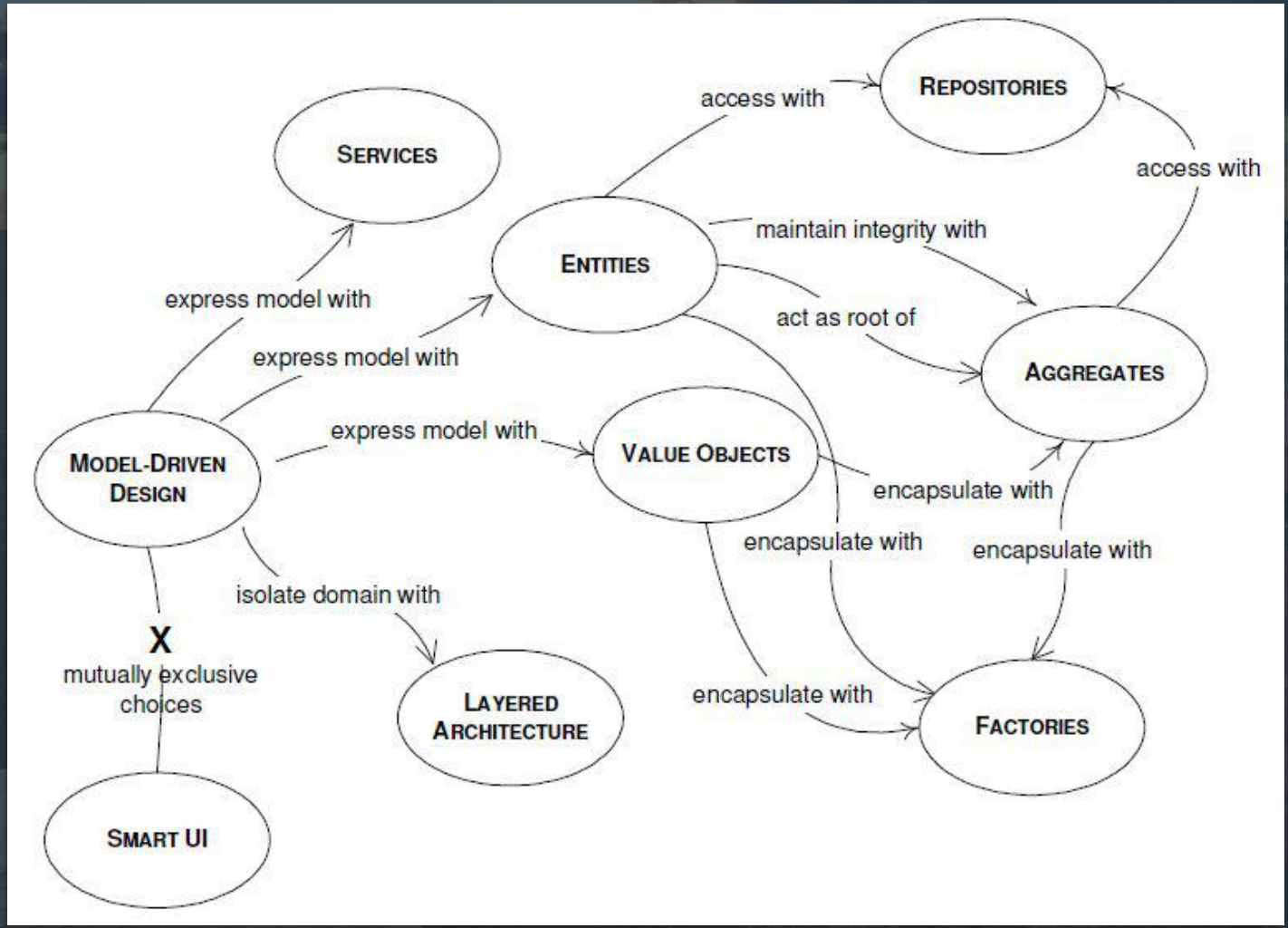

Procurios

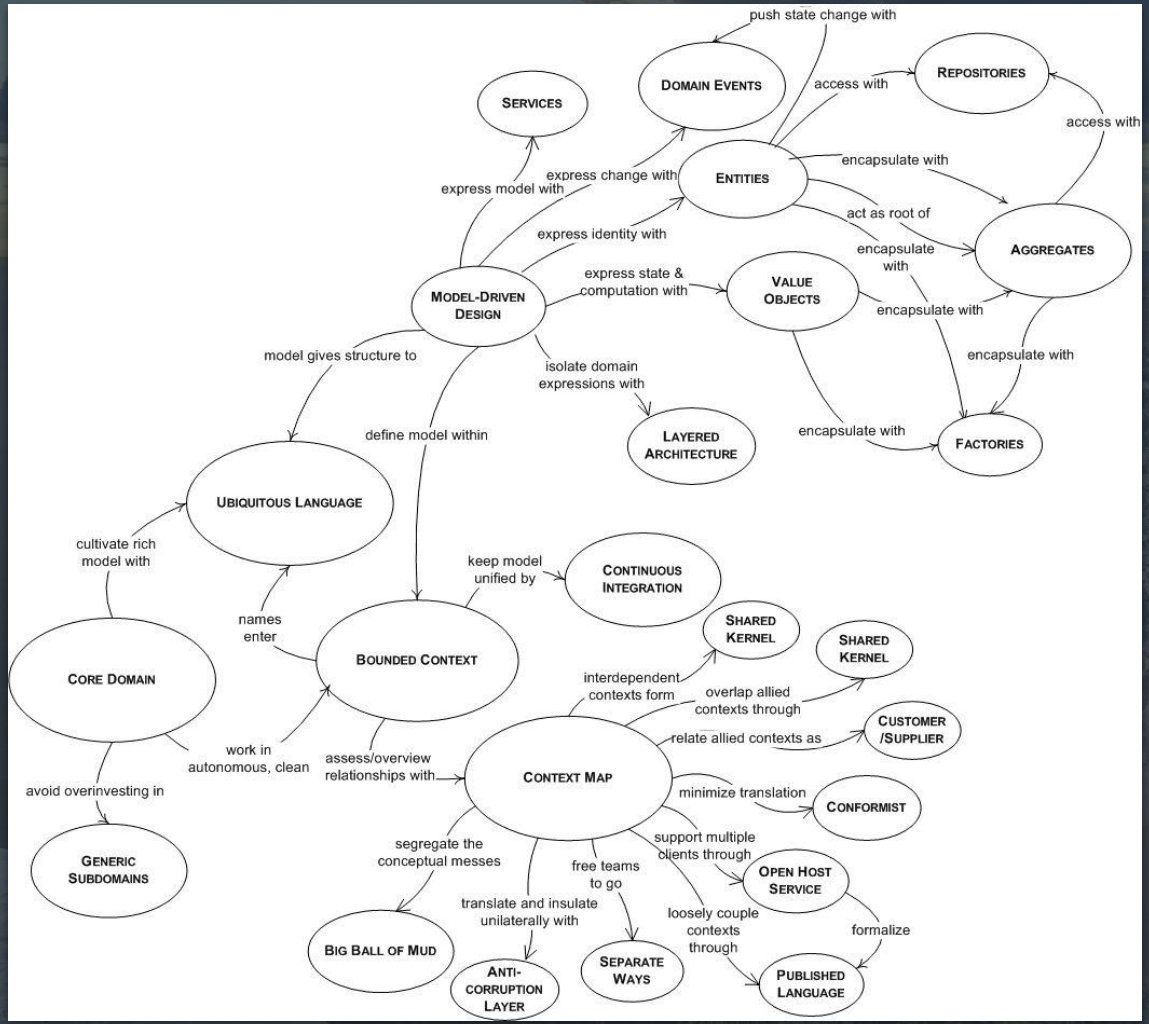
Technically DDD

DevDays Vilnius 2018
Slido.com with #K100

Pim Elshoff

developer.procurios.com
@pelshoff


SERVICES
REPOSITORIES
access with
access with
ENTITIES
maintain integrity with
express model with
act as root of
express model with
AGGREGATES
express model with
MODEL-DRIVEN DESIGN
VALUE OBJECTS
encapsulate with
encapsulate with
encapsulate with
isolate domain with
X
mutually exclusive choices
LAYERED ARCHITECTURE
encapsulate with
FACTORIES
SMART UI


push state change with
Domain Events
Repositories
access with
Services
access with
express model with
express change with
Entities
encapsulate with
express identity with
act as root of
Aggregates
encapsulate with
express state &
computation with
Model-Driven Design
Value Objects
encapsulate with
model gives structure to
isolate domain
expressions with
encapsulate with
Layered Architecture
encapsulate with
Factories
define model within
Ubiquitous Language
cultivate rich
model with
keep model
unified by
Continuous Integration
names
enter
Shared Kernel
Shared Kernel
Core Domain
Bounded Context
interdependent
contexts form
overlap allied
contexts through
Customer
/Supplier
relate allied contexts as
work in
autonomous, clean
assess/overview
relationships with
Context Map
minimize translation
Conformist
avoid overinvesting in
Generic
Subdomains
segregate the
conceptual messes
free teams
to go
support multiple
clients through
Open Host
Service
translate and insulate
unilaterally with
loosely couple
contexts
through
formalize
Big Ball of Mud
Anti-
corruption
Layer
Separate
Ways
Published
Language


Questions?

