

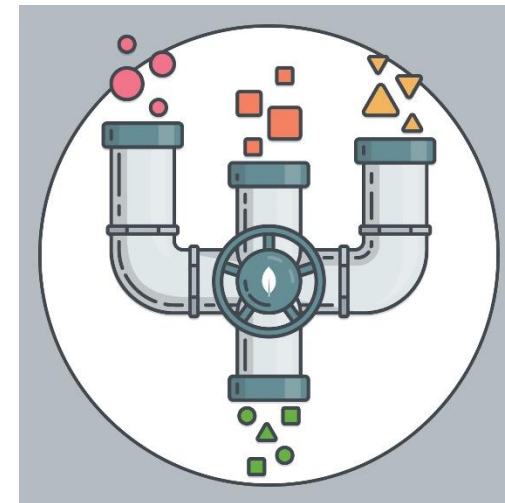


Day 28



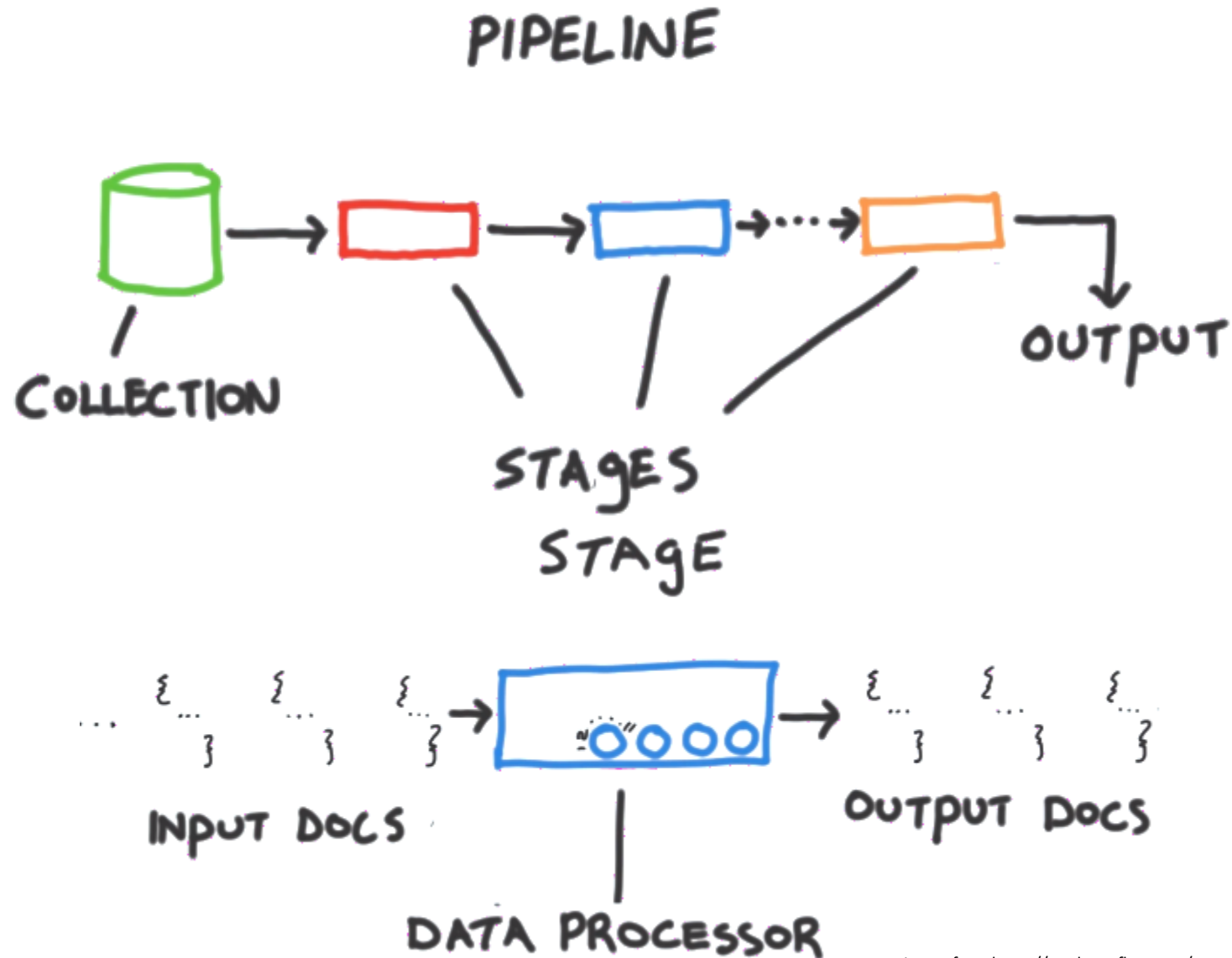
Aggregation

- More powerful way of reshaping data easily
 - Without resorting to writing application
- Aggregation pipeline is a series of document transformers
 - Each transformer will received a stream of documents
 - Either directly from the collection, or
 - From an earlier transformer
 - Transformer process the document and outputs the transformed document



