Big memory – Scale-in vs. Scale-out

Niklas Björkman
VP Technology, Starcounter

STARCOUNTER

Magic?

“Any sufficiently advanced technology is indistinguishable from magic.”

Arthur C. Clarke

STARCOUNTER

Today's topics

History

Database landscape

Scale-In instead of Scale-out

Performance on all levels

STARCOUNTER

History

Where did it start?

(Don’t worry, it’s just 2 slides)

SQL is born

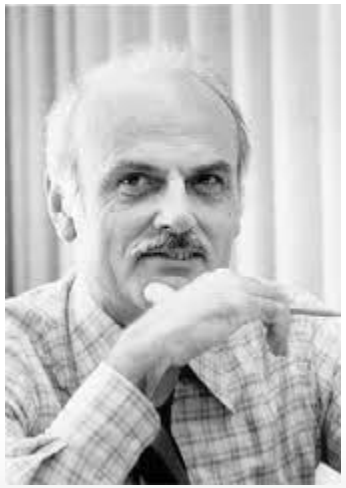
SEQUEL by Dr. Codd in 1970

IBM and Oracle (Relational System) early adopters

First relational database released in 1980

Optimizations in traditional database to optimize for disk access

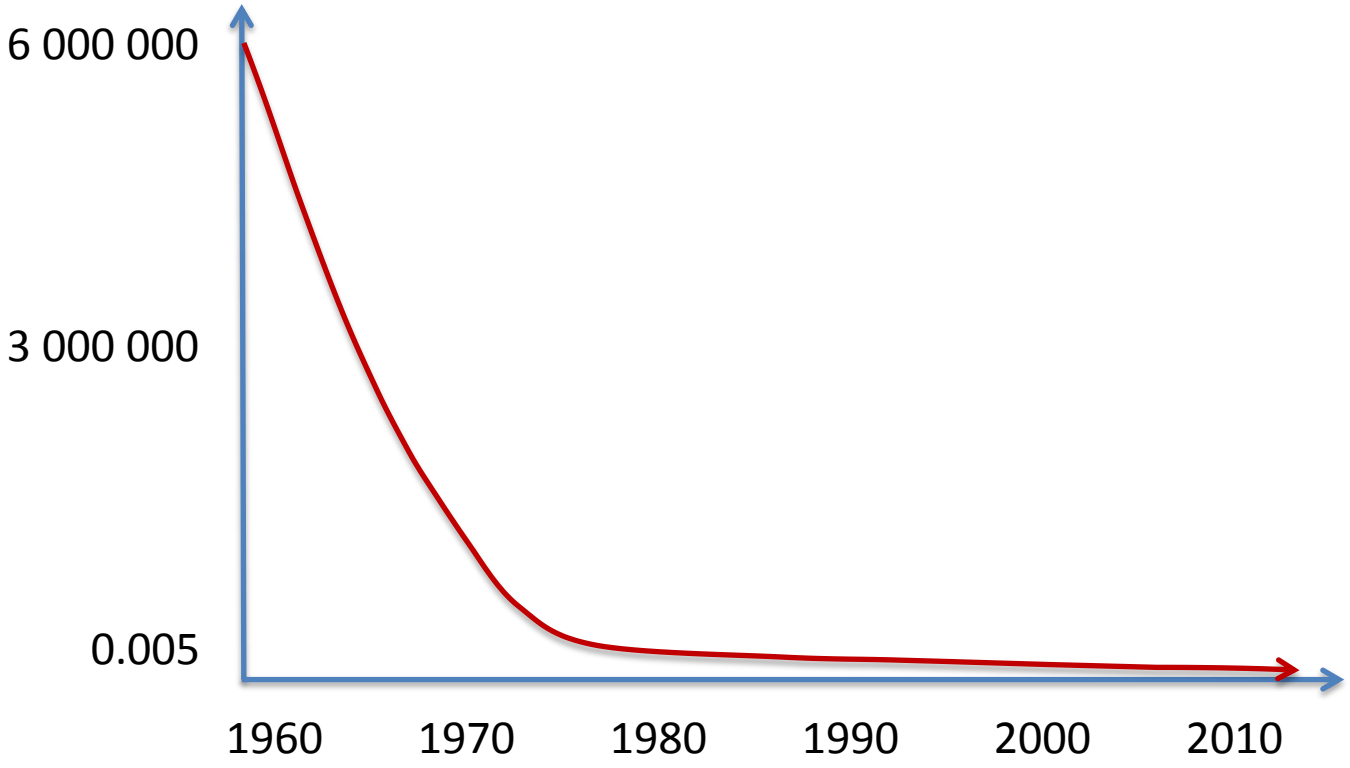
Extreme memory costs – need to store on disk



Dr Codd invented the relational data model around 1970.
"A Relational Model of Data for Large Shared Data Banks",
Communications of the ACM 13 (6):377-387, 1970

