Are we there Yet??
(The long journey of Migrating from close source to opensource solution)

Marco (the Grinch) Tusa
Percona

About me

- Open source enthusiast
- Consulting team manager
- Principal Architect
- Working in DB world from 25 years
- Open source developer and community contributor

Section header

Click to add text

Gartner predicts by 2022:

- **Half of existing commercial database instances** will have migrated or will be in process of migrating to an Open Source DBMS
- **More than 70% of greenfield applications** will use an Open Source DBMS

PERCONA LIVE

Migration Services - Mitigating the Risk - Getting it Right

Motivations

- Average annual spend for enterprise database support: 5M-20M
- Annually compounding support costs are no longer tolerable

Challenges and Risks

- 78% - Reported migration projects more difficult than expected
- 65% - Expect the planning to take more than 6 months
- 90% - Reported migration projects took longer than expected

Three Questions

Why migrate

What can migrate

How to succeed

PERCONA LIVE

Common reasons to migrate

Cost 60%

Business Reasons (Politics) 35%

Technical requirements 5%

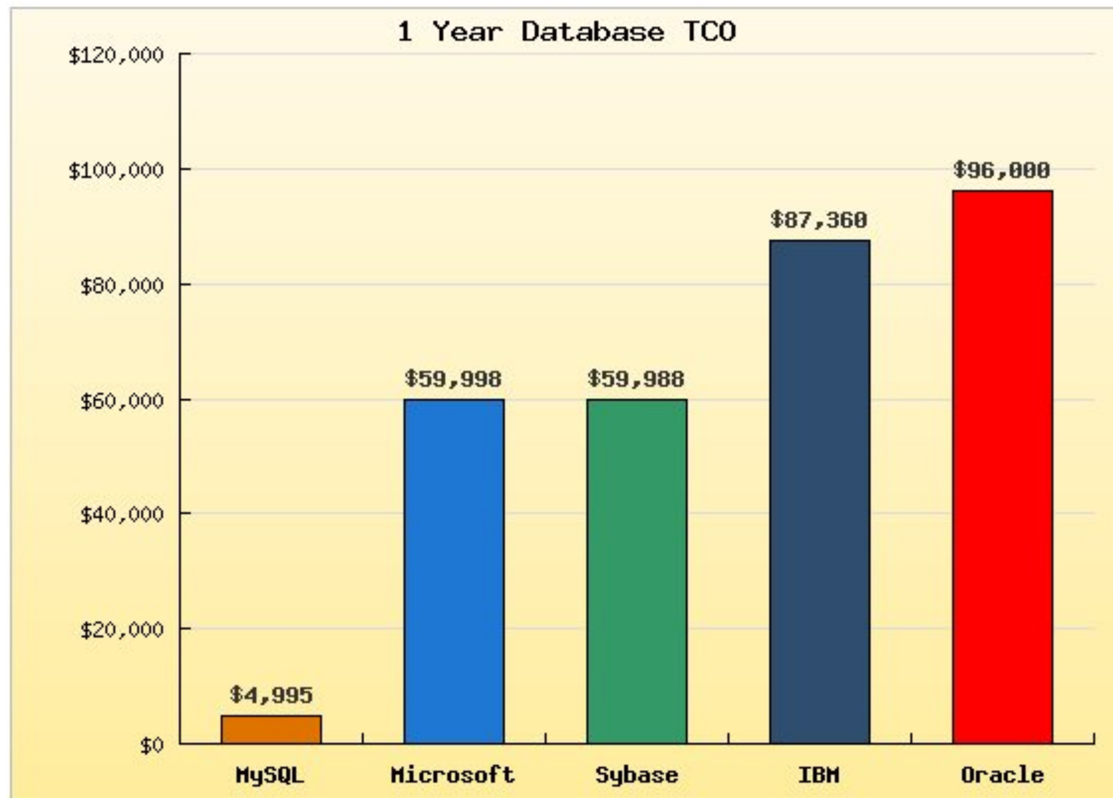
Migration was a THING from long

- Now a huge business
- There is a lot of confusion (too many simplifications)
- Business/Politics often overcome Technical reasons and common sense

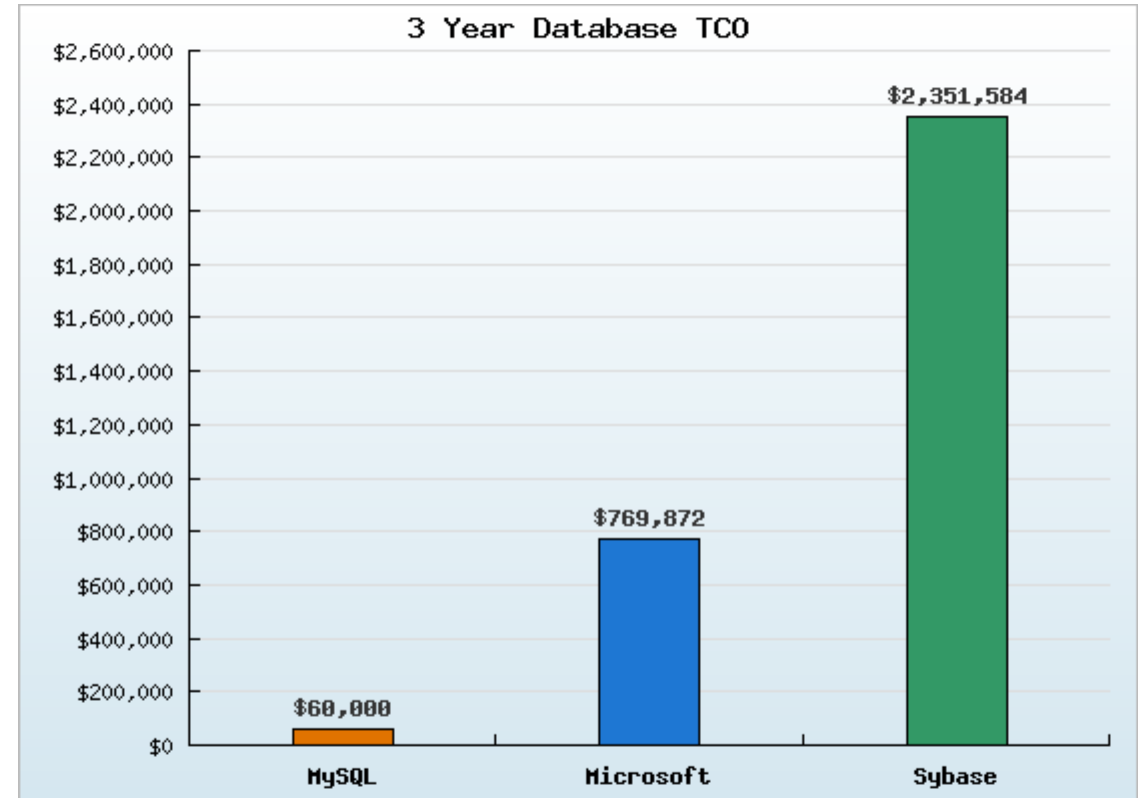
<u>Correct expectations is the key</u>

This is not a new thing

Migration from close source to open source is OLD



2008 pre SUN



2010 Oracle

PERCONA LIVE

Migrate to what?

