Eight Rules

Rule 1: Embrace Plurality

Getting SKUd

What is a Stock Keeping Unit (SKU)?
- Can be sold
- Must be shipped
- Takes up shelf space
- Has a price & cost
- One SKU exists per “kind of thing” that can be sold
- Does not track the individual inventory item

Digital Downloads

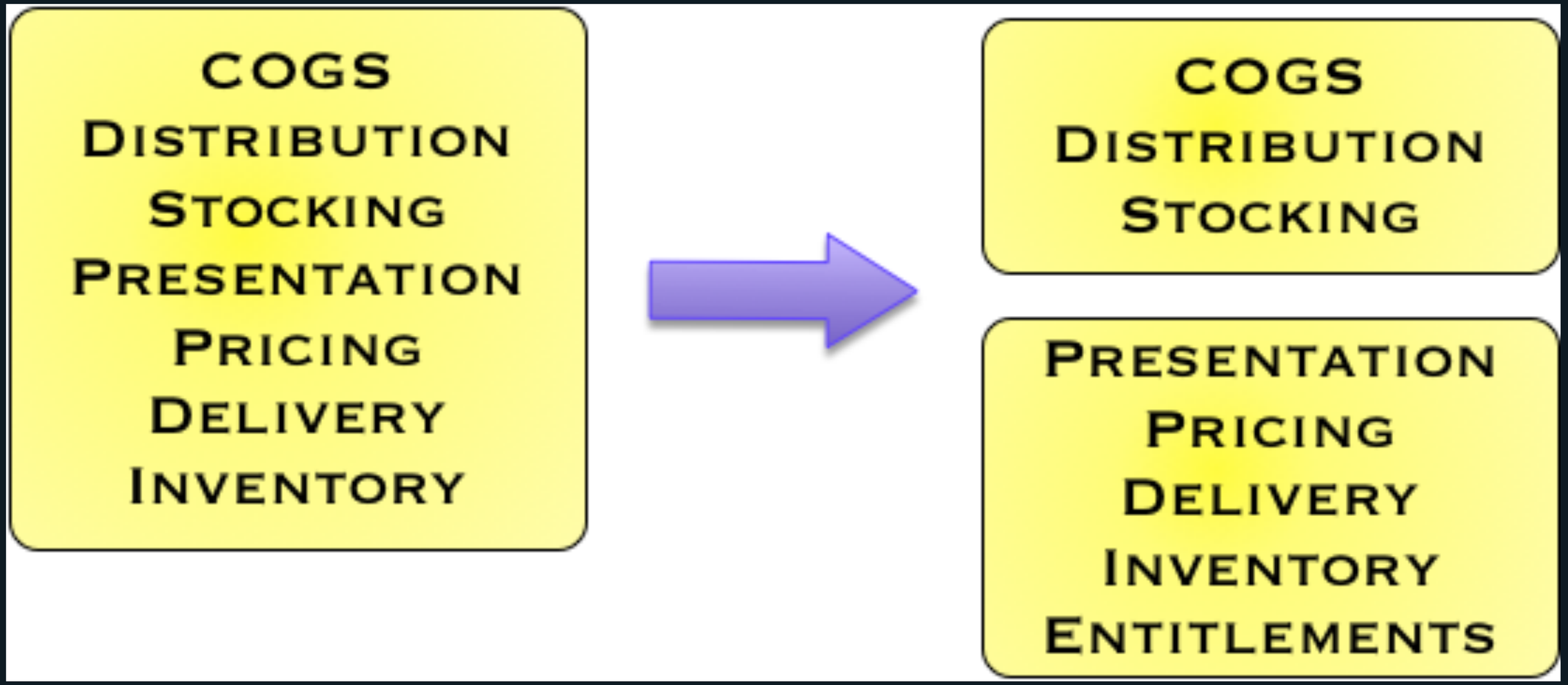
— Added: tracking individual purchase
— Irrelevant: Shelf space, shipping, fixed cost