Context

Technical building blocks

Quiz

Case study I

Case study II

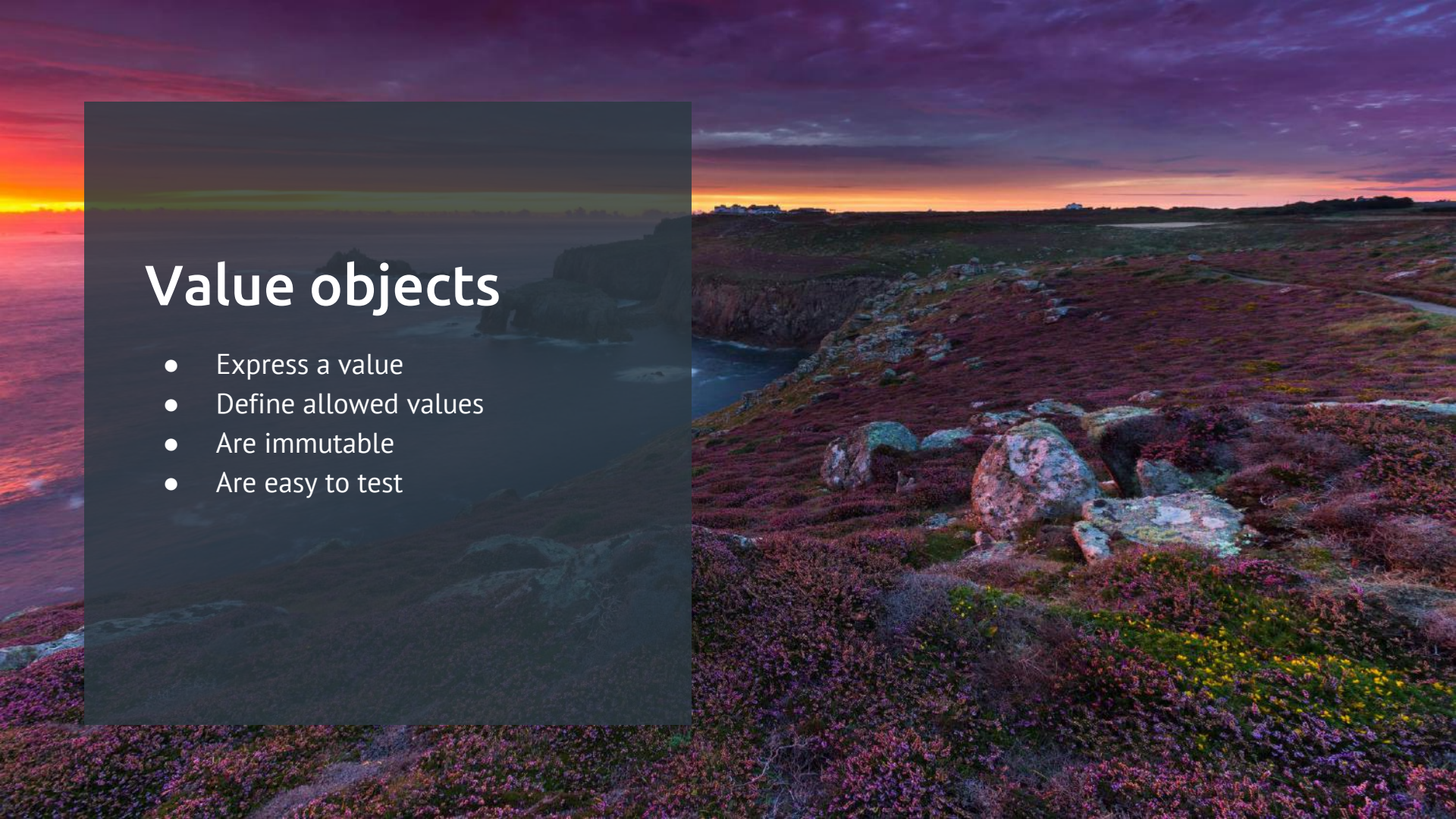

Procurios


Context
Technical building blocks
Quiz
Case study I
Case study II


Value objects

```php
final class Name {
    private $lastName;
    private $firstName;
    private $insertion;

    public function __construct(string $lastName, string $firstName, string $insertion) {
        $this->lastName = $lastName;
        $this->firstName = $firstName;
        $this->insertion = $insertion;
        $this->nameMustHaveLastName();
    }

    private function nameMustHaveLastName(): void {
        if (!$this->lastName) {
            throw InvalidName::becauseLastNameIsMissing();
        }
    }

    public function getLastName(): string { /**/ }
    public function getFirstName(): string { /**/ }
    public function getInsertion(): string { /**/ }
}
```

```php
final class EmailAddress {
    private $emailAddress;

    public function __construct(string $emailAddress) {
        $this->emailAddress = $emailAddress;
        $this->emailAddressMustBeAnActualEmailAddress();
    }

