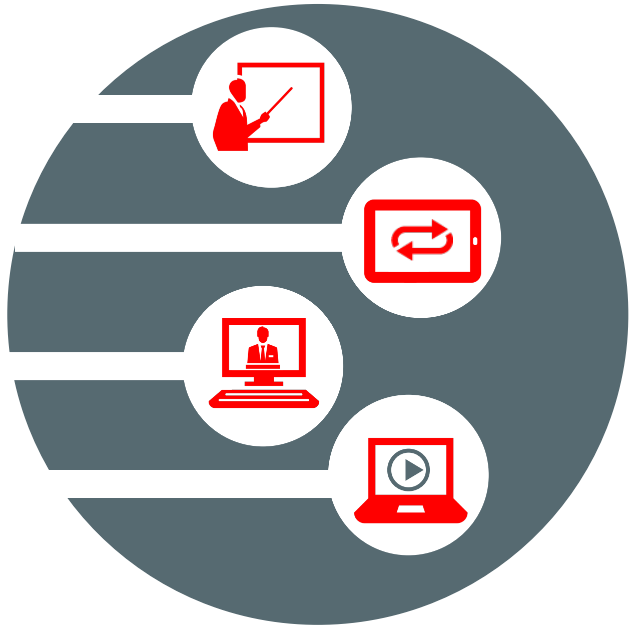
ORACLE®

JavaOne™

ORACLE®

Keep Learning with Oracle University

ORACLE UNIVERSITY



education.oracle.com

Session Surveys

Help us help you!!

- Oracle would like to invite you to take a moment to give us your session feedback. Your feedback will help us to improve your conference.
- Please be sure to add your feedback for your attended sessions by using the Mobile Survey or in Schedule Builder.

Finally, EE Security API JSR 375

Alex Kosowski
JSR 375 Specification Lead
Oracle, WebLogic Server Security
October 27, 2015

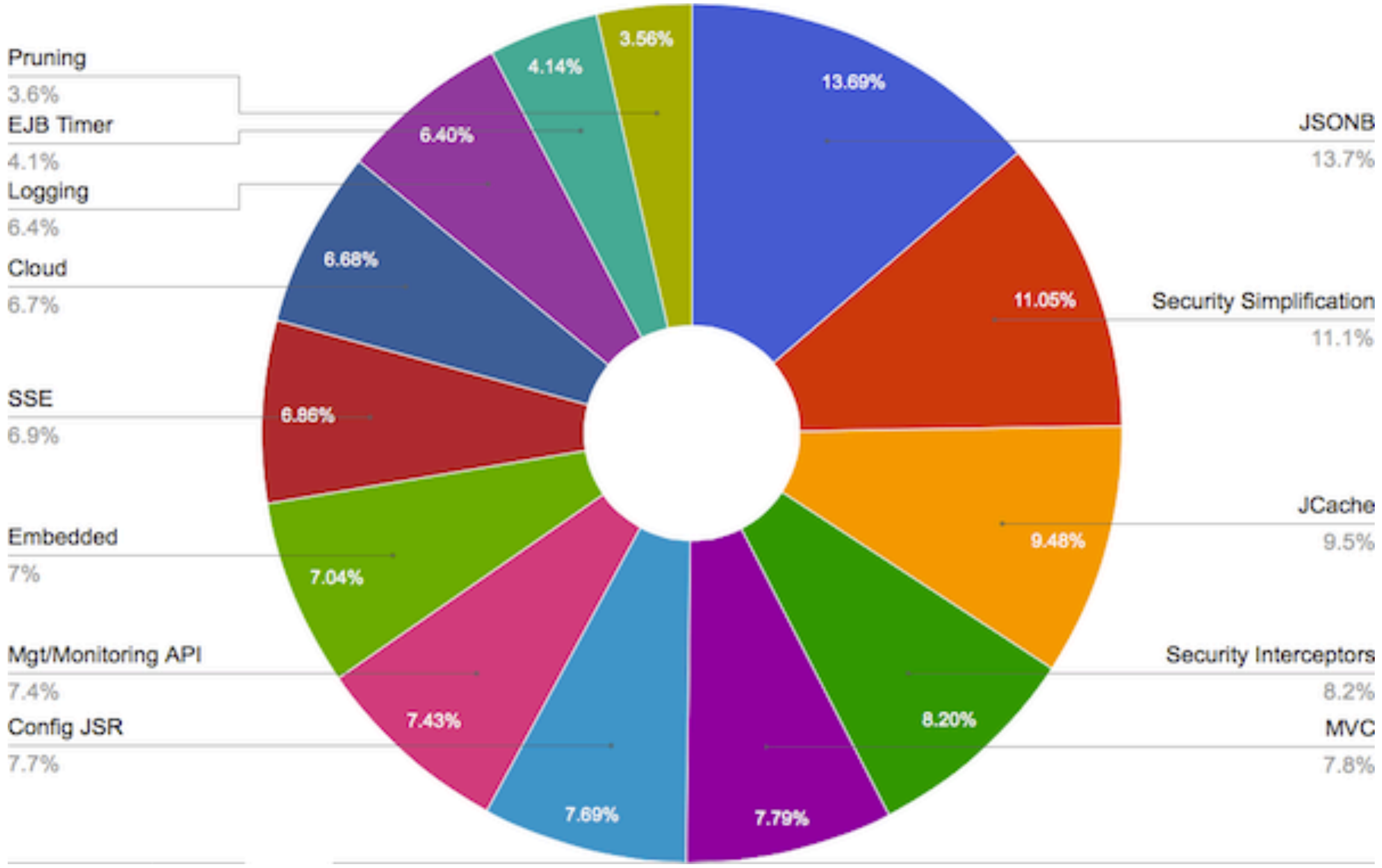
Safe Harbor Statement

Program Agenda

Program Agenda

Why a Java EE Security API JSR?

- EE 8 survey results
- 4500 total responses
- Priorities Pie Chart



What’s wrong with Java EE Security?

The community says…

"The ultimate goal is to have basic security working without the need of any kind of ***vendor specific*** *configuration, deployment descriptors, or whatever. ”*
– Arjan Tijms

“[The EE security] model is problematic in cloud/PaaS environments where developers do not necessarily have easy access to non-standard vendor runtime features and a self-contained application is much easier to manage.”
– Reza Rahman

What’s wrong with Java EE Security?

- Java EE Security viewed as not portable, abstract/confusing, antiquated
- Doesn’t fit cloud app developer paradigm: requires app server configuration
- Losing value to non-standard 3rd Party Frameworks…less likely to move back to Java EE

What to do?

- Plug the portability holes
- Modernize
  - Contexts and Dependency Injection (CDI)
    - Intercept at Access Enforcement Points: POJO methods
  - Expression Language (EL)
    - Enable Access Enforcement Points with complex rules
  - Lambda Expressions
- App Developer Friendly
  - Common security configurations not requiring server changes
  - Annotation defaults not requiring XML

Program Agenda

JSR 375 History

- August 2014: First proposed to Oracle Java EE Architects
- December 2014: Approved by JCP
- Expert Group nominations:
  - EE API veterans: many JSRs, many years struggling with Security API
  - 3rd party security framework creators/developers
  - EE platform security implementers
- March 2015: Expert Group started discussions

JSR 375 – Expert Group

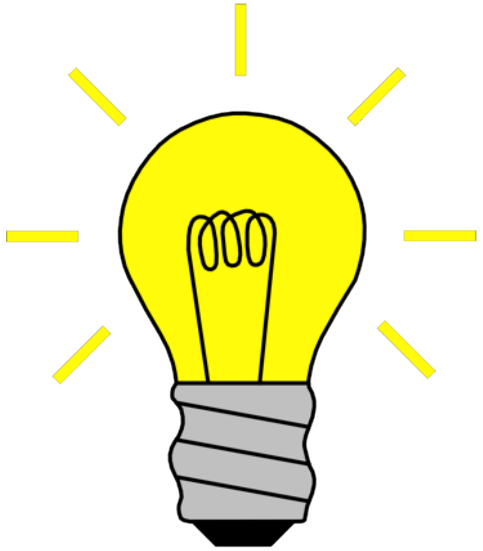
Name	Representing
Adam Bien	Individual
David Blevins	Tomitribe
Rudy De Busscher	Individual
Ivar Grimstad	Individual
Les Hazlewood	Stormpath, Inc.
Will Hopkins	Oracle
Werner Keil	Individual

JSR 375 – Expert Group

Name	Representing
Matt Konda	Jemurai
Alex Kosowski	Oracle
Darran Lofthouse	RedHat
Jean-Louis Monteiro	Tomitribe
Ajay Reddy	IBM
Pedro Igor Silva	RedHat
Arjan Tijms	ZEEF

JSR 375 Expert Group

- In first month, expert group had an EXPLOSION of activity
  - Lot of Brainstorming!
  - 237 messages on EG mailing list
  - 81 commits in Github playgrounds for examples and proposals
  - 24 JIRA issues



