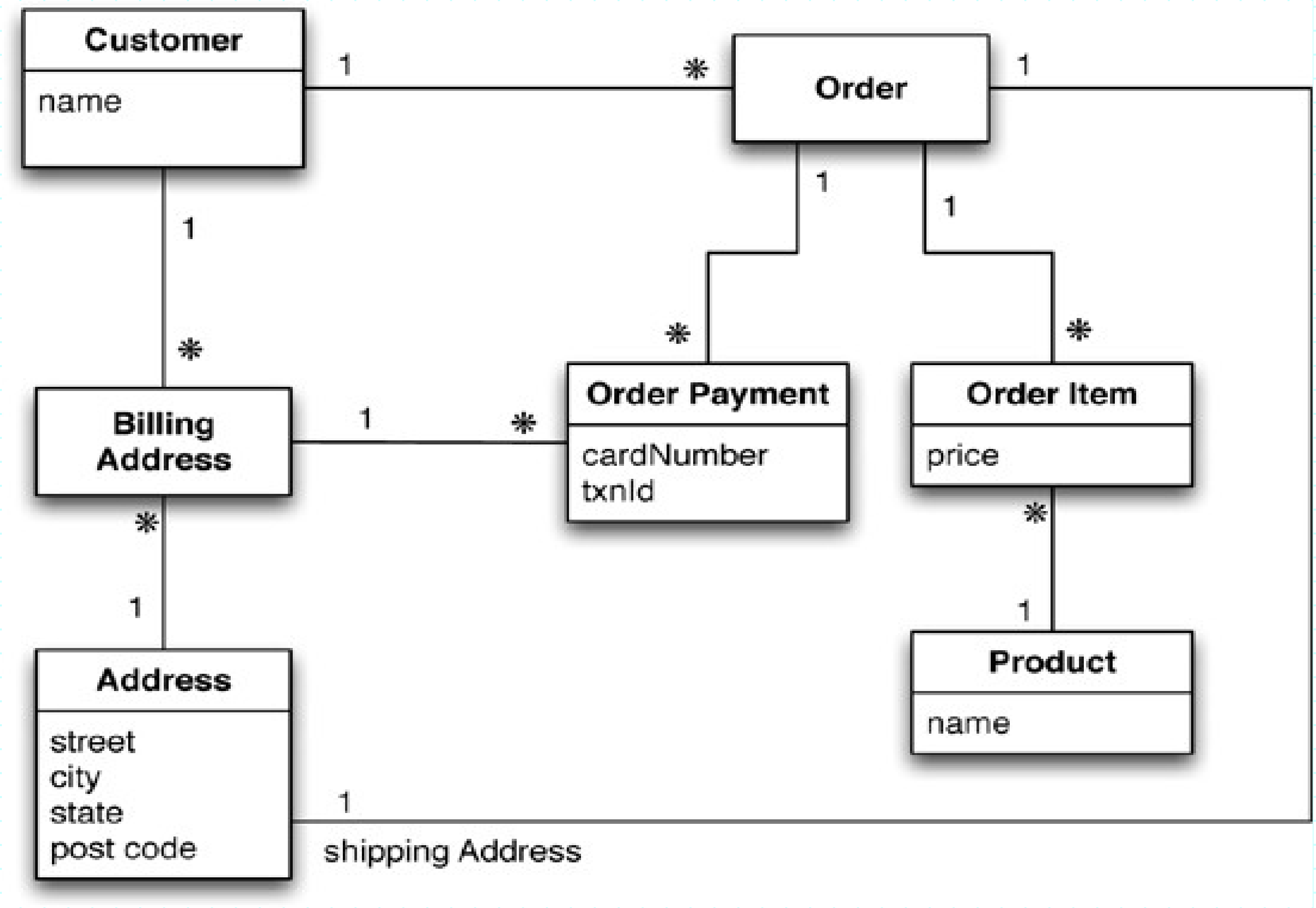
Here comes NoSQL

- What is it?
- How is it different?
- Why should we be using it?

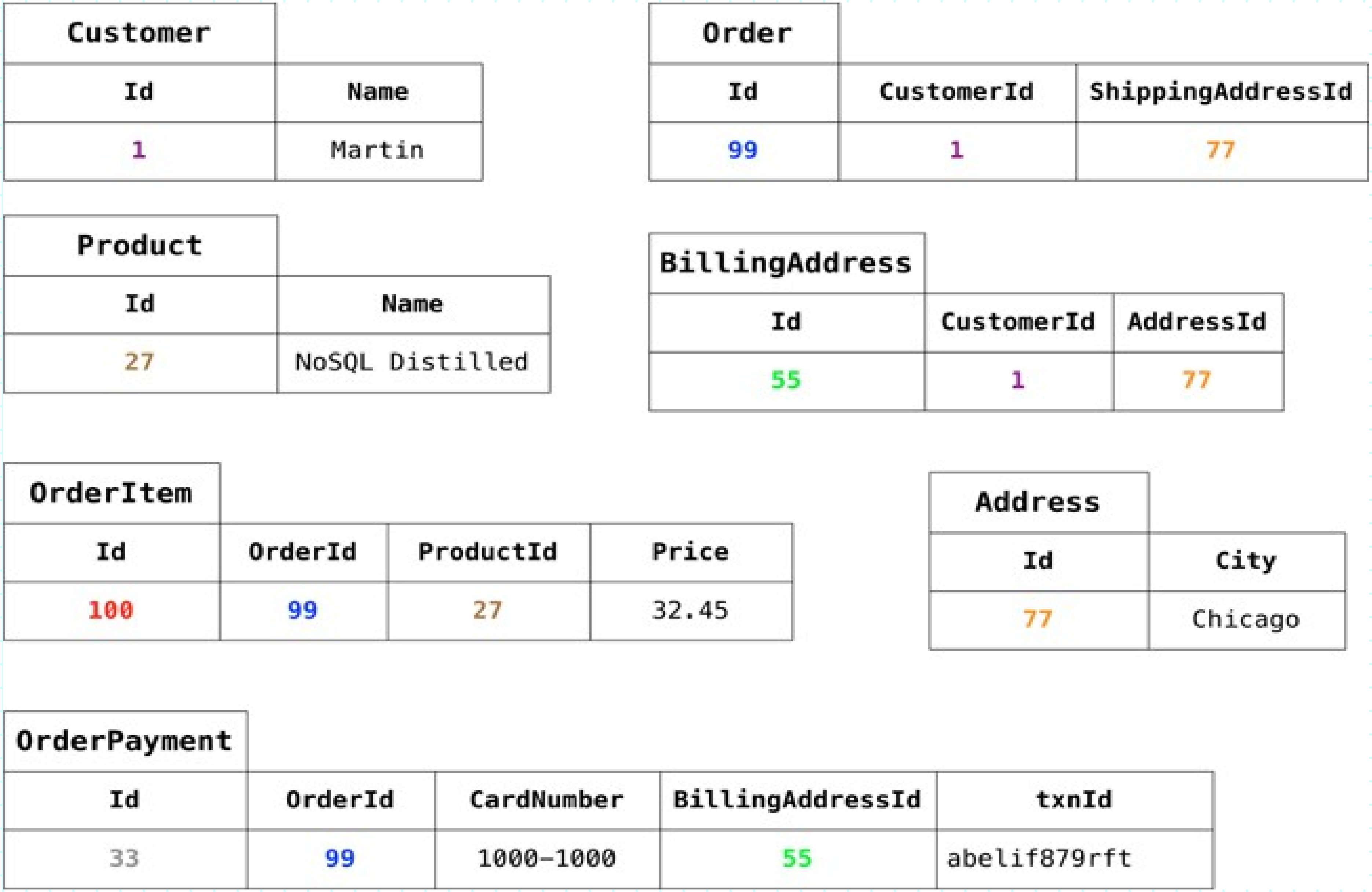
Aggregates

- How does a relational database store records?
  - What are the limitations of this?

