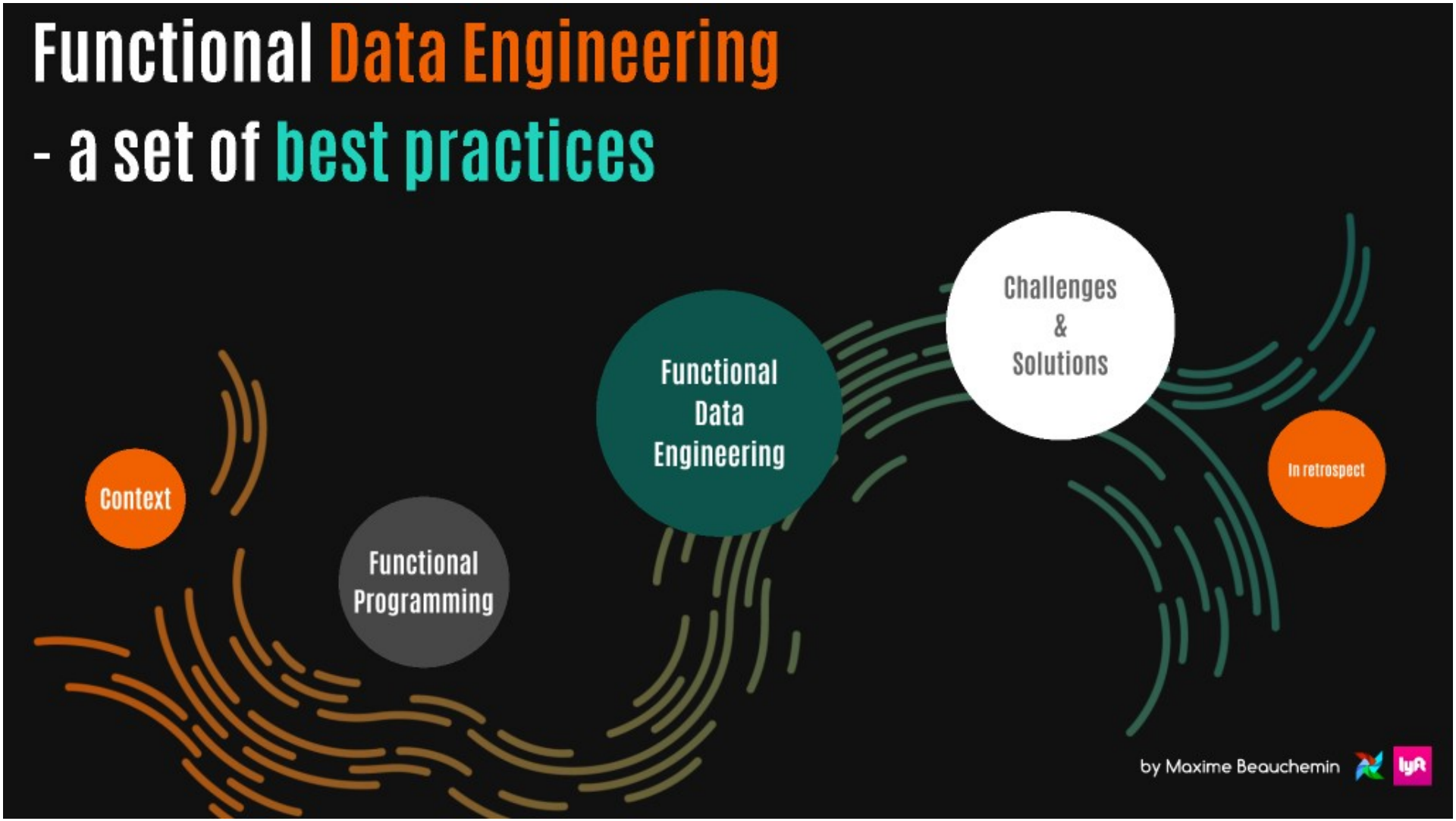
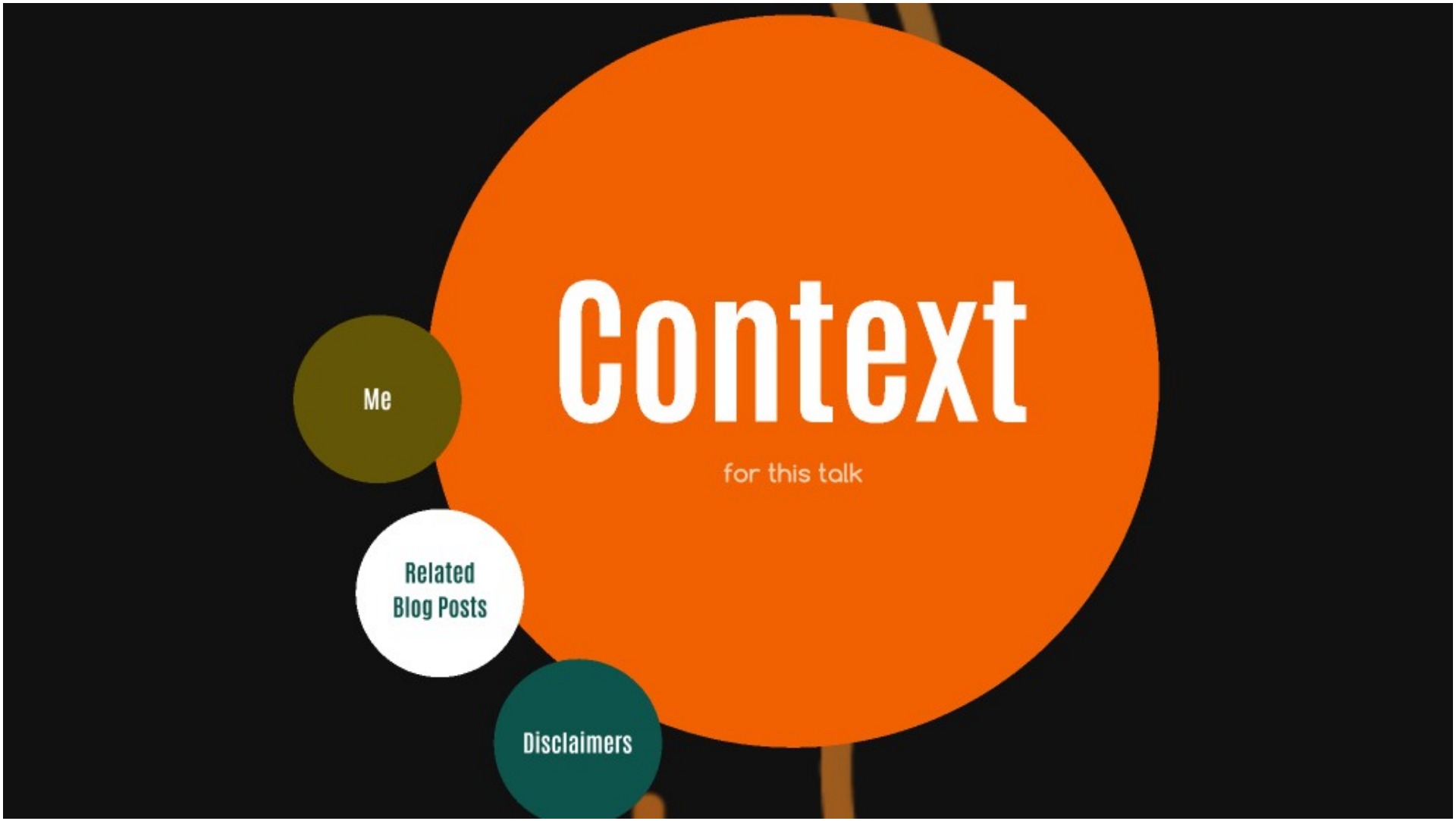