STARCOUNTER

RAM v's history



1 MB of RAM was $750,000 in 1970 compared to 0.5 cents today

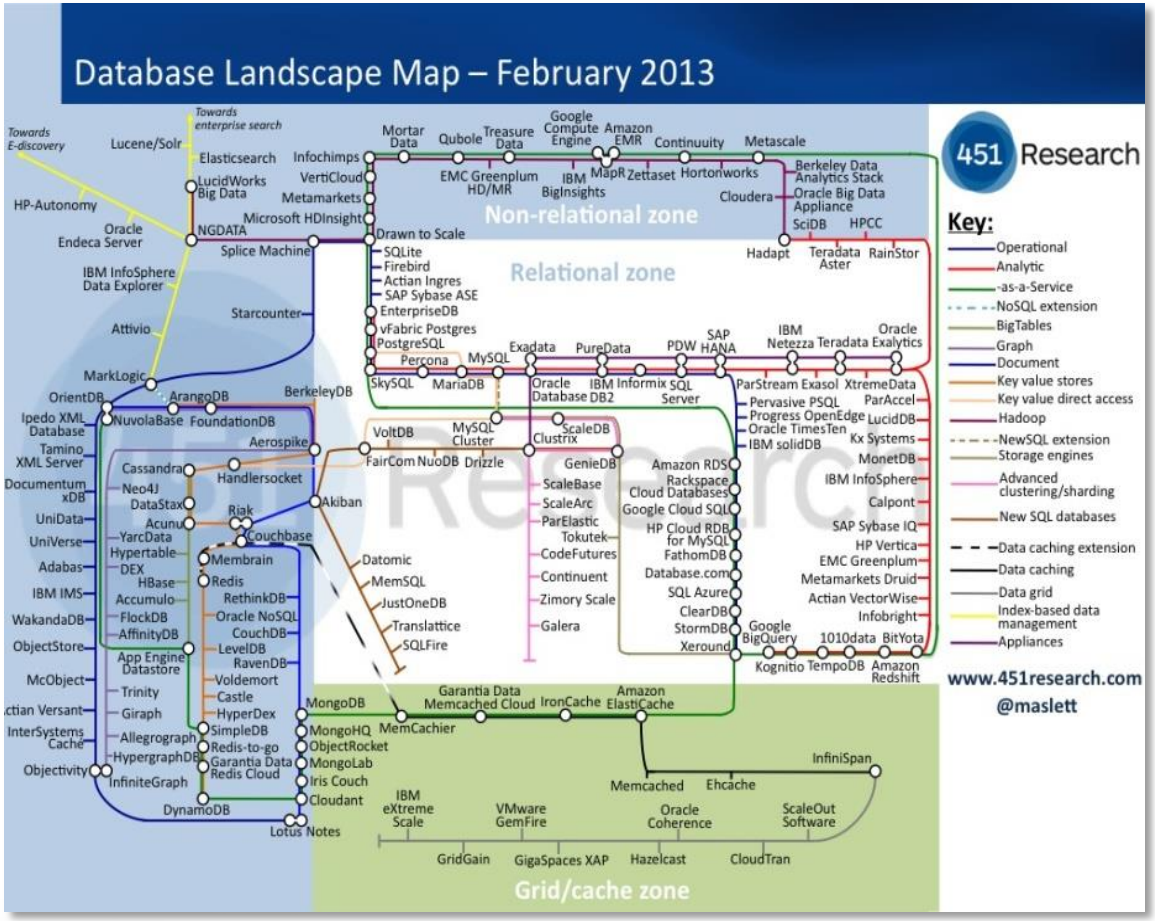
STARCOUNTER

Today

Where are we today?