    private function emailAddressMustBeAnActualEmailAddress(): void {
        if (!filter_var($this->emailAddress, FILTER_VALIDATE_EMAIL)) {
            throw InvalidEmailAddress::becauseThisIsNotAnEmailAddress();
        }
    }

    public function asString(): string {
        return $this->emailAddress;
    }
}
```



Value objects

- Express a value
- Define allowed values
- Are immutable
- Are easy to test


Entities

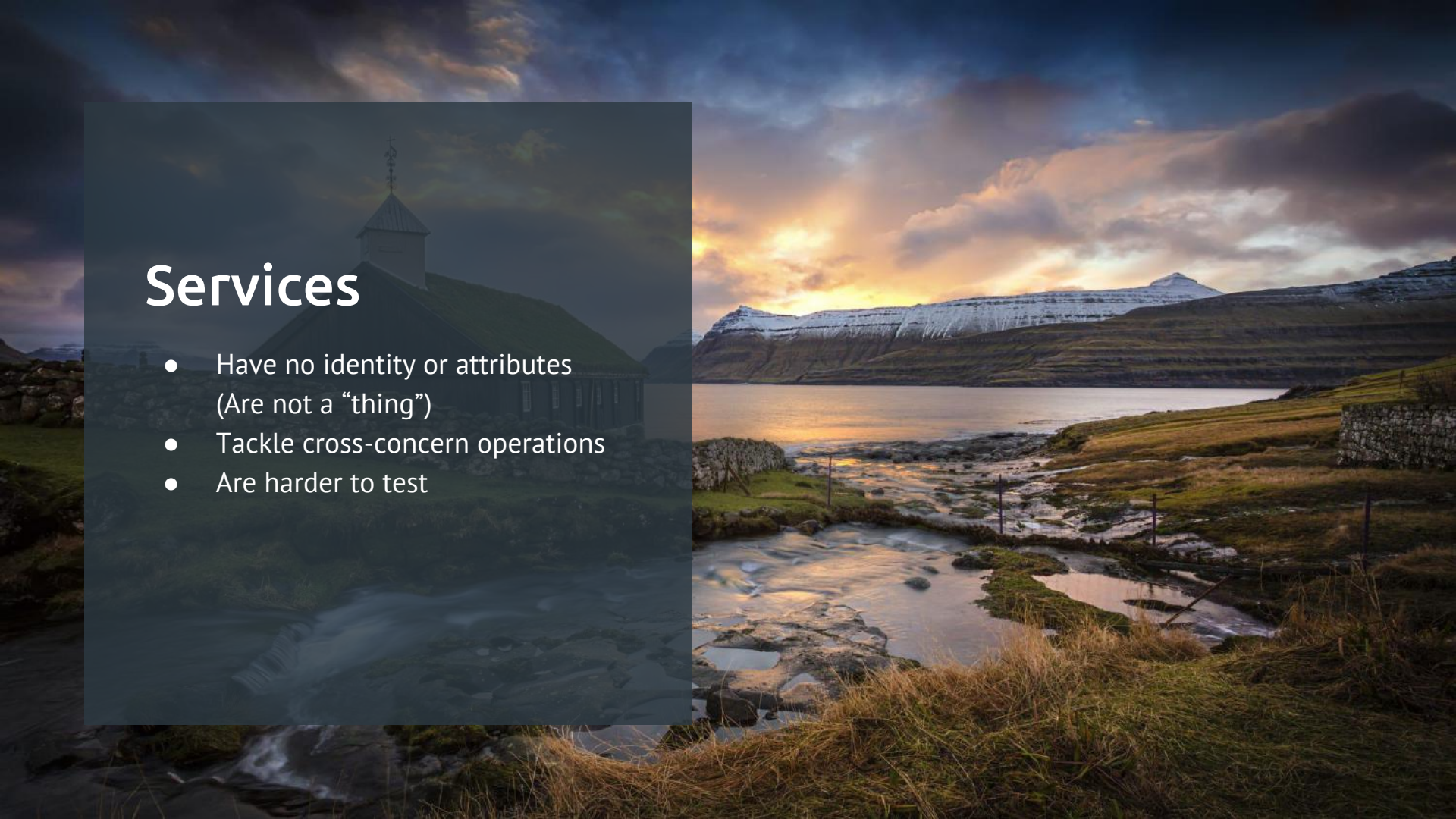
```php
final class Attendee {
    private $id;
    private $name;
    private $emailAddress;

    public function __construct(Uuid $id, Name $name, EmailAddress $emailAddress) {
        $this->id = $id;
        $this->name = $name;
        $this->emailAddress = $emailAddress;
    }

    public function getId(): Uuid { /**/ }
    public function getName(): Name { /**/ }
    public function setName(Name $name): Attendee { /**/ }
    public function getEmailAddress(): EmailAddress { /**/ }

    public function setEmailAddress(EmailAddress $emailAddress): Attendee {
        $this->emailAddress = $emailAddress;
        return $this;
    }
}
```