JSR 375 Roadmap

Early Draft Review

Q4 2015

Public Review/RI β

Q1 2016

Proposed Final Draft

Q3 2016

Final Release/ RI/TCK

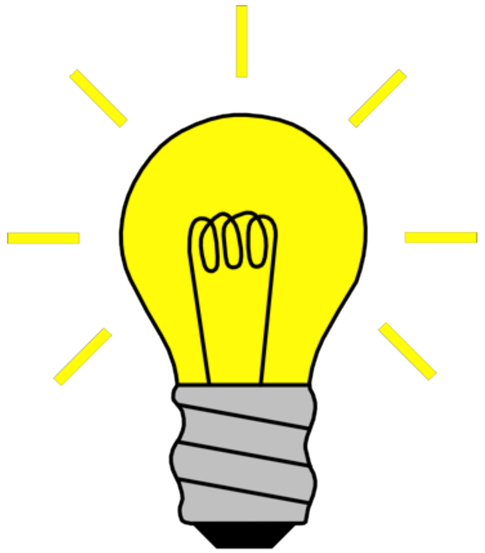
H1 2017

Program Agenda

Ideas

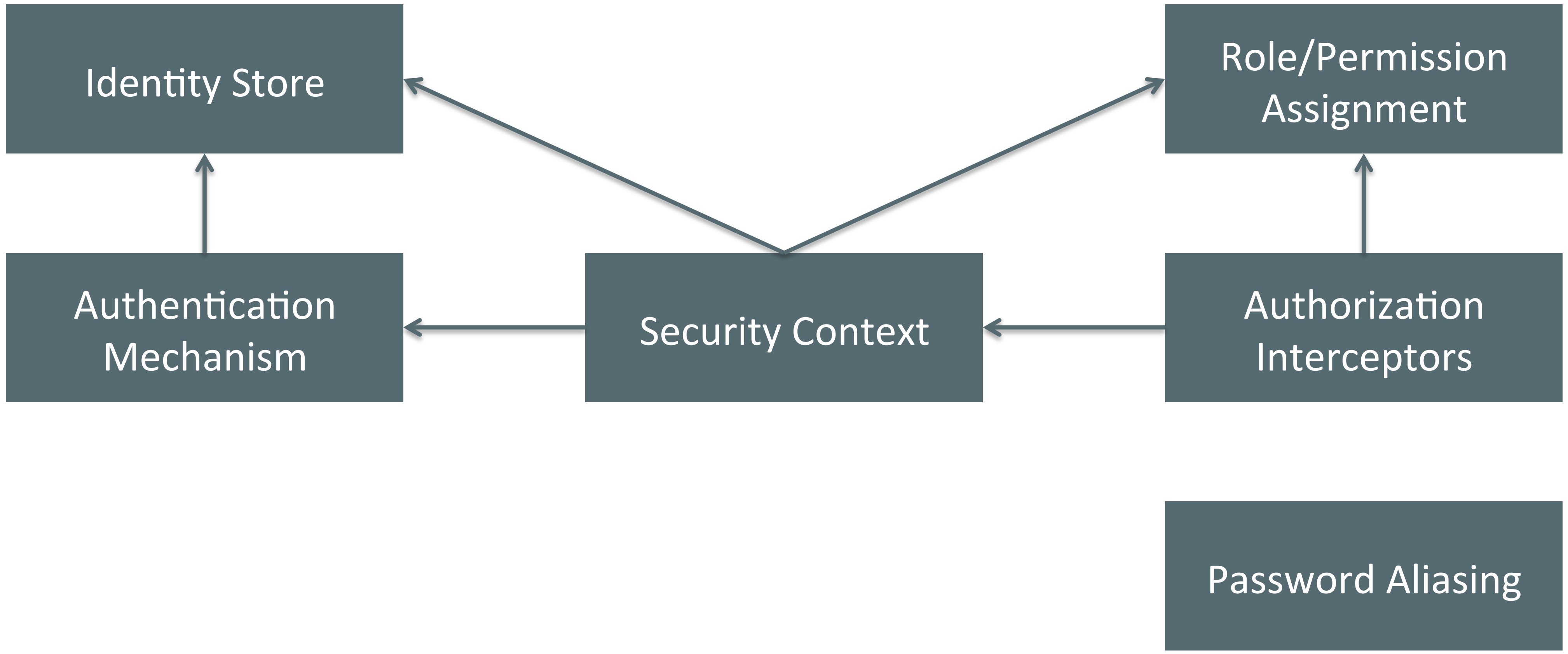
To modernize, standardize, simplify

- Terminology
- API for Authentication Mechanism
- API for Identity Store
- API for Password Aliasing
- API for Role/Permission Assignment
- API for Security Context
- API for Authorization Interceptors



Ideas

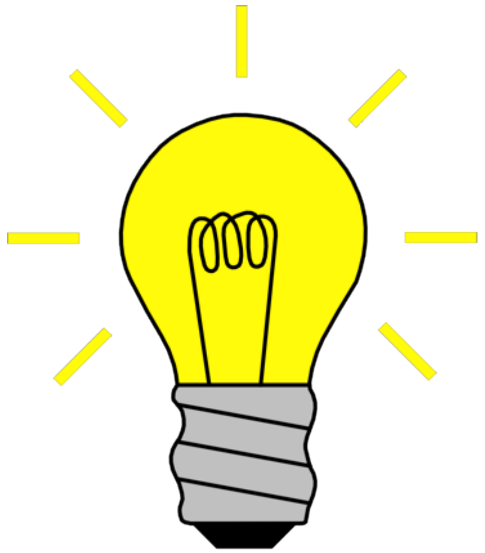
To modernize, standardize, simplify



Ideas

To modernize, standardize, simplify

- Terminology
- API for Authentication Mechanism
- API for Identity Store
- API for Password Aliasing
- API for Role/Permission Assignment
- API for Security Context
- API for Authorization Interceptors



Ideas - Terminology

- EG discussions revealed inconsistency in security API terms
- Different EE containers have different names for the same concepts
- When “something” gets authenticated, is that something a…
  - A User? (e.g. HttpServletRequest.getUserPrincipal)
  - A Caller? (e.g. EJBContext.getCallerPrincipal)
- What is a group?
  - A group of users?
  - A permission
  - Vs Role?

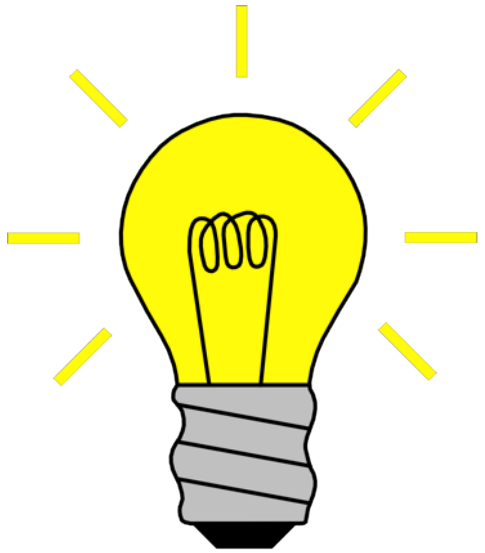
Ideas - Terminology

- What is that “something” where identities are stored?
  - security provider (WebLogic)
  - realm (Tomcat, some hints in Servlet spec)
  - (auth) repository
  - (auth) store
  - login module (JAAS)
  - identity manager (Undertow)
  - authenticator (Resin, OmniSecurity, Seam Security)
  - authentication provider (Spring Security)
  - identity provider

Ideas

To modernize, standardize, simplify

- Terminology
- API for Authentication Mechanism
- API for Identity Store
- API for Password Aliasing
- API for Role/Permission Assignment
- API for Security Context
- API for Authorization Interceptors



Use Case

API for Authentication Mechanism

- Application manages its own users and groups
- Application needs to authenticate users in order to assign Roles
- Application authenticates based on application-domain models
- Application needs to use an authentication method not supported on the server, like OpenID Connect
- Developer wants to use portable EE Authentication standard

Current Solutions

API for Authentication Mechanism

- Proprietary server support
- 3rd party security frameworks provide authentication
- JASPIC: Java Authentication Service Provider Interface for Containers