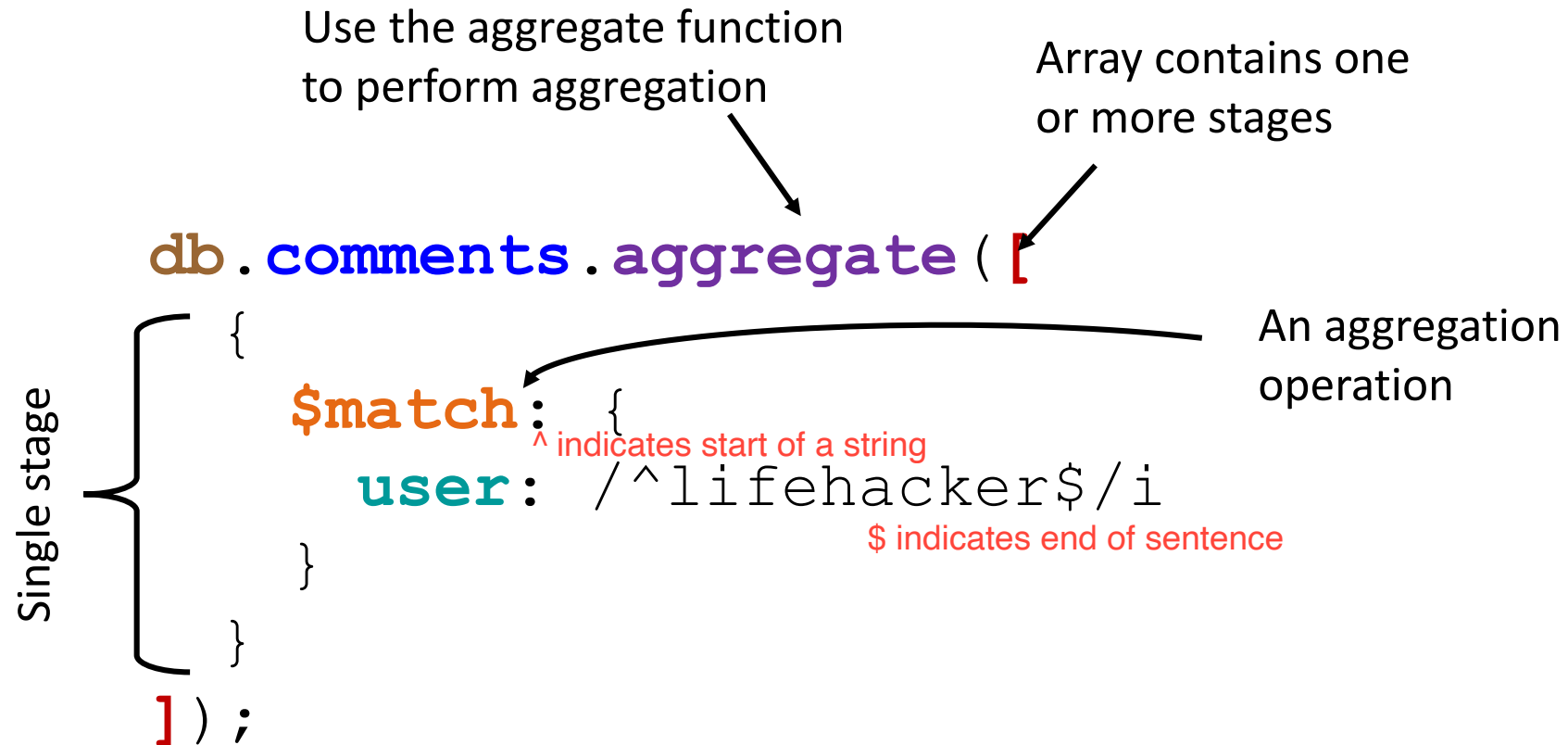


MongoDB Aggregation



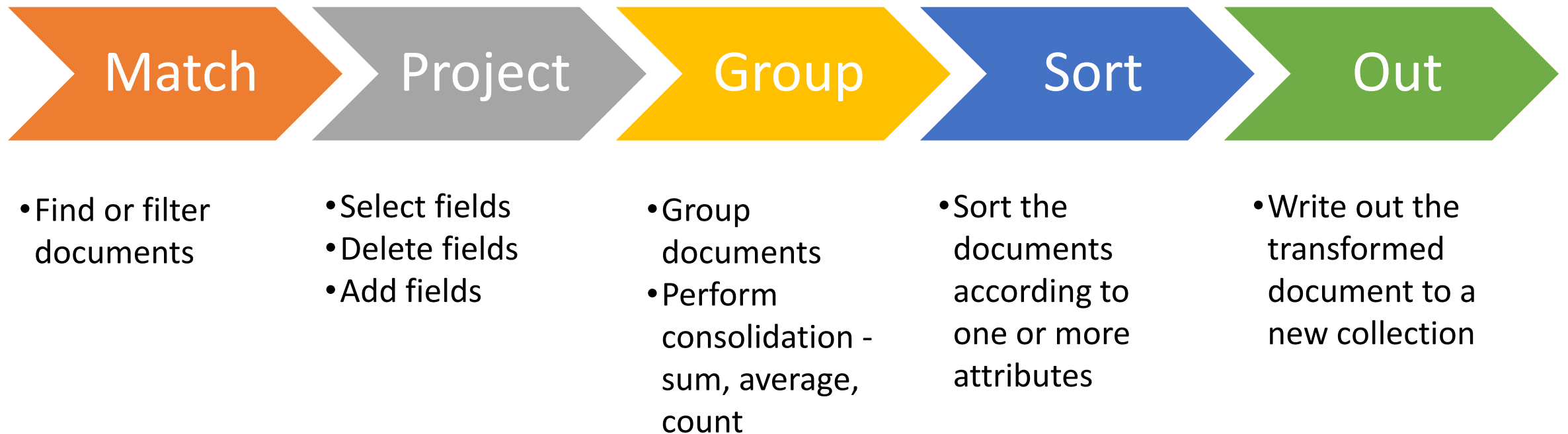


Aggregation Example





Typical Pipeline

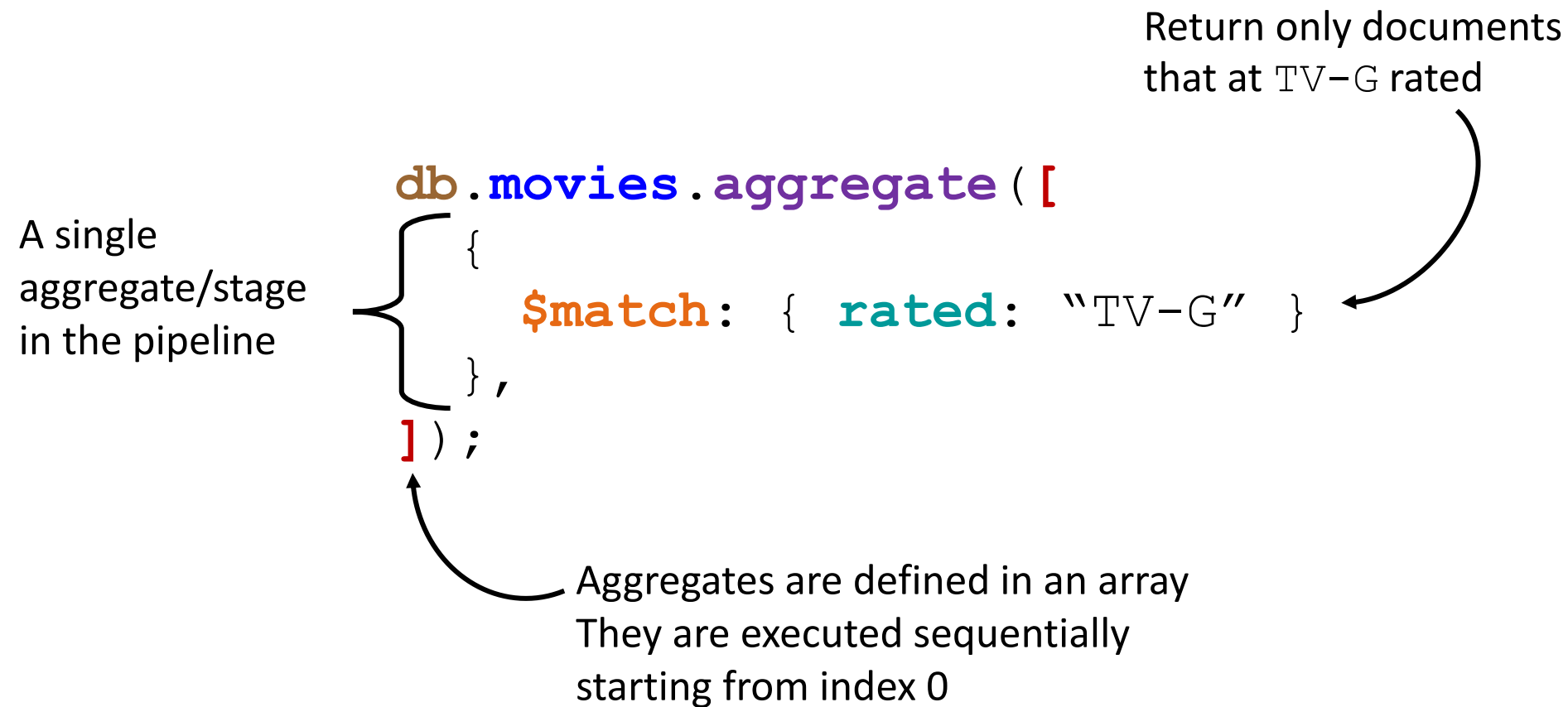




Pipeline Operators

- `$match`
 - Filter documents
- `$project`
 - Reshape documents
- `$group`
 - Summarizes documents
- `$sort`
 - Orders the documents
- `$limit`, `$skip`, `$first`, `$last`
 - Paginate documents
- `$unwind`
 - Unroll an array into multiple documents
- `$lookup`
 - Join 2 collections
- `$out`
 - Create a new collection

