

Heresies and Dogmas in Software Development

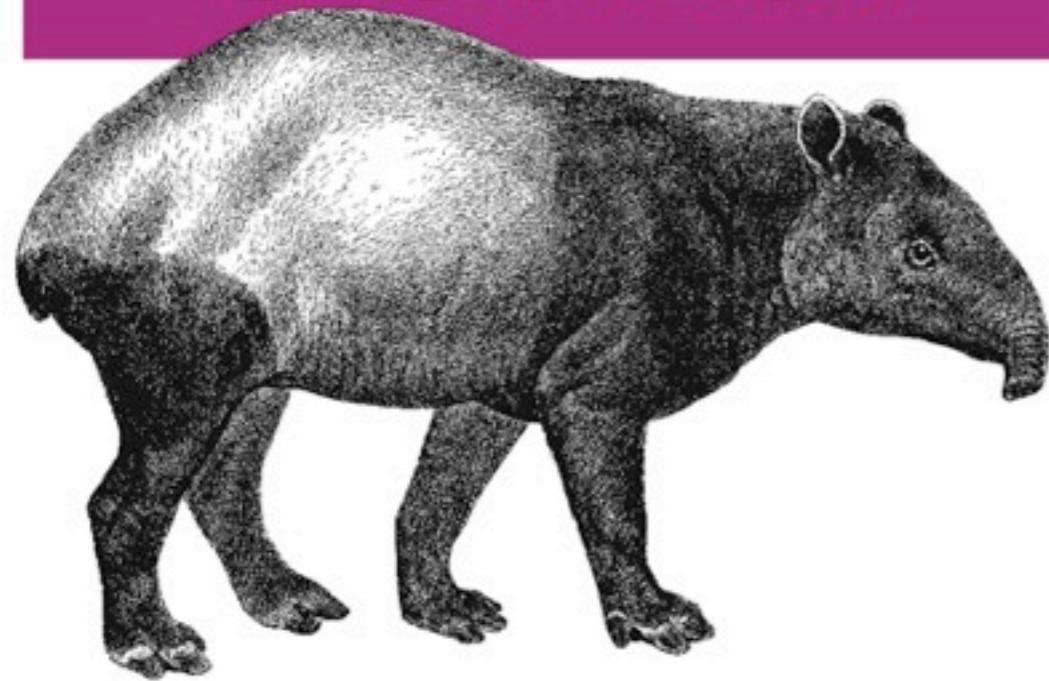
@deanwampler

StrangeLoop 2011

Scalability = Functional Programming + Objects

Programming

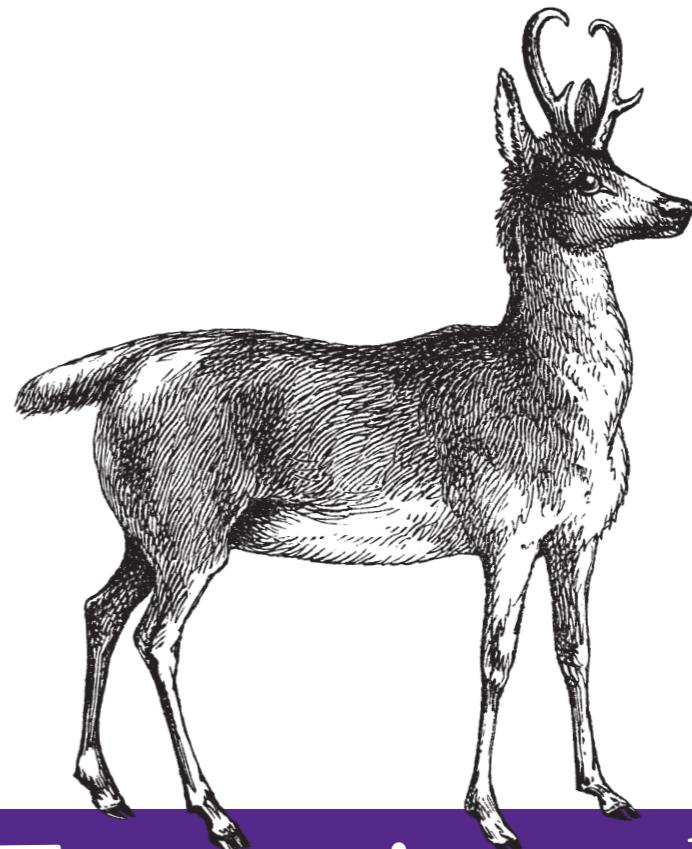
Scala



O'REILLY®

Dean Wampler & Alex Payne

programmingscala.com



Functional Programming

for Java Developers

O'REILLY®

Dean Wampler

[polyglotprogramming.com
/fpjava](http://polyglotprogramming.com/fpjava)



@stesla

Samuel Tesla

So often I find myself wondering how many things in software we actually *know* and how many we just *believe*. Software is faith-based.

31 Mar 10 via Echofon Favorite Retweet Reply

<https://twitter.com/#!/stesla/status/11390744100>

Goto
(Considered
Harmful)





@bpettichord

Bret Pettichord

Both my parents were programmers. As a teenager, to be rebellious, I insisted that "goto" wasn't harmful. True story.

6 Mar 10 via TweetDeck Unfavorite Retweet Reply

Retweeted by [MaggieL](#) and 6 others



<http://twitter.com/bpettichord/status/10062856309>

The Goto

A non-local jump, often to a label

```
while (true) {  
    doSomeWork();  
    if (hasMoreWork() == false)  
        goto finished;  
    wait(1000);  
}  
label finished;
```

“Go To Statement Considered Harmful”

Edsger Dijkstra, Communications of the ACM 11 (3):
147–148 (March 1968).

“Go To Statement Considered Harmful”

- Complicates analysis and verification of program correctness, especially loops.

“Go To Statement Considered Harmful”

- Structured Programming replaces gotos with:
 - Sequence (i.e., sequential instructions)
 - Repetition (e.g., loops)
 - Selection (e.g., branches)

“Structured Programming with Go To Statements”

Donald Knuth,
Computing Surveys 6 (4): 261–301 (1974).

