trivadis
makes IT easier.

Apache incubator ™

Anatole Tresch
Trivadis AG
@atsticks

trivadis
makes IT easier.

Apache incubator ™

Apache Tamaya

Configure Everything...

Bio

Anatole Tresch

- Principal Consultant, Trivadis AG (Switzerland)
- Star Spec Lead JSR 354
- Technical Architect, Lead Engineer
- PPMC Member Apache Tamaya

- Twitter/Google+: @atsticks
- anatole@apache.org
- anatole.tresch@trivadis.com
- JUG Switzerland

Agenda

- Introduction
- What to configure?
- Apache Tamaya
  - Core Concepts
  - Extensions
- Demo & Outlook

Introduction

History of Apache Tamaya

- **2012**: Configuration was voted an important aspect for Java EE 8
- **2013**:
  - Setup of Java EE Configuration JSR failed
  - Standardization on SE Level did not have enough momentum
- **BUT**:
  - Configuration is a crucial cross cutting concern
  - There is no (really NO!) defacto standard
  - Reinventing the wheel is daily business

The People behind Tamaya

- John D. Ament (Mentor)
- David Blevins (Champion)
- Werner Keil
- Gerhard Petracek (Mentor)
- Mark Struberg (Mentor)
- Anatole Tresch
- Oliver B. Fischer
- ...

The Objectives of Apache Tamaya

- Define a common API for accessing configuration
  - Minimalistic
  - Flexible, pluggable and extendible
- Compatible with Java 7 and beyond
- Provide a reference implementation
- Provide Extension Modules for additional features
- Build up a community
- If possible, create a Standard!

What to configure?

What to configure?

- **Most complex question ever!**
- Divergent views
  - Servers, Containers, Machines, Clusters, ...
  - Parameters of a runtime (staging, localization etc.)
  - Deployments, Network, ...
  - Technology-specific components (beans, wirings etc.)
  - Resources (data sources, message queues, services etc.)
  - Descriptive or imperative style
- Different granularities, varying levels of applicability, different formats …

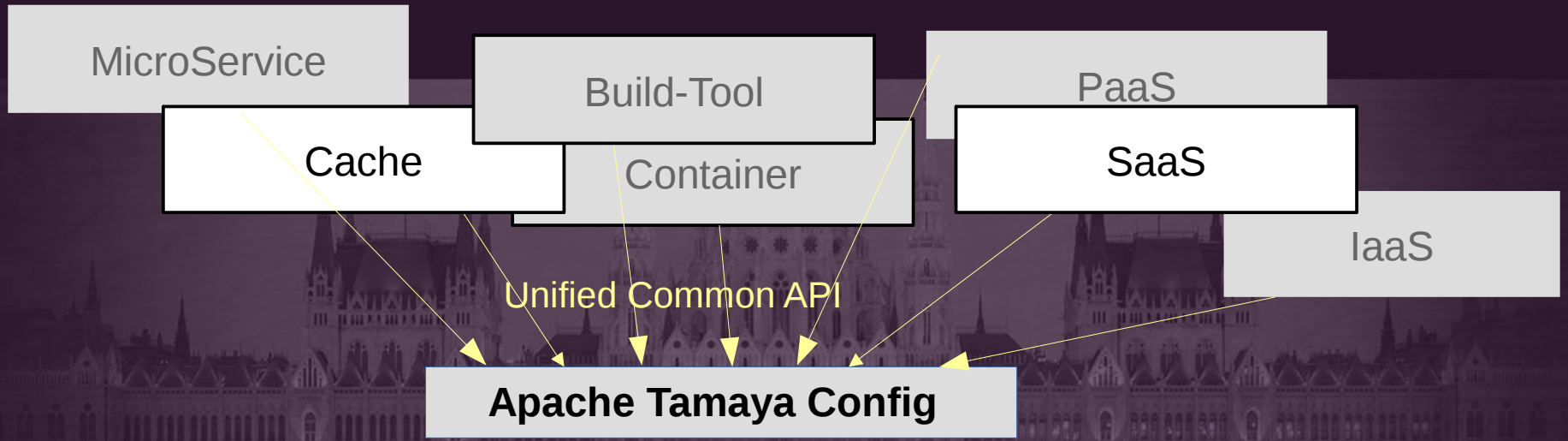
How to configure?