Entities

- Have identity
- Are more than their attributes
- Evolve over time
- Are slightly harder to test

Services

```php
final class Registration {
    private $repository;

    public function __construct(AttendeeRepository $repository) {
        $this->repository = $repository;
    }

    public function registerNew(Attendee $attendee): void {
        $this->emailAddressMustBeUnique($attendee->getEmailAddress());
        $this->repository->add($attendee);
    }

    private function emailAddressMustBeUnique(EmailAddress $emailAddress): void {
        try {
            $this->repository->findByEmailAddress($emailAddress);
        } catch (AttendeeNotFound $e) {
            return;
        }
        throw RegistrationFailed::becauseEmailAddressIsNotUnique();
    }
}
```



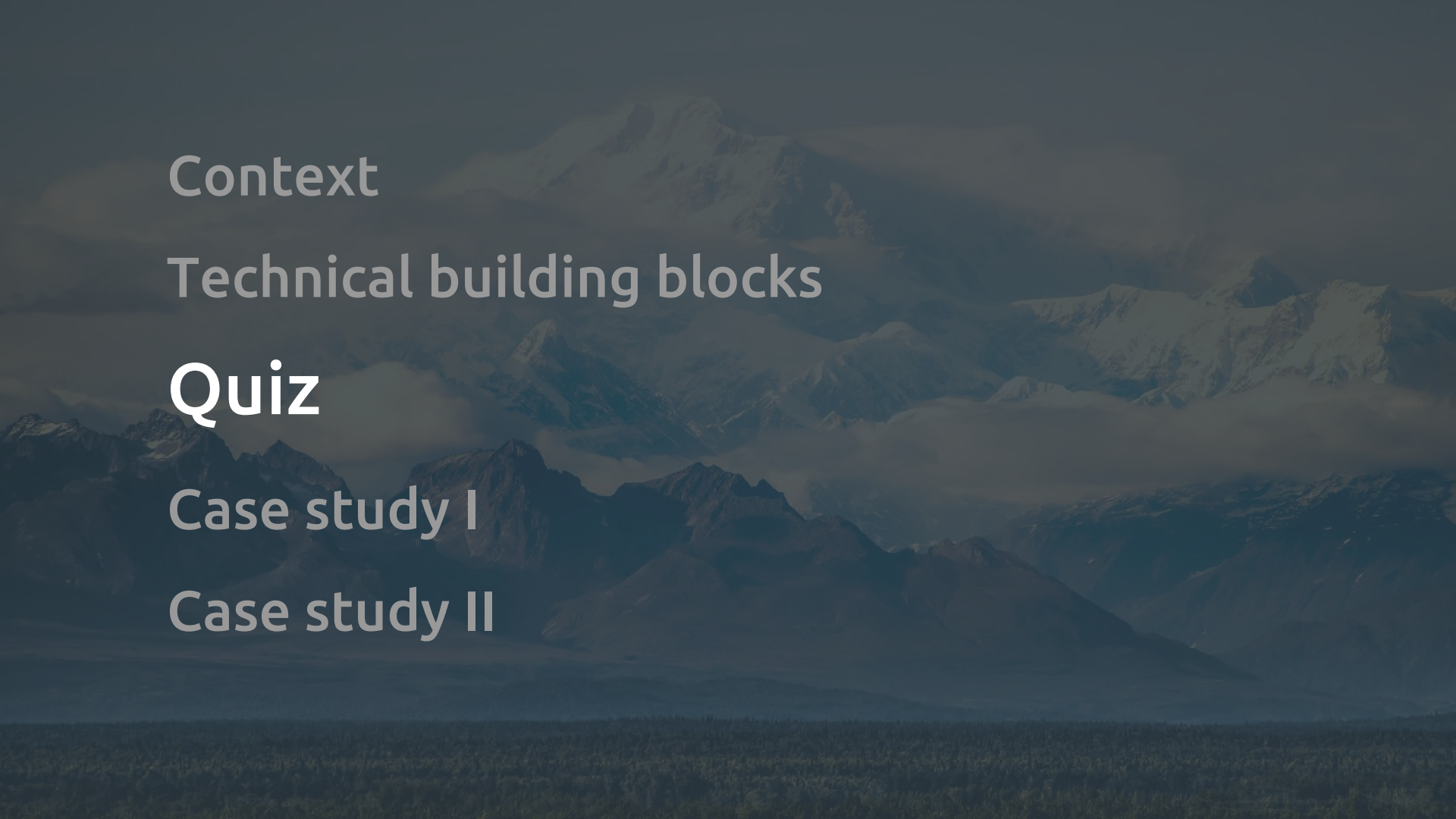
Services

- Have no identity or attributes (Are not a “thing”)
- Tackle cross-concern operations
- Are harder to test