Functional Data Engineering
- a set of best practices
Context
Functional Programming
Functional Data Engineering
Challenges & Solutions
In retrospect
by Maxime Beauchemin


Context
for this talk
Me
Related
Blog Posts
Disclaimers


WANTED
DEAD OR ALIVE
lyft
airbnb
Superset
UBISOFT
Y!
Maxime Beauchemin
$15,000
REWARD

Related Medium posts

The Rise of the Data Engineer

The Downfall of the Data Engineer

Functional Data Engineering—a modern paradigm for batch data processing

"All the News That's Fit to Print"

The New York Times

Weather: Rain, warm today; clear tonight. Sunny, pleasant tomorrow. Temp. range: today 80-66; Sunday 71-66. Temp.-Hum. Index yesterday 69. Complete U.S. report on P. 50.

VOL. CXVIII . No. 40,721 NEW YORK, MONDAY, JULY 21, 1969 10 CENTS

MEN WALK ON MOON

ASTRONAUTS LAND ON PLAIN; COLLECT ROCKS, PLANT FLAG

Voice From Moon: 'Eagle Has Landed'

EAGLE (the lunar module): Houston, Tranquility Base here. The Eagle has landed.

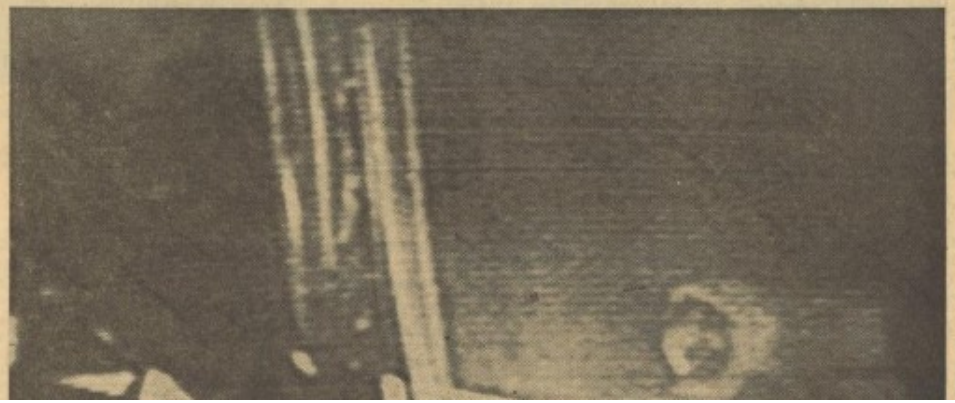
HOUSTON: Roger, Tranquility, we copy you on the ground. You've got a bunch of guys about to turn blue. We're breathing again. Thanks a lot.

TRANQUILITY BASE: Thank you.

HOUSTON: You're looking good here.

TRANQUILITY BASE: A very smooth touchdown.

HOUSTON: Eagle, you are stay for T1. [The first step in the lunar operation.] Over.



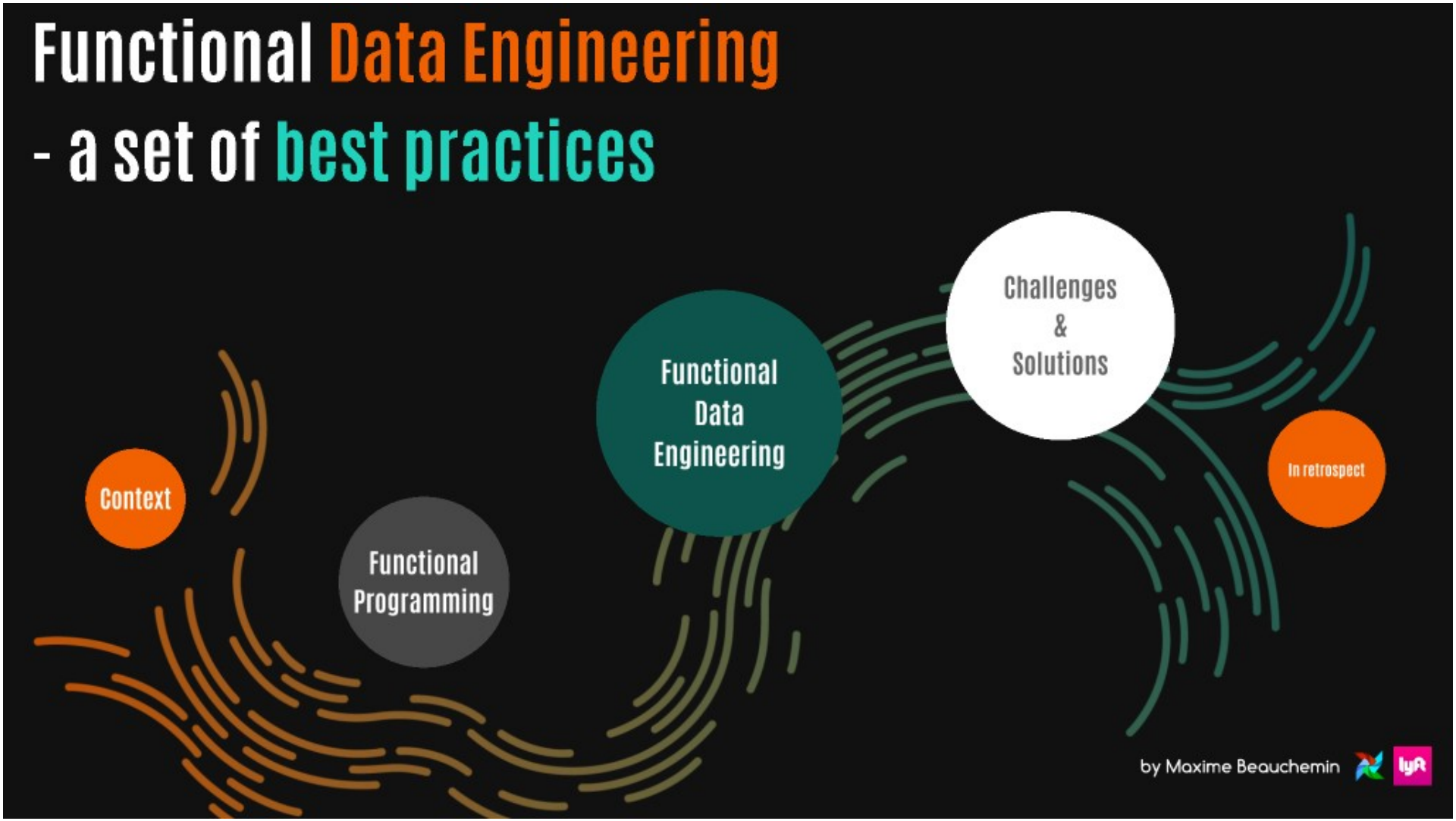
A Powdery Surface Is Closely Explored

By JOHN NOBLE WILFORD
Special to The New York Times

HOUSTON, Monday, July 21—Men have landed and walked on the moon.