Filtering Documents





Aggregation with MongoTemplate

- Uses the **aggregate** method
- Creates one or more aggregating stages with static methods in **Aggregation**
 - `Aggregation.match()`, `Aggregation.group()`,
`Aggregation.project()`, `Aggregation.sort()`,
`Aggregation.bucket()`, `Aggregation.lookup()`,
`Aggregation.limit()`
 - See document for other aggregate operations
- Aggregates are assembled into a pipeline with the `Aggregation.newAggregate()` method



Filtering Documents

Create a match aggregate

```
MatchOperation matchRated = Aggregation.match(  
    Criteria.where("rated").is("TV-G")  
);
```

Assemble the aggregates
into a pipeline

```
Aggregation pipeline = Aggregation.newAggregation(matchRated);
```

```
AggregationResults<Document> results = mongoTemplate.aggregate(  
    pipeline, "movies", Document.class);
```

Perform the aggregation on the collection
with the defined pipeline.
Results are returned as Document



Filtering and Projecting Documents

```
db.movies.aggregate([
  {
    $match: { rated: "TV-G" }
  },
  {
    $project: { _id: -1, title: 1, fullplot: 1 }
  }
]) ;
```



Filtering and Projecting Documents

```
MatchOperation matchRated = Aggregation.match(  
    Criteria.where("rated").is("TV-G")  
);  
ProjectOperation projectFields = Aggregation.project(  
    "title", "fullplot")  
    .andExclude("_id");  
  
Aggregation pipeline = Aggregation.newAggregation(  
    matchRated, projectFields);  
  
AggregationResults<Document> results = mongoTemplate.aggregate(  
    pipeline, "movies", Document.class);
```

Assemble the aggregates
into a pipeline



Grouping

<code>_id: 1</code> <code>title: Spider Man - No Way Home</code> <code>year: 2021</code>
<code>_id: 2</code> <code>title: Top Gun</code> <code>year: 1986</code>
<code>_id: 3</code> <code>title: From Dusk Till Dawn</code> <code>year: 1996</code>
<code>_id: 4</code> <code>title: Highlander</code> <code>year: 1986</code>
<code>_id: 5</code> <code>title: Wargames</code> <code>year: 1983</code>

Group movies by release year



The field used for grouping (`year`) becomes the id of the grouped documents

<code>_id: 2021</code> <code>titles: ["Spider Man - No Way Home"]</code>
<code>_id: 1986</code> <code>titles: ["Top Gun", "Highlander"]</code>
<code>_id: 1996</code> <code>titles: ["From Dusk Till Dawn"]</code>
<code>_id: 1983</code> <code>titles: ["Wargames"]</code>

Grouping

Group the documents by
rated.
Count the number of
documents in each group
Add the `title` field to a new
field which is an array called
`titles`

Sort the document is
ascending order by
`count` field, the
number of titles in
each group

```
db.movies.aggregate ( [
  {
    $group: {
      _id: "$rated",
      count: { $sum: 1 },
      titles: { $push: "$title" }
    },
    $sort: { count: -1 }
  ] );
```

The \$ sign indicates the value of
the attribute from the incoming
document. Only appears in RHS

New fields in the
output document



Grouping - Output

```
{
  _id: "R",
  count: 5538,
  titles: [ ... ]
},
{
  _id: "PG-13",
  count: 2232
  titles: [ ... ]
},
{
  _id: "PG",
  count: 1853,
  titles: [ ... ]
}
...
```



Grouping

Creates a new field in the output document
from the aggregated fields from the input

```
GroupOperation groupByRated = Aggregation.group("rated")  
    .push("title").as("titles")  
    .count().as("count");
```

```
SortOperation sortByCount = Aggregation.sort(  
    Sort.by(Direction.ASC, "count"));
```

```
Aggregation pipeline = Aggregation.newAggregation(  
    groupByRated, sortByCount);
```

```
AggregationResults<Document> results = mongoTemplate.aggregate(  
    pipeline, "movies", Document.class);
```



Projection

- Projection in aggregation can do the following
 - Suppress fields
 - Add new fields
- Uses cases for projection
 - Reduce the result size
 - Transform the structure of a document

Projection

```

db.movies.aggregate ( [
  {
    $project: {
      _id: 1, title : 1,
      summary: "$plot"
    },
    {
      $sort: { title: 1 }
    }
  ] ) ;

```

New attribute

Keep these 2 fields



Projection

and() is used to create a new property

as() is used to create a new property

```
ProjectionOperation projectMovieSummary =  
    Aggregation.project("_id", "title")  
    .and("plot").as("summary")
```

```
SortOperation sortByTitle = Aggregation.sort(  
    Sort.by(Direction.ASC, "title"));
```

```
Aggregation pipeline = Aggregation.newAggregation(  
    projectMovieSummary, sortByTitle);
```

```
AggregationResults<Document> results = mongoTemplate.aggregate(  
    pipeline, "movies", Document.class);
```



AggregationExpression

- Most of the `project()`, `match()`, `sort()` etc provides simple operations to working with fields
 - Usually a single field
- More complex use cases, require using `AggregationExpression`
- Provides string, maths, conversions, array, bucket, date, etc operations

Projection

```

db.movies.aggregate ( [
  {
    $project: {
      _id: 1,
      title: {
        $concat: [ "$title", " (", "$rated", ")" ]
      }
    },
    $sort: { title: 1 }
  }
] );

```

New attribute

Change an existing field



Projection

```
ProjectionOperation projectMovieSummary =
    Aggregation.project("_id", "title")
        .and("plot").as("summary")
        .and(
            StringOperators.Concat.valueOf("name").concat(" ")
                .concatValueOf("rated").concat(" ")
        ).as("title");

SortOperation sortByTitle = Aggregation.sort(
    Sort.by(Direction.ASC, "title"));

Aggregation pipeline = Aggregation.newAggregation(
    projectMovieSummary, sortByTitle);

AggregationResults<Document> results = mongoTemplate.aggregate(
    pipeline, "movies", Document.class);
```



Projection - Output

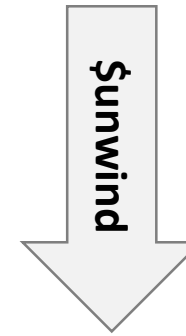
```
{
  _id: ObjectId(...),
  title: "101 Dalmations (G)",
  summary: "...",
},
{
  _id: ObjectId(...),
  title: "101 Reykjavek (NOT RATED)",
  plot: "...",
},
{
  _id: ObjectId(...),
  title: "12 (PG-13)",
  plot: "...",
},
```