Repositories

Repositories

- Are a collection of (almost) all objects of a type


Context
Technical building blocks
Quiz
Case study I
Case study II


final class Address


final class BicycleService

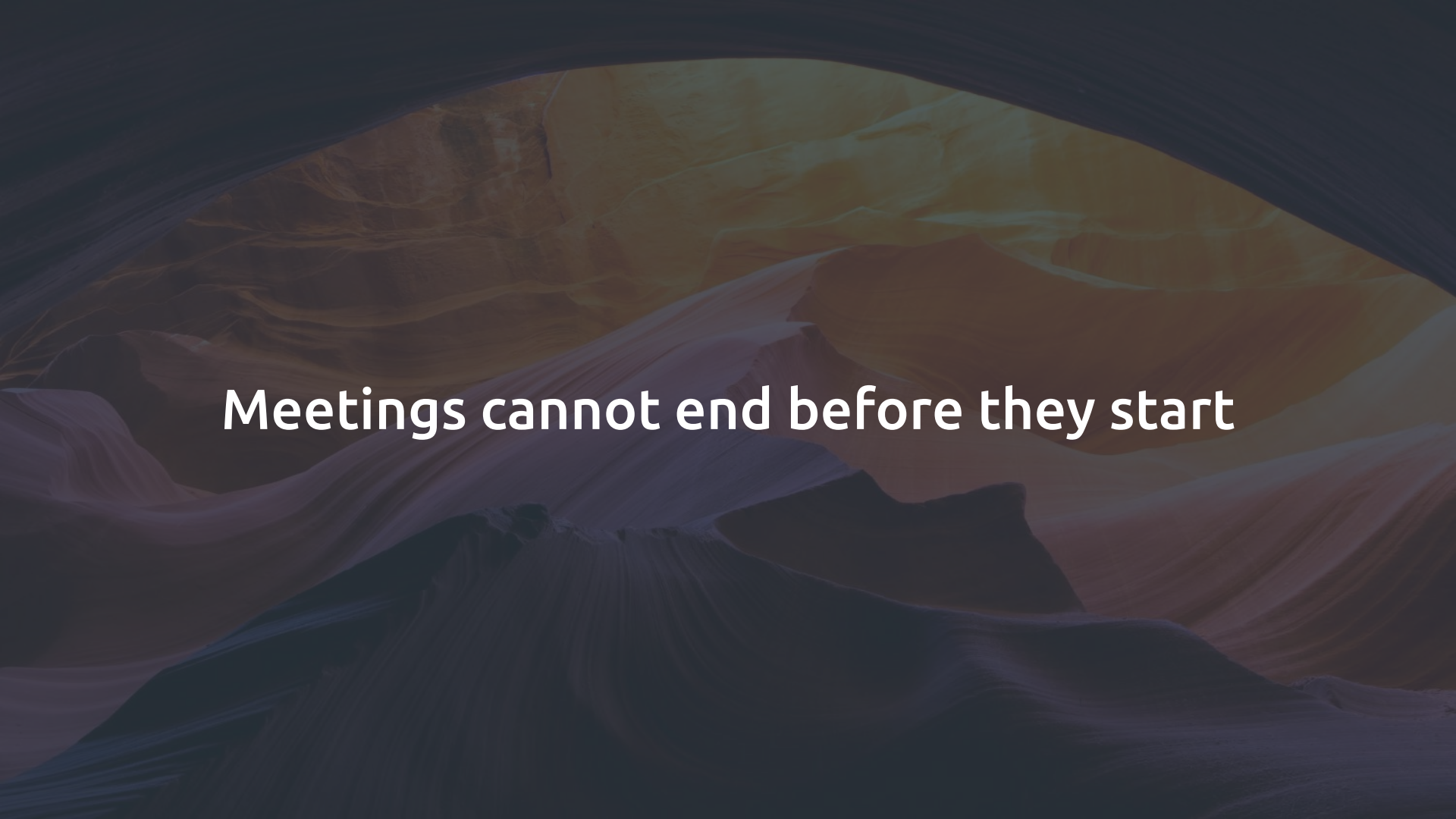

final class StreetAddress


final class Investment


Context
Technical building blocks
Quiz
Case study I
Case study II

```php
final class Meeting {
   private $meetingId;
   private $title;
   private $description;
   private $code;
   private $startDate;
   private $endDate;
   private $startTime;
   private $endTime;
   private $isPublished;
   private $subTitle;
   private $program;

   public function __construct(Uuid $meetingId, string $title, string $description, string
       $code, string $startDate, string $endDate, string $startTime, string $endTime, bool
       $isPublished, string $subTitle, array $program) {
       $this->meetingId = $meetingId;
       $this->title = $title;
       // ...
   }
}
```

```
new Meeting(
   Uuid::generate(), 'DevDays Vilnius 2018', '...', '#K100',
   '2018-05-22', '2018-05-24', '09:00', '18:00', false,
   'Software Development Conference Created for Developers, by Developers',
   [
       [
           'date' => '2018-05-23',
           'startTime' => '15:55',
           'endTime' => '16:40',
           'title' => 'Technically DDD',
           'room' => 'Hall 1',
       ],
       [
           'date' => '2018-05-23',
           'startTime' => '16:45',
           'endTime' => '17:30',
           'title' => 'Closing Keynote',
           'room' => 'Hall 1',
       ],
   ]
);
```


