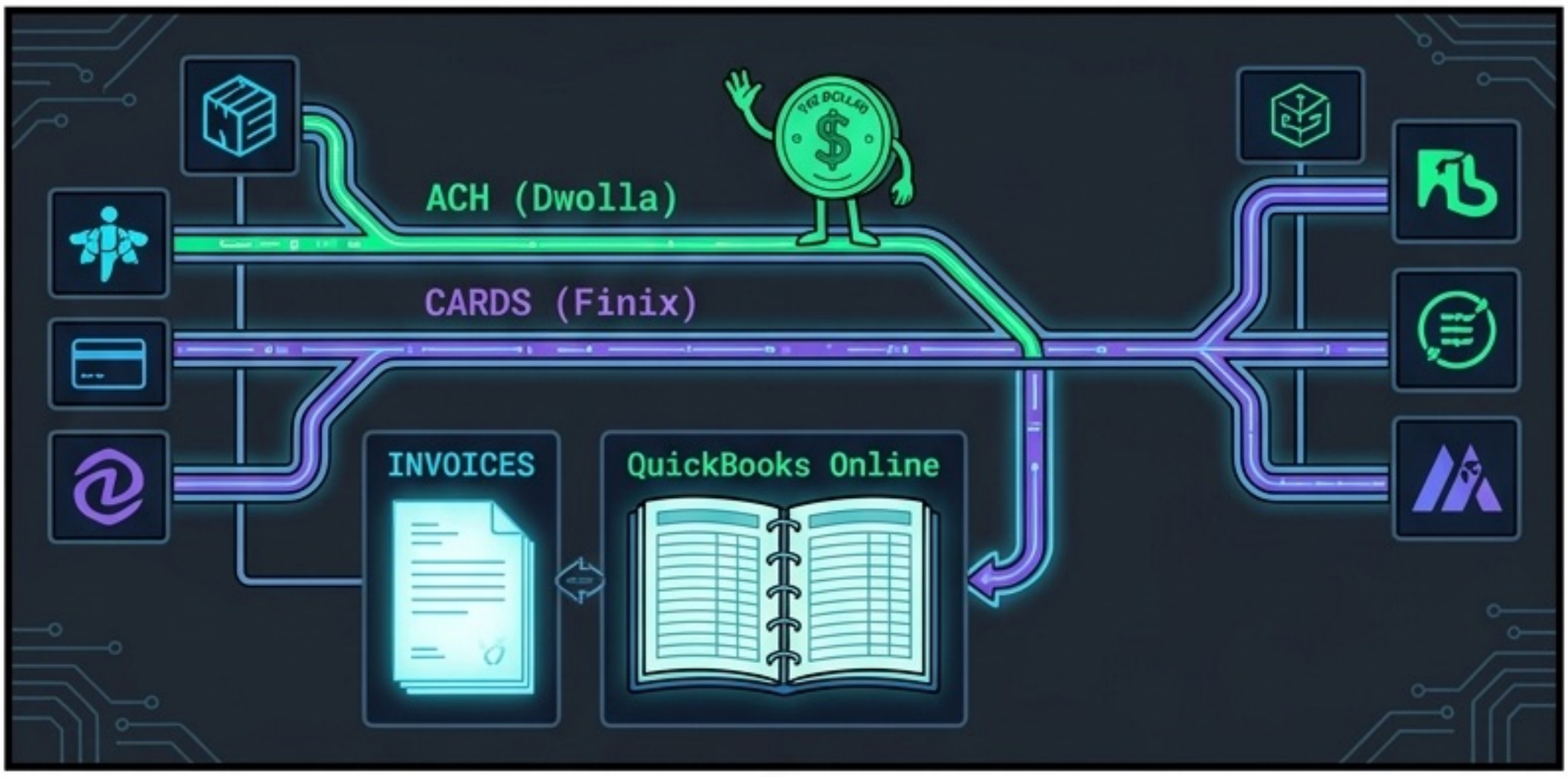
ZENBILL

A Field Guide to the Money Rails

Follow one dollar — from invoice to QuickBooks.

We begin our expedition where the money lives: **ZenBill**.

ZenBill moves **REAL** money between businesses.



Because the money is **REAL, state** and **correc-ctness** are the whole game.

So where do we start?

We follow a single dollar.

FIELD HUD ▸

Rails 7 + React 18 · ACH + cards · invoices·Quitcks

first read: README.md

Before money moves, the system asks: **who are you** — and **for which org?**

Organization

contacts

members

funding sources

Authorization starts at UsersOrganization — NOT User.

User

UsersOrganization

manager

bookkeeper

member

approver

Permissions come from the **MEMBERSHIP** — the **CanCan** ability is built from `UsersOrganization`, not the bare user.

So the **same person** can have **different powers** in **different orgs**.

Exactly.

FIELD HUD ▸

```
auth starts at UsersOrganization ·
app/models/users_organization.rb ·
roles: manager/bookkeeper/member/approver
CanCan
```

The habitat has many entrances — each with its own kind of key.