A relational data model



Relational data



Customer

Id	Name
1	Martin

Order

Id	CustomerId	ShippingAddressId
99	1	77

Product

Id	Name
27	NoSQL Distilled

BillingAddress

Id	CustomerId	AddressId
55	1	77

OrderItem

Id	OrderId	ProductId	Price
100	99	27	32.45

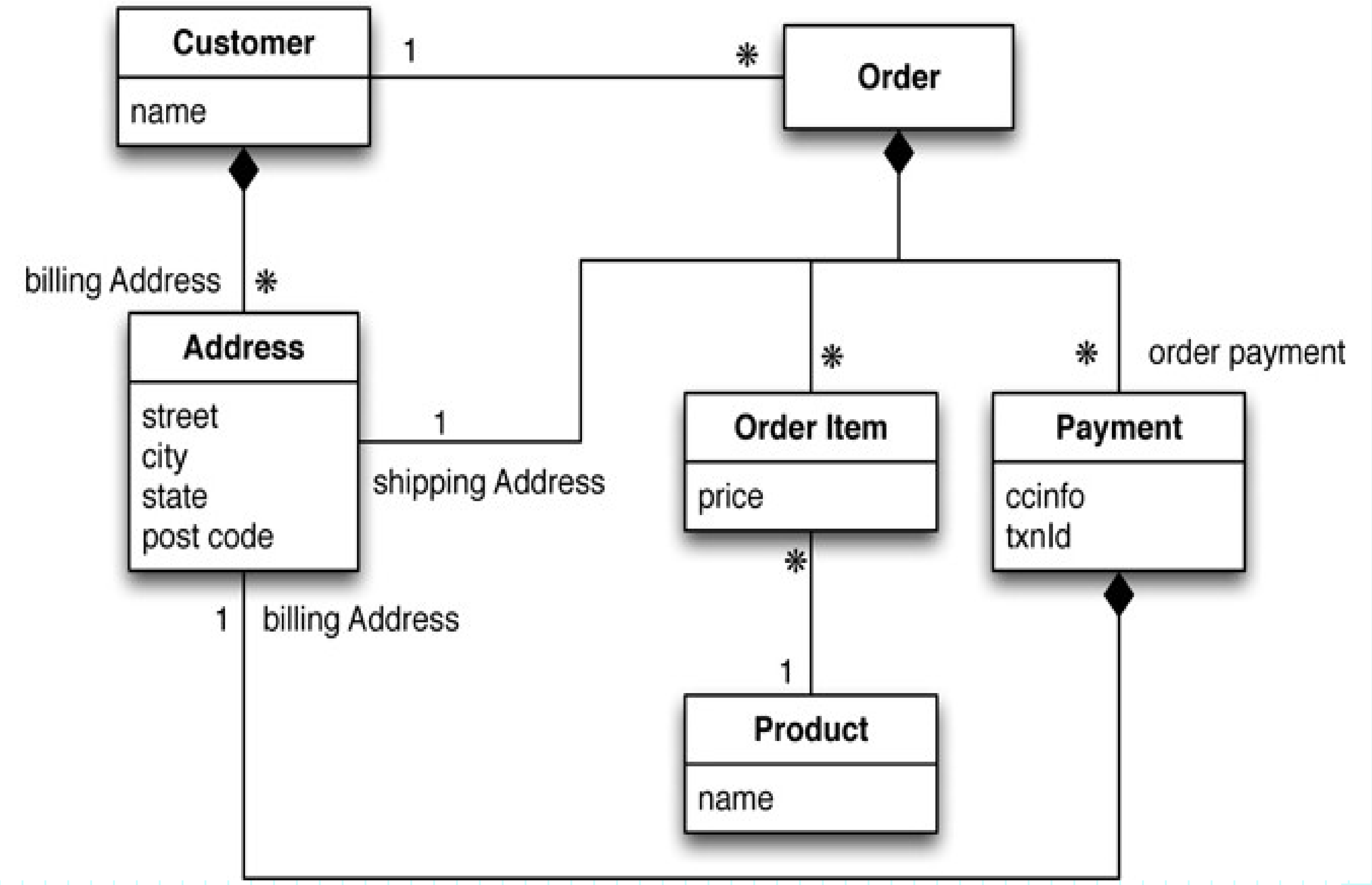
Address

Id	City
77	Chicago

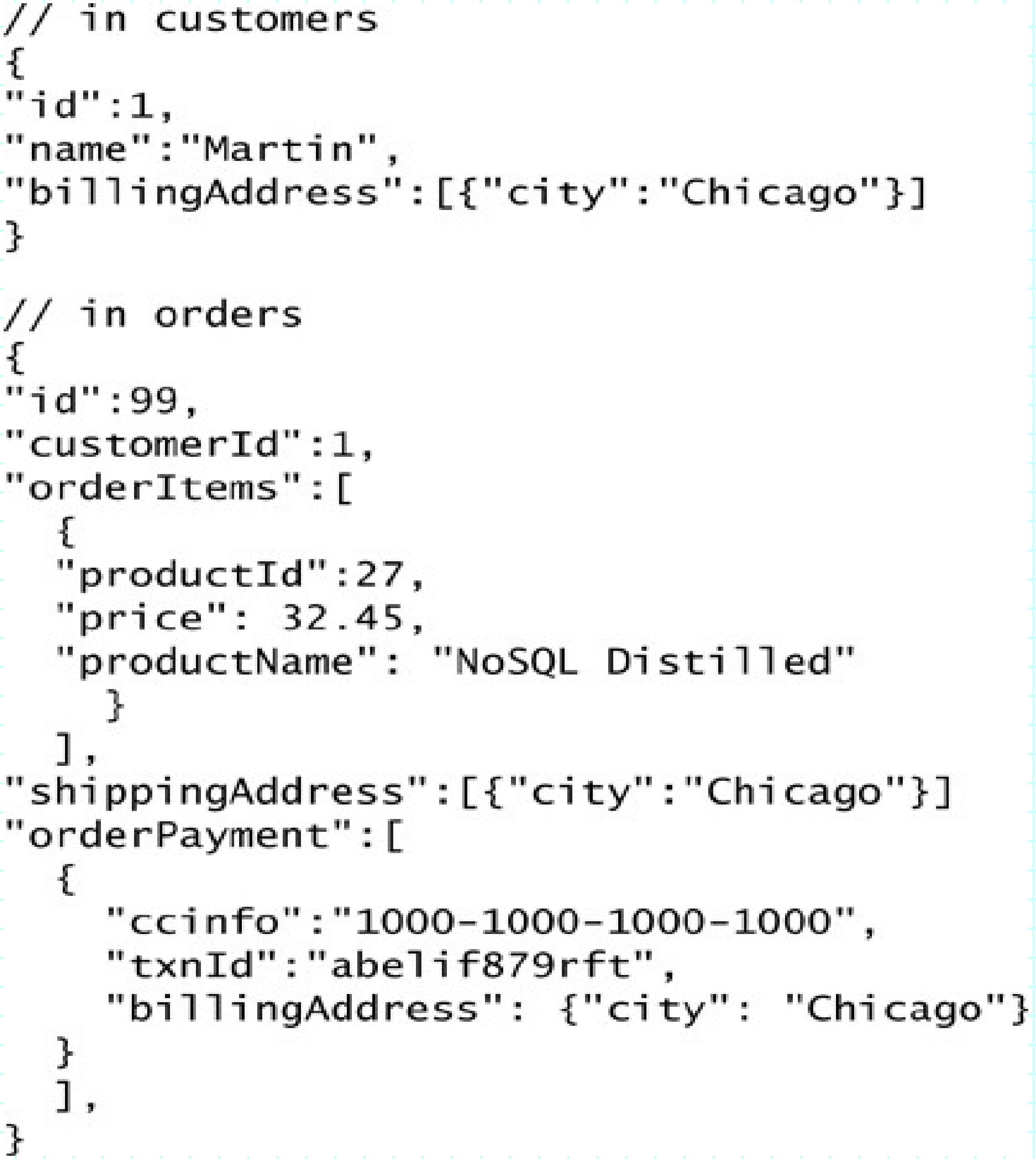
OrderPayment

Id	OrderId	CardNumber	BillingAddressId	txnId
33	99	1000-1000	55	abelif879rft

An aggregate data model



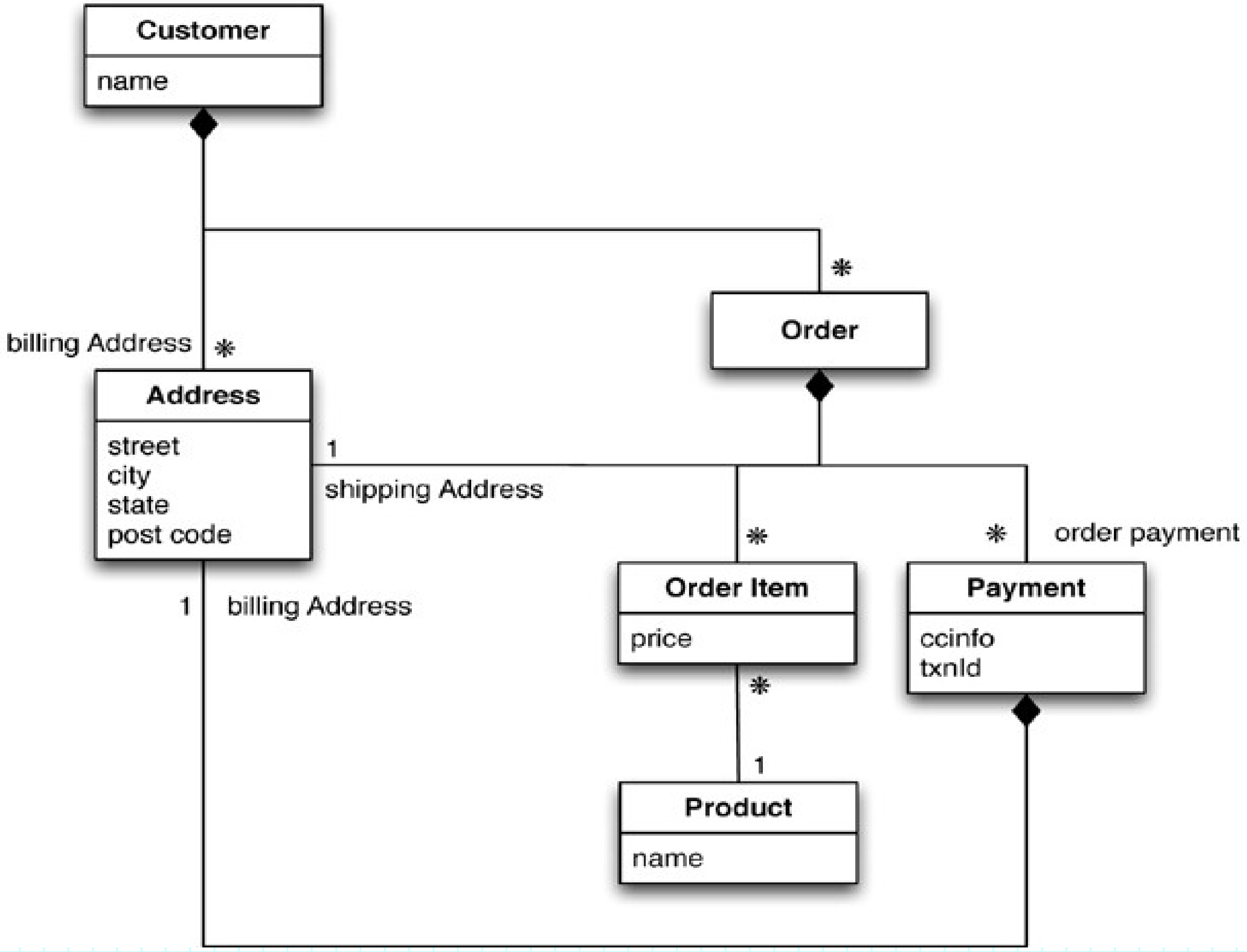
Aggregate Data



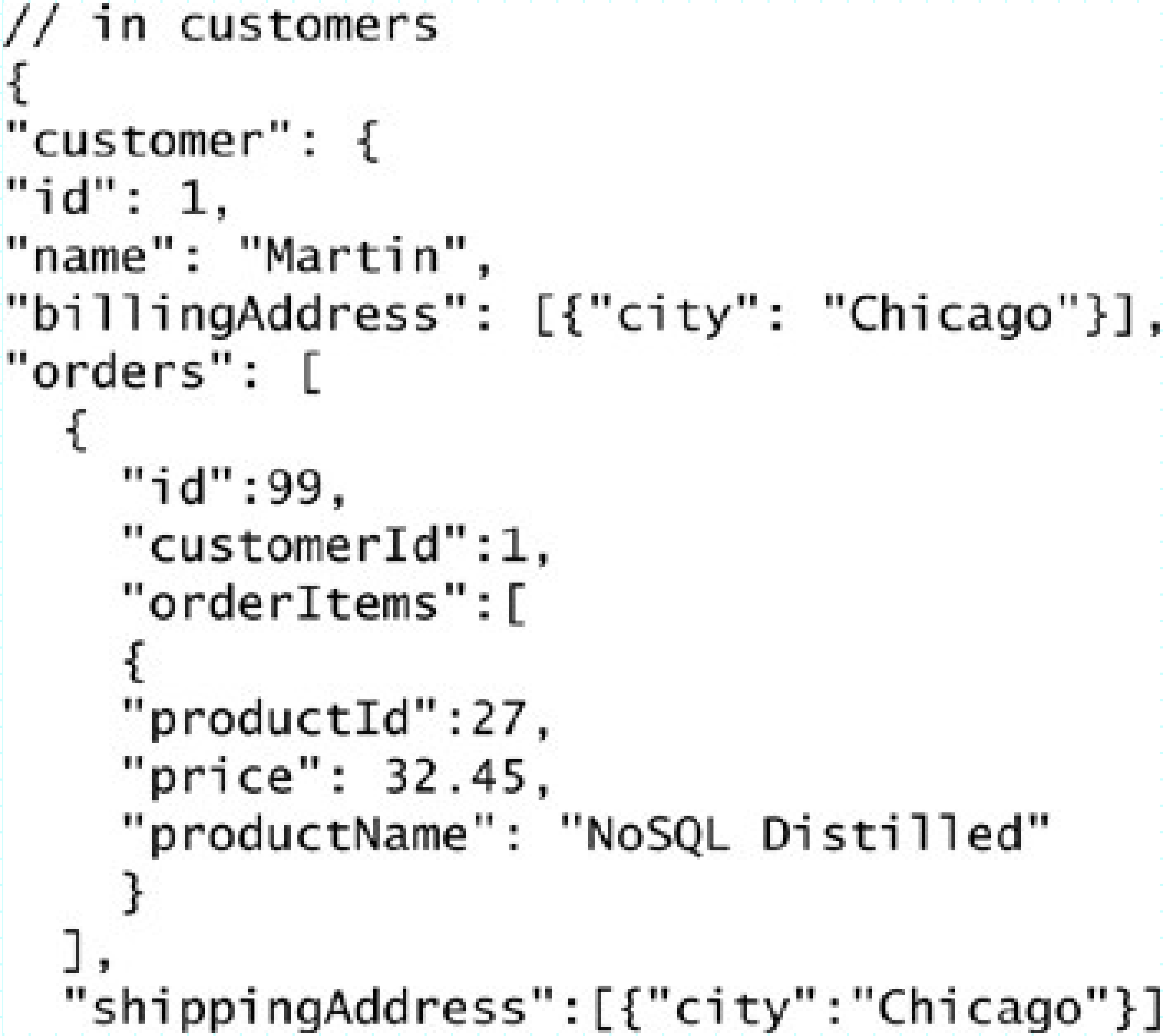
