ThoughtWorks®

PATTERNS FOR DEVELOPING SECURE WEB APPLICATIONS

An Irreverent Look at How to Build Secure Software

Daniel Somerfield
Lead Consultant

WHY APPLICATION SECURITY

We are all Targets

WHY WEB APPLICATION SECURITY IS HARD

We have an increasingly sophisticated adversary.

Software has gotten very, very complicated and we move very, very fast.

Our business owners don't (want to) understand security.

The technologies we use weren't designed with security in mind

HOW TO MAKE IT (A LITTLE BIT) EASIER

We need to develop strategies, patterns and processes that help us do our jobs more effectively.

Here are a few thoughts…

ThoughtWorks®

DEVELOPMENT PATTERNS

IT'S ALL ABOUT CONTEXT

Remember the buffer overflow???

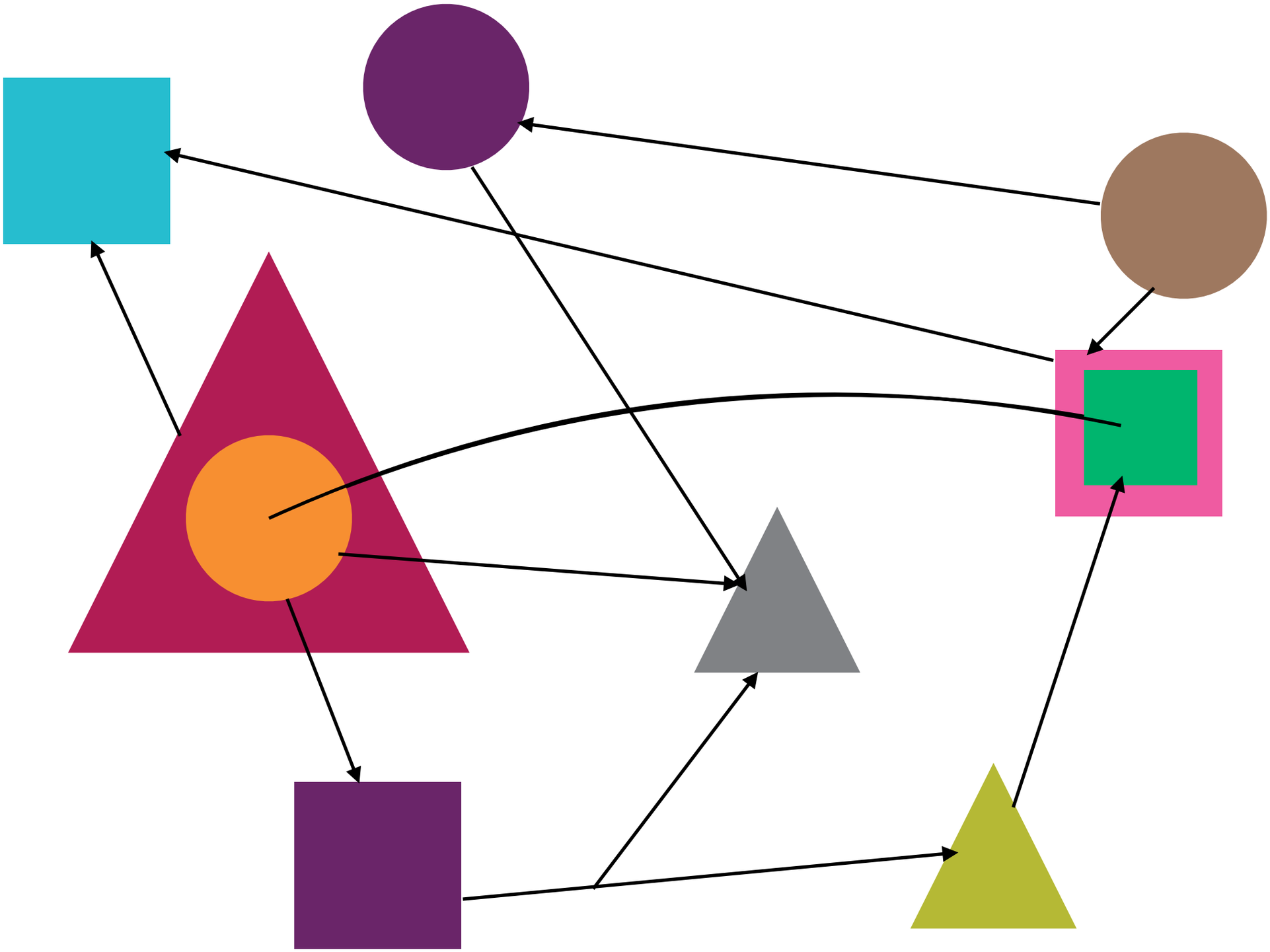
IT'S ALL ABOUT CONTEXT

But...

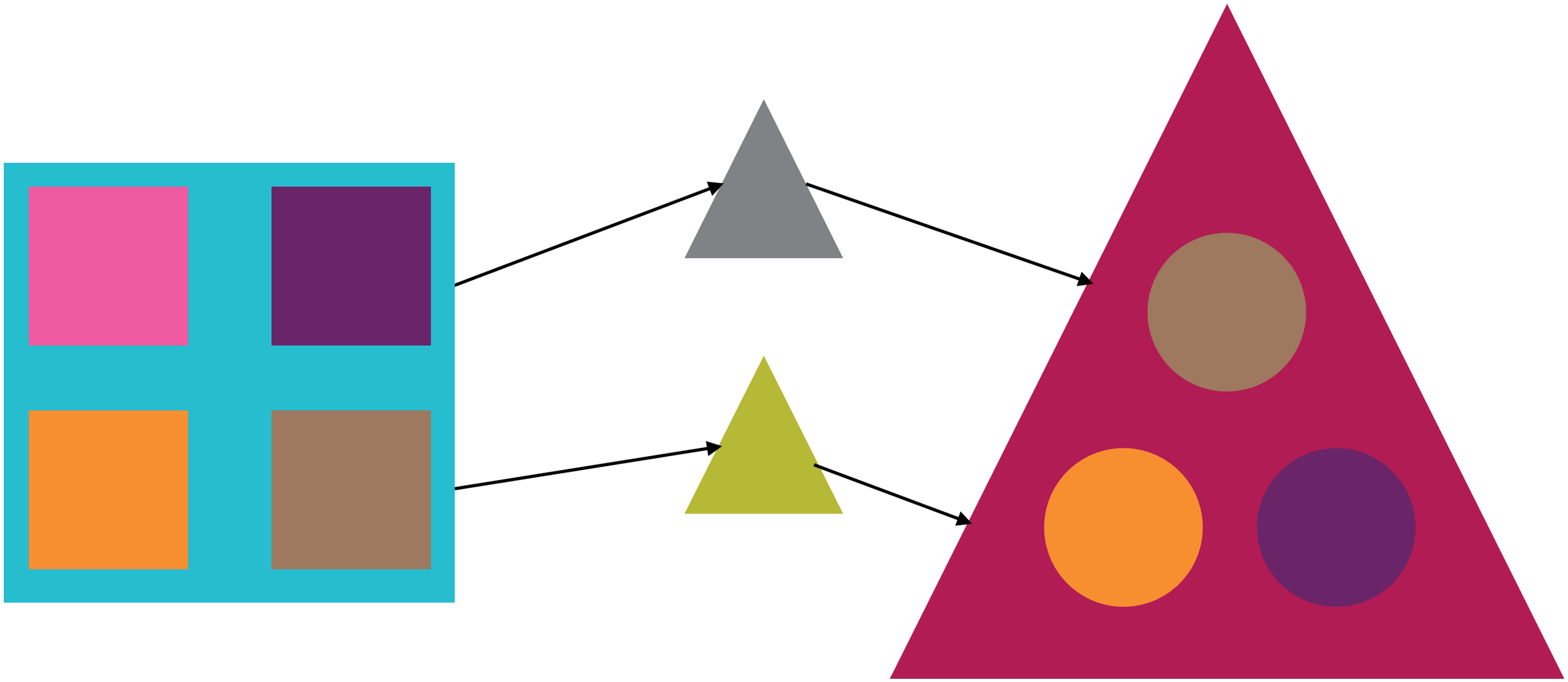
EVERYTHING OLD IS NEW AGAIN

```
<html>
…
  <style>
    div { font-weight: bold; }
  </style>
  <script>
    doSomethingDangerousWith(<%= untrusted.content %>);
  </script>
…
  <div>Hello Dangerous Internet World</div>
…
</html>
```