SESSION
(dashboard)

TOKEN
(external contacts / onboarding)

BASIC AUTH
(public v1 API)

Five subdomains, one Rails app.

`api`
`app`
`onboarding`
`i`
`webhooks`

It all lives in one **routing** file.

It all lives trone. Note: **`internal`** is a **NAMESPACE** under the **`api`** subdomain — not its own subdomain.

One app, three auth styles, five front doors.

FIELD HUD ▸

`config/routes.rb`

- 5 subdomains:
  api / app / onboarding / webhooks / i
- (internal = namespace under api)
- 3 auth:
  - session
  - token
  - Basic Auth

Don't look in the controllers – they're nearly empty. Follow the workbench.

Controller

~64 **ActiveInteraction** classes — **ONE** operation each.

app/interactions/

CreateTransfer

SettleTransfer

CreateInvoice

CreateInvoice

CreateInvoice

compose

compose

Each interaction has **inputs**, an **execute step**, and its own **errors**. They **COMPOSE** — one calls another.

And the **API's shape**? Rendered by **130+ Jbuilder templates**.

*.json.jbuilder

FIELD HUD

- ▸ **app/interactions/** (~64)
  - · **compose** pattern
  - · thin controllers
  - · app/views/**/***.json.jbuilder** (130+)

Our specimen begins its journey – as a line on an invoice.

Every **transfer** plugs into a money source at each end. There are four kinds.

Transfer
origin
destination
money rails
money rails

ports

Plaid
(linked bank)

US Bank
(manual ACH)

Credit Card
(Finix)

US Paper Check

$1

ONE **polymorphic association** — **FOUR implementations.**

`belongs_to :origin_funding_source, polymorphic: true, polymorphic: true` — and the same for the **destination**. The type column says which which of the four.

So a **transfer** is just two **funding-source plugs** and an amount.

FIELD HUD

```
Transfer belongs_to
origin/destination_funding_source,
polymorphic:true

4 types: Plaid · US Bank · Credit
Card(Finix) · US Paper Check
```

Now the dollar enters the main rail line: **ACH**, operated by **Dwolla**.

Dwolla ACH

Transfer

$1

CreateTransferInDwollaJob

handled in the **BACKGROUND**

Transfer

$1

$1

The API doesn't block — a background **JOB** hands the transfer to to **Dwolla**.

Note for the field journal: these jobs run on **delayed_job** — **NOT** Sidekiq.

FIELD HUD ▸

app/services/dwolla_service.rb ·
app/jobs/CreateTransferInDwollaJob
queue = delayed_job_active_record
(NOT Sidekiq)

The dollar's status is never set by hand. The rail itself **reports back**.

webhook `customer_transfer_completed` Dwolla event

webhook `customer_transfer_completed` Dwolla event

webhook `customer_tranfelal` Dwolla event

webhook `customer_transfer_completed` Dwolla event

SIGNAL TOWER

Transfer
"requested
→ scheduled
→ pending
→ sent
→ settled

failed

canceled

PADLOCK
"no_direct_assignment: true"

7 **AASM state machines** — driven by events, never assigned.

MAYA

A **webhook event** fires an interaction like `ACH::SettleTransfer`, which flips the state through a named event — **transfer.settle!**. You never write `` `status = 'settled'` ``.

Seven machines: **Transfer, Invoice, Organization, Contact, PlaidFundingSource, RecurringTransfer, KycBusinessCustomer.**

DEV

(settled)

$1

THE DOLLAR

FIELD HUD ▸
7 AASM machines · no_direct_assignment:true · DwollaWebhookEventService (9 events) → ACH::SettleTransfer/FailTransfer/CancelTransfer

The cardinal rule of money rails: the **dollar** must never move twice.

idempotency_key: sbc

idempotency_key: abc

Gate catches the duplicate envelope and returns a **CACHED reply**.

Duplicate request detected. **CACHED** reply returned.

$1

Same key → **cached response**. **Serializable isolation** blocks the race.

IdempotencyKey

SERIALIZABLE

FIELD HUD ▸

app/controllers/concerns/idempotent_request.rb

IdempotencyKey + Postgres SERIALIZABLE

webhook verify = HMAC-SHA256

And inbound webhooks are **SIGNATURE-VERIFIED** before we trust them — **Dwolla** signs every call.

HMAC-SHA256

Double-spends and forged callbacks: both denied. The **dollar** is safe.

FIELD HUD ▸

app/controllers/concerns/idempotent_request.rb

IdempotencyKey + Postgres SERIALIZABLE

webhook verify = HMAC-SHA256

ACH is the main line — but the habitat has several other rails.