```
// in customers
{
"id":1,
"name":"Martin",
"billingAddress":[{"city":"Chicago"}]
}

// in orders
{
"id":99,
"customerId":1,
"orderItems":[
  {
  "productId":27,
  "price": 32.45,
  "productName": "NoSQL Distilled"
    }
  ],
"shippingAddress":[{"city":"Chicago"}]
"orderPayment":[
  {
    "ccinfo":"1000-1000-1000-1000",
    "txnId":"abelif879rft",
    "billingAddress": {"city": "Chicago"}
  }
  ],
}
```

A different aggregate data model



A different set of data



```
// in customers
{
"customer": {
"id": 1,
"name": "Martin",
"billingAddress": [{"city": "Chicago"}],
"orders": [
  {
    "id":99,
    "customerId":1,
    "orderItems":[
    {
    "productId":27,
    "price": 32.45,
    "productName": "NoSQL Distilled"
    }
  ],
  "shippingAddress":[{"city":"Chicago"}]
```

Consequences

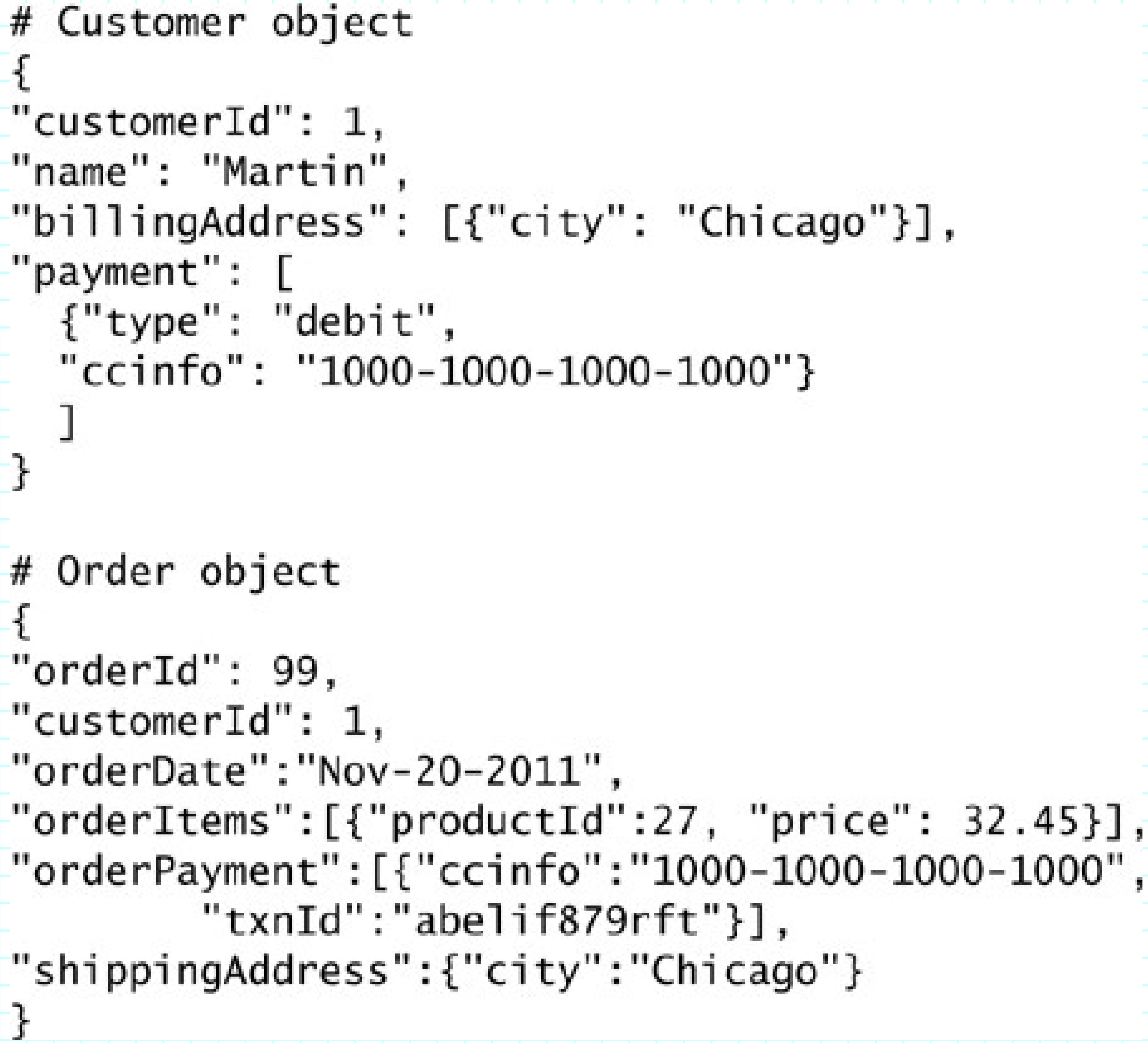
- How are these aggregates represented in relational databases?
- What benefits are gained from using these aggregates in this way?
- ACID Transactions

Schemaless Databases

- Under this model we do not require a schema
  - How is this possible?
    - Advantages?
    - Disadvantages?

- Are they actually schemaless?

