Building Go Sevices with DDD

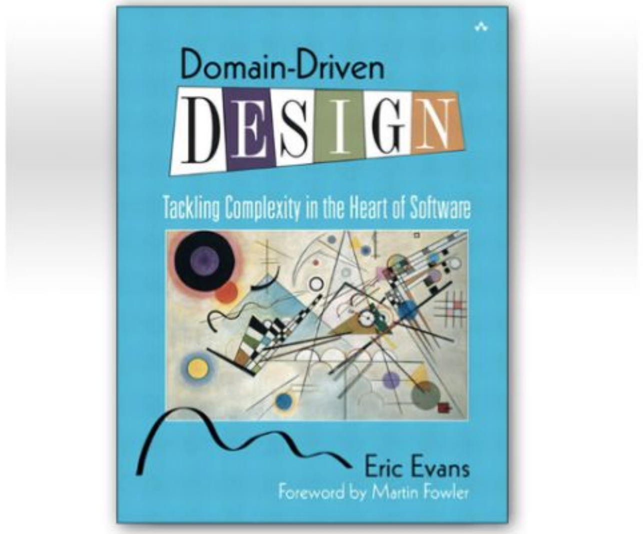
“DRY - Don’t Repeat Yourself.”

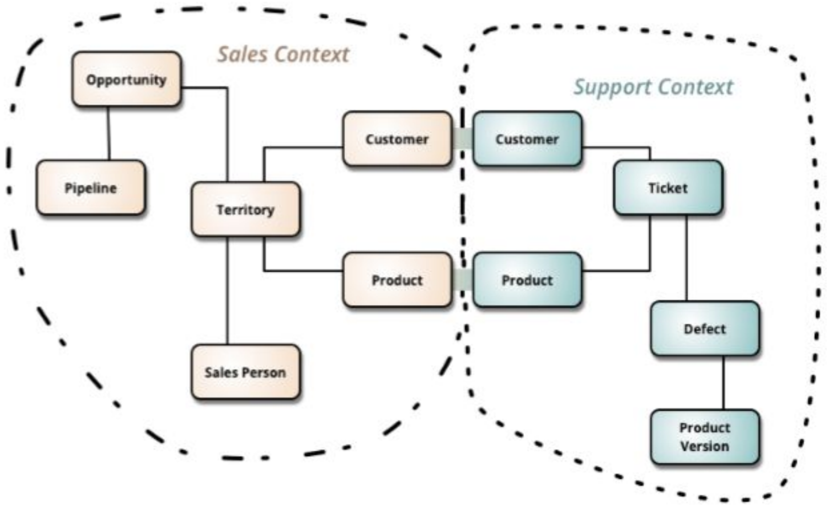
(The Pragmatic Programmer, pg. 24)

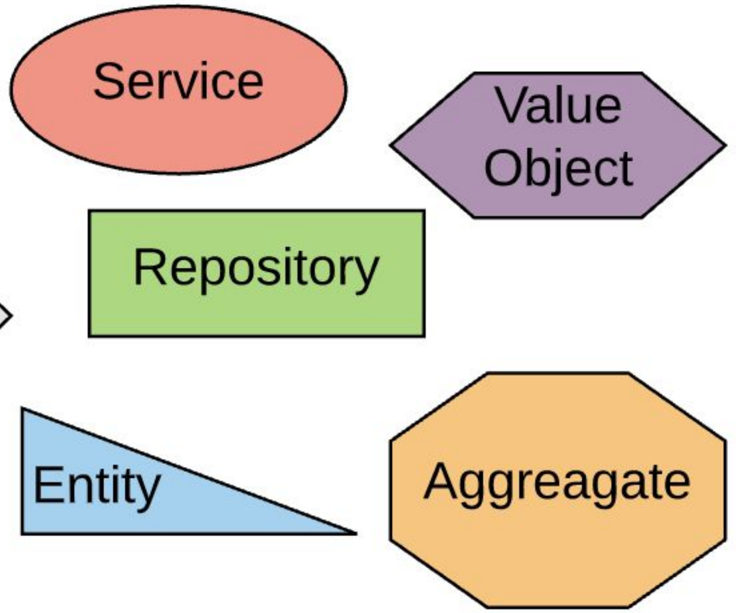
“A little copying is better than a little dependency.”

(Rob Pike, Go Proverbs)

