A pragmatic introduction to

METEOR

Nick Lehmann, 14th September 2016

METEOR

A full-stack, open source platform for building web and mobile apps in JavaScript

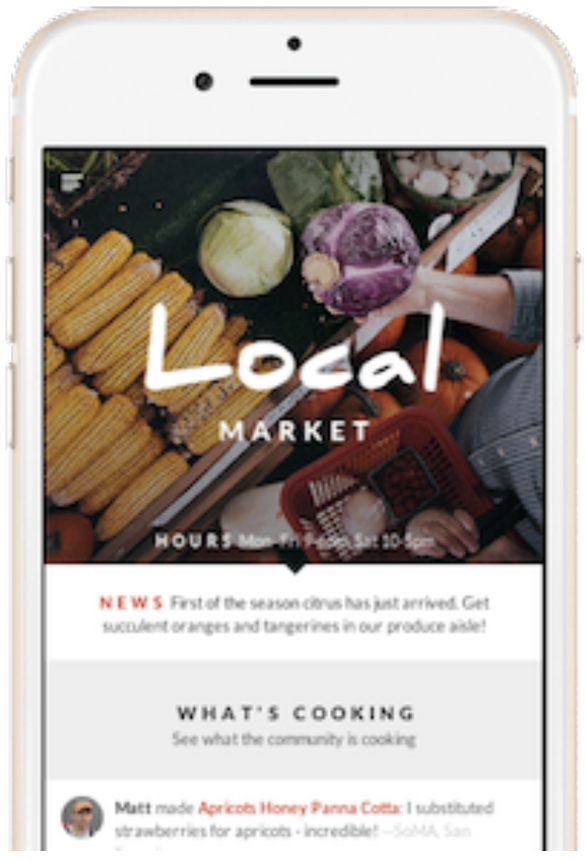
Isomorphic Javascript

One codebase, all platforms

Open, comprehensive ecosystem

Full stack reactivity

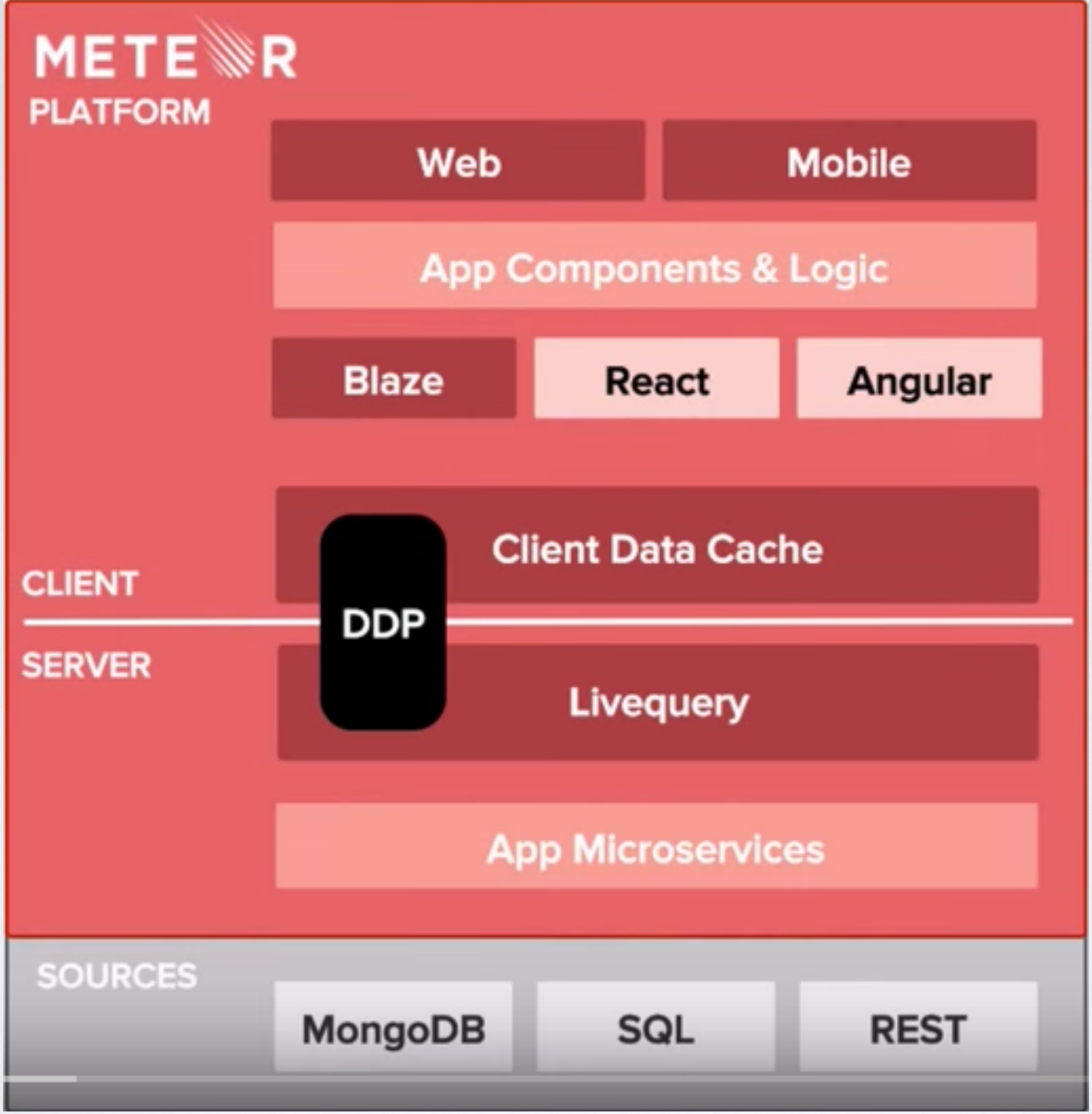
➡ fun development!!!



