RESTfully Async with Grails 2.3

By Graeme Rocher

springone 2GX

RESTfully Async with Grails 2.3

By Graeme Rocher

springone 2GX

SPRING IO EXECUTION:

Grails

Agenda

- Detailed Look at REST
- Fun with Grails Async APIs
- Summary / Q & A

Goals of REST Support in Grails 2.3

- Be Grails-y (i.e. dead simple)
- Be Flexible (start simple, get deep)
- Be pluggable (extend and conquer)
- Be current (support latest techniques)

REST Features in 2.3

- RESTful URL Mappings
- Domain Classes as REST resources
- Controllers as REST resources
- Renderers and Binders

GRAILS

Demo

Domain classes as REST resources

REST: @Resource Transformation

- Automatically exposes a domain class as a REST resource
- Configurable formats
- Configuration HTTP methods

REST: @Resource Transformation

- Automatically exposes a domain class as a REST resource
- Configurable formats
- Configuration HTTP methods

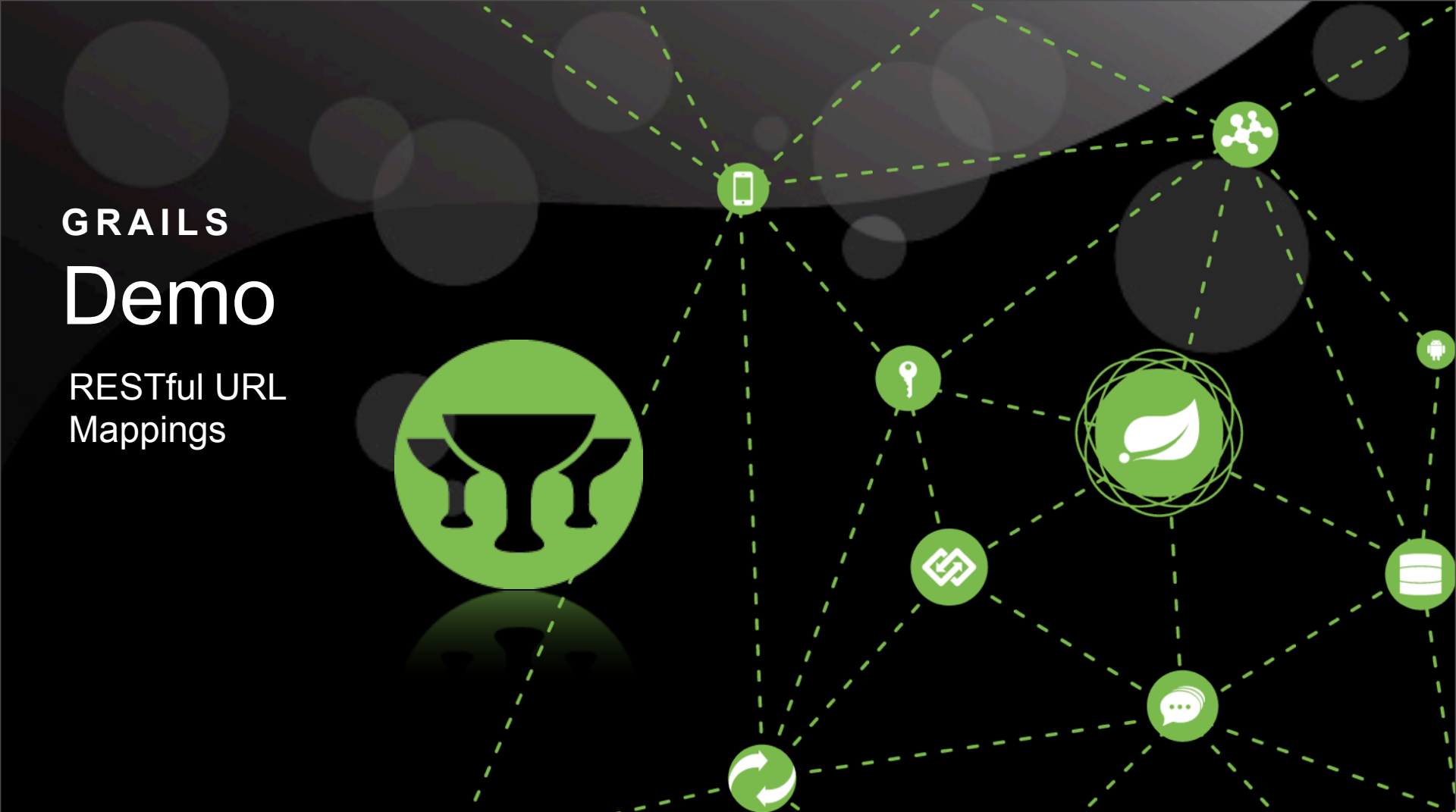
```
import grails.rest.*

@Resource
class Person {
    String name
}
```

GRAILS

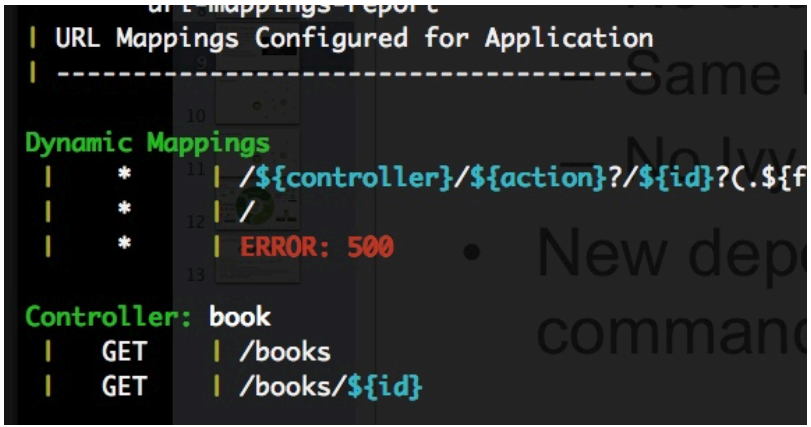
Demo

RESTful URL Mappings



REST: URL Mappings

- RESTful Mappings
  - Single or Multiple resources
  - Versioning
  - Nested resources
- New url-mappings-report command to mappings



REST: Multiple Resources

HTTP Method	URI	Action
GET	/books	index
POST	/books	save
GET	/books/${id}	show
PUT	/books/${id}	update
DELETE	/books/${id}	delete

REST: Multiple Resources

```
"/books"(resources:'book')
```

HTTP Method	URI	Action
GET	/books	index
POST	/books	save
GET	/books/${id}	show
PUT	/books/${id}	update
DELETE	/books/${id}	delete

REST: Singular Resources

HTTP Method	URI	Action
GET	/book	show
POST	/book	save
PUT	/book	update
DELETE	/book	delete