Migration from premises to cloud is not new as well

Premises VS Cloud

Or

The place where you know you need SAs and DBAs and fantasyland where all works by magic

Migrate to what?

Migration to MySQL/MariaDB or PG ?

Or something else???

PERCONA LIVE

What are the most common steps?

First of all, do not rush take the right time to do all well

- Understand
- Map
- Rewrite
- Code/SQL
- Move data
- Test
- Go Live

4 Main phases:

1. Assessment
2. POC
3. Migrate
4. Go Live

PERCONA LIVE

Assessment

The journey will be long … do not start if you are not sure about

Not everything can be migrated, but you don’t know

Not all the schemas will take the same time, but you don’t know

Not all the code can be rewritten, but you do not know

<u>You need to KNOW before you move a finger</u>

PERCONA LIVE

POC

A Proof Of Concept is: *evidence sufficient to establish a thing as true, or to produce belief in its truth.*

As such is a critical moment to define if what was Assess can be done or not.

- Test … test … test
- Not only port the data
- Performance & functionalities

Be courageous, and if it will not work, be ready to drop

PERCONA LIVE

Migrate

Apply knowledge acquire during POC to

Transform schemas

Rewrite or export code

Move data

Implement DBOps procedure (db optimizations; backup/restore: etc)

Test applications

Go live

Replicate data

Keep it up to date

Cutover time

Sounds simple right?

I have news for you, it is not!

Defining the Process - Migration Methodology

Analyze

Understand

Map Src/Dest

Re/Design

Extract Src

Convert

Schemas

Logic

data

Index

Partition

Import

Test/POC

Validate

Something Breaks

PERCONA LIVE

Why MySQL Or Postgres

I like to start from :

- Scalability and Flexibility
- High Performance
- High Availability
- Robust Transactional Support
- Web and Data Warehouse Strengths
- Strong Data Protection
- Comprehensive Application Development
- Management Ease
- Open Source Freedom and 24 x 7 Support
- Lowest Total Cost of Ownership

10 things to know about MySQL

1. Subqueries are poorly optimized (still true)
2. Data integrity checking is very weak, and even basic integrity constraints cannot always be enforced. (replication)
3. Most queries can use only a single index per table; multi-index query plans exist in certain cases, but the cost is usually underestimated by the query optimizer, and they are often slower than a table scan.
4. Foreign keys are not supported in most storage engines.
5. Execution plans are not cached globally, only per-connection.
6. There are no integrated or add-on business intelligence, OLAP cube, etc packages.
7. There are no materialized views (also if we can use Event scheduler)
8. Native replication is asynchronous and has many limitations and edge cases.
9. DDL such as ALTER TABLE or CREATE TABLE is non-transactional. It commits open transactions and cannot be rolled back or crash-recovered.
10. Each storage engine can have widely varying behavior, features, and properties. (positive and negative)

4 things to know about Postgres

1. Changing Primary on a secondary require service restart
2. Partitions implemented as separate tables
3. Also minor version upgrade can be difficult (start from the Primary VS MySQL start from the last Slave)
4. Global Temporary table do not exists

The Motto

Use the right tool for the job

Let us see some more details

Click to add text

The assessment

Prepare a plan, and do not improvise

- Talk with Stakeholders (business/DBA/developers)
- Analyze the source (from application to data design)
- Identify show stoppers
- Identify how to map what to what
- Identify how to organize the target

Most important:

Be ready to do not force migration. If it does not make sense to proceed, STOP!

Most common source cases

- Database is used only to store data all the logic reside in the application
- Database contains logic such as stored procedure and complex package
- Database containing data for data warehouse
- Real time data and historical records (telephone company)

The assessment Mitigating risk of failure

When analyzing the source database(s) what should be the outcome?

- Easy to understand list of what is and out
- Identify Source type (Simple data move; data + code; etc.)
- In detail review per schema of complexity
- Assessment of modification and effort database objects
- Assessment of functions/functionalities used (also in the application) not always possible
- Application assessment and review

The assessment how it looks like

Migration levels (ML):

1. Migration that can be run automatically
2. Migration with code rewrite and a human-days cost up to 10 days
3. Migration with code rewrite and a human-days cost up to 30 days
4. Migration with code rewrite and a human-days cost up to 60 days
5. Migration with code rewrite and a human-days cost above 60 days
6. Not portable: Application code is dependent by the DB platform and is not possible to change the application code

Sub technical levels:

ML.1 = trivial: no stored functions and no triggers
ML.2 = easy: no stored functions but with triggers, no manual rewriting
ML.3 = simple: stored functions and/or triggers, no manual rewriting
ML.4 = manual: no stored functions but with triggers or views with code rewriting
ML.5 = difficult: stored functions and/or triggers with code rewriting

PERCONA LIVE

The assessment how it looks like

Color code	Meaning
	Easy
	Some difficulties but possible
	Some stopper but possible
	Very complex and long
	As it is, Unmovable

Schema	Application	Man days	Hestimated instances	Man days per schema type	Difficulty Level	Color code
ABC	Drupal	2.0	1	2	1-1	
DEF	wordpres	1.0	1	1	1-1	
GHI	My Java app	1.5	1	1.5	1-1	
LMN	Sales App	2.0	1	2	1-1	
OPQR	Booking	2.0	1	2	1-1	
STV	Insurances	5.0	12	30	3-3	
Z1	Tikets	12	10	48	4-5(*)(**)	
AA1	Advertise	2.00	19	30	1-1*	
AA2	Network App	2.0	200	80	1-1*	

