14.–17. 09. 2009
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Bohnenweisheit

Bean Validation (JSR 303)

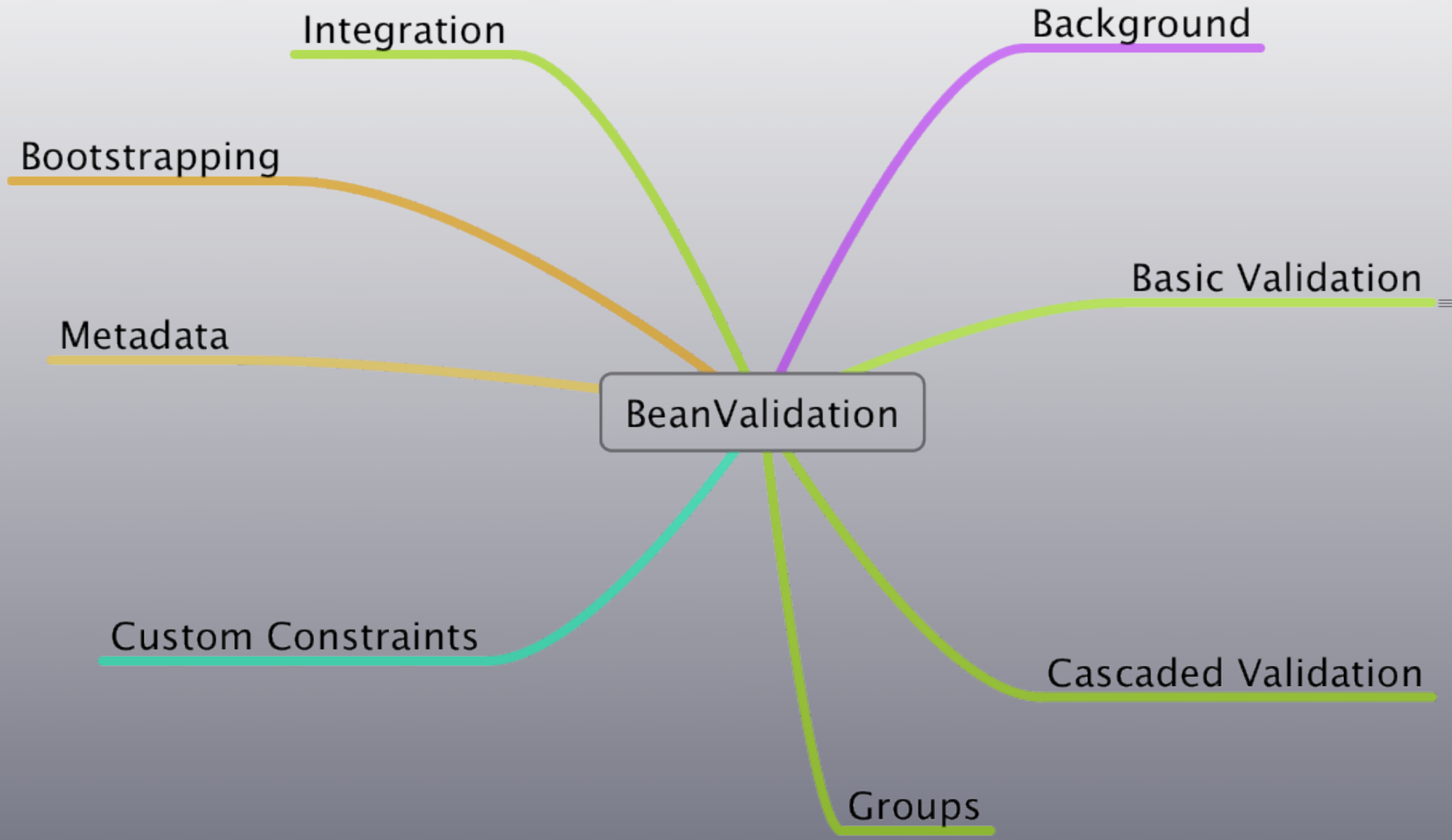
Hardy Ferentschik

JBoss by Red Hat

Bean Validation
JSR 303

redhat.

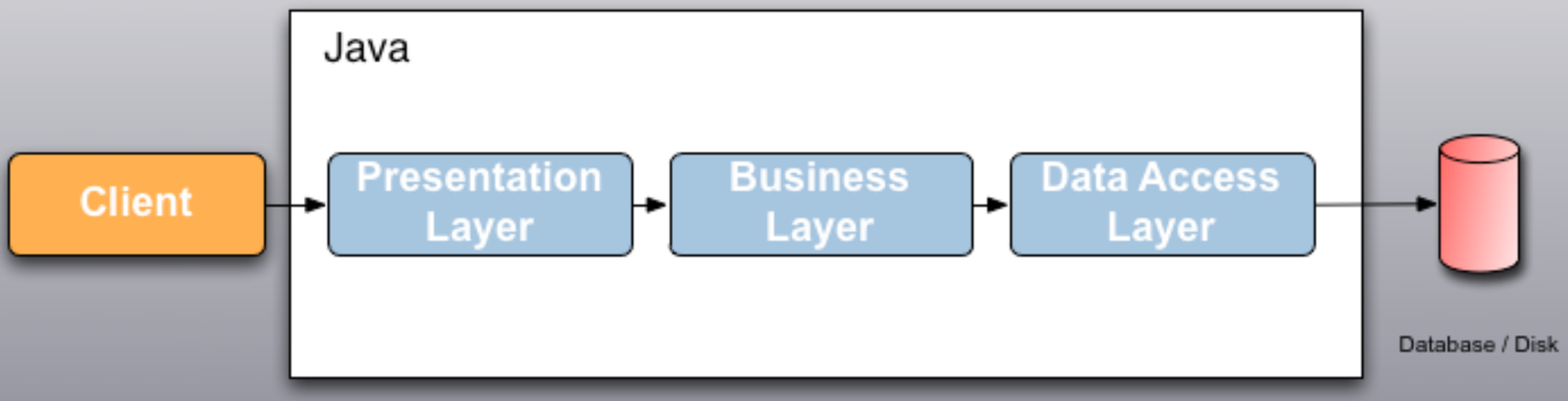
JBoss
a division of Red Hat


Integration
Background
Bootstrapping
Basic Validation
Metadata
BeanValidation
Custom Constraints
Cascaded Validation
Groups

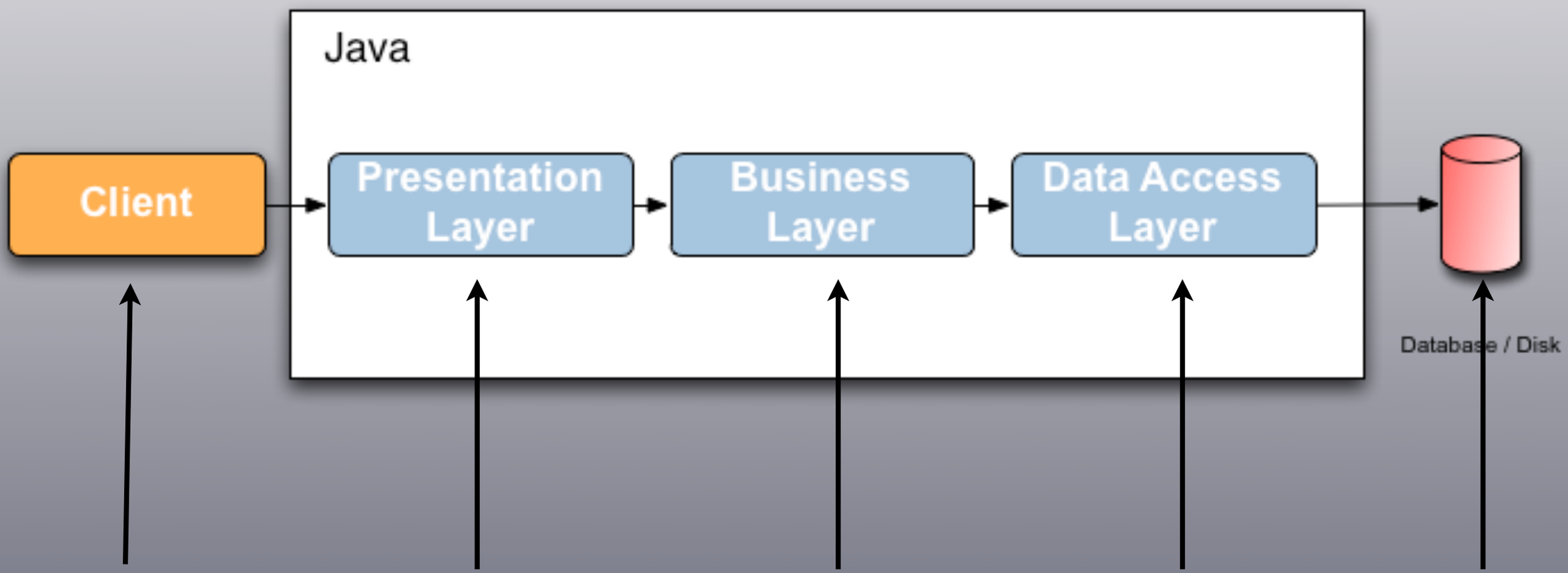
Why validate?