Collaborative iOS and Android app

<1000 lines of JavaScript

#1st in Github's Web Application section!



Source: An Introduction to Meteor, YouTube

Finding Documents

`db.ships.findOne()`	Finds one arbitrary document
`db.ships.find().prettyPrint()`	Finds all documents and using nice formatting
`db.ships.find({}, {name:true, _id:false})`	Shows only the names of the ships
`db.ships.findOne({'name':'USS Defiant'})`	Finds one document by attribute

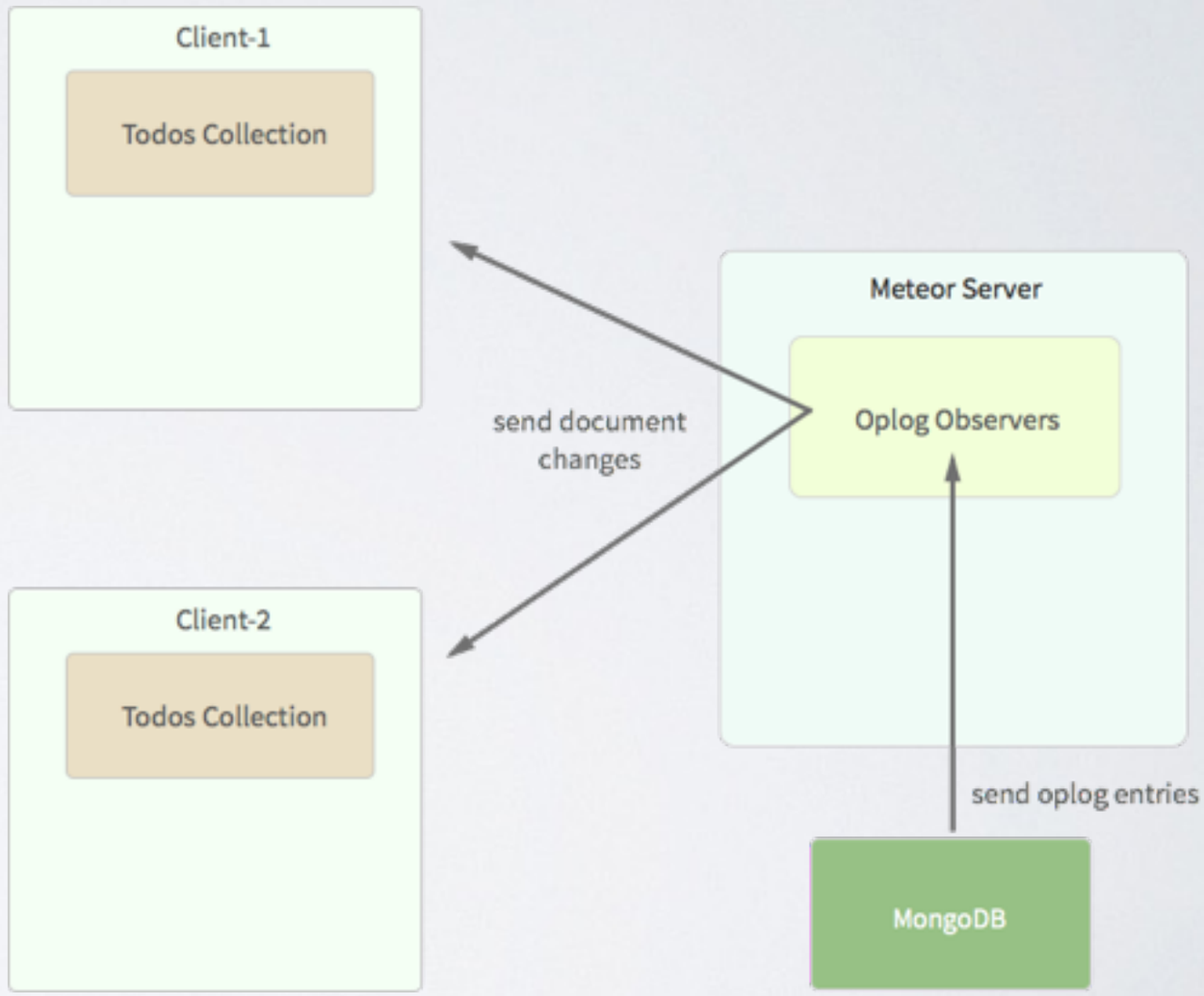
Inserting Documents

```
db.ships.insert({name:'USS Enterprise-D',operator:'Starfleet',type:'
db.ships.insert({name:'USS Prometheus',operator:'Starfleet',class:'P
db.ships.insert({name:'USS Defiant',operator:'Starfleet',class:'Defi
db.ships.insert({name:'IKS Buruk',operator:' Klingon Empire',class:'
db.ships.insert({name:'IKS Somraw',operator:' Klingon Empire',class:
db.ships.insert({name:'Scimitar',operator:'Romulan Star Empire',type
db.ships.insert({name:'Narada',operator:'Romulan Star Empire',type:'
```

What is LiveQuery?