Contents


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler





About DDD

Bounded Context

Building Blocks

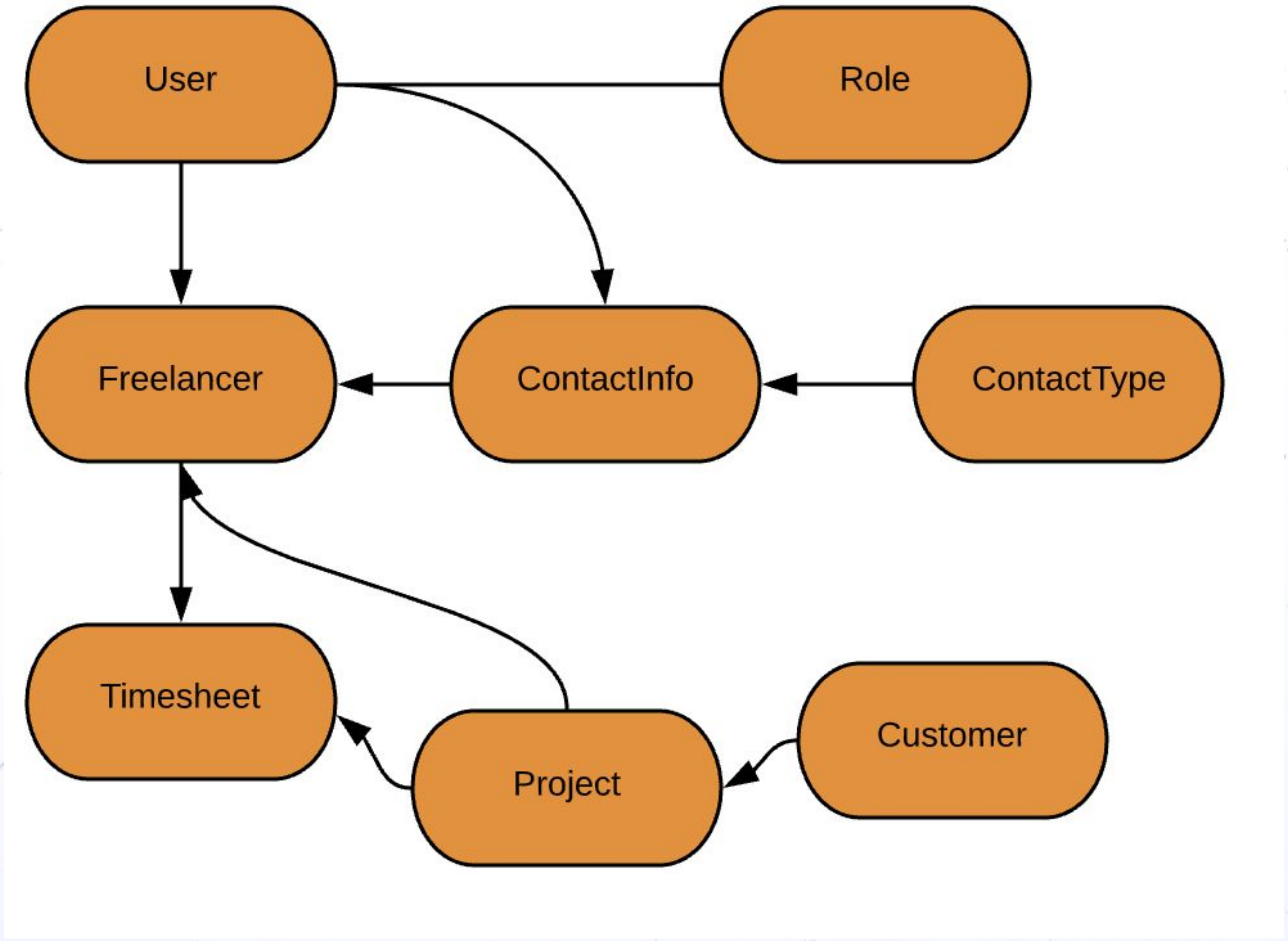
Eddy K

- Director of Engineering @Minute Media
- Twitter/Github: @edkvm

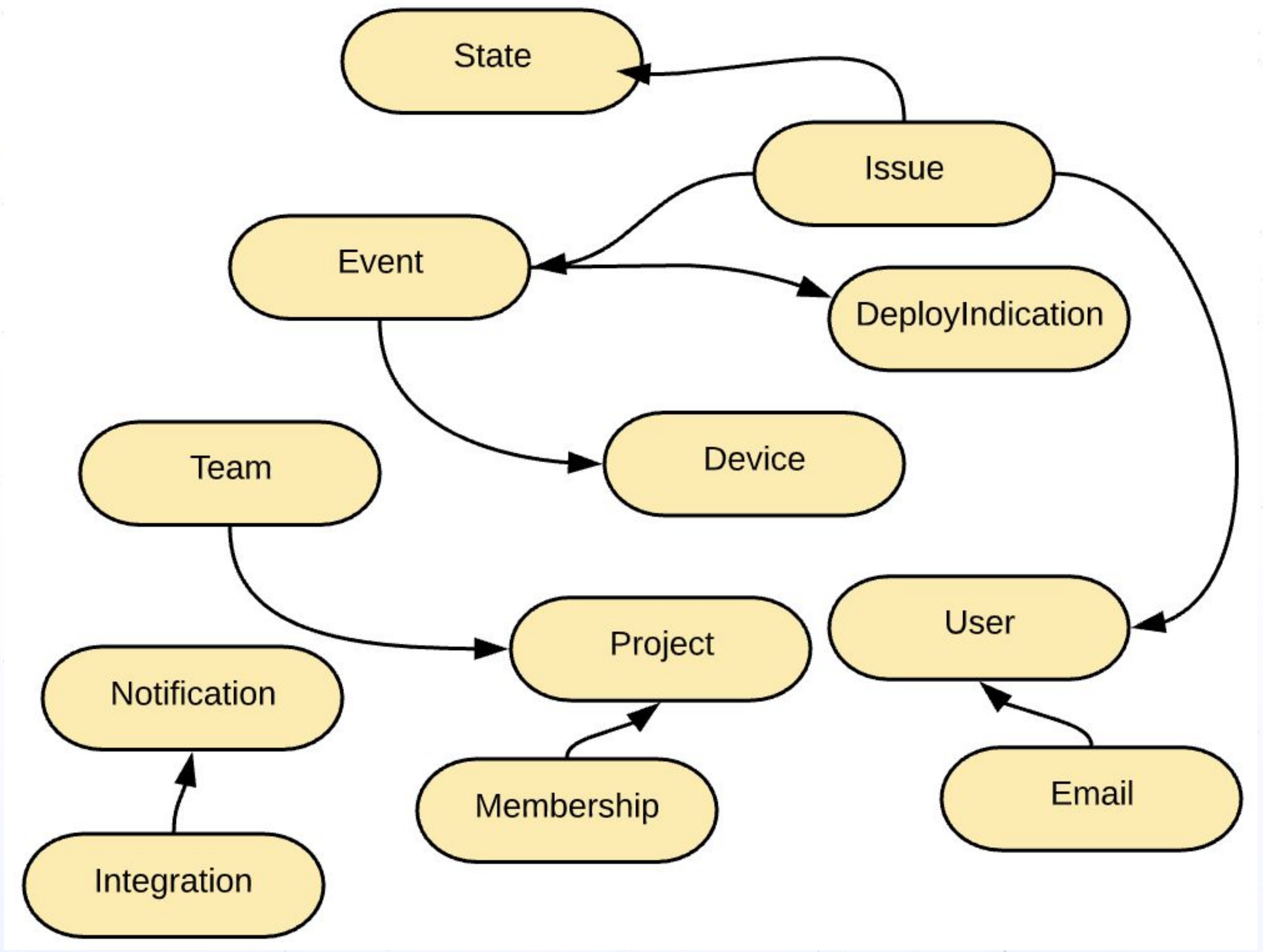
Domain Driven Design (DDD)

- **Complex needs, evolving model**
- **Core domain & domain logic**
- **Collaboration between technical & domain experts**
- **Defines: Context, Domain, Model & Ubiquitous Language**