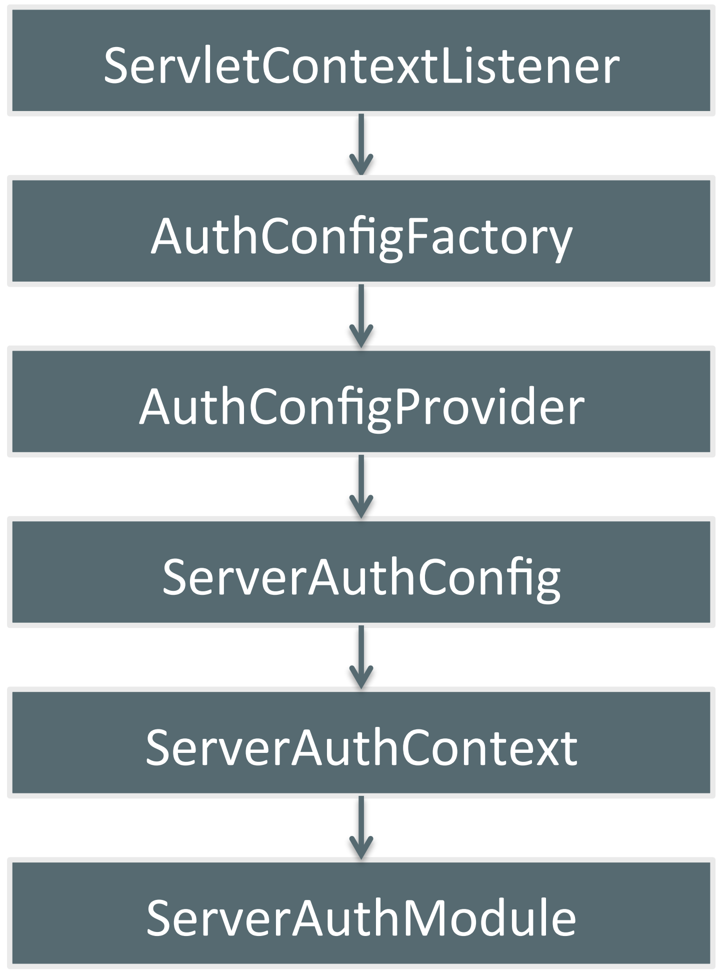
JASPIC

- Java Authentication Service Provider Interface for Containers
- JSR 196, Maintenance Release 1.1, in 2013
- Standardized, portable, thin, low-level authentication framework
- Extensible from within an application
- Integrates with the container to build an authenticated Subject
- Implement most authentication methods

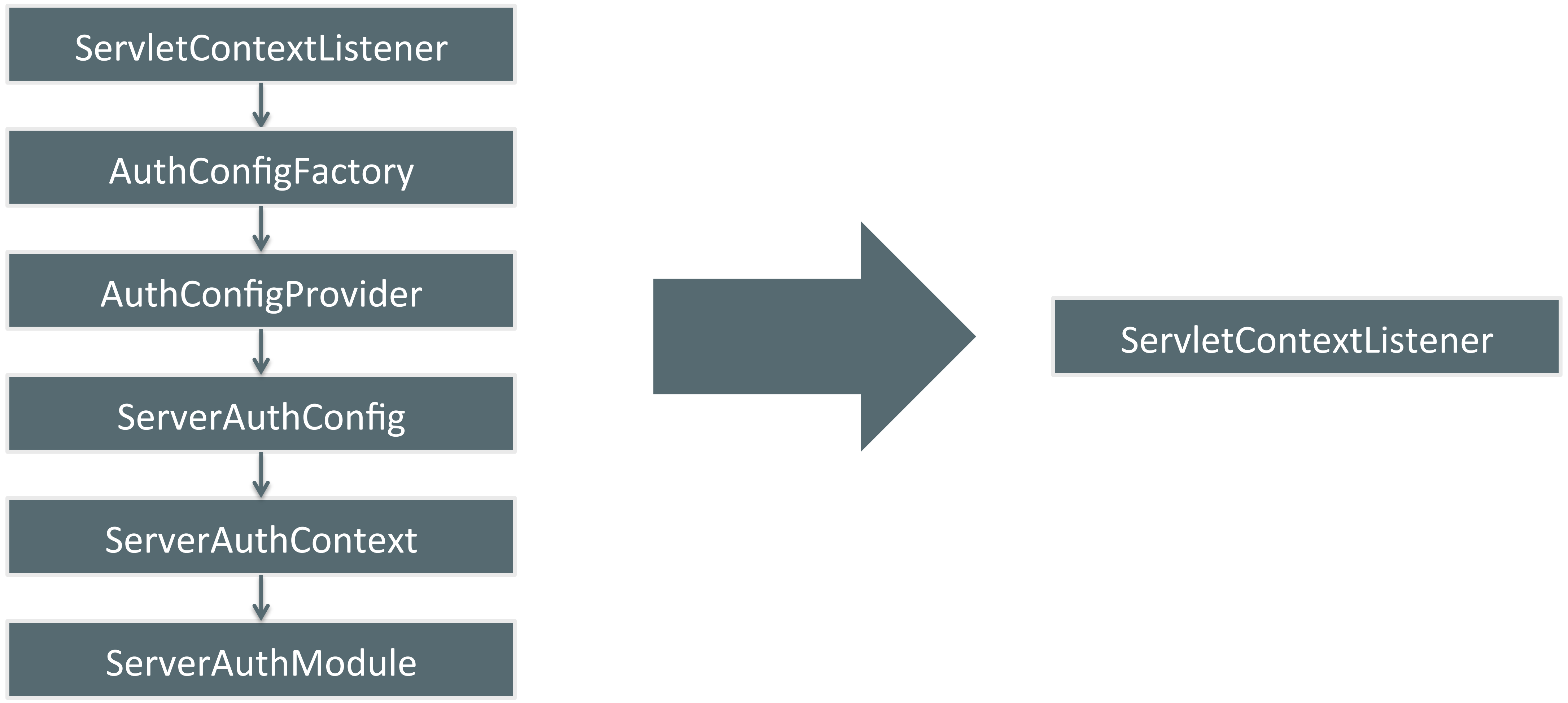
JASPIC Server Auth Module

```
public interface ServerAuthModule {

  public void initialize(MessagePolicy requestPolicy,
    MessagePolicy responsePolicy, CallbackHandler handler,
    Map options) throws AuthException;

  public AuthStatus validateRequest(MessageInfo messageInfo,
    Subject clientSubject, Subject serviceSubject);

  public Class<?>[] getSupportedMessageTypes();

  public AuthStatus secureResponse(MessageInfo messageInfo,
    Subject serviceSubject);

  public void cleanSubject(MessageInfo messageInfo, Subject subject);
}
```

JASPIC Per-Application Installation



Ideas – Simple ServerAuthModule Installation



Ideas – Simple ServerAuthModule Installation

```
@WebListener
public class SamRegistrationListener implements ServletContextListener {

  @Override
  public void contextInitialized(ServletContextEvent sce) {
    Jaspic.registerServerAuthModule(new TokenAuthModule(),
      sce.getServletContext());
  }

  @Override
  public void contextDestroyed(ServletContextEvent sce) {
  }
}
```

Ideas – Simple ServerAuthModule Installation

```
@Authenticator("org.acme.TokenAuthModule")
@WebServlet("/SimpleServlet")
@ServletSecurity(@HttpConstraint(rolesAllowed = {"manager"}))
public class SimpleServlet extends HttpServlet {

  @Override
  protected void doGet(HttpServletRequest request,
    HttpServletResponse response)
      throws ServletException, IOException {
    response.getWriter().print("my GET");
  }
}
```

Ideas – Profile Specific Helper Classes

```
public class BasicServerAuthModule implements ServerAuthModule {

  public void initialize(…) throws AuthException { … }

  public Class<?>[] getSupportedMessageTypes() { … }

  public AuthStatus secureResponse(MessageInfo messageInfo,
    Subject serviceSubject) throws AuthException { … }

  public void cleanSubject(MessageInfo messageInfo, Subject subject)
    throws AuthException { … }

  public AuthStatus validateRequest(MessageInfo messageInfo,
    Subject clientSubject, Subject serviceSubject) throws AuthException {
    final HttpServletRequest request =
      (HttpServletRequest) messageInfo.getRequestMessage();
   … }
```

Ideas – Profile Specific Helper Classes

```
public class BasicServerAuthModule extends HttpServerAuthModule {

  public AuthStatus validateHttpRequest(HttpServletRequest request,
    HttpServletResponse response, HttpMessageContext httpMessageContext)
    throws AuthException {
    …
  }
}
```

Ideas – Profile Specific Helper Classes

```
public class BasicServerAuthModule extends HttpServerAuthModule {

  public AuthStatus validateHttpRequest(HttpServletRequest request,
    HttpServletResponse response, HttpMessageContext httpMessageContext)
    throws AuthException {

    final String header = request.getHeader("Authorization");
    final String[] credentials = parseCredentials(header);
    final String username = credentials[0];
    final String password = credentials[1];
    if (!"snoopy".equals(username) || !"woodst0ck".equals(password)) {
        return FAILURE;
    }    // No callbacks required!!!
    return httpMessageContext.notifyContainerAboutLogin(
      "snoopy",
      // the groups/roles of the authenticated user
      Arrays.asList("RedBaron", "JoeCool", "MansBestFriend"));
}
```

Ideas – Standardized Authenticators

- OpenID Connect ServerAuthModule

```
@Authenticator("javax.security.authenticator.OpenIDConnect")
@WebServlet("/SimpleServlet")
@ServletSecurity(@HttpConstraint(rolesAllowed = {"manager"}))
public class SimpleServlet extends HttpServlet {

  @Override
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
    response.getWriter().print("my GET");
  }
}
```