- **Format**
- **Storage**
- **Lifecycle and versioning**
- **Security**
- **Distribution**
- **Consistency**

Apache incubator ™

Use Cases

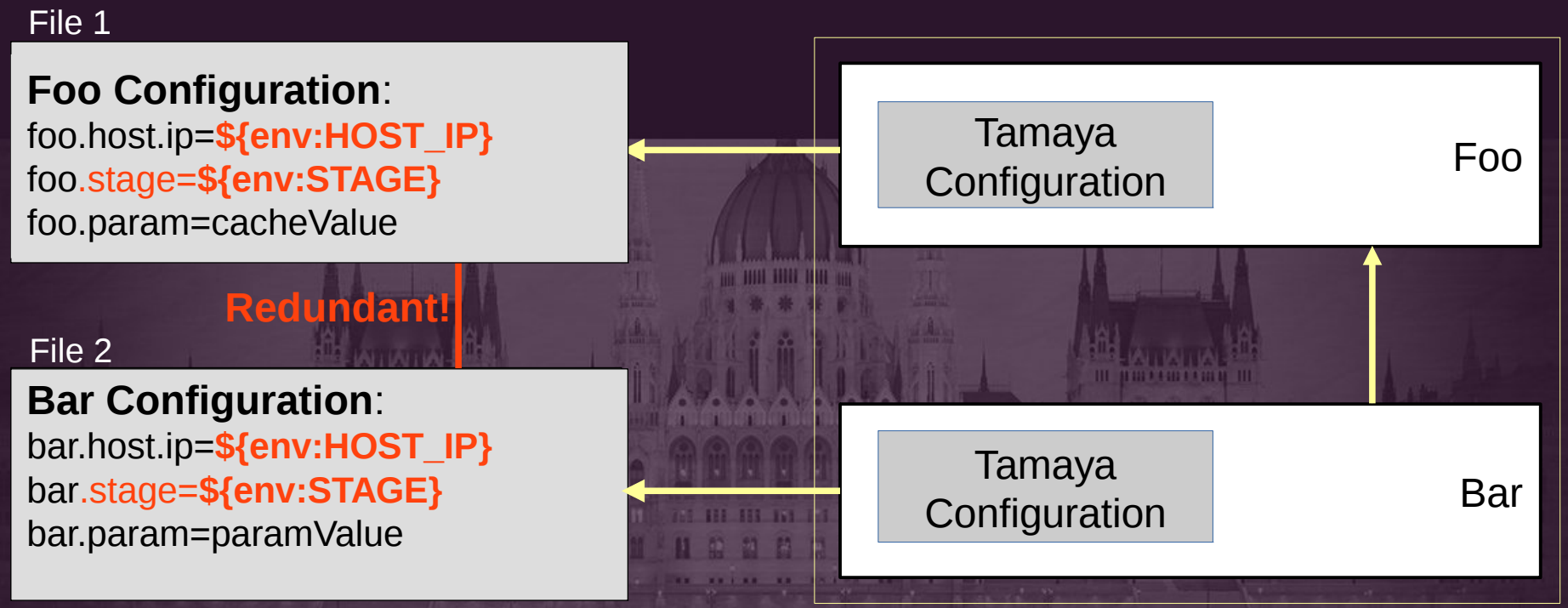
UC: Access Configuration Similarly



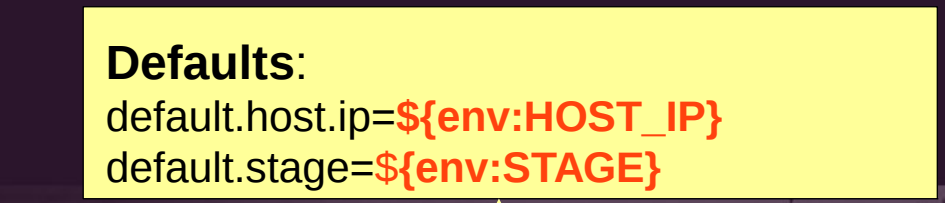
- Any Location
- Any Format
- Any Policy