“Structured Programming with Go To Statements”

- Programmers found it difficult to eliminate gotos.

“Structured Programming with Go To Statements”

- Some code constructs are actually simpler to understand with gotos.
- breaking out of loops.

“Structured Programming with Go To Statements”

- Some code with gotos was noticeably faster.

Even Linus Torvalds has
defended gotos.

<http://kerneltrap.org/node/553>

Whither Gotos?

Heresy or Dogma?

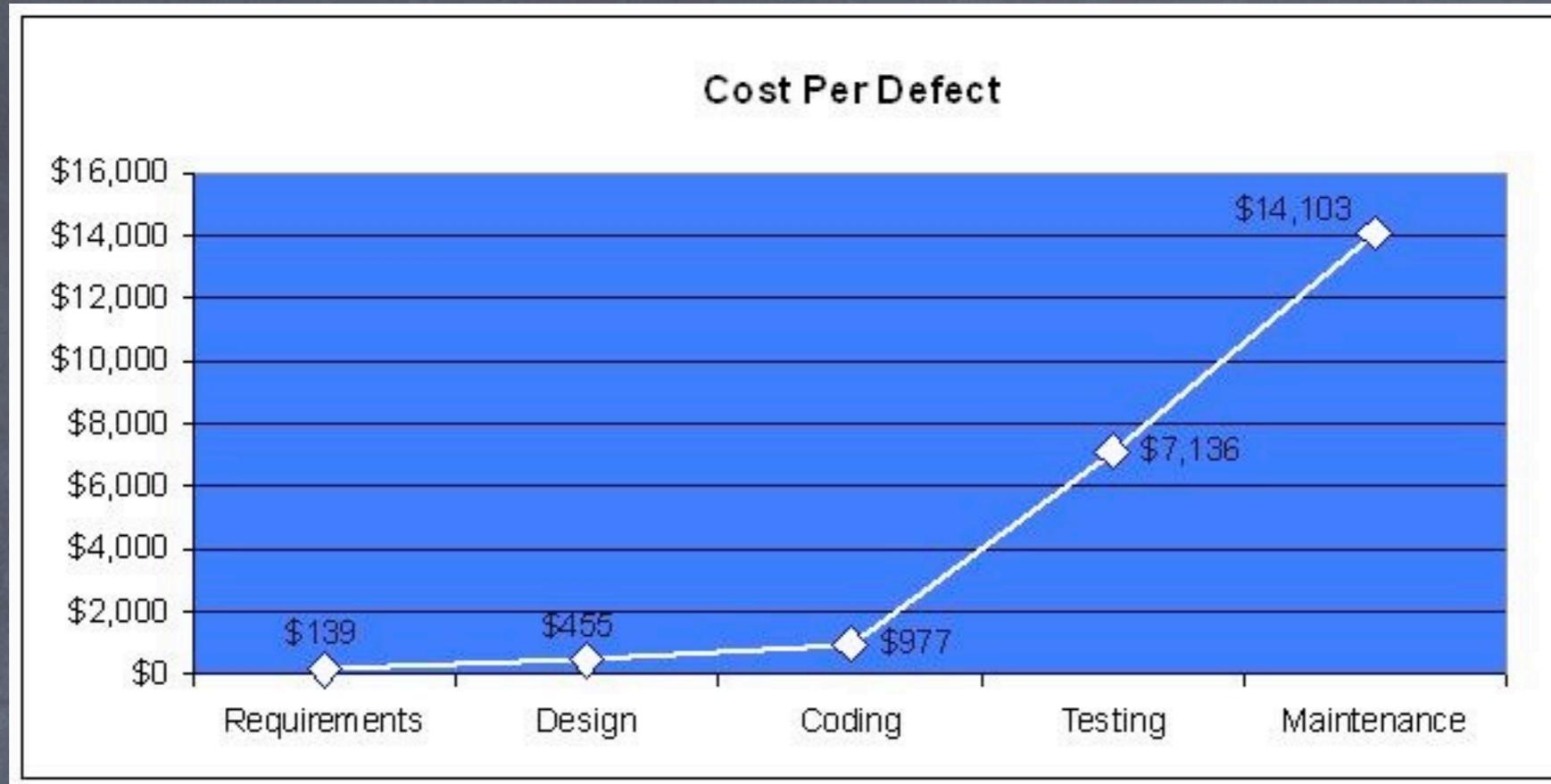
Whither Gotos?

- Can lead to spaghetti code.
- Can also lead to fast, intuitive code.
- Constructs like break are rebranded, constrained gotos.



Design
Before
Code

Wait! That building is
supposed to be square!



???

Capers Jones, Software Assessments, Benchmarks, and Best Practices, Addison-Wesley, 2000

If rework is expensive,
can we eliminate it by
deciding exactly what to
code before we code it?

Agile Taught Us:

- Requirements change is inevitable.
- We learn the requirements while building.

Agile Taught Us:

- Reducing the cost of change to near zero lets us defer decisions to the last responsible moment.

Agile Taught Us:

- Iterations eliminate risk in small chunks.

Design Before Code

Heresy or Dogma?

Design Before Code

Even building
construction is an
adaptive process.

Design Before Code

Since software is
virtual,
it is even more
adaptable.

The background image shows a modern architectural space with a curved, light-colored wall composed of many rectangular panels. In front of the wall, there are several rows of red theater-style seats arranged in a semi-circular pattern, facing towards the center of the curve.

Clojure
(Just Kidding)

Design Patterns



“A solution
to a problem
in a context.”

Obviously good, right?

"Are Design Patterns Missing Language Features?"

[http://www.c2.com/cgi/wiki?
AreDesignPatternsMissingLanguageFeatures](http://www.c2.com/cgi/wiki?AreDesignPatternsMissingLanguageFeatures)

“Design Patterns in Dynamic Languages”

Peter Norvig,

<http://norvig.com/design-patterns/>

“Design Patterns in Dynamic Languages”

“16 of the 23 patterns in Design Patterns were ‘invisible or simpler’ in Lisp.”