REST: Singular Resources

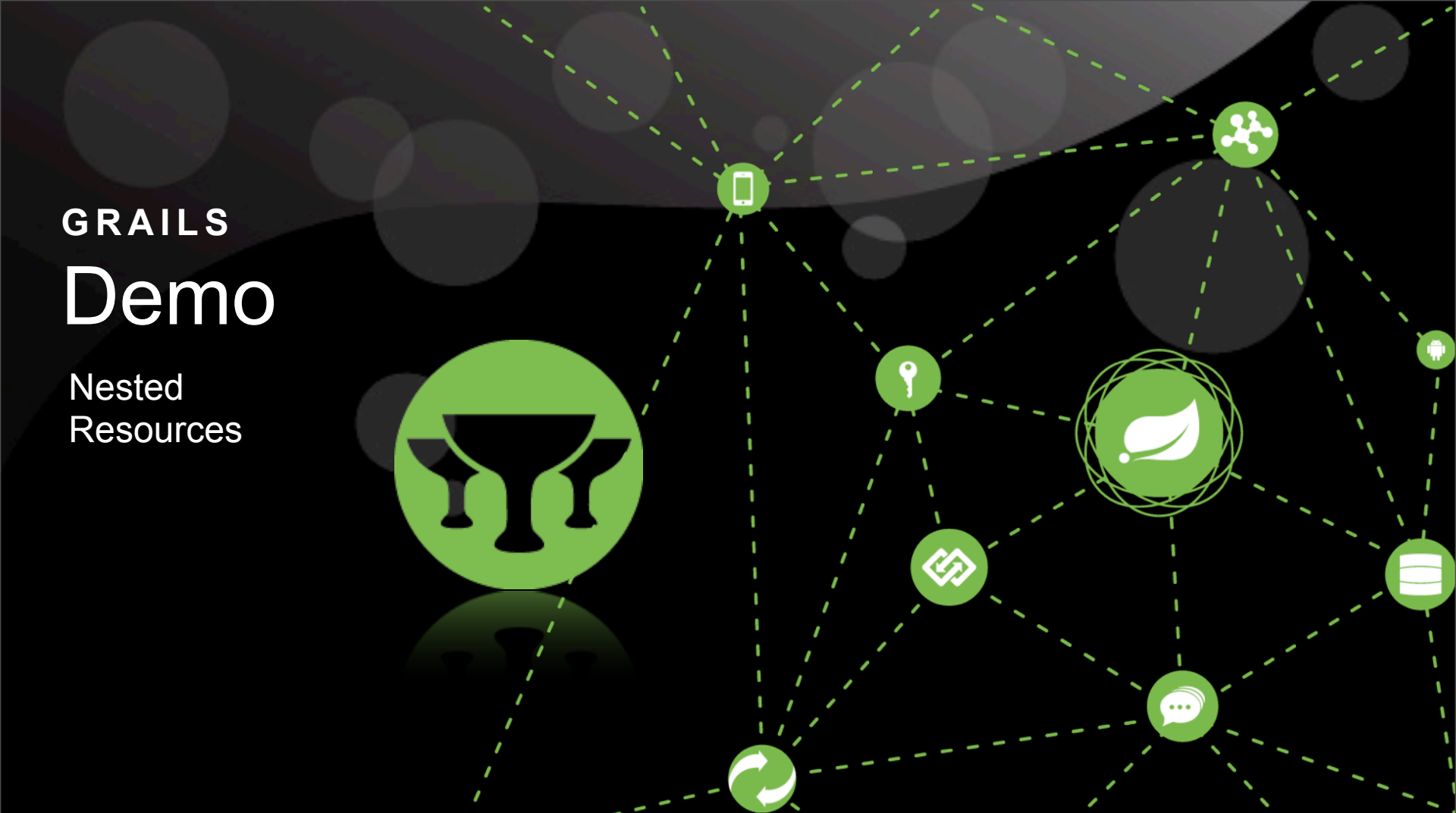
```
"/book"(resource:'book')
```

HTTP Method	URI	Action
GET	/book	show
POST	/book	save
PUT	/book	update
DELETE	/book	delete

GRAILS

Demo

Nested Resources



REST: Nested URL Mappings

- You can nest resources or other URL mappings within resources

REST: Nested URL Mappings

- You can nest resources or other URL mappings within resources

```
"/books"(resources:'book') {
  "/authors"(resources:"author")
}

"/books"(resources: "book") {
    "/publisher"(controller:"publisher")
}
```

REST: Nested Resources

HTTP Method	URI	Action
GET	/books/${bookId}/authors	index
POST	/books/${bookId}/authors	save
GET	/books/${bookId}/authors/${id}	show
PUT	/books/${bookId}/authors/${id}	update
DELETE	/books/${bookId}/authors/${id}	delete

REST: Nested Resources

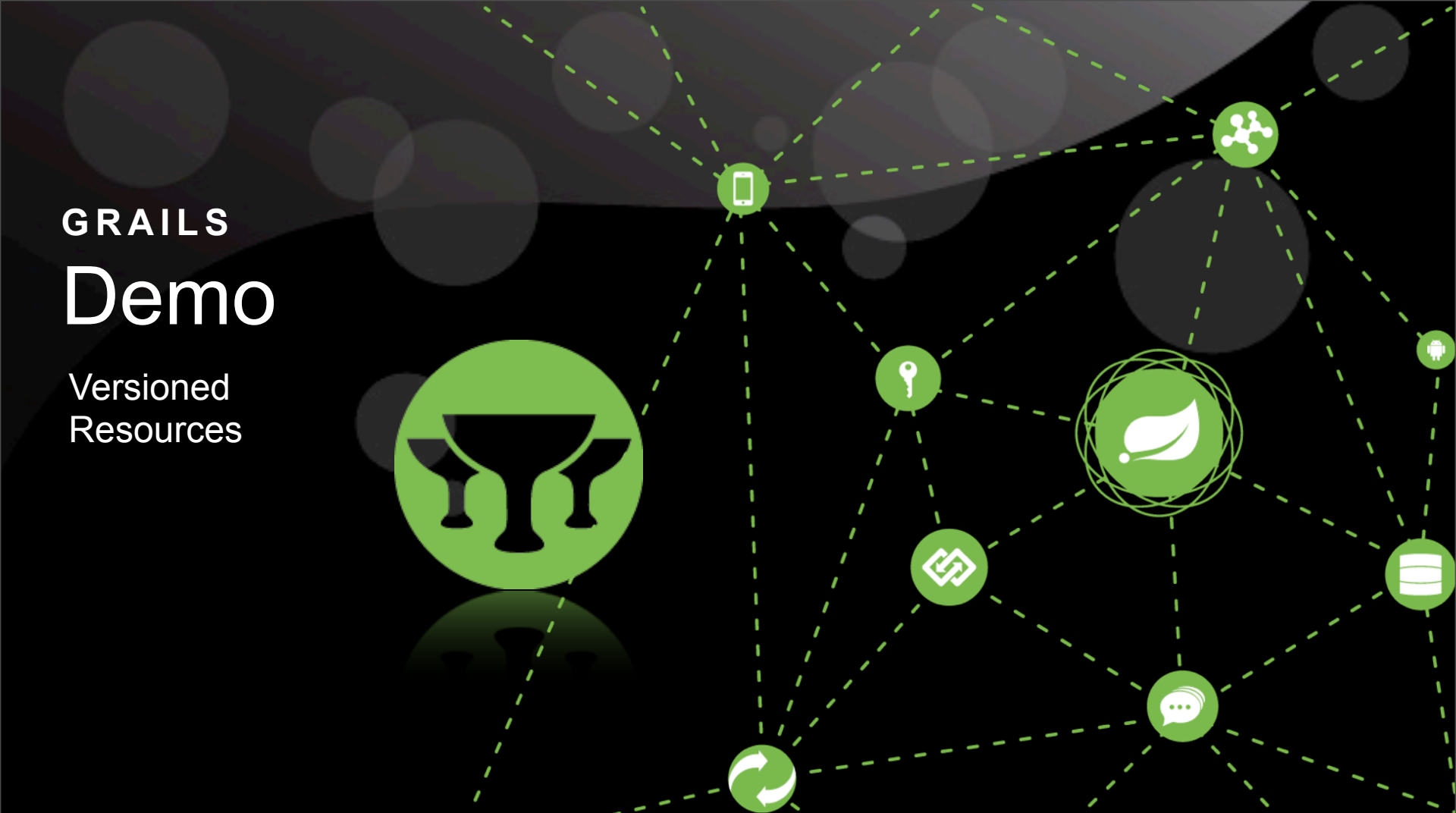
```
"/books"(resources:'book') {
  "/authors"(resources:"author")
}
```

