To ORM or not to ORM

Álvaro Hernández Tortosa
<aht+pgcon2010@olog2n.com>
Olog2n

PGCon, 2010

What a wonderful world...

Most applications process data

↓

Many data applications need to persist data

↓

The relational/SQL model addresses most of these needs

↓

Databases were hard and error-prone to use from applications

↓

ORMs came to the rescue

It’s a wonderful world, we’ve got everything:

✔ Data may be safely persisted

✔ Databases also provide relational model, concurrency, powerful data-aware QL, etc

✔ ORMs make databases very easy to use

… this would be!!!!

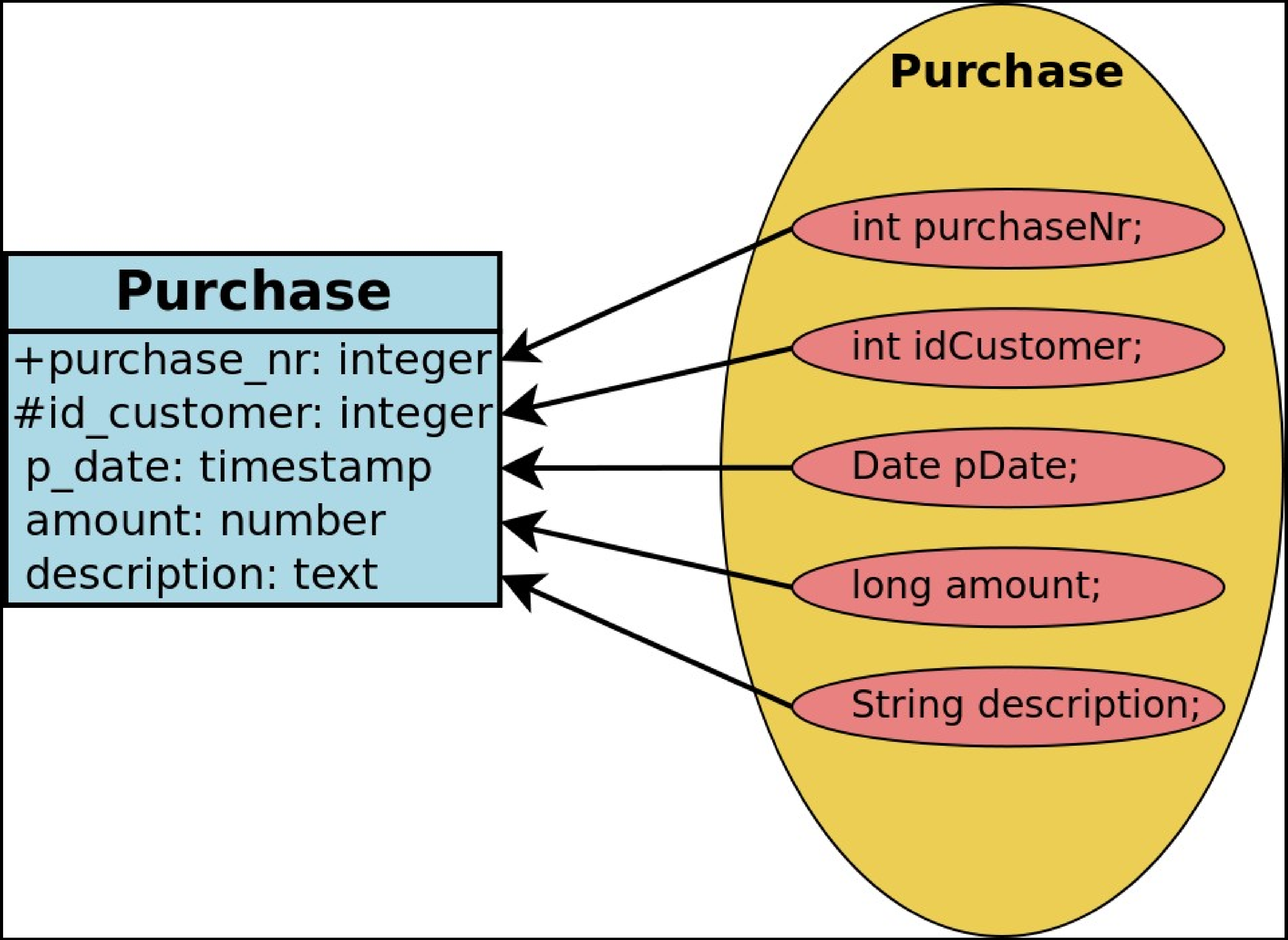
*“DEVs cry about the performance of my DB.
But its their ORM’s fault!”*