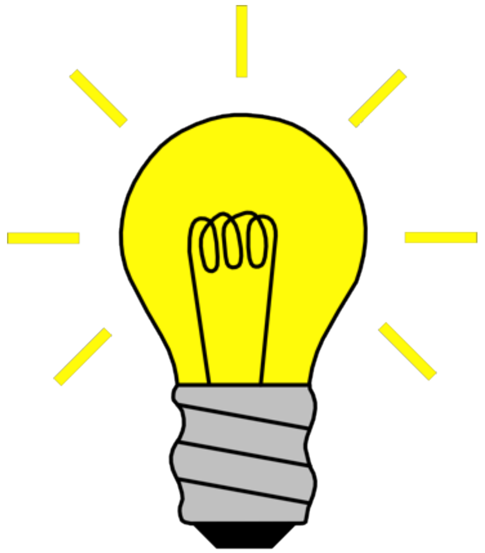
Ideas – Authentication Events

- Throw standardized CDI events at important moments
  - PreAuthenticate Event
  - PostAuthenticate Event
  - PreLogout Event
  - PostLogout Event
- Possible uses:
  - Tracking number of logged-in users
  - Tracking failed login attempts per account
  - Side effects, like creating a new local user after initial successful authentication via a remote authentication provider
  - Loading application-specific user preferences

Ideas

To modernize, standardize, simplify

- Terminology
- API for Authentication Mechanism
- API for Identity Store
- API for Password Aliasing
- API for Role/Permission Assignment
- API for Security Context
- API for Authorization Interceptors



Use Case

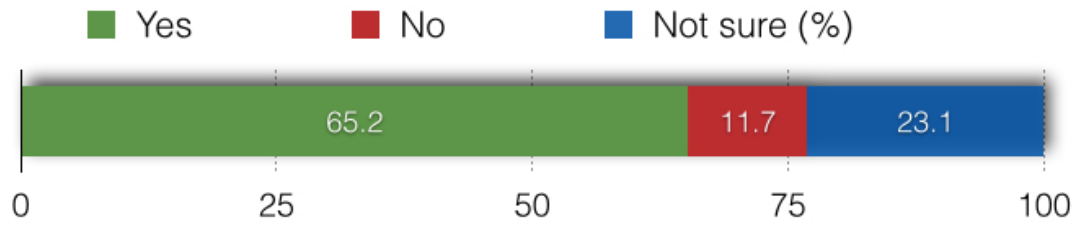
API for Identity Store

- Application manages its own users and groups
- Need to access a repository of identities, like users
- Users may be stored in app-specified repository (e.g. LDAP)
- Users are managed without access to server configuration

Survey Results

API for Identity Store

Should we standardize on requirements for simple security providers and their configuration?