RISKS OF COMPLEXITY



RISKS OF COMPLEXITY



DEVELOPMENT ANTI-PATTERN #1

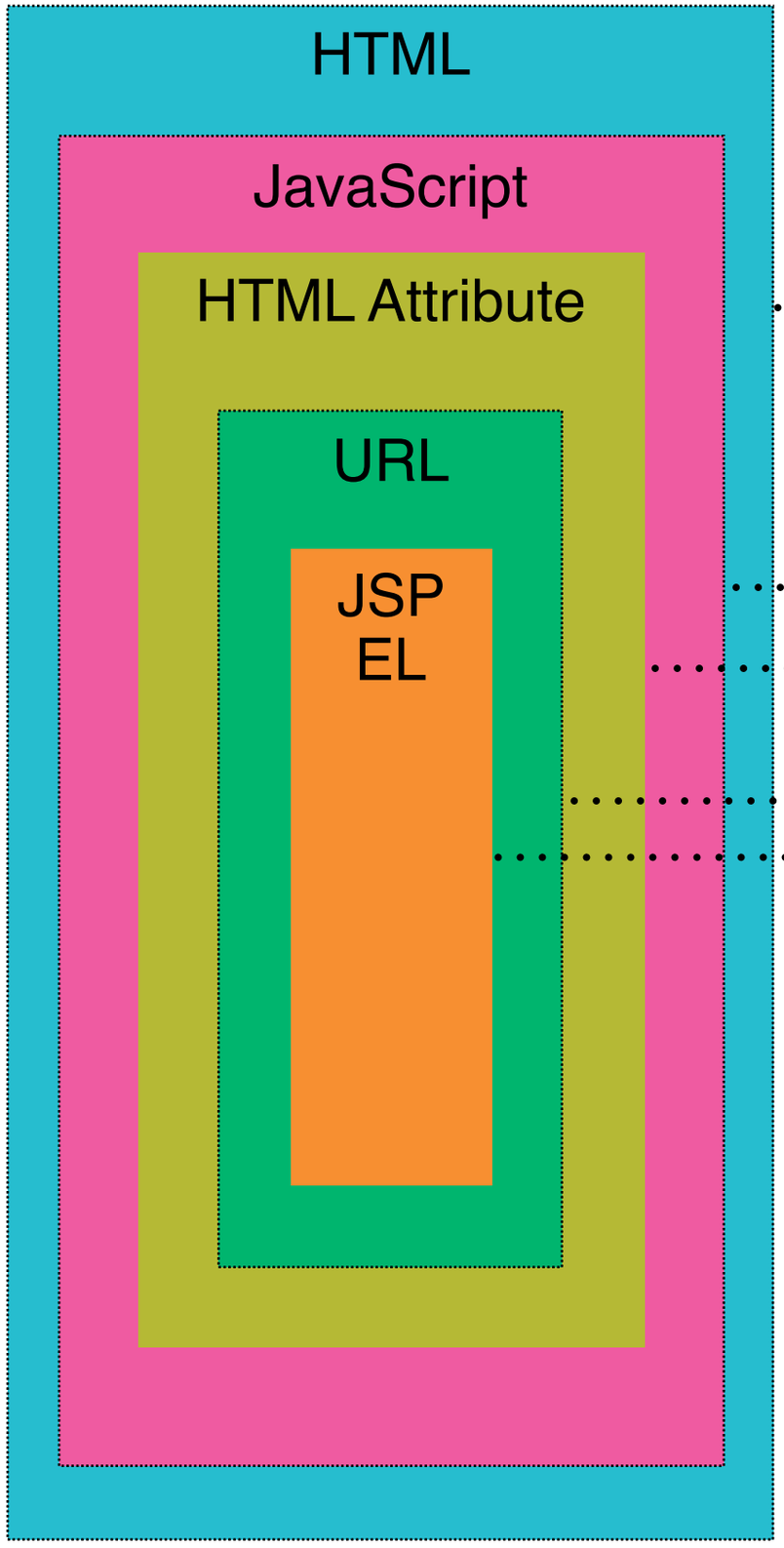
Name: The Russian Doll

AKA: Nested Contexts, Tangled Concerns

Consequences: Excessive complexity, difficult to understand encoding needs

Examples: JavaScript nested within JSP

ANTI-PATTERN: THE RUSSIAN DOLL EXPLAINED




```
…
<head>
<script>
…
$(".show-button", currentLine).html(
"<a href='/show/<%= entry.id %>' />");
…
</script>
…
</head>
<input type="button" class="show-button" />
```

CONSEQUENCES

Does it work?

Yes!

Is it secure?

Well…

DEVELOPMENT PATTERN #1

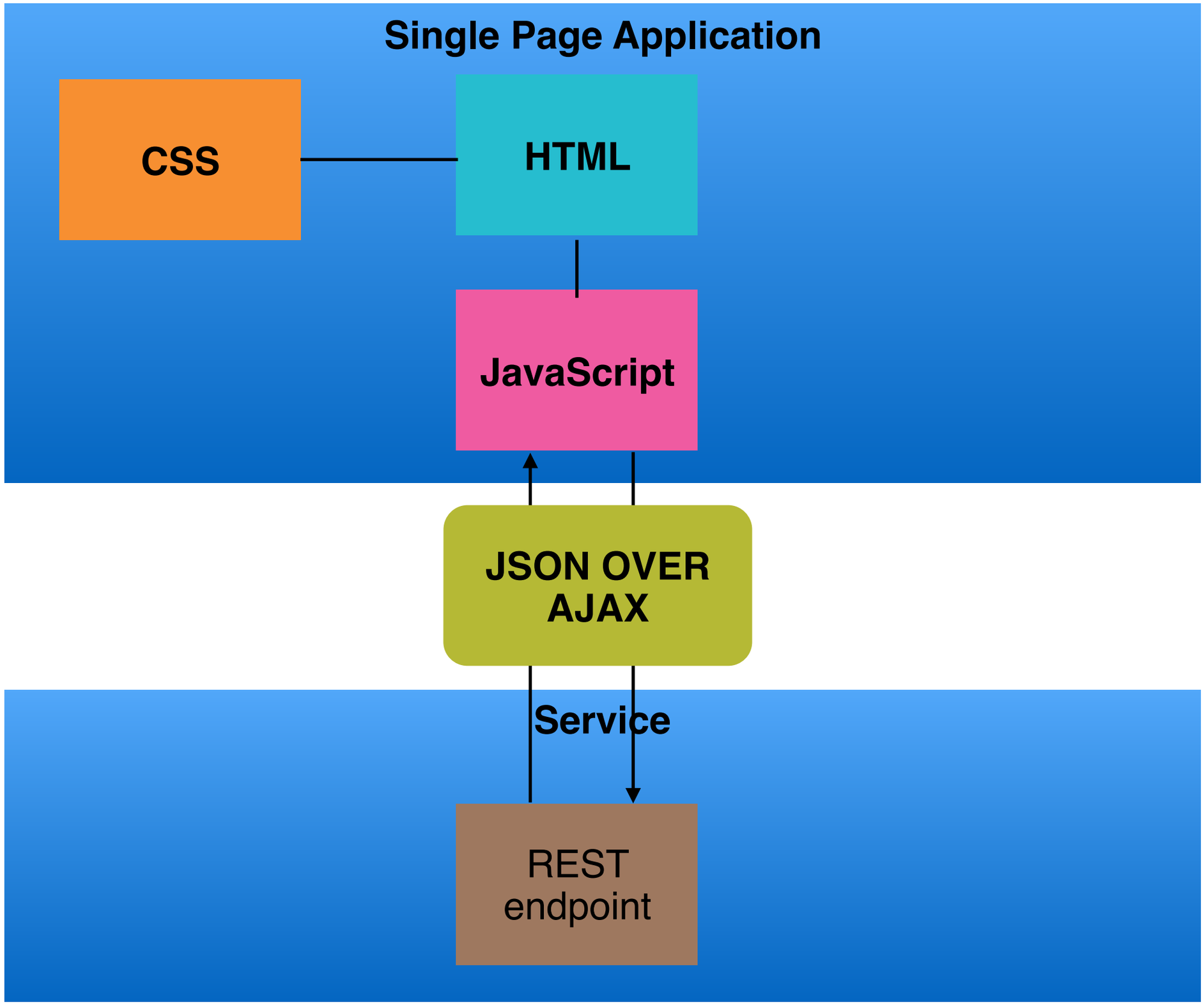
Name: The McDLT

AKA: Separating Web Client Concerns

Benefits: Easy to identify output encoding needs. More enforceable behavior.