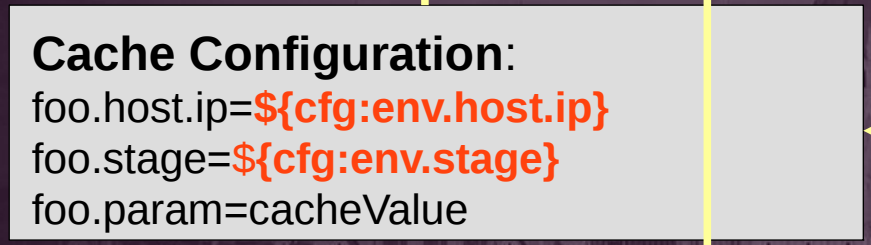
Extension Points:
SPI

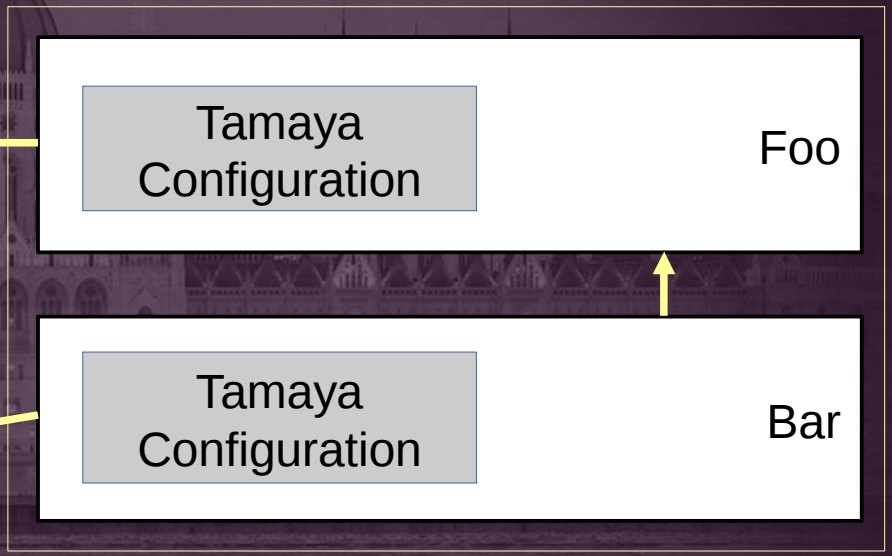
UC: Reduce Redundancy



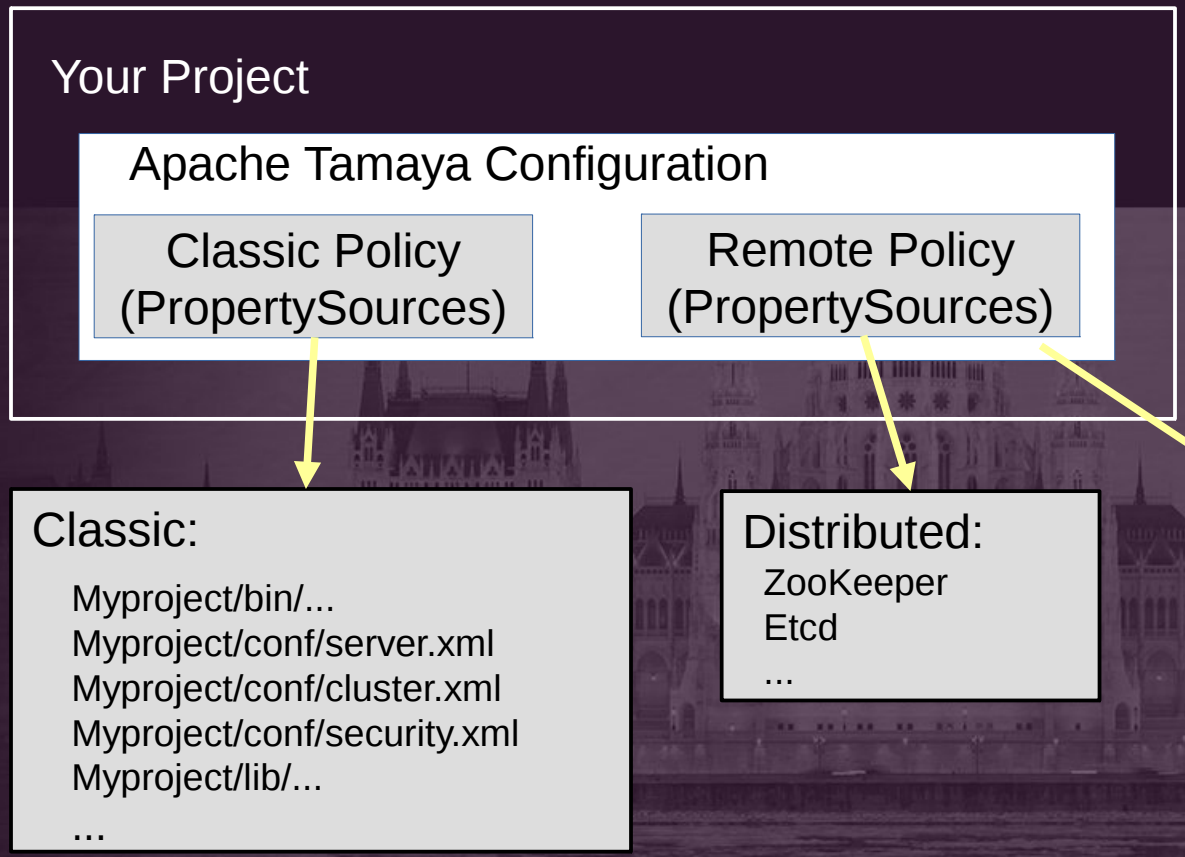
UC: Convention over Configuration







UC: Pluggable Config Backends

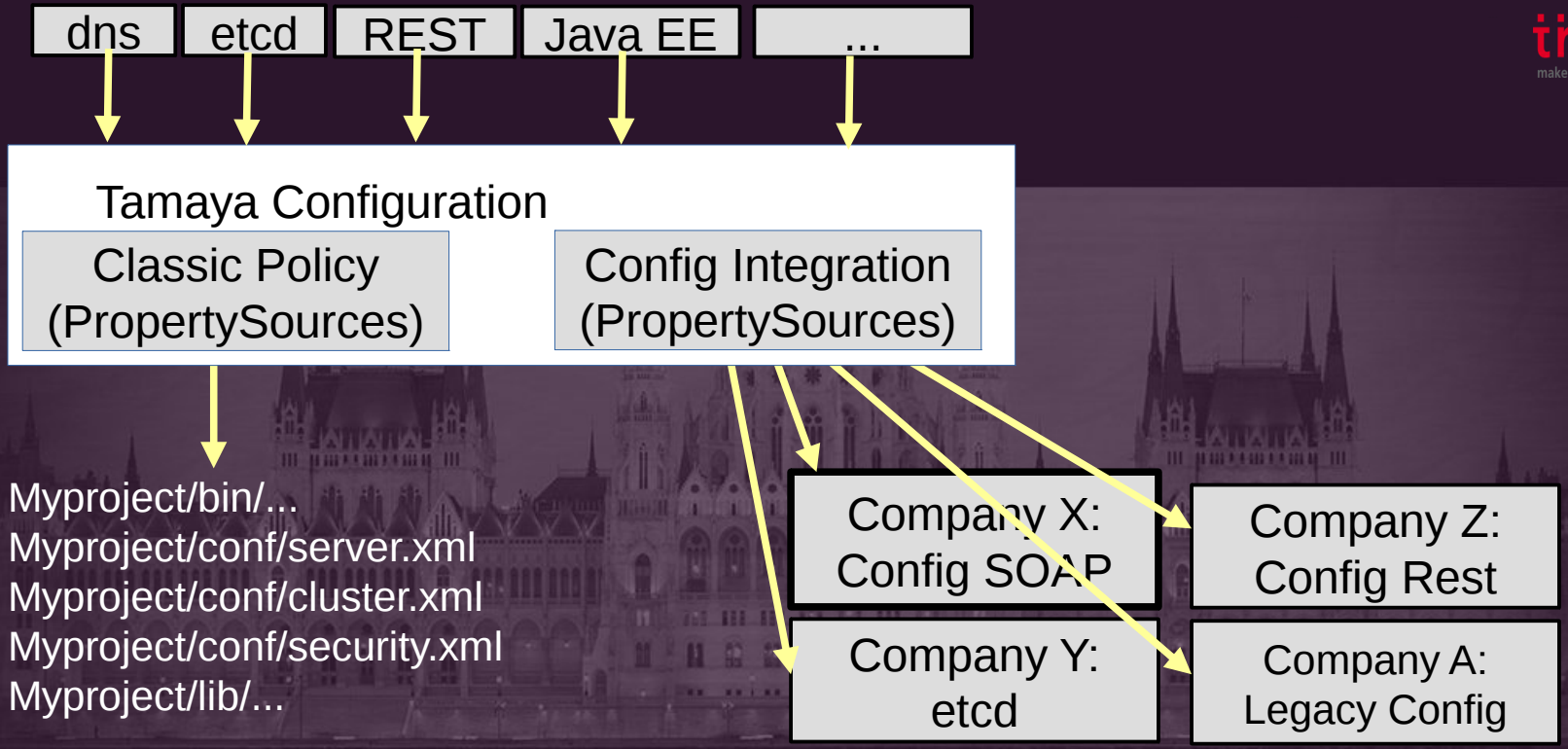


Unified API for configuration access

Policies can be provided as jar-artifacts separately

Additional benefits: config documentation

UC: Enterprise Integration



UC: Enforceable Policies

Define, implement and distribute a Company Wide Configuration Policy

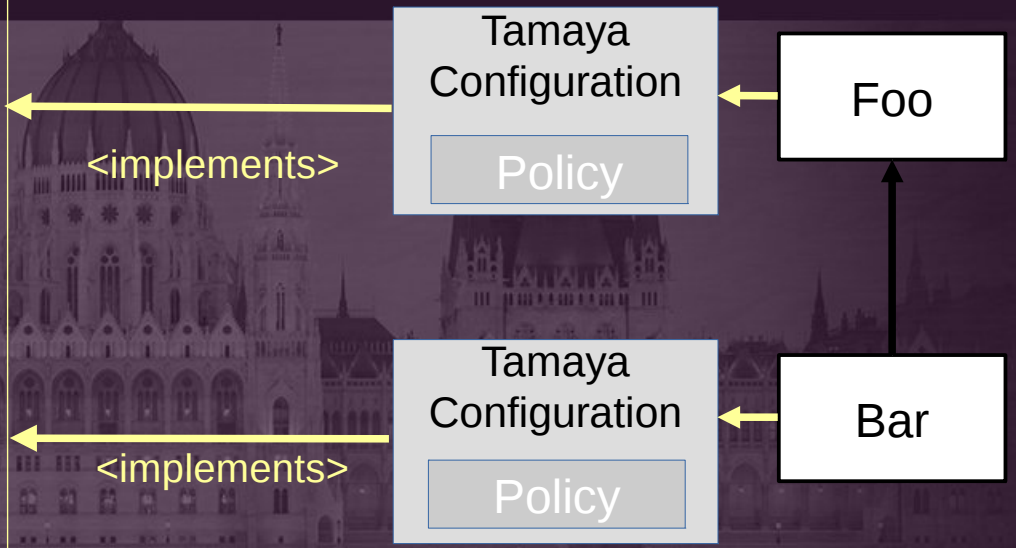
Config Policy

Locations:

- `file:${filedir}`
- `classpath:META-INF/defaults/configuration.*`
- `classpath:META-INF/${STAGE}/configuration.*`
- `url:http://myserver.int/cfg`
- System properties
- Env. Properties

Formats:

- properties
- xml
- json
- yaml



UC: Documentation

Tamaya allows to validate and document the configuration hooks used throughout your system!