Two Americans, astronauts of Apollo 11, steered their fragile four-legged lunar module safely and smoothly to the historic landing yesterday at 4:17:40 P.M., Eastern daylight time.

Neil A. Armstrong, the 38-year-old civilian commander, radioed to earth and the mission control room here:


Functional Data Engineering
- a set of best practices
Context
Functional Programming
Functional Data Engineering
Challenges & Solutions
In retrospect
by Maxime Beauchemin

Functional Programming

Pure funtions

Immutability

Idempotency

Functional programming

From Wikipedia, the free encyclopedia

For subroutine-oriented programming, see Procedural programming.

In computer science, **functional programming** is a programming paradigm—a style of building the structure and elements of computer programs—that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data. It is a declarative programming paradigm, which means programming is done with expressions[1] or declarations[2] instead of statements. In functional code, the output value of a function depends only on the arguments that are passed to the function, so calling a function *f* twice with the same value for an argument *x* produces the same result *f(x)* each time; this is in contrast to procedures depending on a local or global state, which may produce different results at different times when called with the same arguments but a different program state. Eliminating side effects, i.e., changes in state that do not depend on the function inputs, can make it much easier to understand and predict the behavior of a program, which is one of the key motivations for the development of functional programming.

Pure functions

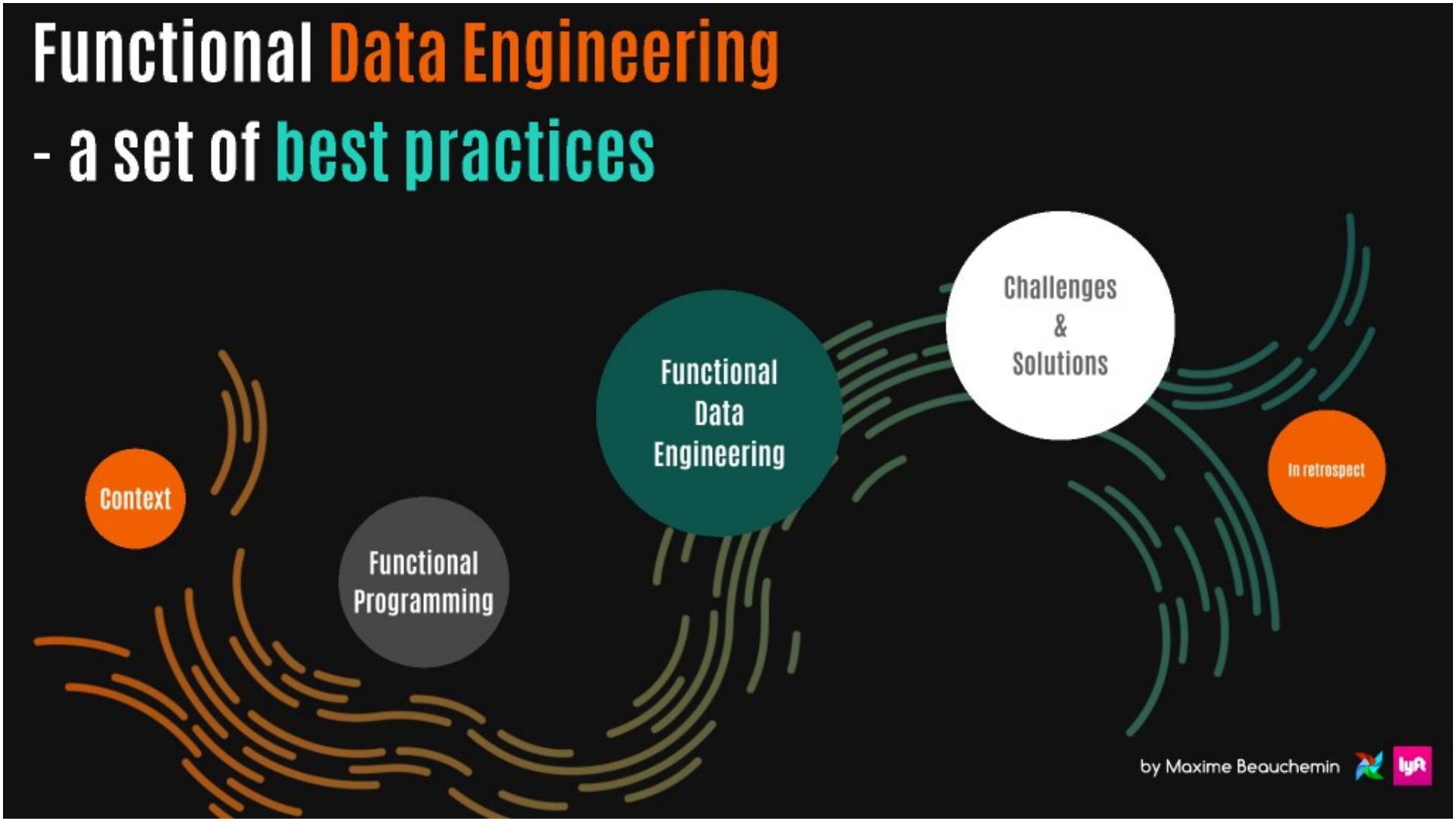
```
def pure_add_one(i):
    return i + 1

counter = 0
def impure_add_one():
    counter += 1
    return counter
```