*Per instance

**Schema may take short time, but Store procedure require rewrite

PERCONA LIVE

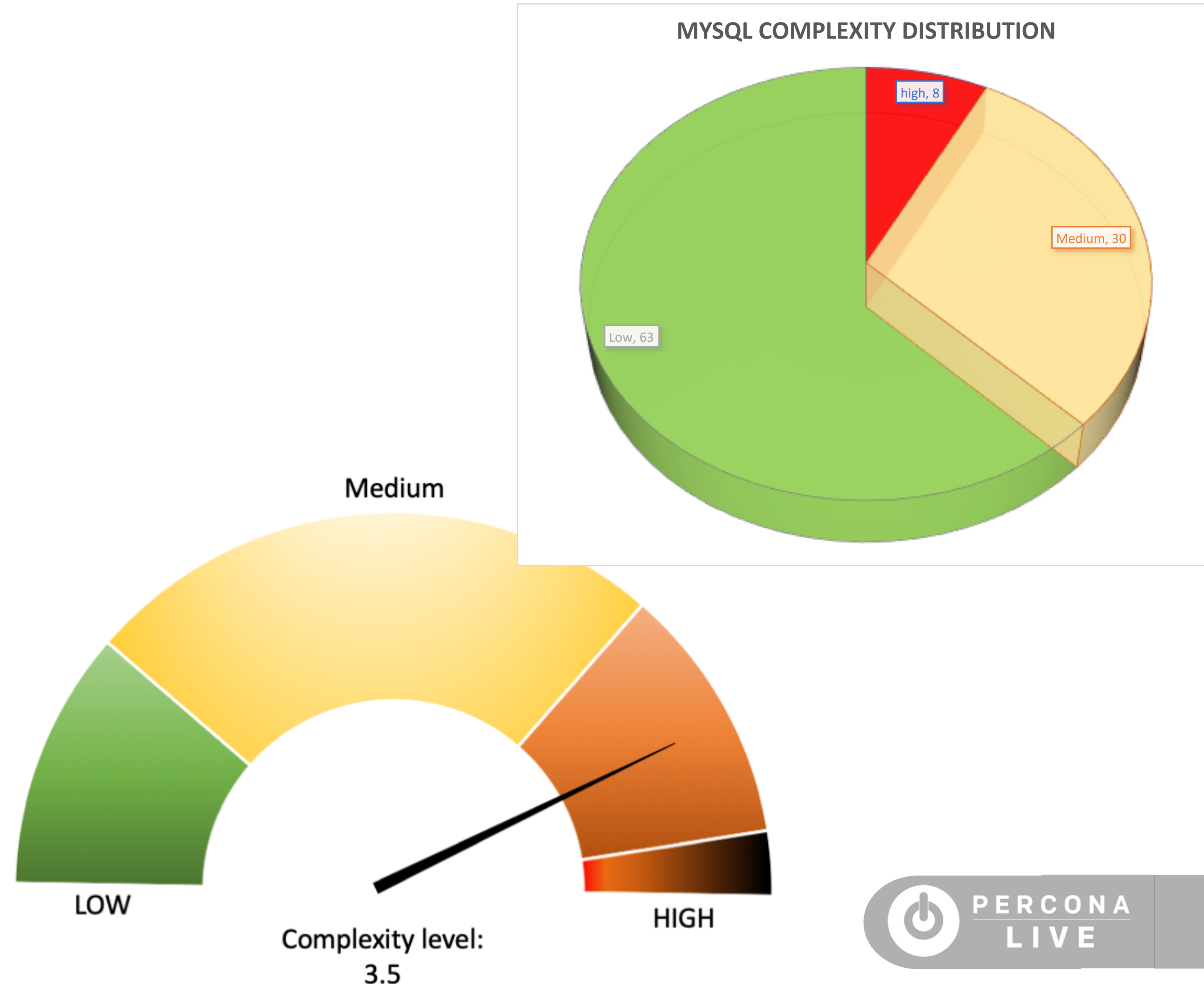
The assessment how it looks like

Color code	Meaning
	Easy
	Some difficulties but possible
	Some stopper but possible
	Very complex and long
	As it is, Unmovable

Schema	Application	Man days	Difficulty Level	Color code (PG)	Color code (My)
Schema1	App1	64	5.5		
Schema2	App2	1	1.1		
Schema3	App2	200	5.5		
Schema4	App2	10	2.5		
Schema5	App3	250	6.5		
Schema6	App4	150	5.4		

The assessment how it looks like

High	**23**
DBMS_UTILITY	1
UTL_FILE	21
Virtual Column Partitions	1
Low	**190**
BTree Indexes	1
Bulk Collect	5
CLOB Data Types	6
Cursors	2
DBMS_OUTPUT	33
Float DataTypes	1
Raw Data Type	3
Sequences	133
Table Triggers	4
Views	2
Medium	**90**
Functions	51
Stored Procedures	33
User Defined Types	6
Grand Total	**303**



Converting schemas

Correct datatypes identification
Identify different and problematic objects like materialized view Global temporary tables etc.

Open source tools exists to support you during this part:

- Ora2PG
- SQLines

Not Open source

- AWS Schema Conversion Tool
- Oracle to MySQL converter
- Inspirer SQLWays

Converting schemas

Understanding DDL differences

Identify conversion between Oracle and MySQL for

- Tables
- Views
- Procedures
- Functions
- Packages
- Triggers
- Sequences, synonyms etc.