Domain Driven Design



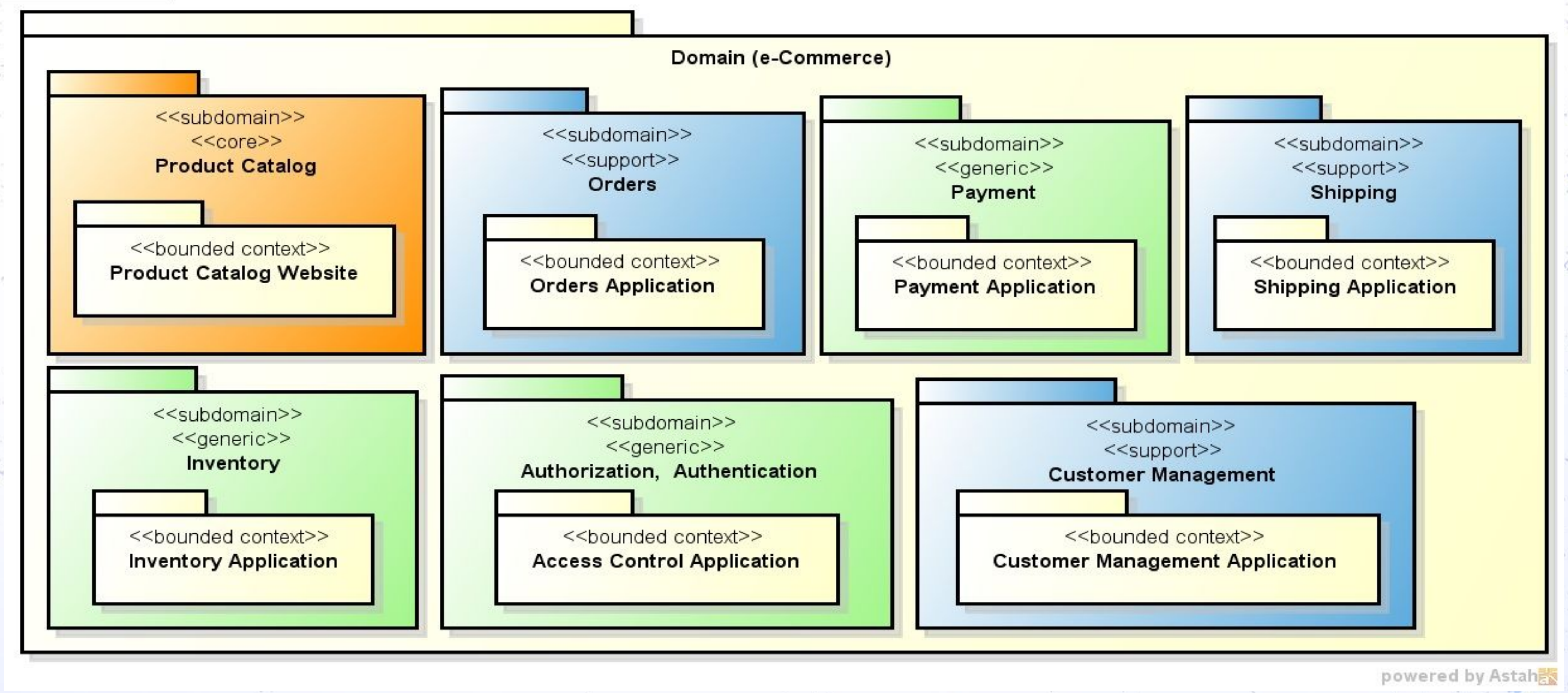
Domain Driven Design



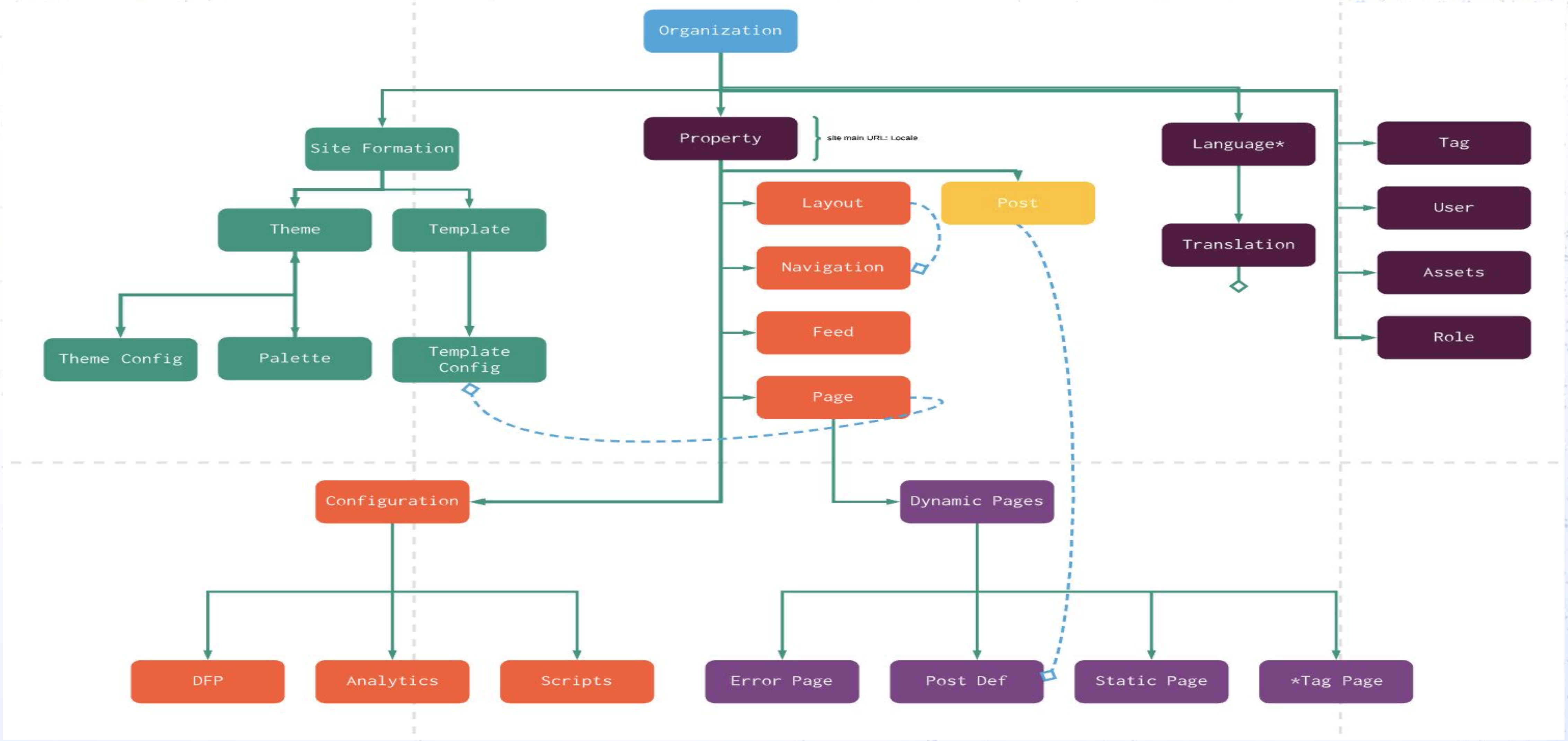
Domain Driven Design: Bounded Context

- For larger systems, it’s hard to build a unified domain
- Defines the boundaries between sub-domains
  - By usage within the app
  - By team organization
- Entities that have multiple definitions

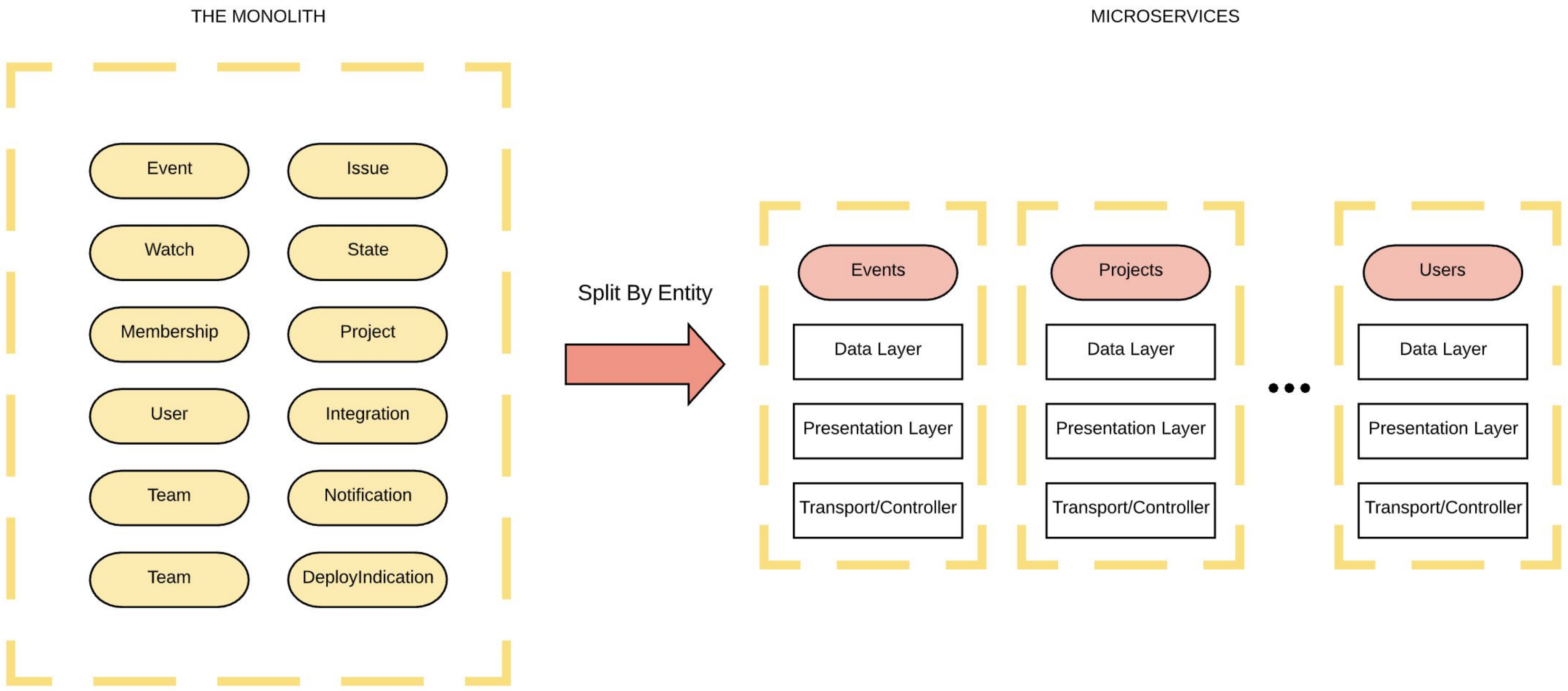
Domain Driven Design: Bounded Context


Domain (e-Commerce)
<<subdomain>> <<core>> Product Catalog
<<bounded context>> Product Catalog Website
<<subdomain>> <<support>> Orders
<<bounded context>> Orders Application
<<subdomain>> <<generic>> Payment
<<bounded context>> Payment Application
<<subdomain>> <<support>> Shipping
<<bounded context>> Shipping Application
<<subdomain>> <<generic>> Inventory
<<bounded context>> Inventory Application
<<subdomain>> <<generic>> Authorization, Authentication
<<bounded context>> Access Control Application
<<subdomain>> <<support>> Customer Management
<<bounded context>> Customer Management Application
powered by Astah