Unwinding Arrays

- Cannot process elements in an array
 - No loops within an aggregation pipeline
- Need to simplify the document by expanding the array
 - Duplicate a document for every element in the array
- Final document will no longer have any arrays in any of its fields

```
_id: 123  
title: Where Eagles Dare  
genres: [ "Action",  
"Adventure", "War" ]
```



```
_id: 123  
title: Where Eagles Dare  
genres: "Action"
```

```
_id: 123  
title: Where Eagles Dare  
genres: "Adventure"
```

```
_id: 123  
title: Where Eagles Dare  
genres: "War"
```



Unwinding an Array

```
{  
  _id: ObjectId("abc123"),  
  title: "Dark Crystal",  
  ...  
  genres: [ "Adventure", "Family", "Fantasy" ],  
  rating: "PG"  
}
```

Unwind this array

```
db.tv_shows.aggregate([  
  {  
    $unwind: "$genres"  
  }  
]);
```

```
{  
  _id: ObjectId("abc123"),  
  title: "Dark Crystal",  
  ...  
  genres: "Adventure",  
  rating: "PG"  
},  
{  
  _id: ObjectId("abc123"),  
  title: "Dark Crystal",  
  ...  
  genres: "Family",  
  rating: "PG"  
},  
{  
  _id: ObjectId("abc123"),  
  title: "Dark Crystal",  
  ...  
  genres: "Fantasy",  
  rating: "PG"  
}
```


Unwinding an Array

```
db.tv_shows.aggregate ([
  {
    $unwind: "$genres"
  },
  {
    $group: {
      _id: "$genres",
      titles: {
        $push: "$title"
      },
      titles_count: {
        $sum: 1
      }
    }
  }
]) ;
```

The incoming document's `genres` attribute is an array. 'Flattens' the genres array producing one document per array element

The unwound document's `genres` is now a single value



Unwinding an Array

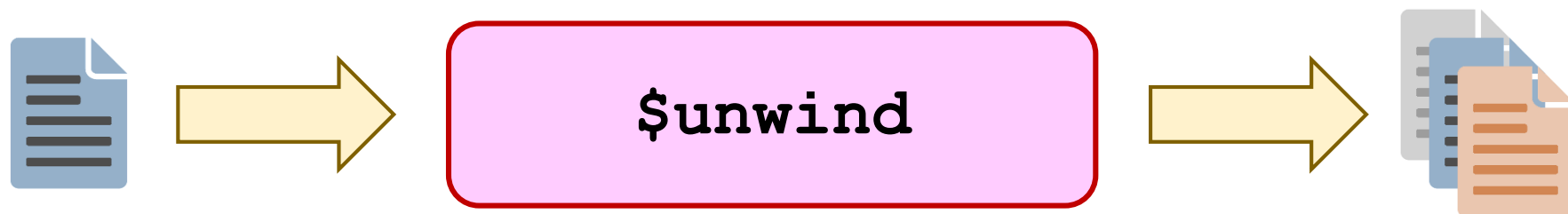
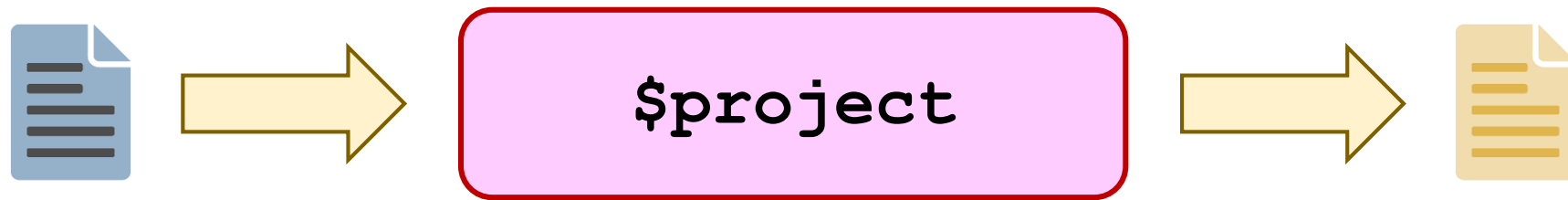
```
AggregationOperation unwindGenres = Aggregation.unwind("genres");
GroupOperation groupByGenres = Aggregation.group("genres")
    .push("title").as("titles")
    .count().as("titles_count");
ProjectionOperation projectGenresSummary =
    Aggregation.project("_id", "titles", "titles_count");
SortOperation sortByTitlesCount = Aggregation.sort(
    Sort.by(Direction.DESC, "titles_count"));

Aggregation pipeline = Aggregation.newAggregation(unwindGenres,
    groupByGenres, projectGenresSummary, sortByTitlesCount);

AggregationResults<Document> results = mongoTemplate.aggregate(
    pipeline, "movies", Document.class);
```



\$project, \$group, \$unwind





Classifying Documents into Buckets

```
db.movies.aggregate ([  
  {  
    $bucket: {  
      groupBy: "$imdb.rating",  
      boundaries: [ 3, 5, 8 ],  
      default: 'Others',  
    }  
  }  
]) ;
```

rating < 3
3 <= rating < 5
5 <= rating < 8
rating >= 8



Create 4 buckets to group the documents based on their IMDB ratings



Bucket - Output

```
{
  _id: 3.0,
  count: <a number>
},
{
  _id: 5.0,
  count: <a number>
},
{
  _id: 8.0,
  count: <a number>
},
{
  _id: ">8",
  count: <a number>
}
```



Classifying Documents into Buckets

```
BucketOperation bucketRating = Aggregation.bucket("imdb.rating")  
    .withBoundaries(3, 5, 8)  
    .withDefaultBucket(">8");
```

```
Aggregation pipeline = Aggregation.newAggregation(bucketRating);
```

```
AggregationResults<Document> results = mongoTemplate.aggregate(  
    pipeline, "movies", Document.class);
```



Classifying Documents into Buckets



```
db.tv_shows.aggregate([
```

```
{
```

```
  $bucket: {
```

```
    groupBy: "$imdb.rating",
```

```
    boundaries: [ 3, 5, 8 ],
```

```
    default: 'Others',
```

```
    output: {
```

```
      count: { $sum: 1 },
```

```
      titles: { $push: "$title" }
```

```
    }
```

```
  }
```

```
}
```

```
] );
```

3 <= rating < 5

5 <= rating < 8

rating >= 8

Documents are classified
according to the following
boundaries

Any document that cannot be
sorted into a bucket goes to
the default bucket

Shape of
the output
document/
bucket



Bucket - Output

```
{
  _id: 3.0,
  count: <a number>,
  titles: [ ... ]
},
{
  _id: 5.0,
  count: <a number>,
  titles: [ ... ]
},
{
  _id: 8.0,
  count: <a number>,
  titles: [ ... ]
},
{
  _id: ">8",
  count: <a number>,
  titles: [ ... ]
}
```




