Go-Food

Changing Car’s Engine While on a Race

Bergas Bimo Branarto
Software Engineer

Covering:

- Go-Jek: multiple moving parts across system, high speed
- Initial Go-Food architecture
- Splitting Go-Food OMS from monolith system: Why, How, Result
- Lesson learnt

Go-Jek: multiple moving parts

- Lots of new products and features across products
- Lots of new services: on-going monolith migrations and/or scaling other services

Why did we need to change Go-Food engine?

- Scalability and reliability
- New feature development time
- Enabling easier parallel development on 4 streams: resto, discovery, discounting, international

Go-Food Order Flow Components & Dependencies - Initial

(part of) StanMarsh

Pre-Order	Order State Flow				Order(s) View	
Price Estimation	Create Order	Order Pickup	Order Cancellation	Order Completion	Order History	Order Detail
pricing platform	pricing platform	pricing platform	driver platform	driver platform	order history	driver platform
gopay-balance	allocation	gopay-balance	gopay-balance	gopay-balance		
gopay-voucher	driver platform	gopay-voucher	gopay-voucher	gopay-voucher		
	gopay-balance	goresto	notification	notification		
	gopay-voucher	notification	termination service	termination service		
	gofood cms					
	goresto					

Supported Payment Types:
Cash, GoPay (+Partial Payment), Vouchers

Supported Merchant Type:
Non-partner, partner-non goresto, partner-goresto

Go-Food Order Flow Components & Dependencies

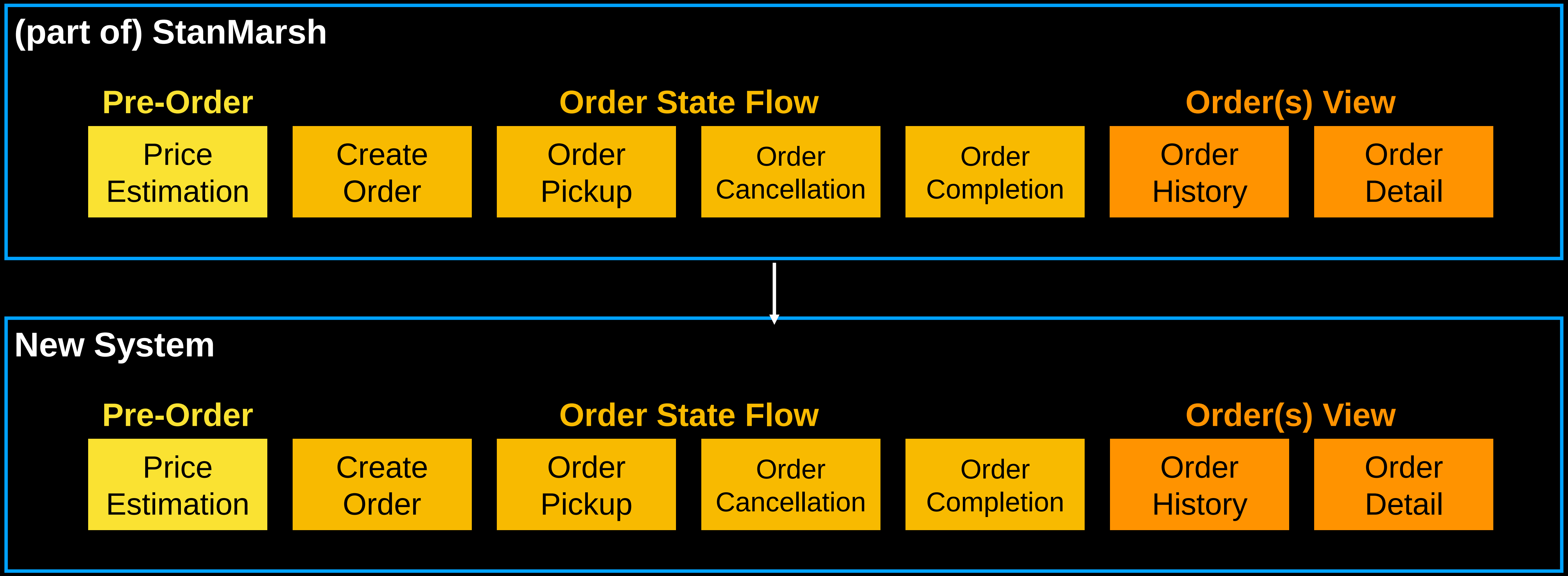
- Many dependencies
- Some dependencies are also in progress of rewrites: API contracts are moving targets
- Tricky implementation especially in GoPay integration: different DC: network unreliability is high

pricing platform	pricing platform	pricing platform	driver platform	driver platform	order history	driver platform
gopay-balance	allocation	gopay-balance	gopay-balance	gopay-balance		
gopay-voucher	driver platform	gopay-voucher	gopay-voucher	gopay-voucher		
	gopay-balance	goresto	notification	notification		
	gopay-voucher	notification	termination service	termination service		
	gofood cms					
	goresto					

Supported Payment Types:
Cash, GoPay (+Partial Payment), Vouchers

Supported Merchant Type:
Non-partner, partner-non goresto, partner-goresto

Go-Food OMS Rewrite Strategy



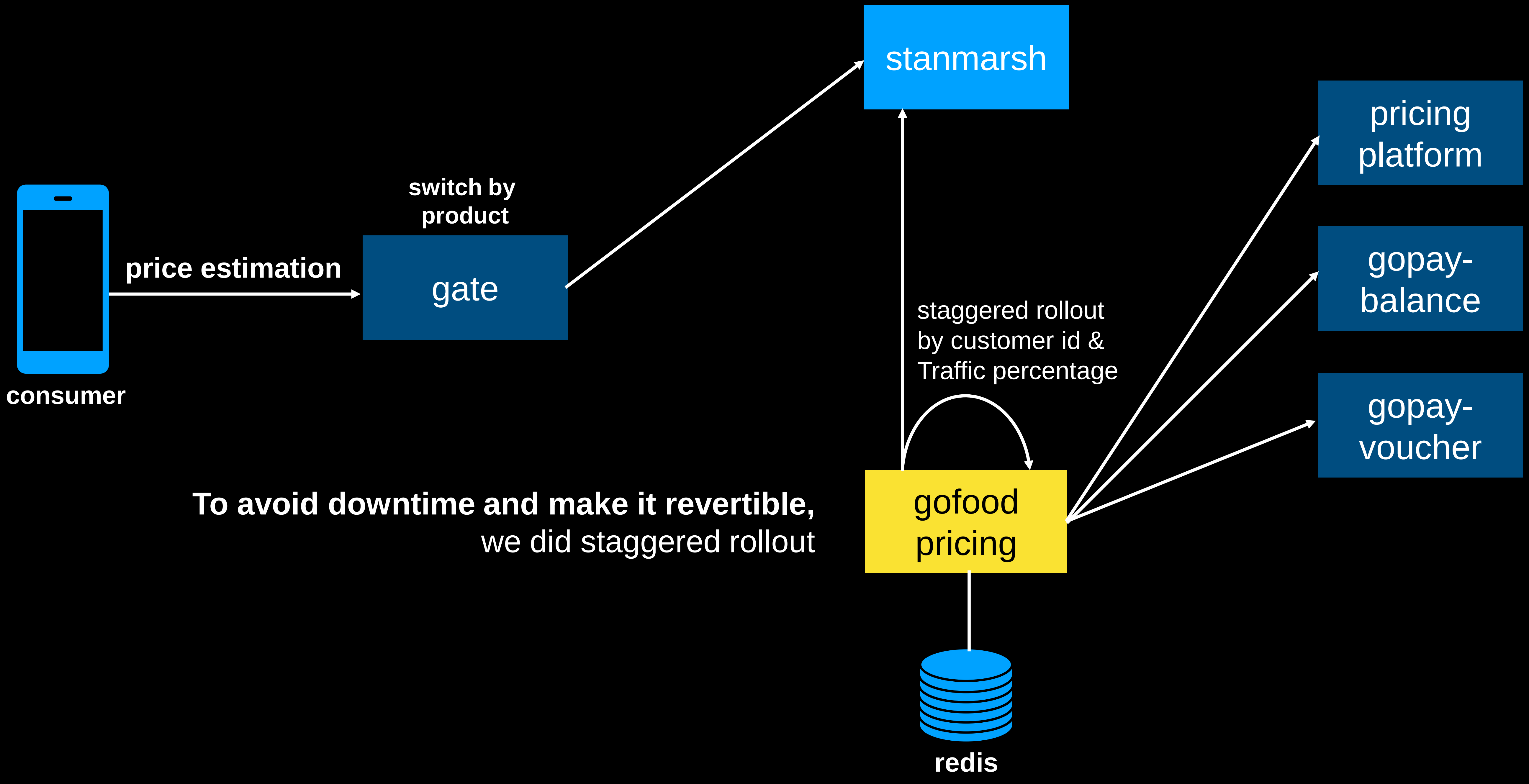
Constraints

1. Fast iteration
2. Avoid changes in StanMarsh (as much as we can)
3. 0 dependency to StanMarsh DB
4. Reusable components (gojek platform)
5. 0 downtime and revertible

For **first iteration,** we picked a **component which (relatively) independent** to the others -> **Price Estimation**