- Give feedback to the user
- Ensure correctness of service and data
- Avoid corrupt persistent data

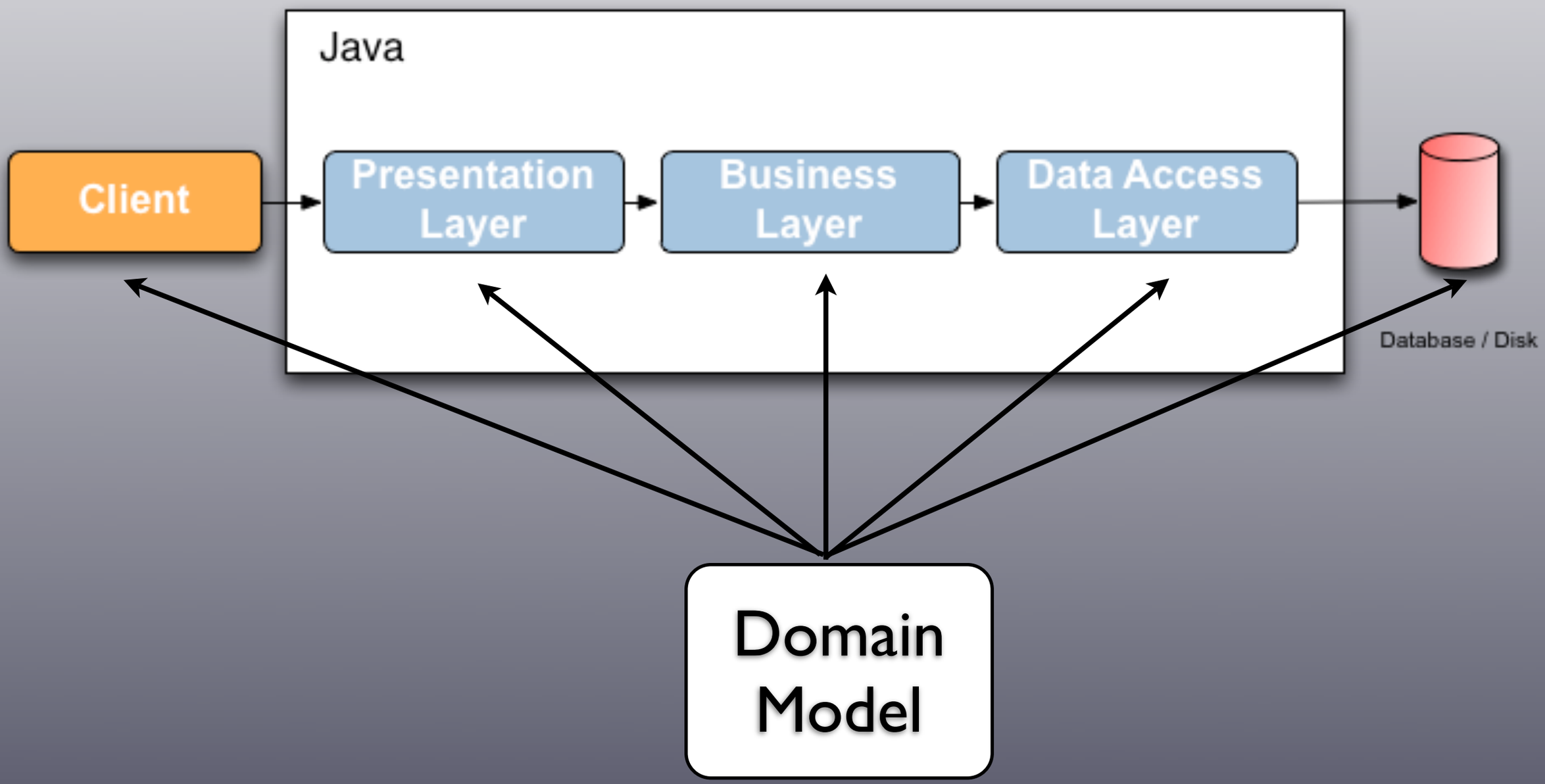
Where do we need validation?



Where do we need validation?



Where do we need validation?

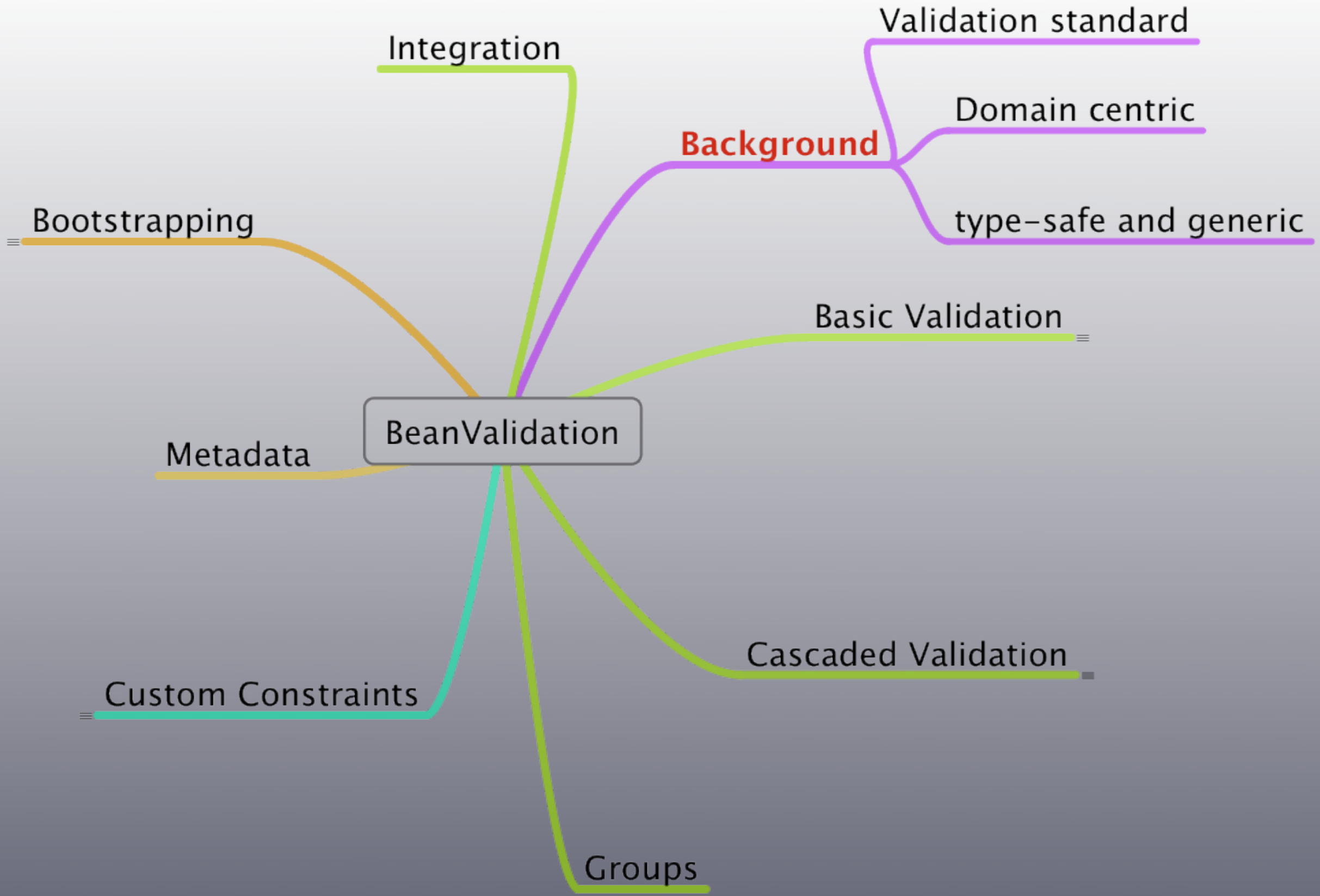


What we want

- A uniform way to express validation rules
- A standard way to check these rules
- Rules centered around domain model
- A bridge to the non Java world

JSR 303

- Introduces constraints on bean, field and property level
- Defines a type-safe and generic validation API
- Exposes powerful meta data API
- Current state - 1.0 CR3 Proposed Final Draft 2


BeanValidation
Integration
Validation standard
Domain centric
Background
type-safe and generic
Bootstrapping
Basic Validation
Metadata
Custom Constraints
Cascaded Validation
Groups

Basic Validation

Declare constraint

```
 5.  public class SimpleEngine {
 6.    @Pattern(regexp = "^[A-Z0-9-]+$",
 7.      message = "must contain alphabetical characters only")
 8.    private String serialNumber;
 9.
10.    public String getSerialNumber() {
11.      return serialNumber;
12.    }
13.
14.    public void setSerialNumber(String serialNumber) {
15.      this.serialNumber = serialNumber;
16.    }
17.  }
```

Declare constraint

```
 5.  public class SimpleEngine {
 6.    @Pattern(regexp = "^[A-Z0-9-]+$",
 7.      message = "must contain alphabetical characters only")
 8.    private String serialNumber;
 9.
10.    public String getSerialNumber() {
11.      return serialNumber;
12.    }
13.
14.    public void setSerialNumber(String serialNumber) {
15.      this.serialNumber = serialNumber;
16.    }
17.  }
```

Declare constraint

```
5.  public class Engine {
6.    @Pattern.List({
7.      @Pattern(regexp = "^[A-Z0-9-]+$",
8.        message = "must contain alphabetical characters only"),
9.      @Pattern(regexp = "^....-....-....$",
10.       message = "must match {regexp}")
11.   })
12.   private String serialNumber;
```

Declare constraint

```
5.  public class Engine {
6.     @Pattern.List({
7.        @Pattern(regexp = "^[A-Z0-9-]+$",
8.          message = "must contain alphabetical characters only"),
9.        @Pattern(regexp = "^....-....-....$",
10.         message = "must match {regexp}")
11.    })
12.    private String serialNumber;
```

Declare constraint

```
5.  public class EngineInherited extends SimpleEngine {
6.
7.     @Pattern(regexp = "^....-....-....$",
8.       message = "must match {regexp}")
9.     private String serialNumber;
```

Declare constraint

```
5.  public class EngineInherited extends SimpleEngine {
6.
7.     @Pattern(regexp = "^....-....-....$",
8.       message = "must match {regexp}")
9.     private String serialNumber;
```

Validate object

```
16.    @Test
17.    public void testBasicValidation() {
18.
19.      ValidatorFactory factory =
20.        Validation.buildDefaultValidatorFactory();
21.      Validator validator = factory.getValidator();
22.
23.      Engine engine = new Engine();
24.      engine.setSerialNumber("ABCDEFGH1234");
25.
26.      Set<ConstraintViolation<Engine>> constraintViolations =
27.        validator.validate(engine);
28.
29.      assertEquals(constraintViolations.size(), 1,
30.        "Wrong number of constraints");
31.
32.      ConstraintViolation<Engine> violation =
33.        constraintViolations.iterator().next();
34.
35.      assertEquals(violation.getMessage(),
36.        "must match ^....-....-....$", "Wrong message");
```