Examples: Single Page Application with external service

MCDLT PATTERN EXPLAINED



CODE EXAMPLE: RISKS OF COMPLEXITY

https://github.com/danielsomerfield/McDLT

THE McDLT PATTERN IMPLEMENTATION

- Contexts are separated by file
- Minimizes XSS possibilities
- AJAX transport allows control of origin which minimizes risk of CSRF attacks

RISKS OF IMPRECISION

DEVELOPMENT ANTI-PATTERN #2

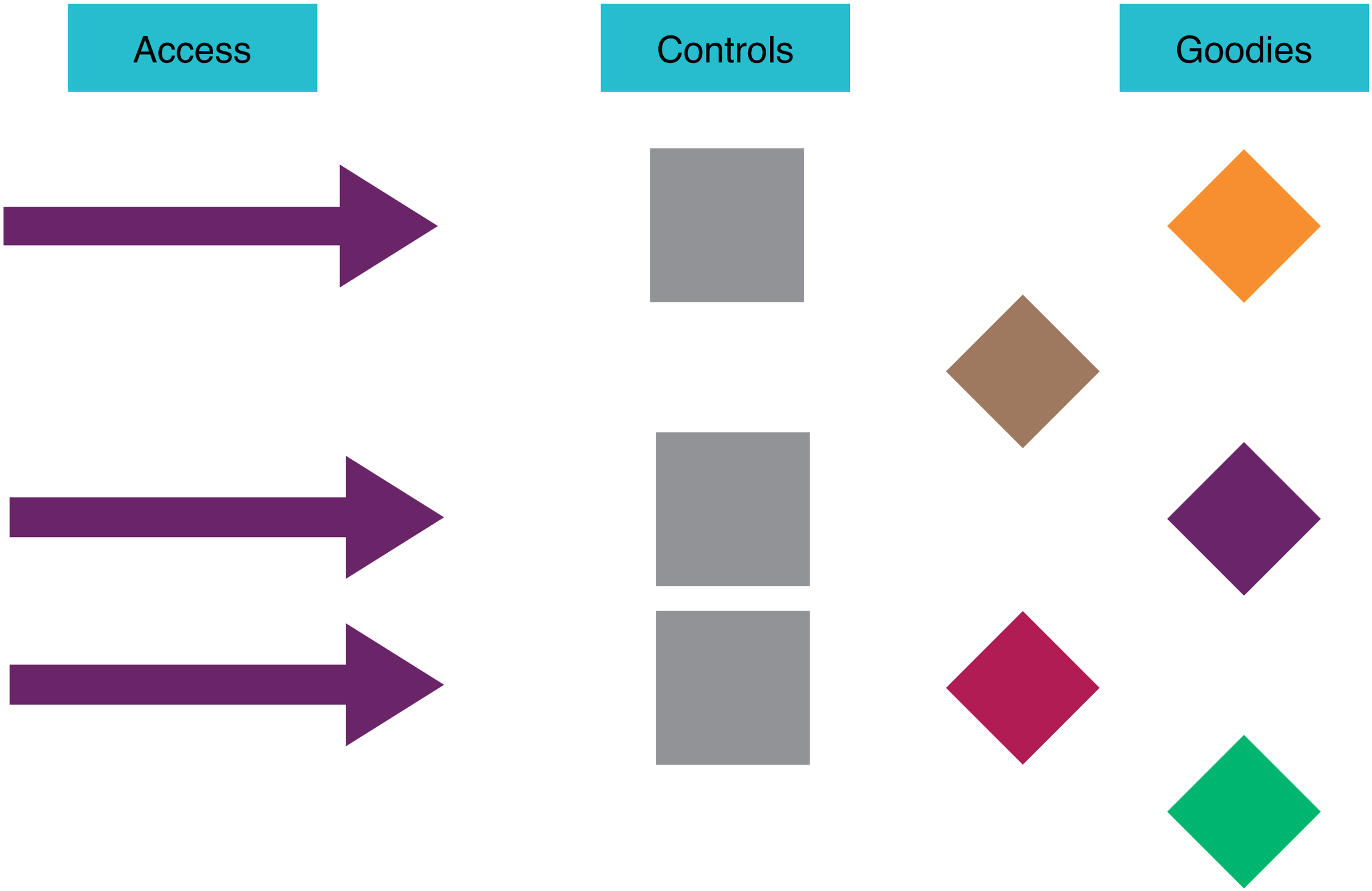
Name: Whack a Mole

AKA: Lock it down, Black-listing

Consequences: Overly permissive endpoints or rule sets. Poorly tested security assertions.

Examples: Piecemeal security filters. End-point black-listing.

DEVELOPMENT ANTI-PATTERN #2: WHACK-A-MOLE



CODE EXAMPLE: RISKS OF IMPRECISION

https://github.com/danielsomerfield/ImprecisionPatterns.git

ACL API: ROLE SET UP

```
//Create Roles
public static Role ROLE_1 = new Role(new
    Permissions());
public static Role ROLE_2 = new Role(
    new Permissions(
        new Permission(READ, Optional.of("/resource/
path"))
    )
);

//Create a Principal
Principal principal = new Principal(Roles.ANONYMOUS);
```

ACL API: SECURITY CONFIGURATION

```
//Build ACL
ACL acl =
ACL.acl().withRootPermissions(emptySet())
    .resource("/resource")
      .requiresPermissionFor(READ)
    .resource("/resource/subresource")
    .build();
```

ACL API: VALIDATION

```
//Check Access
acl.checkIf(principal)
    .can(PermissionType.READ, "/resource/path");
```

REQUIREMENTS

Configure roles such that:

- Anonymous user can read:

  /public/resources/*

- Only admin user can read:

  /admin/resources/*

VALIDATE ANONYMOUS ACCESS

```
public class PermissionsIntegrationTest {

  public static final Principal ANONYMOUS_PRINCIPAL =
    new Principal(Roles.ANONYMOUS);
  public static final Principal ADMIN_PRINCIPAL =
    new Principal(Roles.ADMIN);

  private AppConfiguration appConfiguration = new AppConfiguration();

  @Test
  public void testAnonymousAccessToPublicResource() {
    assertTrue(appConfiguration.acl().checkIf(ANONYMOUS_PRINCIPAL)
     .can(PermissionType.READ, "/public/resource"));
  }
}
```

IMPLEMENT ANONYMOUS ACCESS

```
public class AppConfiguration {
  public ACL acl() {
    return ACL.acl().withRootPermissions(emptySet()).build();
  }
}
```

VALIDATE ADMIN ACCESS

```
public class PermissionsIntegrationTest {

...

  @Test
  public void testAnonymousAccessToAdminResourceIsDenied() {
    assertFalse(appConfiguration.acl().checkIf(ANONYMOUS_PRINCIPAL)
      .can(PermissionType.READ, "/admin/resource"));
  }

  @Test
  public void testAdminAccessToAdminResource(){
  assertTrue(appConfiguration.acl().checkIf(ADMIN_PRINCIPAL)
    .can(PermissionType.READ, "/admin/resource"));
  }
}
```

IMPLEMENT ADMIN RESTRICTIONS

```
public class AppConfiguration {

  public ACL acl() {
    return ACL.acl().withRootPermissions(emptySet())
      .resource("/admin").requiresPermissionFor(PermissionType.READ)
      .build();
  }
}
```

IMPLEMENT NEW REQUIREMENTS

```
public class Roles {
  public static Role ANONYMOUS = new Role(new Permissions());
  public static Role ADMIN = new Role(new Permissions(
    new Permission(READ, of("/admin"))
  ));
}
```

A NEW REQUIREMENT

New Requirement: Principal with USER role should be able to access "/user/resources" not "/admin/resources".

IMPLEMENT NEW REQUIREMENTS

```
public class Roles {
  public static Role ANONYMOUS = new Role(new Permissions());
  public static Role ADMIN = new Role(new Permissions(
    new Permission(READ, Optional.of("/admin"))
  ));

  public static Role USER = new Role(new Permissions(
    new Permission(READ, Optional.of("/user"))
  ));
}
```

VALIDATE NEW REQUIREMENTS

```
public class NewRequirementsIntegrationTest {
  private AppConfiguration appConfiguration = new AppConfiguration();

  @Test
  public void testUserAccessToUserResource() {
    assertTrue(appConfiguration.acl().checkIf(USER_PRINCIPAL)
      .can(PermissionType.READ, "/user/resource"));
  }

  @Test
  public void testUserCannotAccessAdminResource() {
    assertFalse(appConfiguration.acl().checkIf(USER_PRINCIPAL)
      .can(PermissionType.READ, "/admin/resource"));
  }
}
```

OOOPS

```
public class NewRequirementsOopsIntegrationTest {

  private AppConfiguration appConfiguration = new AppConfiguration();

    @Test
    public void testAnonymousAccessToUserResourceIsDenied() {
      assertFalse(appConfiguration.acl().checkIf(ANONYMOUS_PRINCIPAL)
       .can(READ, "/user/resource"));
    }
}
```