HTTP Method	URI	Action
GET	/books/${bookId}/authors	index
POST	/books/${bookId}/authors	save
GET	/books/${bookId}/authors/${id}	show
PUT	/books/${bookId}/authors/${id}	update
DELETE	/books/${bookId}/authors/${id}	delete

GRAILS

Demo

Versioned Resources



REST: Versioned Resources

- Version by URI
- Version by “Accept-Version” header
- Version by Media Type

REST: Versioned Resources

- Version by URI
- Version by “Accept-Version” header
- Version by Media Type

```
"/books"(version:'1.0', resources:"book")
"/books"(version:'2.0', resources:"bookV2")
```

GRAILS

Demo

Controllers as
REST resources

REST: RestfulController Super Class

- Implements methods for all the typical REST operations (save, delete etc.)
- Subclasses can override / augment as necessary

REST: RestfulController Super Class

- Implements methods for all the typical REST operations (save, delete etc.)
- Subclasses can override / augment as necessary

```
import grails.rest.*
class PersonController extends RestfulController{
    PersonController() {
     super(Person)
    }
}
```

REST: RestfulController for Nested Resources

- Super class makes it easy to implement nested resources

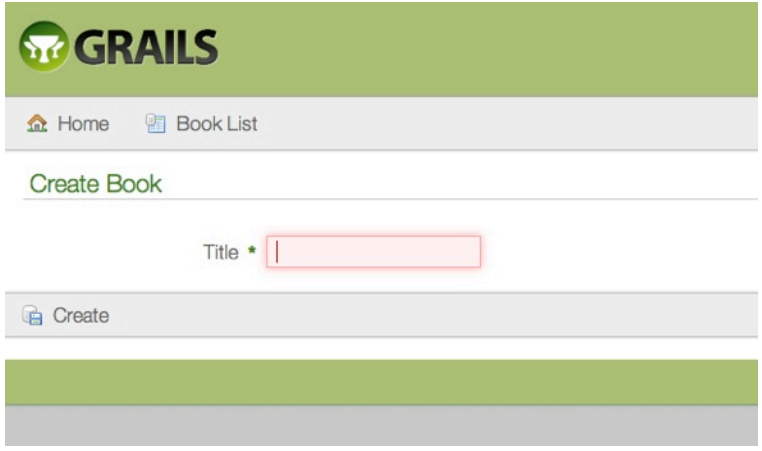
REST: RestfulController for Nested Resources

- Super class makes it easy to implement nested resources

```
class PersonController extends RestfulController{
  ...
  protected queryForResource(Serializable id) {
    def city = City.load(params.cityId)
    Person.findByCityAndId(city, id)
  }
}
```

REST: Scaffolding 2.0

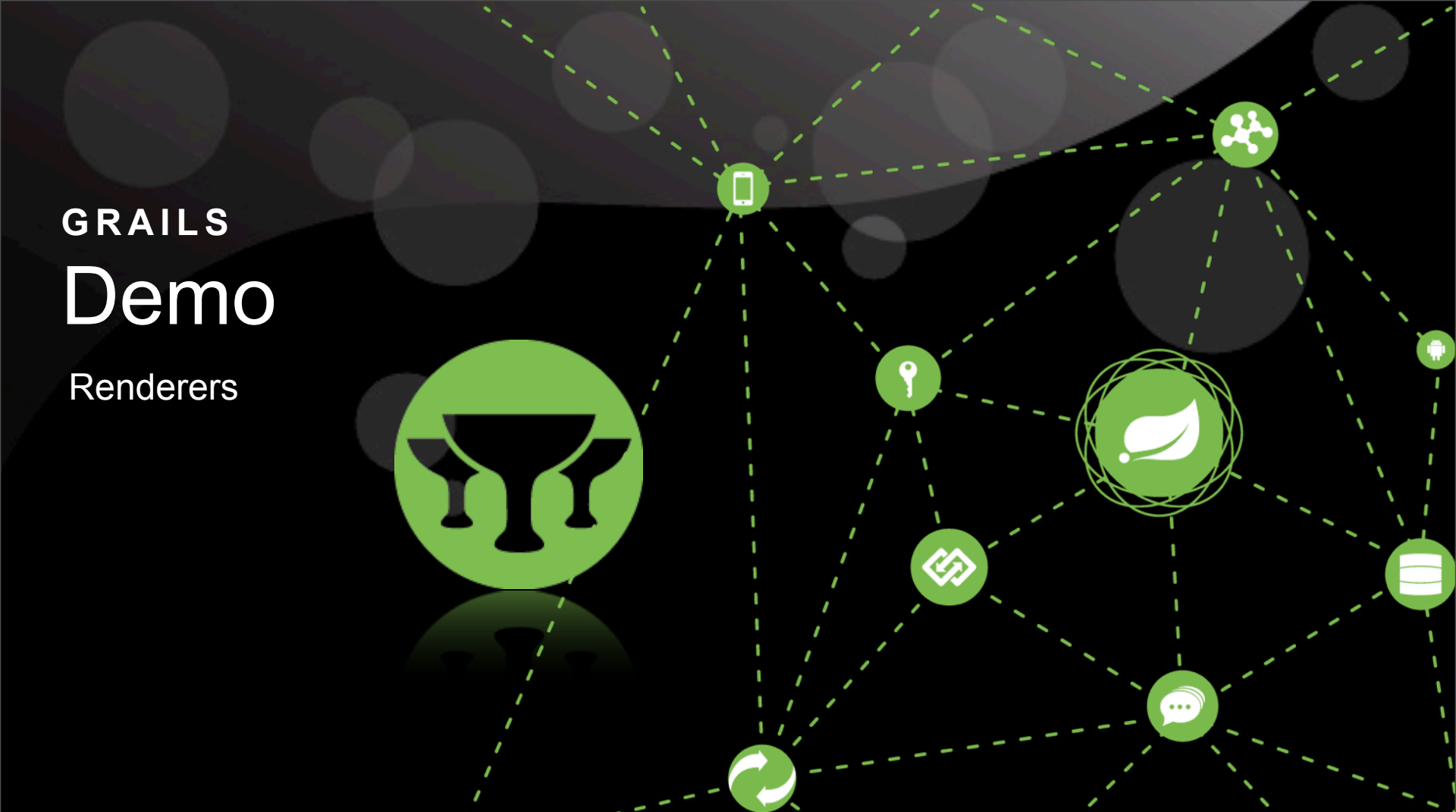
- Scaffolding support now a plugin
  - http://grails.org/plugin/scaffolding
- Generated controllers now RESTful
- New `generate-async-controller` command for Async support