Config Documentation

```
[Group, size: 8:
  >> Section: a.test.existing
  >> Parameter: a.test.existing.aParam, required: true
  >> Parameter: a.test.existing.optionalParam
  >> Parameter: a.test.existing.aABCParam, expression: [ABC].*
  >> Section: a.test.notexisting
  >> Parameter: a.test.notexisting.aParam, required: true
  >> Parameter: a.test.existing.aABCParam2, expression:
     [ABC].*
  >> Section: a.b.c.aRequiredSection.optional-subsection,
    >> Parameter: MyNumber,
      >> Section: a.b.c,
        >> Section: a.b.c.aRequiredSection.nonempty-subsection,
          >> Section: a.b.c.aRequiredSection
•             - Parameter:
a.b.c.aRequiredSection.subsection.param00,
```

Configuration

Config Validation

•MISSING: a.test.notexisting.aParam (Parameter),
ERROR: a.test.existing.aABCParam2 (Parameter)
-> Config value not matching expression: [ABC].*, was MMM
MISSING: a.params2 (Parameter),
MISSING: a.b.c.aRequiredSection.subsection.param1 (Parameter),
UNDEFINED: JAVA_HOME (Parameter)
UNDEFINED: SESSION_MANAGER (Parameter)
[...]

Summary: Why we need Tamaya?

trivadis
makes IT easier.

- Stop Reinventing the wheel
- Stay Focused on what to configure, not how!
- Reduce Redundancies and Inconsistencies
- Document your configuration options
- Integrate with Company Infrastructure („DevOps“)
- Decouple your configuration backends

- Manage Config like APIs !

The API

Let's start simple!

- Add dependency

  `org.apache.tamaya:core: 0.2-incubating` (not yet released)

- Add Config to `META-INF/javaconfiguration.properties`

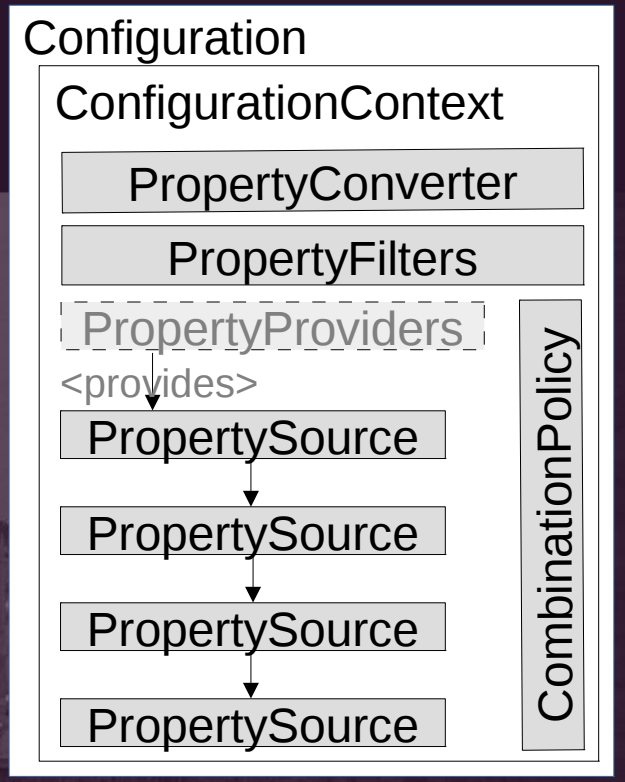
```
Configuration config =
                ConfigurationProvider.getConfiguration();

String name = config.getOrDefault("name", "John");
int ChildNum = config.get("childNum", int.class);
```