I.e. data types:

MySQL Data Type	Oracle Data Type
BIGINT	NUMBER(19, 0)
BIT	RAW
BLOB	BLOB, RAW
CHAR	CHAR
DATE	DATE
DATETIME	DATE
DECIMAL	FLOAT (24)
DOUBLE	FLOAT (24)
DOUBLE PRECISION	FLOAT (24)
ENUM	VARCHAR2
FLOAT	FLOAT

MySQL Data Type	Oracle Data Type
INT	NUMBER(10, 0)
INTEGER	NUMBER(10, 0)
LONGBLOB	BLOB, RAW
LONGTEXT	CLOB, RAW
MEDIUMBLOB	BLOB, RAW
MEDIUMINT	NUMBER(7, 0)
MEDIUMTEXT	CLOB, RAW
NUMERIC	NUMBER
REAL	FLOAT (24)
SET	VARCHAR2
SMALLINT	NUMBER(5, 0)
TEXT	VARCHAR2, CLOB
TIME	DATE
TIMESTAMP	DATE
TINYBLOB	RAW
TINYINT	NUMBER(3, 0)
TINYTEXT	VARCHAR2
VARCHAR	VARCHAR2, CLOB
YEAR	NUMBER

Converting schemas

- Re-organize the schema/table not just convert data types
  - Storage engines
  - Index full redesign
- Data organization
  - Sharding
  - Partition
- Logic rewrite
  - Inside MySQL
  - Move to application

PERCONA LIVE

Converting SQL

1. Join syntax
2. SQL_mode
3. Data comparison using collation
4. other common differences:
   - SQL macro differences
   - NVL() --> IFNULL()
   - ROWNUM --> LIMIT
   - SEQ.CURRVAL --> LAST_INSERT_ID()
   - SEQ.NEXTVAL --> NULL
   - NO DUAL necessary (SELECT NOW())
   - NO DECODE() --> IF() CASE()
   - JOIN (+) Syntax --> INNER|OUTER LEFT|RIGHT
   - No Hierarchical (connect to prior)

Exporting data

3 Different types:

- Easy, Medium (Ora2PG; SQLines Data; other tools)
- Large and complex (SqlWays; exporting code)
- Huge and over (Write proper exporting code)

Mitigating risk of failure with the code

Understanding Function Triggers difference

Given The relevance in a Migration of the presence of SP/Trigger it is worth to talk about it a little bit more in details

Procedure and triggers difference

- one trigger for event in MySQL, all the different actions needs to be group
- no packages, workaround using a fake schema
- different behavior by storage engine and if transactional or not
- Security assignments and security definer/invoker
- Very basic error handling.

Mitigating risk of failure with code

Understanding Function Triggers difference

MySQL stored programs can often add to application functionality and developer efficiency, and there are certainly many cases where the use of a procedural language such as the MySQL stored program language can do things that a non procedural language like SQL cannot.

There are also a number of reasons why a MySQL stored program approach may offer performance improvements over a traditional SQL approach

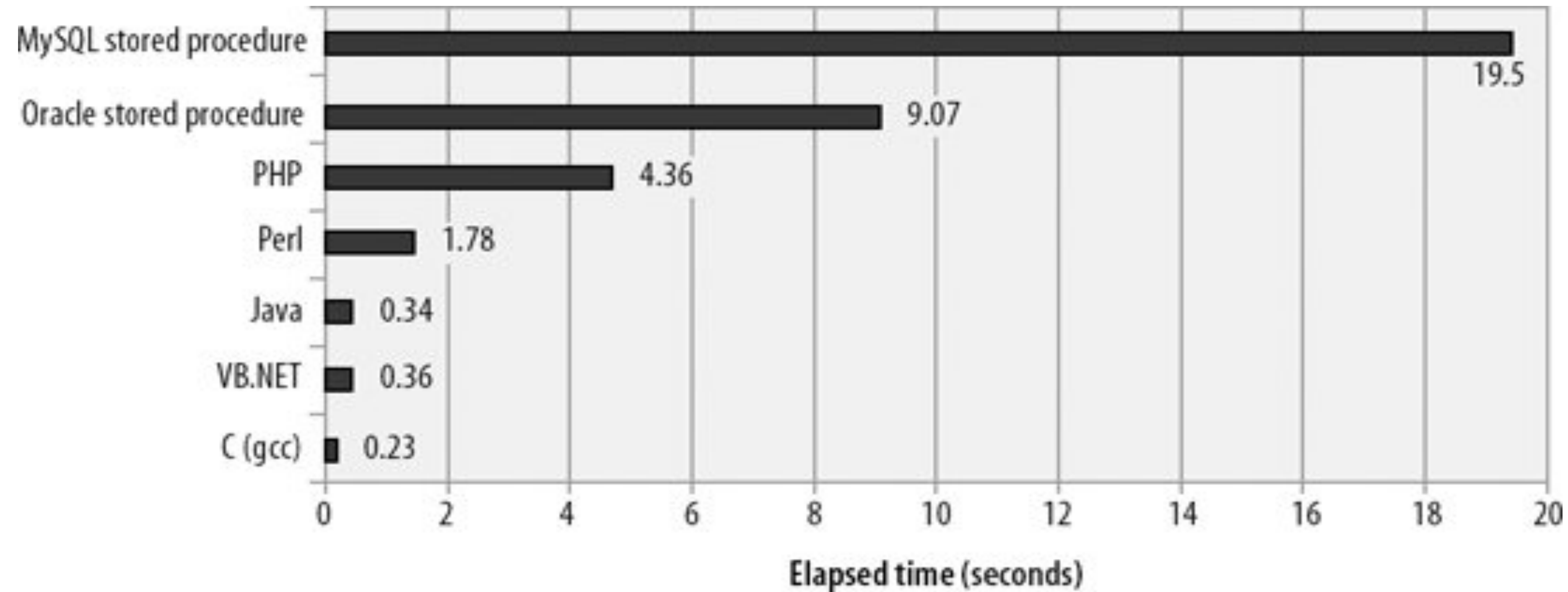
- It provides a procedural approach (SQL is a declarative, non procedural language)
- It reduces client-server traffic
- It allows us to divide and conquer complex statements

But...

Mitigating risk of failure with the code

Understanding Function Triggers difference

One graph tells more then 1,000 words:



Mitigating risk of failure with the code

Understanding Function Triggers difference

IF and CASE Statements

When constructing IF and CASE statements, try to minimize the number of comparisons that these statements are likely to make by testing for the most likely scenarios first.

For instance, in the code in the next slide, the first statement maintains counts of various percentages.

Assuming that the input data is evenly distributed, the first IF condition (percentage>95) will match about once in every 20 executions.
On the other hand, the final condition will match in three out of four executions. So this means that for 75% of the cases, all four comparisons will need to be evaluated.