GRAILS

Demo

Renderers



REST: Renderers

- Rendering customizable in resources.groovy, just render a Spring bean
- JSON, XML, HAL, Atom, Vnd.Error renders included
- Renderer can use any library (Jackson, GSON etc.)

REST: Renderers

- Rendering customizable in resources.groovy, just render a Spring bean
- JSON, XML, HAL, Atom, Vnd.Error renders included
- Renderer can use any library (Jackson, GSON etc.)

```
import grails.rest.render.json.*

beans = {
   personRenderer(JsonRenderer,  Person) {
       excludes = ['class', 'age']
   }
}
```

REST: What is a Render?

- Class that implements the Renderer interface

REST: What is a Render?

- Class that implements the Renderer interface

```
interface Renderer<T> extends MimeTypeProvider{

    /** @return The target type  */
    Class<T> getTargetType()

    /**
     * Renders the object
     */
    void render(T object, RenderContext context)
}
```

REST: Container Renderers

- Container renderers render containers (Lists, Maps, Errors) of a particular type
- You want different output from a single resources vs viewing multiple

REST: Container Renderers

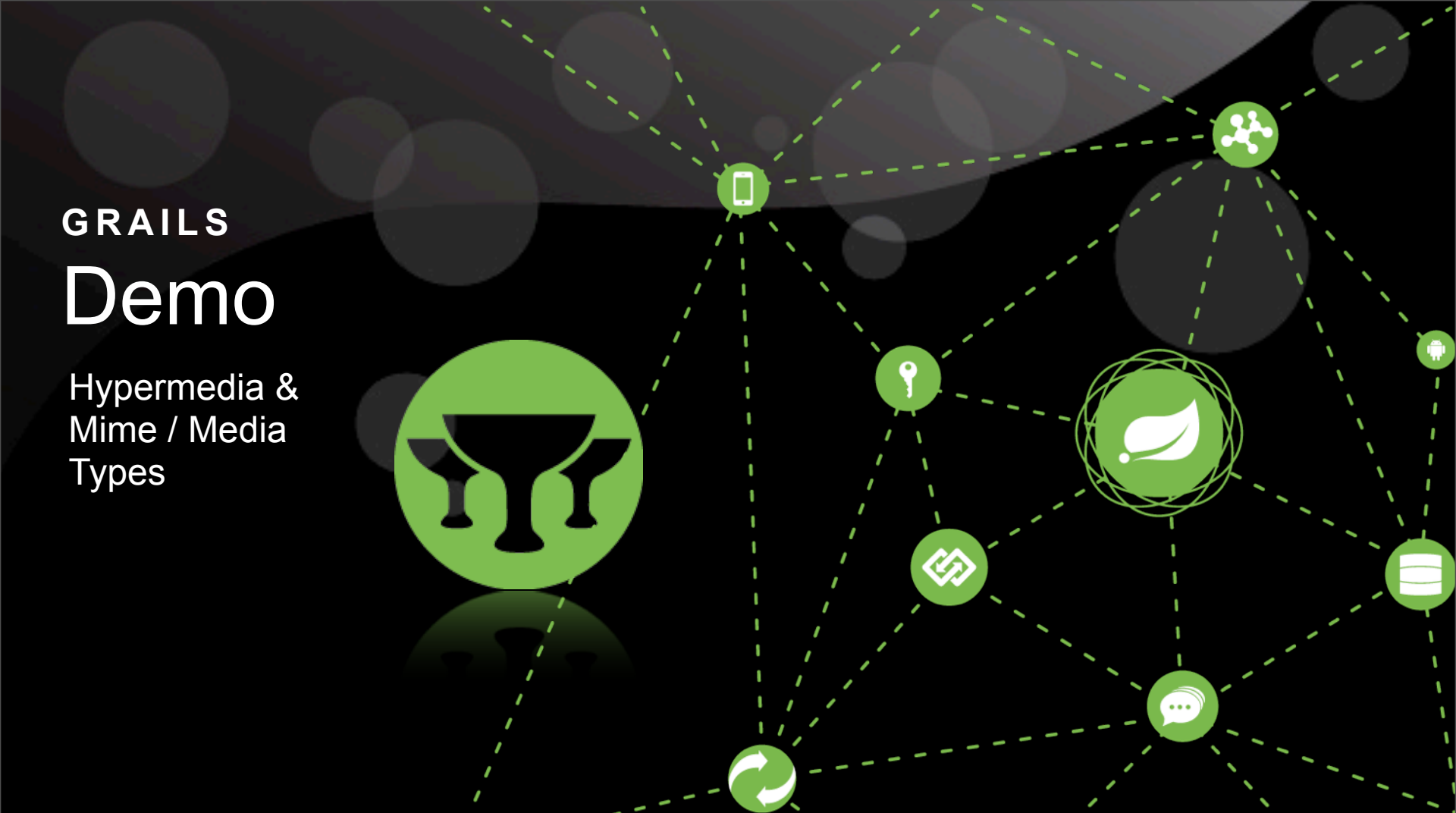
- Container renderers render containers (Lists, Maps, Errors) of a particular type
- You want different output from a single resources vs viewing multiple

```
interface ContainerRenderer<C, T>
   extends Renderer<C>{
    /* The generic type of the container*/
    Class<T> getComponentType()
}
```

GRAILS

Demo

Hypermedia &
Mime / Media
Types



REST: Hyper Media and Mime Types

- Declare Mime Types in Config.groovy
- Don’t declare new types first!