Livequery

- default database: MongoDB
- makes real time updates effective
- publish-and-subscribe mechanism



```
Meteor.publish('tasks', function tasksPublication() {
    return Tasks.find();
});
```

- register publish function
- return Mongo Cursor
- Livequery will observe changes (via oplog)

```
Template.body.onCreated(function bodyOnCreated() {
    this.state = new ReactiveDict();
    Meteor.subscribe('tasks');
});
```

- client subscribes to specific publications
- retrieves changes
- caches them

Merge Box

- process that tries to identify exact changes that need to be sent to the client

- Limitations
  - only operates on top level fields
  - hard with complex sorting algorithms
  - causes additional CPU usage on the server

Does Meteor scale???

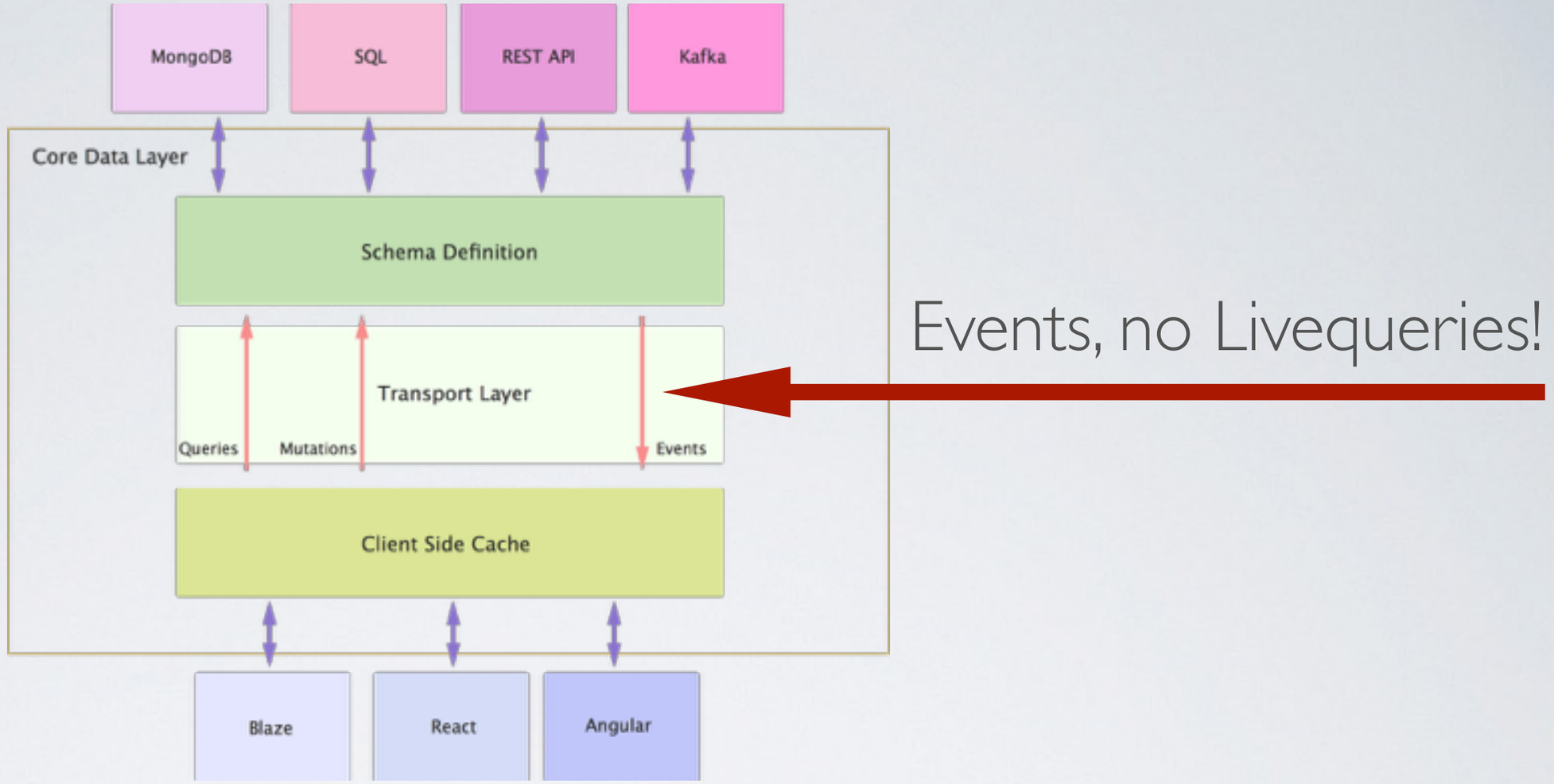
Meteor scaling

- public: Meteor does not scale!
- my version: Meteor does not scale with a lot of writes, to some extent