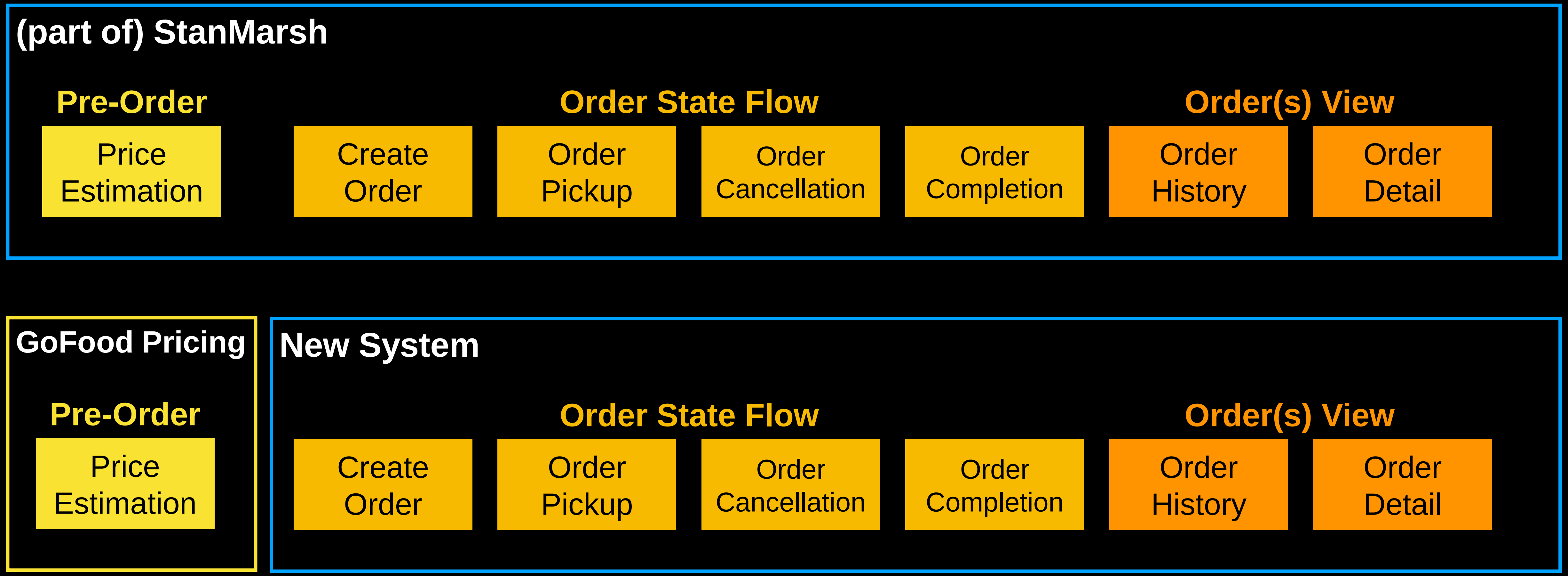
Price Estimation - iteration #1

Goal:
0 RPM on stanmarsh’s go-food price estimation API
gofood pricing as source of truth for Go-Food price calculations



To avoid downtime and make it revertible,
we did staggered rollout

Go-Food OMS Rewrite Strategy - iteration #1 final



To keep ourself **focus** and easier to maintain **data consistency,**
we migrated **1 order state flow in one go**.

HOW TO MANAGE SO MANY DEPENDENCIES
WHILE STILL ACHIEVING FAST ITERATION?

Go-Food OMS Rewrite Strategy - iteration #2 preparing

(part of) StanMarsh

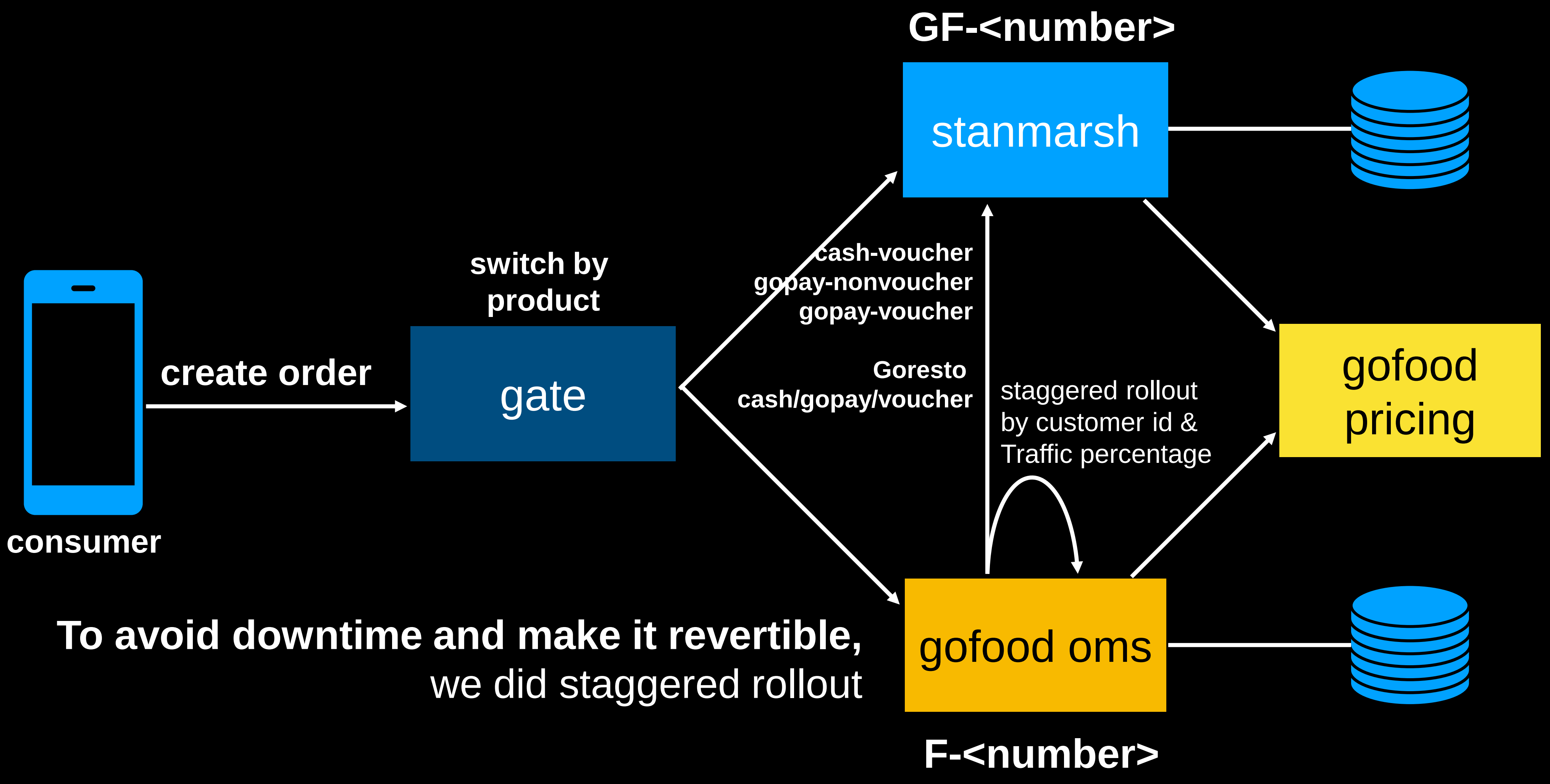
Order State Flow				Order(s) View	
Create Order	Order Pickup	Order Cancellation	Order Completion	Order History	Order Detail
pricing platform	pricing platform	driver platform	driver platform	order history	driver platform
allocation	gopay-balance	gopay-balance	gopay-balance		
driver platform	gopay-voucher	gopay-voucher	gopay-voucher		
gopay-balance	notification	notification	notification		
gopay-voucher	goresto	termination service	termination service		
goresto					

To iterate fast, we removed some dependencies
Flow with minimum dependencies is……

Cash - NonVoucher - NonGoResto Order

Create Order - iteration #2 preparing

Goal:
0 RPM on stanmarsh’s go-food **cash non-voucher non-goresto** order creation
0 records of Go-Food cash non-voucher non-goresto order in stanmarsh DB
Create **structure** of the new Go-Food order management system