REST: Hyper Media and Mime Types

- Declare Mime Types in Config.groovy
- Don’t declare new types first!

```
grails.mime.types = [
    all:           '*/*',
    person: "application/vnd.foo.org.person+json",
    people:  "application/vnd.foo.org.people+json",
```

REST: Hyper Media and Mime Types

- Define Renderers that use those types in resources.groovy

REST: Hyper Media and Mime Types

- Define Renderers that use those types in resources.groovy

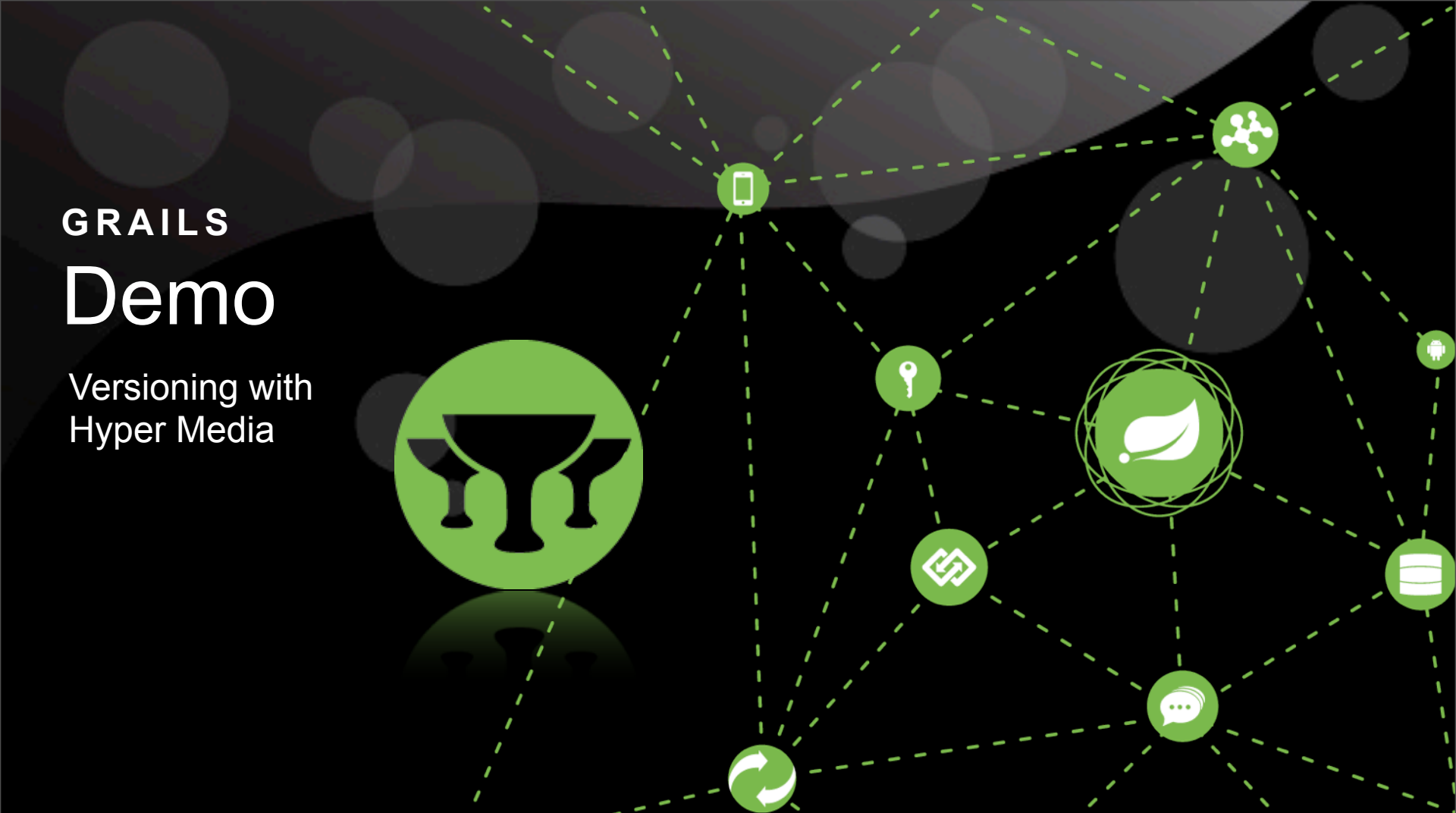
```
def personMimeType =
    new MimeType("application/vnd.foo.org.person+json")
beans = {
    personRenderer(JsonRenderer,
                   Person,
                   personMimeType)

}
```