Better way to publish

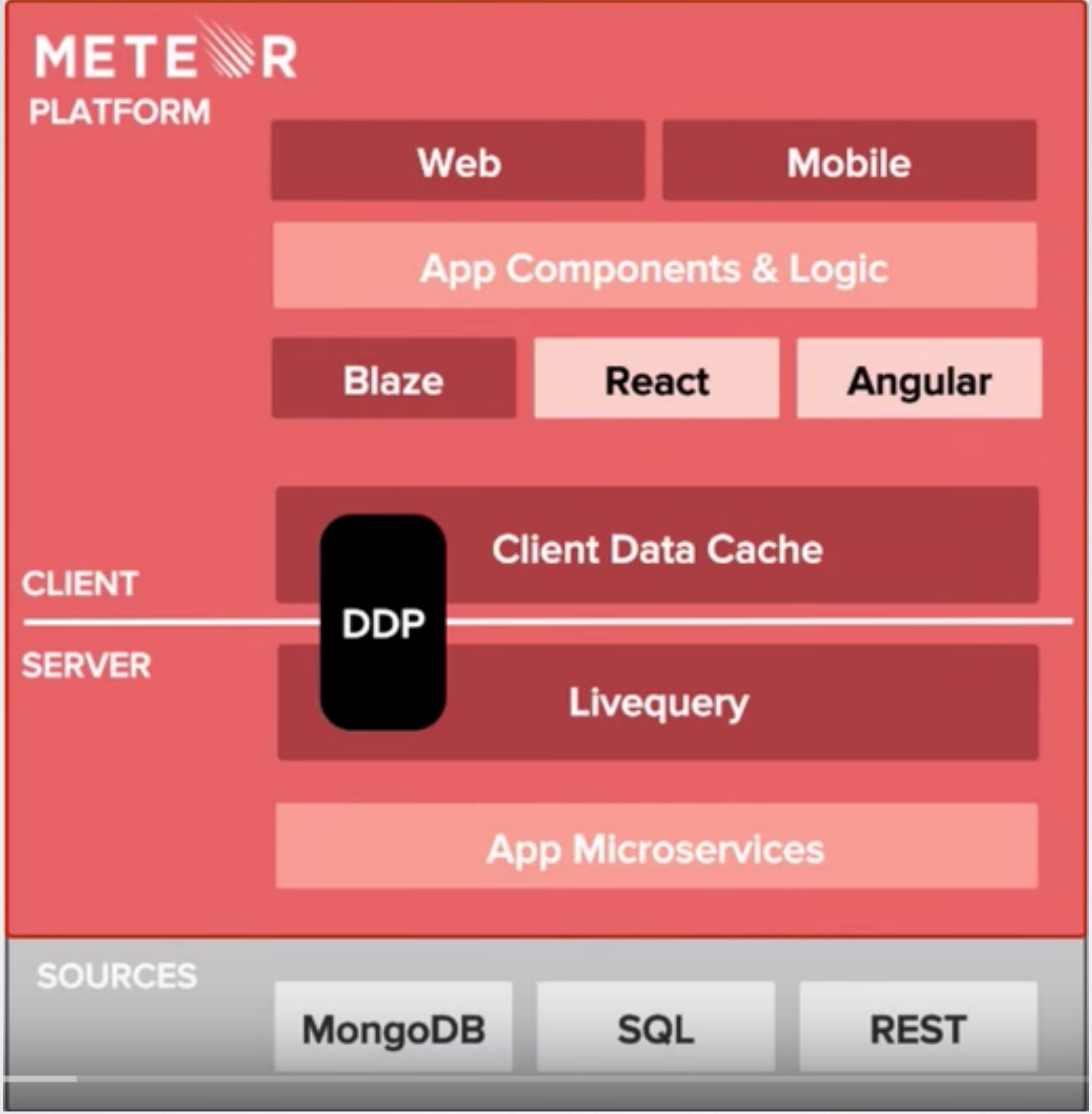
```
Meteor.publish('tasks', function tasksPublication() {
    return Tasks.find({
        $or: [{
            private: {
                $ne: true
            }
        }, {
            owner: this.userId
        }, ],
    });
});
```

There is no need to have all the data on the client!



Events, no Livequeries!

- proposal for new core data layer
- GraphQL seems to be preferred



Source: An Introduction to Meteor, YouTube

What is DDP?