Validate object

```
16.    @Test
17.    public void testBasicValidation() {
18.
19.      ValidatorFactory factory =
20.        Validation.buildDefaultValidatorFactory();
21.      Validator validator = factory.getValidator();
22.
23.      Engine engine = new Engine();
24.      engine.setSerialNumber("ABCDEFGH1234");
25.
26.      Set<ConstraintViolation<Engine>> constraintViolations =
27.        validator.validate(engine);
28.
29.      assertEquals(constraintViolations.size(), 1,
30.        "Wrong number of constraints");
31.
32.      ConstraintViolation<Engine> violation =
33.        constraintViolations.iterator().next();
34.
35.      assertEquals(violation.getMessage(),
36.        "must match ^....-....-....$", "Wrong message");
```

Validate object

```
16.    @Test
17.    public void testBasicValidation() {
18.
19.      ValidatorFactory factory =
20.        Validation.buildDefaultValidatorFactory();
21.      Validator validator = factory.getValidator();
22.
23.      Engine engine = new Engine();
24.      engine.setSerialNumber("ABCDEFGH1234");
25.
26.      Set<ConstraintViolation<Engine>> constraintViolations =
27.        validator.validate(engine);
28.
29.      assertEquals(constraintViolations.size(), 1,
30.        "Wrong number of constraints");
31.
32.      ConstraintViolation<Engine> violation =
33.        constraintViolations.iterator().next();
34.
35.      assertEquals(violation.getMessage(),
36.        "must match ^....-....-....$", "Wrong message");
```

Validate object

```
16.    @Test
17.    public void testBasicValidation() {
18.
19.      ValidatorFactory factory =
20.        Validation.buildDefaultValidatorFactory();
21.      Validator validator = factory.getValidator();
22.
23.      Engine engine = new Engine();
24.      engine.setSerialNumber("ABCDEFGH1234");
25.
26.      Set<ConstraintViolation<Engine>> constraintViolations =
27.        validator.validate(engine);
28.
29.      assertEquals(constraintViolations.size(), 1,
30.        "Wrong number of constraints");
31.
32.      ConstraintViolation<Engine> violation =
33.        constraintViolations.iterator().next();
34.
35.      assertEquals(violation.getMessage(),
36.        "must match ^....-....-....$", "Wrong message");
```

Validate object

```
16.    @Test
17.    public void testBasicValidation() {
18.
19.      ValidatorFactory factory =
20.        Validation.buildDefaultValidatorFactory();
21.      Validator validator = factory.getValidator();
22.
23.      Engine engine = new Engine();
24.      engine.setSerialNumber("ABCDEFGH1234");
25.
26.      Set<ConstraintViolation<Engine>> constraintViolations =
27.        validator.validate(engine);
28.
29.      assertEquals(constraintViolations.size(), 1,
30.        "Wrong number of constraints");
31.
32.      ConstraintViolation<Engine> violation =
33.        constraintViolations.iterator().next();
34.
35.      assertEquals(violation.getMessage(),
36.        "must match ^....-....-....$", "Wrong message");
```

Validate object

```
16.    @Test
17.    public void testBasicValidation() {
18.
19.      ValidatorFactory factory =
20.        Validation.buildDefaultValidatorFactory();
21.      Validator validator = factory.getValidator();
22.
23.      Engine engine = new Engine();
24.      engine.setSerialNumber("ABCDEFGH1234");
25.
26.      Set<ConstraintViolation<Engine>> constraintViolations =
27.        validator.validate(engine);
28.
29.      assertEquals(constraintViolations.size(), 1,
30.        "Wrong number of constraints");
31.
32.      ConstraintViolation<Engine> violation =
33.        constraintViolations.iterator().next();
34.
35.      assertEquals(violation.getMessage(),
36.        "must match ^....-....-....$", "Wrong message");
```

DEMO

Message interpolation

- Per default messages externalized in ResourceBundle
- Message parameters in {}
- Pluggable MessageInterplator

Built-in constraints

Built-in constraints

@Null

@NotNull

@AssertTrue

@AssertFalse

@Min

@DecimalMin

@Max

@DecimalMax

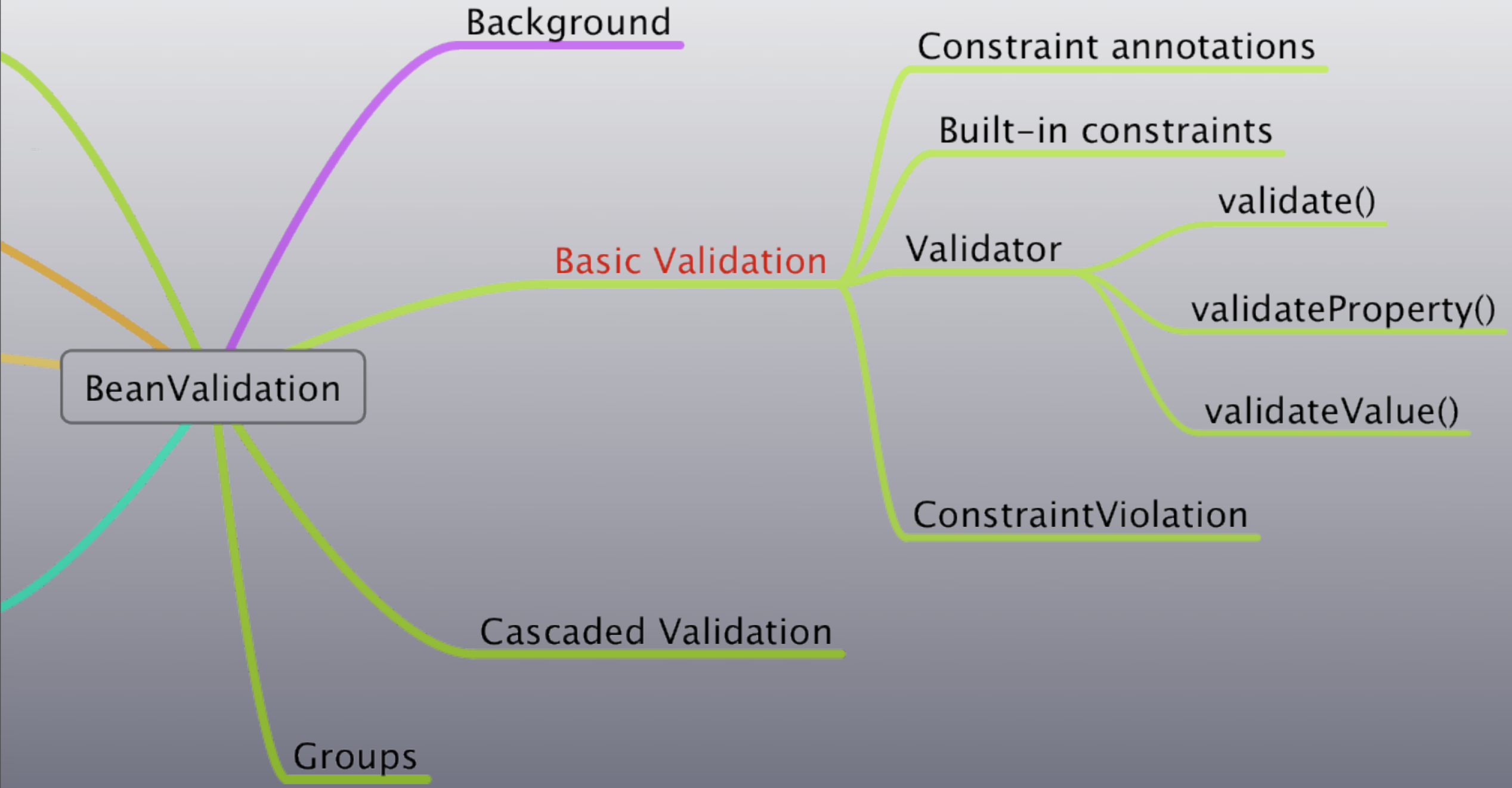
@Digits

@Past

@Future

@Pattern

@Size


BeanValidation
Background
Basic Validation
Constraint annotations
Built-in constraints
Validator
validate()
validateProperty()
validateValue()
ConstraintViolation
Cascaded Validation
Groups

Custom Constraints

Define annotation

```
11.  @Target({METHOD, FIELD, ANNOTATION_TYPE})
12.  @Retention(RUNTIME)
13.  @Constraint(validatedBy = PositiveConstraintValidator.class)
14.  public @interface Positive {
15.    public abstract String message()
16.      default "{constraints.positive}";
17.
18.    public abstract Class<?>[] groups() default {};
19.
20.    public abstract Class<? extends Payload>[] payload()
21.      default {};
22.  }
```

Define annotation

```
11.  @Target({METHOD, FIELD, ANNOTATION_TYPE})
12.  @Retention(RUNTIME)
13.  @Constraint(validatedBy = PositiveConstraintValidator.class)
14.  public @interface Positive {
15.    public abstract String message()
16.      default "{constraints.positive}";
17.
18.    public abstract Class<?>[] groups() default {};
19.
20.    public abstract Class<? extends Payload>[] payload()
21.      default {};
22.  }
```

Define annotation

```
11.  @Target({METHOD, FIELD, ANNOTATION_TYPE})
12.  @Retention(RUNTIME)
13.  @Constraint(validatedBy = PositiveConstraintValidator.class)
14.  public @interface Positive {
15.    public abstract String message()
16.      default "{constraints.positive}";
17.
18.    public abstract Class<?>[] groups() default {};
19.
20.    public abstract Class<? extends Payload>[] payload()
21.      default {};
22.  }
```