*“ORMs cannot frequently hold the abstraction,
so I often break it issuing direct SQL queries”*

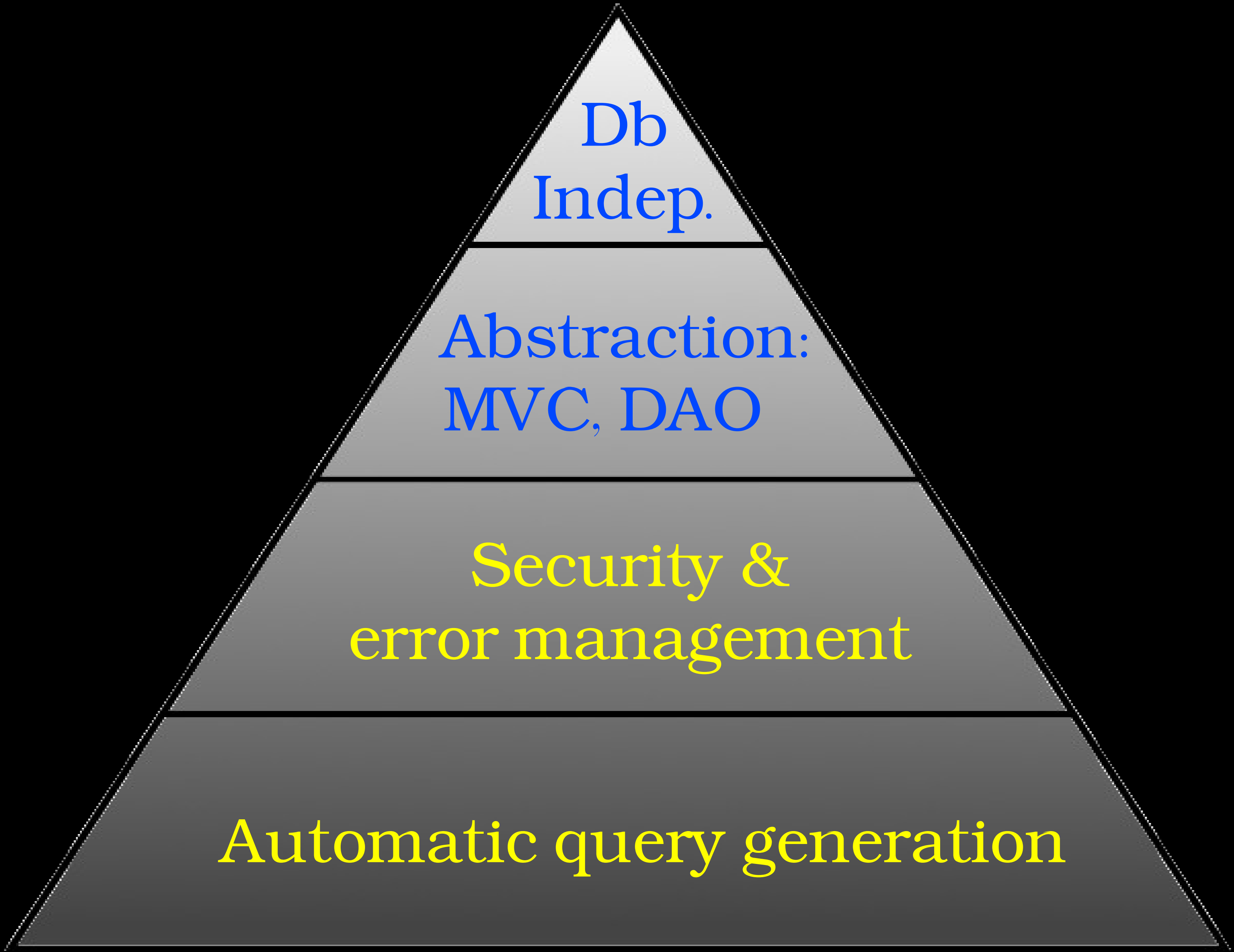
*“I just use the DB as a data storage.
Indeed, why do I need to use a DB?”*