Mitigating risk of failure with the code

Understanding Function Triggers difference

Non Optimized

```
IF (percentage>95) THEN
      SET Above95=Above95+1;
ELSEIF (percentage >=90) THEN
      SET Range90to95=Range90to95+1;
ELSEIF (percentage >=75) THEN
      SET Range75to89=Range75to89+1;
ELSE
      SET LessThan75=LessThan75+1;
END IF;
```

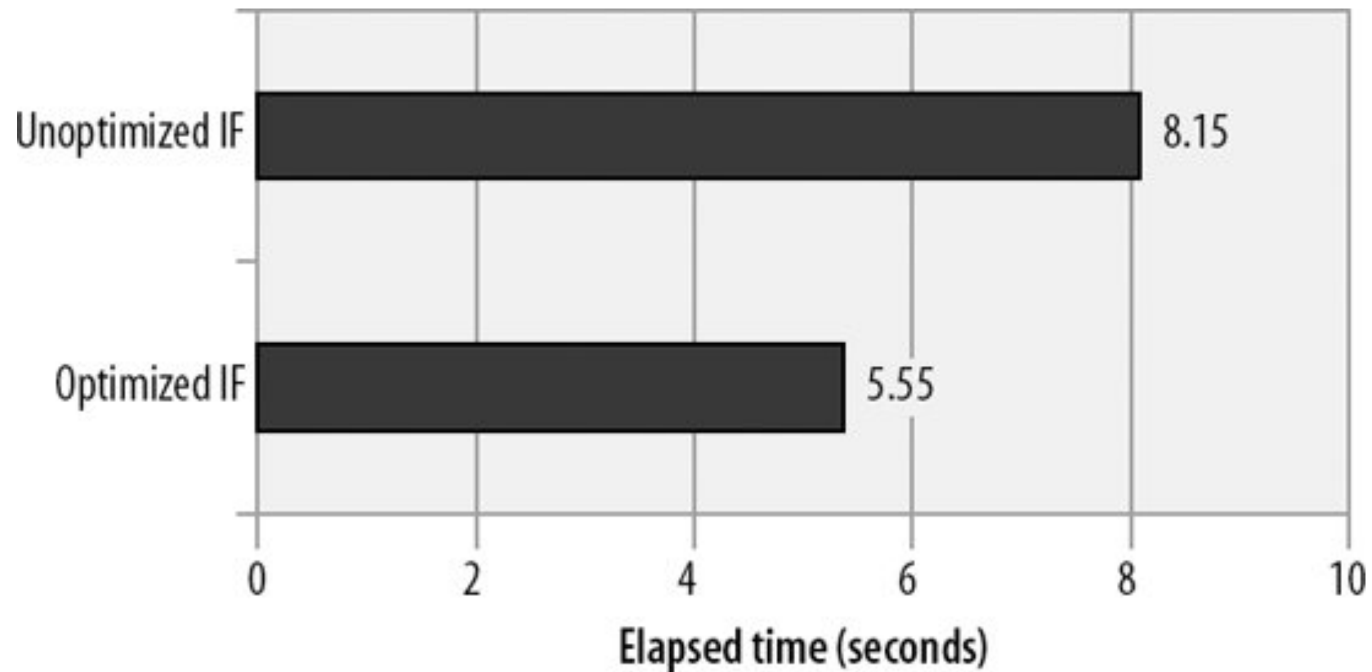
Optimized

```
IF (percentage<75) THEN
      SET LessThan75=LessThan75+1;
ELSEIF (percentage >=75 AND percentage<90) THEN
      SET Range75to89=Range75to89+1;
ELSEIF (percentage >=90 and percentage <=95) THEN
      SET Range90to95=Range90to95+1;
ELSE
      SET Above95=Above95+1;
END IF;
```

Mitigating risk of failure with the code

Understanding Function Triggers difference

Looks simple and the effect is relevant:



Mitigating risk of failure with the code

Understanding Function Triggers difference

Using Recursion

A recursive routine is one that invokes itself.

Recursive routines often offer elegant solutions to complex programming problems, but they also tend to consume large amounts of memory.

They are also likely to be less efficient and less scalable than implementations based on iterative execution.

Mitigating risk of failure with the code

Understanding Function Triggers difference

Recursive

```
CREATE PROCEDURE rec_fib(n
   INT,OUT out_fib INT)
BEGIN
  DECLARE n_1 INT;
  DECLARE n_2 INT;

  IF (n=0) THEN
    SET out_fib=0;
  ELSEIF (n=1) then
    SET out_fib=1;
  ELSE
    CALL rec_fib(n-1,n_1);
    CALL rec_fib(n-2,n_2);
    SET out_fib=(n_1 + n_2);
  END IF;
END
```