Configuration Interface


```
public interface Configuration{

  String get(String key);
  String getOrDefault(String key, String defaultValue);

  <T> T get(String key, Class<T> type);
  <T> T get(String key, TypeLiteral<T> type);
  <T> T getOrDefault(String key, Class<T> type, T defaultValue);
  <T> T getOrDefault(String key, TypeLiteral<T> type, T defaultValue);

  Map<String,String> getProperties();

  // Functional Extension Points
  Configuration with(ConfigOperator operator):
  <T> T query(ConfigQuery<T> query);
}
```

Tamaya Design in 60 Seconds...

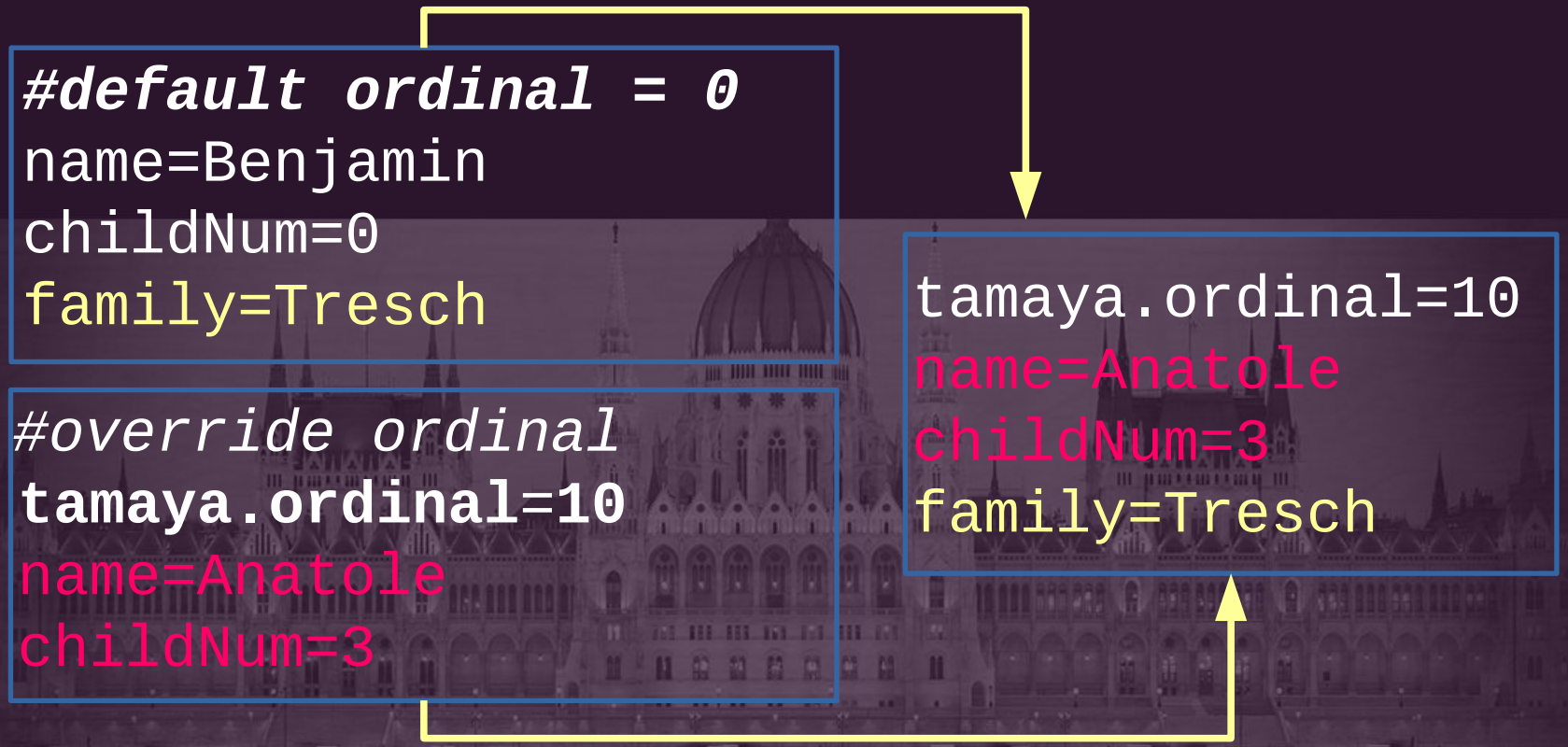
1. **`Configuration`** = ordered list of **`PropertySources`**
2. Properties found are **combined** using a **`CombinationPolicy`**
3. Raw properties are **filtered** by **`PropertyFilter`**
4. For typed access **`PropertyConverters`** have to do work
5. **Extensions** add more features (discussed later)
6. **Lifecycle** is controlled by the **`ServiceContextManager`**