Current Solutions

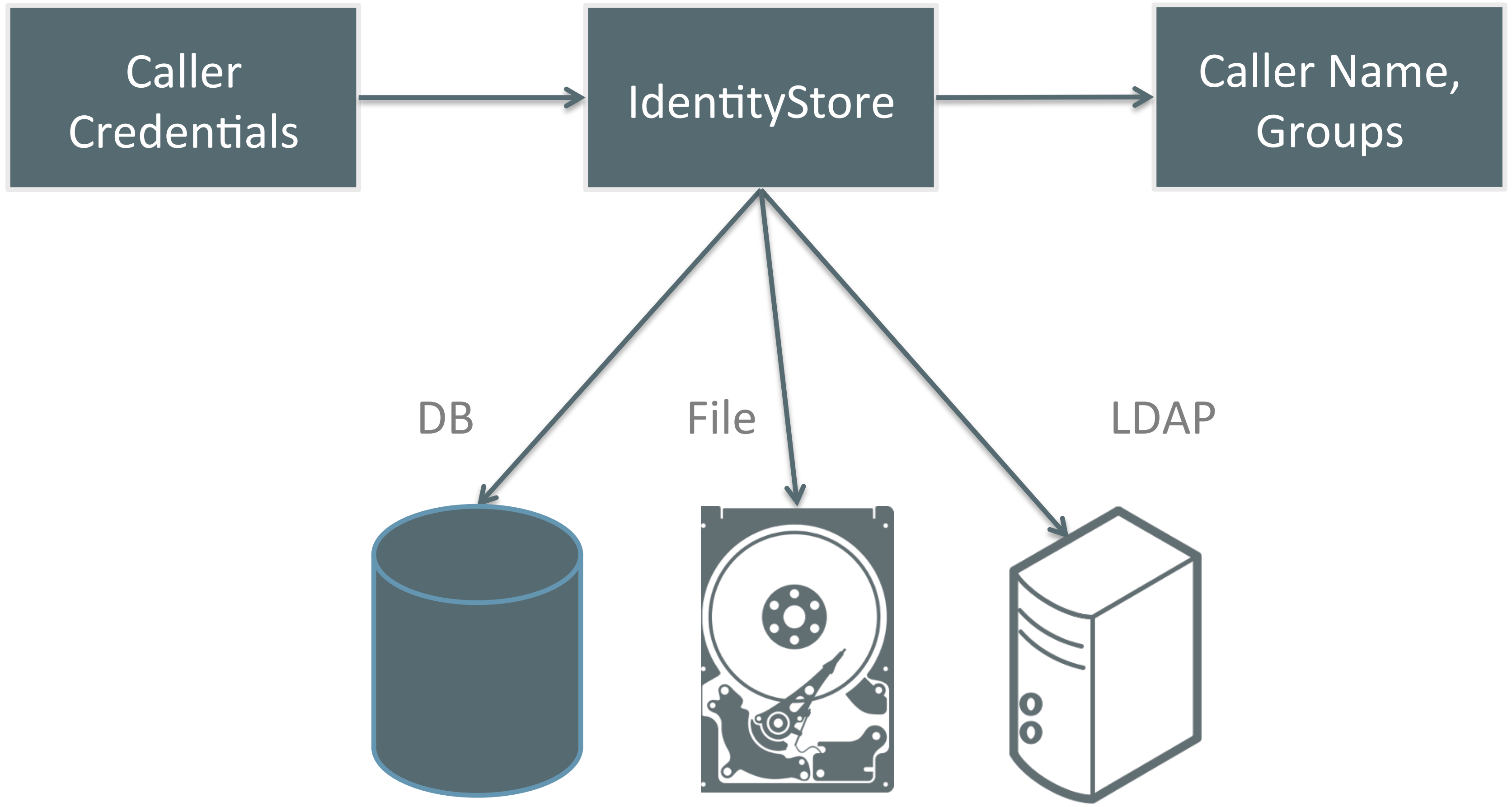
API for Identity Store

No Java EE support

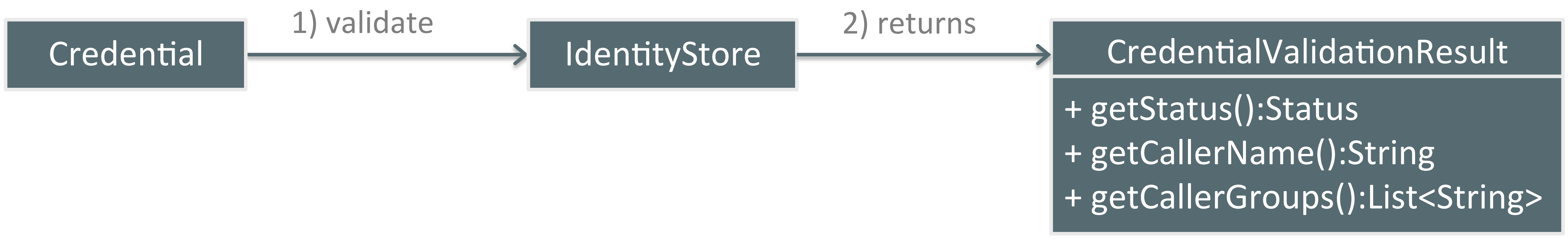
Only proprietary server support

3rd party security frameworks provide user/group APIs

Ideas – Identity Store



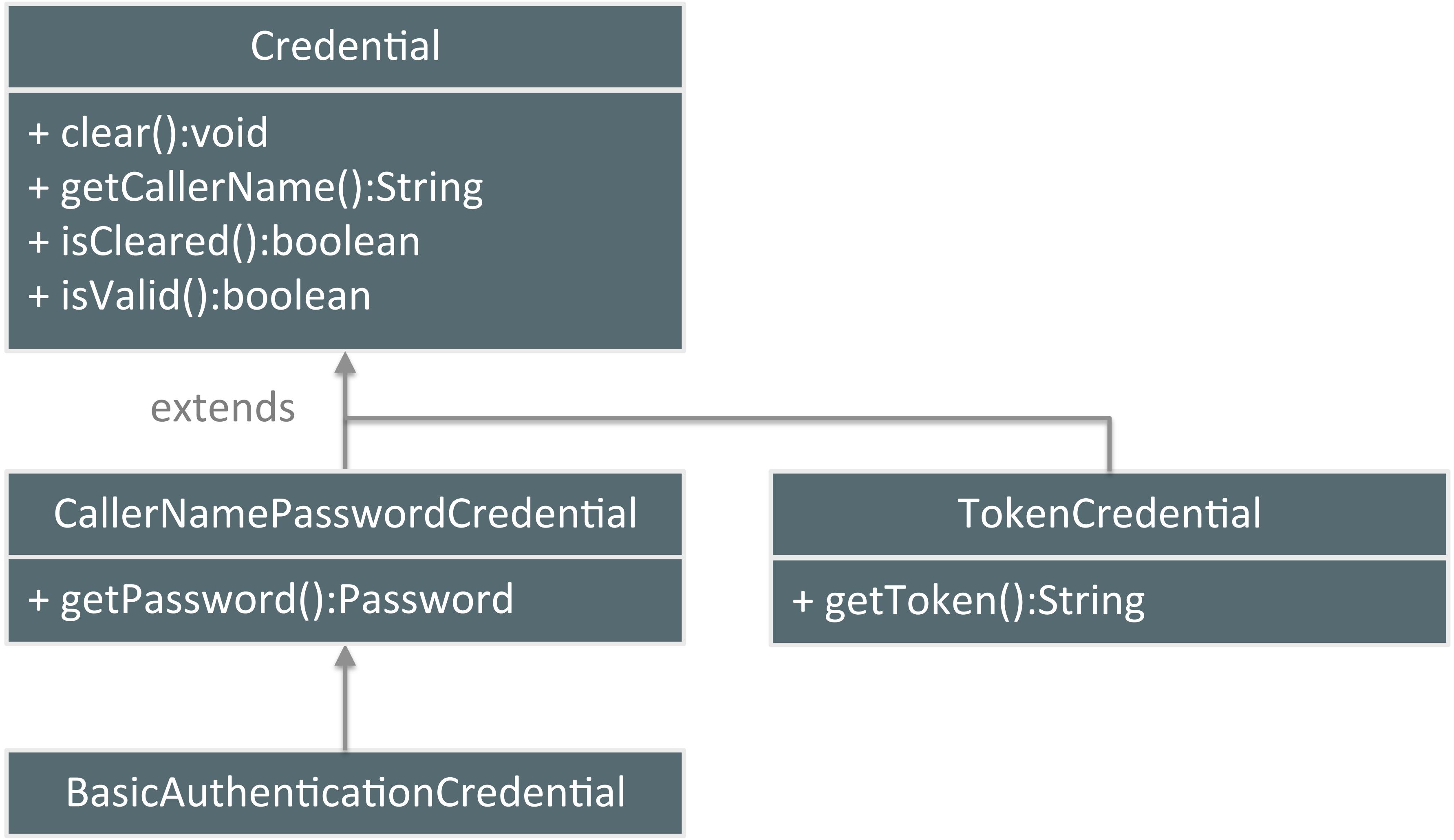
Ideas – Identity Store Interaction



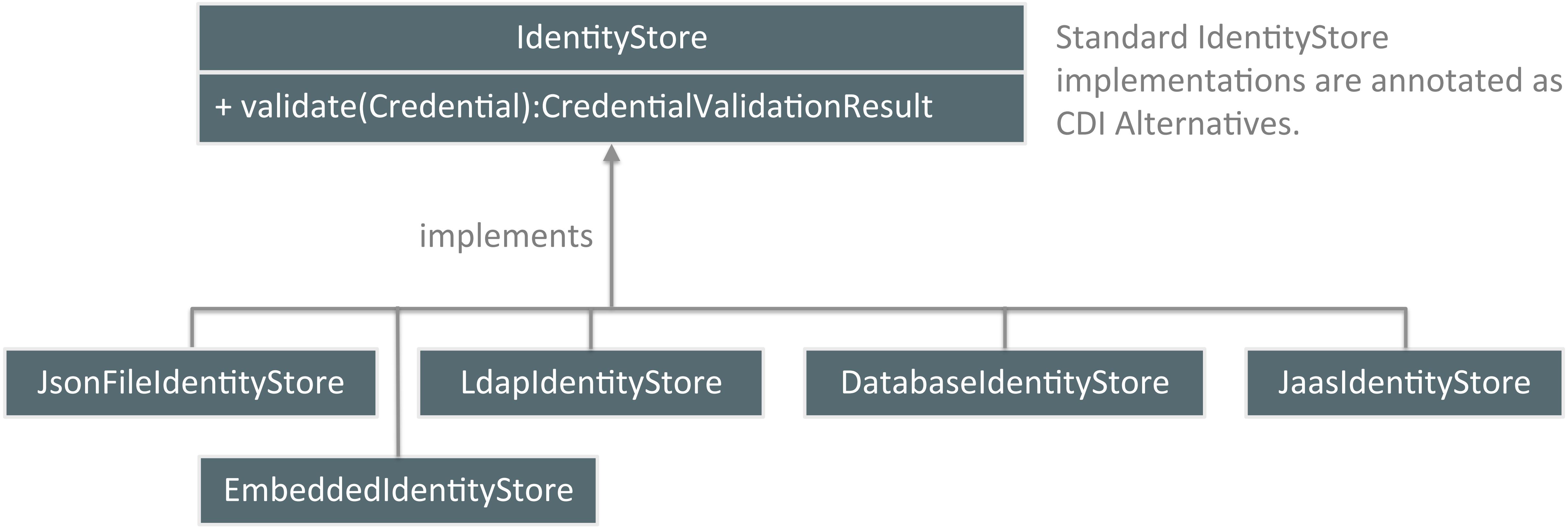
Ideas – Identity Store

```
@Inject
IdentityStore store;

CallerNamePasswordCredentials creds;
creds = new CallerNamePasswordCredentials("scott","password".toCharArray())

CredentialValidationResult result = store.validate(creds);

if (VALID.equals(result.getStatus()) {
    // successful validation
    String callerName = result.getCallerName();
    List<String> groups = result.getCallerGroups();

    // TODO: Apply to container using JASPIC callback handler
 } else {
   // invalid credential
 }
```

Ideas – Identity Store Credentials



Ideas – Identity Store Standard Implementations



Standard IdentityStore implementations are annotated as CDI Alternatives.

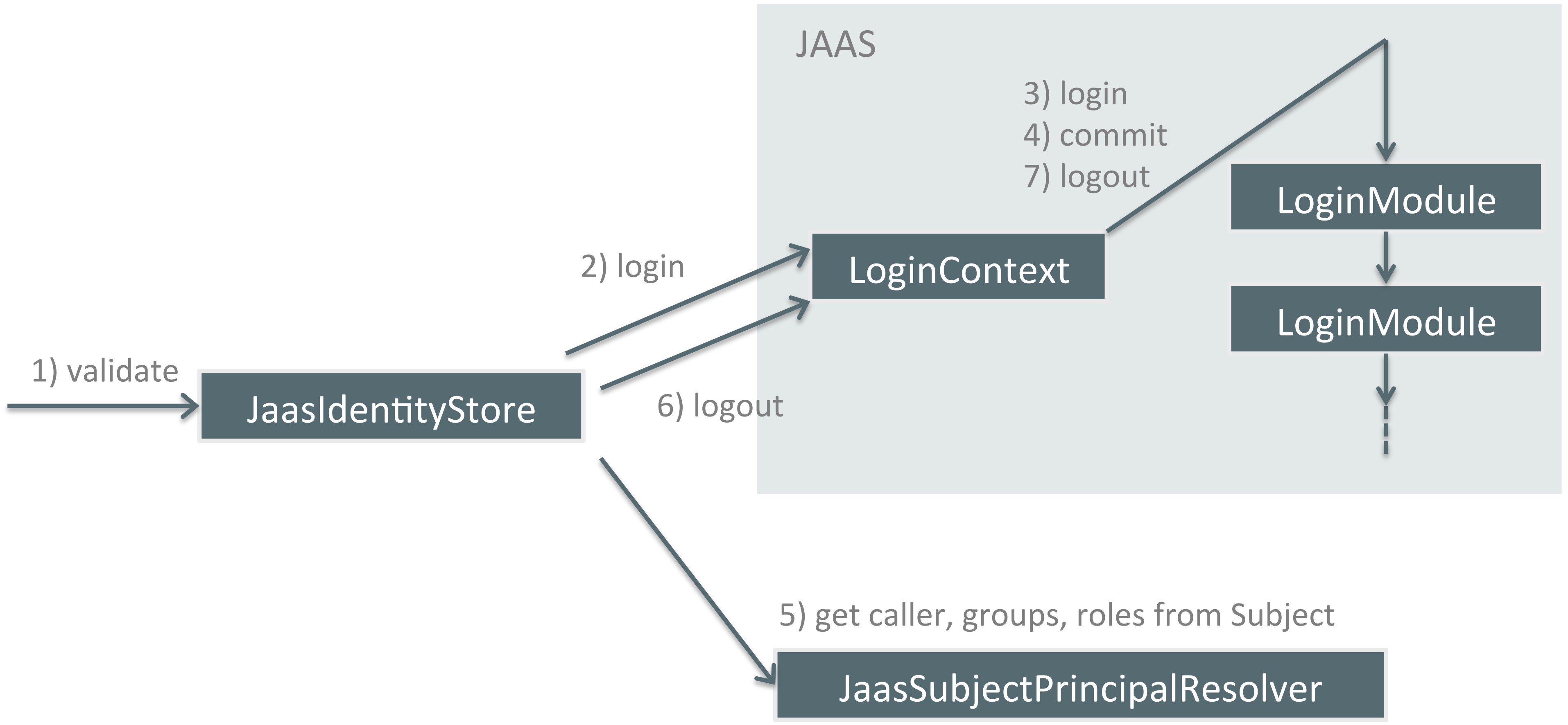
Ideas – Identity Store

```
@LdapIdentityStoreDefinition(
    url="ldap://localhost:10389",
    searchDn="uid=jsr375,dc=simple,dc=jsr375,dc=org"
    searchCredential="secret"
)
public class SomeClass{}


@JsonIdentityStoreDefinition(
    filePath="/idstore.json")
public class SomeClass{}
```

