MUHAMMET ŞAFAK — VISUALS

An invoice's reliability

Number, write, repeat and outcome: an invoice can go wrong four times on its way.

Muhammet Şafak • October 3, 2026



SECTION 01

Number

In e-invoicing, a number must be both free of duplicates and free of gaps.

> Generate too late and you get a race; generate too early and you get a gap.

— **Muhammet Şafak** Number collision in an e-invoice integration

TWO CONSTRAINTS

Four approaches, two constraints

APPROACH	NO DUPLICATES	NO GAPS	FAILURE MODE
MAX()+1 at send time	No	Yes	Collision with parallel workers
Generate at enqueue	Yes	No	Failed or cancelled → gap
DB sequence	Yes	No	Rollback doesn't return the value → gap
JIT reservation	Yes	Yes	Cost: an early commit is mandatory

JIT RESERVATION

The number, right before sending

1 **Lock**

`lockForUpdate`

2 **Reserve**

`if reserved_no is empty, counter++`

3 **Early commit**

`status = SENDING`

4 **Integrator**

`outside the lock`

5 **Retry**

`same reserved_no`

> A lock is held for DB state, not for external I/O.

— **Muhammet Şafak** The integrator call takes 200 ms to 2 seconds · Engineering

SECTION 02

Delivery

The invoice goes to the database, the event to RabbitMQ: two systems, no single atomic step.

DUAL-WRITE

Writing to two systems

Dual-write — save() and publish() are separate

- ✕ Publish fails → lost event
- ✕ Rollback happens → phantom event
- ✕ DB::transaction does not cover RabbitMQ



Transactional outbox — The write drops to a single system

- ✓ **Invoice and outbox row in the same commit**
- ✓ **A separate process publishes to the broker**
- ✓ **Dual-write shrinks to a single write**

REPEAT

Repeat delivery must be harmless

	At-least-once only	Idempotent consumer
DELIVERY	At least once, sometimes twice	✓ At least once, sometimes twice
RECORD	Processed ids are not kept	✓ processed_messages, unique constraint
TRANSACTION	Work and stamp are separate	✓ Work and stamp in the same transaction
OUTCOME	A repeat repeats the effect	✓ Effectively-once

> Write the data to a single system, suppress repeats at the consumer, seal the sensitive field before it leaves.

— **Muhammet Şafak** Outbox, idempotent consumption and field encryption

SECTION 03

Outcome

The outbox guaranteed delivery; it did not guarantee the same outcome.

03

THE REAL EQUATION

outcome = f(message, hidden inputs)

Between production and consumption there can be milliseconds, twenty minutes or three days.

HIDDEN INPUTS

Six hidden inputs

INPUT	IN PRACTICE
Time	Reading the current clock for the document date
Database	Prefix X at creation, Y at send time
Configuration	The same allow-list in two separate services
External system	"This already exists", but what exists is unclear
Environment	One framework swallows the warning, another turns it into an exception
Worker	Parallel replicas, locks, repeat delivery

A NOISY FAILURE BEATS A SILENT WRONG OUTCOME

Four moves

1

Freeze the context

at production time

2

Time is an input

a retry does not change it

3

Name the fork

write the reason to the log

4

Don't resolve silently

stop on ambiguity

Thank you

muhammetsafak.com

What enters the decision is frozen; what doesn't is carried by reference.