Modeling for Data Access



```
# Customer object
{
"customerId": 1,
"name": "Martin",
"billingAddress": [{"city": "Chicago"}],
"payment": [
  {"type": "debit",
  "ccinfo": "1000-1000-1000-1000"}
  ]
}

# Order object
{
"orderId": 99,
"customerId": 1,
"orderDate":"Nov-20-2011",
"orderItems":[{"productId":27, "price": 32.45}],
"orderPayment":[{"ccinfo":"1000-1000-1000-1000",
        "txnId":"abelif879rft"}],
"shippingAddress":{"city":"Chicago"}
}
```

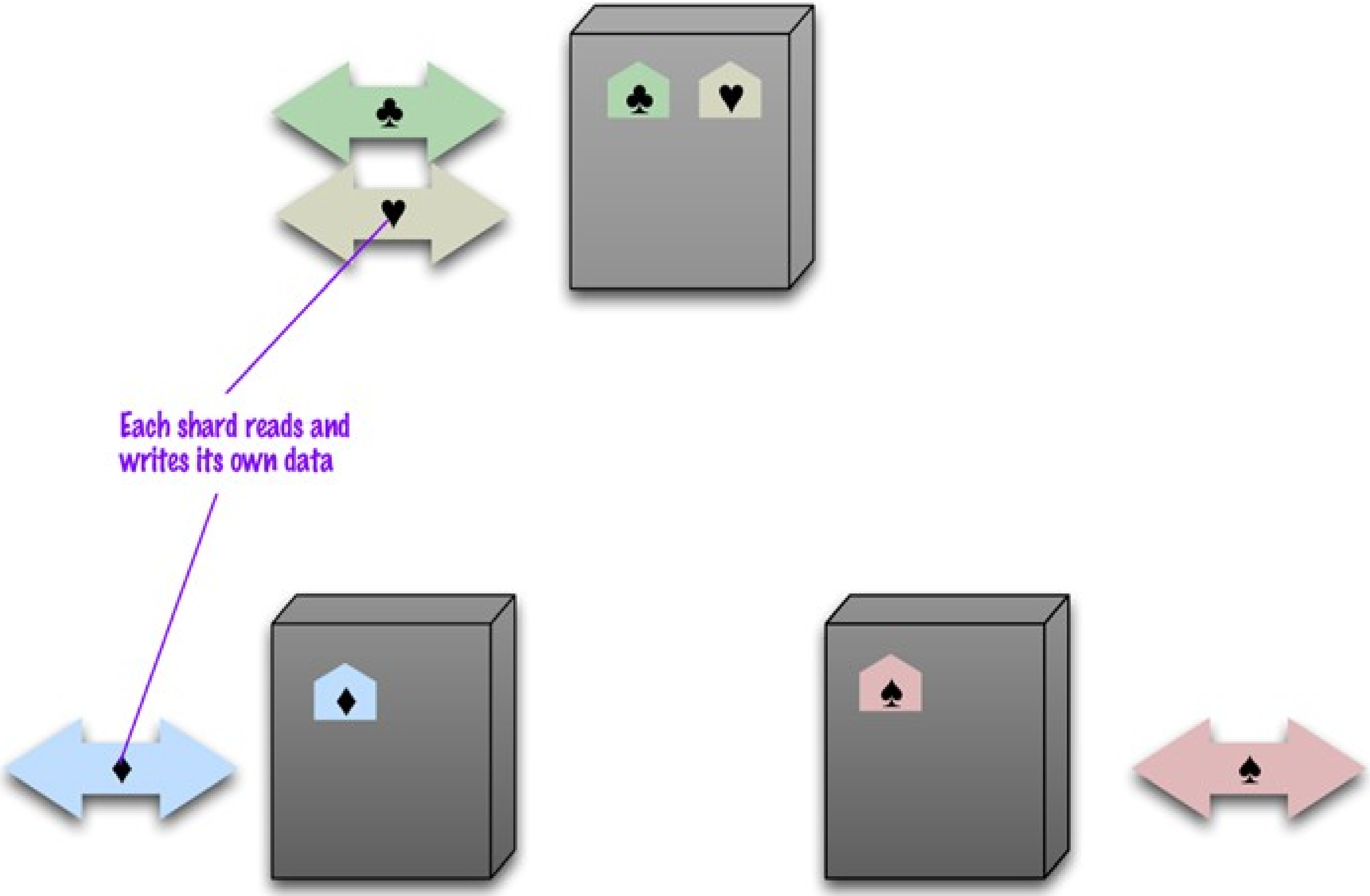
Distribution

- One of the major benefits of NoSQL (why?)
  - Comes with one major drawback...

Single Server

- Is this really distribution?
- Why might you want to do it this way?

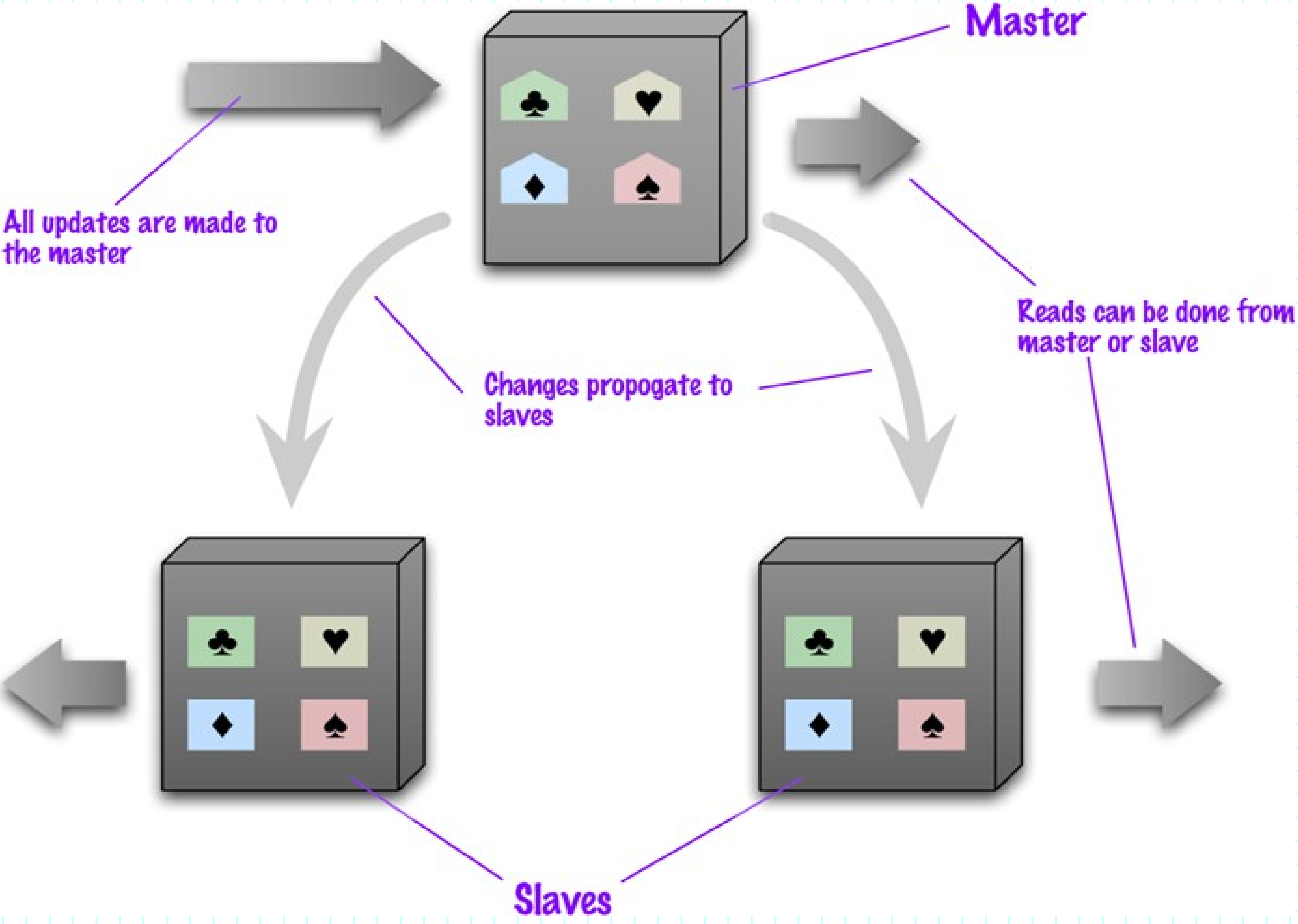
Sharding



Sharding

- What is the ideal case for # of users / servers?
- How do we decide how to split up the data?
  - What data will commonly be accessed together?
- What can we use to help us perform these tasks?