Object to Relational Mapping

Relational (table) OO (class)

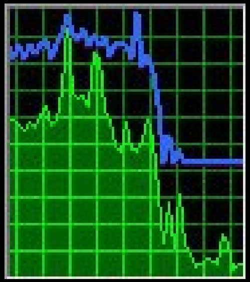


What drove the need for ORMs?



So, what's the deal, then?

DBAs



Performance problems

Loosing DBAs jobs!

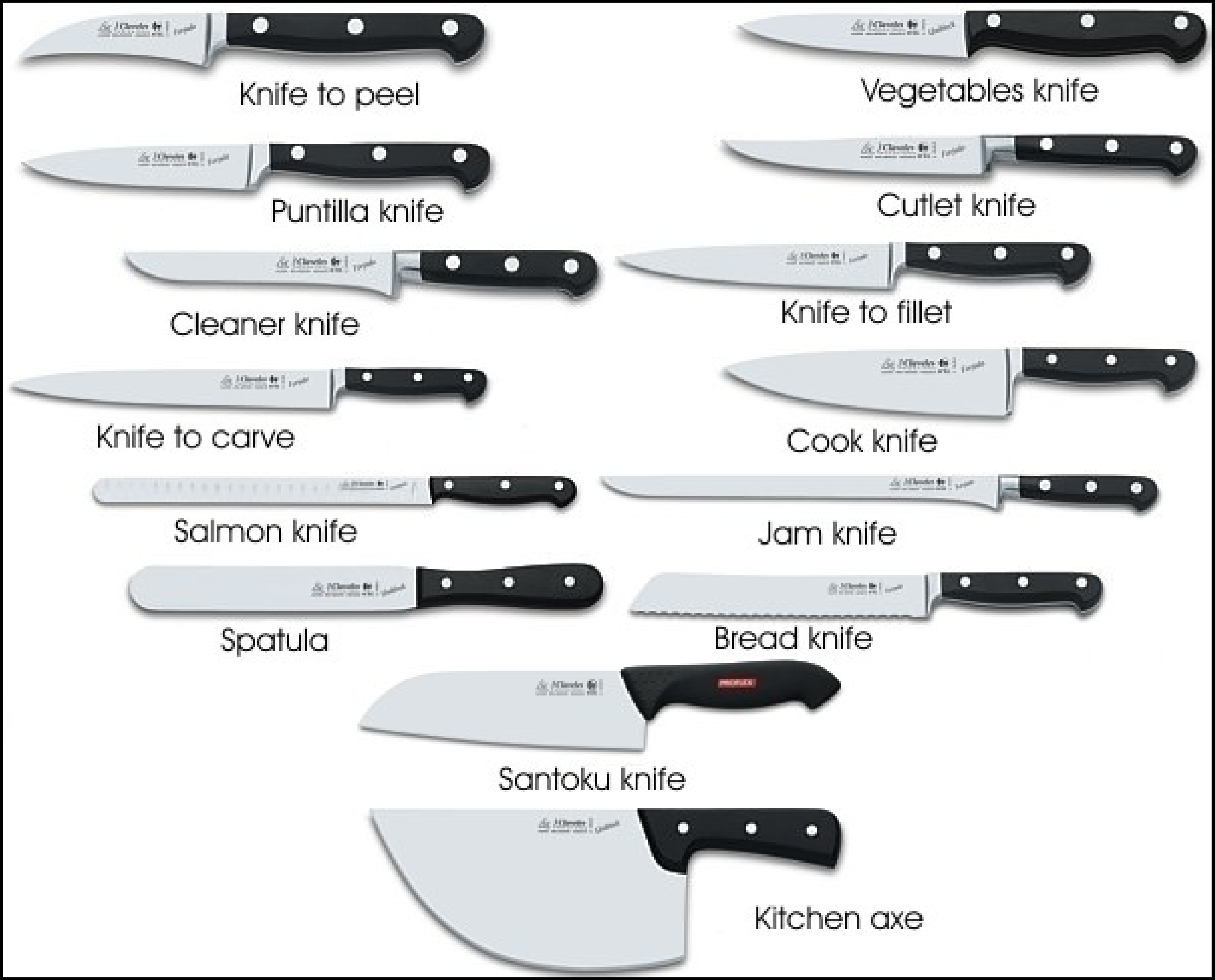
DEVs

Not fully exploiting the DB power

DBs are perceived as slow and heavyweight. Why use them?

Are they capable of fixing them all?
Is it just that they are missused?
Is the tool guilty?

Do you use knives?



Could you live without them?

Do you use them to kill?

Most people use knives
Most people need knives
Only very few people use knives to kill

=> It's not knive's fault, do not ban them

A lot of DEVs use ORMs
Most DEVs want or need ORMs
But results are not good. So even if it weren't ORM's fault,

=> It's ORM's fault if nobody get them right

Then, what's wrong with ORMs? (Part I. The basics)

Let's play "Right & Wrong"

	Right	Wrong	Stupid!
Map datatypes, Automatic SQL	X		
Create the database automatically		X	
Programmed query language			X

Then, what's wrong with ORMs? (Part II. The facts: db features)

- With ORMs, its DEVs who are creating the databases!!

- It's difficult, or impossible, to use some basic-to-intermediate db features, such as:
  - ➔ Domains / validated data types
  - ➔ Functions
  - ➔ Triggers
  - ➔ Custom aggregates
  - ➔ Roles

Then, what's wrong with ORMs?
(Part II. The facts: performance)

ORMs may perform very inefficient queries:
→ Doing JOINs everywhere
→ Doing selects/subselects instead of JOINs
→ Not batching up queries in transactions
→ Issuing more queries than its needed (many UPDATEs vs. UPDATE WHERE)

ORM's abstraction may lead to poor perf:
→ list.size() vs. SELECT COUNT(*)
→ Sorting/filtering in app space
→ It is frequently broken (direct queries)

Then, what's wrong with ORMs?