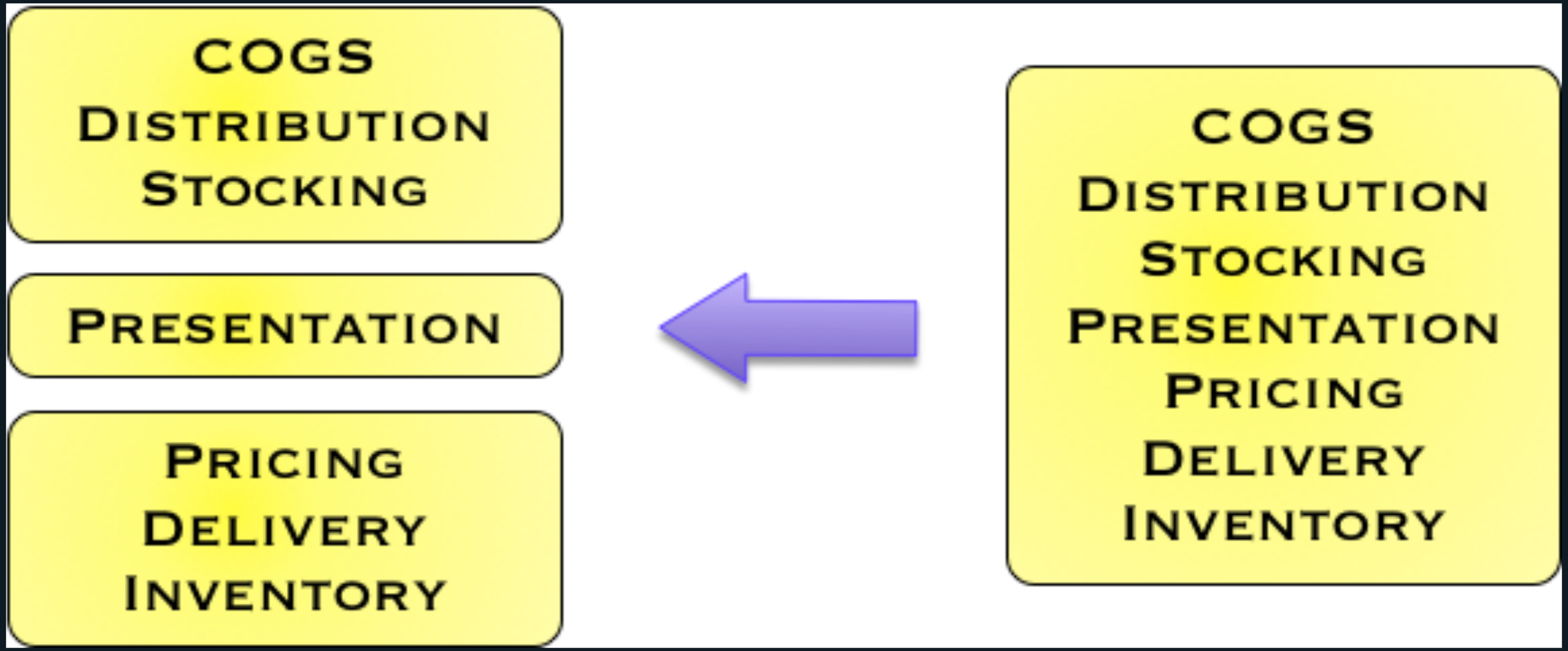
Partner Sales

— Added: Multiple prices per item
— Irrelevant: Controlled ID space

Home Installation and Renovation

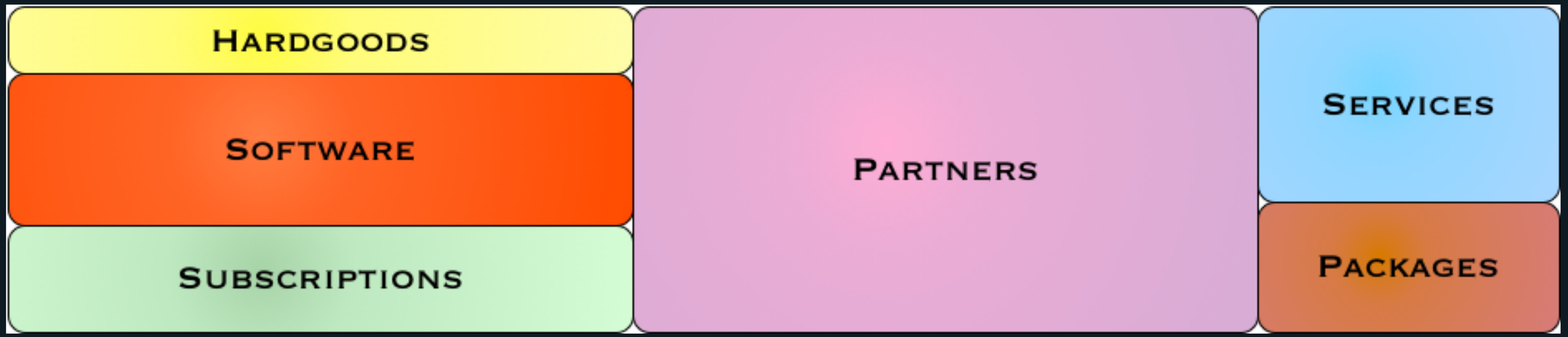
— Added: 16,000,000 new SKUs


COGS
DISTRIBUTION
STOCKING
PRESENTATION
PRICING
DELIVERY
INVENTORY
COGS
DISTRIBUTION
STOCKING
PRESENTATION
PRICING
DELIVERY
INVENTORY
ENTITLEMENTS


COGS
DISTRIBUTION
STOCKING
PRESENTATION
PRICING
DELIVERY
INVENTORY
COGS
DISTRIBUTION
STOCKING
PRESENTATION
PRICING
DELIVERY
INVENTORY

Dark Matter

Federated


HARDGOODS
SOFTWARE
SUBSCRIPTIONS
PARTNERS
SERVICES
PACKAGES

Work Time Yay, Merger!