Classifying Documents into Buckets

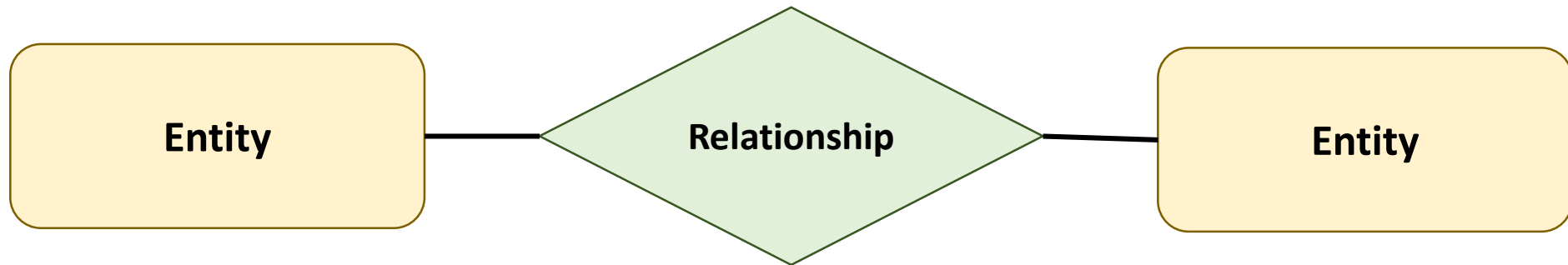
```
BucketOperation bucketRating = Aggregation.bucket("imdb.rating")  
    .withBoundaries(3, 5, 8)  
    .withDefaultBucket(">8")  
    .andOutputCount().as("count")  
    .andOutput("title").push().as("titles");
```

```
Aggregation pipeline = Aggregation.newAggregation(bucketRating);
```

```
AggregationResults<Document> results = mongoTemplate.aggregate(  
    pipeline, "movies", Document.class);
```



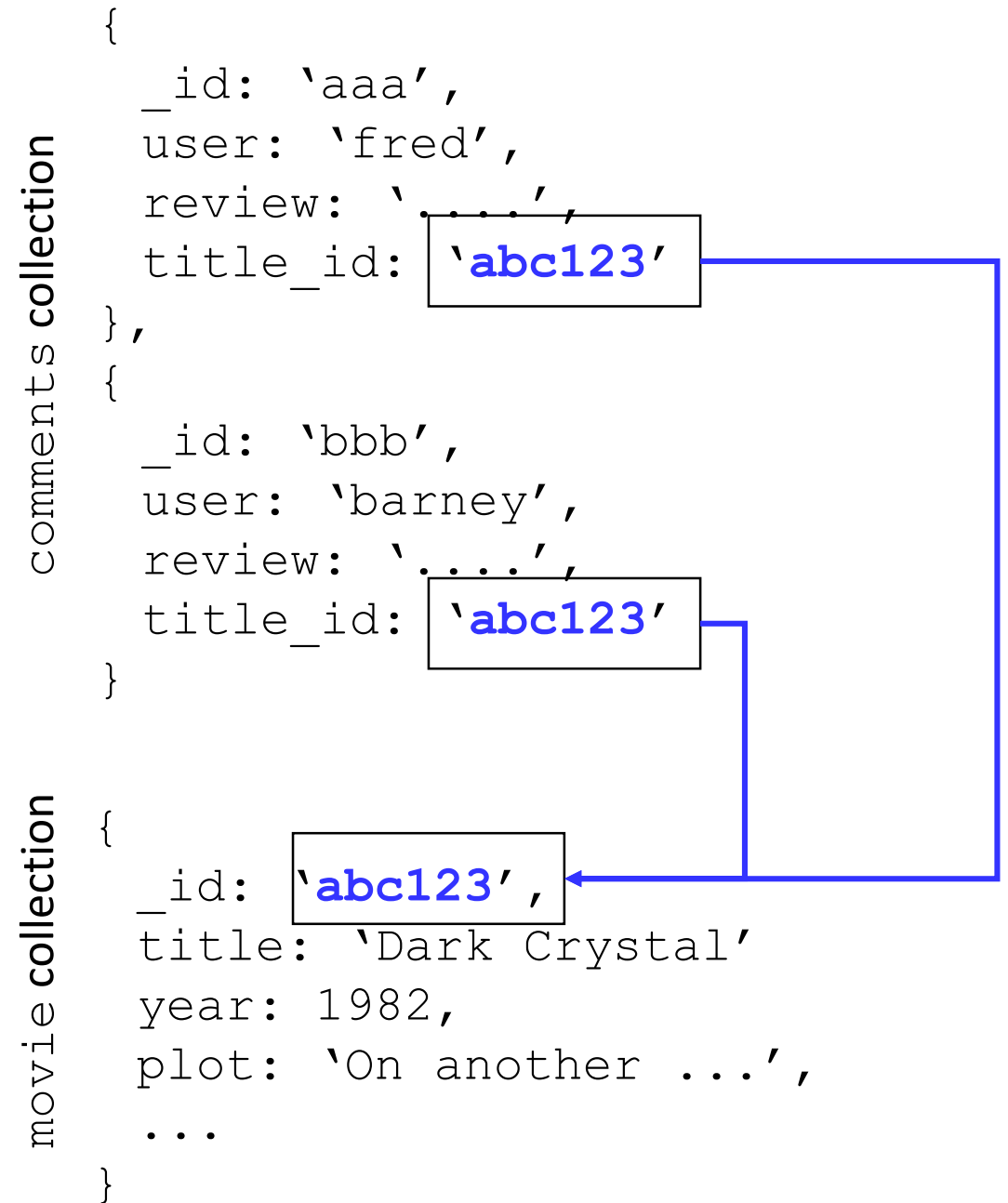
Modelling Relationships





Referenced Document

- Referenced one document from another
 - Using an attribute to hold the value of the other document's `_id`
 - Like relational database
- Require 2 access to retrieve the document and its corresponding 'child'
 - Eg. a movie and all its related reviews

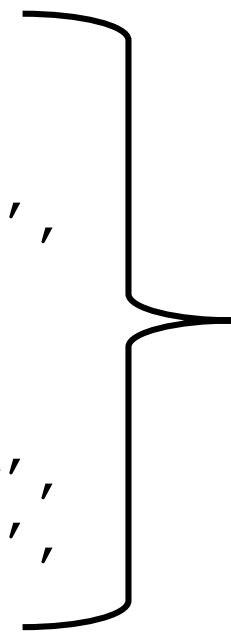




Embedded Document

- Embed related documents into the parent
 - De-normalization
- Embedding defines the relationship
 - Eg. embedding all reviews into the related movie
- MongoDB has a cap of 16MB per document
 - Embedded documents may cause the container to grow beyond this size
 - Have to use GridFS

```
{
  _id: 'abc123',
  title: 'Dark Crystal',
  year: 1982,
  plot: 'On another ...',
  ...
  reviews: [
    {
      _id: 'aaa',
      user: 'fred',
      review: '....',
    },
    {
      _id: 'bbb',
      user: 'barney',
      review: '....',
    }
  ]
}
```

A large right-facing curly bracket groups the two objects within the 'reviews' array. To the right of the bracket, the text 'Embedded documents' is written.