The publish function

```
Mongo.Collection._publishCursor = function(cursor, sub, collection) {
    var observeHandle = cursor.observeChanges({
        added: function(id, fields) {
            sub.added(collection, id, fields);
        },
        changed: function(id, fields) {
            sub.changed(collection, id, fields);
        },
        removed: function(id) {
            sub.removed(collection, id);
        }
    });

    sub.onStop(function() {
        observeHandle.stop();
    });
};
```

Managing Data

- Core part of DDP
- built on top of bidirectional connections (Websocket)
- three notifications: added, changed, removed

Max

Server

subscribe todos(max_id)

added: todo1

added: todo2

ready

added: todo3 (from his mother)

```
1.{"msg": "sub", "id": "random-sub-id", "name": "todos", "params": ["max_id"]}
2.{"msg": "added", "collection": "todos", "id": "todo1_id",
    "fields": {"text": "First todo", "createdAt": "date1"}}
  {"msg": "added", "collection": "todos", "id": "todo2_id",
    "fields": {"text": "Second todo", "createdAt": "date2"}}
3.{"msg": "ready": "subs": ["random-sub-id"]}
4.{"msg": "added", "collection": "todos", "id": "todo3_id", "fields":
    {"text": "Clean up your room", "createdAt": "date3", "by": "mother", "priority": "high"}}
```

1. client sends subscription request
2. client receives added notifications with initial data
3. servers sends ready notification; initial data is sent
4. his mother adds a todo to his collection; will get it via added notification

```
// changed
{"msg": "changed", "collection": "todos", "id": "todo1_id",
    "fields": {"text": "First important todo", "checked": "true"}}

// removed
{"msg": "removed", "collection": "todos", "id": "todo1_id"}
```

Remote Procedure Call

- invoke methods on the server
- notifies client when all writes to other clients are done

Max’s mother

insertTodo(„Clean up your room“,
„max_id“, „high“)

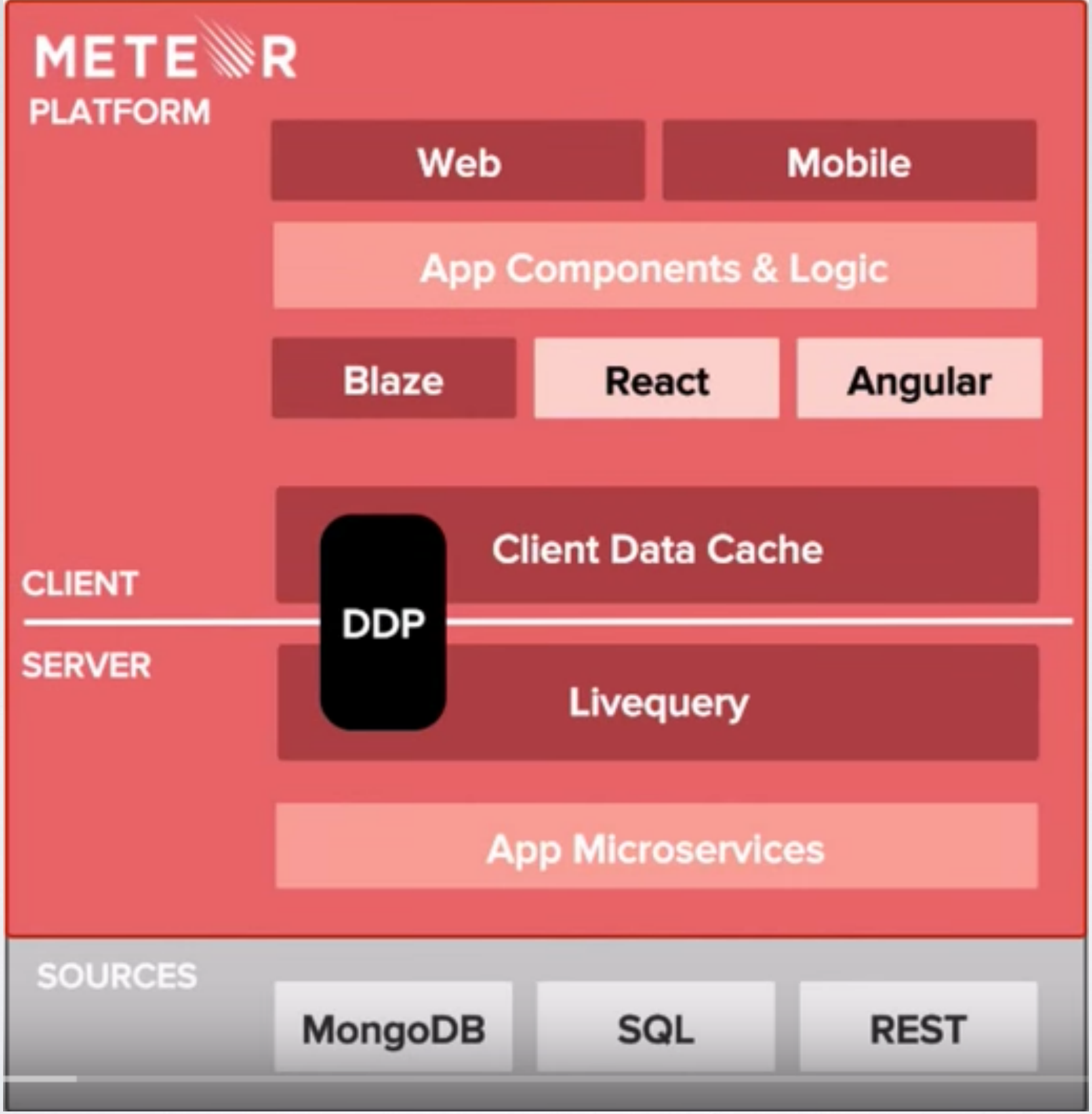
result: 1

updated

Server

```
1.{"msg":"method", "method": "insertTodo", "params":
["Clean up your room", "max_id", "high"], "id": "randomId-1"}
2.{"msg": "result", "id": "randomId-1": "result": "1"}
3.{"msg": "updated", "methods": ["randomId-1"]}
```