(Part II. The facts: db schema)

ORMs lead to problems in schema mgmt:

➔ Denormalized schemas

➔ Discriminator columns and NULLs everywhere (inheritance strategies)

➔ Severe deployment problems (apps are easily deployed, but how are live database schemas incrementally deployed?)

Then, what's wrong with ORMs?
(Part II. The facts: SQL abandonment)

ORMs contribute to SQL abandonment in favor OO databases, NoSQL, because:

- After all, they are not easy to use
- Don't support advanced SQL features
- Do not perform well
- The abstraction is frequently broken

But DEVs praise ORMs, while blame SQL

Then, what's wrong with ORMs? (Part III. Inventing a squared wheel)

I'm using a DB as the portal's backend, but news are stored in an XML file

Wait, what do you say?

News are simple, why bother with a DB? Indeed, they are not relational

But how you handle related news, categories, tags...?

Easy! Within the flat XML, we store FKs and some data constraints

Then, what's wrong with ORMs? (Part IV. The myth of db independence)

ORMs only offer features that are "standardized" among databases (i.e., the minimum subset of the common functionality)

This is very, very limiting (specially if you consider MySQL as a database ;)

It's done to achive *database independence*
- ➔ But who really changes the db?
- ➔ Different testing/production dbs? (Not sane)
- ➔ ORMs already have db-specific engines!
- ➔ Is it really worth to sacrifice a lot of db power?

Using ORMs
(aka test considerations)

- We assume db is generated (by DBAs). The ORM generates the classes ("reverse eng." ?)

- No special tweaking or advanced ORM knowledge would be used: we want to show what the average (or usual) user will do

- Will use Hibernate (probably most widely used ORM) as the testbed.