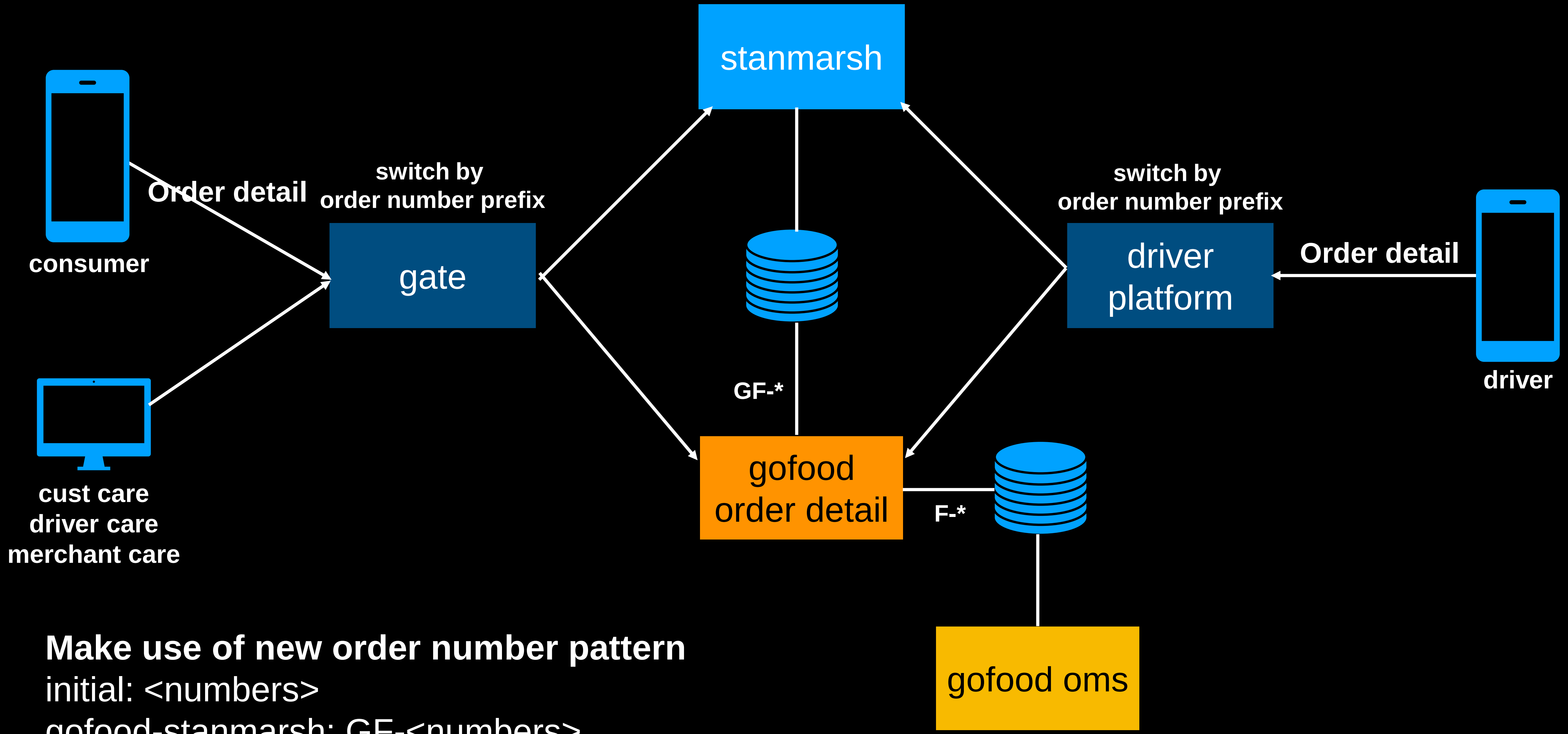


To avoid downtime and make it revertible,
we did staggered rollout

IT’S USING A DIFFERENT DB,
HOW SHALL THE USERS SEE ORDER DETAILS?

Order Detail - iteration #2

Goal:
0 RPM on stanmarsh's go-food order detail APIs
New Gofood Order Detail as source of truth for **all** gofood orders details



Make use of new order number pattern
initial: <numbers>
gofood-stanmarsh: GF-<numbers>
gofood-final: F-<numbers>

Go-Food OMS Rewrite - iteration #2 final

(part of) StanMarsh

Order State Flow

- Create Order
- Order Pickup
- Order Cancellation
- Order Completion

Order(s) View

- Order Detail
- Order History

GoFood Pricing

Pre-Order

- Price Estimation

LET'S
CONTINUE
TO
CASH
NON-VOUCHER
NON-GORESTO
ORDER
FLOW!
ITERATION #3

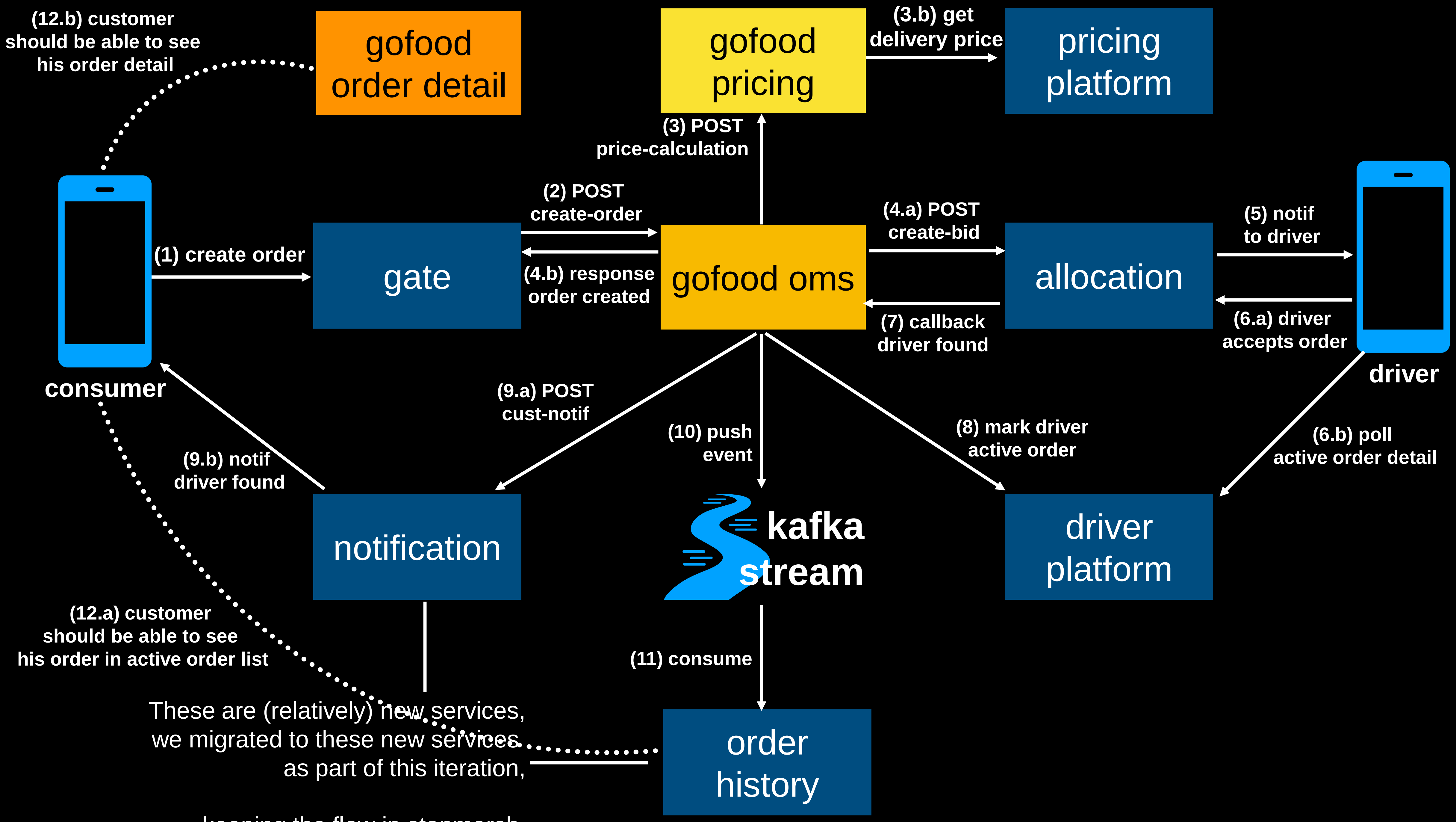
Order(s) View

GoFood Order Detail

- Order Detail

Create Order Cash NonVoucher Non-Goresto - iteration #3

Goal:
Create **structure** of the new Go-Food order management system

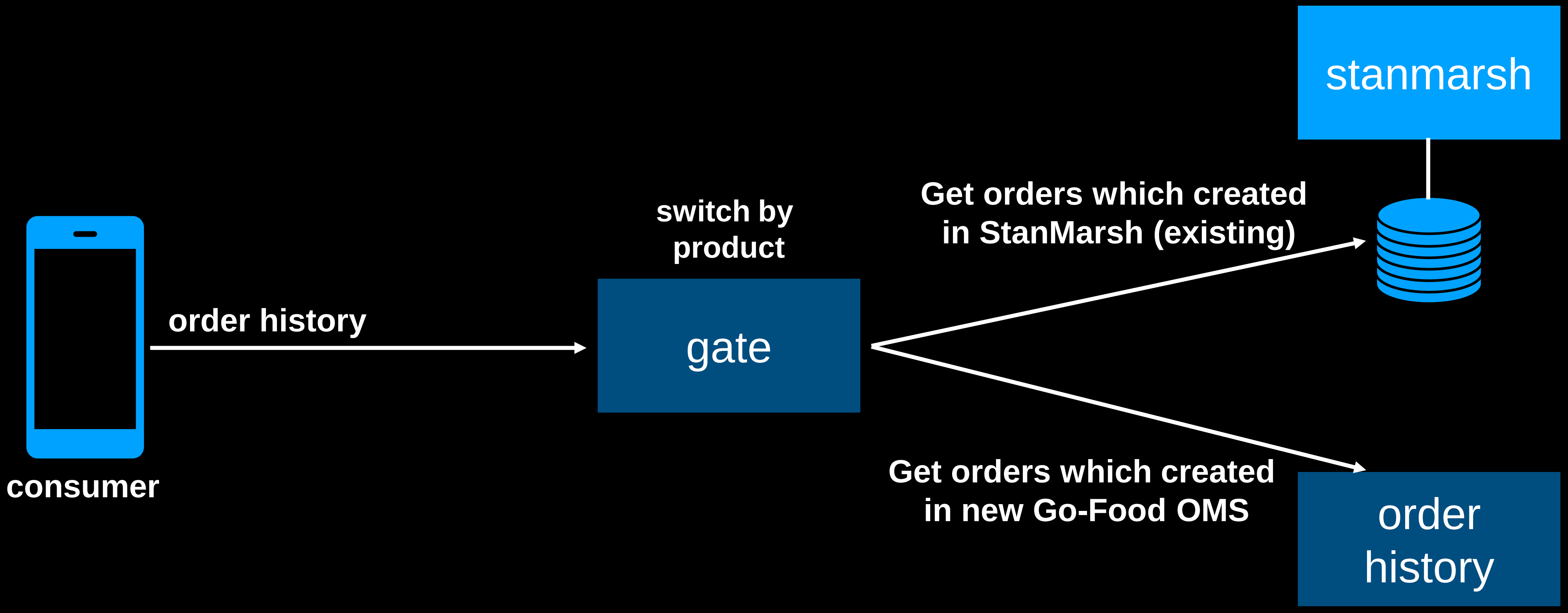


These are (relatively) new services,
we migrated to these new services
as part of this iteration,