1. invoke method with set of params
2. return result (current number of todos)
3. notification that changes have been sent to Max successfully



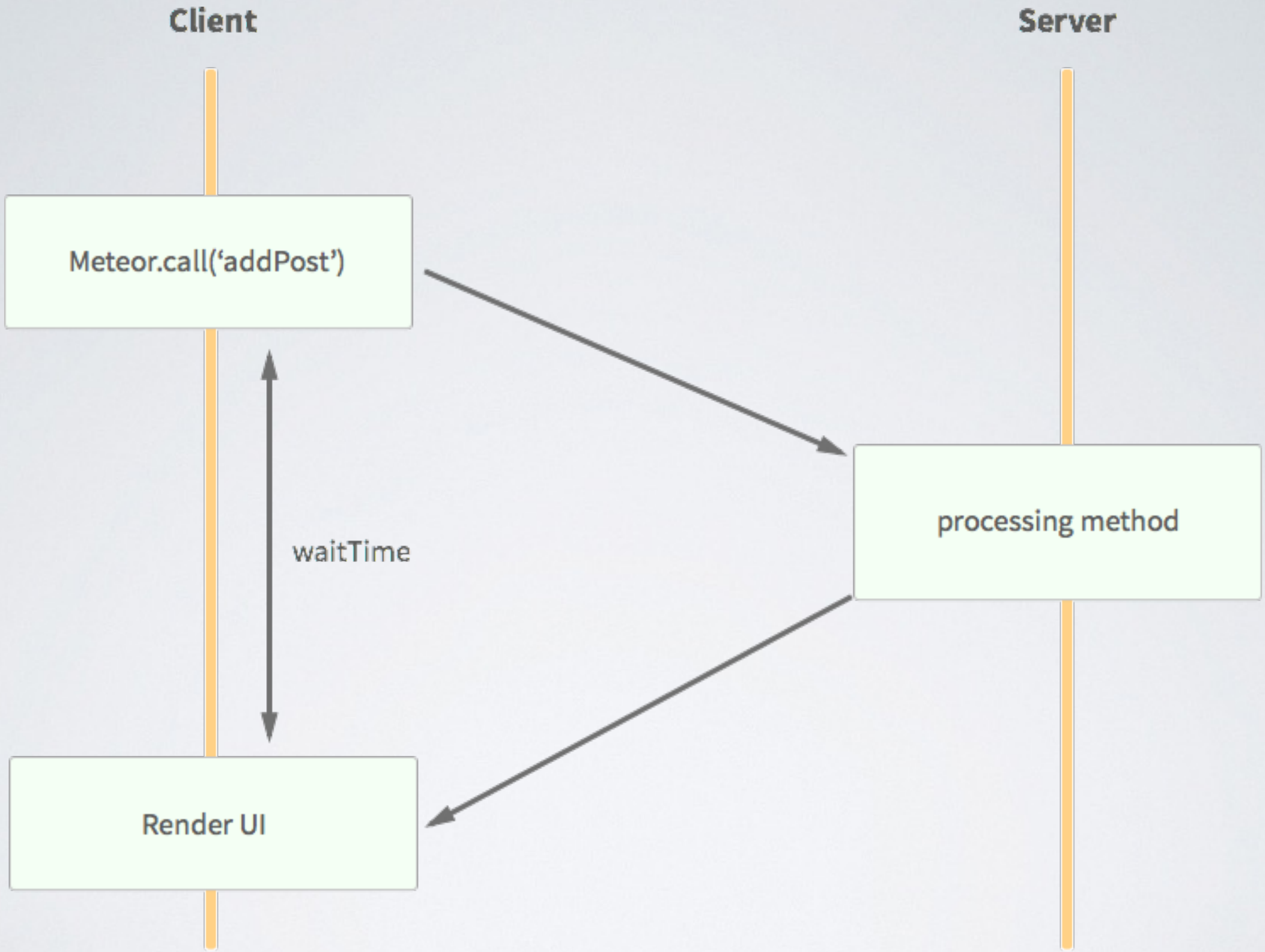
Source: An Introduction to Meteor, YouTube

What is **Minimongo**?