Master-Slave Replication

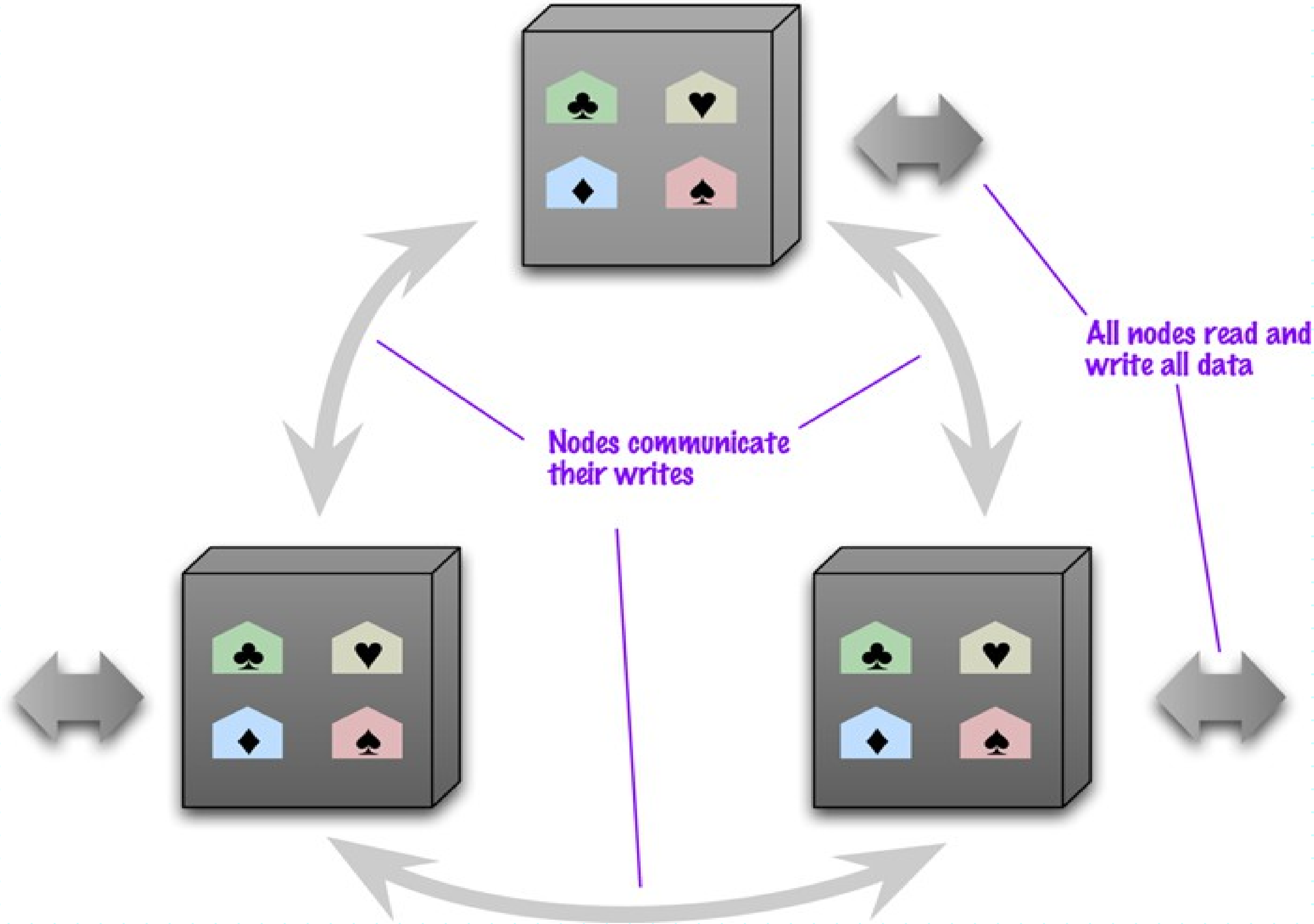


Master
All updates are made to the master
Changes propogate to slaves
Reads can be done from master or slave
Slaves
♣
♥
♦
♠

Master-Slave Replication

- How does this affect performance of reads? Writes?
- How does this affect data resilience?
- What about consistency?