keeping the flow in stanmarsh
to use the old one

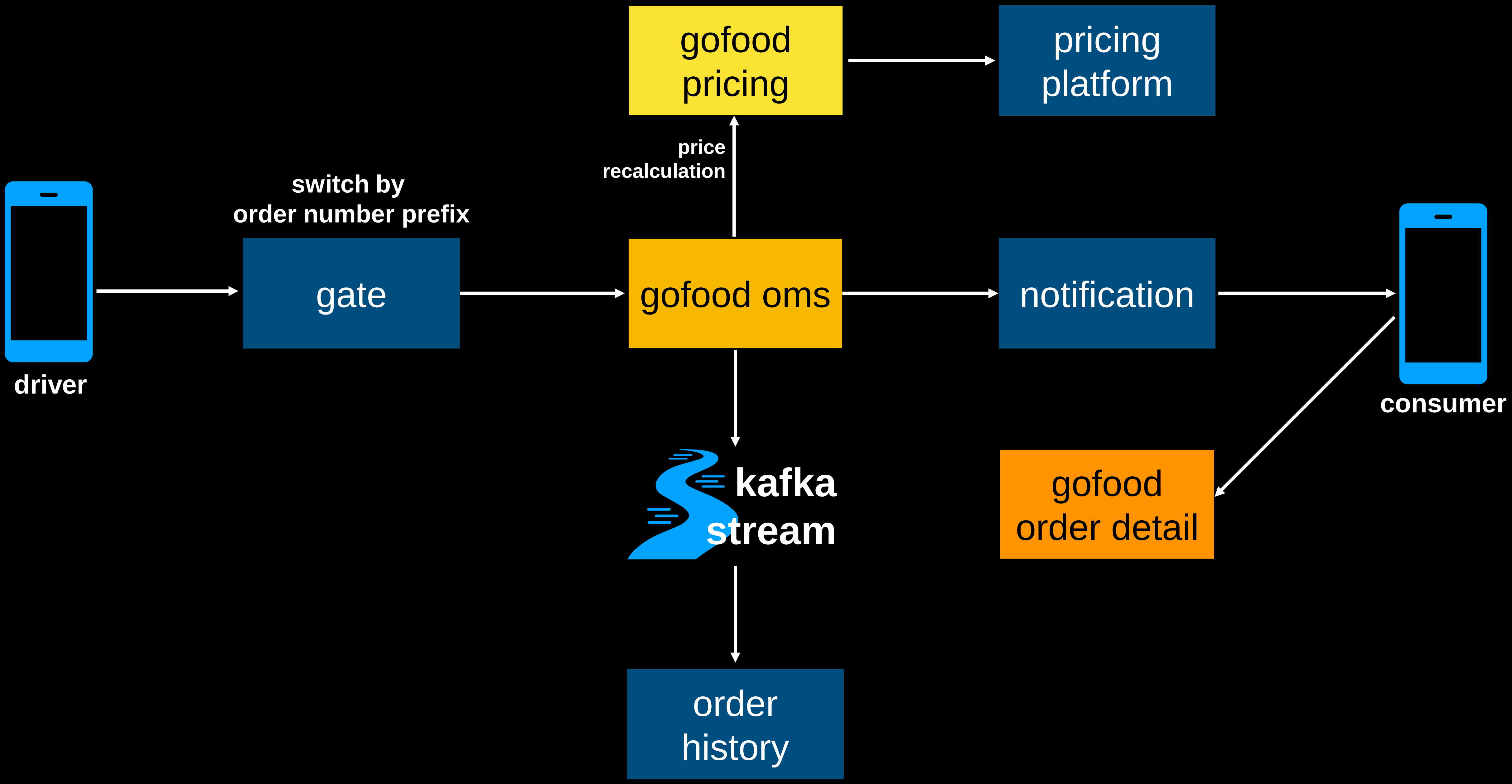
Order History - iteration #3

Goal:
0 RPM on stanmarsh's go-food order history APIs
Use history service as source of truth for customer order history


stanmarsh
switch by product
Get orders which created in StanMarsh (existing)
order history
gate
consumer
Get orders which created in new Go-Food OMS
order history

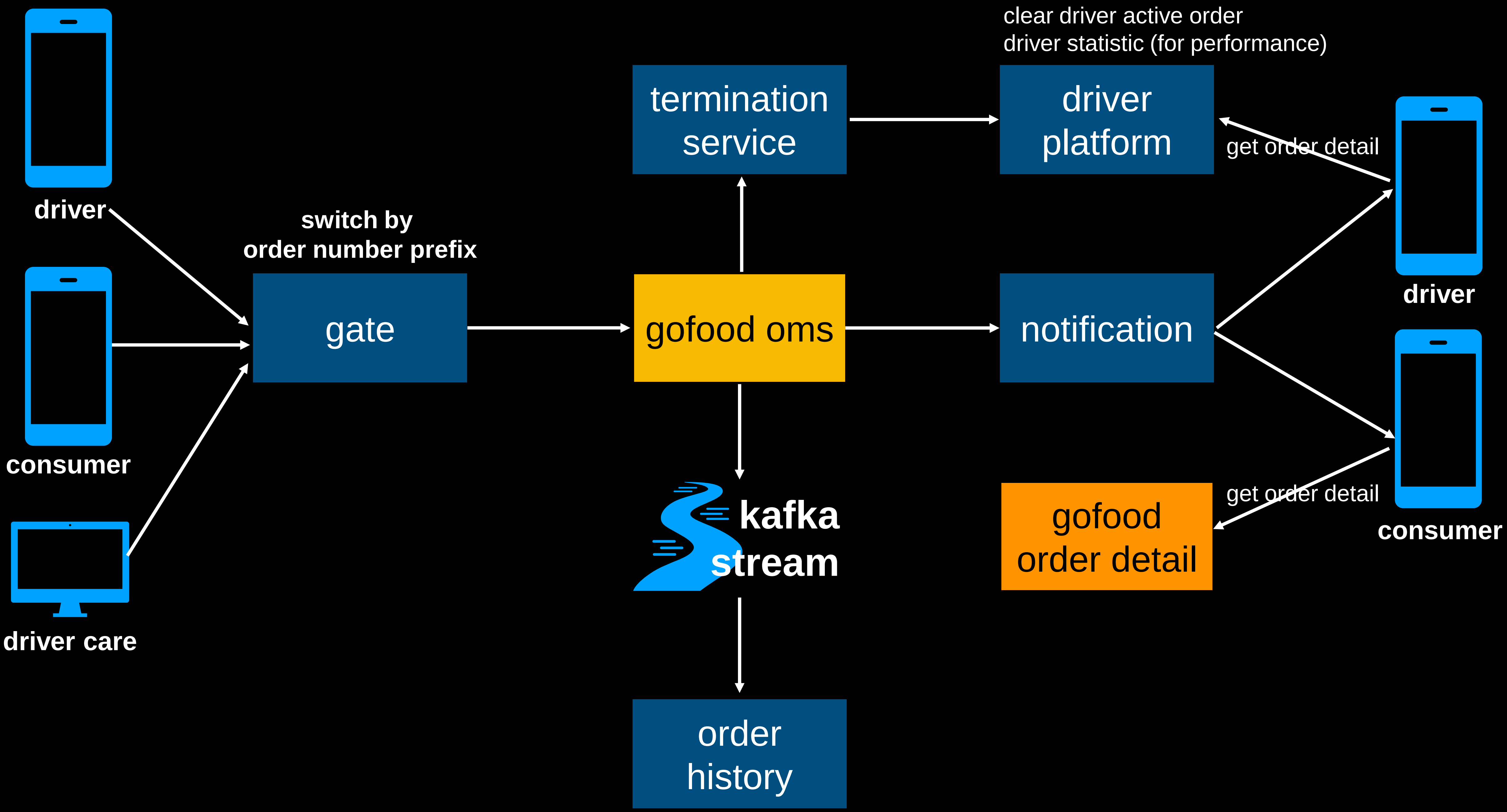
Order Pickup Cash NonVoucher NonGoresto - iteration #3

Goal:
0 RPM on stanmarsh's go-food order pickup APIs



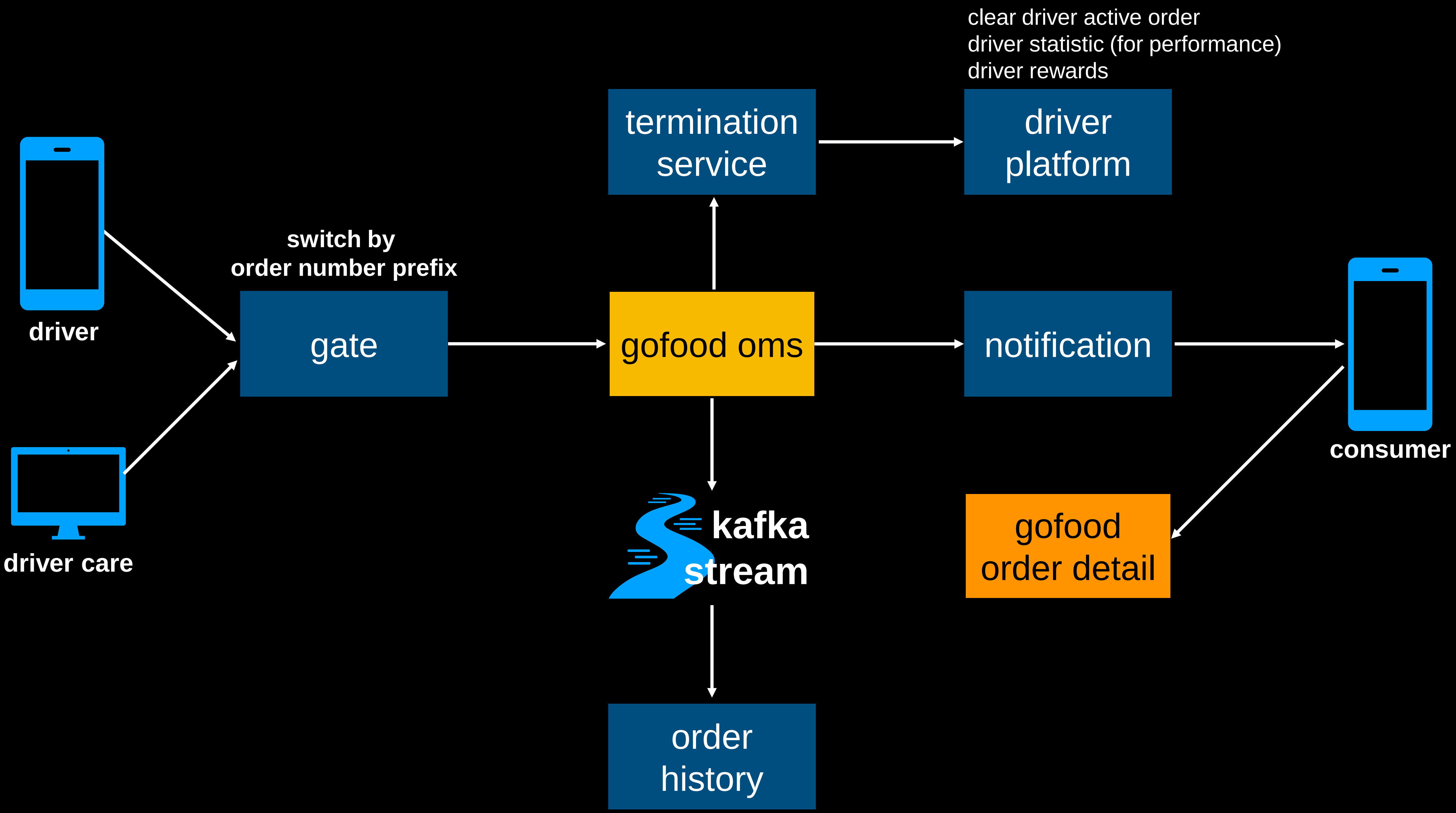
Order Cancellation Cash NonVoucher Non-Goresto - iteration #3