HIBERNATE

Using ORMs
(Test 1: a really simple test)

```
CREATE TABLE not_working (
   foo   integer,
   bar   integer
);
```

```
Result: ORM did not mapped the table.
Reason: it lacked a PK! – so what?
```

Using ORMs
(Test 2: a really simple test – with PK)

```
CREATE TABLE simple_test (
   foo     integer    PRIMARY KEY,
   bar     text
);
```

```
@Entity @Table(name="simple_test",schema="public")
public class SimpleTest {
    @Id
    @Column(name="foo",unique=true,nullable=false)
    public int getFoo() { … }
    @Column(name="bar")
    public String getBar() { … }
}
```

```
insert into public.simple_test (bar, foo) values
($1, $2);
```

Using ORMs
(Test 3: testing sequences)

```
CREATE SEQUENCE with_sequence_seq;
CREATE TABLE with_sequence (
   foo   integer PRIMARY KEY
                 DEFAULT nextval('with_sequence_seq')
);
```

```
@Entity @Table(name="with_sequence",schema="public")
public class WithSequence {
    @Id
    @Column(name="foo", unique=true, nullable=false)
    public int getFoo()
}
```

```
Results: always tries to insert “0” (fails!)
– Its easy to tweak, with PK generation strategies
```

Using ORMs
(Test 4: varchars, column CHECKS...)

```
CREATE TABLE varchar_and_checks (
   foo      varchar(9) PRIMARY KEY,
   bar      integer    CHECK (bar >= 5 AND bar < 100)
);
```

```
    @Id @Column(name="foo", unique=true,
nullable=false, length=9)
    public String getFoo() { … }
    @Column(name="bar")
    public Integer getBar() { … }
```

```
insert into public.varchar_and_checks (bar, foo)
values ($1, $2), $1 = '6', $2 = '0123456789'
insert into public.varchar_and_checks (bar, foo)
values ($1, $2), $1 = '1', $2 = '0'
Tweakable (requieres configuration and manual code)
```

Using ORMs
(Test 5: using DOMAINs)

```
CREATE DOMAIN foo_domain AS char(4) CHECK (
    VALUE IN ('bar1', 'bar2', 'bar3')
);
CREATE TABLE enum_domain (
   i     integer    PRIMARY KEY,
   foo   foo_domain NOT NULL
);
```

```
    private Serializable foo;

    @Column(name="foo", nullable=false)
    public Serializable getFoo() { … }

    public void setFoo(Serializable foo) { … }
```

```
Results: unusable!!!
```

Using ORMs
(Test 6: JOINs and lazy loading)

```
CREATE TABLE person (
   name varchar PRIMARY KEY, surname varchar);
CREATE TABLE student (
   name varchar PRIMARY KEY REFERENCES person(name),
   score   numeric(4,2));
```

```
    @Id @Column(name="name", unique=true,
nullable=false)
    public String getName() { … }
    @ManyToOne(fetch=FetchType.LAZY)
    @JoinColumn(name="name", unique=true,
nullable=false, insertable=false, updatable=false)
    public Person getPerson() { … }
    @Column(name="score", precision=4)
    public BigDecimal getScore() { … }
```

```
@OneToMany(cascade=CascadeType.ALL,
fetch=FetchType.LAZY, mappedBy="person")
    public Set<Student> getStudents() { … }
```

Using ORMs
(Test 6: JOINs and lazy loading)

```
public List<Student> getStudents() { … }
```

```
for(Student student : getStudents(session)) {
    System.out.println("Name: " + student.getName()
+ ", score: " + student.getScore().toPlainString());
    System.out.println("Surname: " +
student.getPerson().getSurname());
}
```