STARCOUNTER

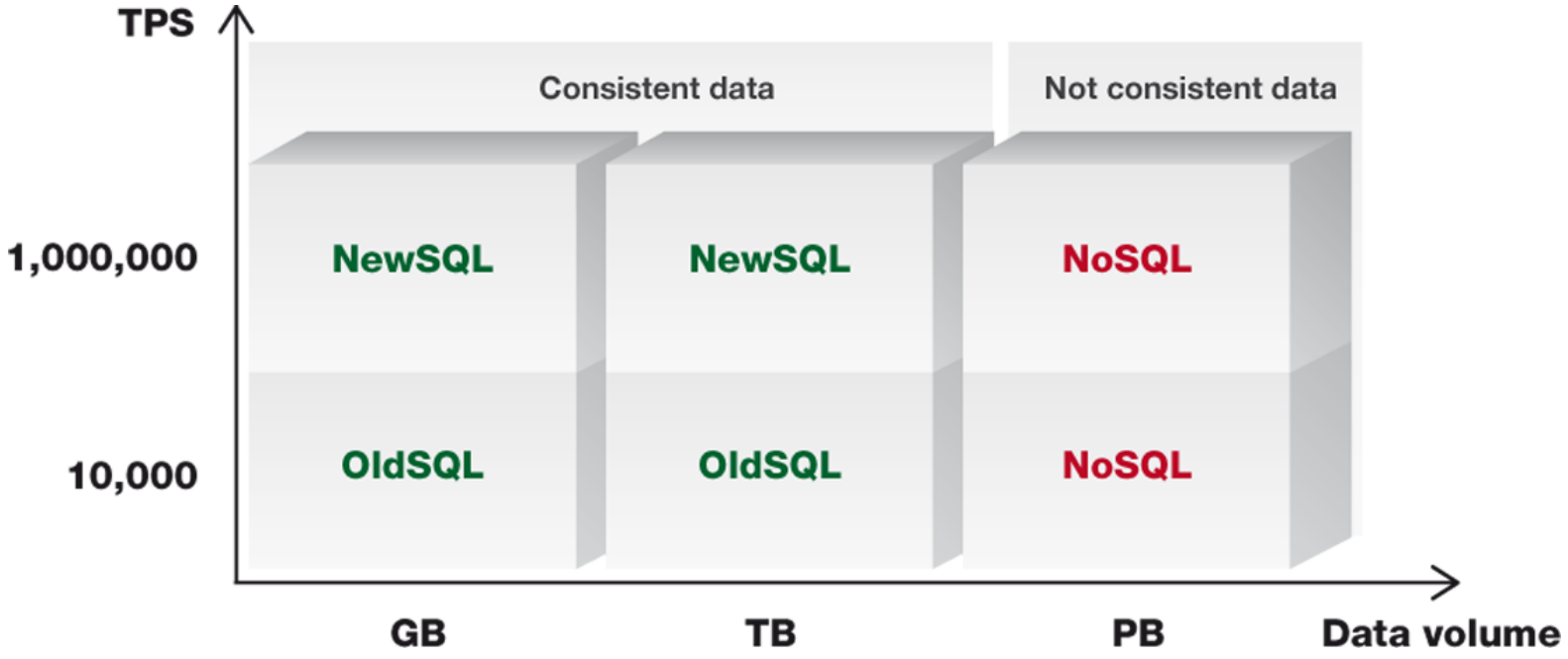
Landscape



Matt Aslett – 451 group

STARCOUNTER

Alternatives today



STARCOUNTER

Use the RAM

Scale-In instead of out

All data in one set of RAM

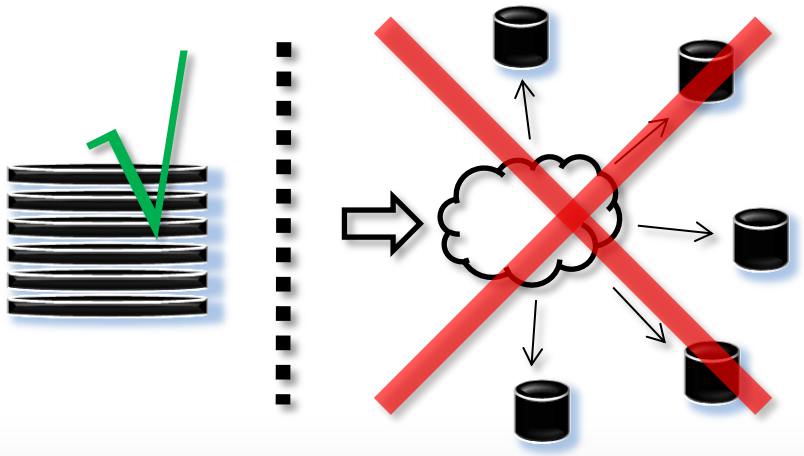
Hardware limit at 2 TB (soon > 10 TB?) of data (64bit)

Compress data about 4 times

Transaction conflicts solved fast

Extreme performance

True ACID



Scale-In and ACID

Atomicity

”C” without suffix or prefix

Isolation level like traditional databases

All writes secured on disk upon committed

CAP theorem (no ACID in scaling out)

STARCOUNTER

Who needs ACID?

Dealing with business-critical data like retail, money transfers and logistics in a multi user environment



Conflicts *will* occur and need to be managed by

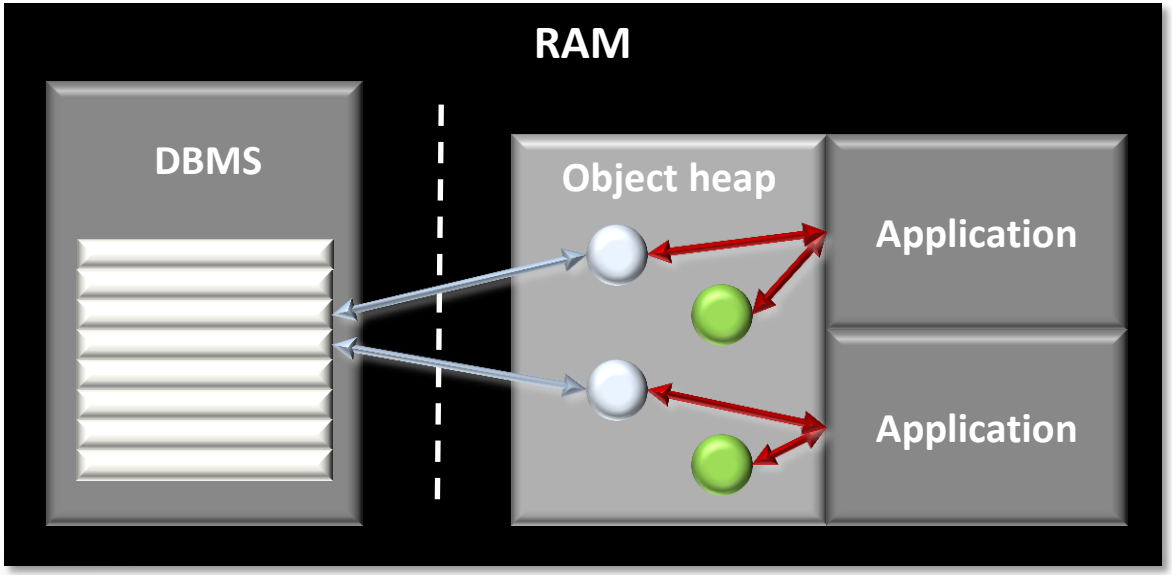
- Database
- Application *(hard for developers)*
- End user:
  "Sorry we have just sold you a product we already sold to someone else"

The extra mile

Let's see what happens if we use the RAM even further

Usually DB objects and App objects in different set of RAM

Share heap between application and database



STARCOUNTER

New opportunities

No need for separate schema

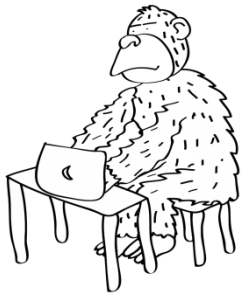
The end of ORMs

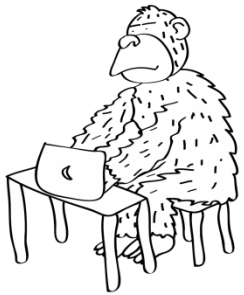
POCO objects are your database

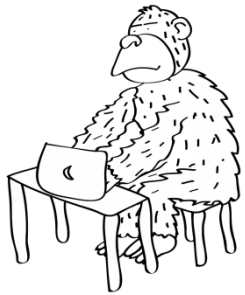
SQL directly on your POCO

Query not only persistent fields

We can simplify the use for developers







STARCOUNTER

Application programming

Traditional database	***Modern database***
Database schema (CREATE TABLE)	*Set attribute [Database] on your CLR class*
Create object (INSERT)	*Native new() operator*
Modify object (UPDATE)	*Native assignment operator (=)*
Delete object (DELETE)	*Use method Delete()*
Query objects (SELECT)	*Db.SQL("SQL92")*