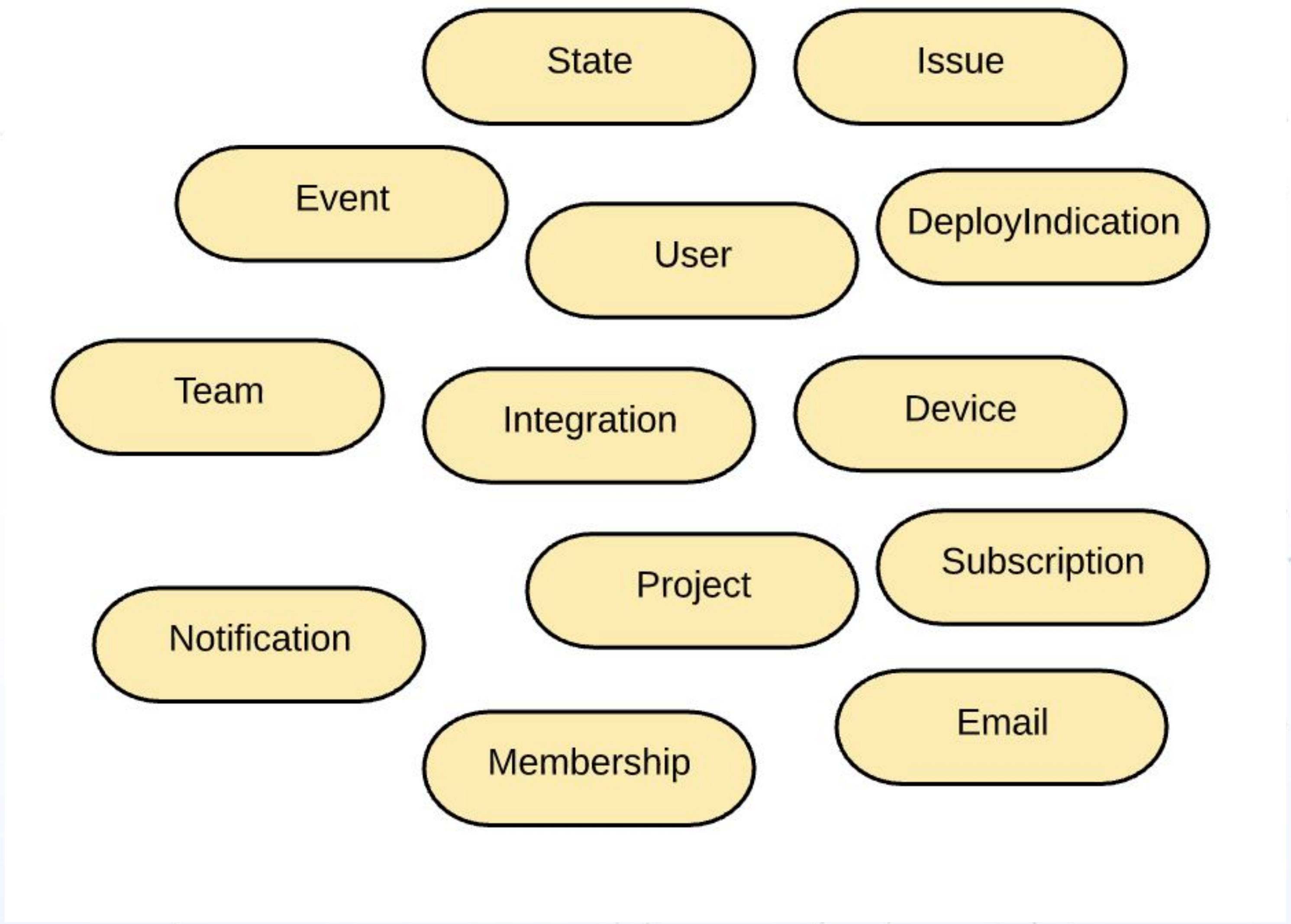
Domain Driven Design: Bounded Context



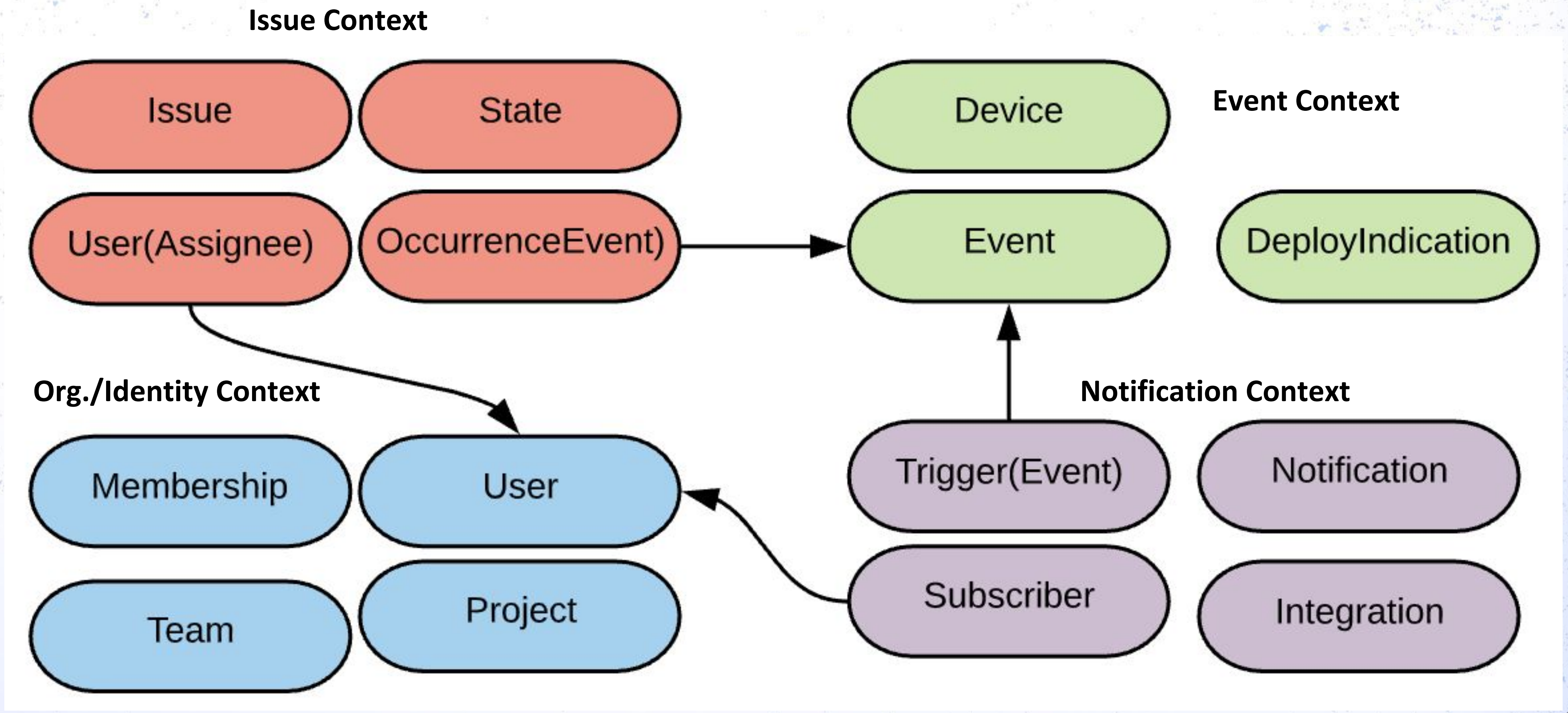
Split by Entity



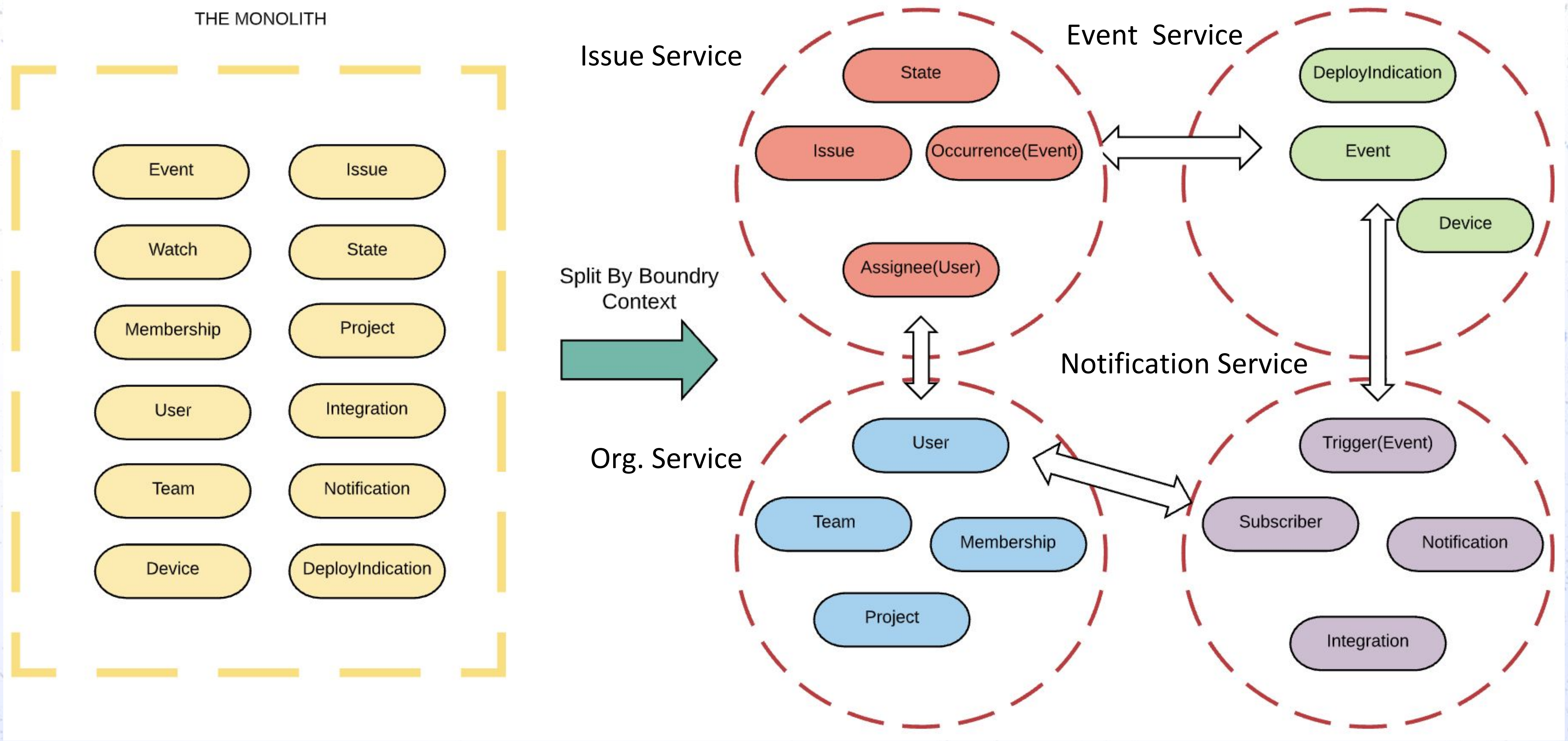
Domain Driven Design: Bounded Context



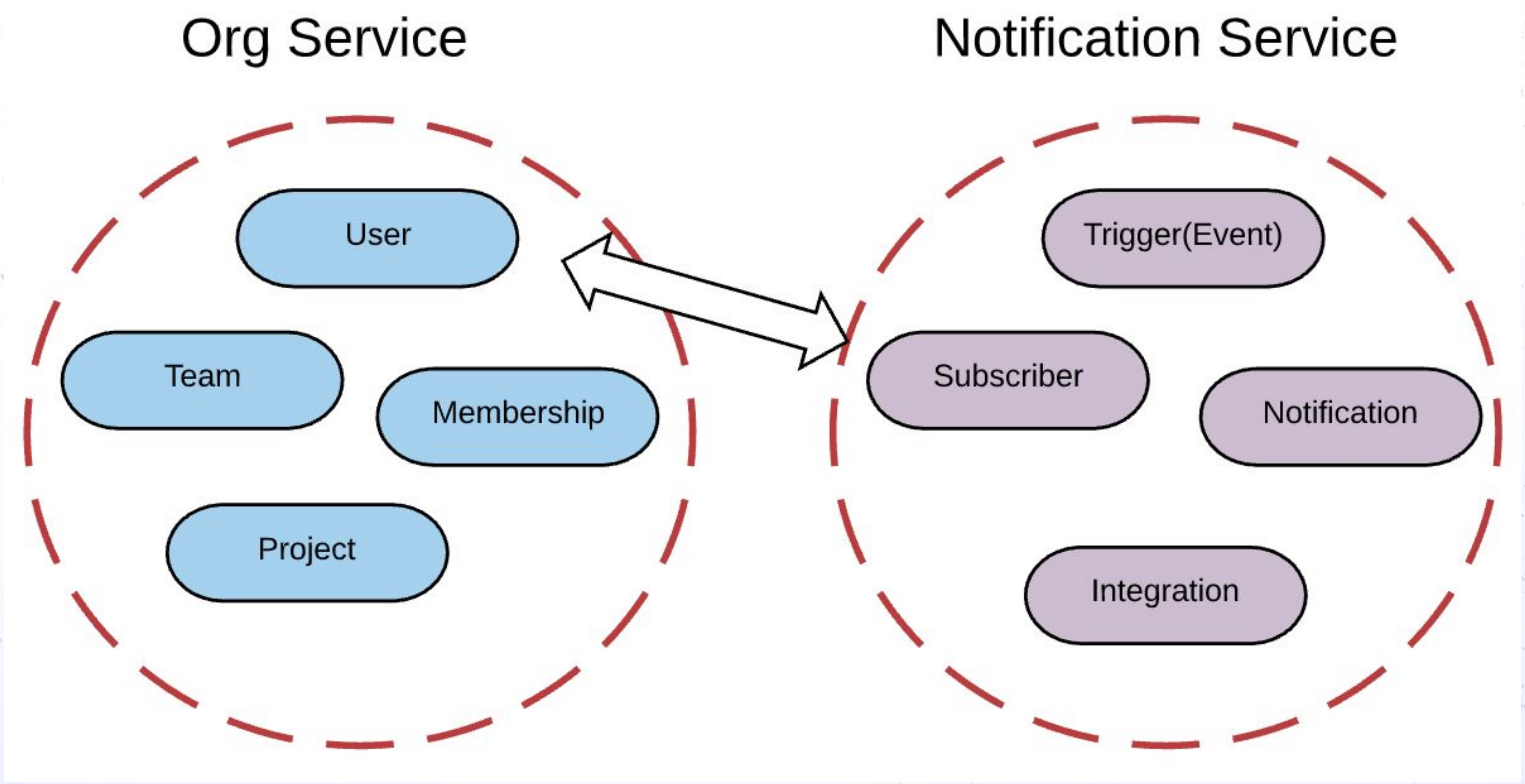
Domain Driven Design: Bounded Context



Domain Driven Design: Bounded Context



Domain Driven Design: Bounded Context



Domain Driven Design: Bounded Context

```
// User(User/Org. Context)
type User struct {
  ID string
  Name string
  Email string
  Password string
  TwoFactorAuth string
}
```

```
// “User”(Notification Context)
type Subscriber struct {
  ID string
  Name string
  Triggers []string
  Integrations []string
}
```