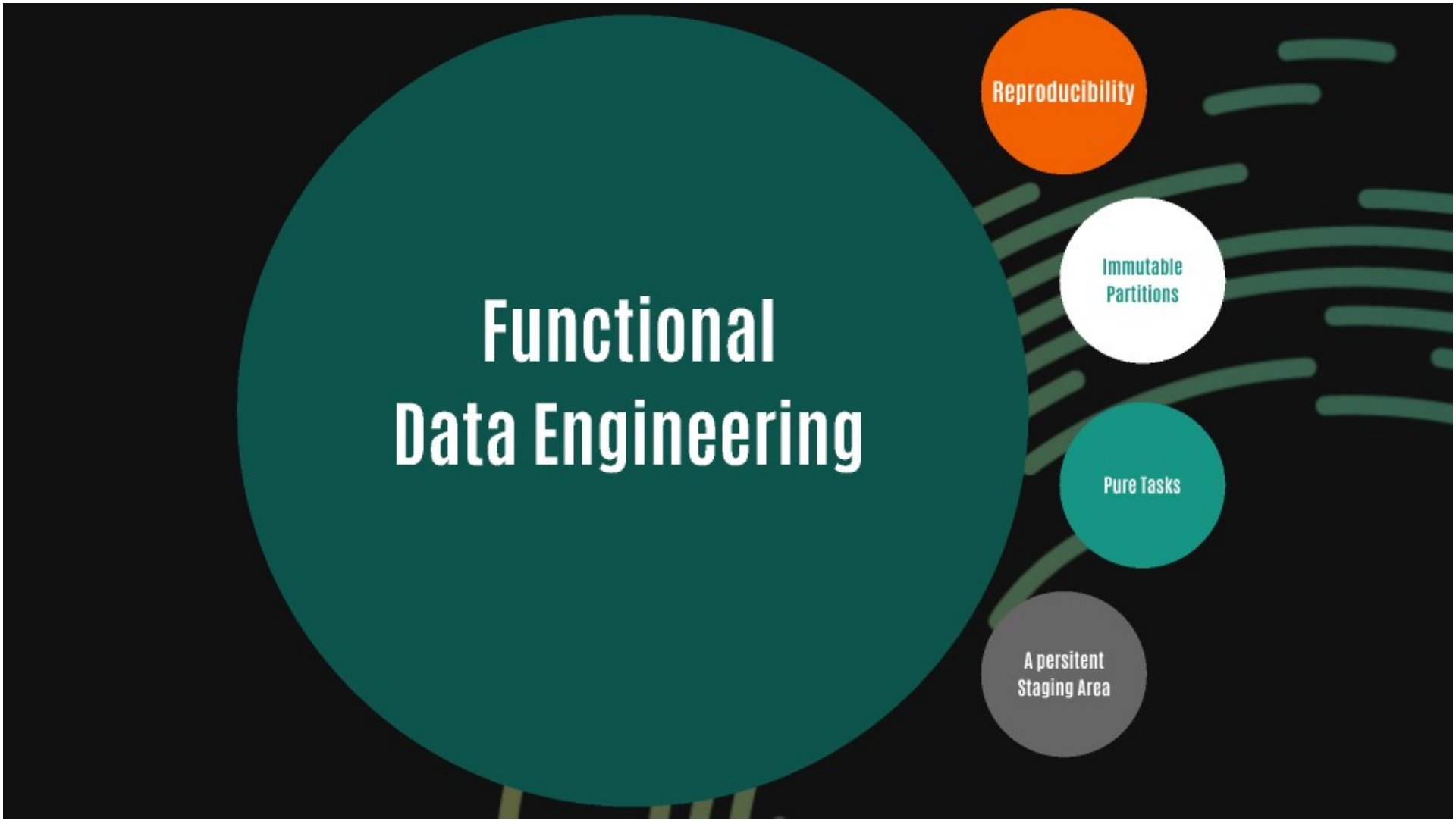
- Given the same input, will give the same output
- Can easily be unit tested
- Easier to reason about
- Brings clarity


Immutability

Idempotency

Idempotence: is the property of certain operations in mathematics and computer science that they can be applied multiple times without changing the result beyond the initial application. The concept of idempotence arises in a number of places in abstract algebra (in particular, in the theory of projectors and closure operators) and functional programming (in which it is connected to the property of referential transparency).


Functional Data Engineering
- a set of best practices
Context
Functional Programming
Functional Data Engineering
Challenges & Solutions
In retrospect
by Maxime Beauchemin

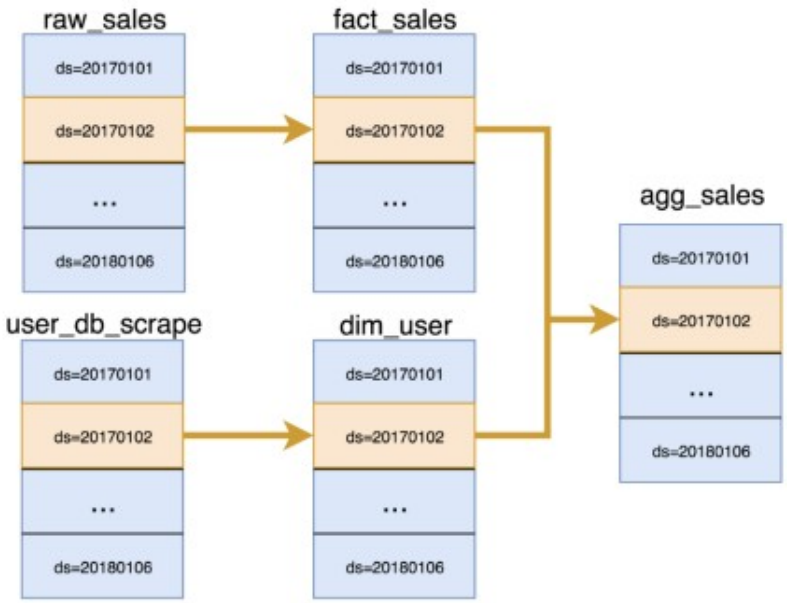

Functional
Data Engineering
Reproducibility
Immutable
Partitions
Pure Tasks
A persitent
Staging Area

Reproducibility

- Foundational to the scientific method
- Critical from a legal standpoint
- Critical from a sanity standpoint
- The functional approach guarantees reproducibility