GRAILS

Demo

Versioning with
Hyper Media



REST: Hyper Media Versioning

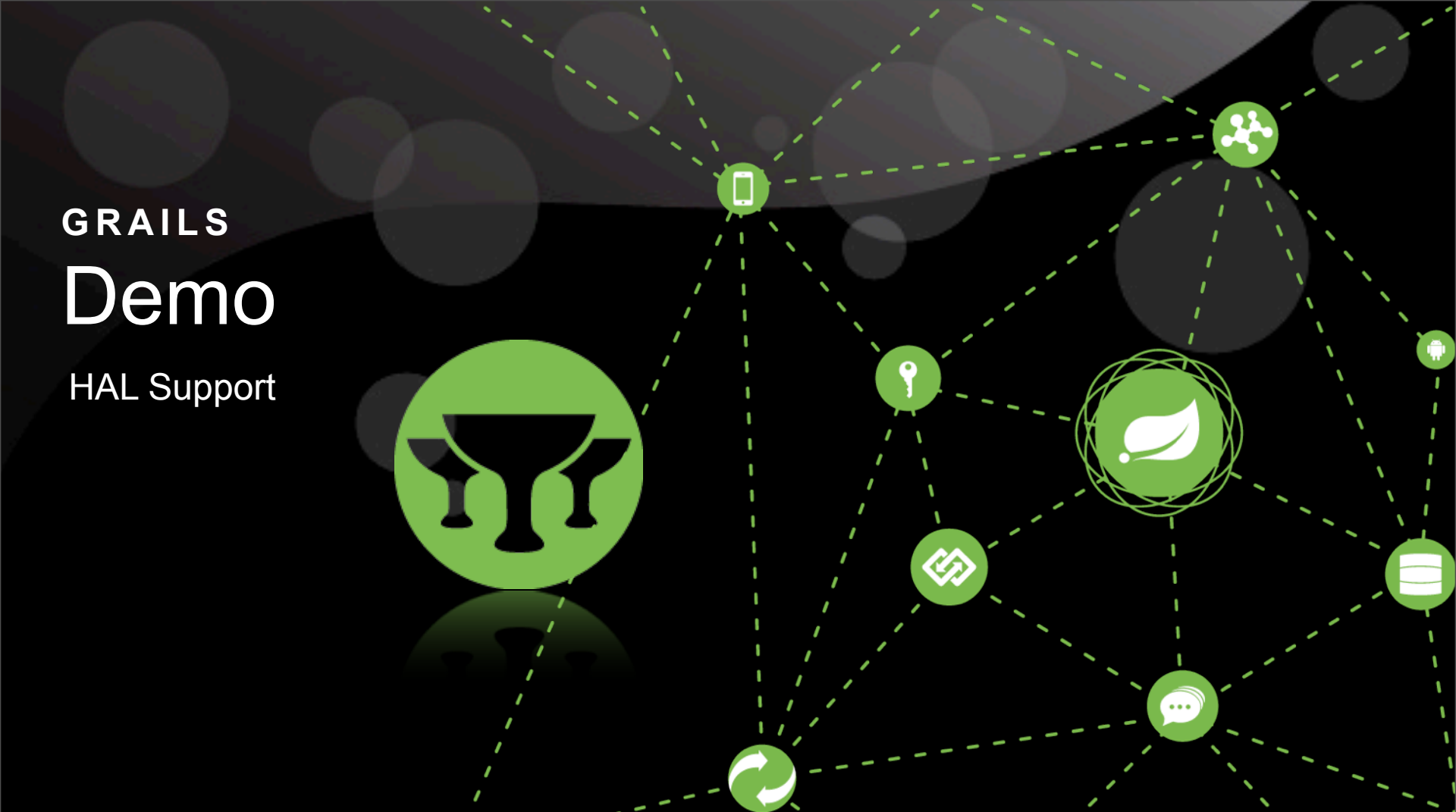
- You can version custom media types and use the media type to render different outputs

REST: Hyper Media Versioning

- You can version custom media types and use the media type to render different outputs

```
def personMimeType =
    new MimeType("application/vnd.foo.org.person+json",
                 [v: "1.0"] )
beans = {
    personRenderer(JsonRenderer,
                   Person,
                   personMimeType)

}
```


GRAILS
Demo
HAL Support

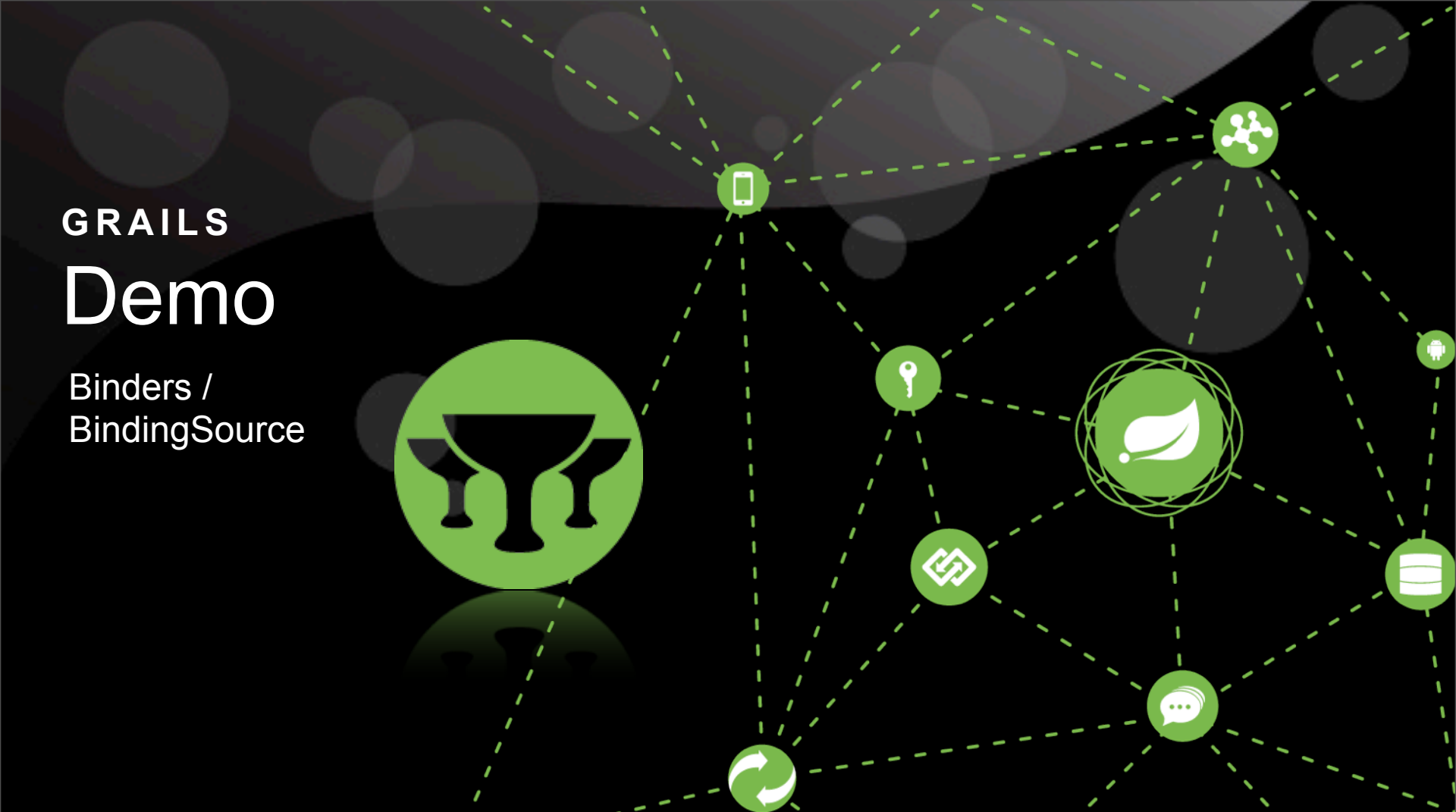
REST: Hyper Media with HAL

- HAL is a standardized format used in conjunction with HATEOS principals

REST: Hyper Media with HAL

- HAL is a standardized format used in conjunction with HATEOS principals

```
def personMimeType =
    new MimeType("application/vnd.foo.org.person+json",
                 [v: "1.0"] )
beans = {
    personRenderer(HalJsonRenderer,
                   Person,
                   personMimeType)
}
```


GRAILS
Demo
Binders /
BindingSource

REST: Binding / BindingSource

- If you define a custom render you may need to define a custom binding source to make binding work