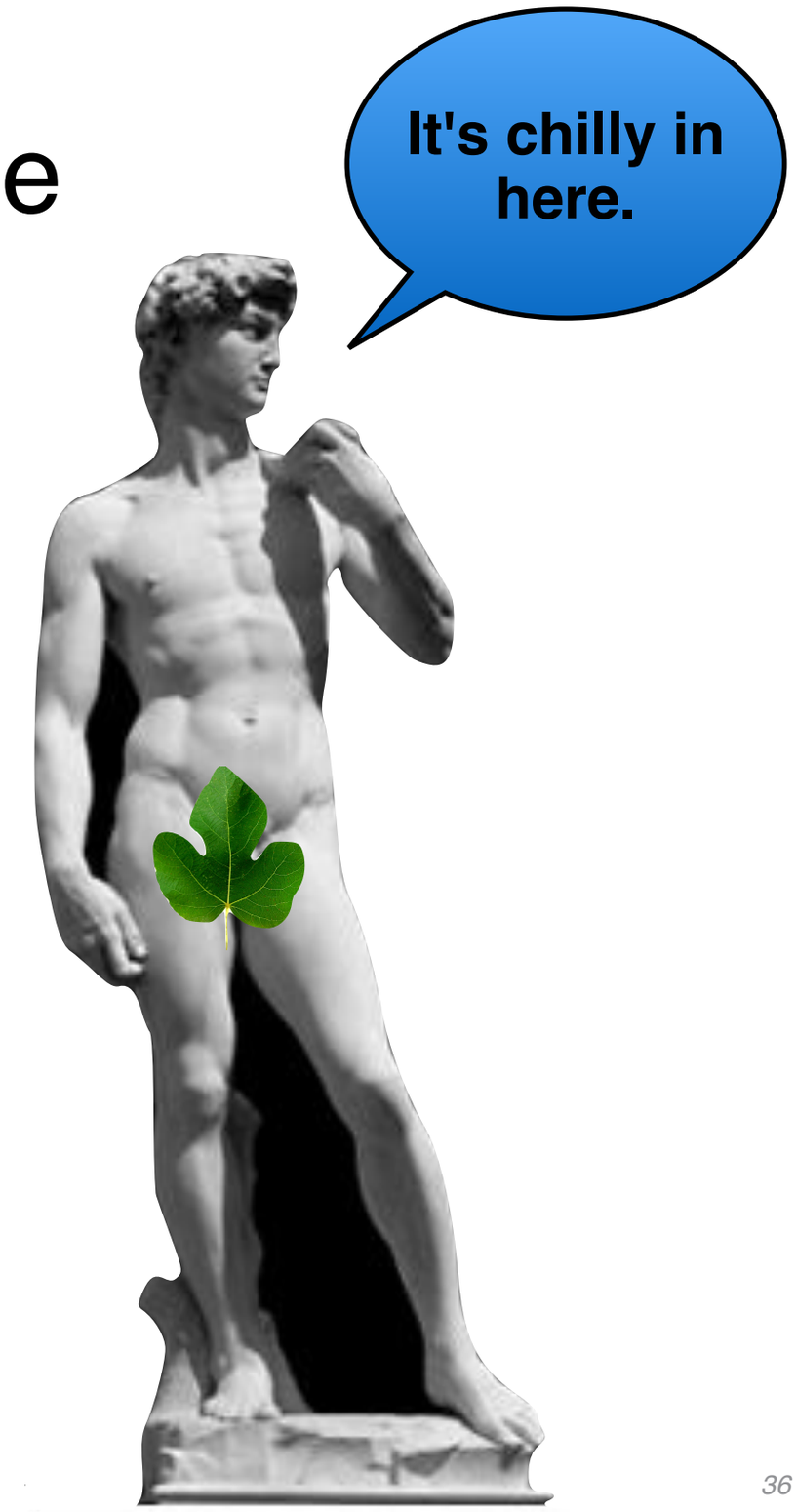
DEVELOPMENT PATTERN #2

Name: The Michelanglo

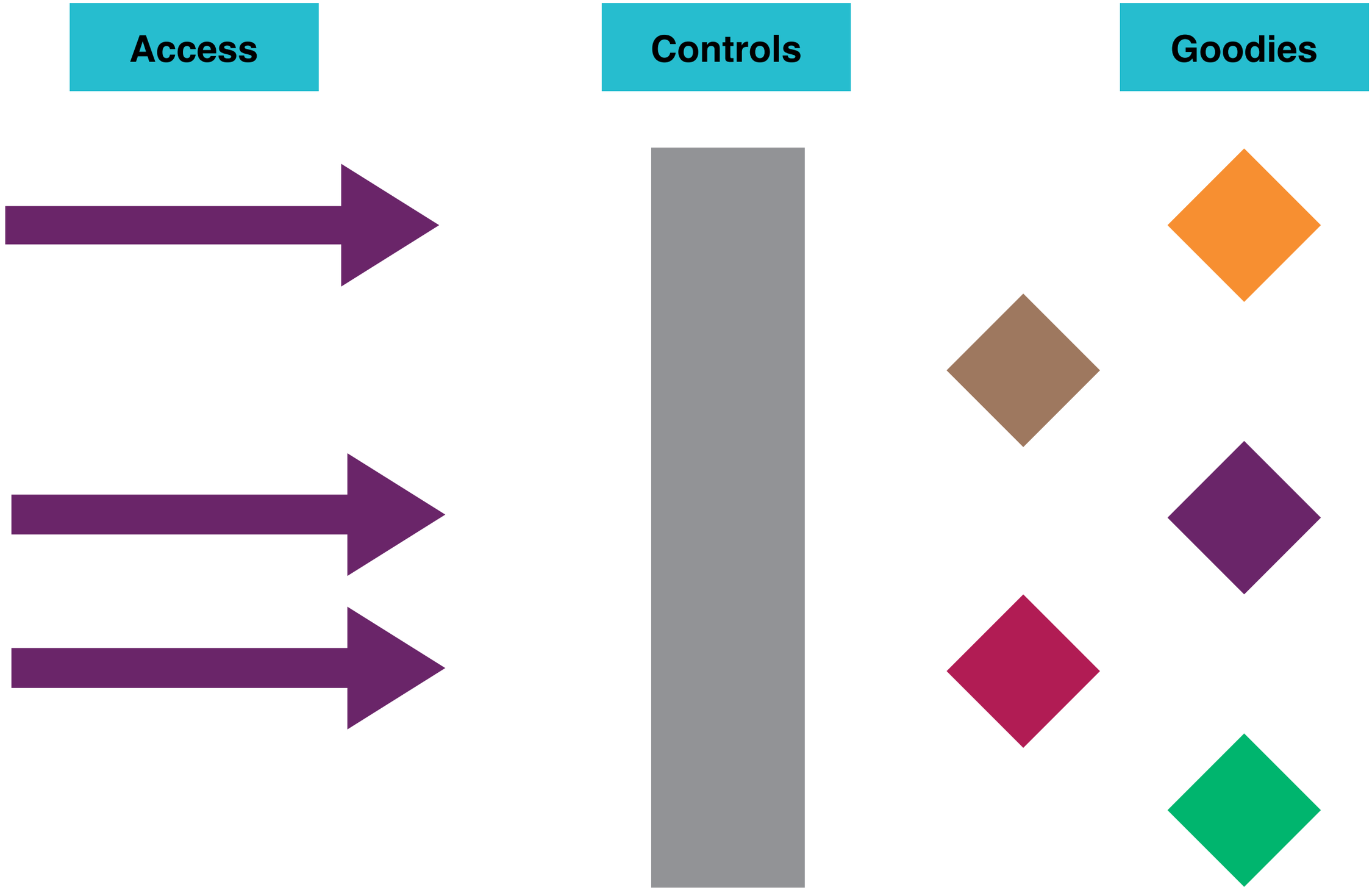
AKA: Open It Up, Least Privilege

Benefits: Safer defaults

Examples: Blocking by default



DEVELOPMENT PATTERN #2: THE MICHELANGELO.



THE MICHELANGELO PROCESS

- start with a single test that asserts no access
- write a series of tests that make a series of assertions about the level of access from least to most
- incrementally make changes to code to increase the level of access until you reach your final goal

VALIDATE EPHEMERAL BASELINE

```
public class PermissionsIntegrationTest {

  @Test
  public void testEphemeralBaseline() {
    assertFalse(appConfiguration.acl().checkIf(ANONYMOUS_PRINCIPAL)
      .can(READ, "/"));
    assertFalse(appConfiguration.acl().checkIf(ANONYMOUS_PRINCIPAL)
      .can(READ, "/public"));
    assertFalse(appConfiguration.acl().checkIf(ANONYMOUS_PRINCIPAL)
      .can(READ, "/admin/resource"));
    assertFalse(appConfiguration.acl().checkIf(ADMIN_PRINCIPAL)
      .can(READ, "/"));
    assertFalse(appConfiguration.acl().checkIf(ADMIN_PRINCIPAL)
      .can(READ, "/public"));
    assertFalse(appConfiguration.acl().checkIf(ADMIN_PRINCIPAL)
      .can(READ, "/admin/resource"));
  }
}
```

IMPLEMENT EPHEMERAL CONDITION

```
public class AppConfiguration {

    public ACL acl() {
        return ACL.acl().withRootPermissions(allPermissionTypes())
                .build();
    }
}
```