Goal:
0 RPM on stanmarsh's go-food order cancellation APIs
OMS as order state manager and orchestrator

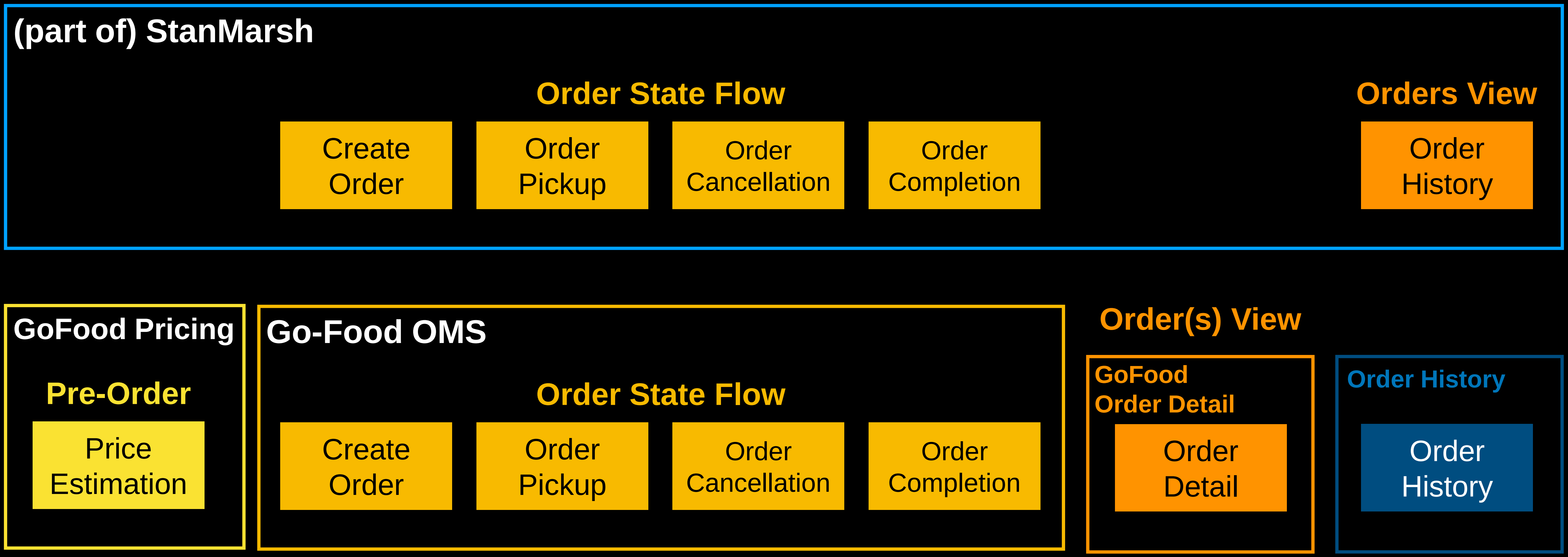


Order Completion Cash NonVoucher Non-Goresto - iteration #3

Goal:
0 RPM on stanmarsh's go-food order completion APIs
OMS as order state manager and orchestrator



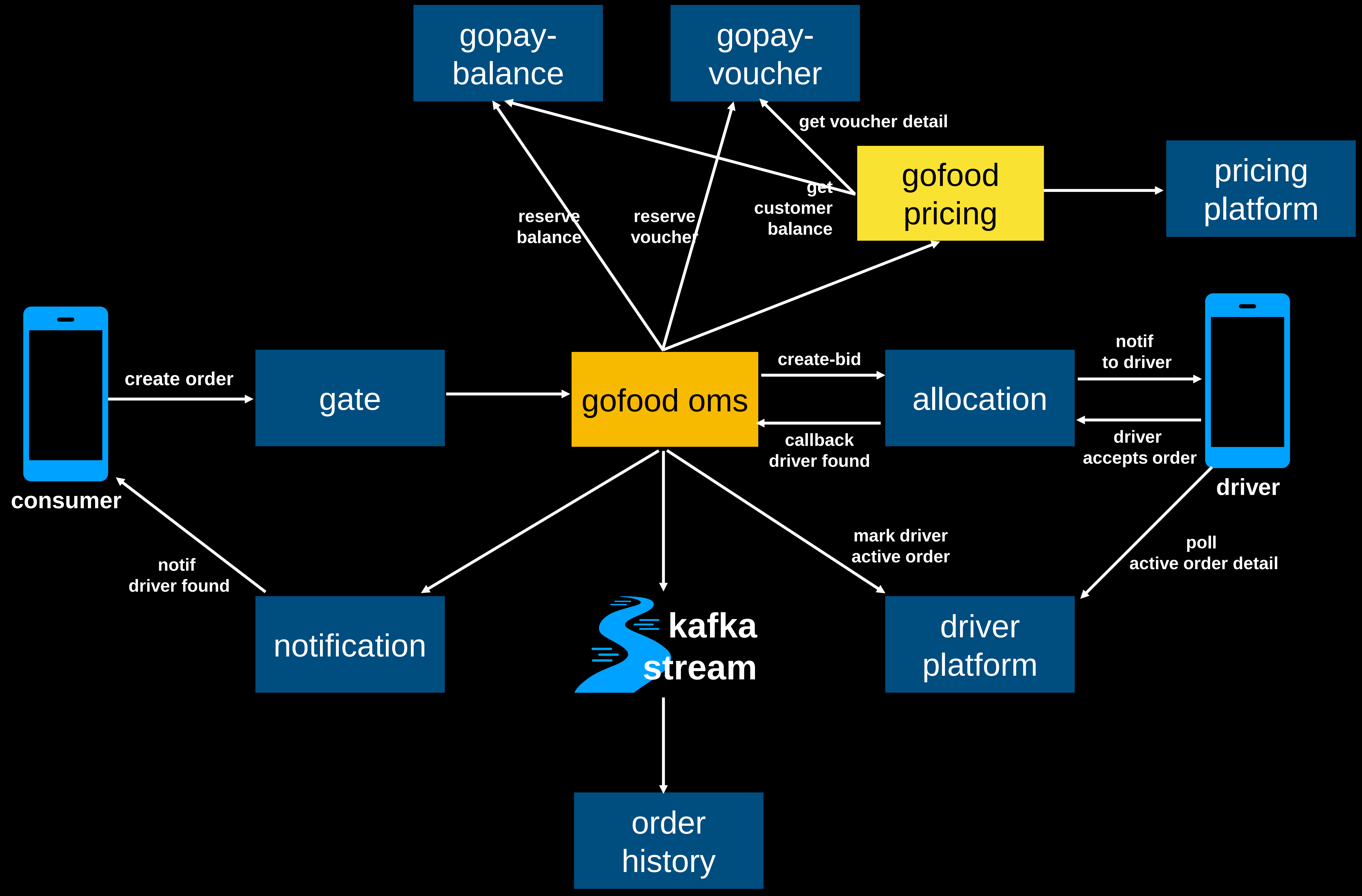
Go-Food OMS Rewrite - iteration #3 final