Meetings cannot end before they start

The Approach™

1. Implement in Entity
2. Extract Value Object
3. Refactor Value Object

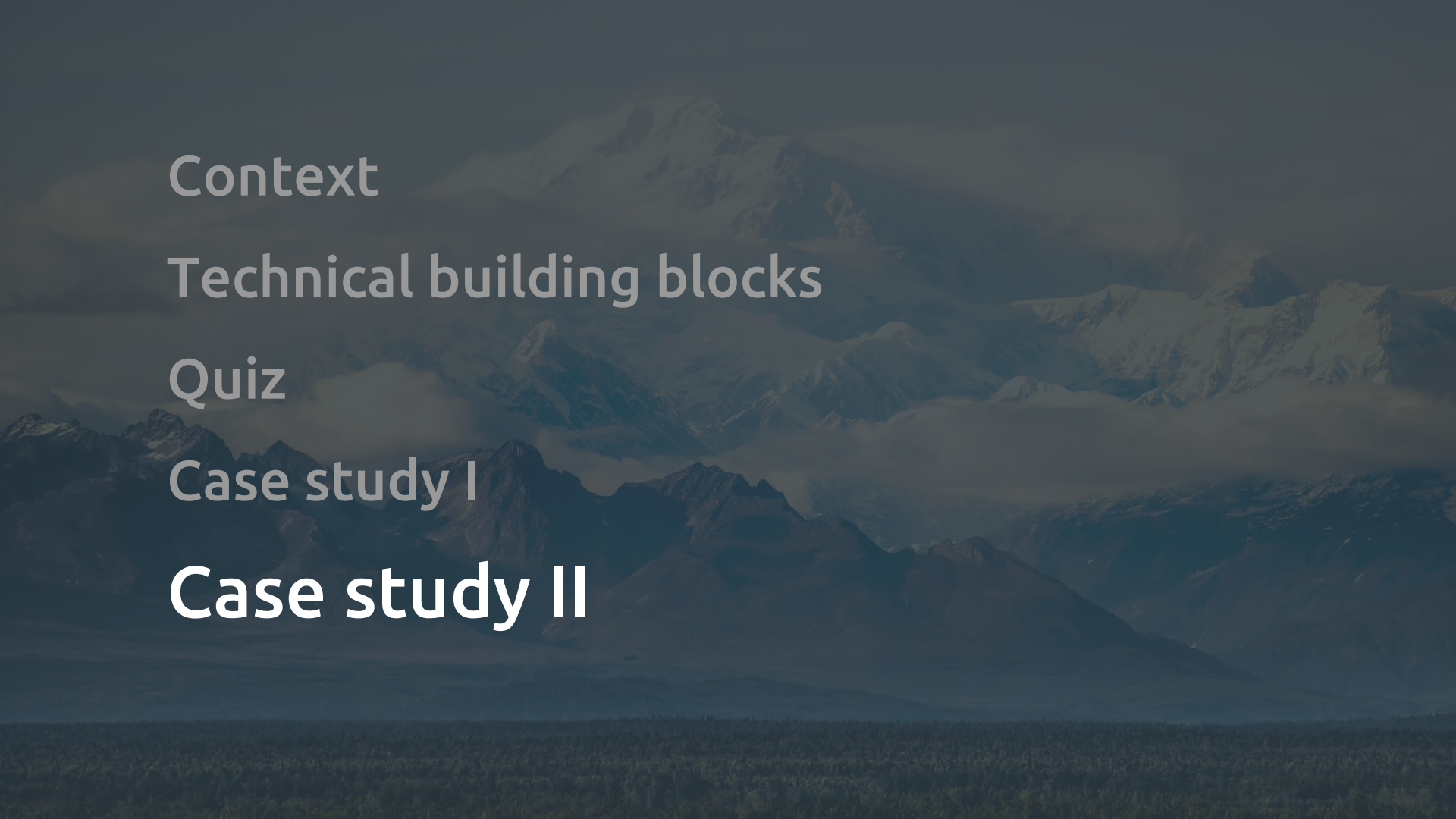
```php
final class Meeting {
    public function __construct(/**/) {
        // ..
        $this->meetingCannotEndBeforeStart();
    }

    private function meetingCannotEndBeforeStart(): void {
        if ($this->startDate < $this->endDate) {
            return;
        }
        if ($this->startDate > $this->endDate) {
            throw InvalidMeeting::becauseMeetingEndsBeforeStarting();
        }
        if ($this->startTime > $this->endTime) {
            throw InvalidMeeting::becauseMeetingEndsBeforeStarting();
        }
    }
}
```