CHIPPING AWAY AT THE MARBLE

```
@Test
public void testEphemeralBaseline() {
  assertFalse(appConfiguration.acl().checkIf(ANONYMOUS_PRINCIPAL)
    .can(PermissionType.READ, "/"));
  assertFalse(appConfiguration.acl().checkIf(ANONYMOUS_PRINCIPAL)
    .can(PermissionType.READ, "/public"));
  assertFalse(appConfiguration.acl().checkIf(ANONYMOUS_PRINCIPAL)
    .can(PermissionType.READ, "/admin/resource"));
  assertFalse(appConfiguration.acl().checkIf(ADMIN_PRINCIPAL)
    .can(PermissionType.READ, "/"));
  assertFalse(appConfiguration.acl().checkIf(ADMIN_PRINCIPAL)
    .can(PermissionType.READ, "/public"));
  assertFalse(appConfiguration.acl().checkIf(ADMIN_PRINCIPAL)
    .can(PermissionType.READ, "/admin/resource"));
}

@Test
public void testAdminAccessToAdminResource() {
  assertTrue(appConfiguration.acl().checkIf(ADMIN_PRINCIPAL)
    .can(PermissionType.READ, "/admin/resource"));
}
```

CHIPPING AWAY AT THE MARBLE

```
public class Roles {
  public static Role ANONYMOUS = new Role(new Permissions());
  public static Role ADMIN = new Role(new Permissions(
    new Permission(READ, Optional.of("/admin"))
  ));
}
```

CHIPPING AWAY AT THE MARBLE

```
  @Test
  public void testEphemeralBaseline() {
    assertFalse(appConfiguration.acl().checkIf(ANONYMOUS_PRINCIPAL)
      .can(PermissionType.READ, "/"));
    assertFalse(appConfiguration.acl().checkIf(ANONYMOUS_PRINCIPAL)
      .can(PermissionType.READ, "/public"));
    assertFalse(appConfiguration.acl().checkIf(ANONYMOUS_PRINCIPAL)
      .can(PermissionType.READ, "/admin/resource"));
    assertFalse(appConfiguration.acl().checkIf(ADMIN_PRINCIPAL)
      .can(PermissionType.READ, "/public"));
    assertFalse(appConfiguration.acl().checkIf(ADMIN_PRINCIPAL)
      .can(PermissionType.READ, "/"));
  }
...
  @Test
  public void testAdminAccessToOtherResources() {
    assertTrue(appConfiguration.acl().checkIf(ADMIN_PRINCIPAL)
      .can(PermissionType.READ, "/resource"));
    assertTrue(appConfiguration.acl().checkIf(ADMIN_PRINCIPAL)
      .can(PermissionType.READ, "/public/resource"));
  }
```

CHIPPING AWAY AT THE MARBLE

```
public class Roles {
  public static Role ANONYMOUS = new Role(new Permissions());
  public static Role ADMIN = new Role(new Permissions(
    new Permission(READ, Permission.GLOBAL)
  ));
}
```

CHIPPING AWAY AT THE MARBLE

```
  @Test
   public void testEphemeralBaseline() {
     assertFalse(appConfiguration.acl().checkIf(ANONYMOUS_PRINCIPAL)
       .can(PermissionType.READ, "/"));
     assertFalse(appConfiguration.acl().checkIf(ANONYMOUS_PRINCIPAL)
       .can(PermissionType.READ, "/public/resource"));
   }

...

   @Test
   public void testAnonymousAccessToPublicResource() {
     assertTrue(appConfiguration.acl().checkIf(ANONYMOUS_PRINCIPAL)
       .can(PermissionType.READ, "/public/resource"));
   }
}
```

CHIPPING AWAY AT THE MARBLE

```
public class Roles {
    public static Role ANONYMOUS = new Role(new Permissions(
            new Permission(READ, Optional.of("/public"))
    ));

    public static Role ADMIN = new Role(new Permissions(
            new Permission(READ, Permission.GLOBAL)
    ));

    public static Role USER = new Role(new Permissions(
            new Permission(READ, Optional.of("/user"))
    ));
}
```

NEW REQUIREMENT

New Requirement: Principal with USER role should be able to access "/user/resources" not "/admin/resources".

CHIPPING AWAY AT THE MARBLE

```
public class Roles {
  public static Role ANONYMOUS = new Role(new Permissions());
  public static Role ADMIN = new Role(new Permissions(
    new Permission(READ, Permission.GLOBAL)
  ));

  public static Role USER = new Role(new Permissions(
    new Permission(READ, Optional.of("/user"))
  ));
}
```

OOOPS????

```
public class NewRequirementsOopsIntegrationTest {

  private AppConfiguration appConfiguration = new AppConfiguration();

    @Test
    public void testAnonymousAccessToUserResourceIsDenied() {
      assertFalse(appConfiguration.acl().checkIf(ANONYMOUS_PRINCIPAL)
       .can(READ, "/user/resource"));
    }
}
```

APPLICATION

- Application Permissions
- Resource Permissions
- End point white-listing

ThoughtWorks®

ORGANIZATIONAL PATTERNS

ORGANIZATIONAL PATTERNS

Security is a cross-cutting concern, like "quality"

If your development teams believe that security is the problem of the security team, you are in trouble