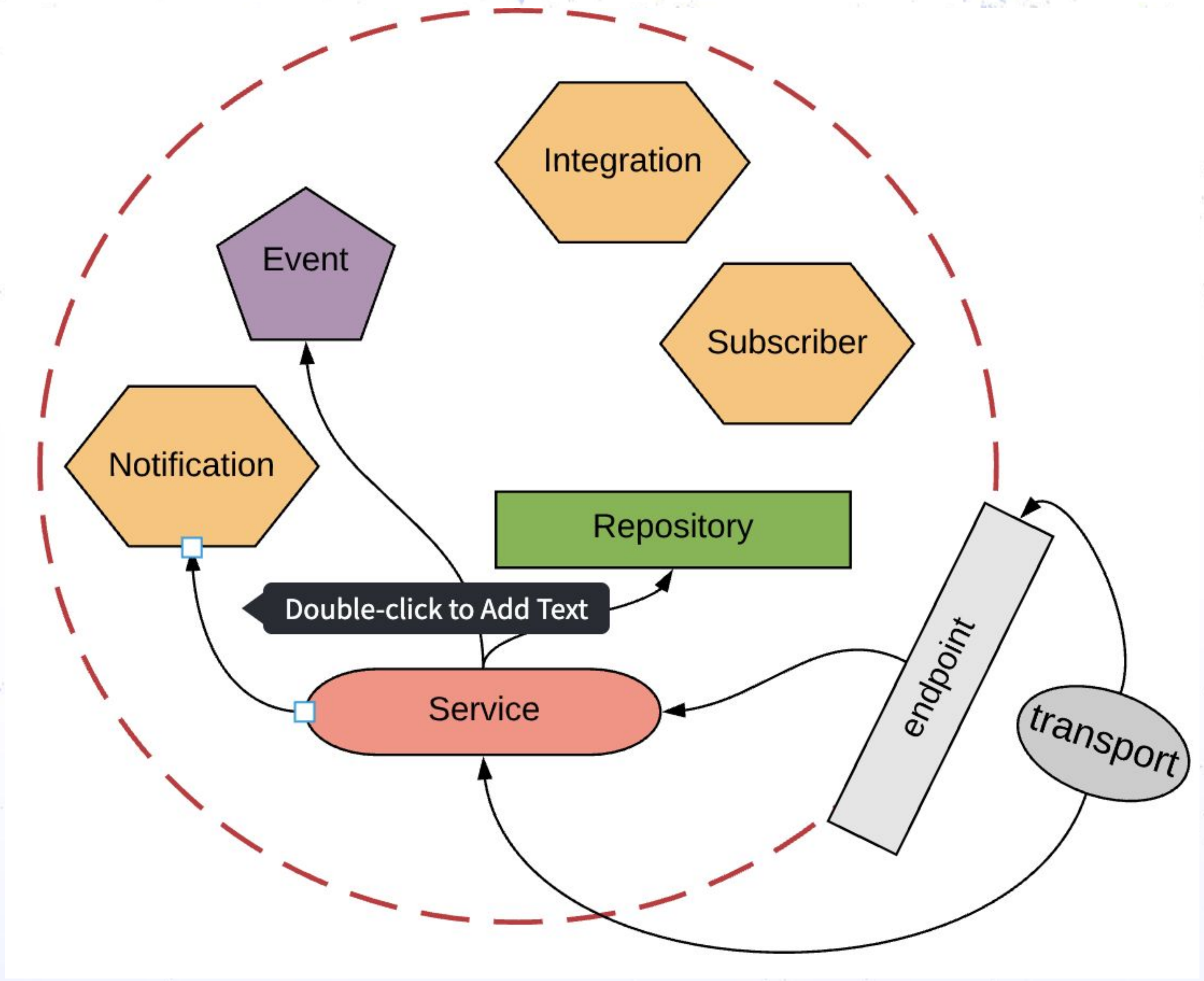
Domain Driven Design: Building Blocks

- **Building Blocks**:
  - **Entity, Aggregate**: Control all the entity/multi-entity specific BL
  - **Repository**: Delegates interaction with persistence
  - **Service**: Multi-entity control & decoupling between Repository and Entities/Aggregate
- Code organization

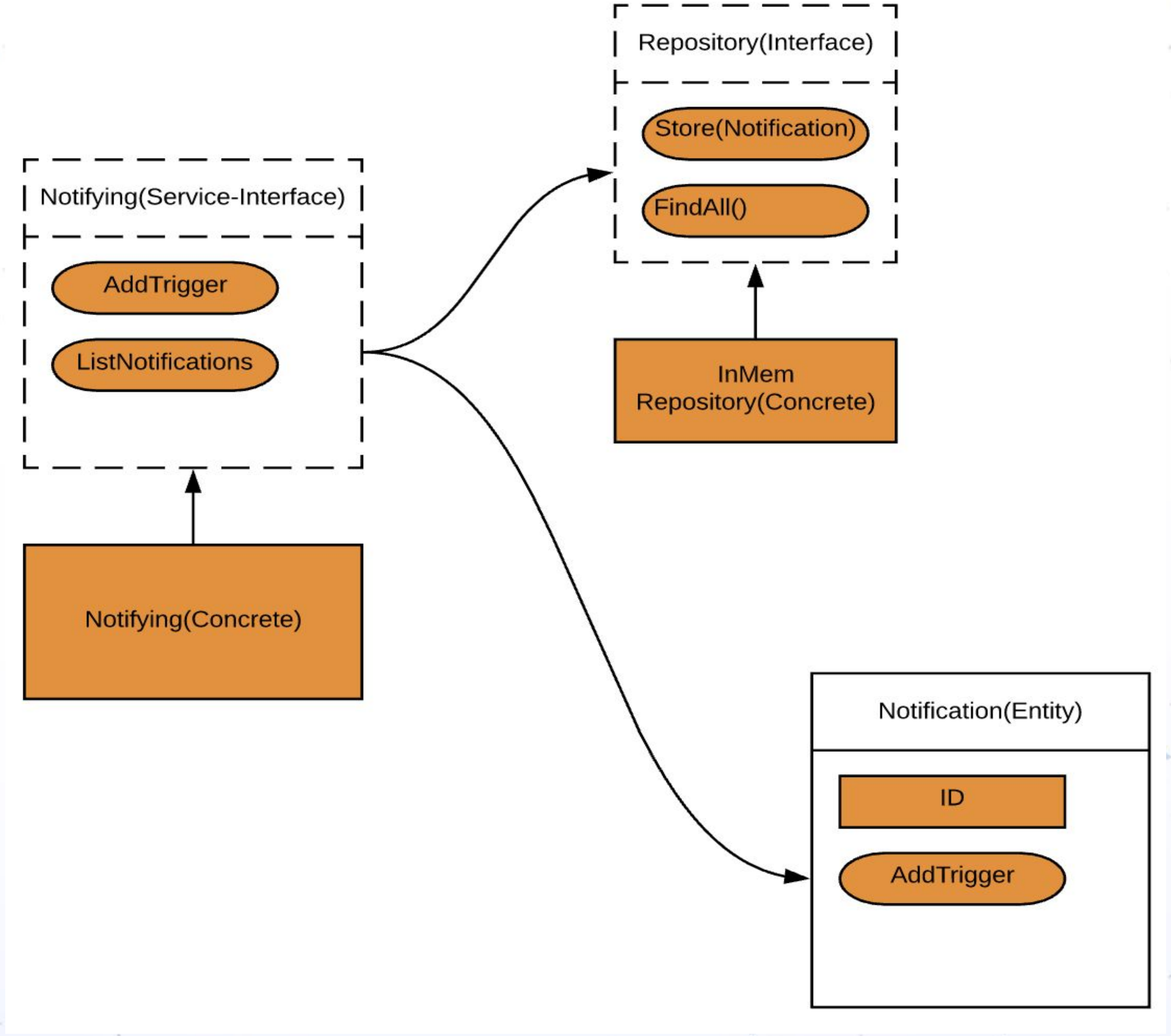
Domain Driven Design: Building Blocks

- **Context**: Event Notification
- **Language**: Triggers, Notification, Integrations, Subscriptions, Watching
- **Entities**: Notification, Subscriber, Integration
- **Value Objects**: Trigger, Subscription
- **Service**: Notifying, Integrating, Subscribing

Notification Service: Building Blocks



Notification Service: Building Blocks



Service Internals: DDD Entity

```go
package notification

// Entity
type Notification struct {
  ID string
  ev Event
  wasSent bool
  Recipients []string
}

// Value Object
type Event struct {
  Content string
  Link string
  OccurredAt time.Time
}
```

```go
func NewNotification() *Notification {
  return &Notification{
     ID: genrateRandomID(),
  }
}

func (n Notification) AddEvent(ev
Event) {
  ...
}
```