Define annotation

```
11.  @Target({METHOD, FIELD, ANNOTATION_TYPE})
12.  @Retention(RUNTIME)
13.  @Constraint(validatedBy = PositiveConstraintValidator.class)
14.  public @interface Positive {
15.    public abstract String message()
16.      default "{constraints.positive}";
17.
18.    public abstract Class<?>[] groups() default {};
19.
20.    public abstract Class<? extends Payload>[] payload()
21.      default {};
22.  }
```

Implement validator(s)

```
 7.  public class PositiveConstraintValidator
 8.    implements ConstraintValidator<Positive, Integer> {
 9.
10.    public void initialize(Positive annotation) {
11.    }
12.
13.    public boolean isValid(Integer value,
14.                           ConstraintValidatorContext context) {
15.      if (value == null) {
16.        return true;
17.      }
18.      return value >= 0;
19.    }
20.  }
```

Implement validator(s)

```
 7.  public class PositiveConstraintValidator
 8.      implements ConstraintValidator<Positive, Integer> {
 9.
10.    public void initialize(Positive annotation) {
11.    }
12.
13.    public boolean isValid(Integer value,
14.                           ConstraintValidatorContext context) {
15.      if (value == null) {
16.        return true;
17.      }
18.      return value >= 0;
19.    }
20.  }
```

Implement validator(s)

```
 7.  public class PositiveConstraintValidator
 8.    implements ConstraintValidator<Positive, Integer> {
 9.
10.    public void initialize(Positive annotation) {
11.    }
12.
13.    public boolean isValid(Integer value,
14.                           ConstraintValidatorContext context) {
15.      if (value == null) {
16.        return true;
17.      }
18.      return value >= 0;
19.    }
20.  }
```

Implement validator(s)

```
 7.  public class PositiveConstraintValidator
 8.    implements ConstraintValidator<Positive, Integer> {
 9.
10.    public void initialize(Positive annotation) {
11.    }
12.
13.    public boolean isValid(Integer value,
14.                           ConstraintValidatorContext context) {
15.      if (value == null) {
16.        return true;
17.      }
18.      return value >= 0;
19.    }
20.  }
```

Constraint Composition

```
14.  @Target({METHOD, FIELD})
15.  @Retention(RUNTIME)
16.  @NotNull
17.  @Size(min = 1)
18.  @Constraint(validatedBy = {})
19.  @ReportAsSingleViolation
20.  public @interface NotEmpty {
21.    String message() default "{constraints.NotEmpty.message}";
22.
23.    Class<?>[] groups() default {};
24.
25.    Class<? extends Payload>[] payload() default {};
26.  }
```

Constraint Composition

```
14.  @Target({METHOD, FIELD})
15.  @Retention(RUNTIME)
16.  @NotNull
17.  @Size(min = 1)
18.  @Constraint(validatedBy = {})
19.  @ReportAsSingleViolation
20.  public @interface NotEmpty {
21.    String message() default "{constraints.NotEmpty.message}";
22.
23.    Class<?>[] groups() default {};
24.
25.    Class<? extends Payload>[] payload() default {};
26.  }
```

Constraint Composition

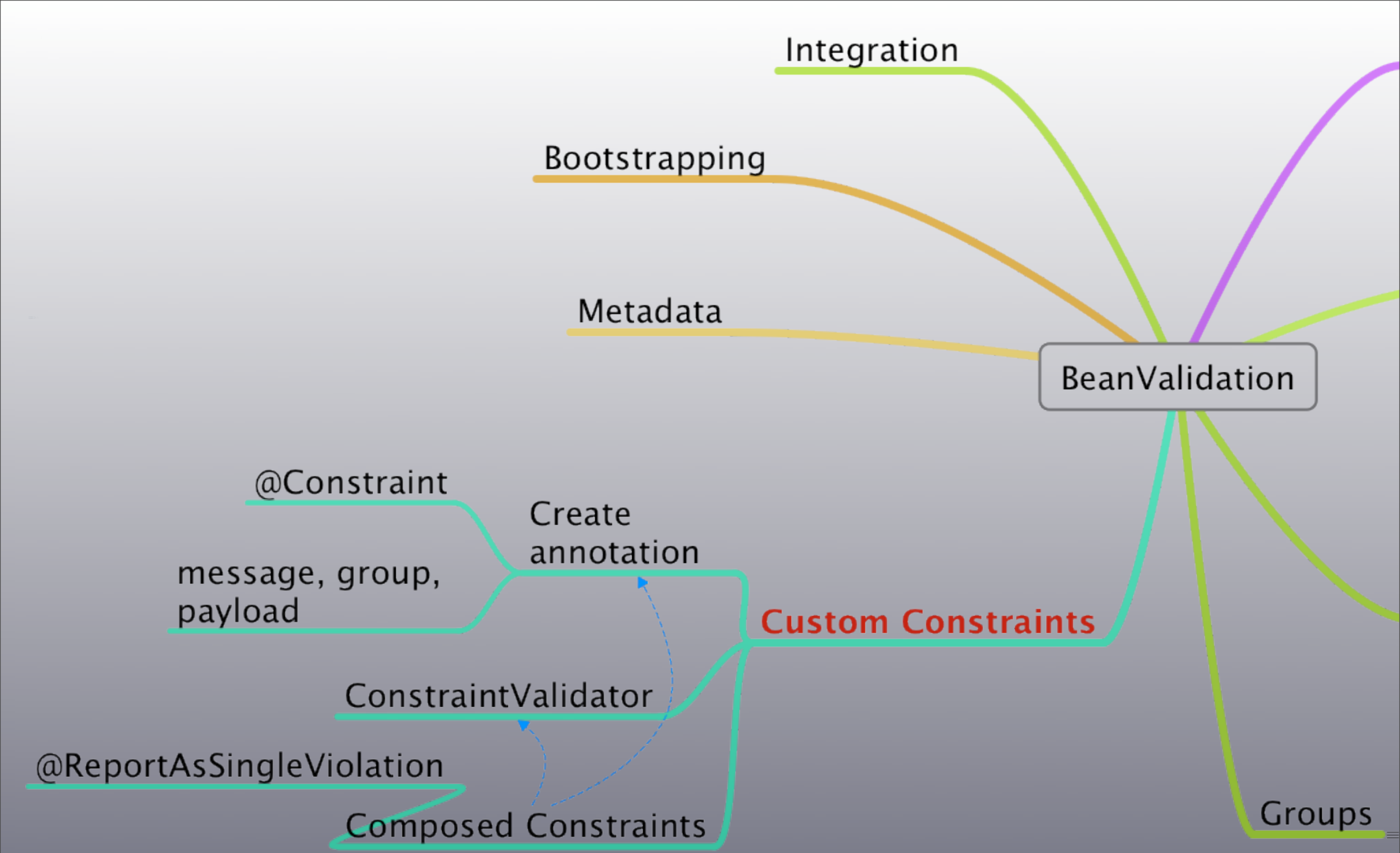
```
14.  @Target({METHOD, FIELD})
15.  @Retention(RUNTIME)
16.  @NotNull
17.  @Size(min = 1)
18.  @Constraint(validatedBy = {})
19.  @ReportAsSingleViolation
20.  public @interface NotEmpty {
21.    String message() default "{constraints.NotEmpty.message}";
22.
23.    Class<?>[] groups() default {};
24.
25.    Class<? extends Payload>[] payload() default {};
26.  }
```

Constraint Composition

```
14.  @Target({METHOD, FIELD})
15.  @Retention(RUNTIME)
16.  @NotNull
17.  @Size(min = 1)
18.  @Constraint(validatedBy = {})
19.  @ReportAsSingleViolation
20.  public @interface NotEmpty {
21.    String message() default "{constraints.NotEmpty.message}";
22.
23.    Class<?>[] groups() default {};
24.
25.    Class<? extends Payload>[] payload() default {};
26.  }
```

Constraint Composition

```
14.  @Target({METHOD, FIELD})
15.  @Retention(RUNTIME)
16.  @NotNull
17.  @Size(min = 1)
18.  @Constraint(validatedBy = {})
19.  @ReportAsSingleViolation
20.  public @interface NotEmpty {
21.    String message() default "{constraints.NotEmpty.message}";
22.
23.    Class<?>[] groups() default {};
24.
25.    Class<? extends Payload>[] payload() default {};
26.  }
```


Integration
Bootstrapping
Metadata
BeanValidation
@Constraint
Create annotation
message, group, payload
Custom Constraints
ConstraintValidator
@ReportAsSingleViolation
Composed Constraints
Groups

Cascaded Validation

@Valid

```
 9.  public class Customer {
10.    @NotNull
11.    private String name;
12.
13.    @Valid
14.    private List<Order> orders = new ArrayList<Order>();
15.
16.    public Customer(String name) {
17.      this.name = name;
18.    }
19.
20.    public void addOrder(Order order) {
21.      orders.add(order);
22.    }
23.
```

@Valid

```
 9.  public class Customer {
10.     @NotNull
11.     private String name;
12.
13.     @Valid
14.     private List<Order> orders = new ArrayList<Order>();
15.
16.     public Customer(String name) {
17.       this.name = name;
18.     }
19.
20.     public void addOrder(Order order) {
21.       orders.add(order);
22.     }
23.
```

@Valid

```
 5.  public class Order {
 6.          @NotNull(message = "Order number must be specified")
 7.          private Integer orderNumber;
 8.
 9.          public Integer getOrderNumber() {
10.                  return orderNumber;
11.          }
12.
13.          public void setOrderNumber(Integer orderNumber) {
14.                  this.orderNumber = orderNumber;
15.          }
16.  }
17.
```

@Valid

```
 5.  public class Order {
 6.          @NotNull(message = "Order number must be specified")
 7.          private Integer orderNumber;
 8.
 9.          public Integer getOrderNumber() {
10.                  return orderNumber;
11.          }
12.
13.          public void setOrderNumber(Integer orderNumber) {
14.                  this.orderNumber = orderNumber;
15.          }
16.  }
17.
```