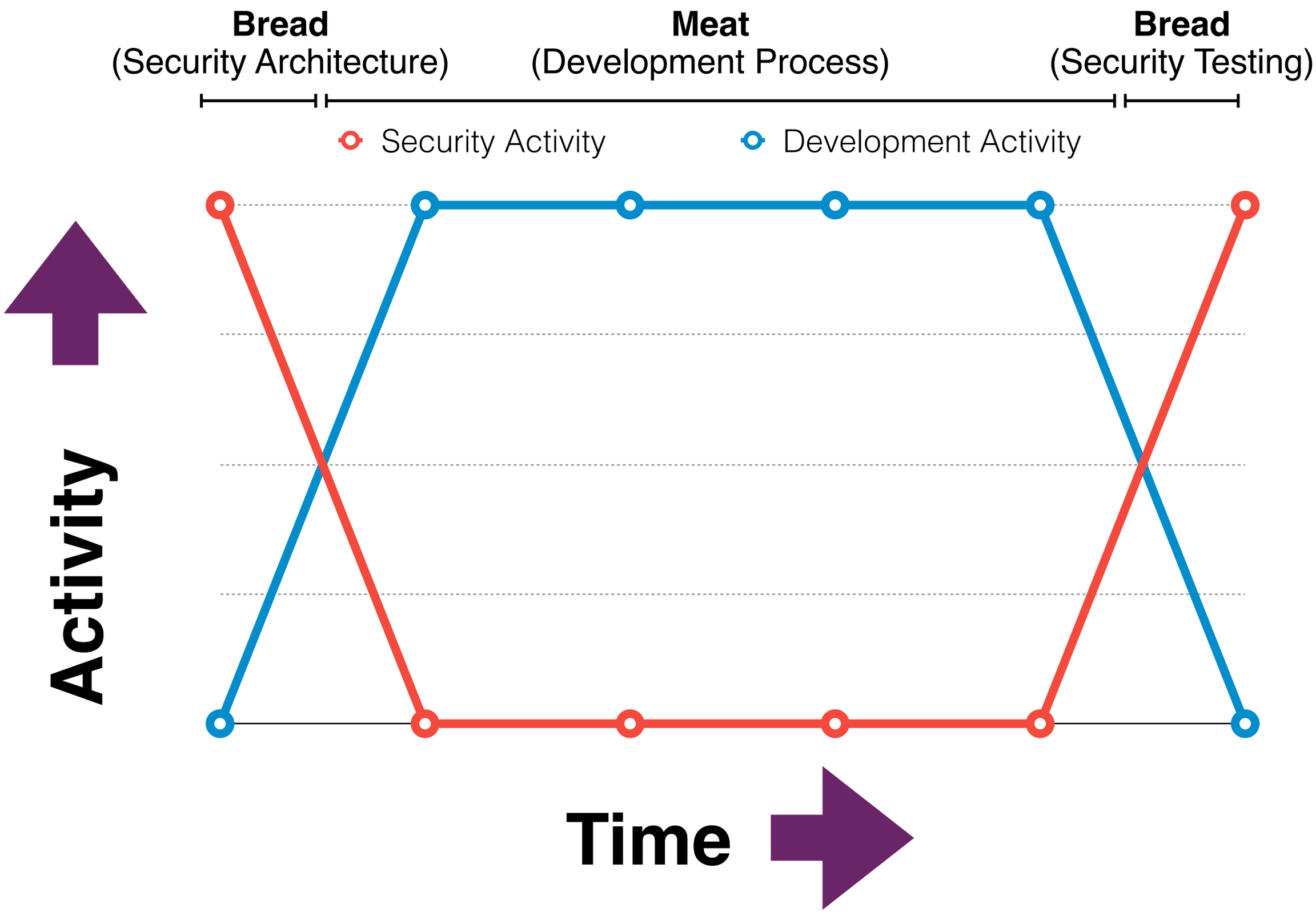
ORGANIZATIONAL ANTI-PATTERN #1

Name: Security Sandwich

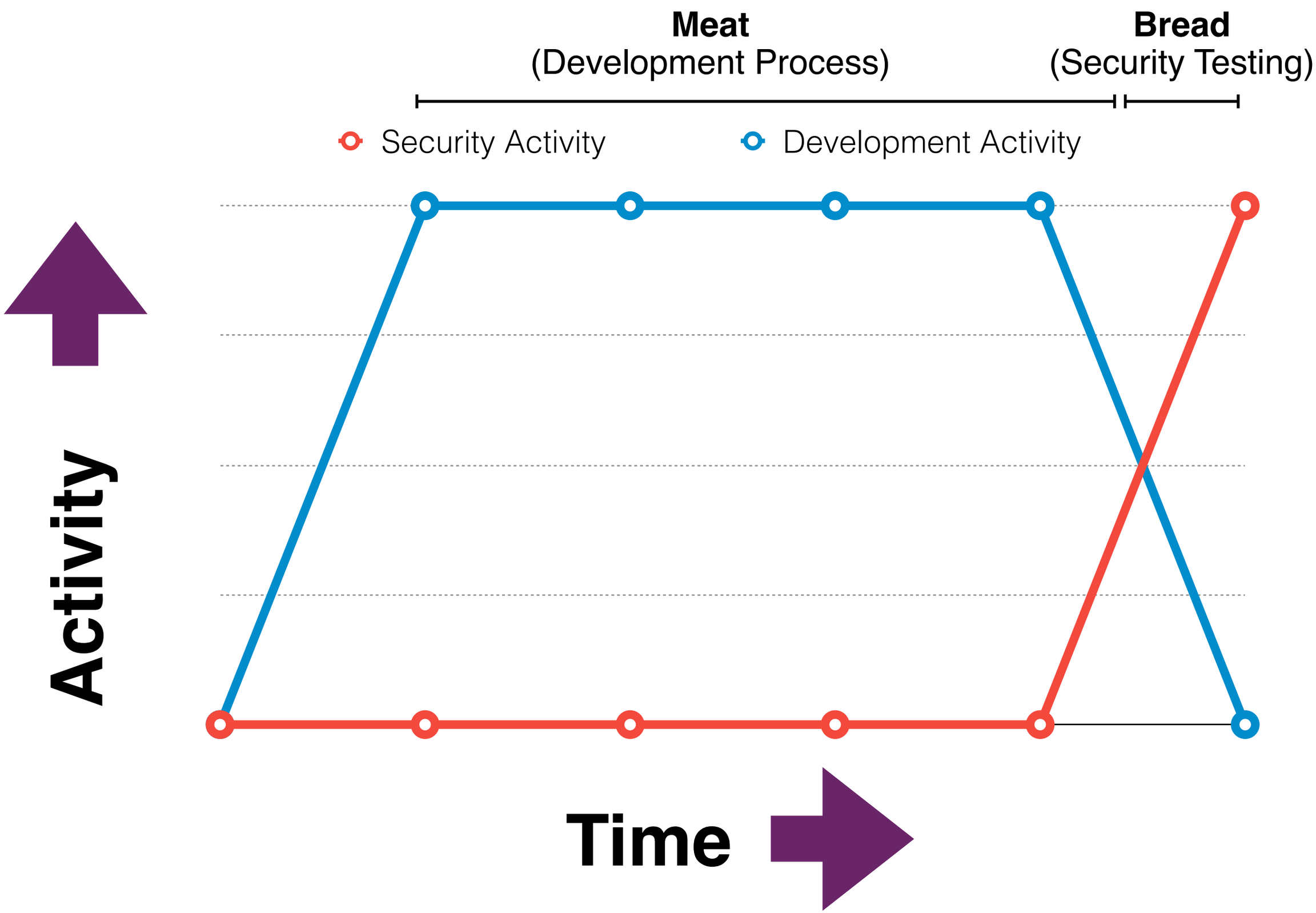
AKA: Phase / Gate Security

Consequences: Expensive, high risk fixes, buried vulnerabilities

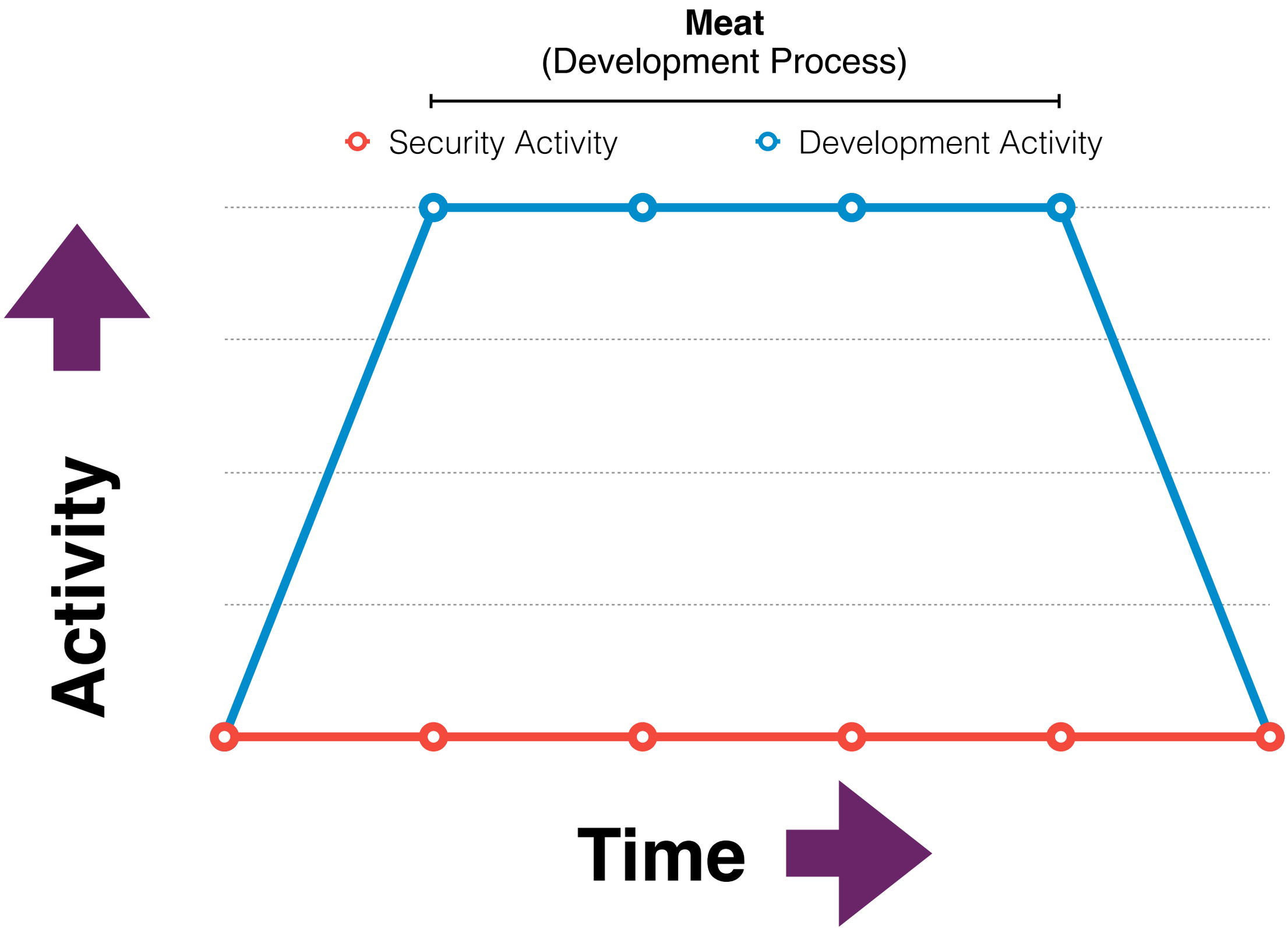
ORGANIZATIONAL ANTI-PATTERN: SECURITY SANDWICH



ORGANIZATIONAL ANTI-PATTERN: SECURITY SANDWICH



ORGANIZATIONAL ANTI-PATTERN: SECURITY SANDWICH



CONSEQUENCES OF THE SECURITY SANDWICH

- Cost of Mitigation
- Cost of Detection
- Risk of Mitigation
- Risk of Not Mitigating