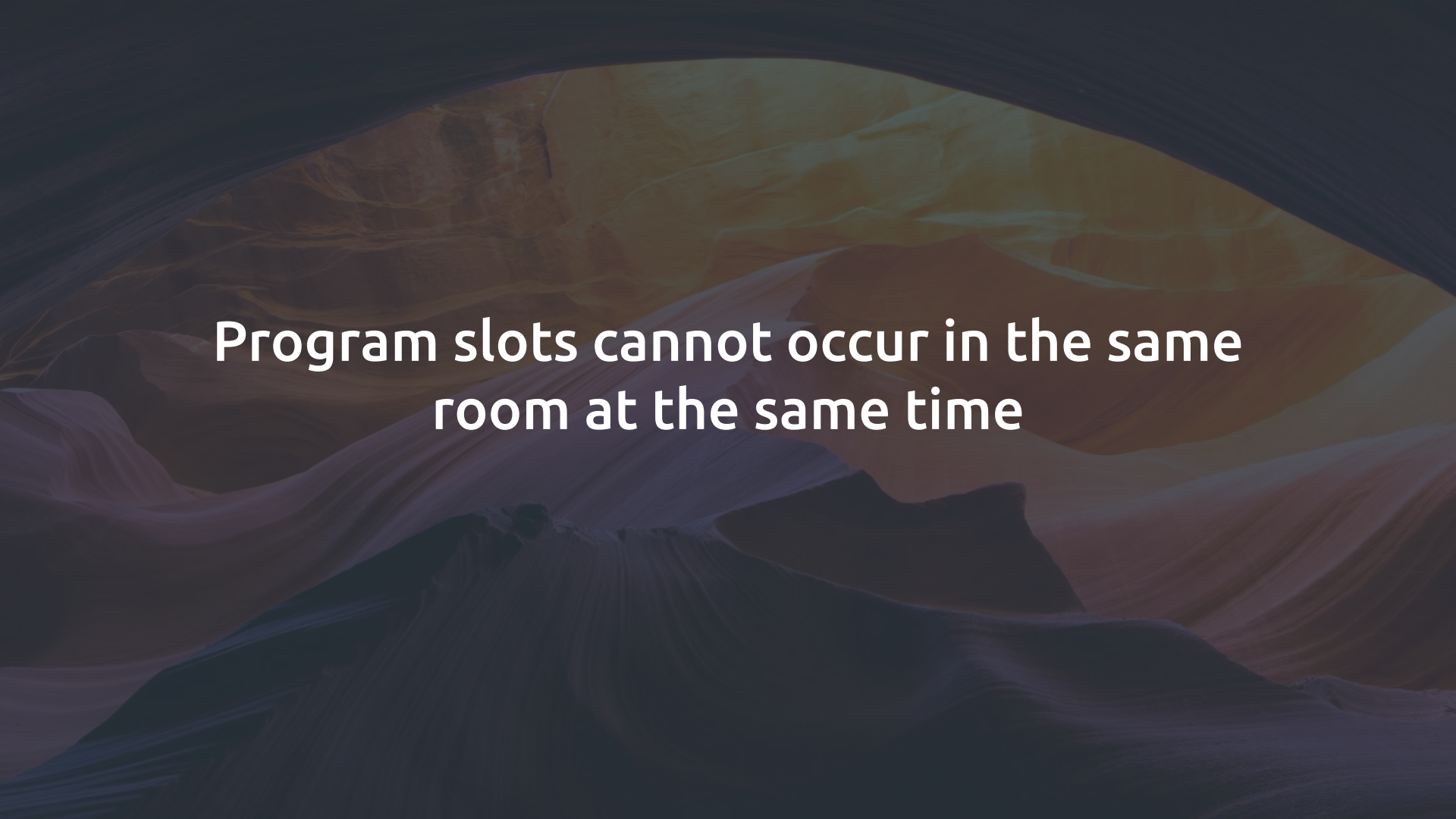
```php
final class Meeting {
    private $meetingId;
    private $title;
    private $description;
    private $code;
    private $duration;
    private $isPublished;
    private $subTitle;
    private $program;

    public function __construct(Uuid $meetingId, string $title, string $description, string
        $code, MeetingDuration $duration, bool $isPublished, string $subTitle, array $program) {
        $this->meetingId = $meetingId;
        $this->title = $title;
        $this->description = $description;
        $this->code = $code;
        $this->duration = $duration;
        $this->isPublished = $isPublished;
        $this->subTitle = $subTitle;
        $this->program = $program;
    }
}
```

```php
final class MeetingDuration {
    public function __construct(string $startDate, string $endDate, string $startTime, string
        $endTime) {
        $this->startDate = $startDate;
        $this->endDate = $endDate;
        $this->startTime = $startTime;
        $this->endTime = $endTime;
        $this->meetingCannotEndBeforeStart();
    }

    private function meetingCannotEndBeforeStart(): void {
        if ($this->startDate < $this->endDate) {
            return;
        }
        if ($this->startDate > $this->endDate) {
            throw InvalidMeetingDuration::becauseDurationEndsBeforeStarting();
        }
        if ($this->startTime > $this->endTime) {
            throw InvalidMeetingDuration::becauseDurationEndsBeforeStarting();
        }
    }
}
```

```php
final class MeetingDuration {
   private $start;
   private $end;

   public function __construct(DateTimeImmutable $start, DateTimeImmutable $end) {
       $this->start = $start;
       $this->end = $end;
       $this->meetingCannotEndBeforeStart();
   }

   private function meetingCannotEndBeforeStart(): void {
       if ($this->start > $this->end) {
           throw InvalidMeetingDuration::becauseDurationEndsBeforeStarting();
       }
   }
}
```

```
new Meeting(
   Uuid::generate(), 'DevDays Vilnius 2018', '...', '#K100',
   new MeetingDuration(
       new DateTimeImmutable('2018-05-22' . ' ' . '09:00'),
       new DateTimeImmutable('2018-05-24' . ' ' . '18:00')
   ), false, 'Software Development Conference Created for Developers, by Developers',
   [
       [
           'date' => '2018-05-23',
           'startTime' => '15:55',
           'endTime' => '16:40',
           'title' => 'Technically DDD',
           'room' => 'Hall 1',
       ],
       [
           'date' => '2018-05-23',
           'startTime' => '16:45',
           'endTime' => '17:30',
           'title' => 'Closing Keynote',
           'room' => 'Hall 1',
       ], // ..
```