@Valid

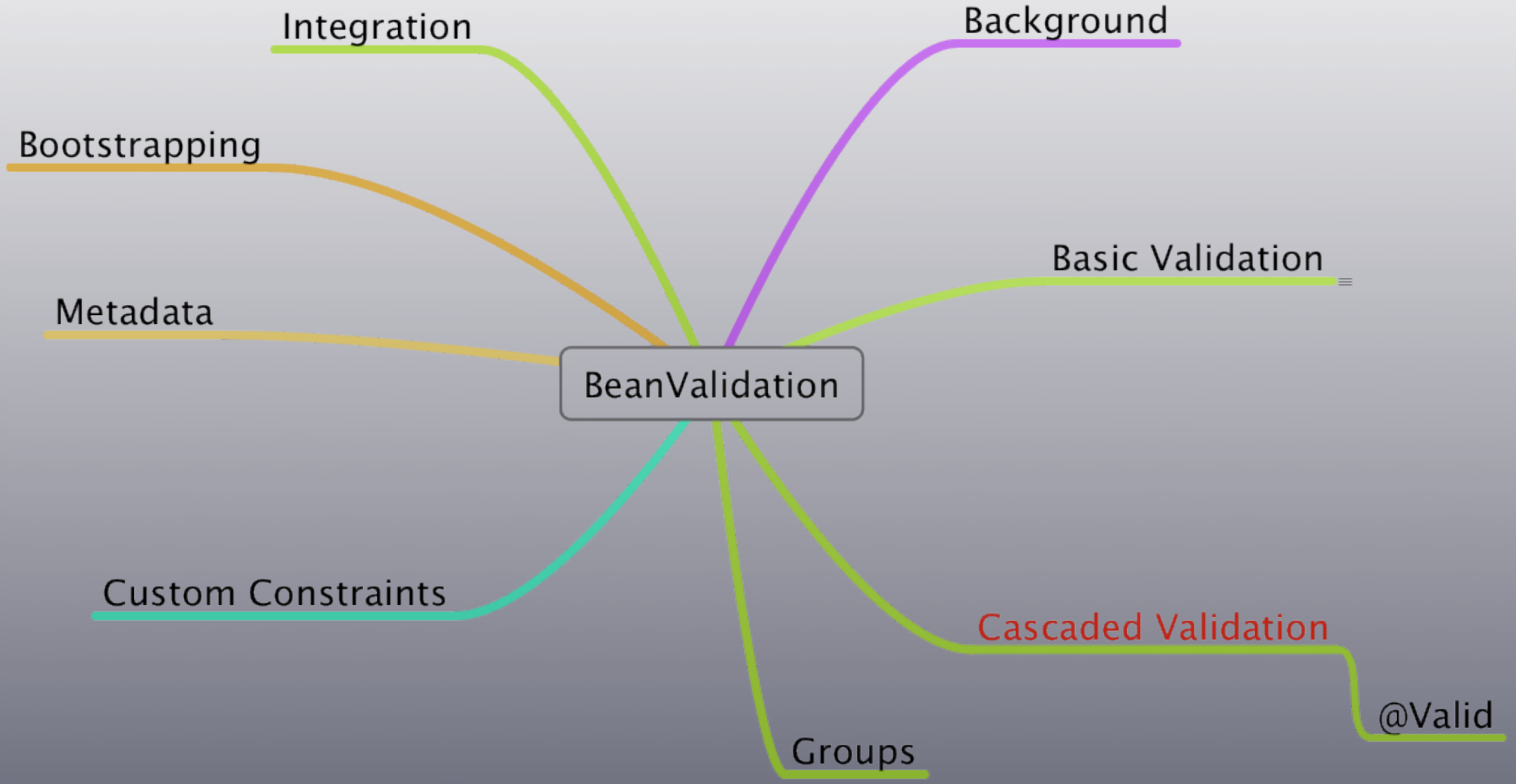
```
24.       Customer customer = new Customer("John Doe");
25.
26.       Set<ConstraintViolation<Customer>>
27.         constraintViolations =
28.         validator.validate(customer);
29.
30.       assertEquals(constraintViolations.size(), 0,
31.         "Wrong number of constraints");
32.
33.       Order order = new Order();
34.       customer.addOrder(order);
35.
36.       constraintViolations =
37.         validator.validate(customer);
38.
39.       assertEquals(constraintViolations.size(), 1,
40.         "Wrong number of constraints");
41.       assertEquals(
42.         constraintViolations.iterator().next().getMessage(),
43.         "Order number must be specified");
44.     }
```

@Valid

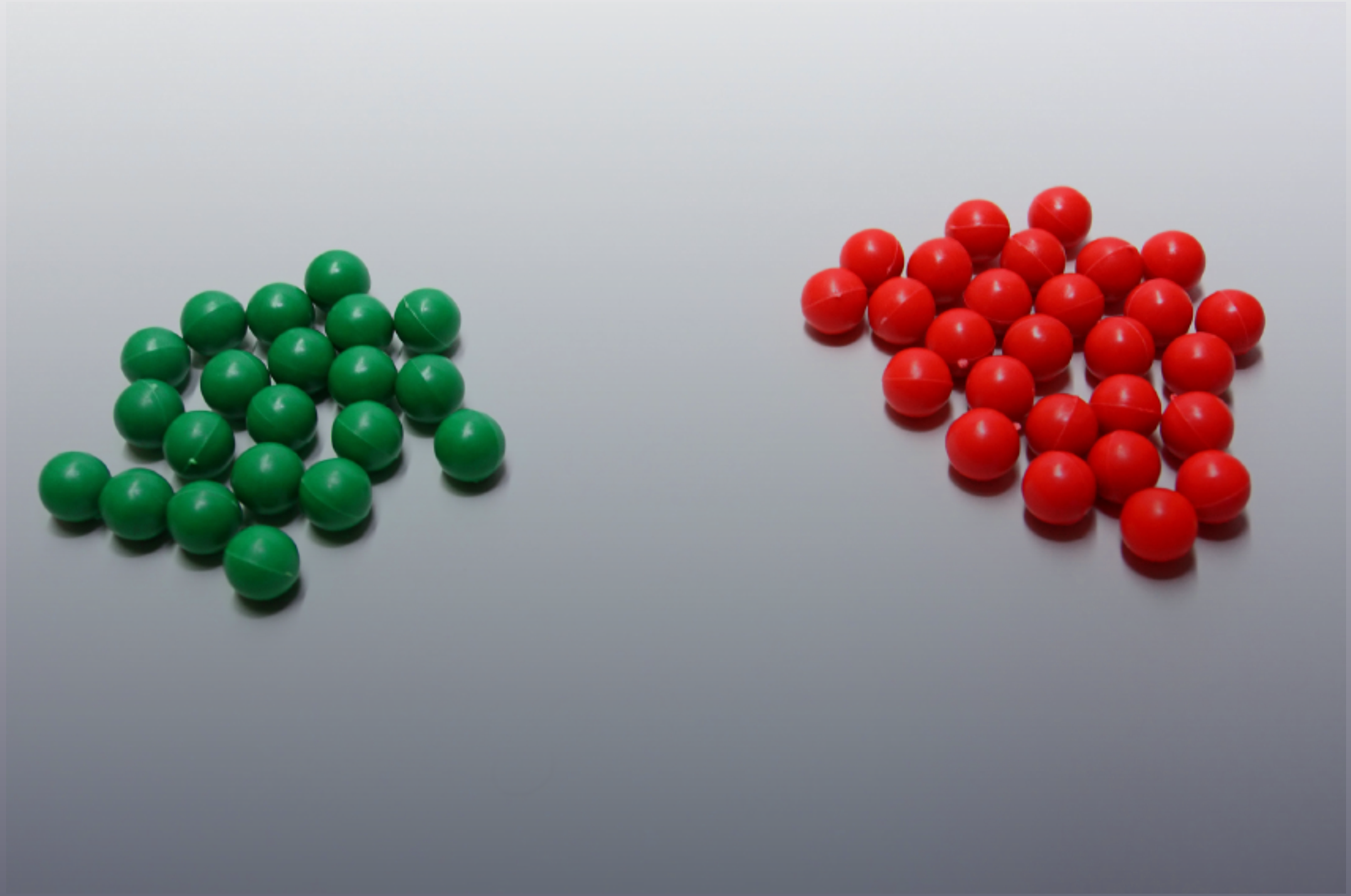
```
24.     Customer customer = new Customer("John Doe");
25.
26.     Set<ConstraintViolation<Customer>>
27.       constraintViolations =
28.       validator.validate(customer);
29.
30.     assertEquals(constraintViolations.size(), 0,
31.       "Wrong number of constraints");
32.
33.     Order order = new Order();
34.     customer.addOrder(order);
35.
36.     constraintViolations =
37.       validator.validate(customer);
38.
39.     assertEquals(constraintViolations.size(), 1,
40.       "Wrong number of constraints");
41.     assertEquals(
42.       constraintViolations.iterator().next().getMessage(),
43.       "Order number must be specified");
44.   }
```

@Valid

```
24.     Customer customer = new Customer("John Doe");
25.
26.     Set<ConstraintViolation<Customer>>
27.       constraintViolations =
28.       validator.validate(customer);
29.
30.     assertEquals(constraintViolations.size(), 0,
31.       "Wrong number of constraints");
32.
33.     Order order = new Order();
34.     customer.addOrder(order);
35.
36.     constraintViolations =
37.       validator.validate(customer);
38.
39.     assertEquals(constraintViolations.size(), 1,
40.       "Wrong number of constraints");
41.     assertEquals(
42.       constraintViolations.iterator().next().getMessage(),
43.       "Order number must be specified");
44.   }
```


Integration
Background
Bootstrapping
Basic Validation
Metadata
BeanValidation
Custom Constraints
Cascaded Validation
@Valid
Groups

Validation Groups



Groups

- Allow to execute constraints in logical groups (eg wizards)
- Groups are interfaces!
- Group execution order is undefined
- To enforce order use @GroupSequence