REST: Binding / BindingSource

- If you define a custom render you may need to define a custom binding source to make binding work

```
protected DataBindingSource
            createBindingSource(Reader reader) {
   def json = JSON.parse(reader)
   def map = [
       firstName:json.first_name,
       lastName:json.last_name
   ]
   new SimpleMapDataBindingSource(map)
}
```

Goals of Async Support in Grails 2.3

- Be Grails-y (i.e. dead simple)
- Be Flexible (start simple, get deep)
- Be pluggable (extend and conquer)
- Be current (support latest techniques)

Async Features in 2.3

- Async Primitives (Promises, Lists, Maps etc.)
- Async Data (GORM)
- Async Everywhere (Tasks)
- Async Request Handling

GRAILS

Demo

Async Primitives

Async: Promise Primitives

- Foundational API for Async programming
- Includes notion of Promise
- Ability to combine / chain promises

Async: Promise Primitives

- Foundational API for Async programming
- Includes notion of Promise
- Ability to combine / chain promises

```
import static
grails.async.Promises.*

def p1 = task { 2 * 2 }
def p2 = task { 4 * 4 }
def p3 = task { 8 * 8 }
assert [ 4, 16, 64 ] ==
waitAll(p1, p2, p3)
```

Async: Promise Chaining

- Chain promises to formulate a value asynchronously

Async: Promise Chaining

- Chain promises to formulate a value asynchronously

```
import static grails.async.Promises.*
import static grails.async.*
Promise p =
   task { 2 * 2 } << { it * 4 } << { it * 8 }

p.onComplete { Integer value ->
   println "Total: $value"
}
```

Async: Promise Chaining

- Create list and map data structures from promises!

Async: Promise Chaining

- Create list and map data structures from promises!

```
import static grails.async.Promises.*
import static grails.async.*
PromiseList p =
   task { 2 * 2 } << { it * 4 } << { it * 8 }

p.onComplete { Integer value ->
   println "Total: $value"
}
```

Async: Promise Factory

- Override the creation of promise instances
  - Plugin in Reactor
  - Change to synchronous promises for testing

Async: Promise Factory

- Override the creation of promise instances
  - Plugin in Reactor
  - Change to synchronous promises for testing

```
import org.grails.async.factory.*
import grails.async.*
Promises.promiseFactory =
  new SynchronousPromiseFactory()
```

Async: @DelegateAsync Transform

- Transform any synchronous API into an asynchronous one
- Takes each method and returns a promise

Async: @DelegateAsync Transform

- Transform any synchronous API into an asynchronous one
- Takes each method and returns a promise

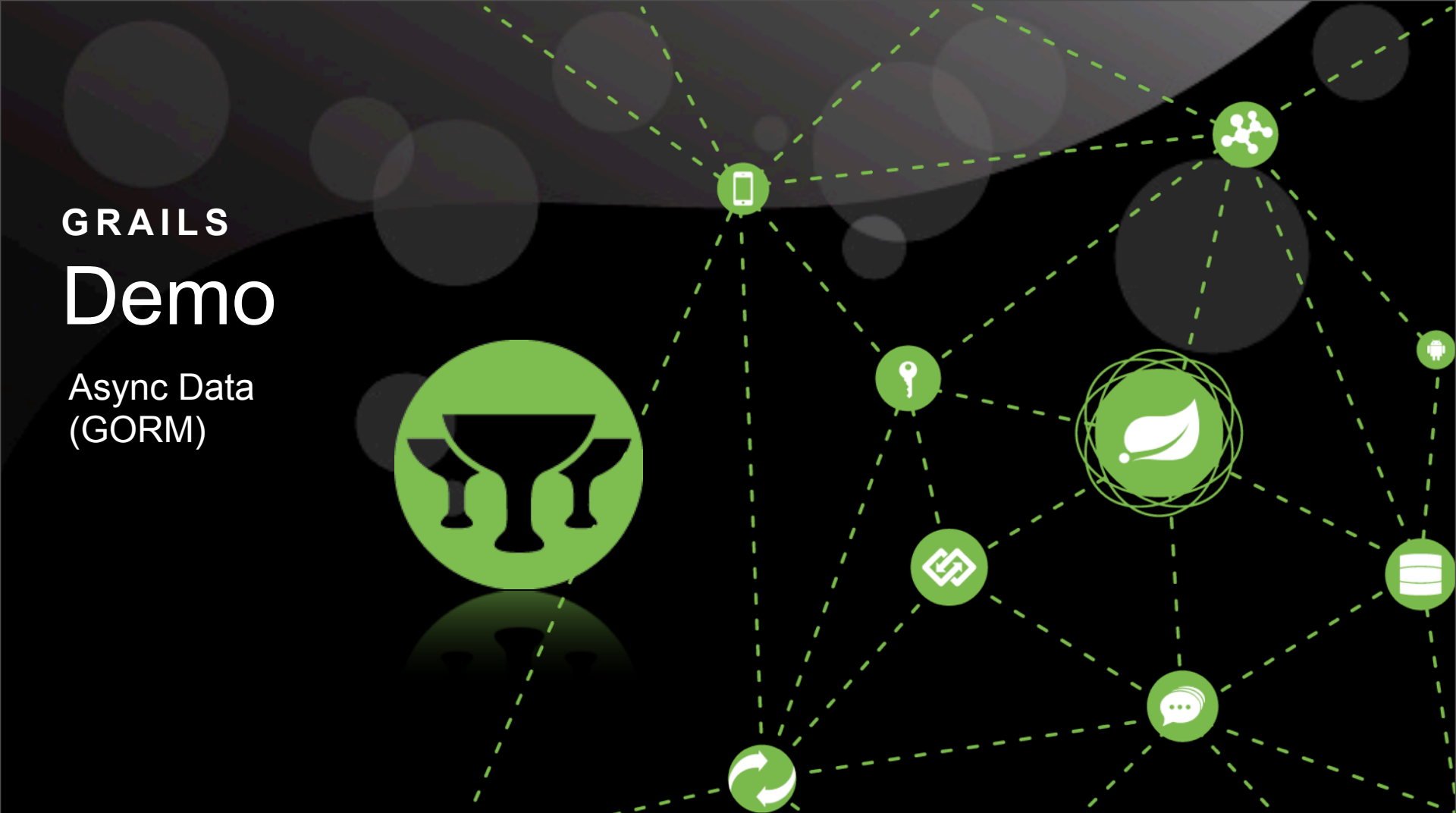
```
import grails.async.*

class AsyncBookService {
    @DelegateAsync
    BookService bookService
}
```

GRAILS

Demo

Async Data
(GORM)



Async: GORM

- Makes all GORM methods async
- Each method returns a `Promise`
- Deals with binding session to background thread
- Works across all datastores (MongoDB, GORM for REST etc.)