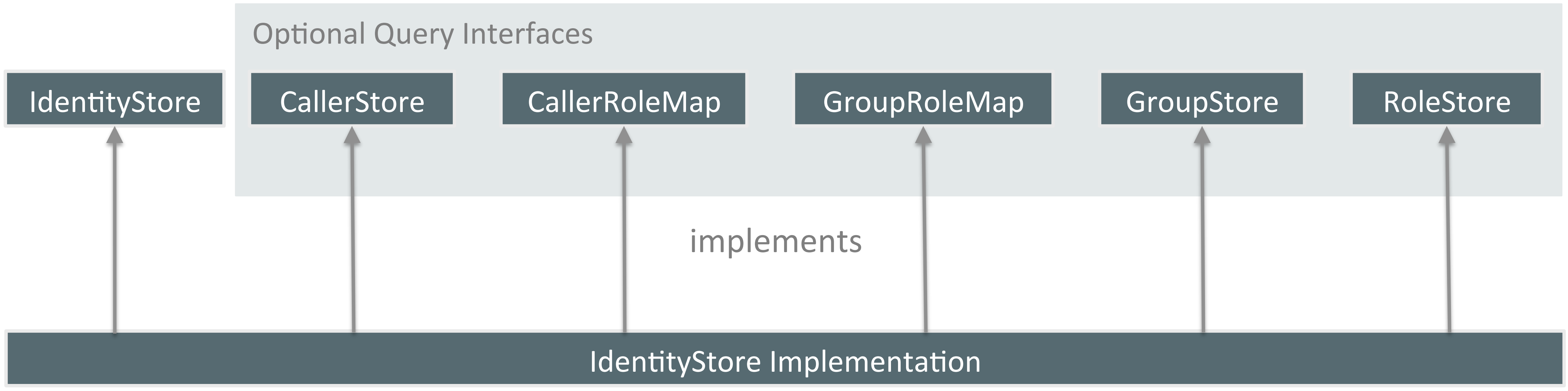
Ideas – Identity Store

```
@EmbeddedIdentityStoreDefinition({
    @Credentials(callerName = "reza", password = "secret1", groups = { "foo", "bar" }),
    @Credentials(callerName = "alex", password = "secret2", groups = { "foo", "kaz" }),
    @Credentials(callerName = "arjan", password = "secret3", groups = { "foo" }) })
public class SomeClass{}


@DataBaseIdentityStoreDefinition(
    dataSourceLookup="java:/app/myDS",
    callerQuery="select password from caller where name = ?",
    groupsQuery="select group from caller_groups where caller_name = ?",
    hashAlgorithm="SHA-256",
    hashEncoding="base64"
)
public class SomeClass{}
```

Ideas – Identity Store Standard Implementations



Ideas – Identity Store Optional Interfaces



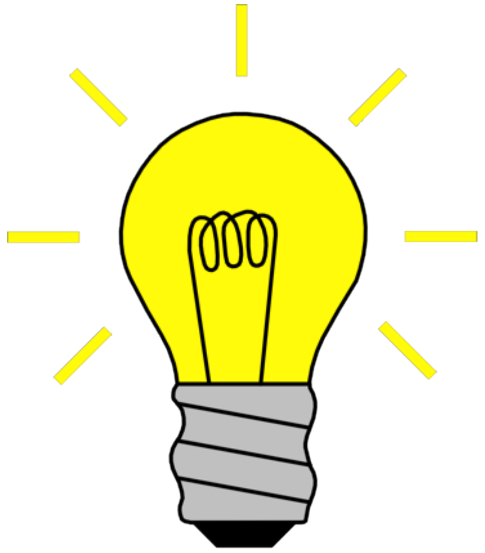
Ideas – Identity Store Optional Interfaces

```
@Inject
IdentityStore idStore;


List<String> callers = idStore.getCallers("smith");

List<String> groups = idStore.getGroups("*");

boolean inGroup = idStore.isCallerInGroup ("jsmith", "Manager");
```

Ideas

To modernize, standardize, simplify

- Terminology
- API for Authentication Mechanism
- API for Identity Store
- API for Password Aliasing
- API for Role/Permission Assignment
- API for Security Context
- API for Authorization Interceptors



Use Case

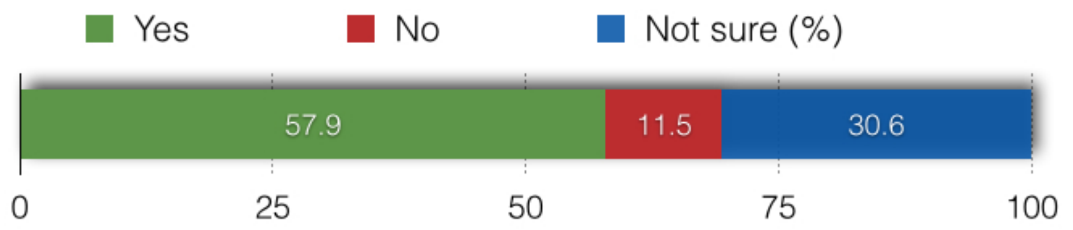
API for Password Aliasing

- Application uses passwords to access resources like LDAP and DB
- Passwords stored in annotations, deployment descriptors
- Best practices dictate that passwords are never stored in clear text
- Need a portable way to protect stored passwords

Survey Results

API for Password Aliasing

Should we add support for password aliases (including the ability to provision credentials along with the application)?



- Deferred from Java EE 7

Current Solutions

API for Password Aliasing

- No Java EE support
- Proprietary server support, e.g. GlassFish
- 3rd party security framework support for embedded password encryption, not aliasing

Ideas – Password Aliasing

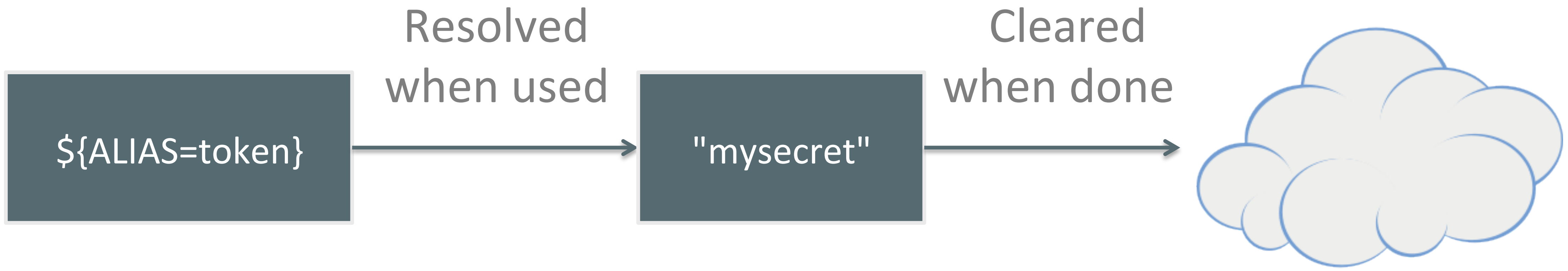
- For annotations

```
@DataSourceDefinition(
 name="java:app/jdbc/test",
 user="root",
 password="${ALIAS=password}",…)
```