Some patterns are
language features in
functional languages.

Iterator, Composite,
Command...

Other patterns are
(fortunately) eliminated.

Visitor

Functional programming
has its own patterns.

Fold, Monoid, Monad,
Iteratee, Applicative...

Design Patterns

Heresy or Dogma?

Design Patterns

The concept of
patterns remains useful.

Specific examples
come and go.

CORBA vs. REST



CORBA

- Binary encoding.
- Object method calls are the protocol.

REST (HTTP)

- Text, platform neutral encoding.
- URLs and HTTP are the protocol.

CORBA's Flaws

- ⦿ Every version change forced a global upgrade.
- ⦿ Binary change.
- ⦿ Interface instability.

Interface Instability

Objects are not
very modular.

Modularity

interface	Single responsibility, clear abstraction, hides internals
composable	Easily combines with other modules to build up behavior
reusable	Can be reused in many contexts

Modularity

- Two successful modularity schemes:
 - Digital circuits.
 - HTTP.

Digital Circuits

- Each wire: 0 or 1
- 32 together: 4 Billion unique values!

HTTP

- 9 “Request Methods”
 - GET, POST, HEAD, OPTIONS, ...
- Text Oriented
- Key-Value header fields.
- Payload encoding - MIME type.

Reuse

- Simple abstractions.
- Low-level of abstraction.
- Enabled higher-level abstractions.

Paradox of Objects

Unconstrained freedom
to create abstractions
undermines reuse.

Paradox of Objects

Abstraction boundary is
too high, without a
lower-level boundary.

CORBA vs. REST

Heresy or Dogma?

CORBA vs. REST

- REST/HTTP meets requirements for modularity.
- Low-level, simple abstraction.
- Minimal coupling.
- The constraints enable reuse.

CORBA vs. REST

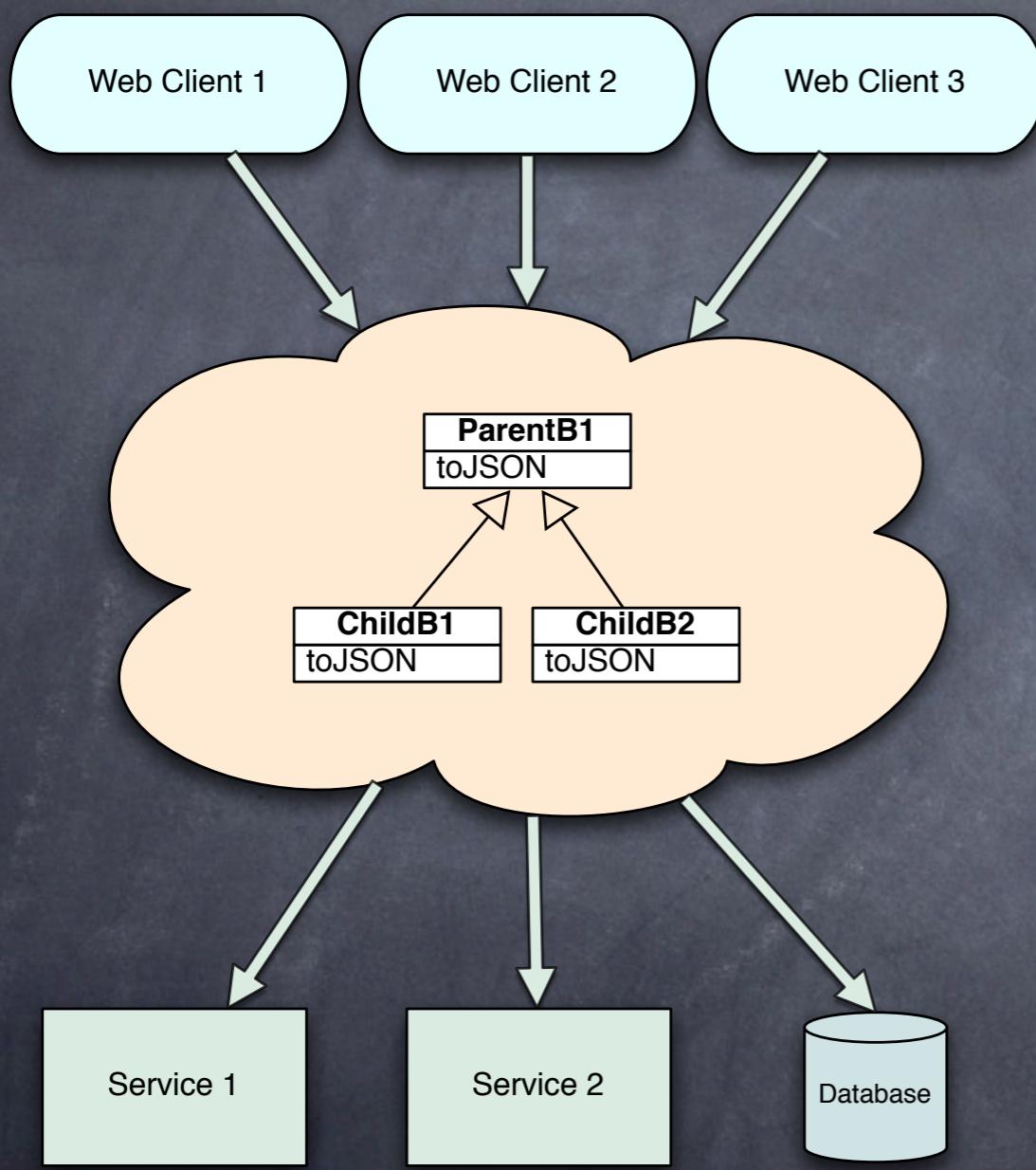
- ⦿ CORBA isn't modular.
- ⦿ High-level, ad-hoc abstractions.
- ⦿ Maximal coupling.