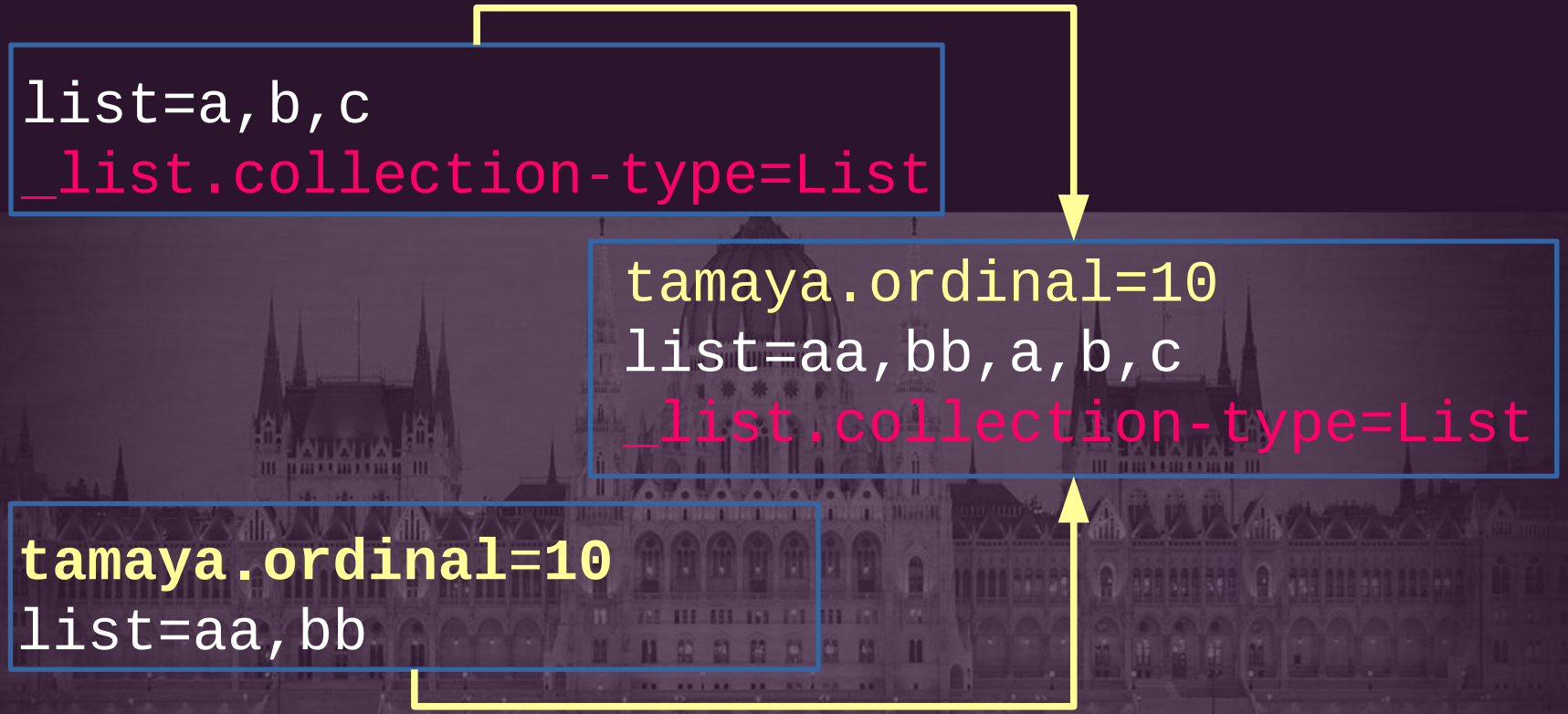
SPI: PropertySource PropertySourceProvider


```
public interface PropertySource {

  static final String TAMAYA_ORDINAL = "tamaya.ordinal";

  String getName();
  int getOrdinal();
  String get(String key);
  Map<String, String> getProperties();
}

public interface PropertySourceProvider {
  Collection<PropertySource> getPropertySources();
}
```

Overriding Explained



CombinationPolicy



There is more! - Extension Modules

Extensions: a topic on its own!