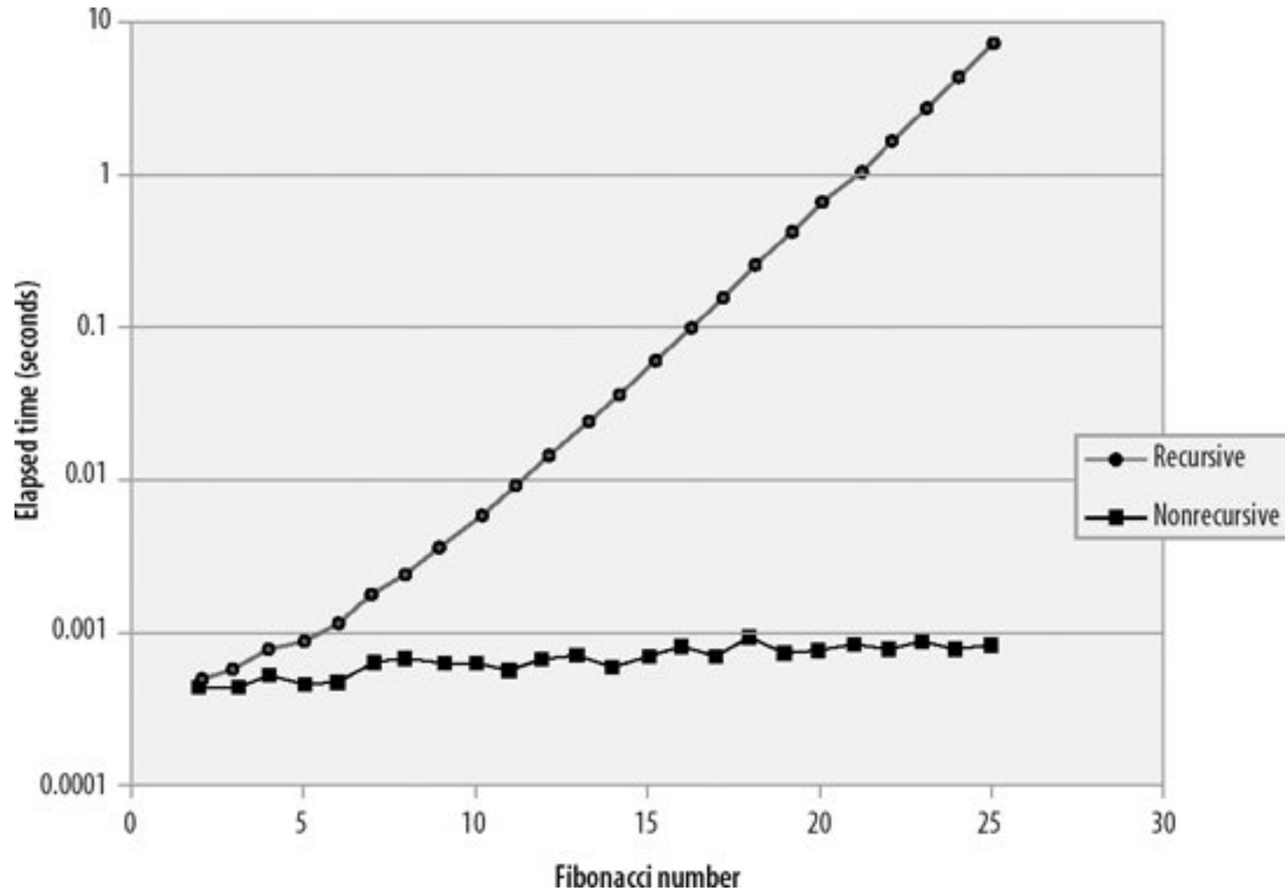
Not Recursive

```
CREATE PROCEDURE nonrec_fib(n
   INT,OUT out_fib INT)
BEGIN
  DECLARE m INT default 0;
  DECLARE k INT DEFAULT 1;
  DECLARE i INT;
  DECLARE tmp INT;

  SET m=0;
  SET k=1;
  SET i=1;

  WHILE (i<=n) DO
    SET tmp=m+k;
    SET m=k;
    SET k=tmp;
    SET i=i+1;
  END WHILE;
  SET out_fib=m;
 END
```

Mitigating risk of failure with the code

Understanding Function Triggers difference

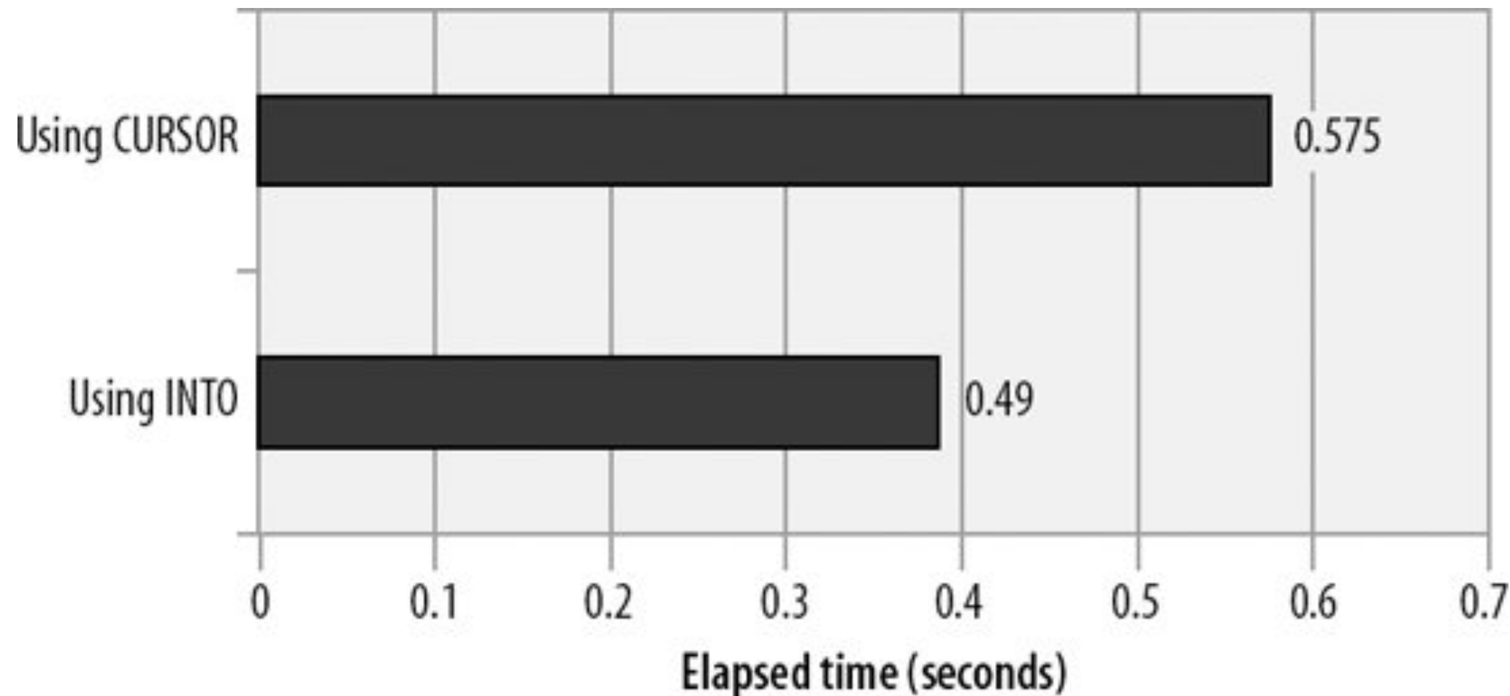
The difference is quite impressive and evident



Mitigating risk of failure with the code

Understanding Function Triggers difference

When you need to retrieve only a single row from a SELECT statement, using the INTO clause is far easier than declaring, opening, fetching from, and closing a cursor. But does the INTO clause generate some additional work for MySQL or could the INTO clause be more efficient than a cursor?



Mitigating risk of failure with the code

Understanding Function Triggers difference

Trigger Overhead

Every database trigger is associated with a specific DML operation (INSERT, UPDATE, or DELETE) on a specific table the trigger code will execute whenever that DML operation occurs on that table.