STARCOUNTER

Database schema

```
[Database]
public class Employee
{
    public string Name;
    public DateTime? HireDate;
    public decimal Salary;
    public Department Department;
    public DateTime BirthDate;

    public int Age
    {
       get
       {
         return DateTime.Now.Year-BirthDate.Year;
       }
    }
}
```

STARCOUNTER

Create object

```
[Database]
public class Employee
{
    public Employee() { }
}


Employee e = new Employee();
```

Modify object

```
Department d = new Department();
Employee e = new Employee();

e.Name = ”John”;
e.HireDate = null;
e.Salary = 20000;
e.Department = d;
```

Delete object

```
Department d = new Department();
Employee e = new Employee();
e.Name = ”John”;
e.HireDate = null;
e.Salary = 20000;
e.Department = d;

e.Delete();
```

STARCOUNTER

Transactions

Transaction scopes

```
Db.Transaction(()=>
{
      Person p = new Person();
      p.Name = ”Albert”;
}
```

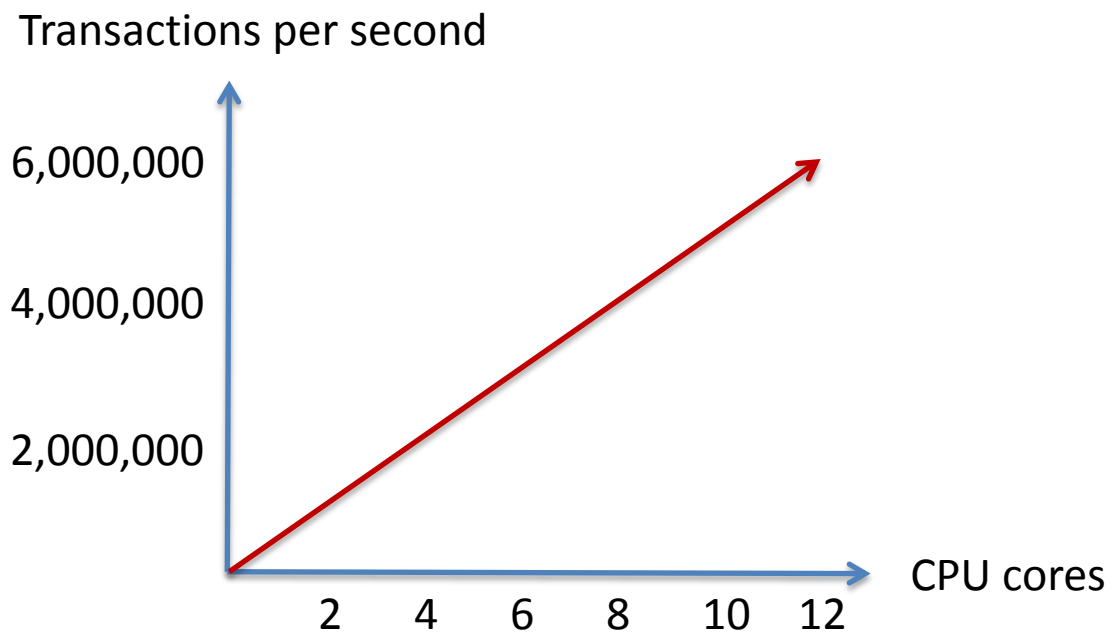
Long lived transactions

```
Transaction t = new Transaction();
...
t.Commit(); // t. Rollback();
```

Parallel transactions

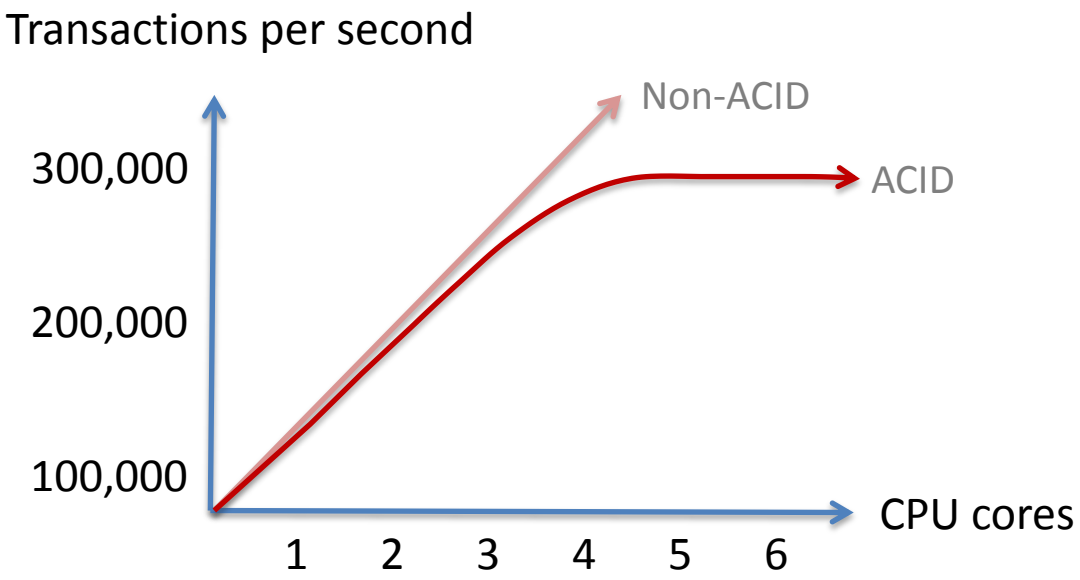
Performance - reading

If used wisely we can scale read transactions linearly over the number of cores



Performance - writing

Depending of the number of transactional conflicts, writes will level out at a certain level (ACID). Still 100 times faster than traditional relational databases.



STARCOUNTER

Next steps

Is the future here already?