ORGANIZATIONAL PATTERN #1

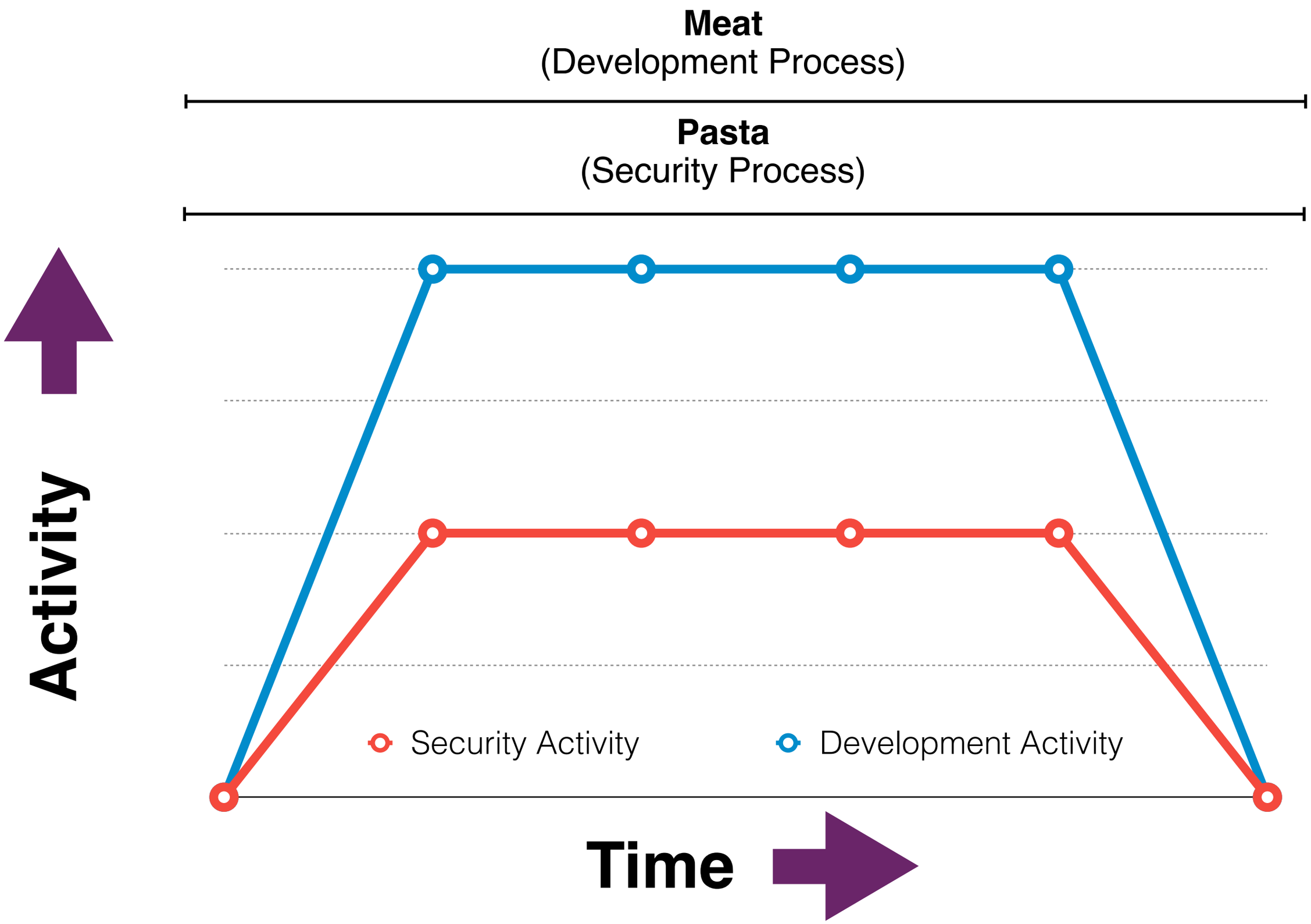
Name: Security Lasagna

AKA: Secure SDLC

Benefits: Safer, less expensive mitigation

Examples: Continuous Security, Security in the pipeline

ORGANIZATIONAL PATTERN: SECURITY LASAGNA

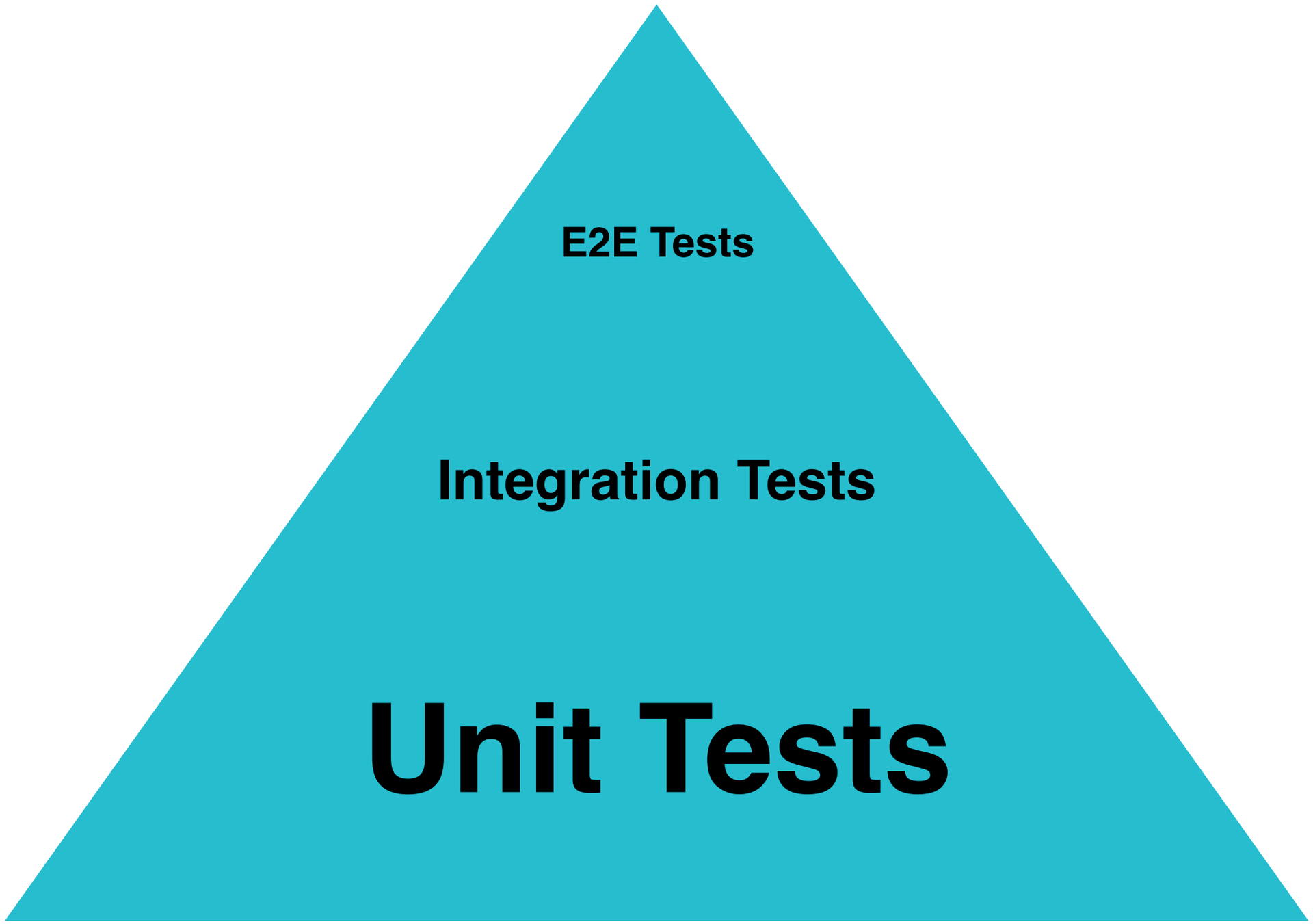


SECURITY LASAGNA IMPLEMENTATION

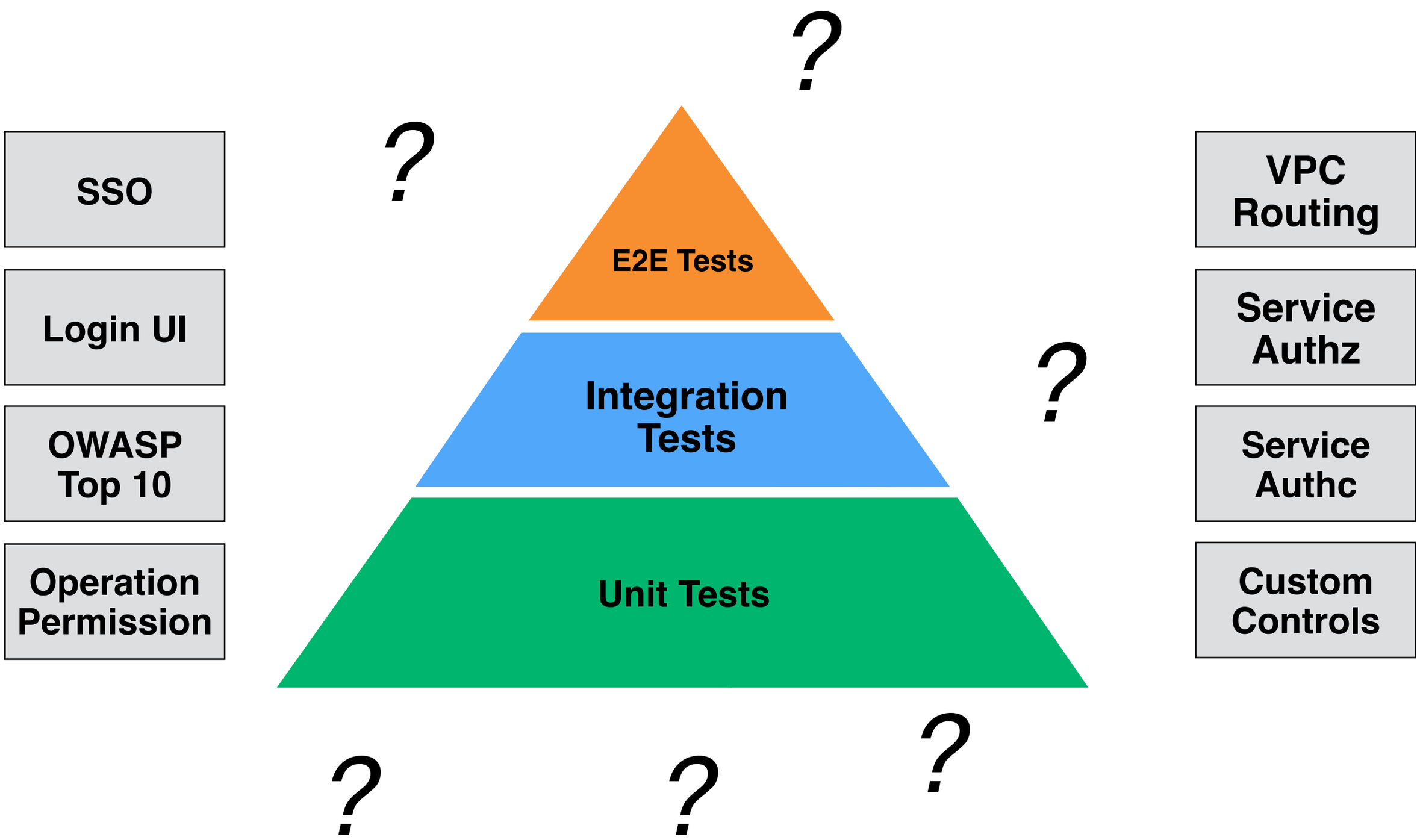
- Application security starts in the stories
- Developers write security assertions, just like they would any other functional concern, with support of tools, as necessary.
- Story sign-off includes validation of security requirements