Yay, Merger!

We've acquired two competitors (in addition to our own system)
Write down your strategy.

1. Subscribr.com
   — Monolithic Ruby on Rails application with a Postgres backend
2. subsee.ly
   — SPA front end calls API
   — Node.js API layer in front of
   — Customer service - python on Cassandra
   — Contains customers and subscription info
   — Product service - C# over Oracle
   — Each Vendor has its own Customer and Product service instances
   — Vendors have direct database access to their Customer and Product instance DBs
   — Payments via Stripe

Presenting Solutions

1. Start with relevant parts of original system.
2. What are you changing for the mergers?

Audience: Ask questions to understand, not to critique.

Rule 2: Augment Upstream, Contextualize Downstream

Augment Upstream

— Add to data as "early" as possible
— Avoid creating privileged downstreams
— Everybody wants the best data available

Work Time

Bundles

— Package of multiple goods or services
— Defined by a vendor, limited to that vendor
— Has it's own presentation and pricing

Contextualize Downstream

- Things closer to users and APIs change more frequently
  - Presentation
  - Policies
  - Limits and ranges

Work Time
Vendor CSRs

Vendor CSRs

Must be able to:
- View a customer's current subscriptions
- View a customer's complete history
  - Sub, re-sub, upgrade, downgrade.
  - Reminders, payment methods, expirations
  - Email bounces
  - Past CSR interactions and notes
- Upgrade, downgrade, or cancel a sub
- Force re-delivery of item
- Add notes to the customer's file

Must not be able to:
- See the customer's interaction with other vendors
- See products, items, prices, etc from other vendors
- Alter the delivery address of other subscriptions that customer has
- See payment methods the customer has used with other vendors but not this one

Rule 3: Decentralize

Many New Requirements

1. Per-vendor email templates
2. Private domains for vendors: email and web addresses
3. Multiple catalogs
4. Multiple email renderers
5. Some vendors want to provide their own catalog system
6. We made a deal with PayPal. Must support as PayPal payment method.

Rule 4: Beware Grandiosity

— Enterprise Data Dictionary
— Global Object Model
— "One World" Model

Bounded Context

The antidote to grandiosity.

Contexts

— Draw the interior services in your contexts.
— Where is data being replicated?
— Where are the anticorruption layers?
— How can we limit the spread of concepts?

Rule 5: Isolate Failure Domains

— Mail handler blocks forever.
— CCVS sometimes rejects requests.
— WMS drops connections.

Activation Sets

— Every service that participates in a call graph

— Services appearing in many activation sets must change less, be more available.

Failure Domain

— The "shadow" of a service.
— Every call type with that service in it's activation set.

Isolating Failure Domains

1. Convert hard dependencies into weak
    1. Internal cache
    2. Secondary service
    3. Default or fallback value
2. Add replicas
3. Cleave nouns along their adjectives

Work Time

Coupled Features

— Draw the activation sets for 1 key feature of your choice

— Draw the failure domains for 1 key service of your choice

— Which features have coupled failure modes?