STARCOUNTER

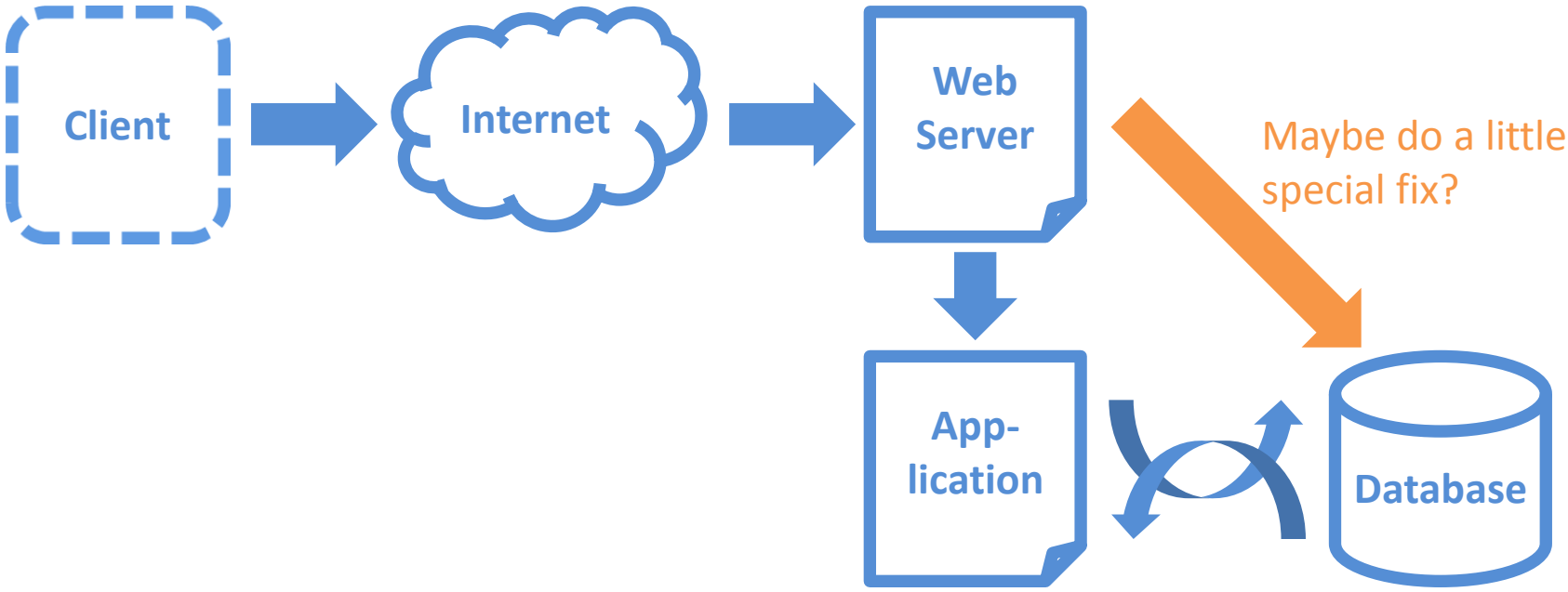
What's next?

Today we have databases with
- Extreme performance
- Reliability (ACID)
- Easy to use API:s

Logical next steps
- Easy to use connectivity
- Super fast communication servers
- Easy to use in modern applications