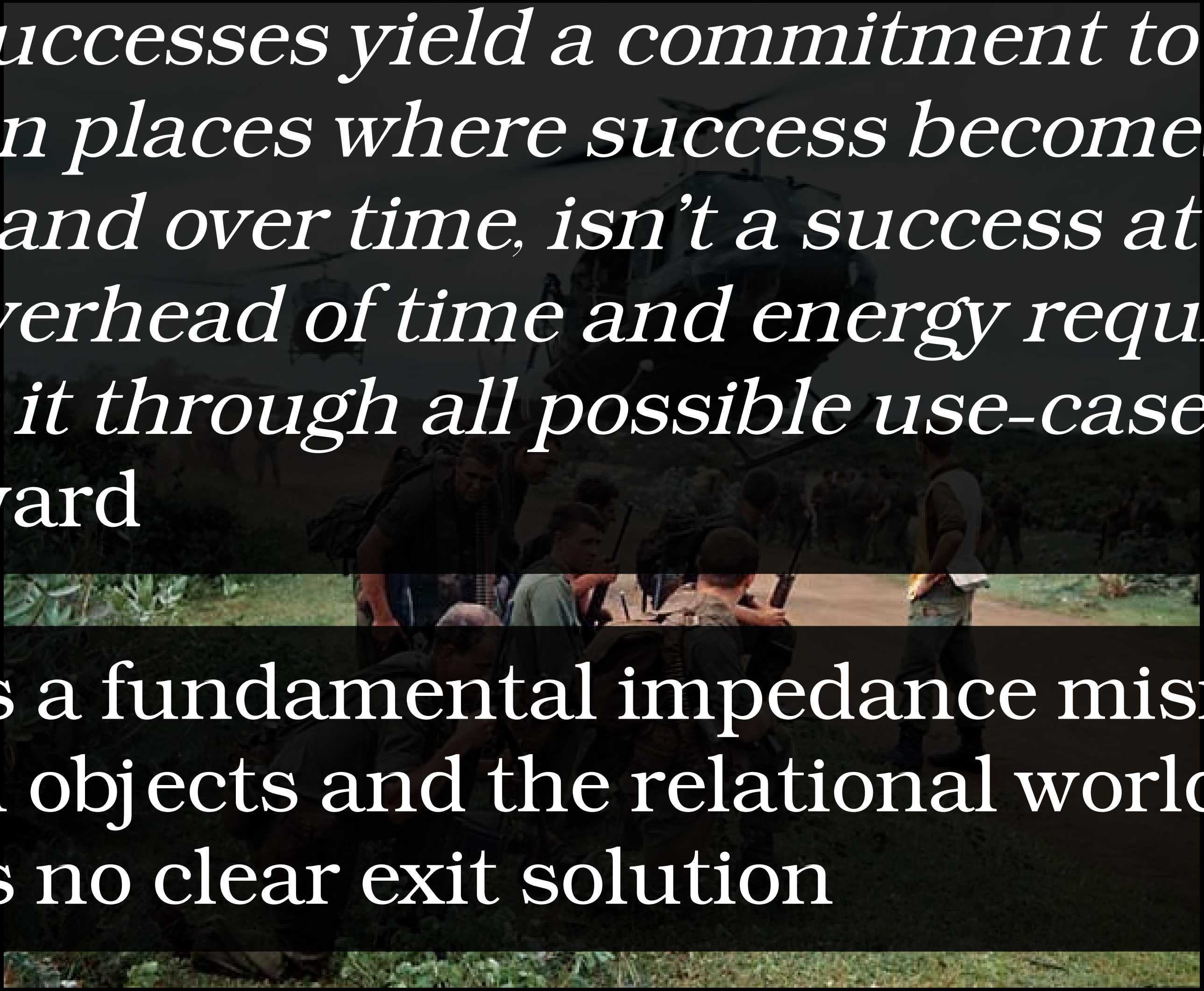
```
select student0_.name as name4_, student0_.score as
score4_ from public.student student0_

select person0_.name as name5_0_,
person0_.id_address as id2_5_0_, person0_.surname as
surname5_0_ from public.person person0_ where
person0_.name=$1
[repeated N times, one per row]

Result: M*N+1 queries (M: # of FKs, N: # of rows)
```

The Vietnam of ORMs
(aka why is it so difficult to get it right?)