Object Middleware and ORMs

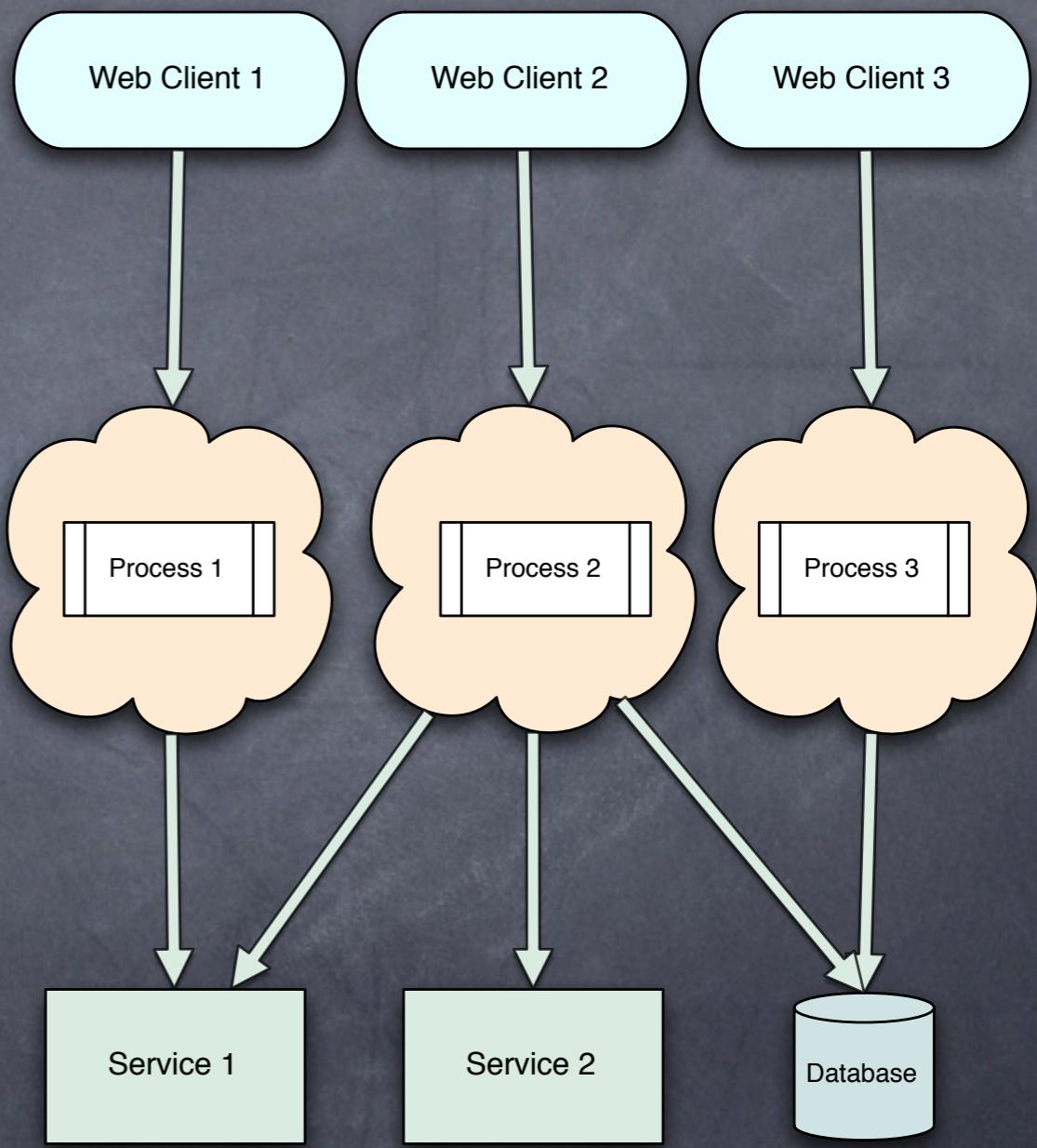


In a highly concurrent
world, do we really want
a middle?

Which Scales Better?



or



Implementing a
rich domain model
encourages
fewer, fatter services.