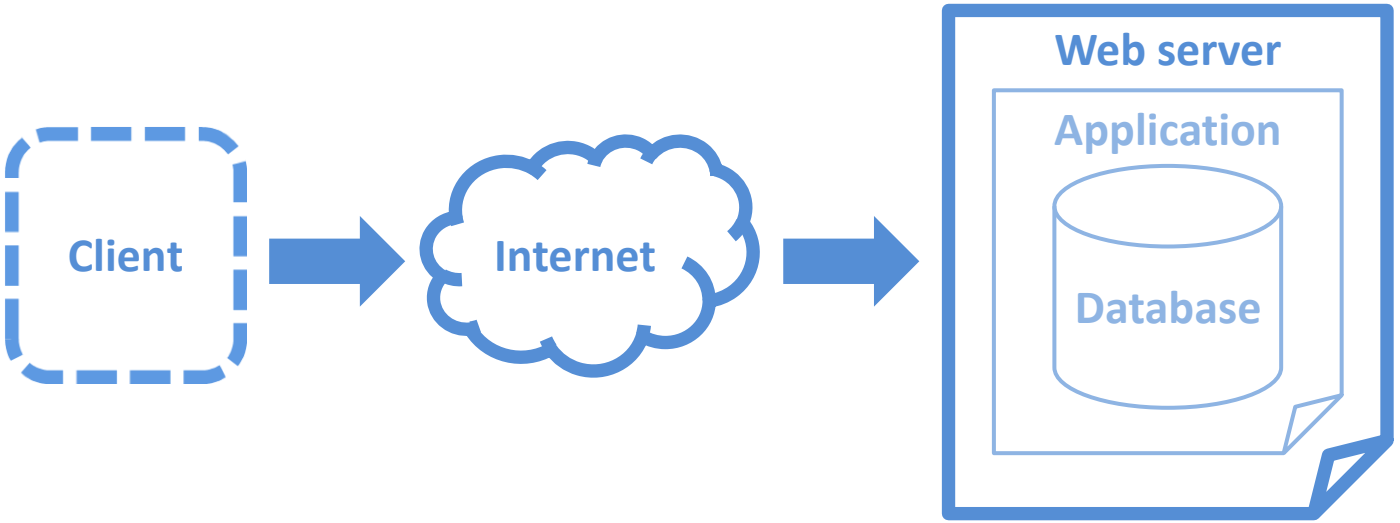
Communication Performance

A normal setup for a web based application

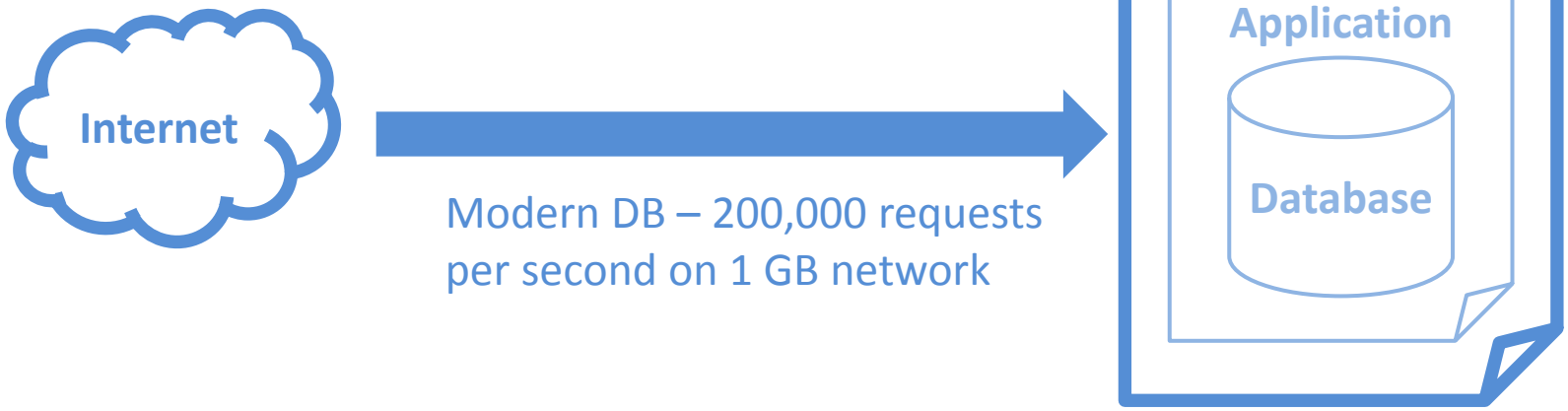


Communication Performance

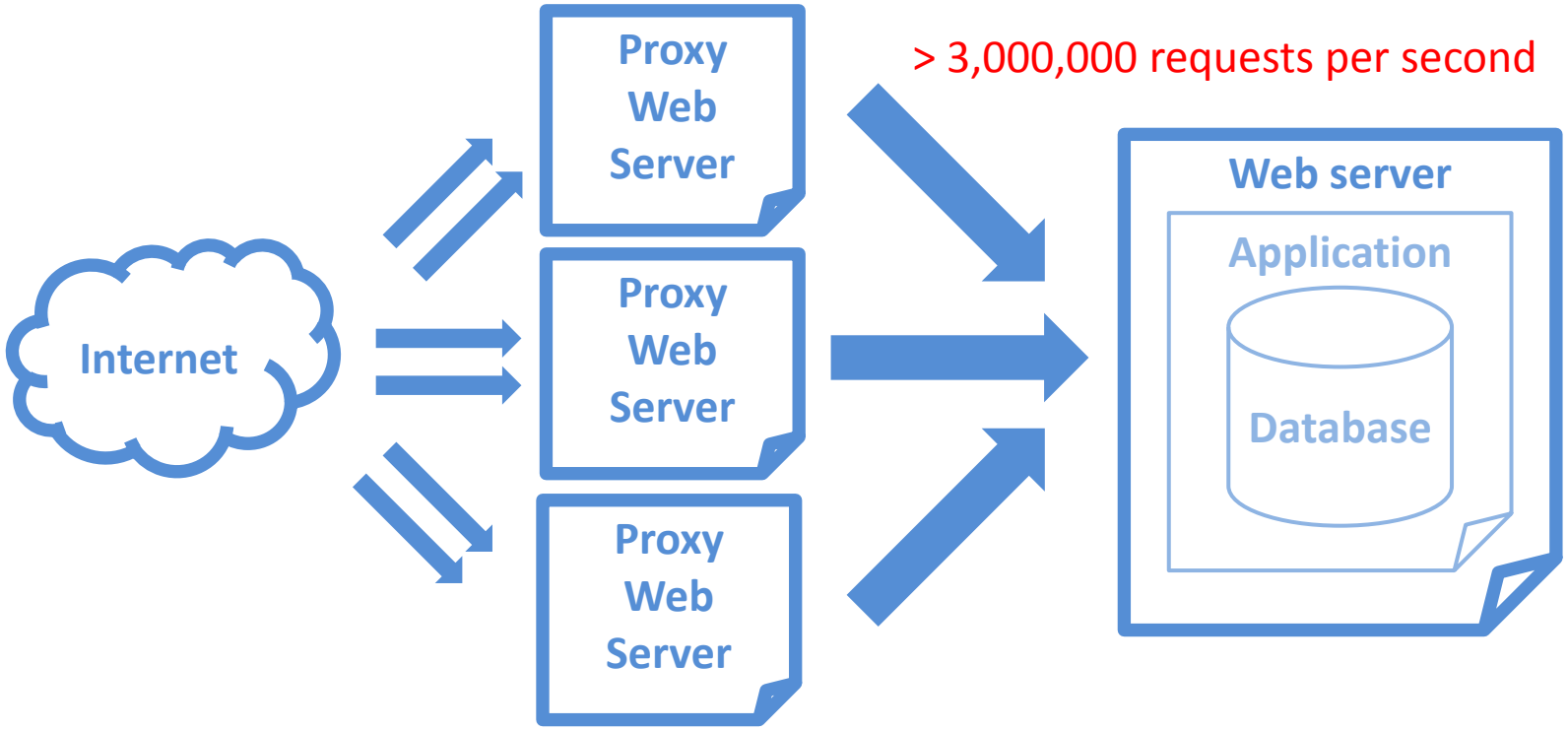
Tie the web server, application and database closer together