Specifying Groups

```
 8.  public class User {
 9.     @NotNull
10.     private String firstname;
11.
12.     @NotNull(groups = Default.class)
13.     private String lastname;
14.
15.     @Pattern(regexp = "[0-9 -]?", groups = Optional.class)
16.     private String phoneNumber;
17.
18.     @NotNull(groups = Billable.class)
19.     private CreditCard defaultCreditCard;
20.
21.     public interface Billable {
22.     }
23.
24.     public interface Optional {
25.     }
26.
27.     public interface BuyInOneClick extends Default, Billable {
28.     }
```

Specifying Groups

```
 8.  public class User {
 9.     @NotNull
10.     private String firstname;
11.
12.     @NotNull(groups = Default.class)
13.     private String lastname;
14.
15.     @Pattern(regexp = "[0-9 -]?", groups = Optional.class)
16.     private String phoneNumber;
17.
18.     @NotNull(groups = Billable.class)
19.     private CreditCard defaultCreditCard;
20.
21.     public interface Billable {
22.     }
23.
24.     public interface Optional {
25.     }
26.
27.     public interface BuyInOneClick extends Default, Billable {
28.     }
```

Specifying Groups

```
 8.  public class User {
 9.     @NotNull
10.     private String firstname;
11.
12.     @NotNull(groups = Default.class)
13.     private String lastname;
14.
15.     @Pattern(regexp = "[0-9 -]?", groups = Optional.class)
16.     private String phoneNumber;
17.
18.     @NotNull(groups = Billable.class)
19.     private CreditCard defaultCreditCard;
20.
21.     public interface Billable {
22.     }
23.
24.     public interface Optional {
25.     }
26.
27.     public interface BuyInOneClick extends Default, Billable {
28.     }
```

Specifying Groups

```
 8.  public class User {
 9.     @NotNull
10.     private String firstname;
11.
12.     @NotNull(groups = Default.class)
13.     private String lastname;
14.
15.     @Pattern(regexp = "[0-9 -]?", groups = Optional.class)
16.     private String phoneNumber;
17.
18.     @NotNull(groups = Billable.class)
19.     private CreditCard defaultCreditCard;
20.
21.     public interface Billable {
22.     }
23.
24.     public interface Optional {
25.     }
26.
27.     public interface BuyInOneClick extends Default, Billable {
28.     }
```

@GroupSequence

```
26.  @GroupSequence(value = {Default.class, User.Billable.class})
27.  public interface BuyInOneClick {
28.  }
```

Group test

```
24.     Set<ConstraintViolation<User>> constraintViolations =
25.       validator.validate(new User());
26.     assertEquals(constraintViolations.size(), 2);
27.
28.     constraintViolations =
29.       validator.validate(new User(), Default.class);
30.     assertEquals(constraintViolations.size(), 2);
31.
32.     constraintViolations =
33.       validator.validate(new User(), User.Billable.class);
34.     assertEquals(constraintViolations.size(), 1);
35.
36.     constraintViolations = validator
37.       .validate(new User(), Default.class,
38.         User.Billable.class);
39.     assertEquals(constraintViolations.size(), 3);
40.
41.     constraintViolations =
42.       validator.validate(new User(), User.BuyInOneClick.class);
43.     assertEquals(constraintViolations.size(), 3);
```

Group test

```
24.     Set<ConstraintViolation<User>> constraintViolations =
25.       validator.validate(new User());
26.     assertEquals(constraintViolations.size(), 2);
27.
28.     constraintViolations =
29.       validator.validate(new User(), Default.class);
30.     assertEquals(constraintViolations.size(), 2);
31.
32.     constraintViolations =
33.       validator.validate(new User(), User.Billable.class);
34.     assertEquals(constraintViolations.size(), 1);
35.
36.     constraintViolations = validator
37.       .validate(new User(), Default.class,
38.         User.Billable.class);
39.     assertEquals(constraintViolations.size(), 3);
40.
41.     constraintViolations =
42.       validator.validate(new User(), User.BuyInOneClick.class);
43.     assertEquals(constraintViolations.size(), 3);
```

Group test

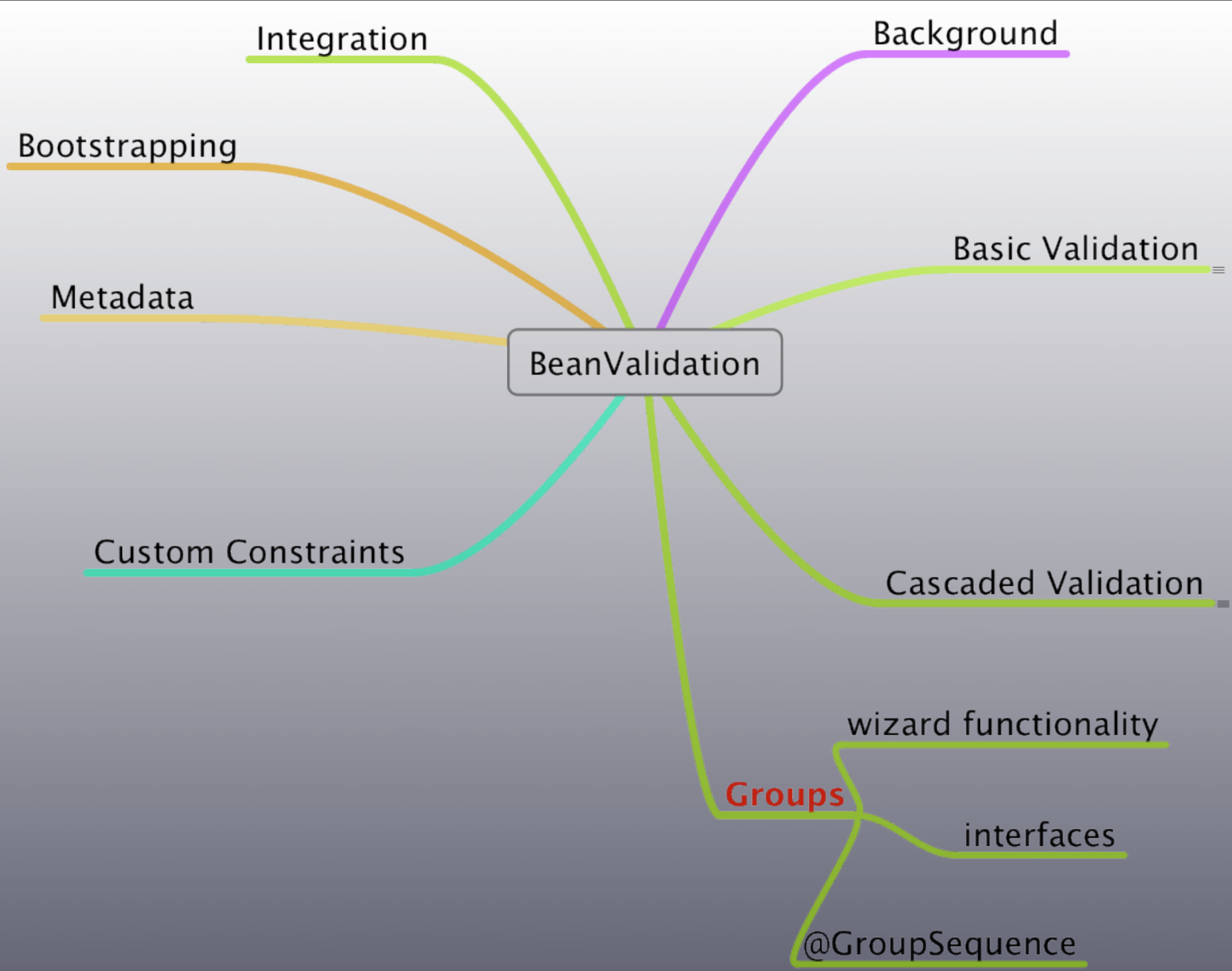
```
24.     Set<ConstraintViolation<User>> constraintViolations =
25.       validator.validate(new User());
26.     assertEquals(constraintViolations.size(), 2);
27.
28.     constraintViolations =
29.       validator.validate(new User(), Default.class);
30.     assertEquals(constraintViolations.size(), 2);
31.
32.     constraintViolations =
33.       validator.validate(new User(), User.Billable.class);
34.     assertEquals(constraintViolations.size(), 1);
35.
36.     constraintViolations = validator
37.       .validate(new User(), Default.class,
38.         User.Billable.class);
39.     assertEquals(constraintViolations.size(), 3);
40.
41.     constraintViolations =
42.       validator.validate(new User(), User.BuyInOneClick.class);
43.     assertEquals(constraintViolations.size(), 3);
```

Group test

```
24.     Set<ConstraintViolation<User>> constraintViolations =
25.       validator.validate(new User());
26.     assertEquals(constraintViolations.size(), 2);
27.
28.     constraintViolations =
29.       validator.validate(new User(), Default.class);
30.     assertEquals(constraintViolations.size(), 2);
31.
32.     constraintViolations =
33.       validator.validate(new User(), User.Billable.class);
34.     assertEquals(constraintViolations.size(), 1);
35.
36.     constraintViolations = validator
37.       .validate(new User(), Default.class,
38.         User.Billable.class);
39.     assertEquals(constraintViolations.size(), 3);
40.
41.     constraintViolations =
42.       validator.validate(new User(), User.BuyInOneClick.class);
43.     assertEquals(constraintViolations.size(), 3);
```