- *Tamaya-**spi-support**:* Some handy base classes to implement SPIs
- *Tamaya-**functions**:* Functional extension points (e.g. remapping, scoping)
- *Tamaya-**events**: Detect and publish ConfigChangeEvents*
- *Tamaya-**optional**: Minimal access layer with optional Tamaya support*
- *Tamaya-**filter**: Thread local filtering*
- *Tamaya-**inject-api**: Tamaya Configuration Injection Annotations*
- *Tamaya-**inject**:* Configuration Injection and Templates SE Implementation (lean, no CDI)
- *Tamaya-**resolver**:* Expression resolution, placeholders, dynamic values
- *Tamaya-**resources**:* Ant styled resource resolution
- *Format Extensions:* yaml*, json, ini, … including formats-SPI
- Integrations with **CDI, Spring, OSGI, Camel, etcd**
- *Tamaya-**classloader-support**:* Managing Tamaya Services within Classloading Hierarchies
- *Tamaya-**mutable-config**: Writable ConfigChangeRequests*
- *Tamaya-**server**:* REST/JSON Configuration Server
- *Tamaya-**remote***: Integrate Tamaya Server resources as PropertySource
- *Tamaya-**model***: Configuration Model and Auto Documentation
- *Tamaya-**collections***: Collection Support
- ...

* work in progress

Planned Features

- *Java EE:* Configuring EE, where possible
- More Integrations:
  - Commons-config (currently in experimental stage)
  - Additional Backends: Consul, Zookeeper, ElasticSearch, Redis, …
  - ...
- Runtime Integrations:
  - Vertx
  - Docker?
  - ...

- *„We are looking for committers!“*

Example: Configuration Injection


```
@ConfiguredType(defaultSections=”com.mycomp.tenantAdress”)
public final class MyTenant{

  private String name;

  @ConfiguredProperty(
   defaultValue=”2000”)
  private long customerId;

  @ConfiguredProperty(keys={
   ”privateAddress”,”businessAdress”,
   ”[my.qualified.adress]”
  })
  private String address;
  ...
}
```

```
MyTenant t = new MyTenant();
ConfigurationInjection
    .getConfigurationInjector()
    .configure(t);
```

```
@RequestScoped
public class MyClass{
  @Inject
  private MyTenant t;
  …
}
```

Configuration Template


```
@ConfiguredType(defaultSections=”com.mycomp.tenantAdress”)
public interface MyTenant{

  public String getName();

  @ConfiguredProperty(
    defaultValue=”2000”)
  public long getCustomerId();

  @ConfiguredProperty(keys={
   ”privateAddress”,”businessAdress”,
   ”[my.qualified.adress]”
  })
  public String getAddress();

}
```

```
MyTenant t =
ConfigurationInjection
      .getConfigurationInjector()
      .createTemplate(MyTenant.class);
```

Demo

Demo Setup - Microservices

- What you will see:
- - Starting with a simple config client
- - Adding configuration based user/password auth
  1) - Local Properties Only
  2) - Etcd d Backend
- - Starting with DropWizard
- - Adding Spring Boot

```
GET     /config (dropexample.HelloWorldResource)
GET     /hello (dropexample.HelloWorldResource)
GET     /login (dropexample.LoginResource)
POST    /login (dropexample.LoginResource)
GET     /user (dropexample.LoginResource)
```

Summary

Apache Tamaya Provides

- A Complete thread- and type-safe Configuration API compatible with all commonly used runtimes
- A simple, but extendible design
- A myriad of extensions
- A small footprint
- Remote Support
- Almost „Swiss Made“

Links

- Project Page: http://tamaya.incubator.apache.org
- Twitter: @tamayaconfig
- Blog: http://javaeeconfig.blogspot.com
- Presentation by Mike Keith on JavaOne 2013: https://oracleus.activeevents.com/2013/connect/sessionDetail.ww?SESSION_ID=7755
- Apache Deltaspike: http://deltaspike.apache.org
- Java Config Builder: https://github.com/TNG/config-builder
- Apache Commons Configuration: http://commons.apache.org/proper/commons-configuration/
- Jfig: http://jfig.sourceforge.net/
- Carbon Configuration: http://carbon.sourceforge.net/modules/core/docs/config/Usage.html
- Comparison on Carbon and Others: http://www.mail-archive.com/commons-dev@jakarta.apache.org/msg37597.html
- Spring Framework: http://projects.spring.io/spring-framework/
- Owner: http://owner.aeonbits.org/

Q&A

trivadis
makes IT easier.

Thank you!

Anatole Tresch
Trivadis AG
Principal Consultant
Twitter/Google+: @atsticks
anatole@apache.org
anatole.tresch@trivadis.com

API Entry Point: ConfigurationProvider


```
public class ConfigurationProvider{

  public static Configuration getConfiguration();
  public static ConfigurationContext getConfigurationContext();

  public static ConfigurationContextBuilder
                                  getConfigurationContextBuilder()

  public static void setConfigurationContext(
                                  ConfigurationContext context);
}
```

Configuration vs Components

Configuration:
- Descriptive Data
- Key, value pairs
- Static (Relatively)
- Serializable
- Overriding
- References

Components:
- Behaviour and state
- Code
- Dynamic
- Not serializalbe
- Inheritance
- Usage