Limitation - Network/OS



Solution - Network/OS



STARCOUNTER

Development performance

Modern applications with web standards

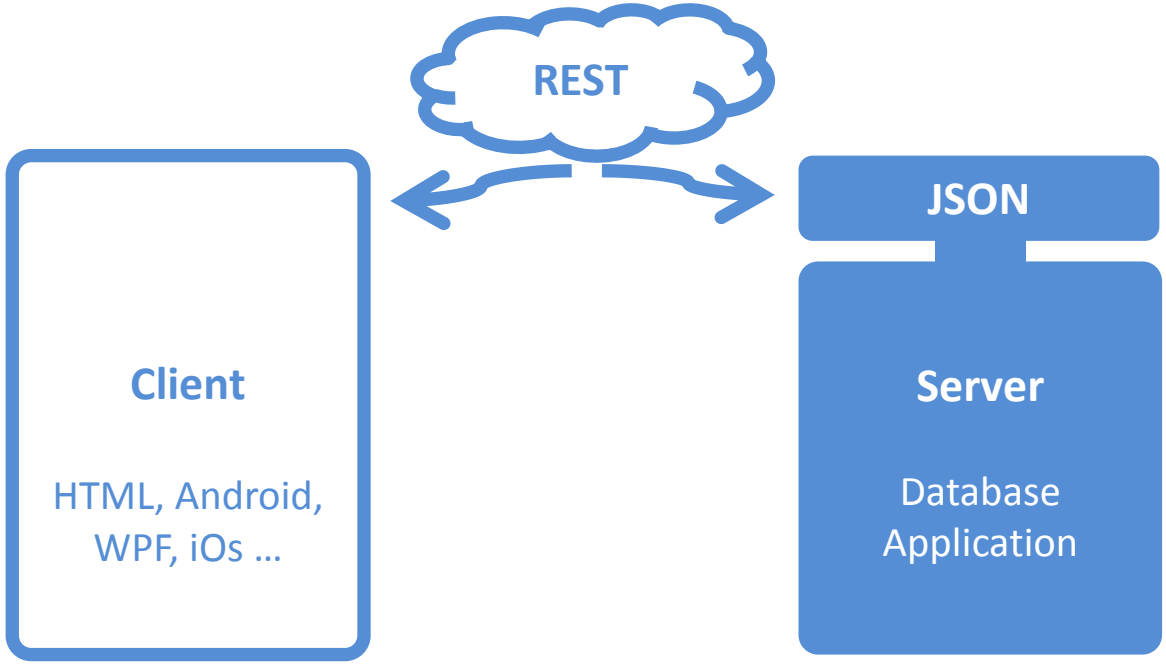
- REST/JSON
- JSON Patch (RFC 6902)
- WebSockets

Modern Interactive applications using

- MVVM
- WebComponents
- AngularJs

Modern standards

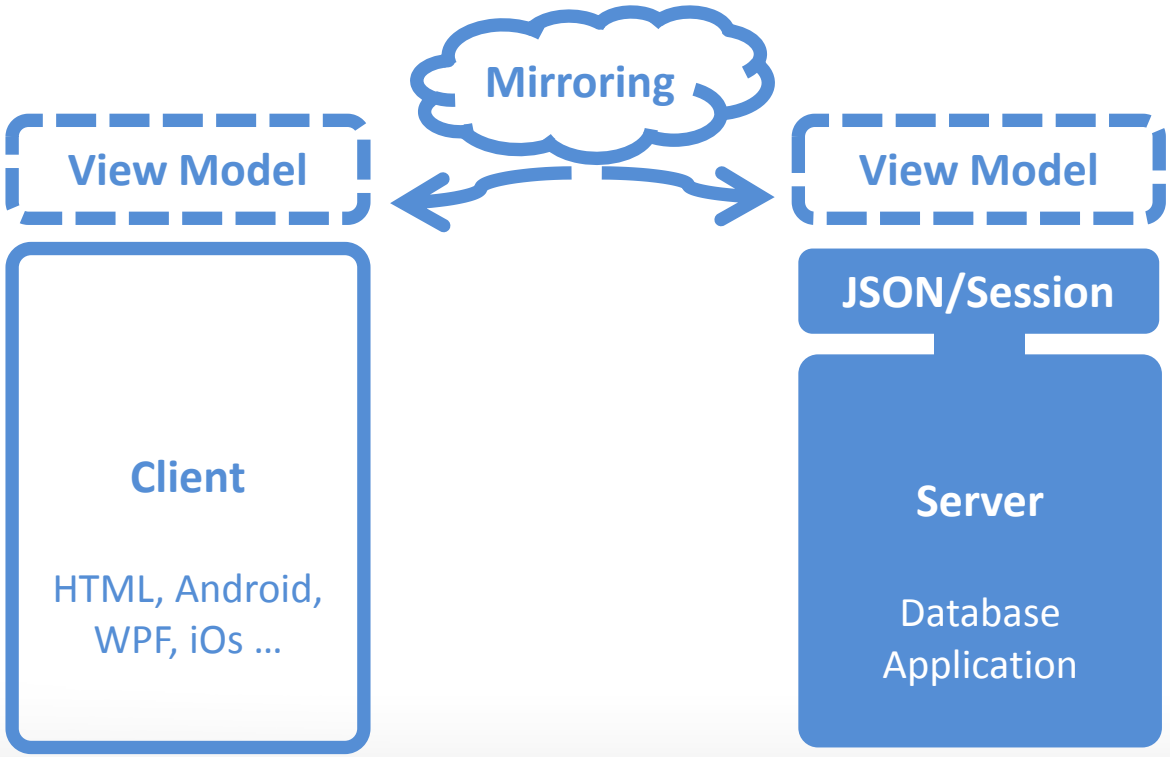
Easy to use REST API



Trends

The trend is that we move away from server side rendering

Data is fetched from server - on the fly or mirrored MVVM



STARCOUNTER

JSON models

Simple JSON model

```
“FirstName$”:"Albert",
“LastName$”:"Einstein",
“Quotes”: [
     { Text:"This is an example" }
]
```

JSON model automatically bound to persistent data

```
PersonModel model = new PersonModel();
model.Data = new Person();
```