Group test

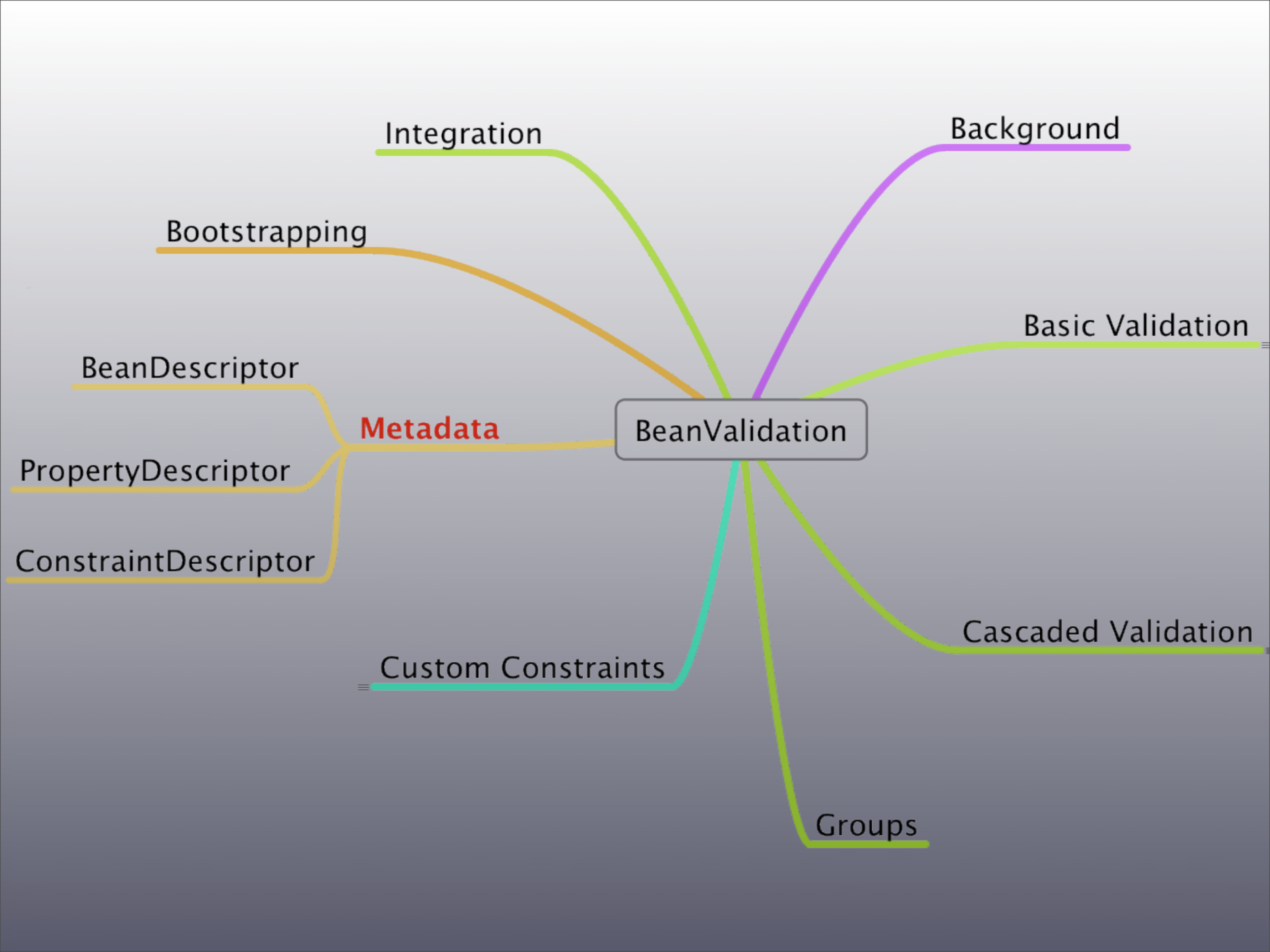
```
24.     Set<ConstraintViolation<User>> constraintViolations =
25.       validator.validate(new User());
26.     assertEquals(constraintViolations.size(), 2);
27.
28.     constraintViolations =
29.       validator.validate(new User(), Default.class);
30.     assertEquals(constraintViolations.size(), 2);
31.
32.     constraintViolations =
33.       validator.validate(new User(), User.Billable.class);
34.     assertEquals(constraintViolations.size(), 1);
35.
36.     constraintViolations = validator
37.       .validate(new User(), Default.class,
38.         User.Billable.class);
39.     assertEquals(constraintViolations.size(), 3);
40.
41.     constraintViolations =
42.       validator.validate(new User(), User.BuyInOneClick.class);
43.     assertEquals(constraintViolations.size(), 3);
```

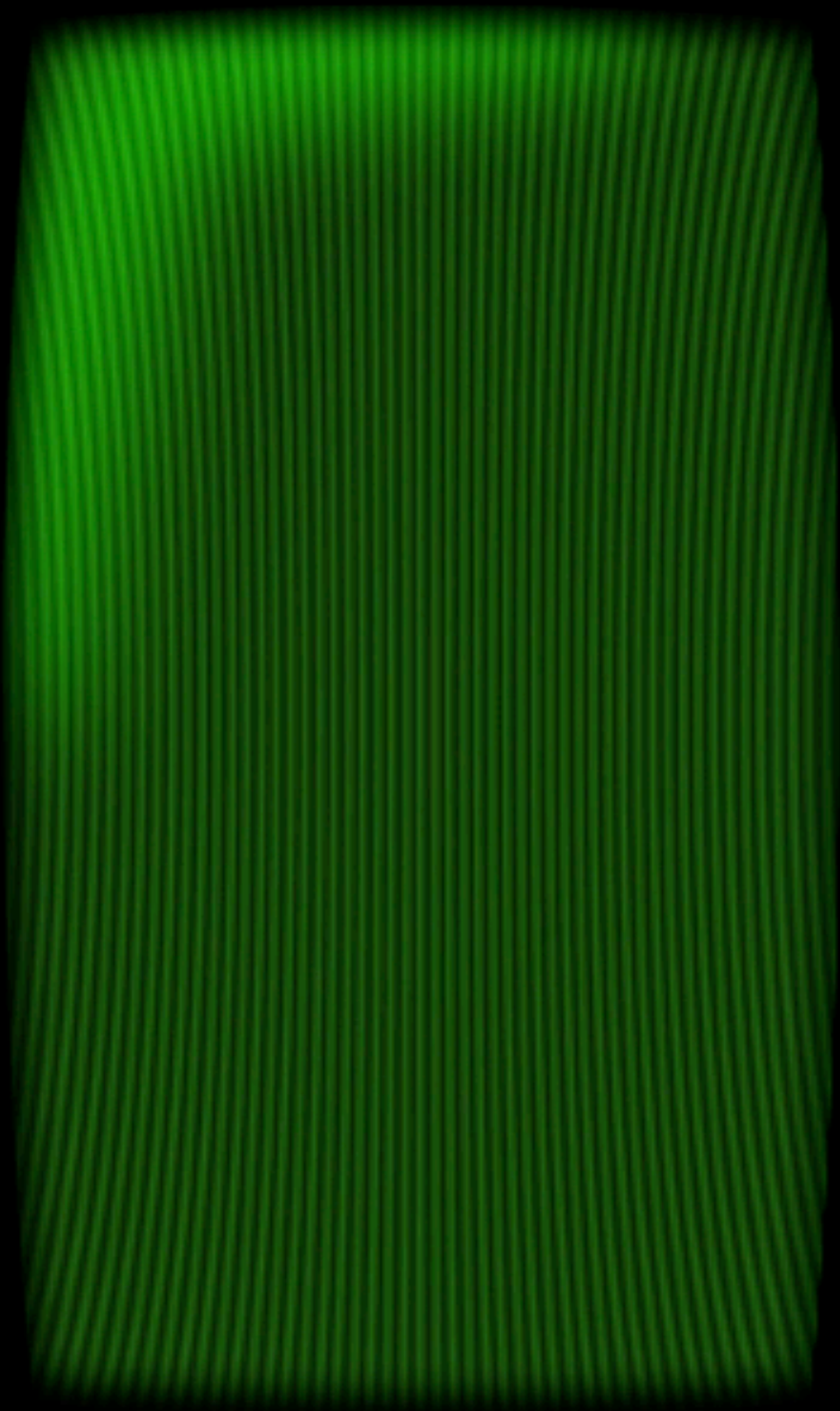

Integration
Background
Bootstrapping
Basic Validation
Metadata
BeanValidation
Custom Constraints
Cascaded Validation
wizard functionality
Groups
interfaces
@GroupSequence

Metadata API

Metadata classes

- BeanDescriptor
  - validator.getConstraintsForClass(User.class)
- PropertyDescriptor
  - beanDescr.getConstrainedProperties()
- ConstraintDescriptor
  - constraintViolation.getConstraintDescriptor()
  - propertyDesc.getConstraintDescriptors()


Integration
Background
Bootstrapping
Basic Validation
BeanDescriptor
Metadata
BeanValidation
PropertyDescriptor
ConstraintDescriptor
Custom Constraints
Cascaded Validation
Groups



Requirements

- Every provider must be able to bootstrap any implementation available in the classpath
- Default provider via /META-INF/services/javax.validation.spi.ValidationProvider

Bootstrap methods

```
16.       ValidatorFactory vf1 =
17.         Validation.buildDefaultValidatorFactory();
18.
19.
20.       ValidatorFactory vf2 = Validation.byDefaultProvider(
21.         .configure()
22.         .messageInterpolator(new MyMessageInterpolator())
23.         .traversableResolver(new MyTraversableResolver())
24.         .constraintValidatorFactory(
25.           new MyConstraintValidatorFactory())
26.         .buildValidatorFactory();
27.
28.
29.       ValidatorFactory vf3 = Validation
30.         .byProvider(HibernateValidator.class).configure()
31.         .messageInterpolator(new MyMessageInterpolator())
32.         .buildValidatorFactory();
```