Furthermore, all MySQL 8.x triggers are of the FOR EACH ROW type, which means that the trigger code will execute once for each row affected by the DML operation.

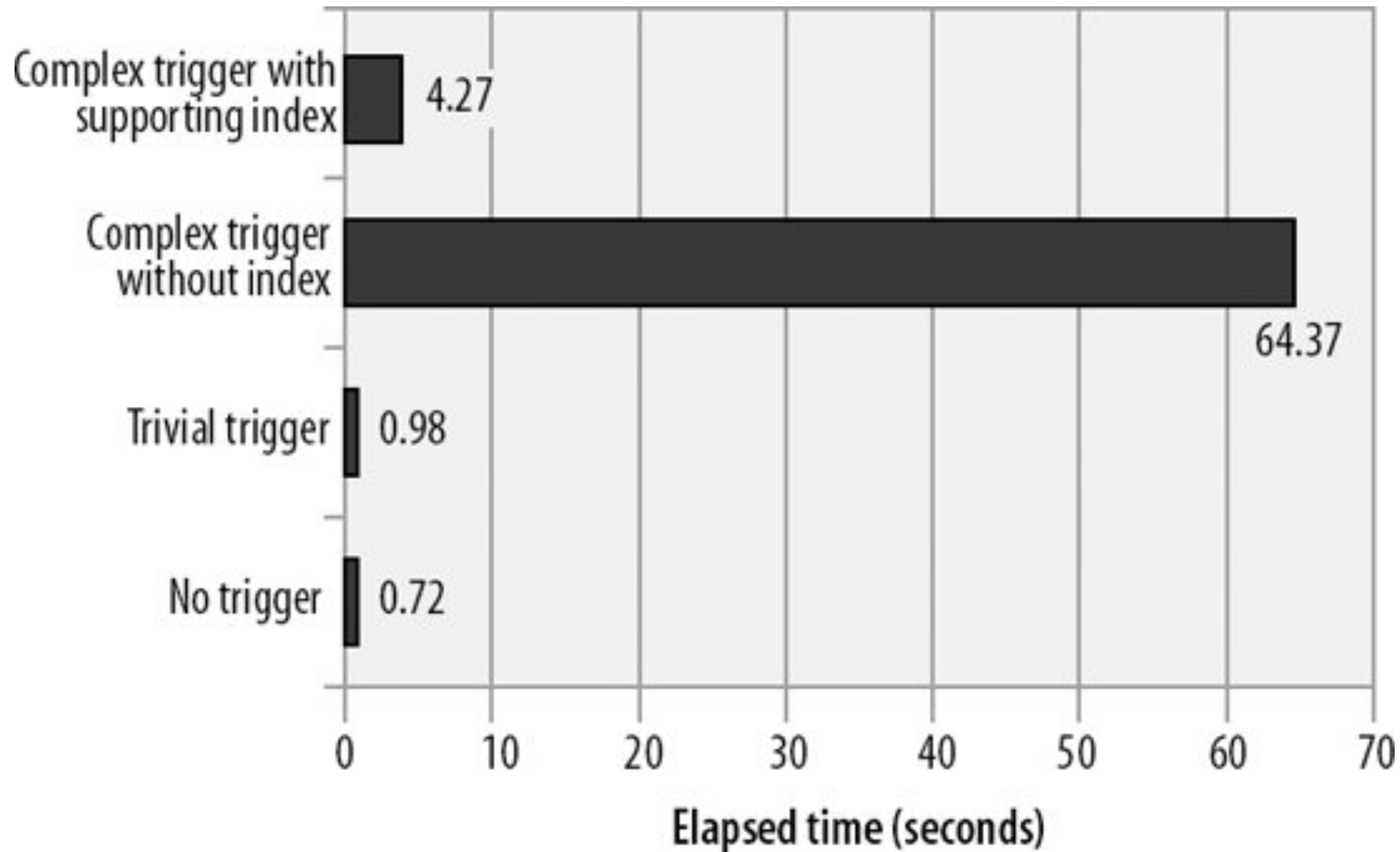
Given that a single DML operation might potentially affect thousands of rows, should we be concerned that our triggers might have a negative effect on DML performance?

Absolutely yes!

Mitigating risk of failure with the code

Understanding Function Triggers difference

When using Trigger be ALWAYS sure to have the right indexes.



Prepare for the POC

✧Don't work Alone

✧Involve Oracle experienced DBA

✧Involve MySQL experience DBA

✧Involve the developers

✧Use real data

✧Use real traffic

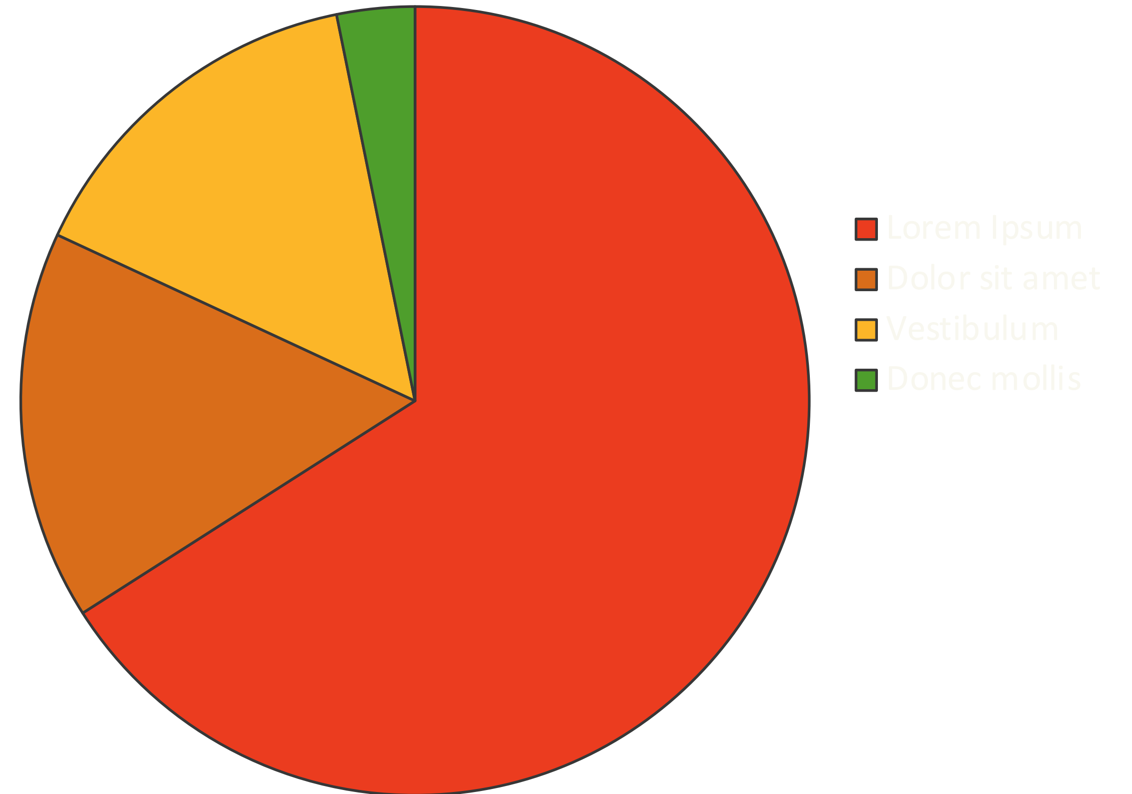
✧Take one source for each type; start with the easy one