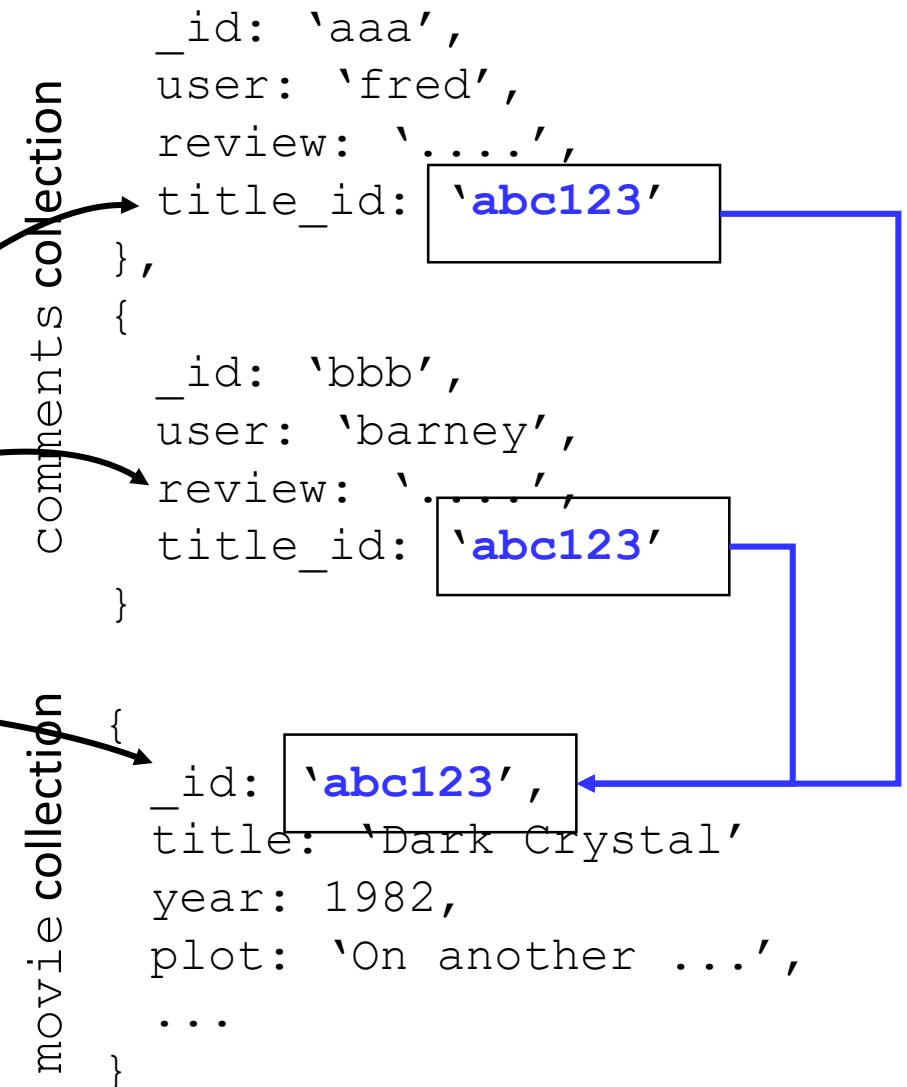
Embedded documents



Returning Referenced Documents as Embedded

```
db.tv_shows.aggregate([
{
  $match: { title: 'Brazil' }
},
{
  $lookup: {
    from: 'comments',
    foreignField: 'title_id',
    localField: '_id',
    as: 'reviews'
  }
}
]) ;
```

Create a new attribute in each of the output document





Returning Referenced Documents as Embedded

```
MatchOperation matchTitle = Aggregation.match(  
    Criteria.where("name").is("Brazil")  
);  
LookupOperation lookupComments = Aggregation.lookup(  
    "comments", "title_id", "_id", "reviews");  
  
Aggregation pipeline = Aggregation.newAggregation(  
    matchTitle, lookupComments);  
  
AggregationResults<Document> results = mongoTemplate.aggregate(  
    pipeline, "movies", Document.class);
```



Property of Database - ACID

- **A**tomicity
 - **C**onsistency
 - **I**solation
 - **D**urability
- Strong consistency
 - Isolation
 - Transactions
 - Vertical scaling/scale up
 - Precise result
 - Consistency first



Property of Database - BASE

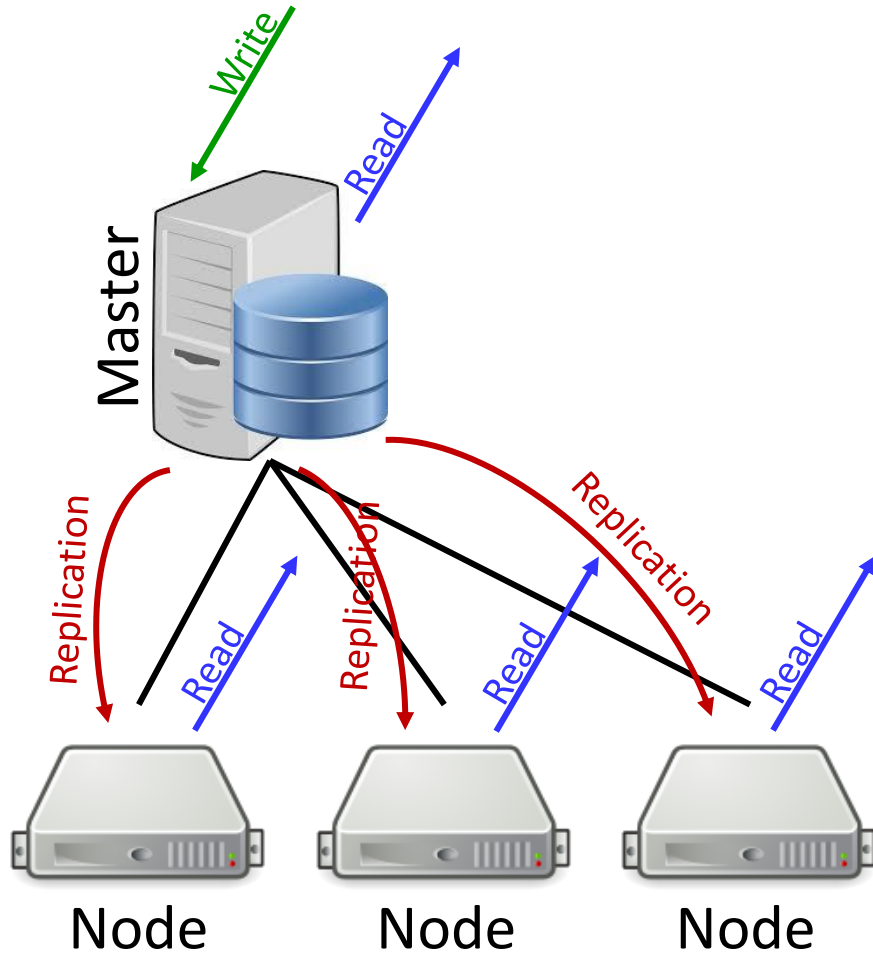
- **B**asically **A**vailable
- **S**oft state
- **E**ventual consistency
- Weak consistency
- Last write wins
- No transaction support
 - No longer true
- Horizontal scaling/scale out
- Approximate results
- Availability first



biggest problem with p2p is if there is a split in the network
(network partition)