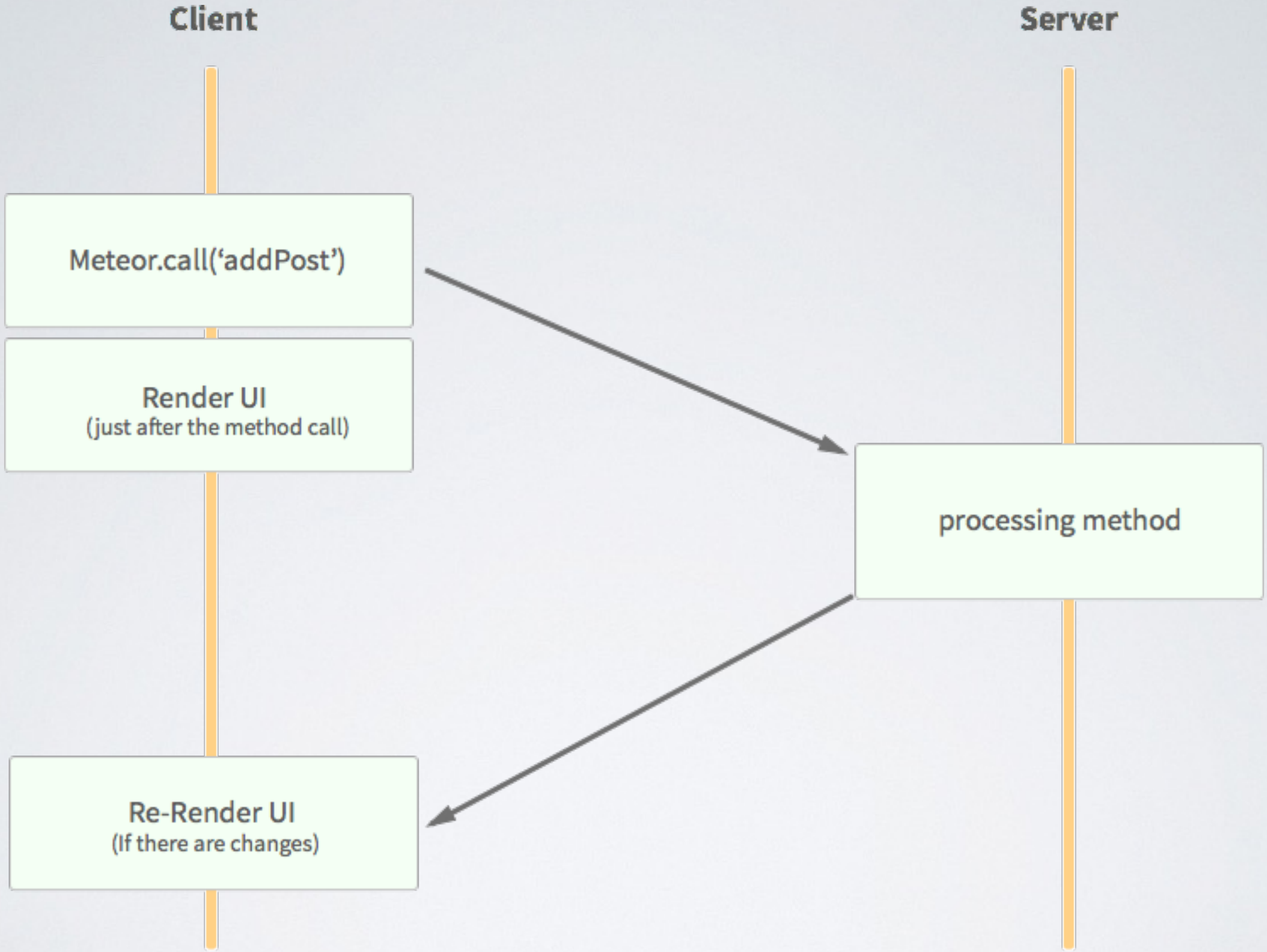
```
Todos = new Mongo.Collection('Todos');
Todos.insert({_id: 'my-todo'});
const todo = Todos.findOne({_id: 'my-todo'});
console.log(todo);
```

- MongoDB API implemented in JavaScript
- collection on client-side is a cache of the database
- **offline support**

Latency Compensation



Without Compensation



With Compensation

- optimistic UI, no waiting time
- problems:
  - only works for writes to data store supported by Meteor (MongoDB, Redis)
  - not working for third-party data
  - no problem with local collections, but methods must be available on client (method stubs)

More

can be skipped when time is coming to an end

What about the community?

WAIT BUT...

NPM!!

Packaging

1. NPM support since 1.3
2. **Full Stack** Package System
   (over 7000 packages)
3. Version solver

WAIT BUT...

Gulp!!

Build Tool

- Users don’t need to play with it
- Hot code reload
- **Multi Platform** build system

How does it look like?

MEAN - Isomorphic??!

DB

```
SELECT name
FROM users
WHERE id = 12345
```

Server

```
GET
http://server/users/
        name/12345
```

Client

```
var name =
    response.name;
```

Isomorphic!