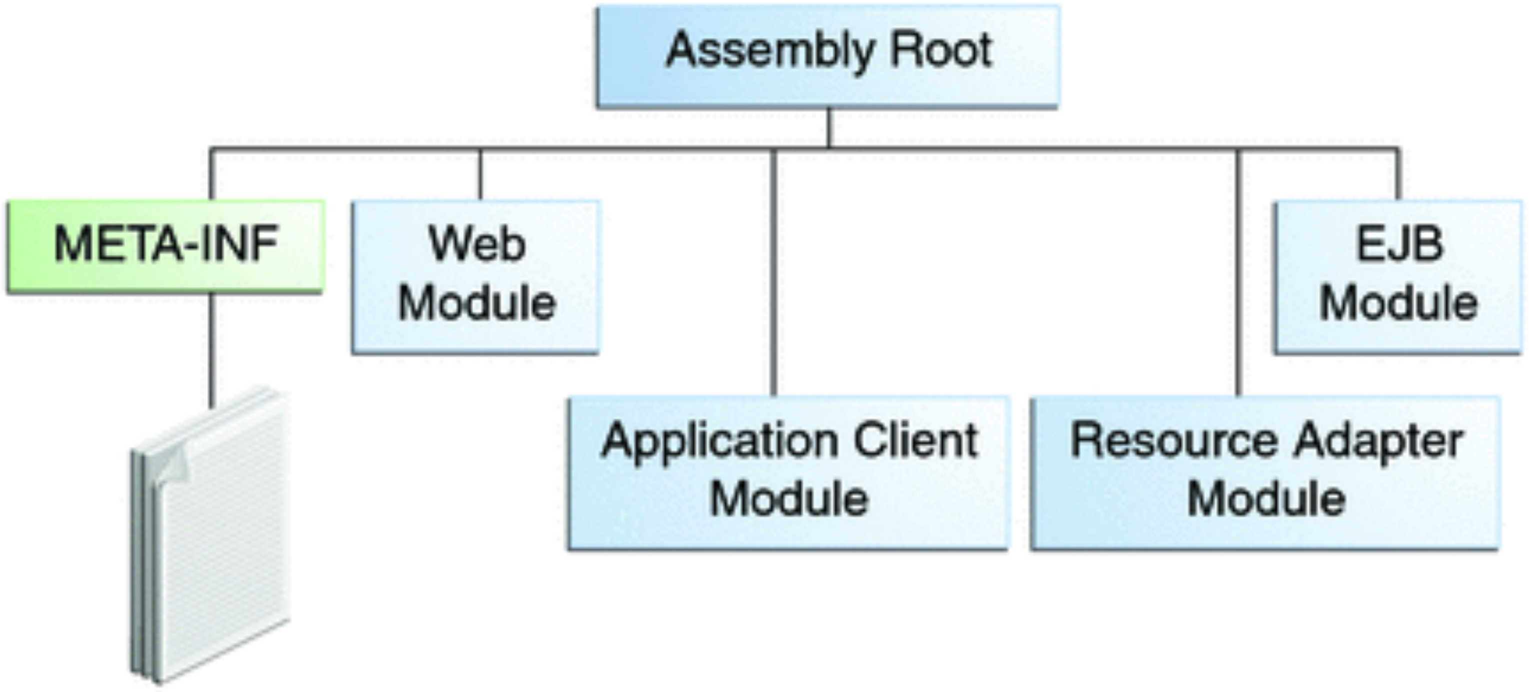
- For deployment descriptors

```
<data-source>
<name>java:app/env/testDS</name>
<user>APP</user>
<password>${ALIAS=password}</password>
…
</data-source>
```

Ideas – Password Aliasing



Ideas – Password Aliasing



Password Alias Archive

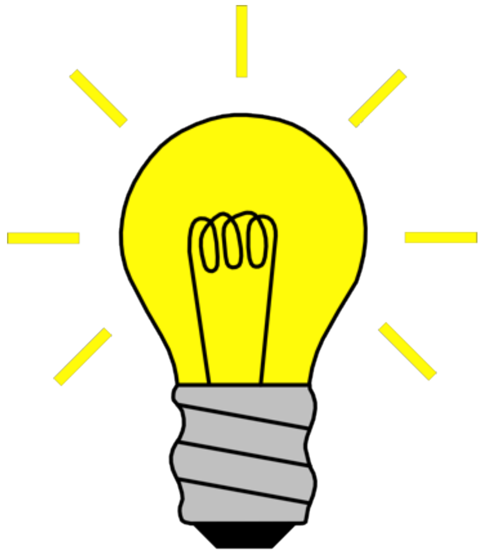
Ideas – Password Aliasing

- For configuration: Annotations, Deployment Descriptors
- Secure credentials archive for bundling the alias and actual password values with applications
- Platform consumes the credentials archive upon deployment
- Standard tooling for CRUD operations on the credential archive, e.g. keytool

Ideas

To modernize, standardize, simplify

- Terminology
- API for Authentication Mechanism
- API for Identity Store
- API for Password Aliasing
- API for Role/Permission Assignment
- API for Security Context
- API for Authorization Interceptors



Use Case

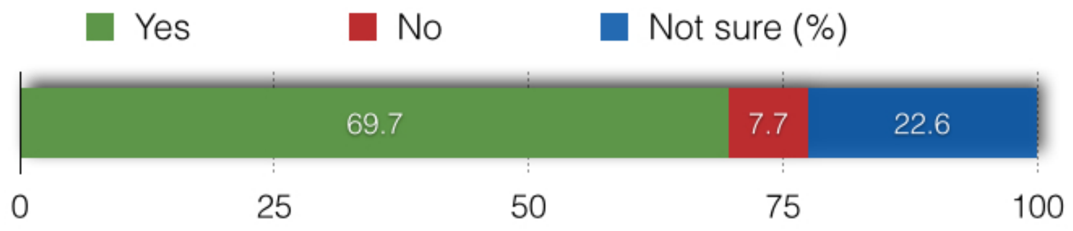
API for Role/Permission Assignment

- Application manages its own users and groups
- Application needs to assign roles (i.e., authorities, permissions) to users and groups, based on application-specific model
- Users may be stored in app-specified repository (e.g. LDAP)
- Users are managed without access to server configuration

Survey Results

API for Role/Permission Assignment

Should we standardize group-to-role mapping?



Current Solutions

API for Role/Permission Assignment

- No Java EE support
- Only proprietary server support
- 3rd party security frameworks provide role/authority/permission APIs

Ideas – Standardized Role Mapping

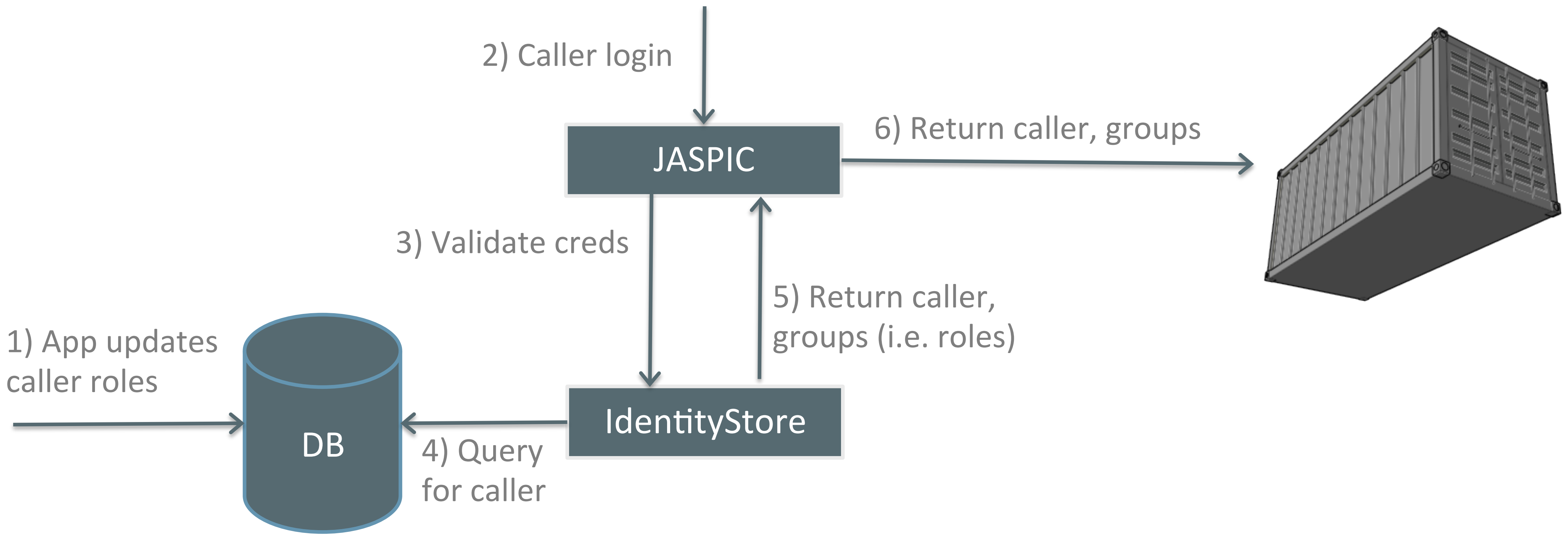
- Support in Deployment Descriptors, e.g. web.xml

```
<security-role-map>
    <!-- Role name as set/returned by Authentication Module -->
    <group>MANAGER</group>

    <!-- Role name for mapping -->
    <role-name>EDIT_ACCOUNTS</role-name>
</security-role-map>


<!– One-to-one group to role mapping -->
<security-role-map groupToRoleMapping="false" />
```

Ideas – Role Mapping Annotation

```
@EmbeddedIdentityStoreDefinition({
    @Credentials(callerName = "reza", password = "secret1", groups = { "foo", "bar" }),
    @Credentials(callerName = "alex", password = "secret2", groups = { "foo", "kaz" }),
    @Credentials(callerName = "arjan", password = "secret3", groups = { "foo" }) })
public class MyServlet {
}
```