Go Back to the analysis phase if you have to

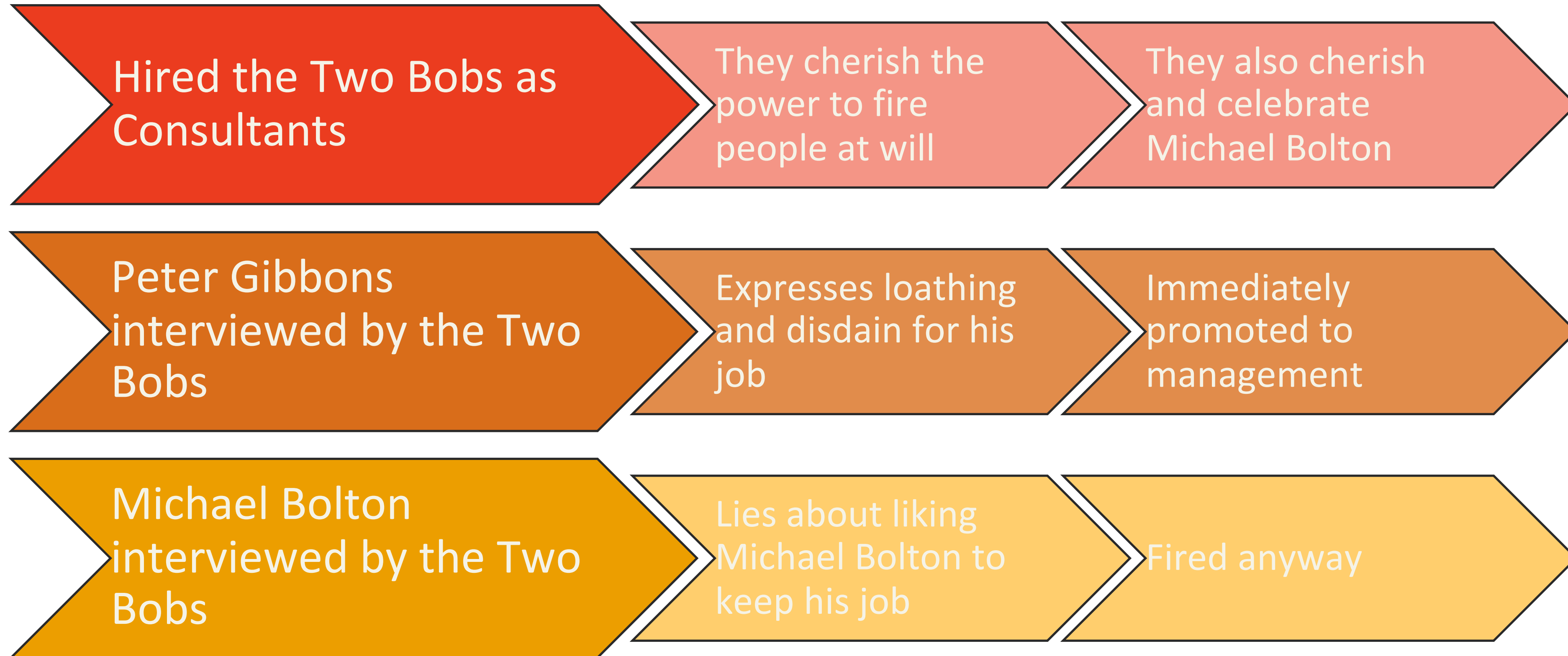
Ok I have migrate all … now what ??

Lorem Ipsum market share

- Lorem ipsum dolor sit amet
- **Curabitur scelerisque malesuada auctor**
- Vestibulum molestie pharetra mauris
- Donec finibus mi eu ultricies tincidunt



Lorem Ipsum some title here and there



Section header

Click to add text

Another slide Title

Past Keyote Speakers:

- Lorem Sapien
- Justo Nulem
- Finibus Proin
- Aliquam Hendrerit
- Donec Dapibus
- Tempus Orci

IniTech Culture and Values

We put cover sheets on TPS reports

Did you get the memo about the cover sheets?

Focused 100% on Cover Sheets

If you could go ahead and start using the cover sheet, that would be great. Mkay?

Oh, and Friday is Hawaiian shirt day

Vendor Agnostic Lorem Ipsum Performance Services

Lorem, Ipsum, Dolar, Sit Amet, Vestibulum, Consectetu

Vestibulum molestie pharetra mauris condimentum mollis.

- Nemo enim ipsam voluptatem quia voluptas sit aspernatur
- Neque porro quisquam est
- We are great at Lorem and Ipsum
- Ut enim ad minima veniam!

Rate My Session

Schedule
Timezone: Europe/Berlin +02:00

MON 3 TUE 4 WED 5

11:20

TAP THE SESSION

Introducing gh-ost: triggerless, painless, trusted online schema migrations
11:20 - 12:10, Matterhorn 2

Details

Introducing gh-ost: triggerless, painless, trusted online schema migrations
11:20 → 12:10
Matterhorn 2

Rate & Review

TAP TO RATE & REVIEW

Rate & Review

Tap a star to rate

Feedback (optional)

Anonymously

SUBMIT