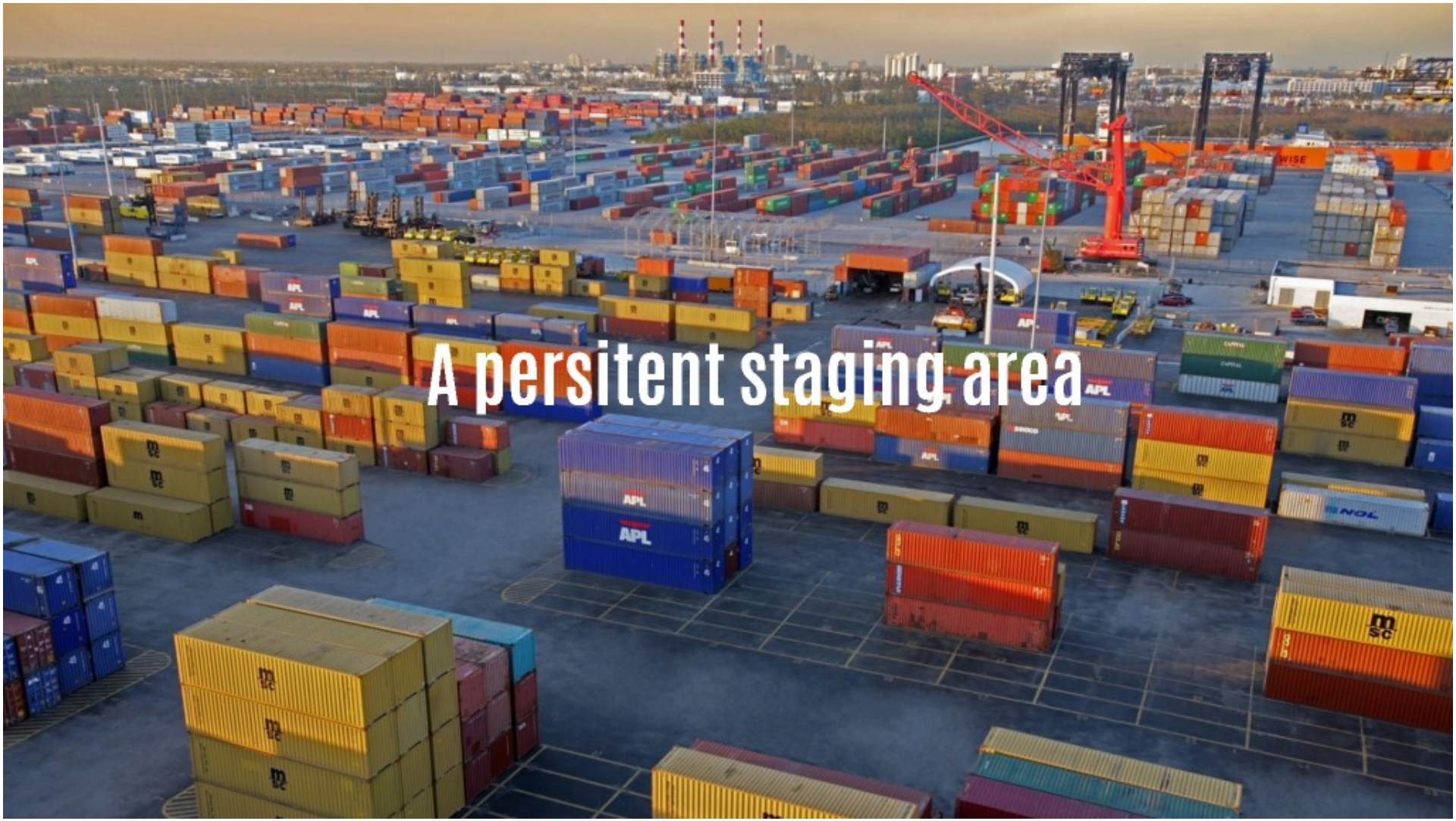
Immutable partitions

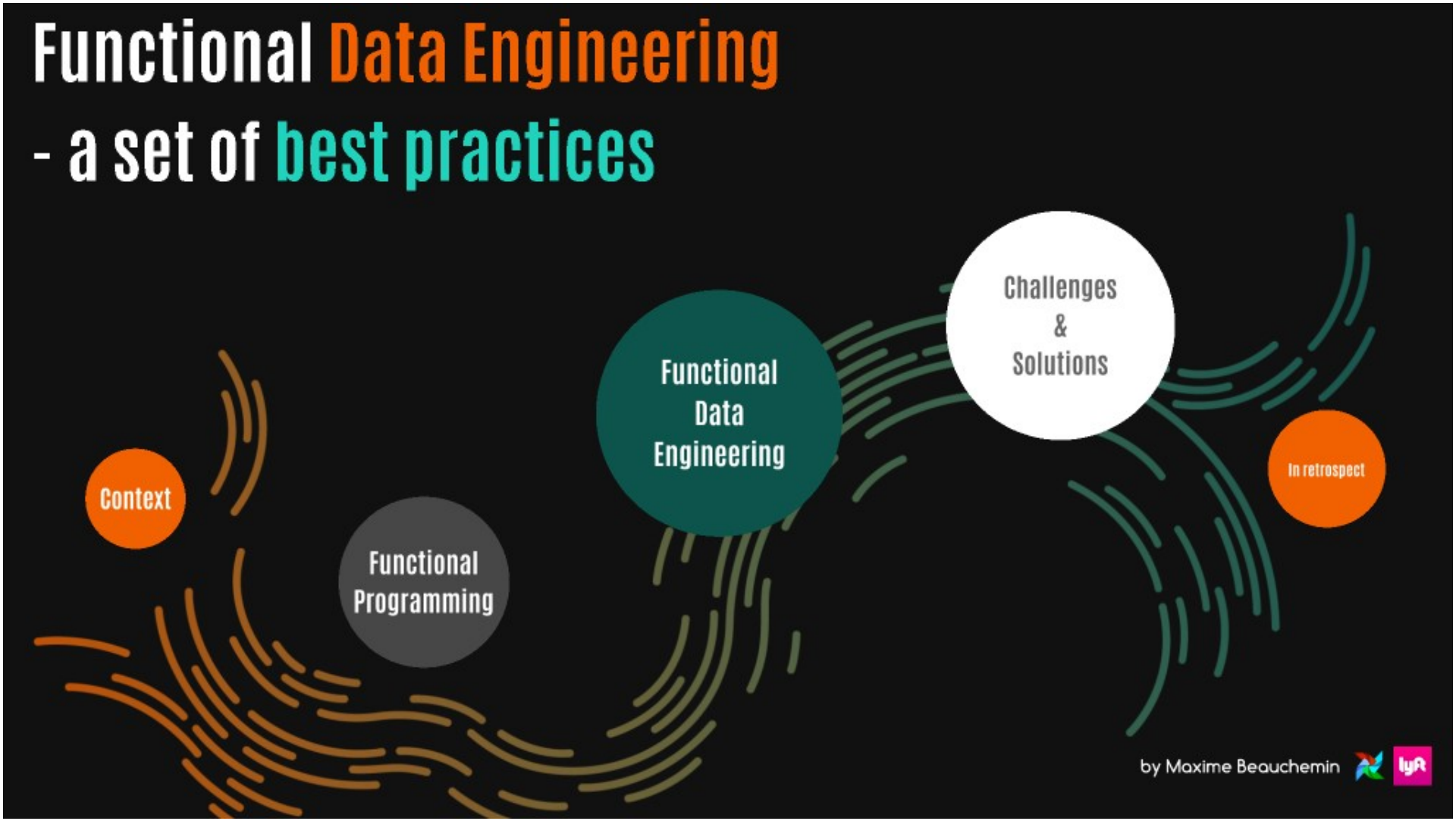
- partitions are the equivalent of immutable objects
- partition ALL TABLES
- partitions align with ETL schedules
- write once, read multiple times
- ETL becomes a lineage DAG of partitions



Pure ETL Tasks

- are idempotent
- determinitic
- have no side-effects
- use immutable sources
- usually target a single partition
- Never UPDATE, UPSERT, APPEND, DELETE (mutations)
- Limit the number of source partitions (no wide-scans)

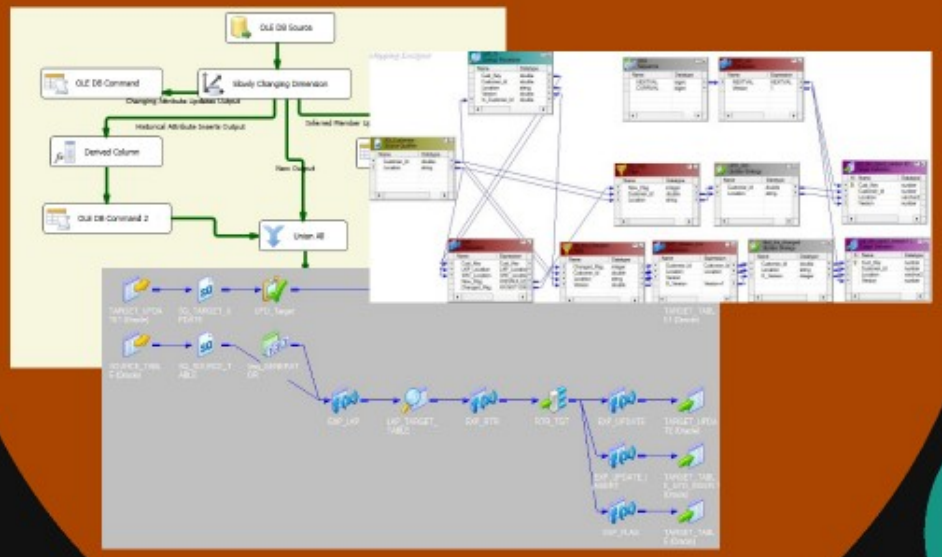

A persitent staging area


Functional Data Engineering
- a set of best practices
Context
Functional Programming
Functional Data Engineering
Challenges & Solutions
In retrospect
by Maxime Beauchemin


Common Chanllenges
&
Solutions
SCD
Late Arriving Data
Self or Past Dependencies
File Explosion

"Slowly Changing Dimension"