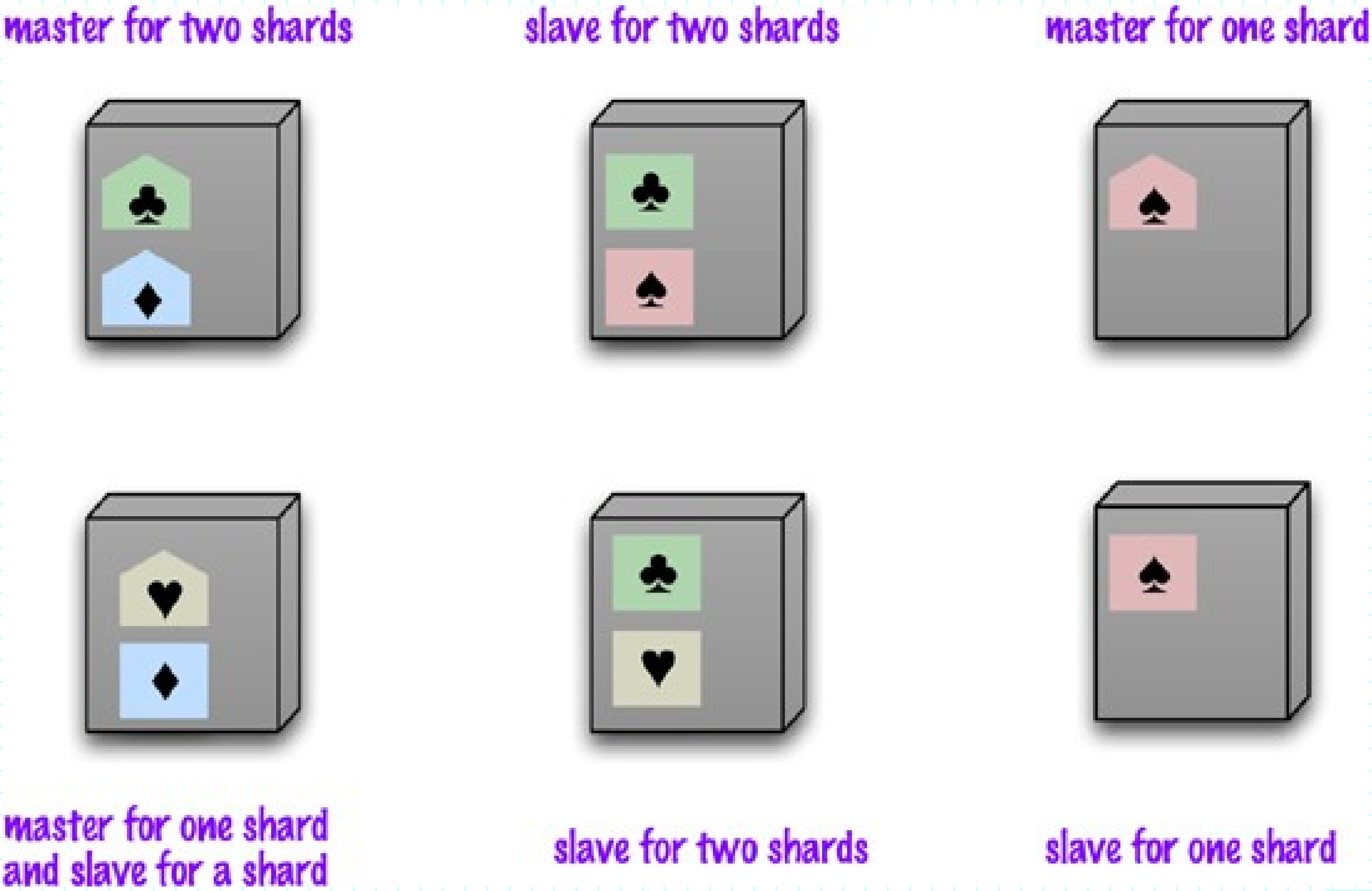
Peer-to-Peer Replication


All nodes read and write all data
Nodes communicate their writes

Peer-to-Peer Replication

- How does this affect performance of reads? Writes?
- How does this affect data resilience?
- What about consistency?

Combinations

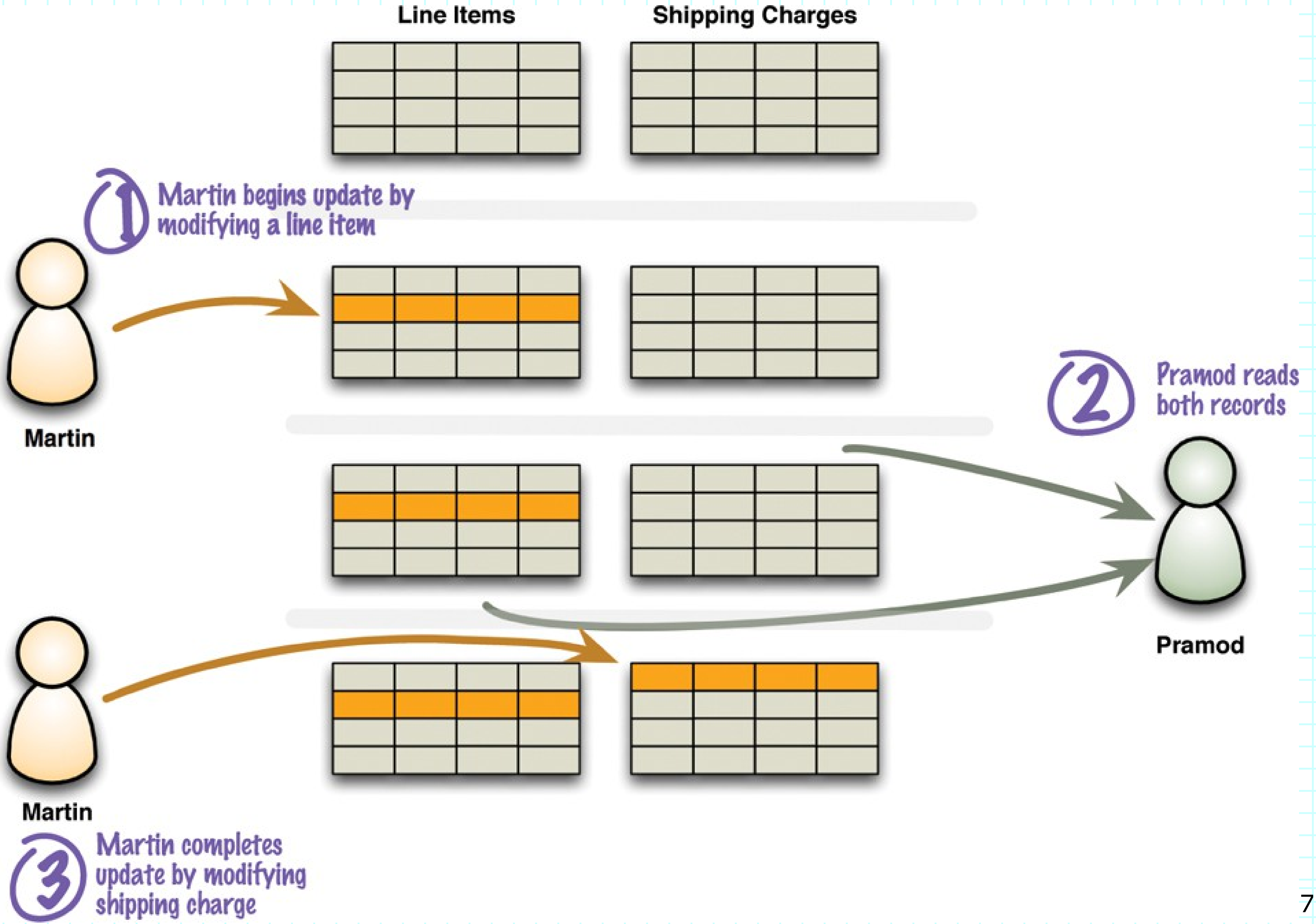


Update Consistency

- Imagine two people try to update the same piece of data at the same time
  - What happens?
  - What do we want to happen?

- Pessimistic vs. Optimistic

Read Consistency


Line Items
Shipping Charges
1 Martin begins update by modifying a line item
Martin
2 Pramod reads both records
Pramod
Martin
3 Martin completes update by modifying shipping charge