Notification Service: DDD Service

```
type Service interface {
  AcceptTrigger(ctx context.Context, ev
notification.Trigger) (string, error)

  ListNotifications(ctx context.Context)
([]notification.Notification, error)
}



type NotificationRepository interface {
  Store(ctx context.Context, *notification.Notification)
error
  FindAll(ctx context.Context)
([]notification.Notification, error)
}

type service struct {
  notifRepo NotificationRepository
}
```

```
func NewService(repo NotificationRepository) *service {
  return &service{repo}
}

func (s *service) AcceptTriggert(ctx context.Context, tr notification.Trigger)
(string, error) {
  notif := notification.New()
  notif.AddTrigger(tr)
  err := s.notifRepo.Store(ctx, notif)
  if err != nil {
     return "", err
  }

  return notif.ID, nil
}

func (s *service) ListNotifications(ctx context.Context)
([]notification.Notification, error) {
  return s.notifRepo.FindAll(ctx)
}
```

Notification Service: DDD Repository

```
package inmem

import (
  "context"
  "github.com/edkvm/scout/notification-service/notification"
  "sync"
)

type notificationRepo struct {
  mtx sync.RWMutex
  notifications map[string]notification.Notification
}

func NewNotificationRepo() *notificationRepo {
  return &notificationRepo{
     notifications: make(map[string]notification.Notification, 0),
  }
}
```

```
func (r *notificationRepo) Store(notif notification.Notification) error {
  r.mtx.Lock()
  defer r.mtx.Unlock()

  r.notifications[notif.ID] = notif

  return nil
}

func (r *notificationRepo) FindAll(_ context.Context)
([]notification.Notification, error) {

  notifs := make([]notification.Notification, 0, len(r.notifications))

  for _, v := range r.notifications {
     notifs = append(notifs, v)
  }

  return notifs, nil
}
```

Service Internals: Endpoint

```go
package notifying

import (
  "context"
  "github.com/edkvm/scout/notification-service/notification"
  "github.com/edkvm/scout/pkg/api"
)



func makeAcceptEventEndpoint(s Service) api.Endpoint {
  return func(ctx context.Context, req interface{}) (interface{}, error) {
     event := req.(notification.Event)
     result, err := s.AcceptEvent(ctx, event)
     if err != nil {
        return nil, err
     }
     return result, nil
  }
}
```

Service Internals: Transport/HTTP

```go
type transporter struct {
  service     Service
  makeHandler api.HandlerMaker
}

func NewTransport(s Service, handlerMaker api.HandlerMaker)
*transporter {
  return &transporter{service: s, makeHandler: handlerMaker}
}
```

```go
func (t *transporter) MakeRoutesDefinitions() http.Handler {
  r := httprouter.New()

  acceptEventHandler := t.makeHandler(
     makeAcceptEventEndpoint(t.service),
     decodeAcceptEventRequest,
     api.EncodeResponse,
  )

  r.POST("/notifications/accept", acceptEventHandler)

  return r
}
```

Notification Service: Connecting It All

```
func main() {

  notifInMem := inmem.NewNotificationRepo()
  notifService := notifying.NewService(notifInMem)
  notifServer := notifying.NewTransport(notifService, api.HandlerMaker)

  mux := http.NewServeMux()

  mux.Handle("/notifications/", notifServer)
  http.ListenAndServe(":6060", mux)

}
```

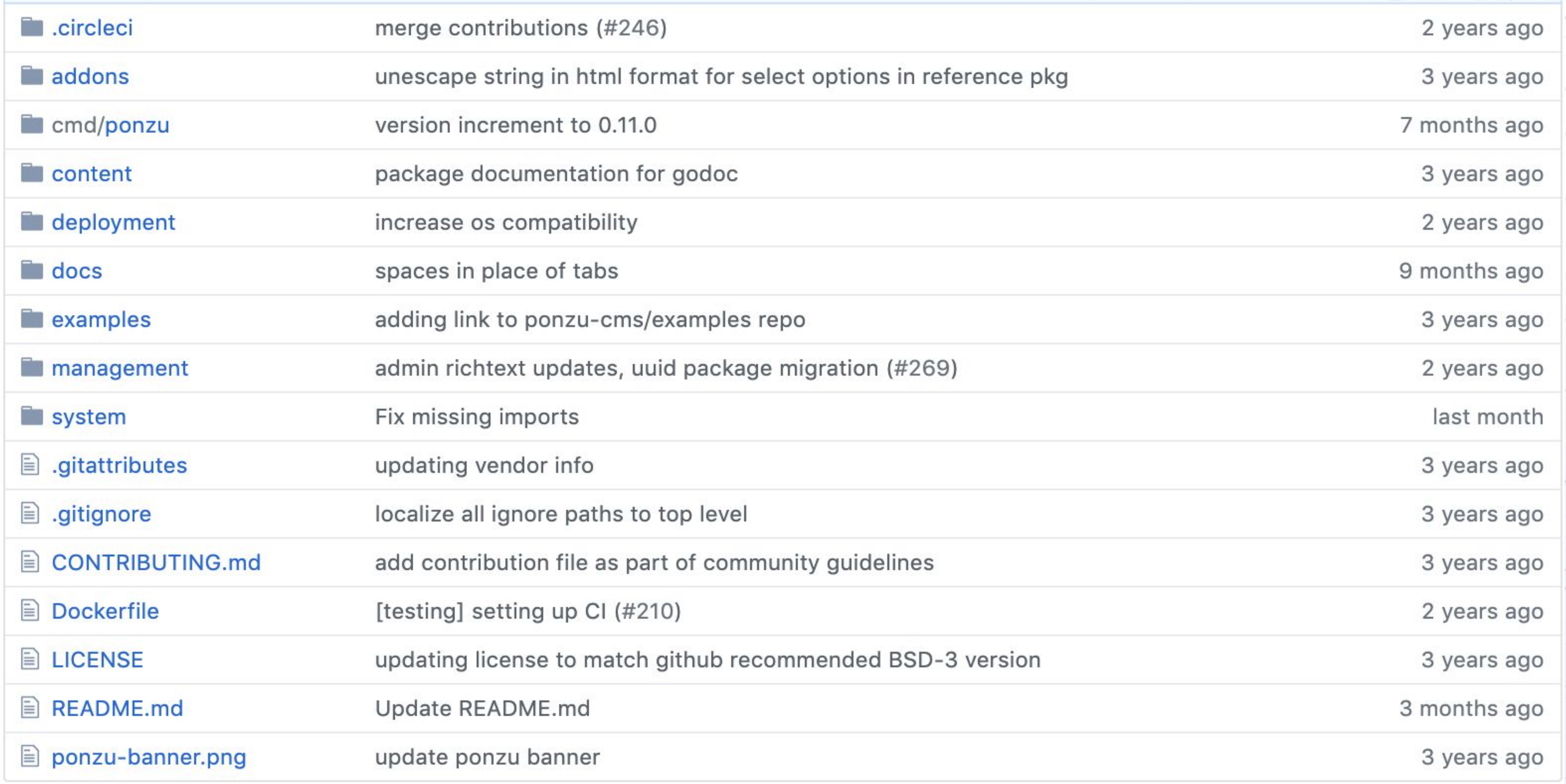
Notification Service: Additions

```
func NewAuditWrapper(s Service) *wrapper {
  return &wrapper{audit.New(), s}
}

func (w *wrapper) AcceptTriggert(ctx context.Context, tr
notification.Trigger) (string, error) {
  w.audit(ctx, tr)

  return w.s.AcceptTriggert(ctx,tr)
}

func (w *wrapper) ListNotifications(ctx context.Context)
([]notification.Notification, error) {
  w.audit(ctx)

  return w.s.ListNotifications(ctx)
}
```