Plaid (bank links)

Finix (cards)

Stripe (premium)

Stripe (premium)

Plaid

3-5 bank-link verification

unverified → linking → verified

Finix

webhook signal

Stripe

Organization → Upgraded

higher limits

faster settle

Plus **RECURRING** transfers on a schedule, and invoices that scan themselves —

ParseInvoiceJob

OCR via Base64.ai

Reads PDF

Amount
Date
Vendor

Invoice

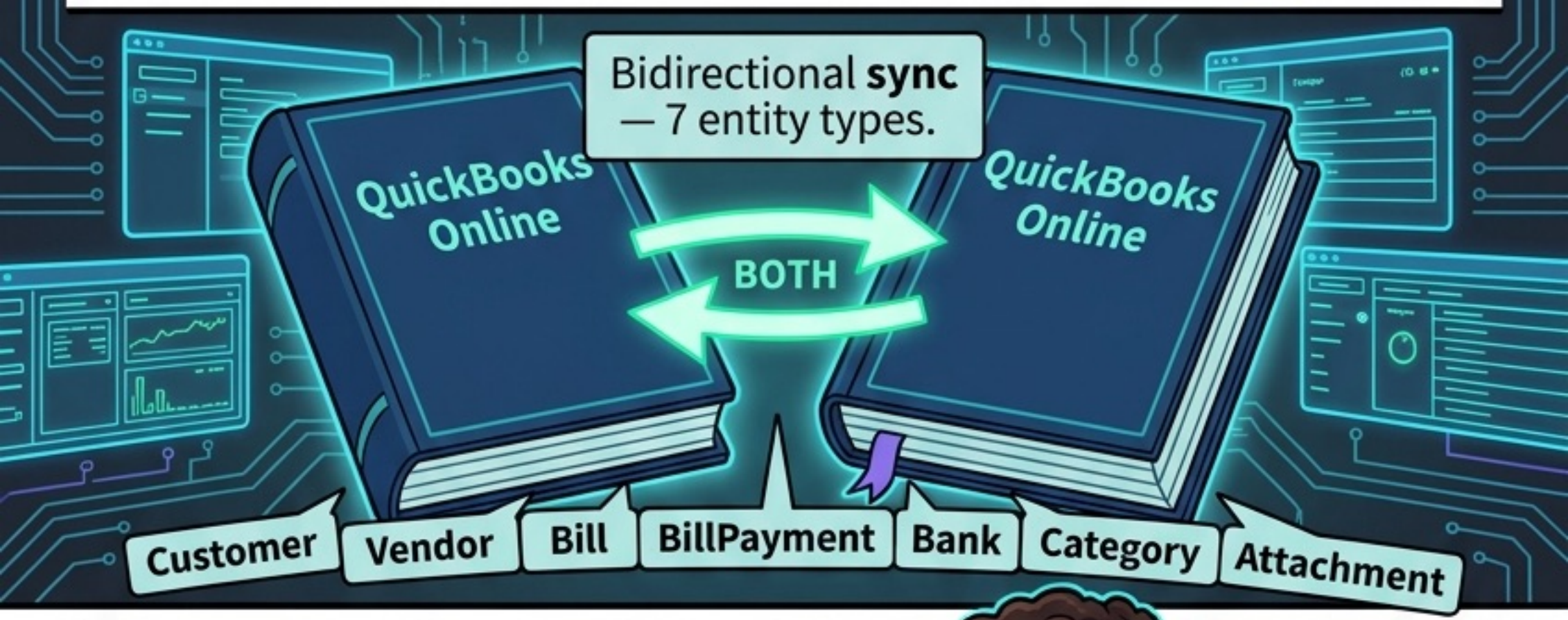
Same dollar story, different on-ramp.

FIELD HUD ▸

Plaid 3-step link · Finix token+webhook
Stripe premium · RecurringTransfer (delayed_job)
ParseInvoiceJob = OCR via Base64.ai

The **dollar** has settled. Now its record must appear in the books.

Bidirectional **sync** — 7 entity types.



When a **transfer settles**, an `after_commit` hook **queues** the **QBO** bill-payment sync.

```
after_commit
  :sync_with_quick_books_online
```

The **dollar** now exists in two places that always agree.

FIELD HUD ▸
after_commit :sync_with_quick_books_online
app/interactions/quick_books_online/ (17)
7 entity types

And so, the full migration of a single dollar through **ZenBill**.

FIELD GUIDE

logic in interactions

Invoice

Plaid bank

Credit Card/ Finiix

Paper Check

Dwolla ACH

Funding Source

$

$1

THE DOLLAR

Org-centric auth

logic in interactions

Invoice

Transfer

webhooks drive AASM

signal bolts

`no_direct_assignment`

idempotent & verified

QuickBooks

4 ports: Plaid bank

Credit Card/Finiix

US

Funding Source

Dwolla ACH

YOUR FIRST READS

1 README.md
2 app/models/transfer.rb
3 app/interactions/transfers/
4 app/services/dwolla_service.rb
5 the *_webhook_service.rb files

State is **DRIVEN**, never set by hand.
Welcome to ZenBill.

$1

$1

'FIELD HUD ▸

run the suite: bundle exec rspec

money flow = Invoice → Transfer → Funding Source → Dwolla → webhook → AASM → QuickBooks