Object-Relational Mapping

ORM Pros

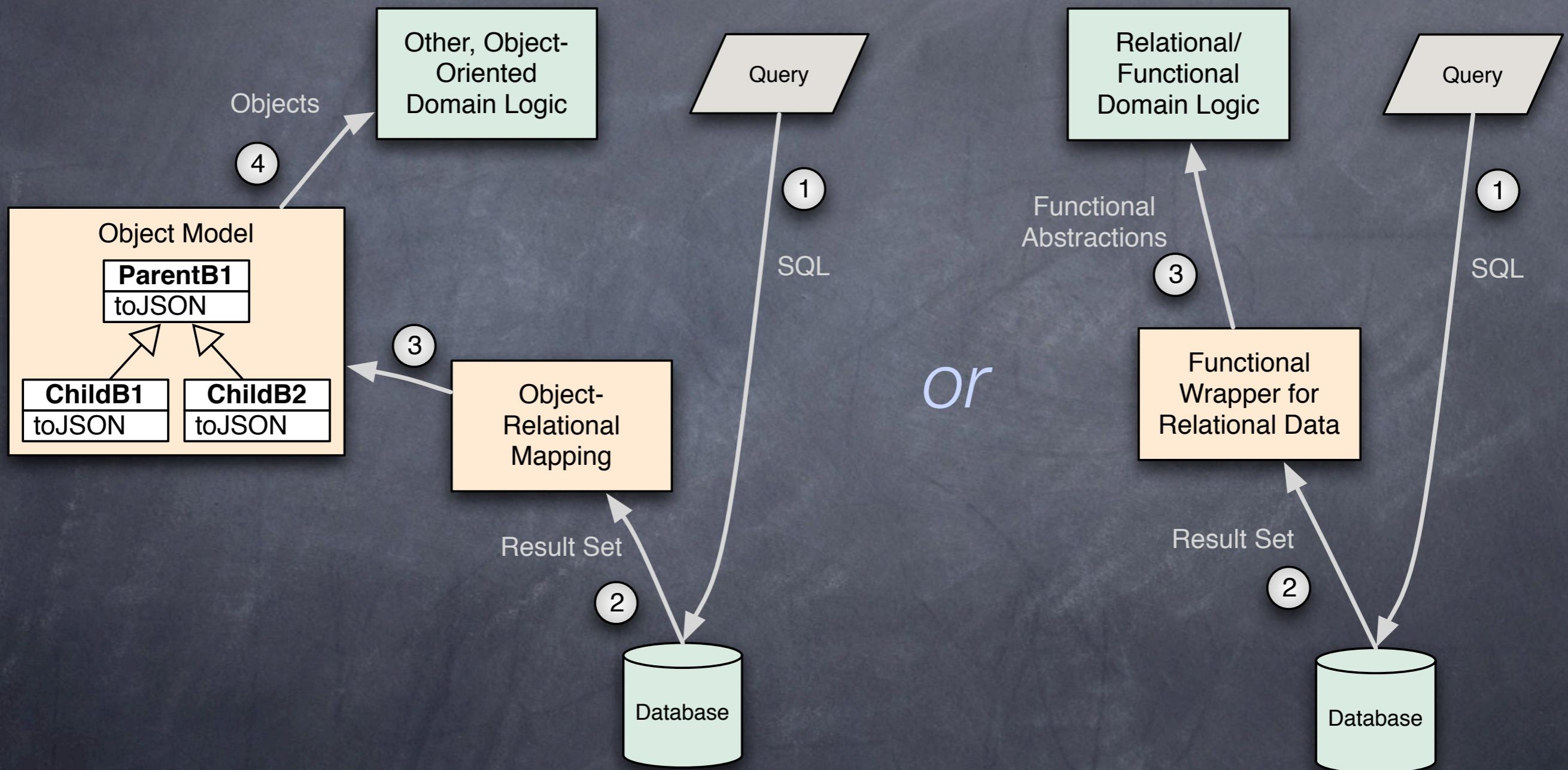
- Mostly eliminate the need for SQL.
- Generate boilerplate code.
- Inefficient, but “good enough”.

ORM Cons

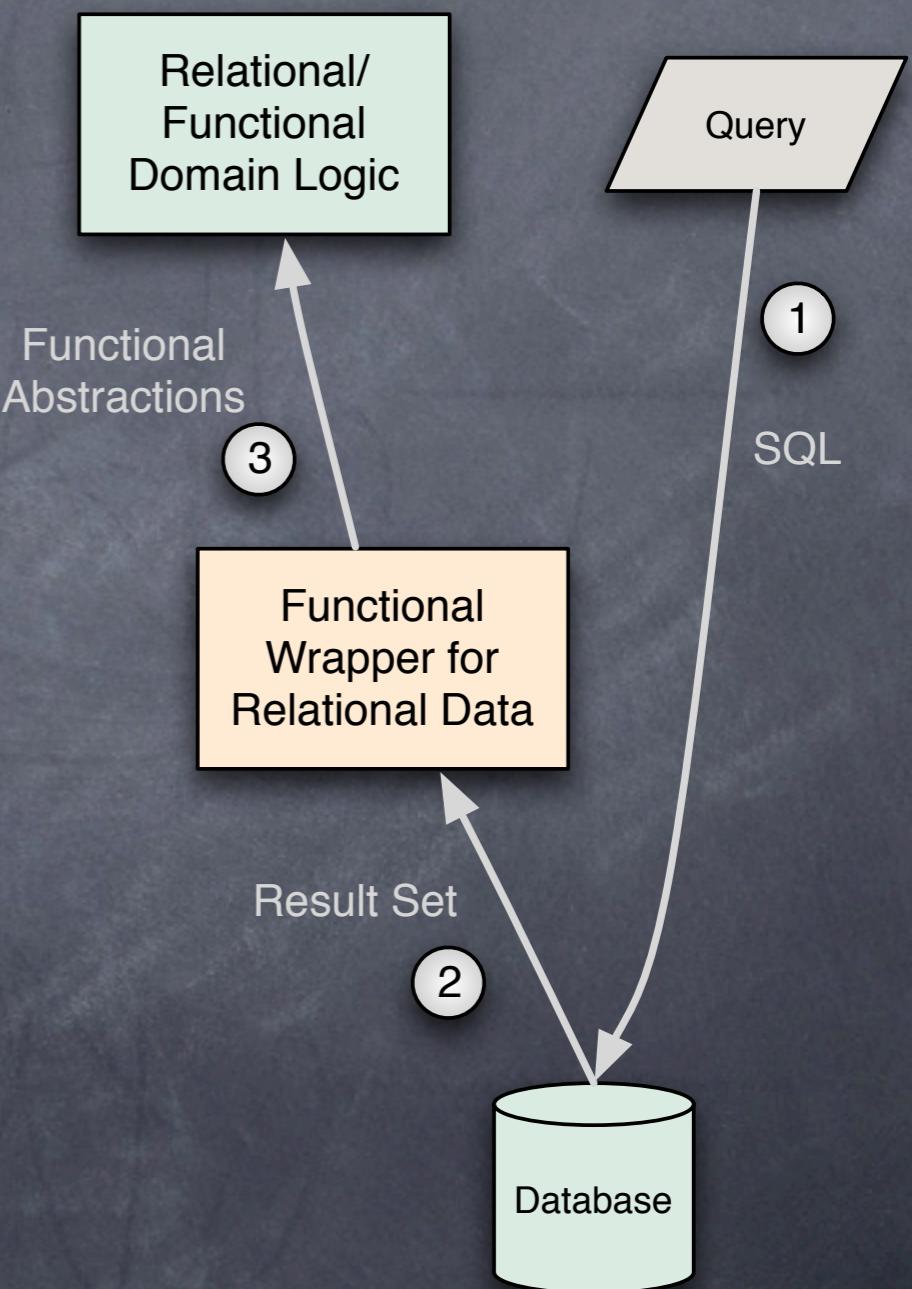
- Poor abstraction - don't eliminate SQL.
- Objects are a poor fit for relational data.
- Not really efficient enough, especially for "big data".