Ideas – Dynamic Application-based Roles

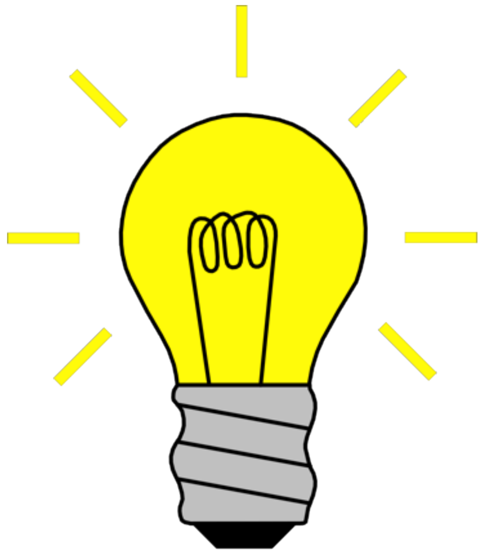


Assuming 1:1 Group-Role Mapping

Ideas

To modernize, standardize, simplify

- Terminology
- API for Authentication Mechanism
- API for Identity Store
- API for Password Aliasing
- API for Role/Permission Assignment
- API for Security Context
- API for Authorization Interceptors



Use Case

API for Security Context

- Application needs to access the security API
- To get the authenticated user
- To check roles
- To invoke runAs.
- Application needs the same API to access security context, regardless of container

Current Solutions

API for Security Context

- No Java EE support
- 3rd party security frameworks provide a security context

Current Solutions

```
@Singleton
public class MyEjb {
    @Resource
    private SessionContext sessionContext;

    public String sayHello() {
        if (sessionContext.isCallerInRole("admin")) {
            return "Hello World!";
        }
        throw new SecurityException("User is unauthorized.");
    }
}
```

Current Solutions

```
public class MyServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request,
        HttpServletResponse resp) throws ServletException, IOException {

        if (request.isUserInRole("admin")) {
            // do something
        }
        throw new ServletException("User is unauthorized.");
    }
}
```

Current Solutions

```
@RequestScoped
public class MyCdiBean {

    // Oh snap!  No SecurityContext class for CDI
}
```

Current Solutions

```
public class MyJaxRsService {
    @GET
    @Produces("text/plain;charset=UTF-8")
    @Path("/hello")
    public String sayHello(@Context SecurityContext sc) {

        if (sc.isUserInRole("admin")) {
            return "Hello World!";
        }
        throw new SecurityException("User is unauthorized.");
    }
}
```

Ideas – Security Context

```
public interface SecurityContext {
    String getUserPrincipal();
    boolean isUserInRole(String role);
    List<String> getAllUsersRoles();
    boolean isAuthenticated();
    boolean isUserInAnyRole(List<String> roles);
    boolean isUserInAllRoles(List<String> roles);
    void login(Object request, Object response);
    void login(Map map);
    void logout();
    void runAs(String role);
    boolean hasAccessToResource();
    boolean hasAccessToBeanMethod();
}
```

Ideas – Security Context

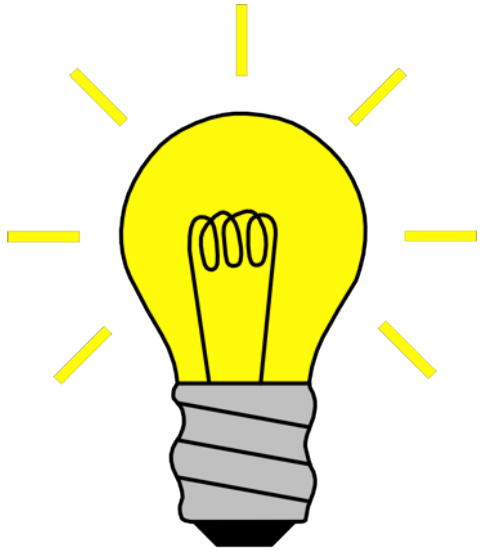
- For all managed beans: CDI, Servlet, EJB, JAX-RS, etc

```
public class MyFutureCdiBean {

    @Inject
    private SecurityContext securityContext;

    public String sayHello() {

        if (securityContext.isUserInRole("admin")) {
            return "Hello World!";
        }

        throw new SecurityException("User is unauthorized.");
    }
}
```

Ideas

To modernize, standardize, simplify

- Terminology
- API for Authentication Mechanism
- API for Identity Store
- API for Password Aliasing
- API for Role/Permission Assignment
- API for Security Context
- API for Authorization Interceptors



Use Case

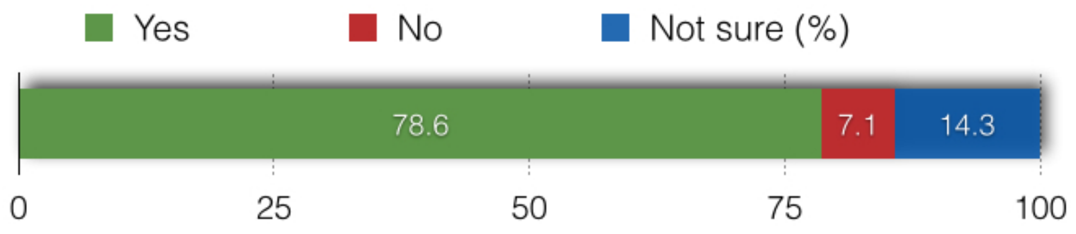
API for Authorization Interceptors

- Application needs to restrict specific methods to authorized users
- Application-model rules are used to make access decisions
- Role is insufficient

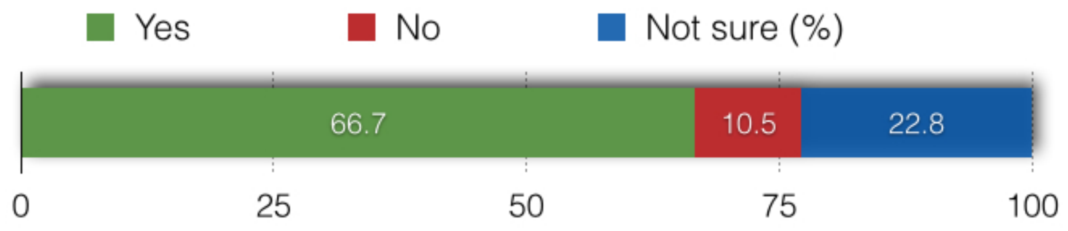
Survey Results

API for Authorization Interceptors

Should we consider adding Security Interceptors in Java EE 8?



Should we simplify authorization by introducing an EL-enabled authorization annotation?



Current Solutions

API for Authorization Interceptors

- EE authorization has no rule based authorization, only role based
- 3rd party security frameworks provide rule, role and permission based APIs

Ideas – EL Authorization Rules

- Expression Language rule would have access to managed beans for SecurityContext and InvocationContext

```
@EvaluateSecured("security.hasRoles('MANAGER') && schedule.nowIsOfficeHrs")
void transferFunds() {..};
```

Ideas – EL Authorization Rules

- EL Authorization rules centrally managed in a repository

```
@LdapAuthorizationRules (
       name="java:app/accountAuthRules",
       ldapUrl="ldap://blah",
       ldapUser="ElDap",
       ldapPassword=“mysecret”)
public class MyBean {

  @EvaluateSecured(ruleSourceName="java:app/accountAuthRules", rule="transferFunds")
  void transferFunds() {..};

  …
}
```

Ideas – AccessDecisionVoter

- A user-defined class for making access decisions

```
@Secured(AccountAccessDecisionVoter.class)
void transferFunds() {..};
```

Ideas – AccessDecisionVoter

```
public class AccountAccessDecisionVoter implements AccessDecisionVoter {

  @Override
  public SecurityViolation checkPermission(AccessDecisionVoterContext ctx) {

    // Check for violations
    Method method = ctx.<InvocationContext>getSource().getMethod();

    …
    return new SecurityViolation("Sorry, not allowed");
  }
}
```

Program Agenda

1. Motivations
2. A New JSR
3. Ideas
4. Get Involved
5. Q & A

Get Involved

Contribute to the JSR!

- Project Page: The starting point to all resources
  https://java.net/projects/javaee-security-spec

- Users List: Subscribe and contribute
  users@javaee-security-spec.java.net

- Github Playground: Fork and Play!
  https://github.com/javaee-security-spec/javaee-security-proposals

Get Involved

Attend related sessions!

- How Would You Improve the Java EE Security API? [BOF3666]
  - Tonight at 8 PM | Hilton—Plaza Room A
  - Hosted by Ivar Grimstad and Alex Kosowski

- The Java EE 8 Opportunity [CON6086]
  - Presented by David Blevins, Tomitribe
  - Wednesday, Oct 28, 4:30 PM | Parc 55—Cyril Magnin II/III

Program Agenda

Q & A

JSR 375 – EE Security API

Safe Harbor Statement

Integrated Cloud

Applications & Platform Services

JavaOne™

ORACLE®

ORACLE®