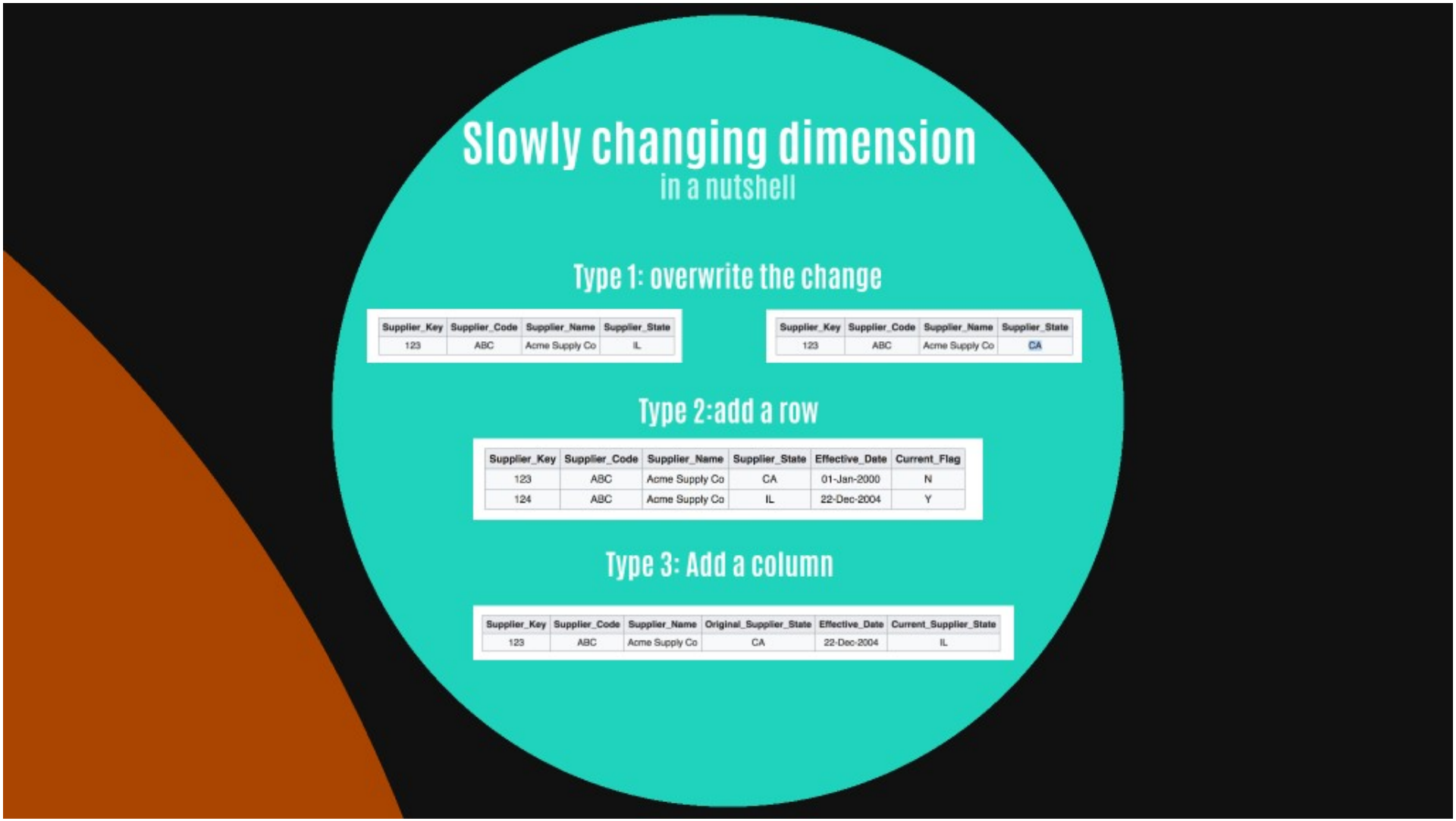
or how to go crazy while mutating mutations



WTF?

Why do SCD sucks?

The functional approach


Slowly changing dimension
in a nutshell
Type 1: overwrite the change
Supplier_Key Supplier_Code Supplier_Name Supplier_State
123 ABC Acme Supply Co IL
Supplier_Key Supplier_Code Supplier_Name Supplier_State
123 ABC Acme Supply Co CA
Type 2:add a row
Supplier_Key Supplier_Code Supplier_Name Supplier_State Effective_Date Current_Flag
123 ABC Acme Supply Co CA 01-Jan-2000 N
124 ABC Acme Supply Co IL 22-Dec-2004 Y
Type 3: Add a column
Supplier_Key Supplier_Code Supplier_Name Original_Supplier_State Effective_Date Current_Supplier_State
123 ABC Acme Supply Co CA 22-Dec-2004 IL

The shortcomings of SCD methodolgy

- Type 1
  - Mutations! history is lost
  - No reproducibility!
- Type 2
  - Hard to create/maintain on the dimension side
  - Make facts harder to manage (surrogate key lookups)
  - Reproducibility is challenging, when possible
- Type 3
  - A bad compromise
- Type N
  - Typically a combination of the issues above


SNAPSHOT
ALL THE
DATA!
-- With current attribute
SELECT *
FROM fact a
JOIN dimension b ON
a.dim_id = b.dim_id AND
date_partition = '{{ latest_partition('dimension') }}'
-- With historical attribute
SELECT *
FROM fact a
JOIN dimension b ON
a.dim_id = b.dim_id AND
a.date_partition = b.date_partition

Late arriving facts

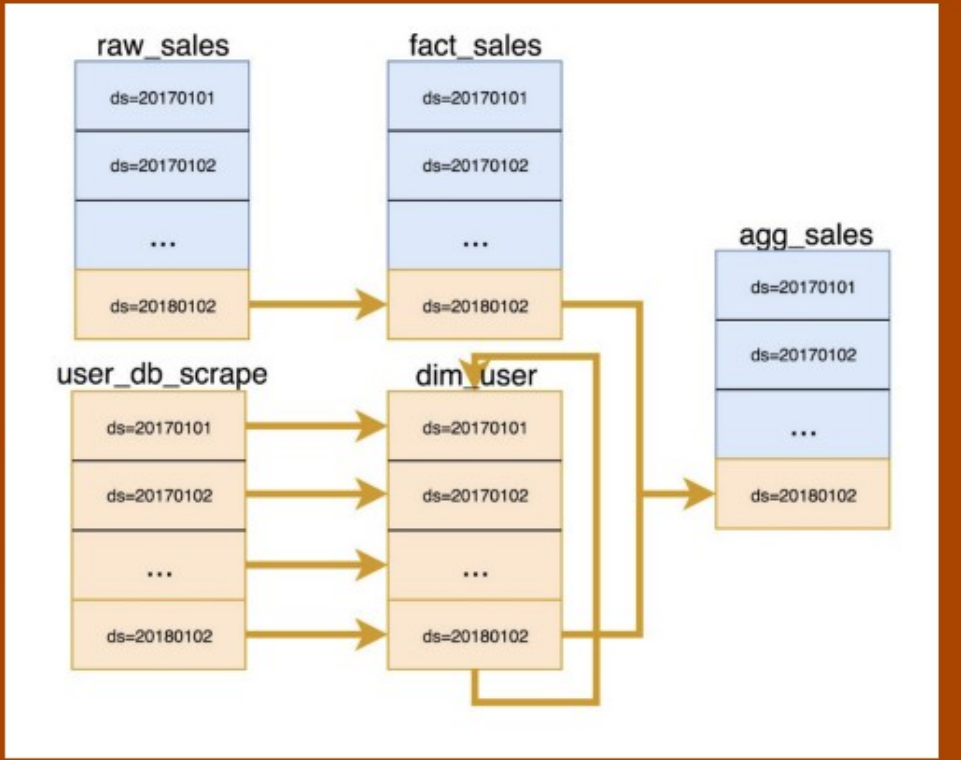
- Late arriving facts are common
- Two time dimensions: event processing time and event time
- Partition based on event processing time to close the loop on immutable blocks
- Partition pruning is lost when filtering on event time