http://seldo.com/weblog/2011/08/11/orm_is_an_antipattern

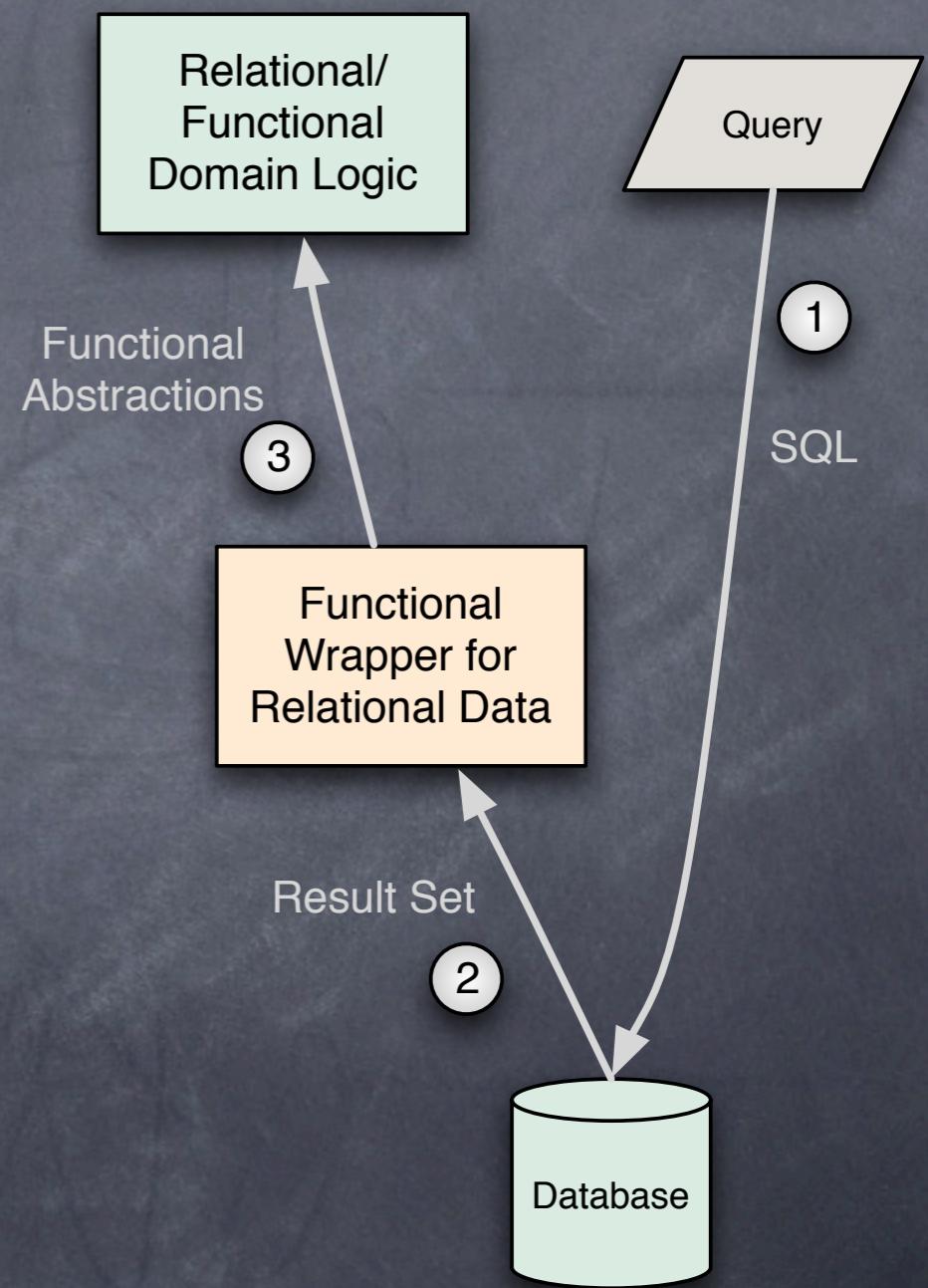
Which Is Simpler?