STARCOUNTER

REST verbs

Handle REST verbs server side

```
Handle.GET("/new-person", () =>
{
    PersonModel c = new PersonModel();
    Person p = new Person();
    p.FirstName = ”Albert”;
    p.LastName = ”Einstein”;
    c.Data = p;
    return c;
});
```

STARCOUNTER

REST with body

Complete model in body

```
Handle.POST("/new-person-wModel", (PersonModel model) =>
{
    Person comp = new Person();
    comp.FirstName = model.FirstName;
    comp.LastName = model.LastName;
});
```

Controller

Handle client modifications on the server

```
class PersonModel : Json {

    void Handle( Input.FirstName input ) {
        if (input.Value == ”Albert”) {
            Message = ”Not accepted value”;
            input.Cancel();
        }
    }
}
```

STARCOUNTER

Binding

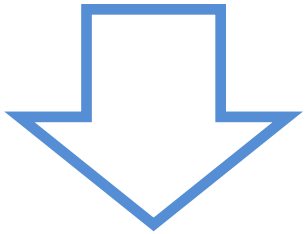
Using declarative programming and binding allows automated updates

```
FirstName:"Albert",
LastName:"Einstein",
Quotes: [
   { Text:"This is an example" }
]
```

```
public class Person
{
     public String FirstName;
     public String LastName;
     public IEnumerable<Quote> Quotes
     {
          get
          {
               return Db.SQL<Quote>(
                    "SELECT q FROM Quote WHERE q.Who=?", this);
          }
     }
}
```

Binding – automatic updates

```
void Handle(Input.Text input) {
     new Quote(){
          Text = input.Value,
          Who = Data
     };
}
```

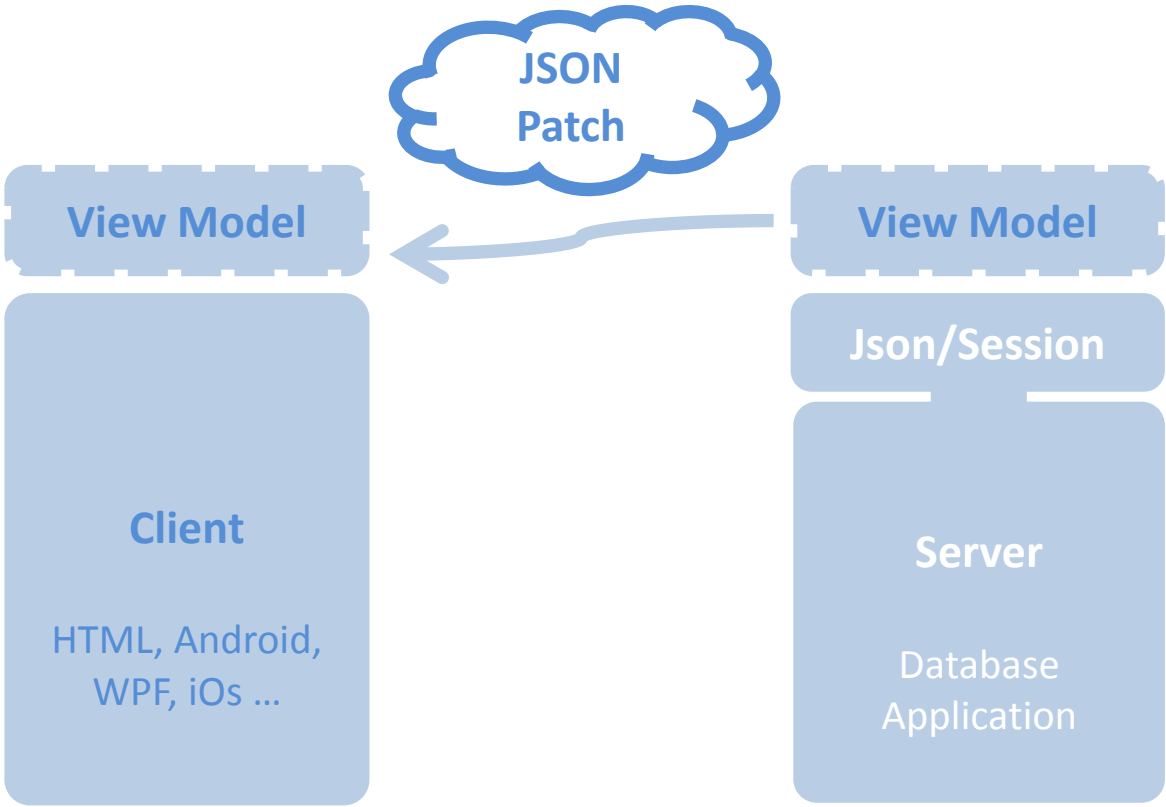


The ”Quotes” property will change in class Person

```
public IEnumerable<Quote> Quotes
{
     get
     {
          return Db.SQL<Quote>(
               “SELECT q FROM Quote WHERE q.Who=?”, this);
     }
}
```

STARCOUNTER

Delta sent to client



STARCOUNTER

Example setup MVVM

Client

```
<label>
     {{item.FirstName$}}
</label>
```

JSON patch

Server

```
{
     “FirstName$”:"Albert",
}
```

Controller

```
void Handle( Input.FirstName input ) {
     if (input.Value == ”John”)
      {
          Message = ”Not accepted value”;
          input.Cancel();
      }
  }
```

DB

```
public class Person{
     public String FirstName;
}
```

STARCOUNTER

Super fast, super easy

Super fast

- Database core
- Application
- Communication server

Super easy

- Database API
- Client-Server API
- To maintain
- *Less lines of code*

As ABBA would say

Money, money, money

Save money on

- Hardware
- Shorter time to market
- Maintenance
- Fewer developers
- Faster to learn
- Shorter time to market
- Less DBA costs

STARCOUNTER

Summary

History

Database landscape

Scale-In instead of Scale-out

Performance on all levels

Easy to use

Magic

STARCOUNTER

Thanks for listening!