Mitigation mechanisms

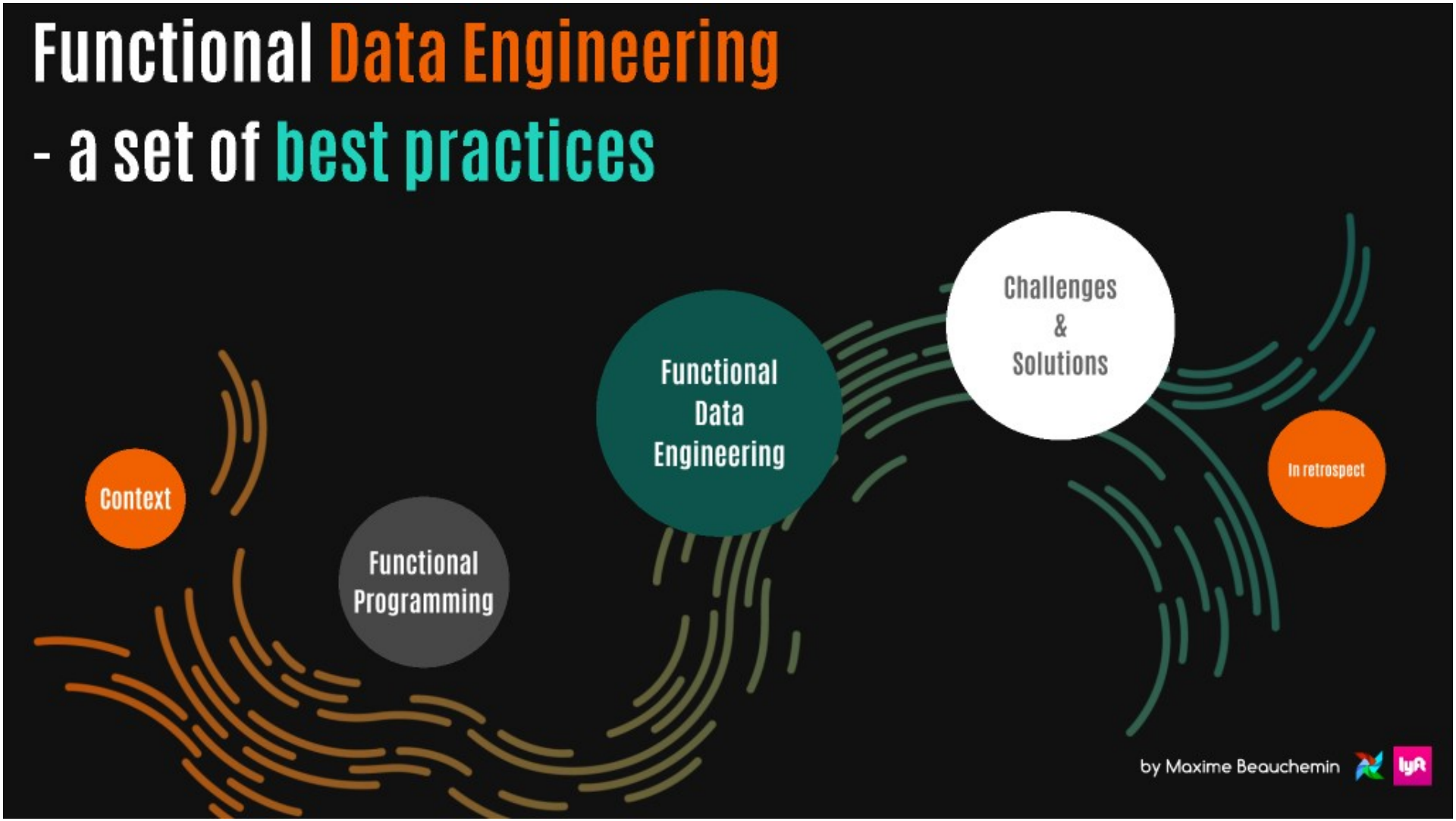
- rely on execution engine optmizations
- instruct people to apply predicates on partionned fields
- sub-partition on event time
- pivot on event time on fact tables

Self or past dependencies are bad!

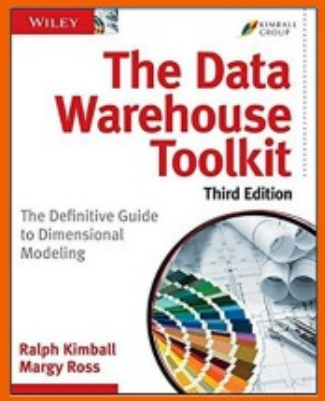
- Higher "complexity score"
- Makes parallelising backfills impossible
- Avoid metrics in dims when possible
- Relly on specialized frameworks for cumulative metrics