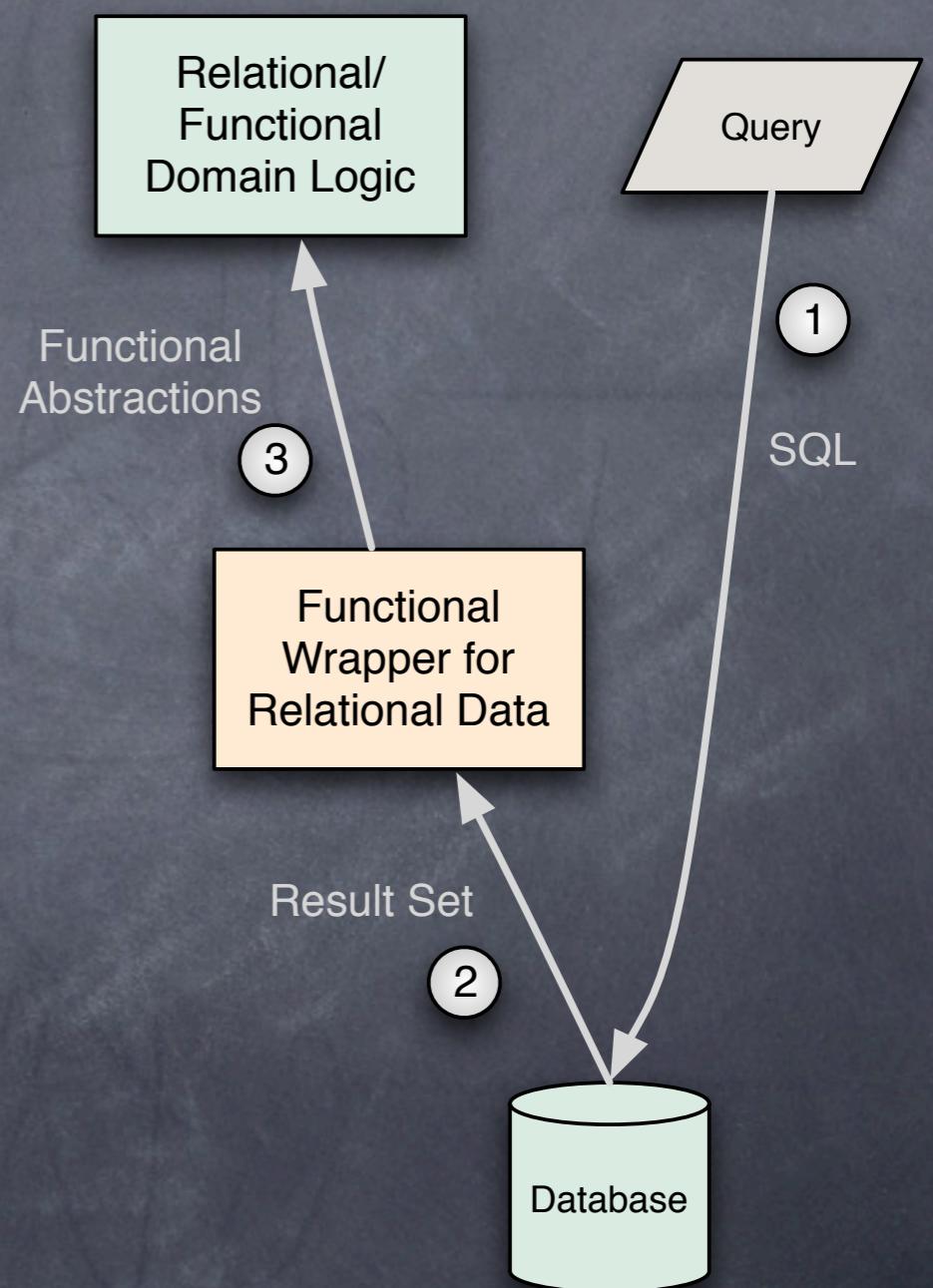
Functional data structures fit Relational data.



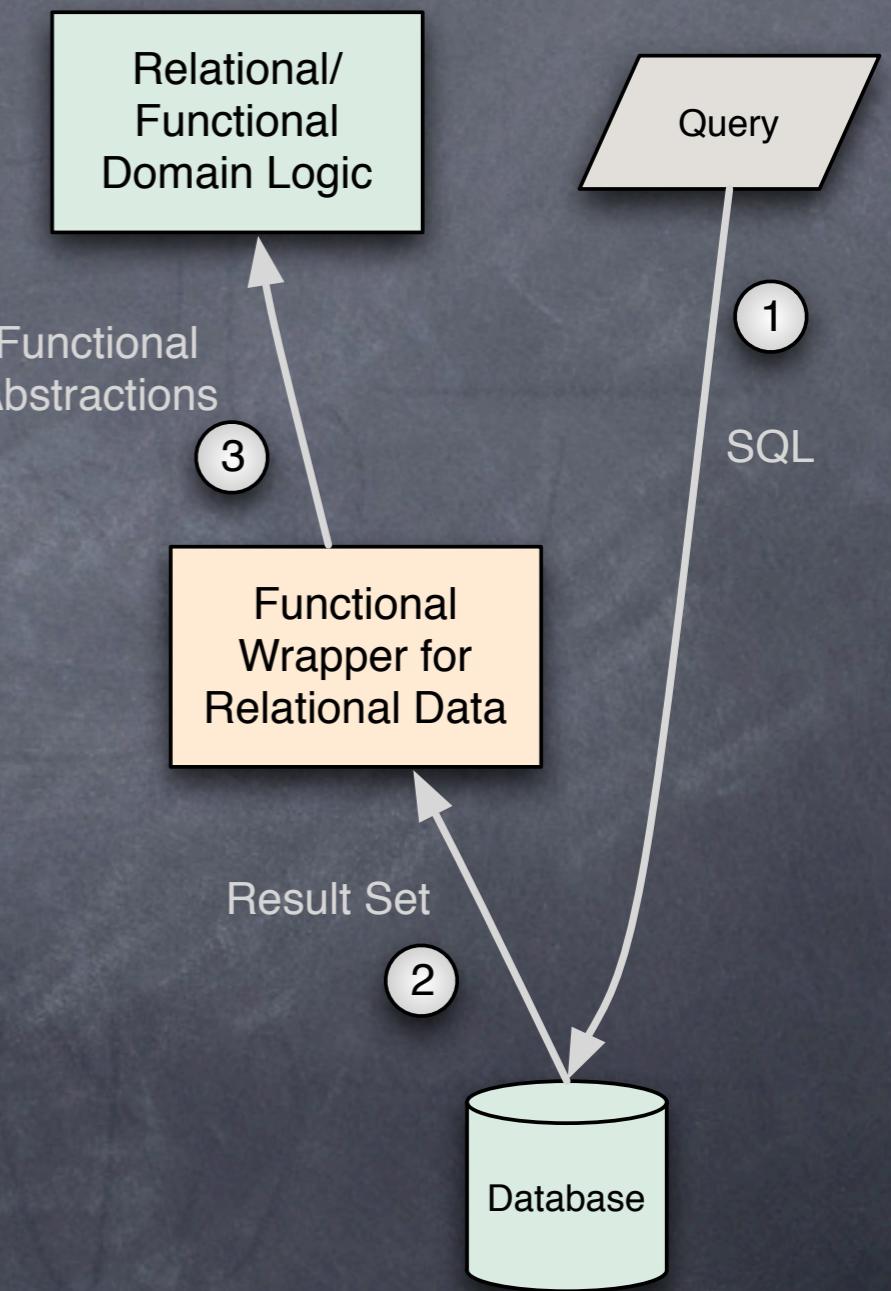
LINQ and
similar tools
minimize the
object-relational
impedance.