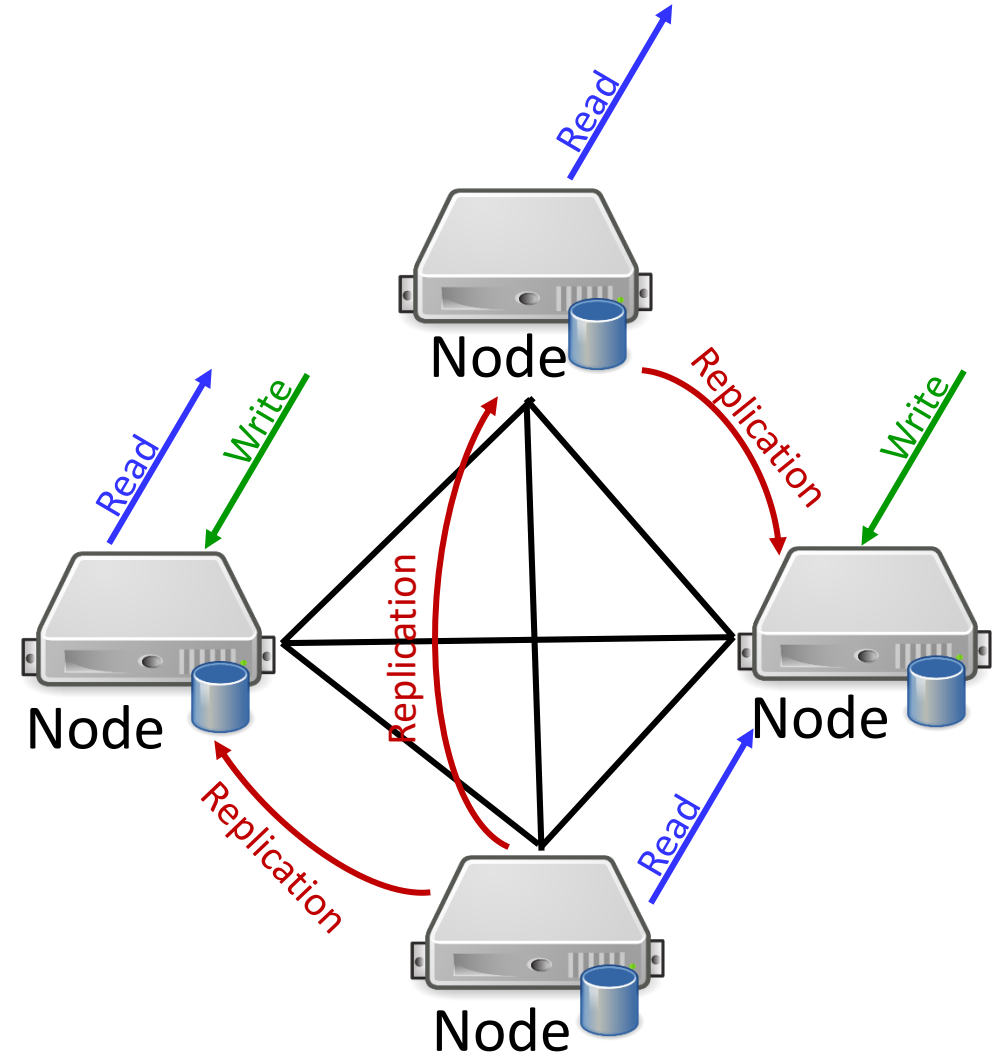
Architecture

Master-Slave



ACID

Peer-to-Peer



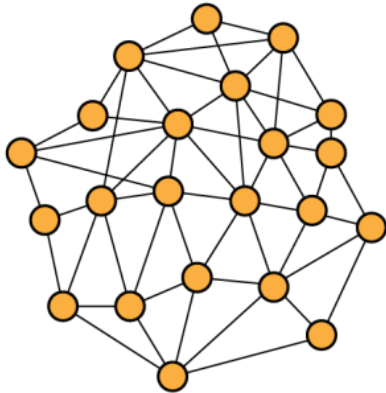
BASE



Property of the Architecture - CAP

C : Consistency

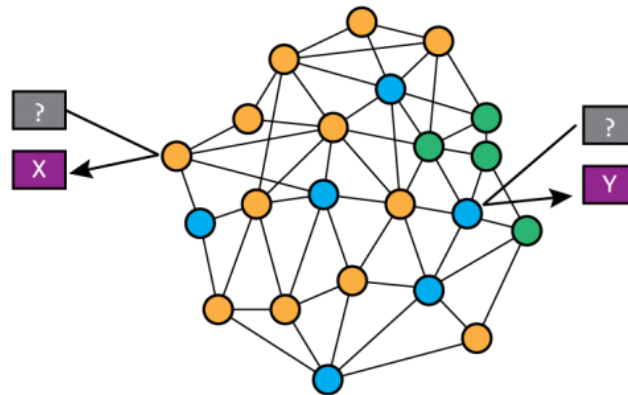
At any given time, all nodes in the network have exactly the same (most recent) **value**.



● = Value: X @ 2018-05-03T08:52:40

A : Availability

Every **request** to the network receives a **response**, though without any guarantee that returned data is the most recent.