Bootstrap methods

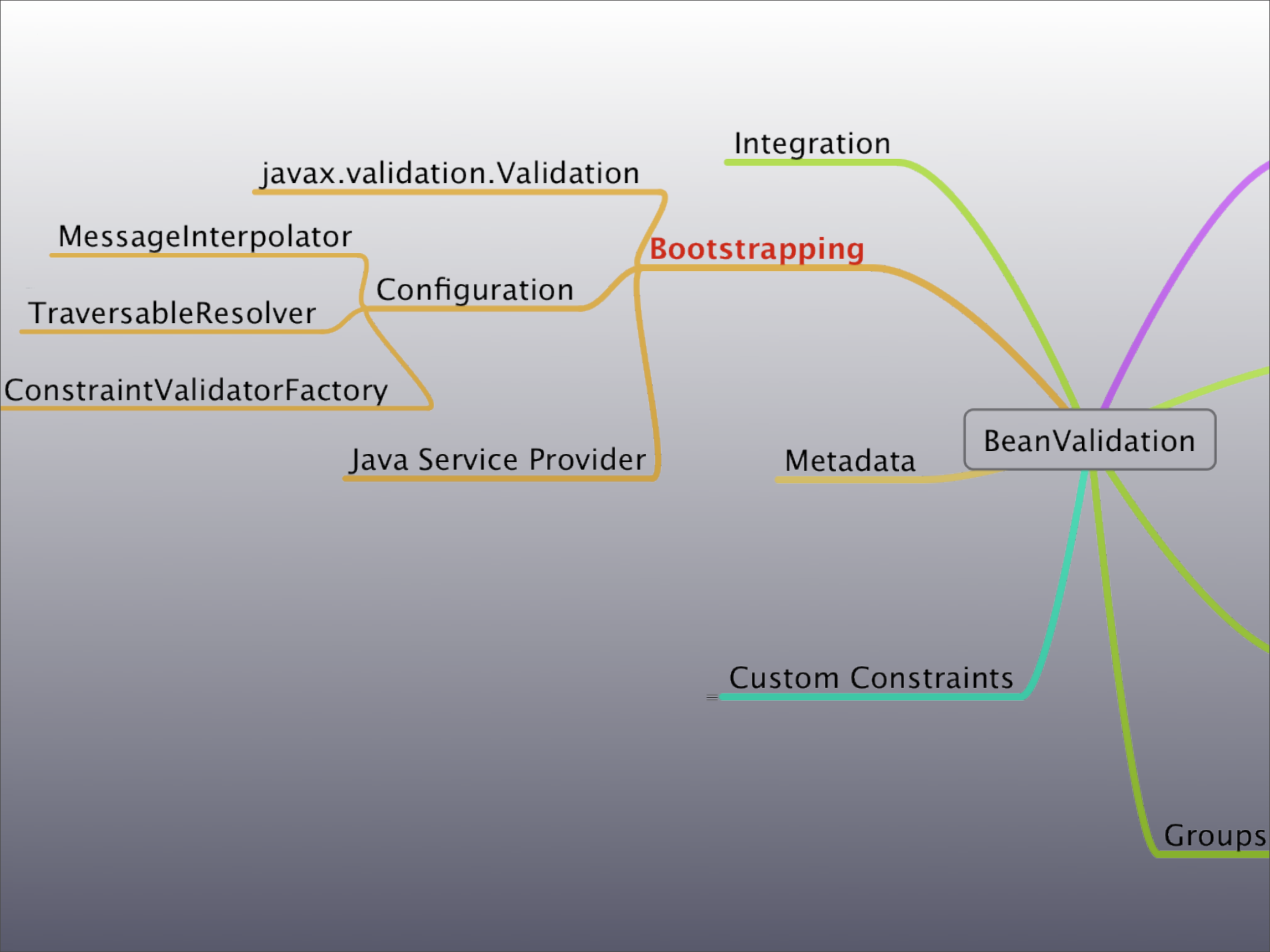
```
16.       ValidatorFactory vf1 =
17.         Validation.buildDefaultValidatorFactory();
18.
19.
20.       ValidatorFactory vf2 = Validation.byDefaultProvider(
21.         .configure()
22.         .messageInterpolator(new MyMessageInterpolator())
23.         .traversableResolver(new MyTraversableResolver())
24.         .constraintValidatorFactory(
25.           new MyConstraintValidatorFactory())
26.         .buildValidatorFactory();
27.
28.
29.       ValidatorFactory vf3 = Validation
30.         .byProvider(HibernateValidator.class).configure()
31.         .messageInterpolator(new MyMessageInterpolator())
32.         .buildValidatorFactory();
```

Bootstrap methods

```
16.       ValidatorFactory vf1 =
17.         Validation.buildDefaultValidatorFactory();
18.
19.
20.       ValidatorFactory vf2 = Validation.byDefaultProvider(
21.         .configure()
22.         .messageInterpolator(new MyMessageInterpolator())
23.         .traversableResolver(new MyTraversableResolver())
24.         .constraintValidatorFactory(
25.           new MyConstraintValidatorFactory())
26.         .buildValidatorFactory();
27.
28.
29.       ValidatorFactory vf3 = Validation
30.         .byProvider(HibernateValidator.class).configure()
31.         .messageInterpolator(new MyMessageInterpolator())
32.         .buildValidatorFactory();
```

Bootstrap methods

```
16.       ValidatorFactory vf1 =
17.         Validation.buildDefaultValidatorFactory();
18.
19.
20.       ValidatorFactory vf2 = Validation.byDefaultProvider(
21.         .configure()
22.         .messageInterpolator(new MyMessageInterpolator())
23.         .traversableResolver(new MyTraversableResolver())
24.         .constraintValidatorFactory(
25.           new MyConstraintValidatorFactory())
26.         .buildValidatorFactory();
27.
28.
29.       ValidatorFactory vf3 = Validation
30.         .byProvider(HibernateValidator.class).configure()
31.         .messageInterpolator(new MyMessageInterpolator())
32.         .buildValidatorFactory();
```


BeanValidation
Integration
Bootstrapping
javax.validation.Validation
Configuration
MessageInterpolator
TraversableResolver
ConstraintValidatorFactory
Java Service Provider
Metadata
Custom Constraints
Groups

Integration

JSF 2

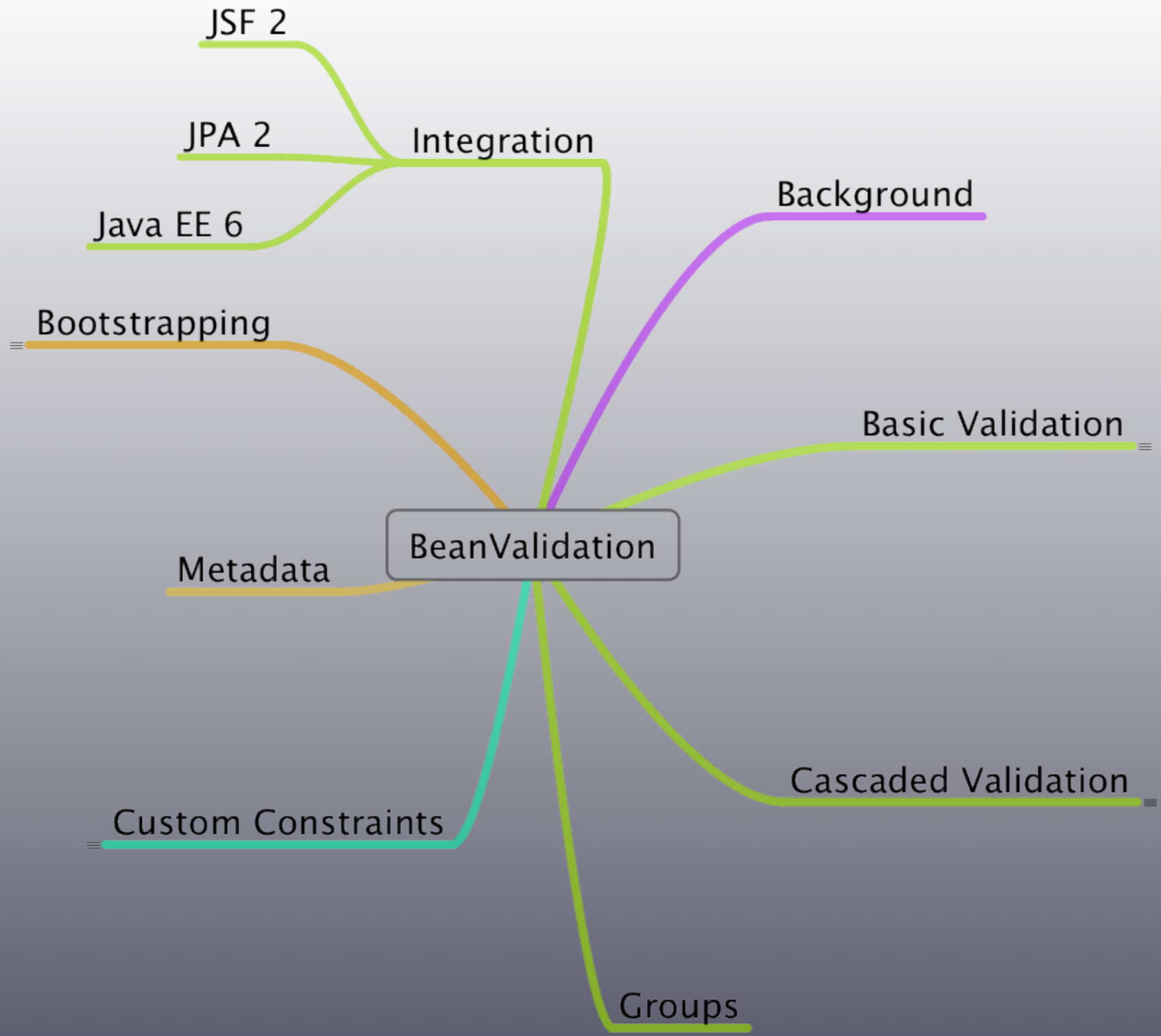
- Zero conf
- Validation of input components
  - Find property via EL
  - Call Validator.validateValue on input value
  - Return localized error message
  - Use JSF Locale

JPA 2

- Validation on entity change
- Make use of TraverableResolver

Java EE 6

- Validator as injectable resource
- Integrated in JBoss 5.2


JSF 2
JPA 2
Java EE 6
Integration
Background
Bootstrapping
Basic Validation
BeanValidation
Metadata
Cascaded Validation
Custom Constraints
Groups

More Info

- http://validator.hibernate.org
- http://in.relation.to
- http://forum.hibernate.org/viewforum.php?f=9
- hardy.ferentschik@redhat.com