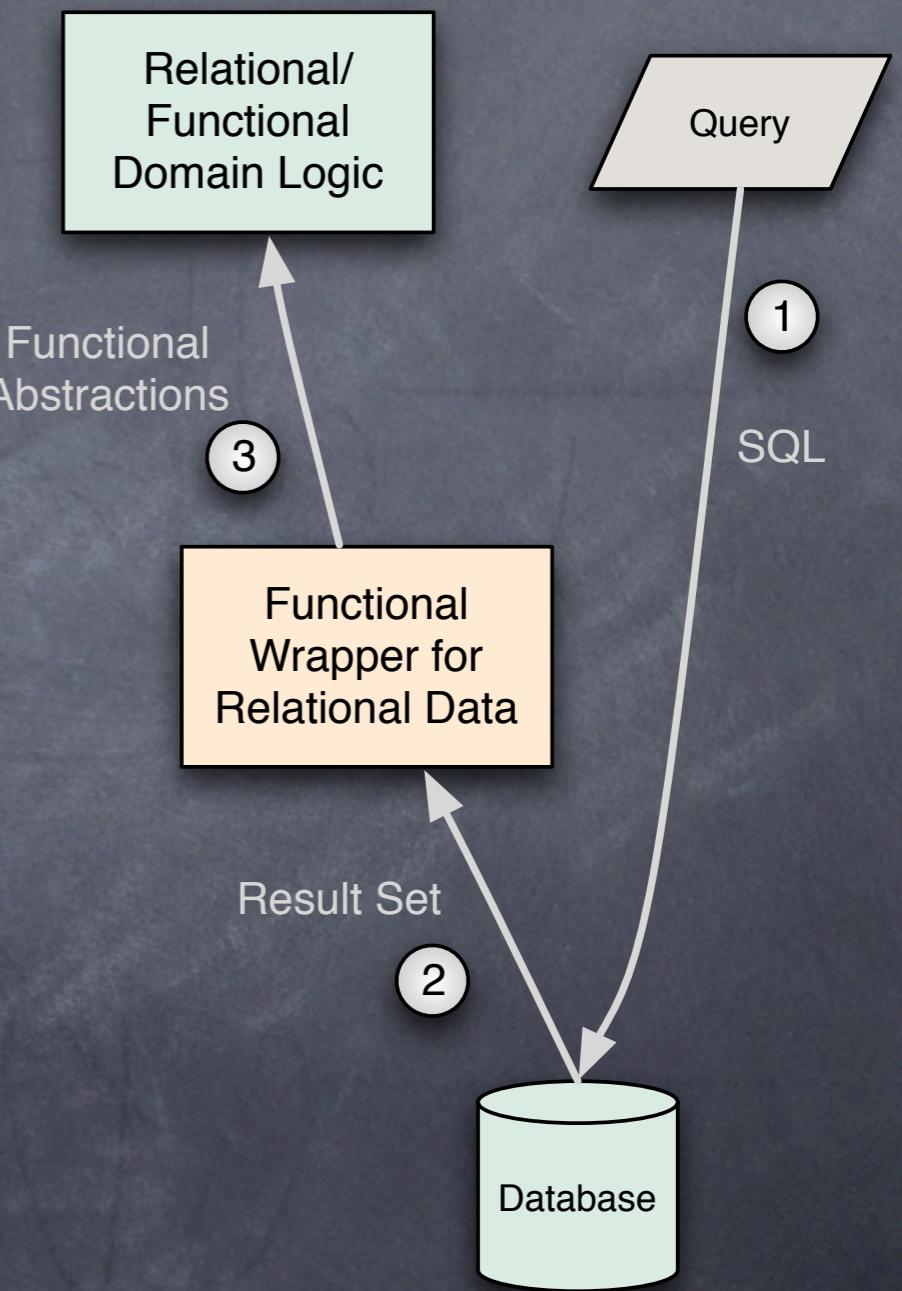
Also, your
browser wants
JSON...



Javascript stack: Browser, Node.js and MongoDB/CouchDB.



Uniform language and data representation.



Object Middleware and ORMs

Heresy or Dogma?

Object Middleware and ORMs

- + if your object model relatively stable.
- + for many OO languages.

Object Middleware and ORMs

- if high performance is essential.
- for functional languages.

The background image shows a dramatic sky at sunset or sunrise. The sun is low on the horizon, casting a bright orange glow that transitions into darker reds and purples. Large, dark, billowing clouds are scattered across the sky, some catching the light and appearing orange. The foreground is dark and indistinct.

Stupid Scala Trick...

Identifiers
with Spaces

```
scala> case class `My Class Has Spaces`(  
           `some int`: Int)
```

```
defined class My$u0020Class$u0020Has  
$u0020Spaces
```

```
scala> val `a value` =  
           new `My Class Has Spaces`(1)  
a value: My Class Has Spaces = My Class Has  
Spaces(1)
```

```
scala> println(`a value`)  
My Class Has Spaces(1)
```

blog.polyglotprogramming.com/2011/9/14/scala-identifiers-with-spaces

Identifiers with Spaces

Heresy or Dogma?

```
// JUnit tests:  
@Test public static void  
`delete(n) removes the nth item`() {  
    ...  
}  
  
// Enums  
enum ErrorCodes {  
    `Not Found`,  
    `Permission Denied`,  
    `Corrupt Format`;  
}
```

Sometimes,
whether it's
a Dogma or
a Heresy is
a matter of
branding...





@jaykreps

Jay Kreps

How to manage software technical debt:
(1) repackage it and sell it off as
collateralized debt obligations, (2) await
govt bailout.

8 Jan via Echofon



Favorite



Retweet



Reply

Retweeted by cmuller13 and 100+ others



<https://twitter.com/#!/jaykreps/status/23814156104769536>

Pictures from around Chicago.

© Dean Wampler

Thank You!

dean@deanwampler.com
@deanwampler

StrangeLoop 2011