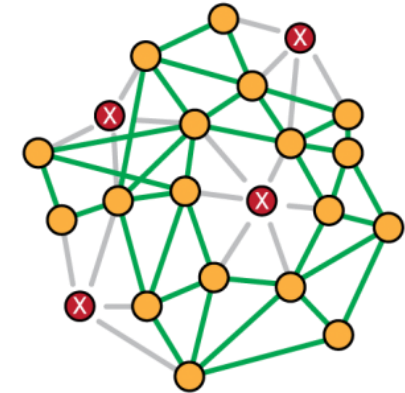
● = Value: X @ 2018-05-03T08:52:40

● = Value: Z @ 2018-05-03T08:32:58

● = Value: Y @ 2018-05-03T07:12:12

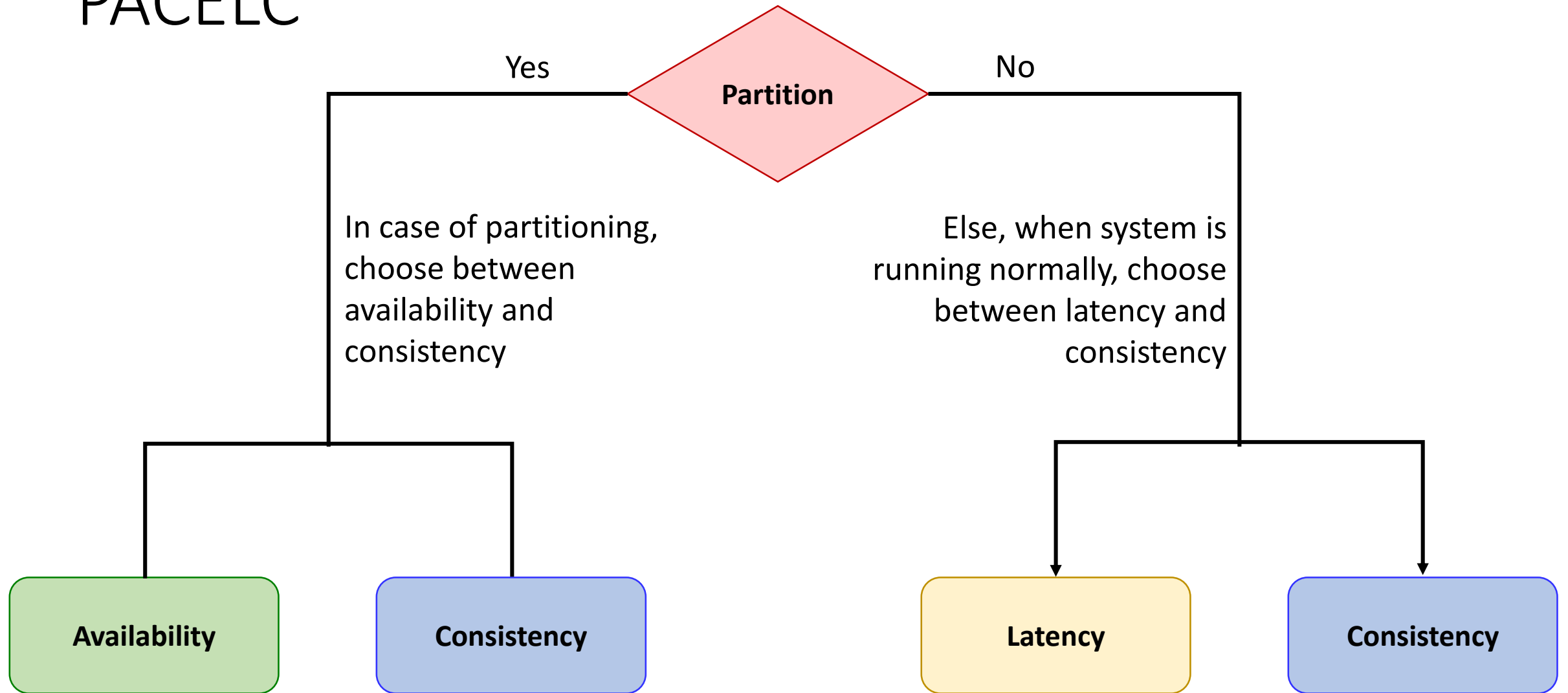
P : Partition tolerance

The network continues to operate, even if an arbitrary number of nodes are **failing**.





PACELC

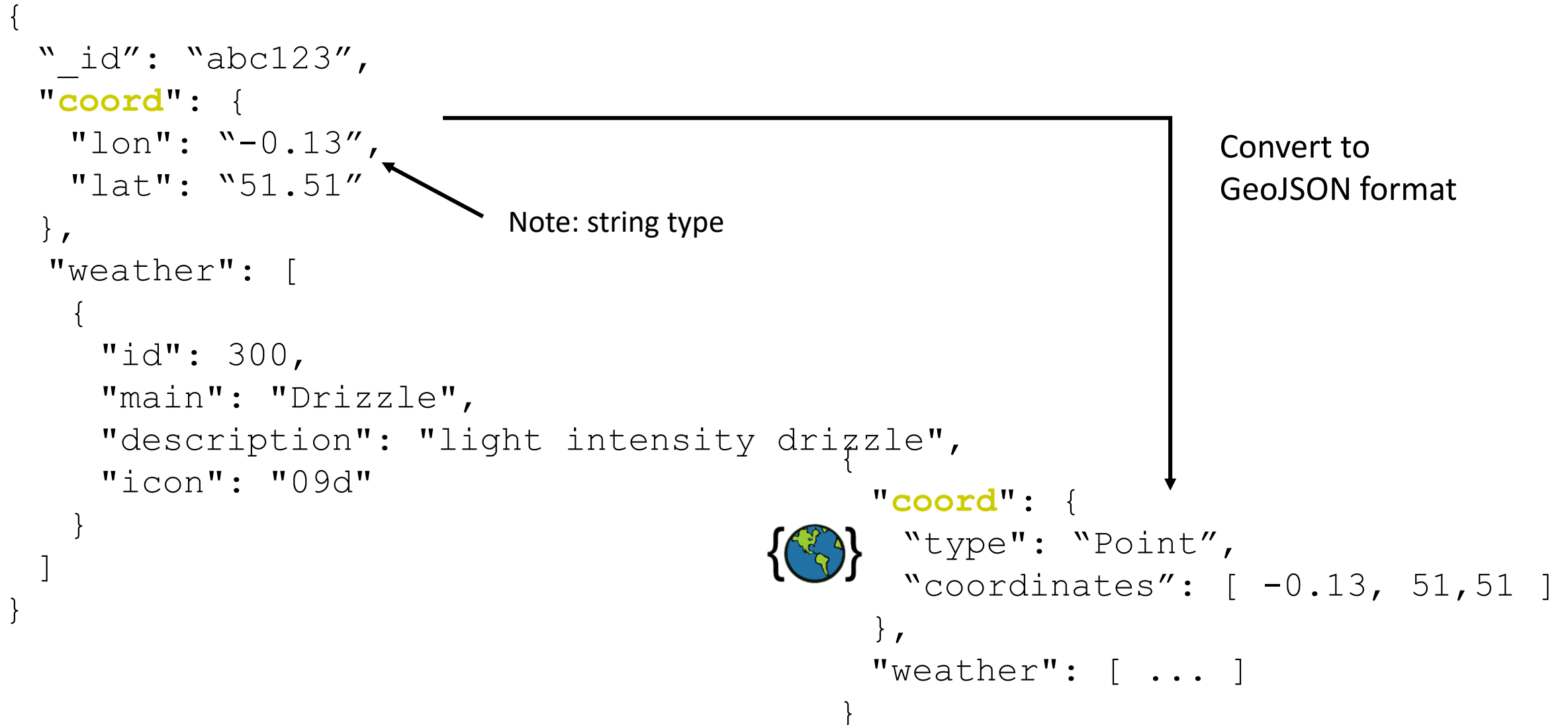




Unused



Projection - Converting Data





Projection

```
db.tv_shows.aggregate([
  {
    $project: {
      _id: 1,
      weather: 1
      coord: {
        type: "Point",
        coordinates: [
          { $convert: {
            input: "$coord.lon", to: "double",
            onError: 0.0, onNull: 0.0 }
          },
          { $convert: {
            input: "$coord.lat", to: "double",
            onError: 0.0, onNull: 0.0 }
          }
        ]
      }
    }
  }
]) ;
```

Convert from string to double. If the input fields are erroneous or null, then set the value to 0.0



Data Conversion - \$convert

```
{ dob: {
  $convert {
    input: "$dob.date",
    to: "date"
    onError: {
      $dateFromString: {
        dateString: "1970-01-01"
      }
    }
  }
}
```

```
$convert {
  input: "<input value>",
  to: "<data type>",
  onError: "<data type>",
  onNull: "<data type>"
}
```

• Shorthand

- \$toBool, \$toDate, \$toInt, \$toObjectId, ...



Projection - Array Elements

```
db.tv_shows.aggregate ([
{
  $project: {
    _id: 1,
    genres: {
      $filter: {
        input: "$genres"
        as: "genre",
        cond: { $eq: [ "$$genre": "Family" ] }
      }
    },
    plot: 1
  }
}] ) ;
```

\$filter loops through the array genres,
assign each value to control variable genre
Access control variable genre with \$\$

Will only project those array
elements that passes the predicate