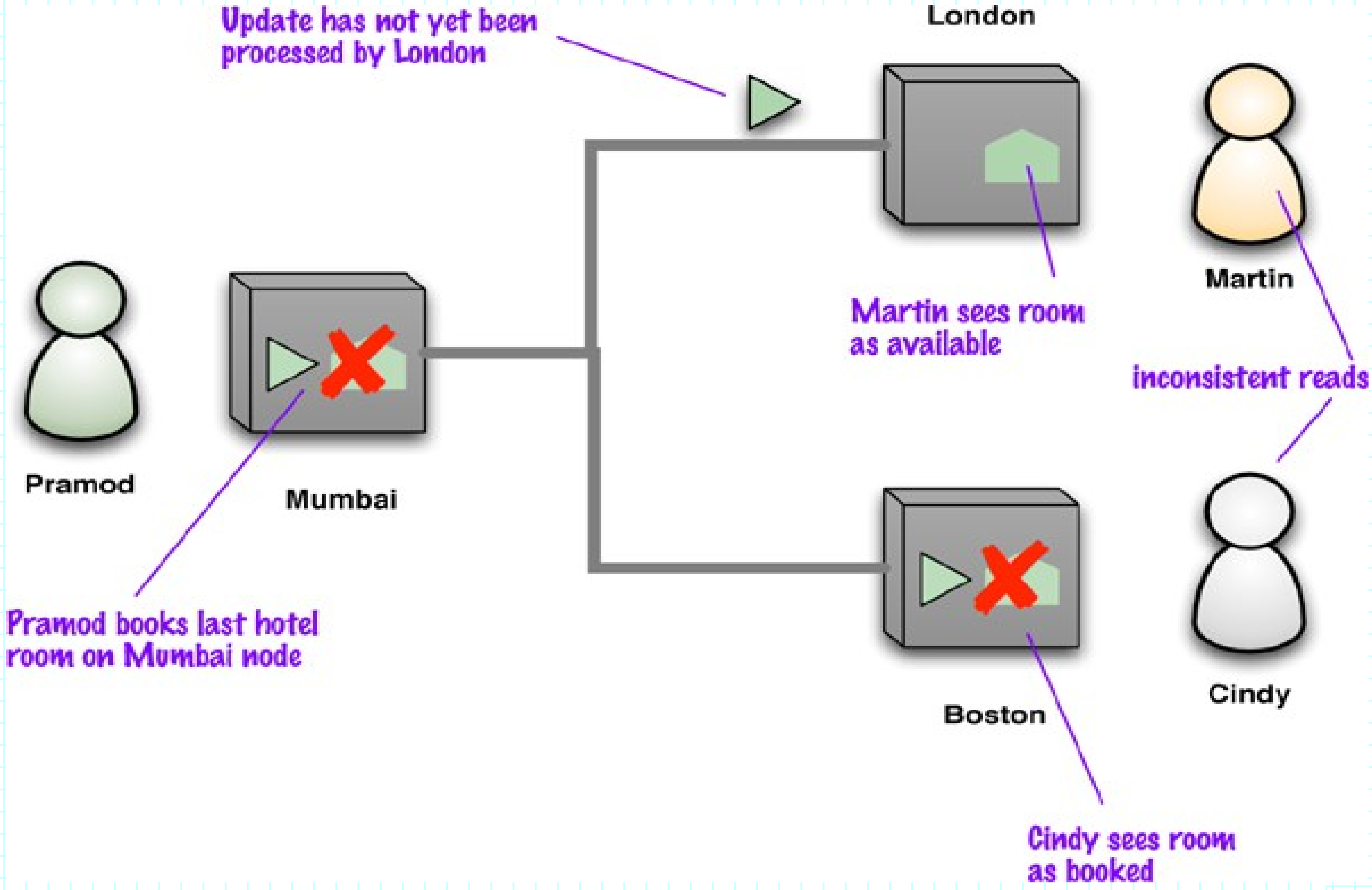
Read Consistency

- Logical consistency
  - How do relational DBs handle this?

- How does NoSQL handle it?
  - Inconsistency window

- How does replication complicate this kind of consistency?

Read Consistency



Update has not yet been processed by London

Martin sees room as available

inconsistent reads

Pramod books last hotel room on Mumbai node

Cindy sees room as booked

Read Consistency

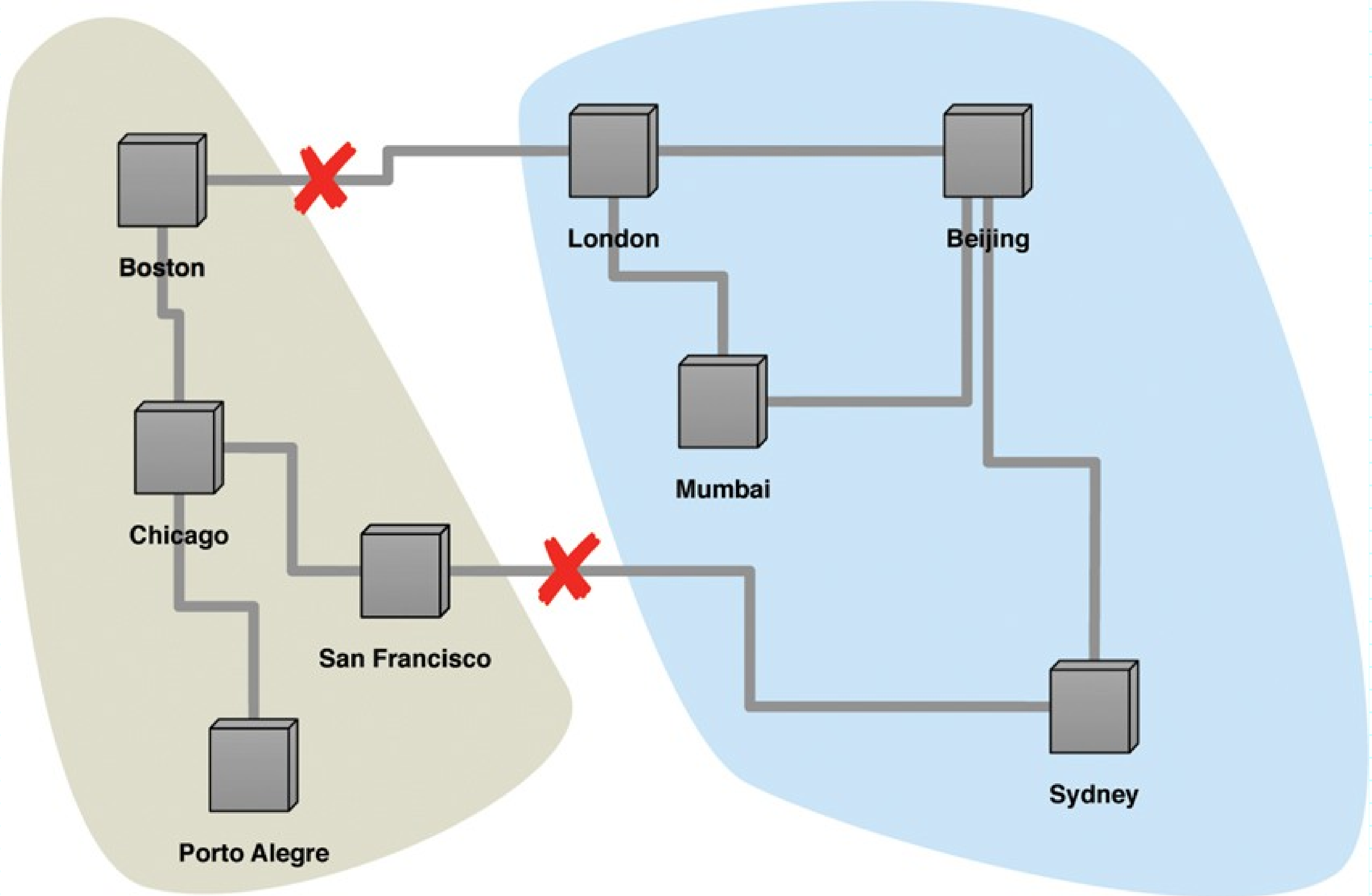
- Read-your-writes consistency
- Session consistency
  - Sticky Sessions

Relaxing Consistency

- It is always possible to design a system that is consistent
  - Why might we want to sacrifice this?

- CAP Theorem

Partition Tolerance



CAP Theorem

- What is an example of a system without partition tolerance?
  - Should we aim for this?

- It is all about the trade off!

CAP Theorem

- How can we improve consistency?
- How can we improve availability?
- Inconsistent writes
  - What about reads?

Quorums

- How many nodes do we actually need to get consistency?
- Write quorums
- Replication factor
- How does this affect reads?