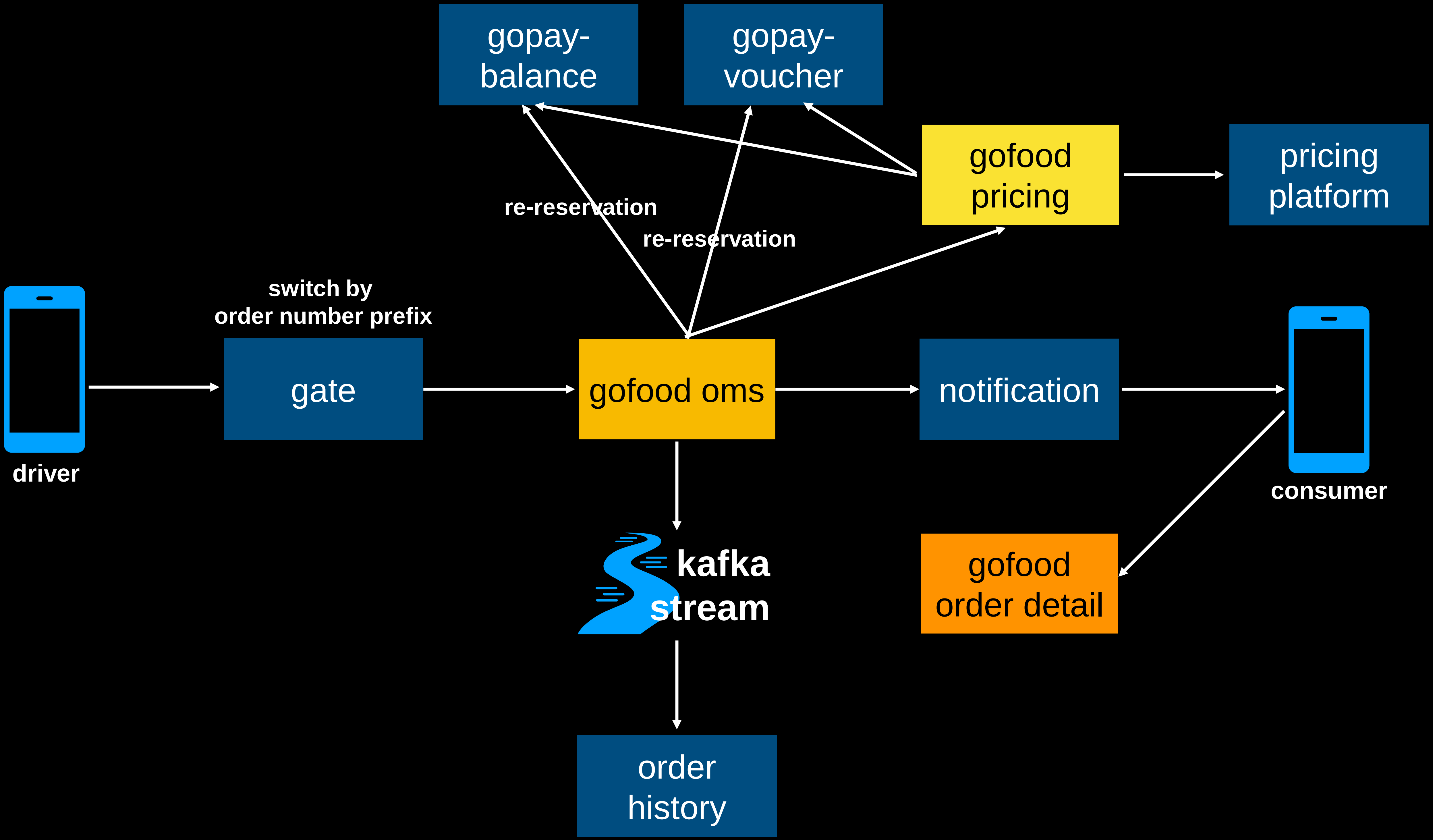
cash non-voucher non-goresto flow is covered

NEXT:
GOPAY VOUCHER NON-GORESTO
ORDER FLOW!
ITERATION #4

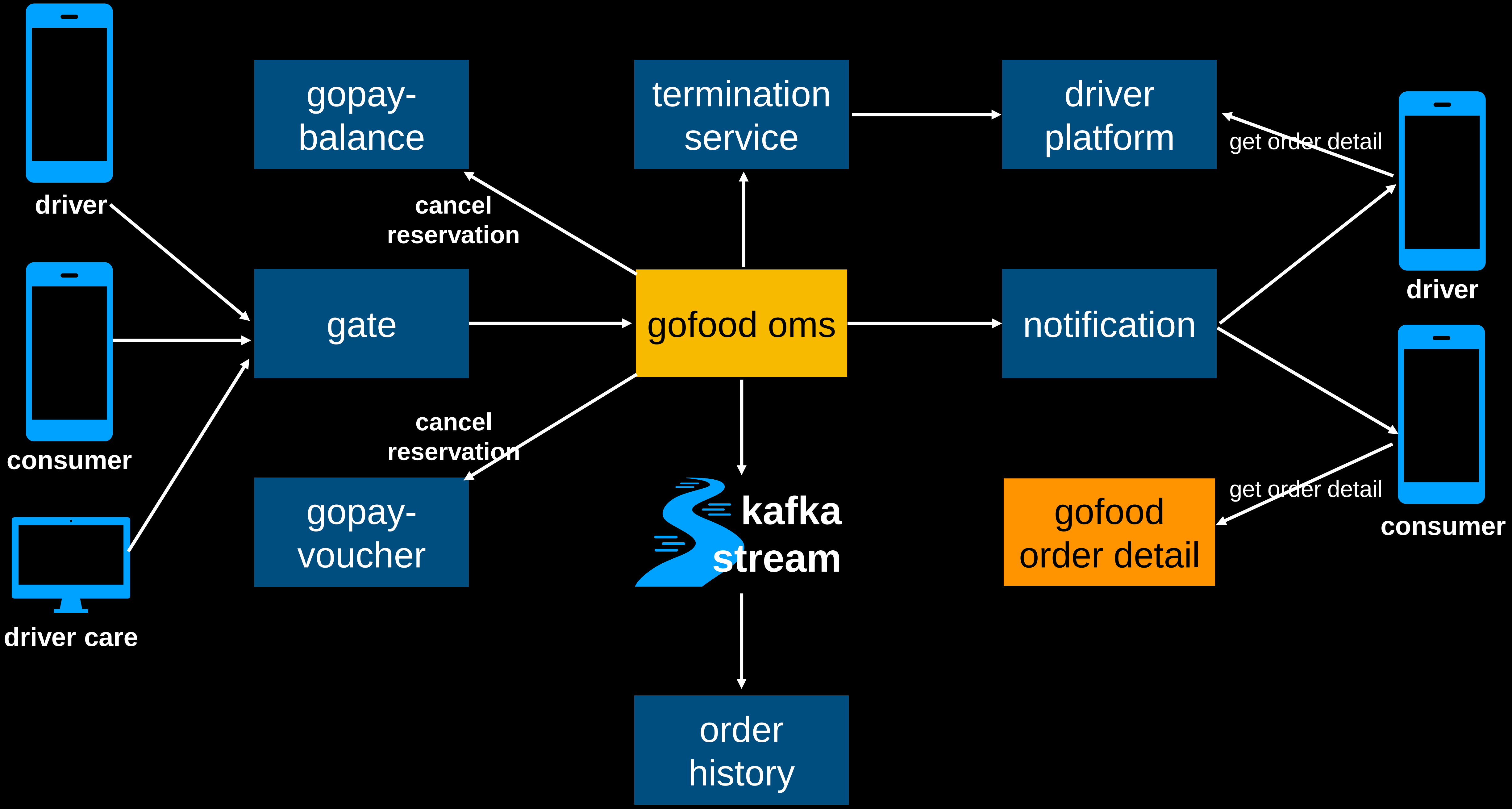
Create Order GoPay Voucher NonGoresto - iteration #4



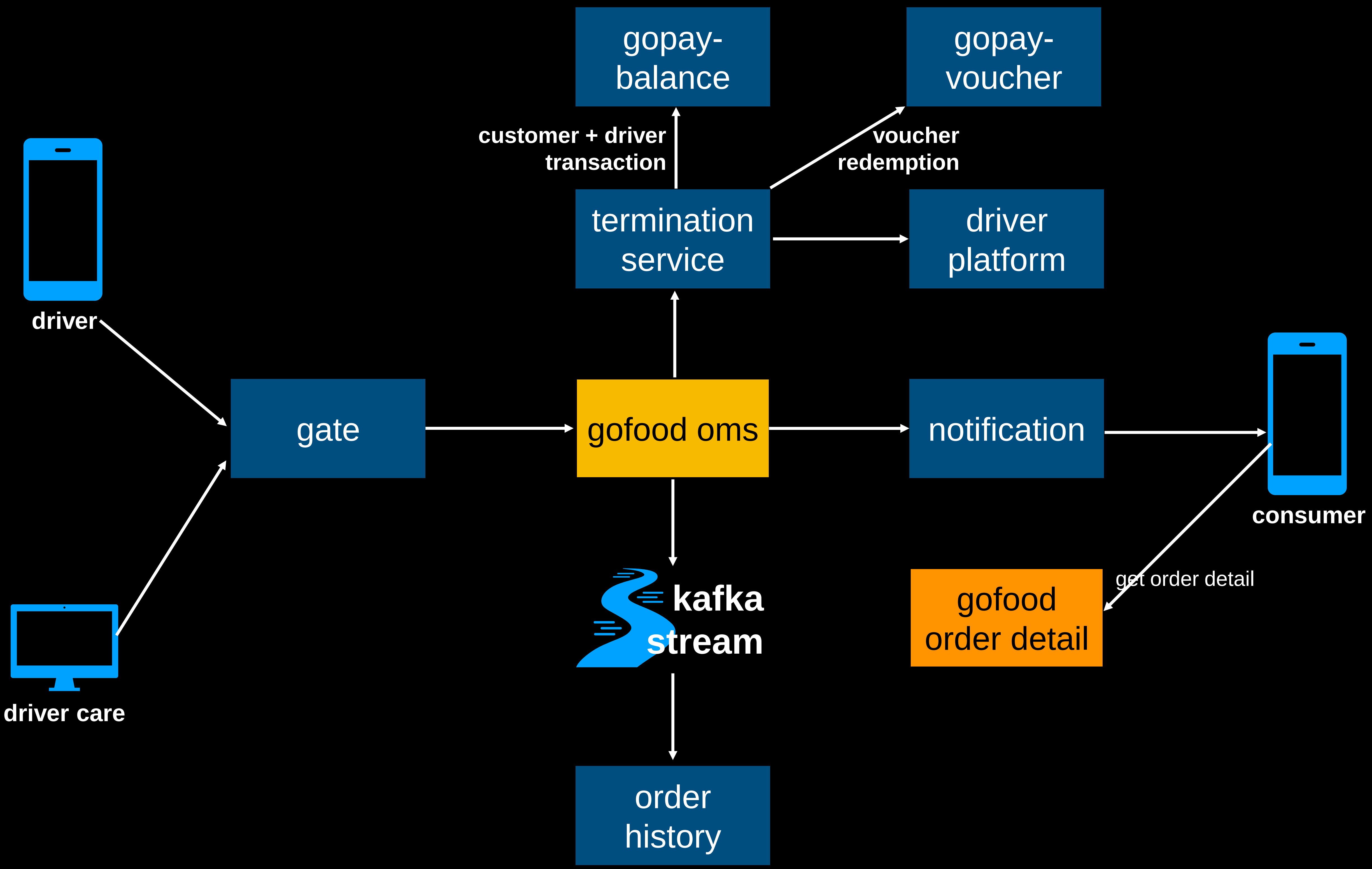
Order Pickup GoPay Voucher NonGoresto - iteration #4