— Isolate those features into separate services

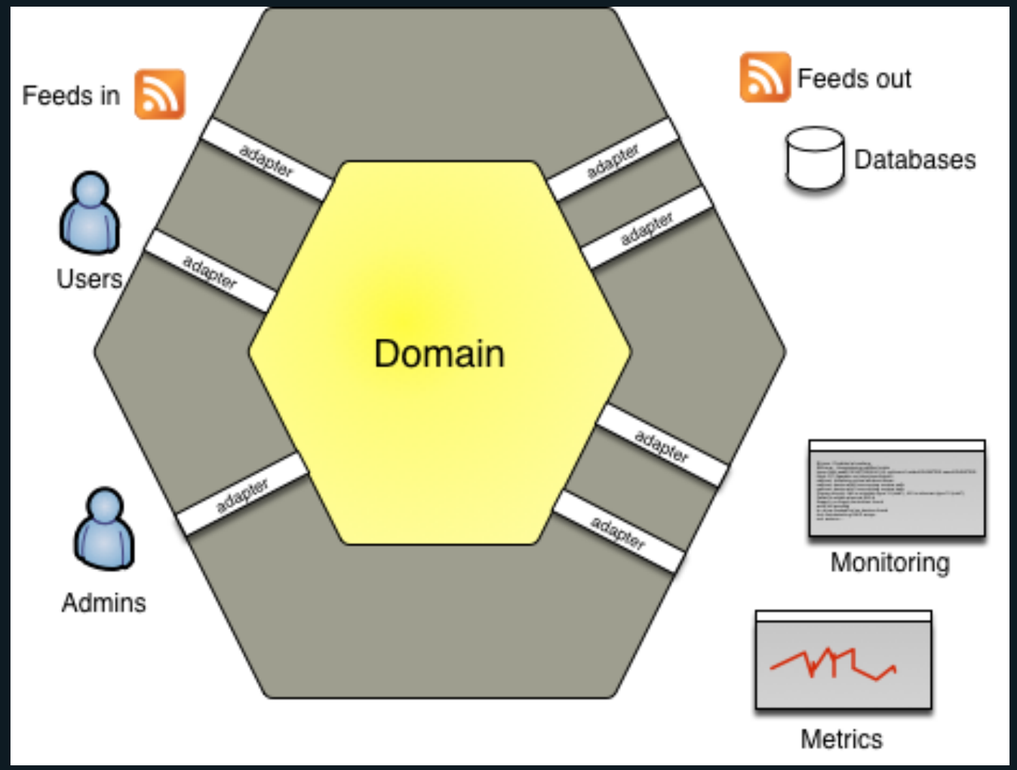
Rule 6: Data Outlives Applications

— ISAM
— VSAM
— Network
— Hierarchic
— Relational
— Graph
— KVS
— Document

Rule 7: Applications Outlive Integrations

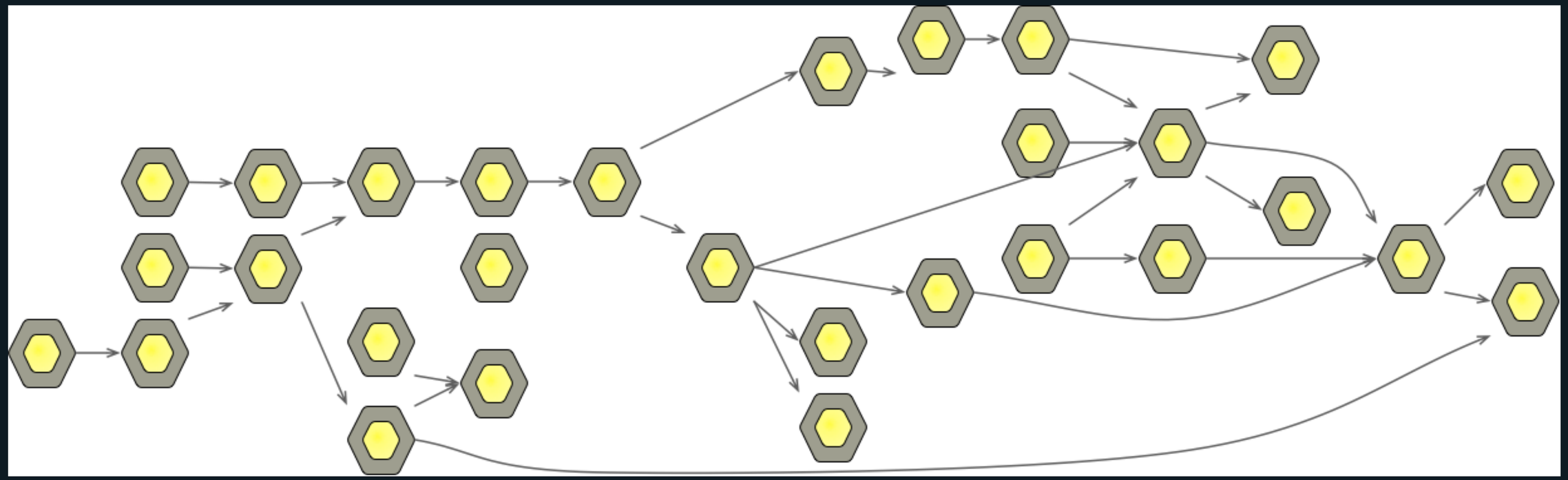
— CICS
— FTP
— RPC
— Sockets
— RPC
— CD-ROM
— XML-HTTP
— RPC
— ESB

Hexagonal Architecture



A.k.a. Ports and Adapters

Honeycomb



Rule 8: Increase Discoverability

Work Time

Residual Value

— Another team wants to do (metered) pay-as-you-go services
— Another team wants to do prepaid, but not renewing, services
— How will other teams know what services you have?
— How can they use what you have created?
— How will you restructure your services and teams?

The Eight Rules

1. Embrace Plurality
2. Augment Upstream; Contextualize Downstream
3. Beware Grandiosity
4. Decentralize
5. Isolate Failure Domains
6. Data Outlives Applications
7. Applications Outlive Integrations
8. Increase Discoverability

THANK YOU!