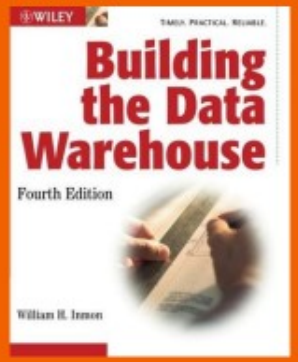



File explosion

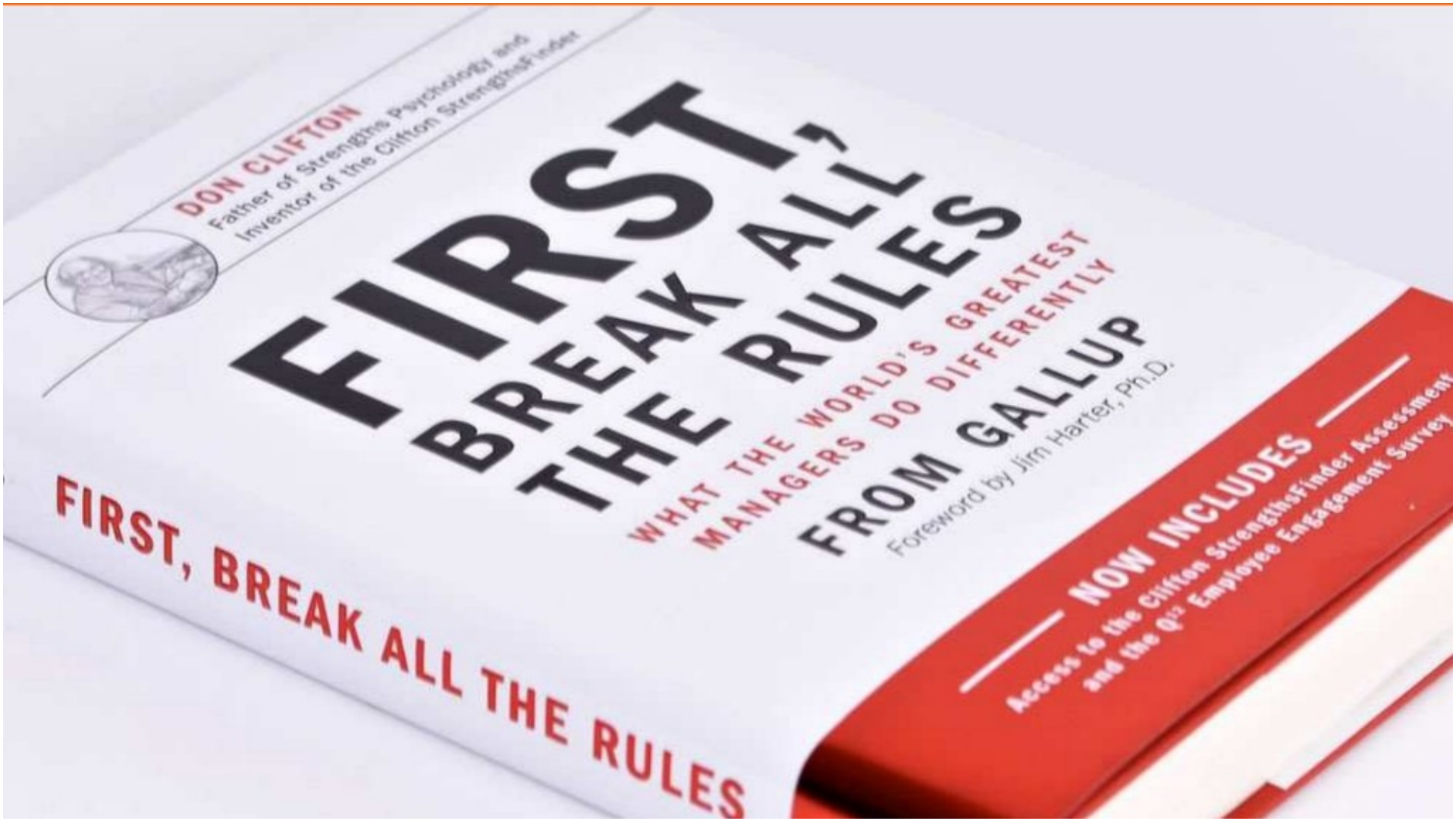

Functional Data Engineering
- a set of best practices
Context
Functional Programming
Functional Data Engineering
Challenges & Solutions
In retrospect
by Maxime Beauchemin

Times have changed

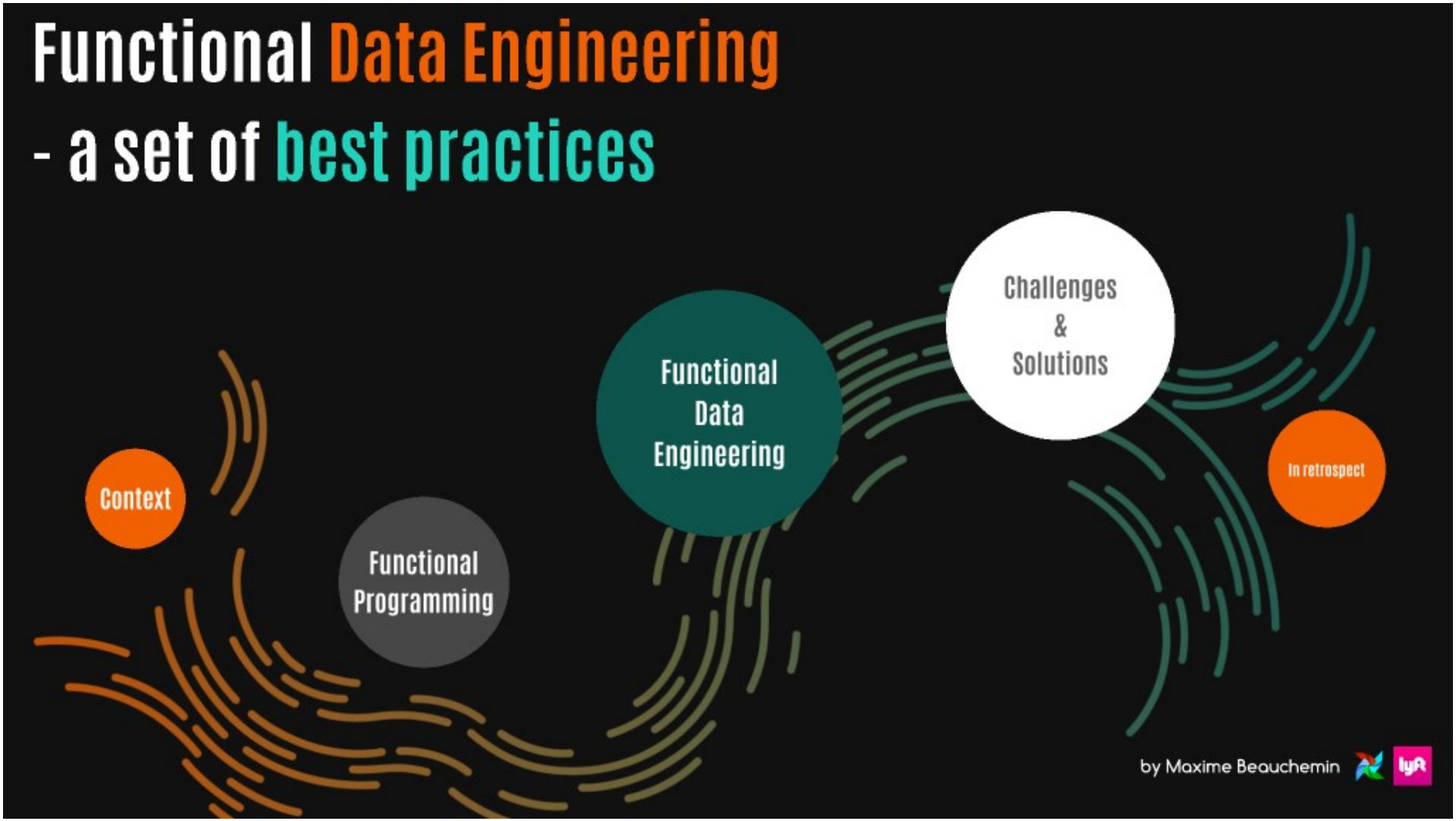




- cheap, limitless storage
- distributed databases
- the rise of read-optimized stores using immutable file formats (Parquet, ORC)
- large, decentralized data teams


DON CLIFTON
Father of Strengths Psychology and
Inventor of the Clifton StrengthsFinder
FIRST, BREAK ALL THE RULES
WHAT THE WORLD'S GREATEST MANAGERS DO DIFFERENTLY
FROM GALLUP
Foreword by Jim Harter, Ph.D.
FIRST, BREAK ALL THE RULES
NOW INCLUDES
Access to the Clifton StrengthsFinder Assessment
and the Q12 Employee Engagement Survey


That's all Folks!


Functional Data Engineering
- a set of best practices
Context
Functional Programming
Functional Data Engineering
Challenges & Solutions
In retrospect
by Maxime Beauchemin