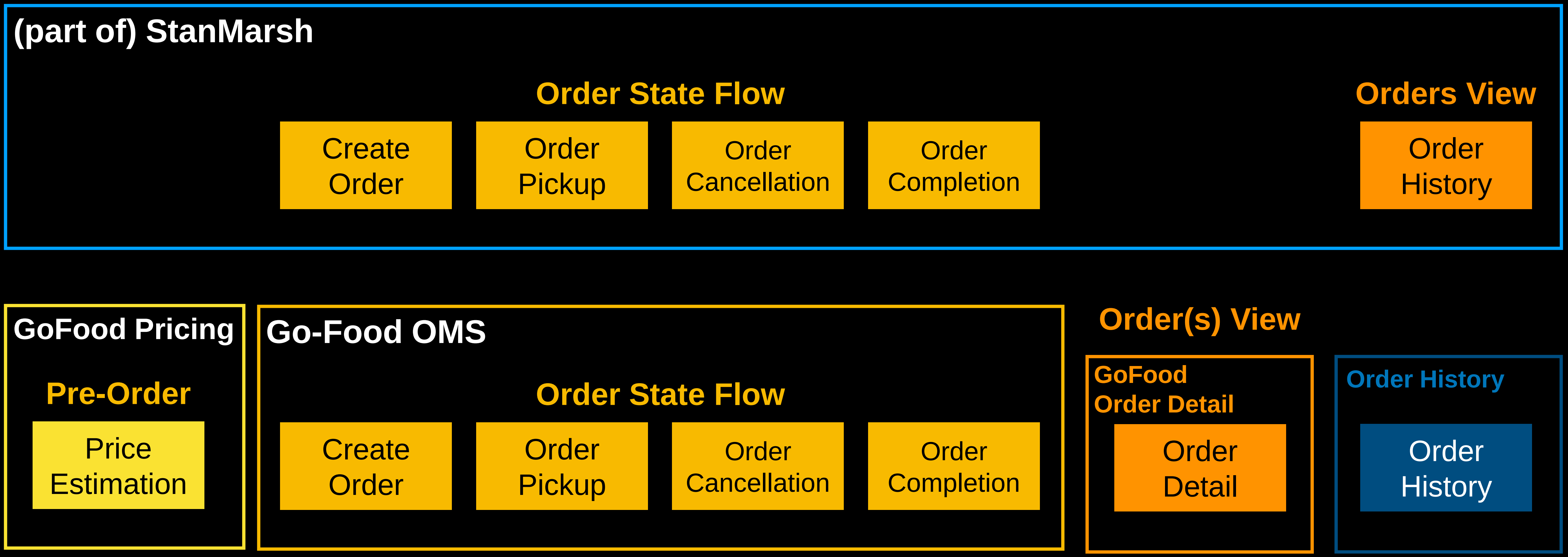
Order Cancellation GoPay Voucher Non-Goresto - iteration #4



Order Completion GoPay Voucher Non-Goresto - iteration #4



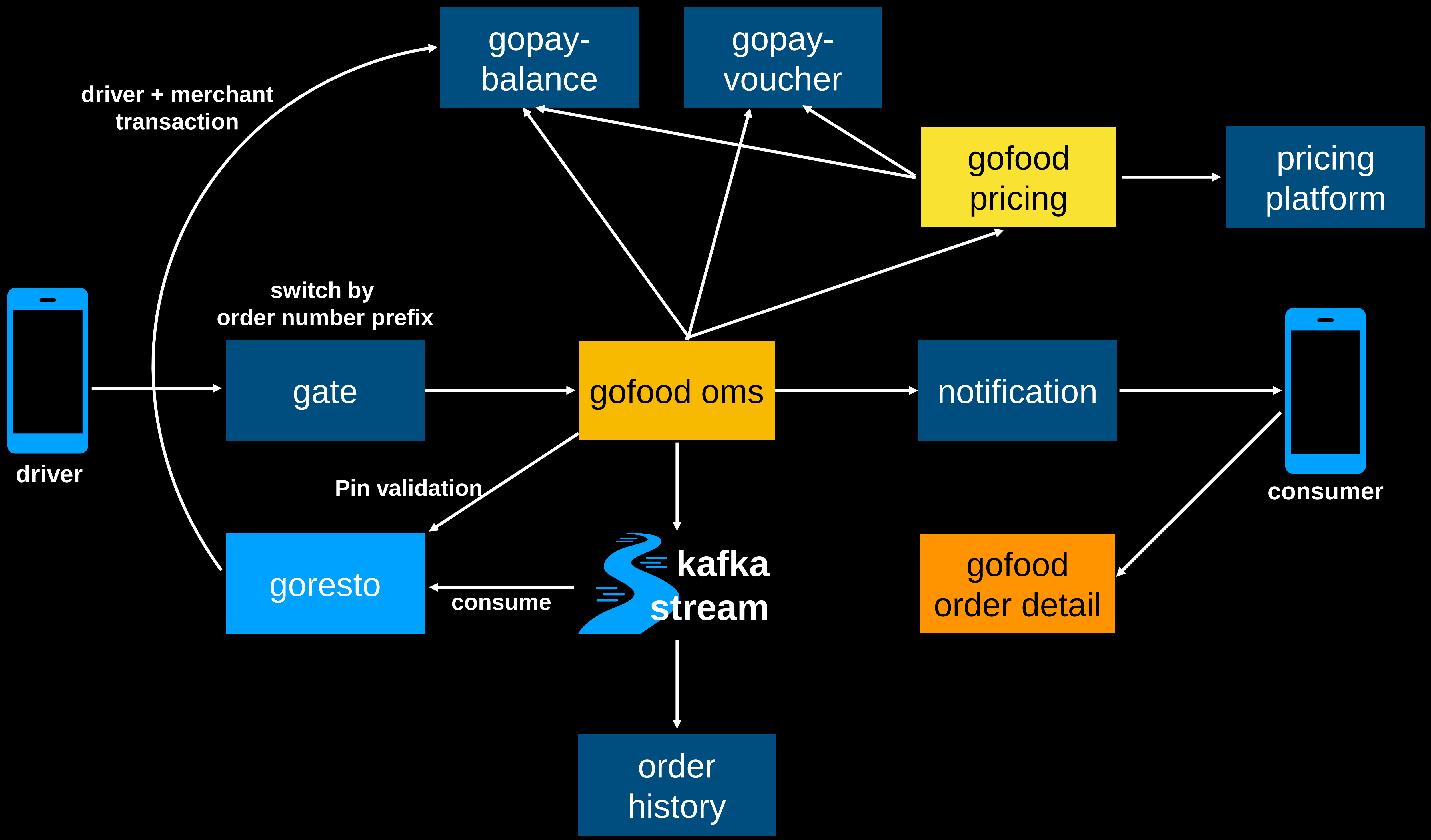
Go-Food OMS Rewrite - iteration #4 final