Context
Technical building blocks
Quiz
Case study I
Case study II


Program slots cannot occur in the same room at the same time

```
[
    [
        'date' => '2018-05-23',
        'startTime' => '15:55',
        'endTime' => '16:40',
        'title' => 'Technically DDD',
        'room' => 'Hall 1',
    ],
    [
        'date' => '2018-05-23',
        'startTime' => '16:45',
        'endTime' => '17:30',
        'title' => 'Closing Keynote',
        'room' => 'Hall 1',
    ],
]
```

```php
private function programSlotsCannotOccurInTheSameRoomAtTheSameTime(): void {
    foreach ($this->program as $index => $slot) {
        foreach (array_slice($this->program, $index + 1) as $comparison) {
            if ($slot['room'] !== $comparison['room']) {
                continue;
            }
            if ($slot['date'] !== $comparison['date']) {
                continue;
            }
            if ($slot['startTime'] >= $comparison['endTime']) {
                continue;
            }
            if ($slot['endTime'] <= $comparison['startTime']) {
                continue;
            }
            throw InvalidProgram::becauseProgramSlotsOverlap();
        }
    }
}
```

```php
final class Meeting {
   private $meetingId;
   private $title;
   private $description;
   private $code;
   private $duration;
   private $isPublished;
   private $subTitle;
   private $program;

   public function __construct(Uuid $meetingId, string $title, string $description, string
       $code, MeetingDuration $duration, bool $isPublished, string $subTitle, Program $program){
       $this->meetingId = $meetingId;
       $this->title = $title;
       $this->description = $description;
       $this->code = $code;
       $this->duration = $duration;
       $this->isPublished = $isPublished;
       $this->subTitle = $subTitle;
       $this->program = $program;
   }
}
```

```php
final class Program {
    // ..
    private function programSlotsCannotOccurInTheSameRoomAtTheSameTime(): void {
        foreach ($this->program as $index => $slot) {
            foreach (array_slice($this->program, $index + 1) as $comparison) {
                if ($slot['room'] !== $comparison['room']) {
                    continue;
                }
                if ($slot['date'] !== $comparison['date']) {
                    continue;
                }
                if ($slot['startTime'] >= $comparison['endTime']) {
                    continue;
                }
                if ($slot['endTime'] <= $comparison['startTime']) {
                    continue;
                }
                throw InvalidProgram::becauseProgramSlotsOverlap();
            }
        }
    } // ..
```

```php
final class Program {
    private $program;

    /** @param Slot[] $program */
    public function __construct(array $program) {
        $this->program = $program;
        $this->programSlotsCannotOccurInTheSameRoomAtTheSameTime();
    }

    private function programSlotsCannotOccurInTheSameRoomAtTheSameTime(): void {
        foreach ($this->program as $index => $thisSlot) {
            foreach (array_slice($this->program, $index + 1) as $thatSlot) {
                if ($thisSlot->overlapsWith($thatSlot)) {
                    throw InvalidProgram::becauseProgramSlotsOverlap();
                }
            }
        }
    }
}
```

```php
final class Slot {
    private $duration;
    private $title;
    private $room;

    public function __construct(MeetingDuration $duration, string $title, string $room) {
        $this->duration = $duration;
        $this->title = $title;
        $this->room = $room;
    }

    public function overlapsWith(Slot $that): bool {
        return $this->room === $that->room
            && $this->duration->overlapsWith($that->duration);
    }
}
```

```php
final class MeetingDuration {
   private $start;
   private $end;

   public function __construct(DateTimeImmutable $start, DateTimeImmutable $end) { /**/ }

   private function meetingCannotEndBeforeStart(): void { /**/ }

   public function overlapsWith(MeetingDuration $that): bool {
       return $this->start >= $that->start && $this->start <= $that->end
           || $this->end >= $that->start && $this->end <= $that->end
           || $that->start >= $this->start && $that->start <= $this->end
           || $that->end >= $this->start && $that->end <= $this->end;
   }
}
```

```php
final class MeetingDuration {
   private $start;
   private $end;

   public function __construct(DateTimeImmutable $start, DateTimeImmutable $end) { /**/ }

   private function meetingCannotEndBeforeStart(): void { /**/ }

   public function overlapsWith(MeetingDuration $that): bool {
       return $that->contains($this->start) || $that->contains($this->end)
           || $this->contains($that->start) || $this->contains($that->end);
   }

   private function contains(DateTimeImmutable $date): bool {
       return $date >= $this->start && $date <= $this->end;
   }
}
```

```php
final class MeetingDuration {
   private $start;
   private $end;

   public function __construct(DateTimeImmutable $start, DateTimeImmutable $end) { /**/ }

   private function meetingCannotEndBeforeStart(): void { /**/ }

   public function overlapsWith(MeetingDuration $that): bool {
       return !($this->start > $that->end || $that->start > $this->end);
   }
}
```

```php
final class MeetingDuration {
   private $start;
   private $end;

   public function __construct(DateTimeImmutable $start, DateTimeImmutable $end) { /**/ }

   private function meetingCannotEndBeforeStart(): void { /**/ }

   public function overlapsWith(MeetingDuration $that): bool {
       return !$this->before($that) && !$that->before($this);
   }

   private function before(MeetingDuration $that): bool {
       return $that->start > $this->end;
   }
}
```

```
[
    'date' => '2018-05-23',
    'startTime' => '15:55',
    'endTime' => '16:40',
    'title' => 'Technically DDD',
    'room' => 'Hall 1',
],
```

```
private function before(MeetingDuration $that): bool {
    return $that->start > $this->end;
}
```

```php
final class SlotDuration {
    private $start;
    private $end;

    public function __construct(DateTimeImmutable $start, DateTimeImmutable $end) { /**/ }

    private function slotCannotEndBeforeStart(): void { /**/ }

    private function slotMustStartAndEndOnTheSameDay(): void {
        if ($this->start->diff($this->end)->days >= 1) {
            throw InvalidSlotDuration::becauseSlotEndsOnDifferentDay();
        }
    }

    public function overlapsWith(SlotDuration $that): bool {
        return !$this->before($that) && !$that->before($this);
    }

    private function before(SlotDuration $that): bool {
        return $that->start >= $this->end;
    }
}
```

```
new Meeting(
   Uuid::generate(), 'DevDays Vilnius 2018', '...', '#K100',
   '2018-05-22', '2018-05-24', '09:00', '18:00', false,
   'Software Development Conference Created for Developers, by Developers',
   [
       [
           'date' => '2018-05-23',
           'startTime' => '15:55',
           'endTime' => '16:40',
           'title' => 'Technically DDD',
           'room' => 'Hall 1',
       ],
       [
           'date' => '2018-05-23',
           'startTime' => '16:45',
           'endTime' => '17:30',
           'title' => 'Closing Keynote',
           'room' => 'Hall 1',
       ],
   ]
);
```

```
new Meeting(
    Uuid::generate(), 'DevDays Vilnius 2018', '...', '#K100',
    new MeetingDuration(
        new DateTimeImmutable('2018-05-22' . ' ' . '09:00'),
        new DateTimeImmutable('2018-05-24' . ' ' . '18:00')
    ), false, 'Software Development Conference Created for Developers, by Developers',
    new Program([
        new Slot(
            new SlotDuration(
                new DateTimeImmutable('2018-05-23' . ' ' . '15:55'),
                new DateTimeImmutable('2018-05-23' . ' ' . '16:40')
            ),
            'Technically DDD', 'Hall 1'
        ),
        new Slot(
            new SlotDuration(
                new DateTimeImmutable('2018-05-23' . ' ' . '16:45'),
                new DateTimeImmutable('2018-05-23' . ' ' . '17:30')
            ),
            'Closing Keynote', 'Hall 1'
        ),
    ]) // ..
```


Procurios

Procurios

Pim Elshoff

developer.procurios.com
@pelshoff
https://speakerdeck.com/pelshoff/technically-ddd-v5
Slido.com with #K100
Please fill in the Feedback Forms