DB

```
Users.find(
  {_id: 12345},
  {fields:
    {name : 1}
  }
)
```

Server

```
Users.find(
  {_id: 12345},
  {fields:
    {name : 1}
  }
)
```

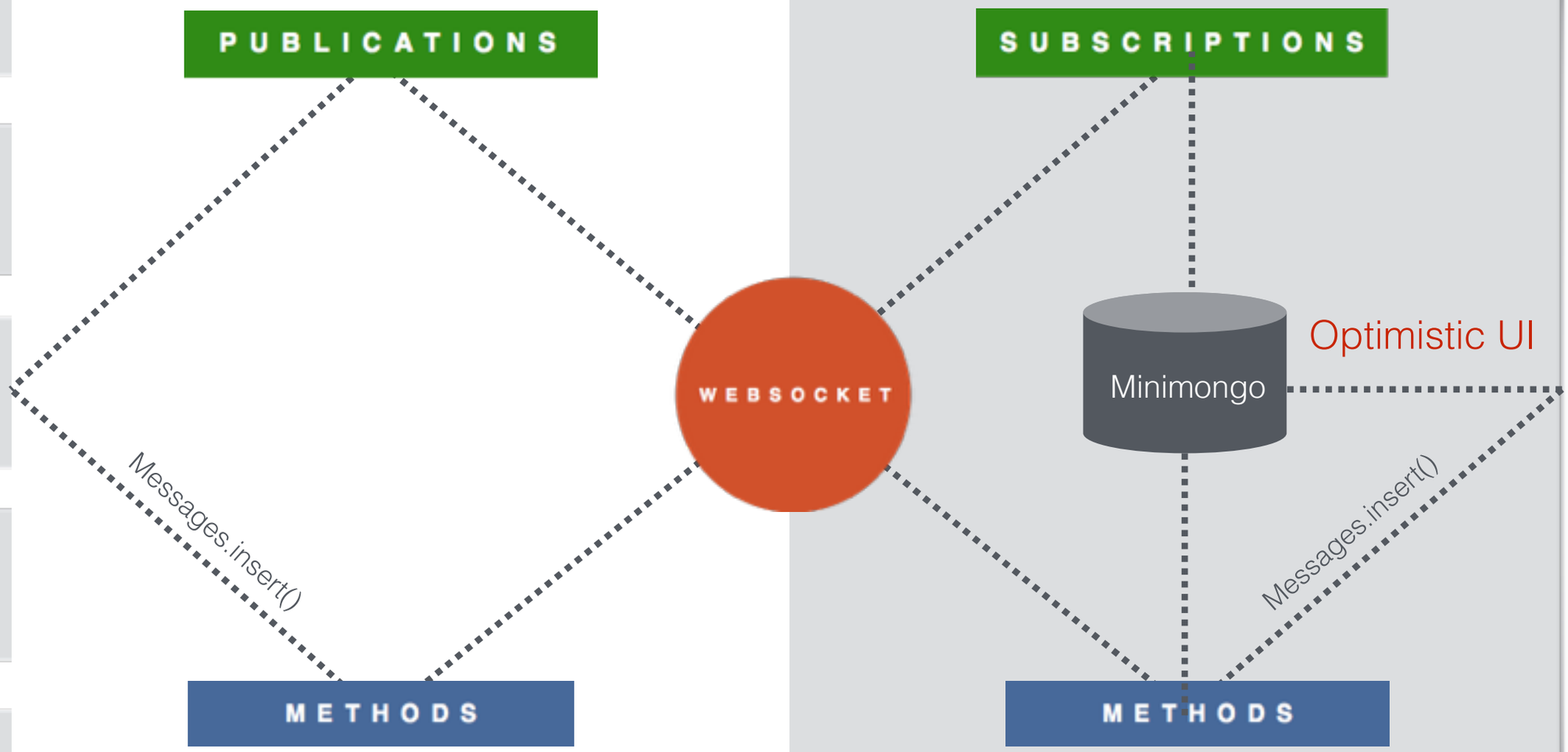
Client

```
Users.find(
  {_id: 12345},
  {fields:
    {name : 1}
  }
)
```


WHAT DO I LOOK LIKE
A GUY WHO'S NOT LAZY?

Now sum it up!

Meteor Data Flows



Assemble it yourself

HTML Templates

App Logic

Reactive UI update system

Native mobile container

Speculative client-side updates

Client-side data store

Custom data sync protocol

Realtime database monitoring

Build & update system

Microservices

Database

With Meteor

HTML Templates

App Logic

METEOR

Open-source
JavaScript app platform

Microservices

Database

And now some code!!!

Angular 2.0 Meteor

Why Angular?

- Angular 2.0 is built for connected client architecture
- easy syntax, just connect to Meteor collection
- easiest way to set up Angular 2 project

```
meteor create --example angular2-boilerplate
```

- all dependencies

- complete build process

```
import { ClassesCollection } from '../../../both/collections/classes-collection';
import { Class } from '../../../both/models/classes.interface';

import template from './classes.component.html';

@Component({
    selector: 'classes',
    template,
    directives: [ PageHeadingComponent ]
})
export class ClassesComponent extends MeteorComponent implements OnInit {
    classes: Mongo.Cursor<any>;
    public class: Class;

    constructor() {
        super();

        this.classes = ClassesCollection.find();
    }
```

```
<div class="wrapper wrapper-content animated fadeInRight">
    <div class="row">
        <div class="col-lg-4" *ngFor='let class of classes'>
            <div class="ibox">
                <div class="ibox-title">
                    <span class="label label-primary pull-right">NEW</span>
                    <h5>Grade {{ class.grade }} {{ class.subject }}</h5>
                </div>
                <div class="ibox-content">
                    <div class="team-members">
                        <a href="#"></a>
                    </div>
```

"What Angular 1 did for the jQuery world - Meteor is doing for DB/the server side."

– Uri Goldstein

"What Rails did in the REST/stateless world - Meteor is doing in the connected world."

– Uri Goldstein