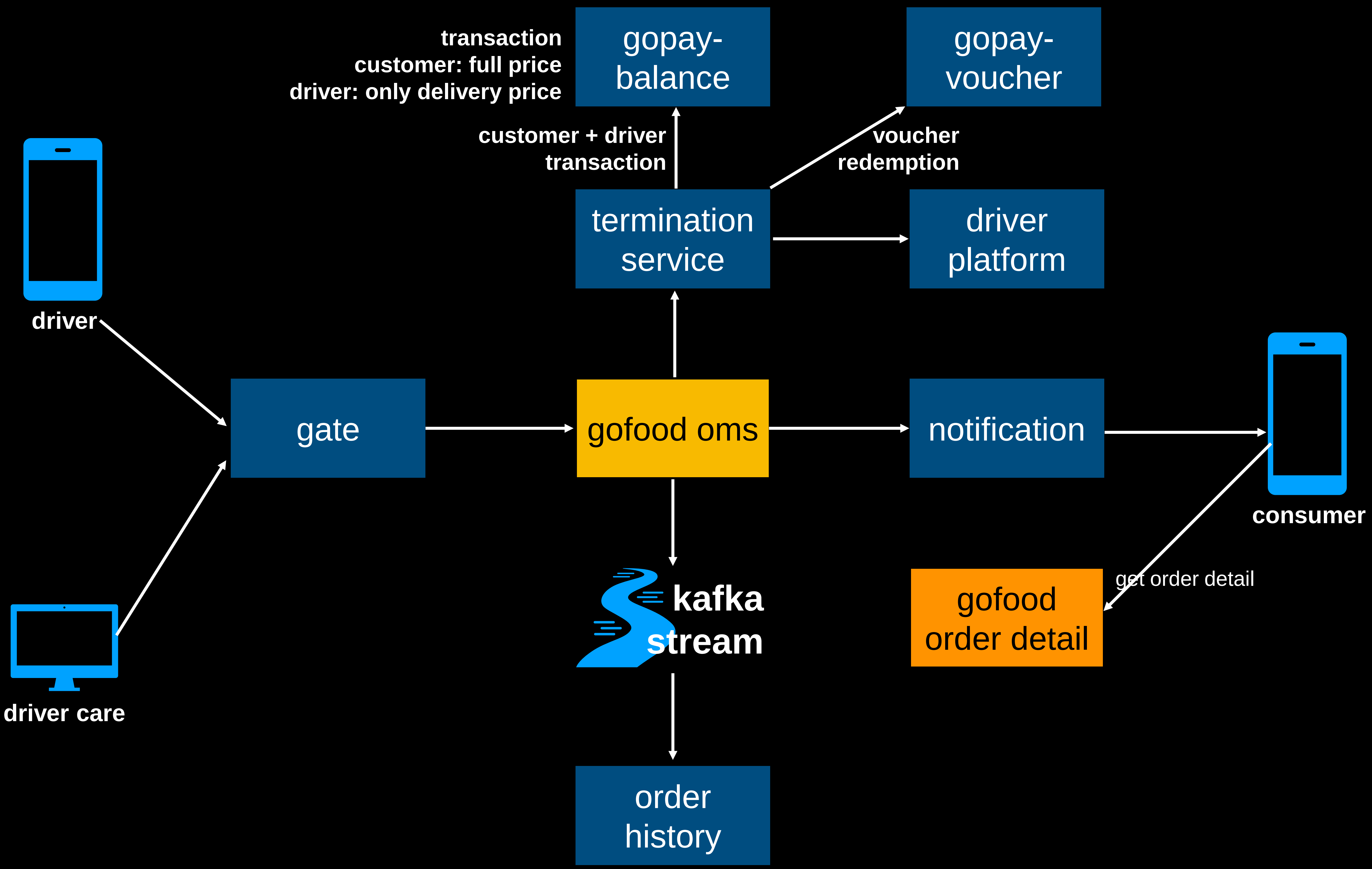
cash non-voucher non-goresto flow is covered
gopay voucher non-goresto flow is covered

NEXT:
GORESTO
ORDER FLOW!
ITERATION #5

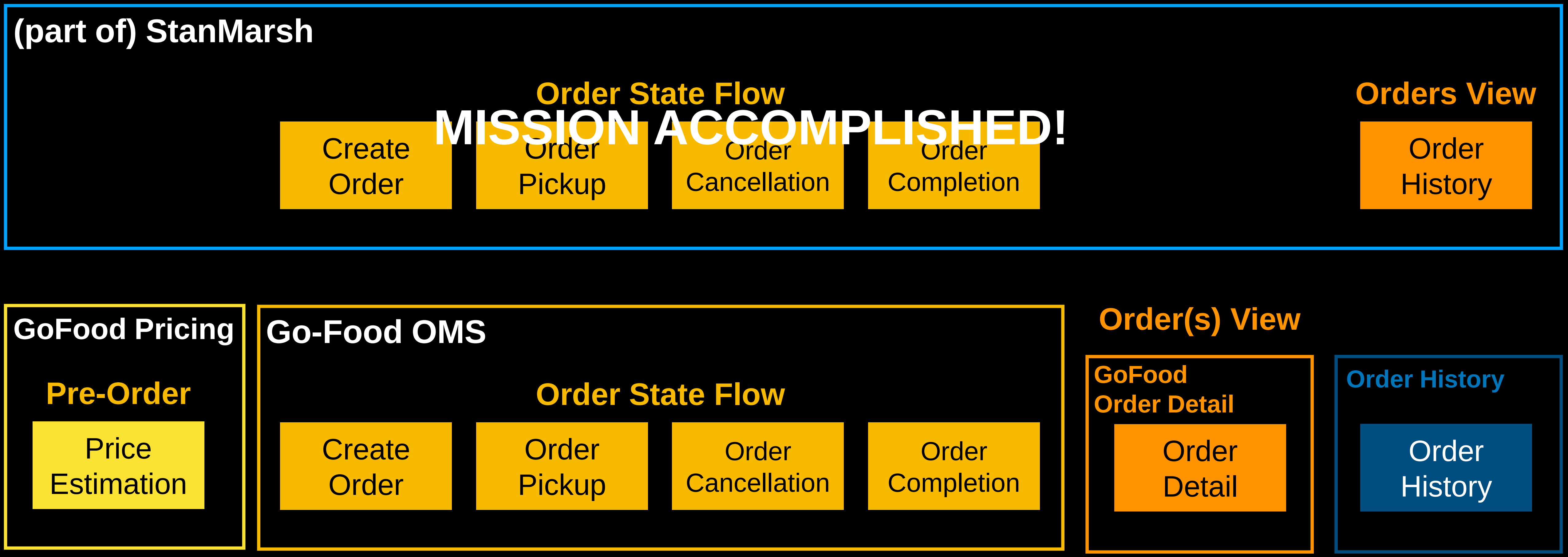
Order Pickup GoResto - iteration #5



Order Completion Goresto - iteration #5

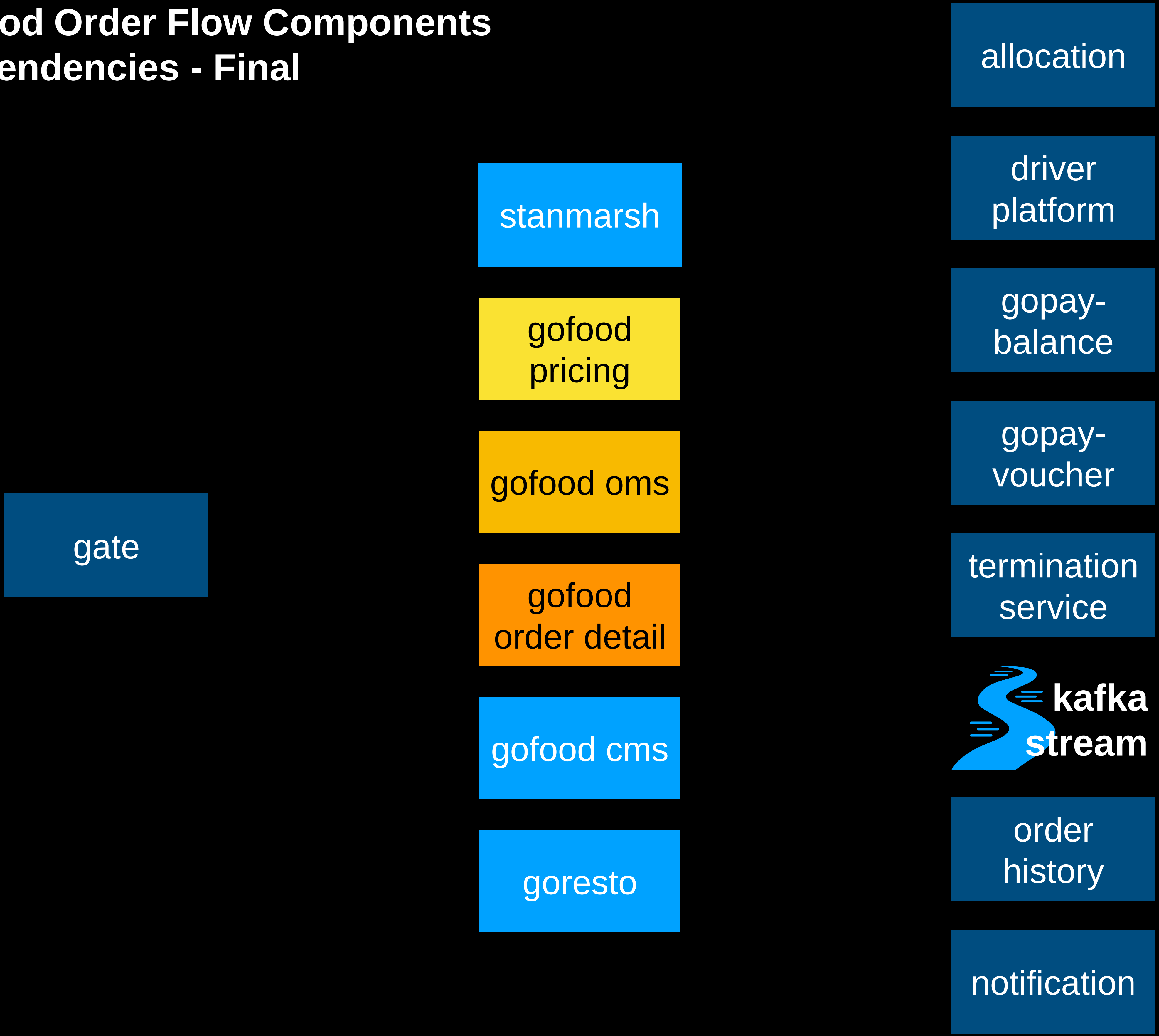


Go-Food OMS Rewrite - iteration #5 final



cash non-voucher non-goresto flow is covered
gopay voucher non-goresto flow is covered
goresto flow is covered

Go-Food Order Flow Components & Dependencies - Final



Lesson Learnt

Migrate business flow

Faster to execute, minimize moving dependencies
Proving scalability: independent between flows (domain driven design)
Easy to control traffic for the rest of the flow

Split datasource first

To avoid data consistency potential issue

Staggered rollout

Load control
Isolating impact
Easy to rollback

Automated test

Defining the expected behaviour for the new service
Make development a lot faster

Eventual consistency

To avoid multi-dependencies transaction hell
Stable on sudden traffic

Monitoring and alerts

Visualize the traffic switching
Get failure feedbacks, enabling fast rollback

Thanks

If you think this is a challenging stuff,
just keep in mind that

We’re always hiring.

https://www.go-jek.com/careers/search/?type=function&search=engineer