Async: GORM

- Makes all GORM methods async
- Each method returns a `Promise`
- Deals with binding session to background thread
- Works across all datastores (MongoDB, GORM for REST etc.)

```
def p1 = Person.async.get(1)
def p2 = Person.async.get(2)
def p3 = Person.async.get(3)

def results =
       waitAll(p1, p2, p3)
```

Async: GORM Tasks

- Batch up a bunch of Grails queries to be executed asynchronously

Async: GORM Tasks

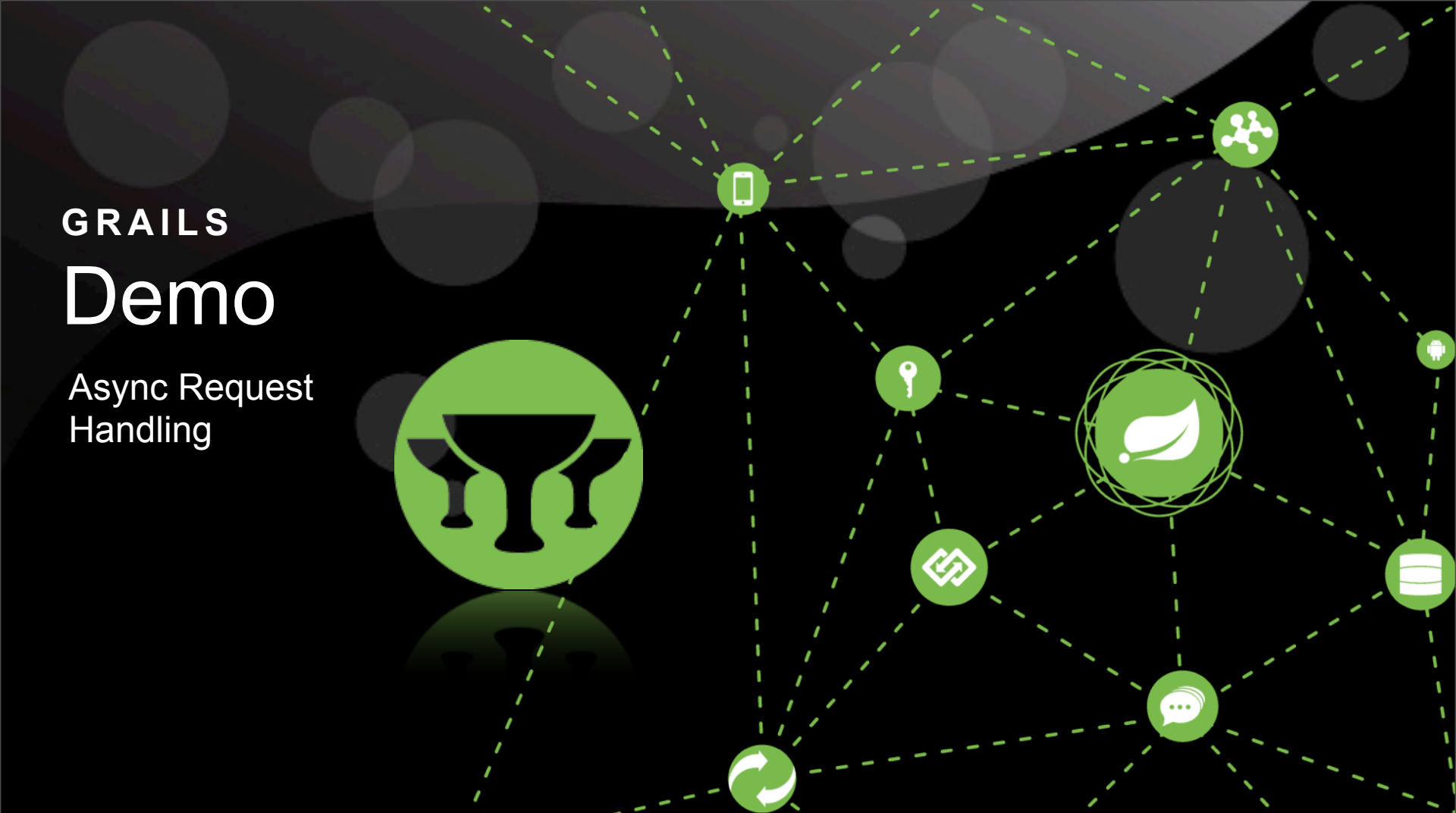
- Batch up a bunch of Grails queries to be executed asynchronously

```
def promise = Person.async.task {
    withTransaction {
        def person = findByFirstName("Homer")
        person.firstName = "Bart"
        person.save(flush:true)
    }
}
Person updatedPerson = promise.get()
```

GRAILS

Demo

Async Request Handling



Async: Request Processing

- Handle requests asynchronously
- Uses Servlet 3.0 async under the covers
- Create asynchronous models

Async: Request Processing

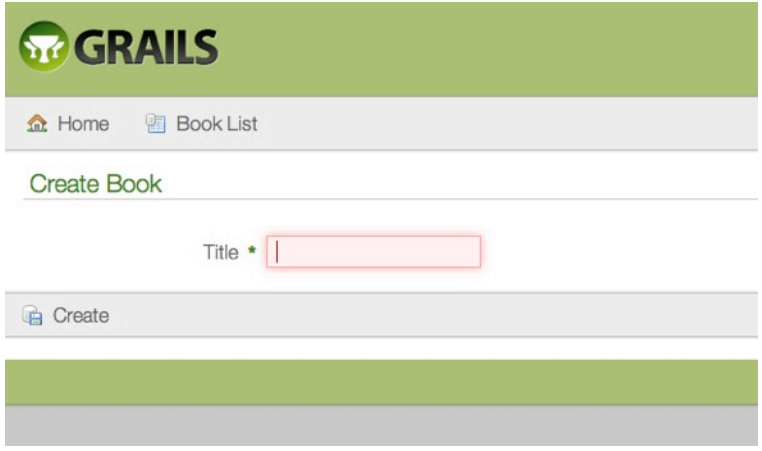
- Handle requests asynchronously
- Uses Servlet 3.0 async under the covers
- Create asynchronous models

```
import static
grails.async.Promises.*

class PersonController {
  def index() {
   tasks books: Book.async.list(),
         total: Book.async.count(),
         otherValue: {
            // do hard work
         }
   }
}
```

REST: Generating Async Controllers

- New scaffolding can generate an example Async controller
  - http://grails.org/plugin/scaffolding
- Use `generate-async-controller` command



Q & A

REST & Async

Learn More. Stay Connected.

Web: grails.org
Twitter: twitter.com/grailsframework
LinkedIn: http://linkedin.com/groups/Grails-User-Group-39757
Google +: https://plus.google.com/communities/109558563916416343008