Notification Service: Connecting It All

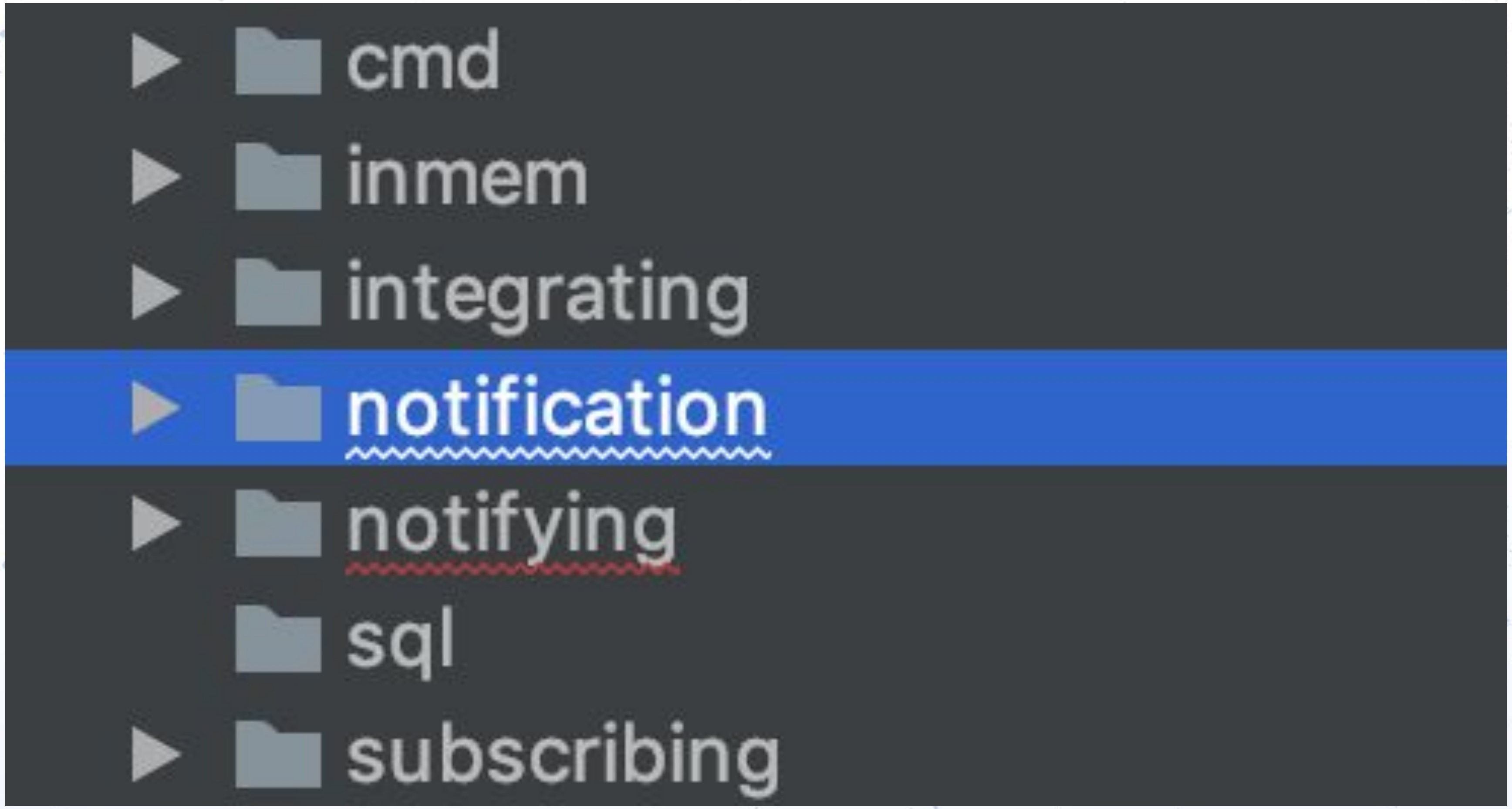
```
func main() {

  notifInMem := inmem.NewNotificationRepo()
  notifService := notifying.NewService(notifInMem)
  notifService := notifying.NewAuditWrapper(notifService)
  notifServer := notifying.NewTransport(notifService, api.HandlerMaker)

  mux := http.NewServeMux()

  mux.Handle("/notifications/", notifServer)
  http.ListenAndServe(":6060", mux)

}
```

Code Organization

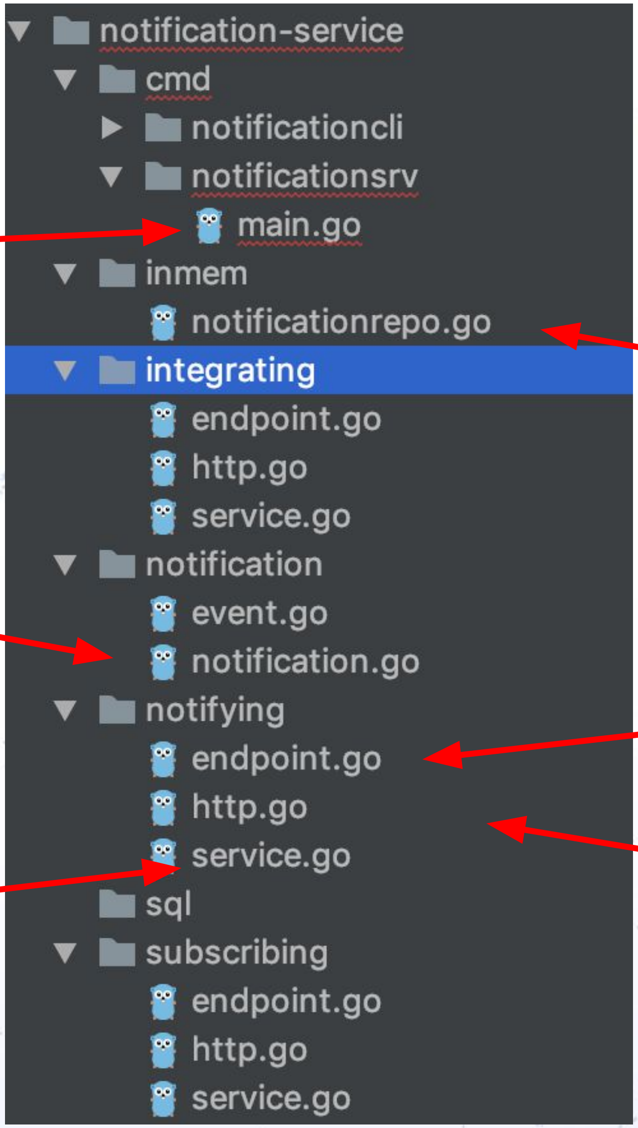


.circleci	merge contributions (#246)	2 years ago
addons	unescape string in html format for select options in reference pkg	3 years ago
cmd/ponzu	version increment to 0.11.0	7 months ago
content	package documentation for godoc	3 years ago
deployment	increase os compatibility	2 years ago
docs	spaces in place of tabs	9 months ago
examples	adding link to ponzu-cms/examples repo	3 years ago
management	admin richtext updates, uuid package migration (#269)	2 years ago
system	Fix missing imports	last month
.gitattributes	updating vendor info	3 years ago
.gitignore	localize all ignore paths to top level	3 years ago
CONTRIBUTING.md	add contribution file as part of community guidelines	3 years ago
Dockerfile	[testing] setting up CI (#210)	2 years ago
LICENSE	updating license to match github recommended BSD-3 version	3 years ago
README.md	Update README.md	3 months ago
ponzu-banner.png	update ponzu banner	3 years ago

Code Organization


cmd
inmem
integrating
notification
notifying
sql
subscribing

Notification Service: Code Organization



```
▼ notification-service
  ▼ cmd
    ▶ notificationcli
    ▼ notificationsrv
        main.go
  ▼ inmem
      notificationrepo.go
  ▼ integrating
      endpoint.go
      http.go
      service.go
  ▼ notification
      event.go
      notification.go
  ▼ notifying
      endpoint.go
      http.go
      service.go
    sql
  ▼ subscribing
      endpoint.go
      http.go
      service.go
```

Bind All the Components

One Type of Repo Implementation

Entity/Aggregate

Convert Service to Generic RPC

Bind RPC to Specific Transport

Access to Repo, Entity & Complex Actions

Conclusion

- Use DDD to model the system
  - Inform architectural decisions & internal design
- Organize your code in a consistent & informative way
- Don’t be afraid of <u>*some duplication*</u>

Go for Life!

@edkvm

Medium

References

- Cyrille Martraire: Hexagonal at Scale, with DDD and microservices!
- Martin Fowler: BoundedContext
- https://www.mirkosertic.de/blog/2013/04/domain-driven-design-example/
- https://about.sourcegraph.com/go/gophercon-2018-how-do-you-structure-your-go-apps