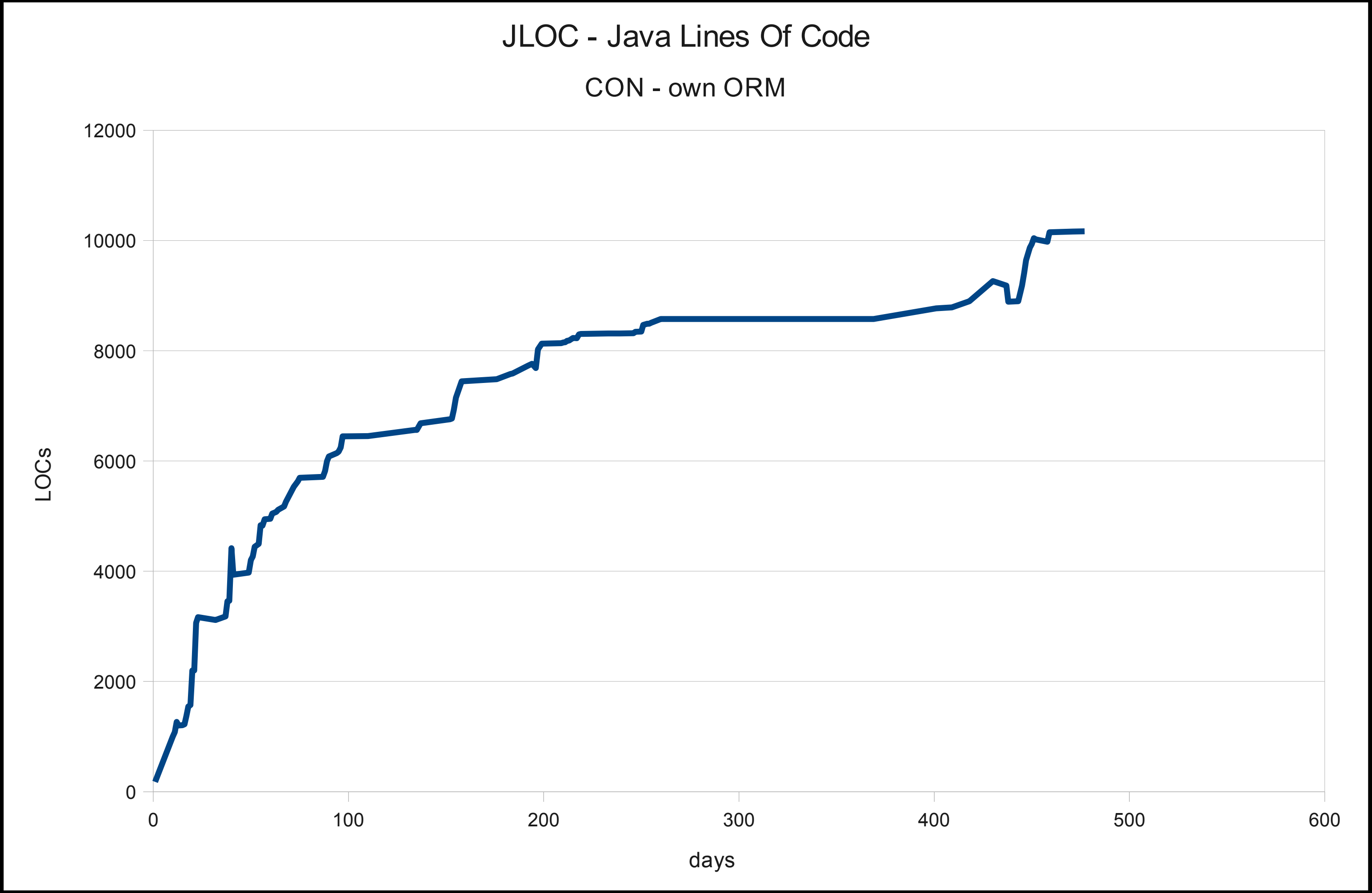
http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx



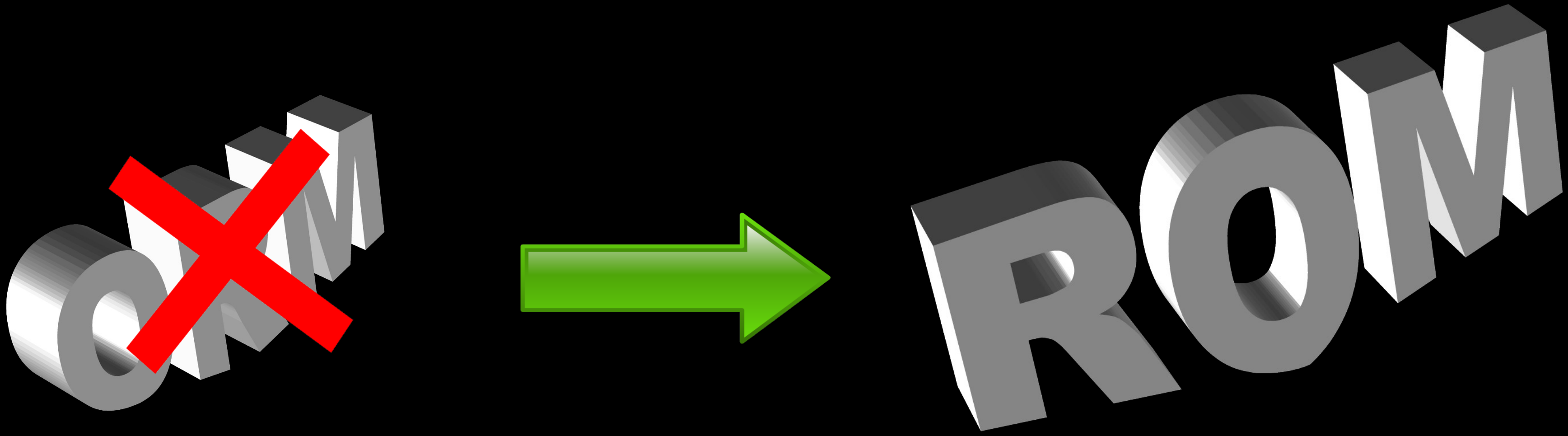
"Early successes yield a commitment to use O/R-M in places where success becomes more elusive, and over time, isn't a success at all due to the overhead of time and energy required to support it through all possible use-cases"
Ted Neward

- There's a fundamental impedance mismatch between objects and the relational world (sets)
- There's no clear exit solution

LOC evolution of a custom ORM



Looking for an exit strategy
First break all the rules!!!



No 1:1 type/relation field/column mapping.
Maybe columns to datatypes mapping, where a class is like any other query: a set of cols

Looking for an exit strategy
What DEVs should do (ROMs)

- ✓ DEVs should never create/alter db schemas
- ✓ ROMs should be able to create code from an existing database
- ✓ All data validation done within the db must be supported and also performed by ROMs
- ✓ ROMs should easily support db features such as function calls, aggregates, custom types, triggers... and map them to code automatically

Looking for an exit strategy
What DBAs should do

- ✓ DBAs are responsible for db alter/creation (*isn't it weird to be reclaiming this back?*)
- ✓ Create and define an "API" db ↔ app:
  - Is a contract with DEVs
  - May consist of tables, views, insert/update functions and the queries to obtain the data
  - Allows for normalized schemas while it's simple for DEVs
- ✓ Get involved in the design of ROMs

The PostgreSQL community
What’s our role here?

- DBAs, get involved!
- The PostgreSQL community is a very open one (BSD), open to changes. We need to send a message to the world
- Don’t sit back and wait! DBAs are loosing their jobs, or their current tasks! DEVs are running away to NoSQL. Say YeSQL!!
- Help in the next generation ROMs and collaborate with DEVs in the db/app API

Conclusion
Still an open debate

¿To ORM or not to ORM?

- Programmers need abstraction
- So, YES, but in a different way:
  - ROM
  - DB is done by DBAs, which offer an API
  - Rethink the 1:1 mapping mantra
  - Remove bloat, improve performance
  - Offer advanced SQL features

To ORM or not to ORM

Álvaro Hernández Tortosa
<aht+pgcon2010@olog2n.com>
Olog2n

PGCon, 2010