SECURITY LASAGNA IMPLEMENTATION

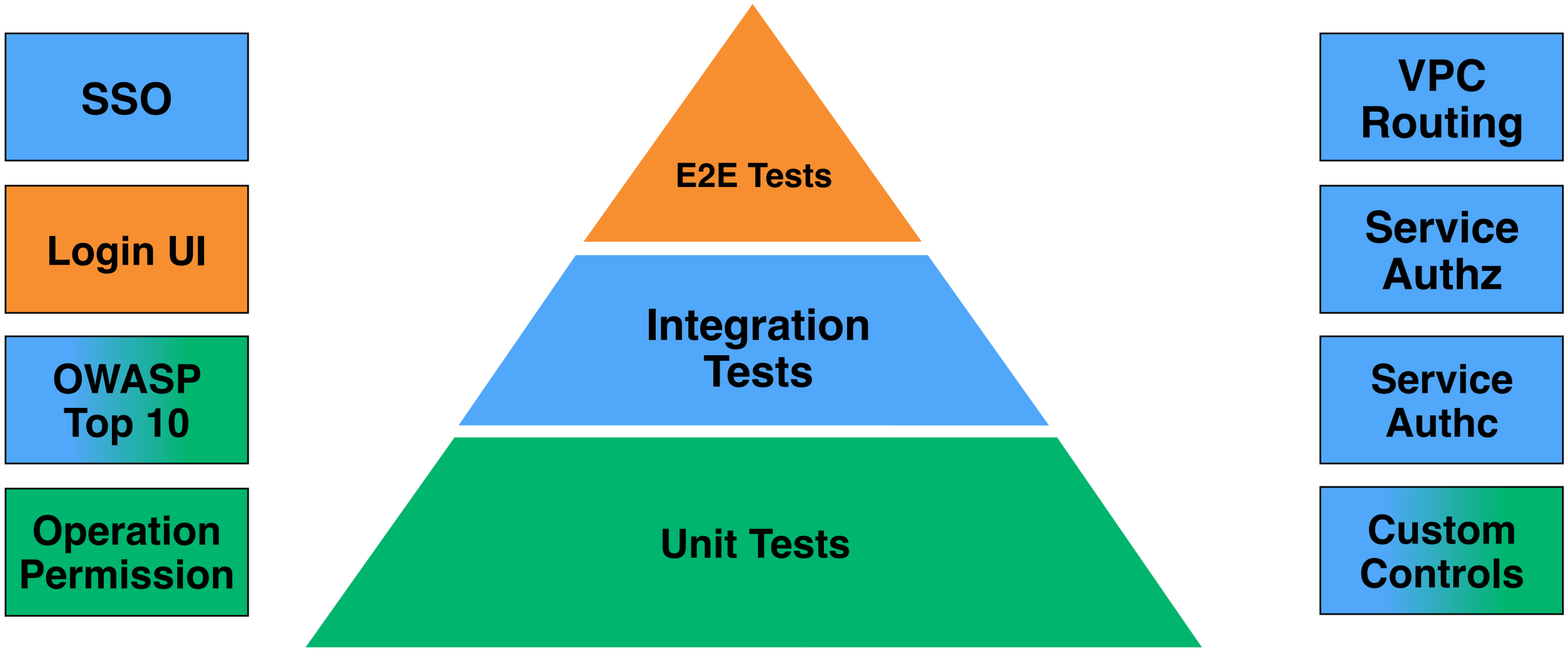
Where do you put your security assertions?



THE TESTING PYRAMID



THE TESTING PYRAMID



ORGANIZATIONAL ANTI-PATTERN #2

Name: I'll Take a Side of Security

AKA: Security Checkbox

Consequences: Inappropriately targeted security posture. High costs for little business benefit.

THE CONVERSATION

"The Business"

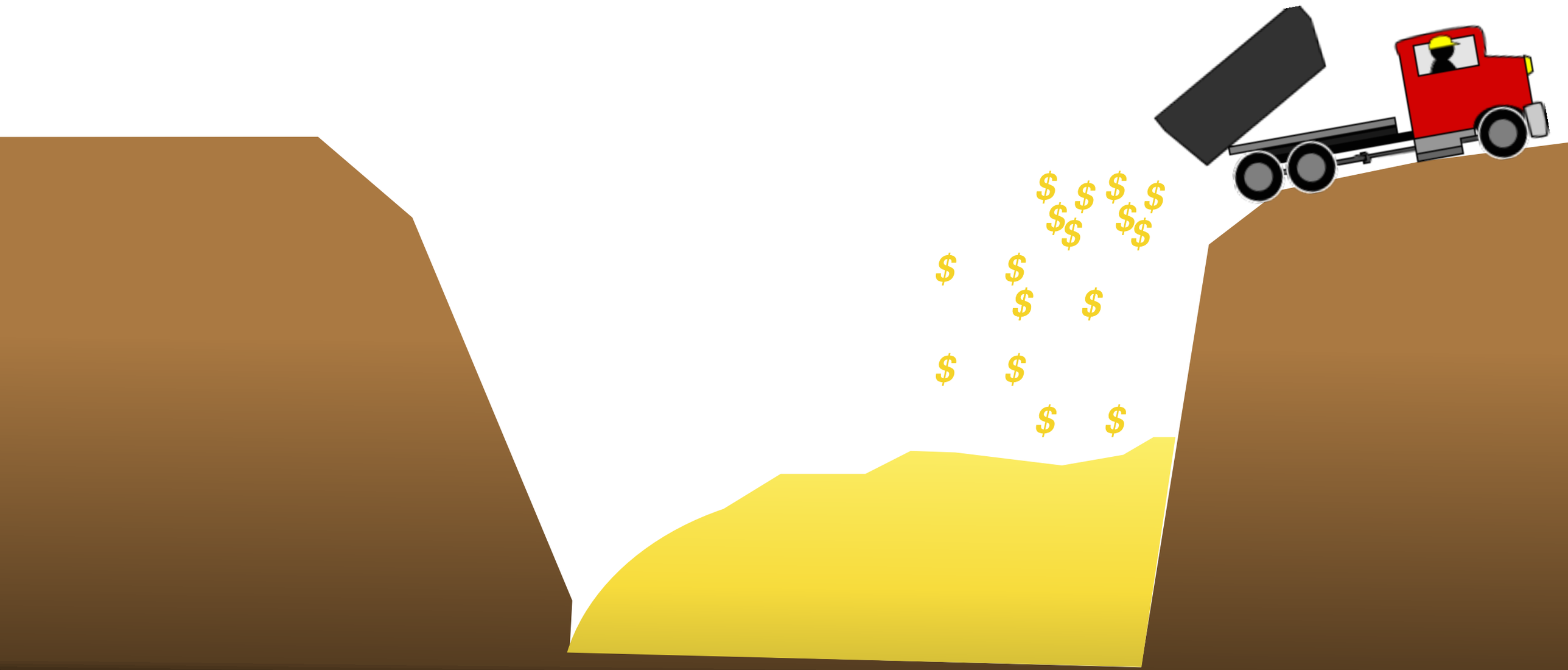
Please create features X, Y, and Z… and make them secure.

Ok…

"The Geeks"

We'll integrate our OAuth 2.0-based SSO using a secure HTTP header with SAML and blah blah blah blah blah blah….

THE CONSEQUENCES



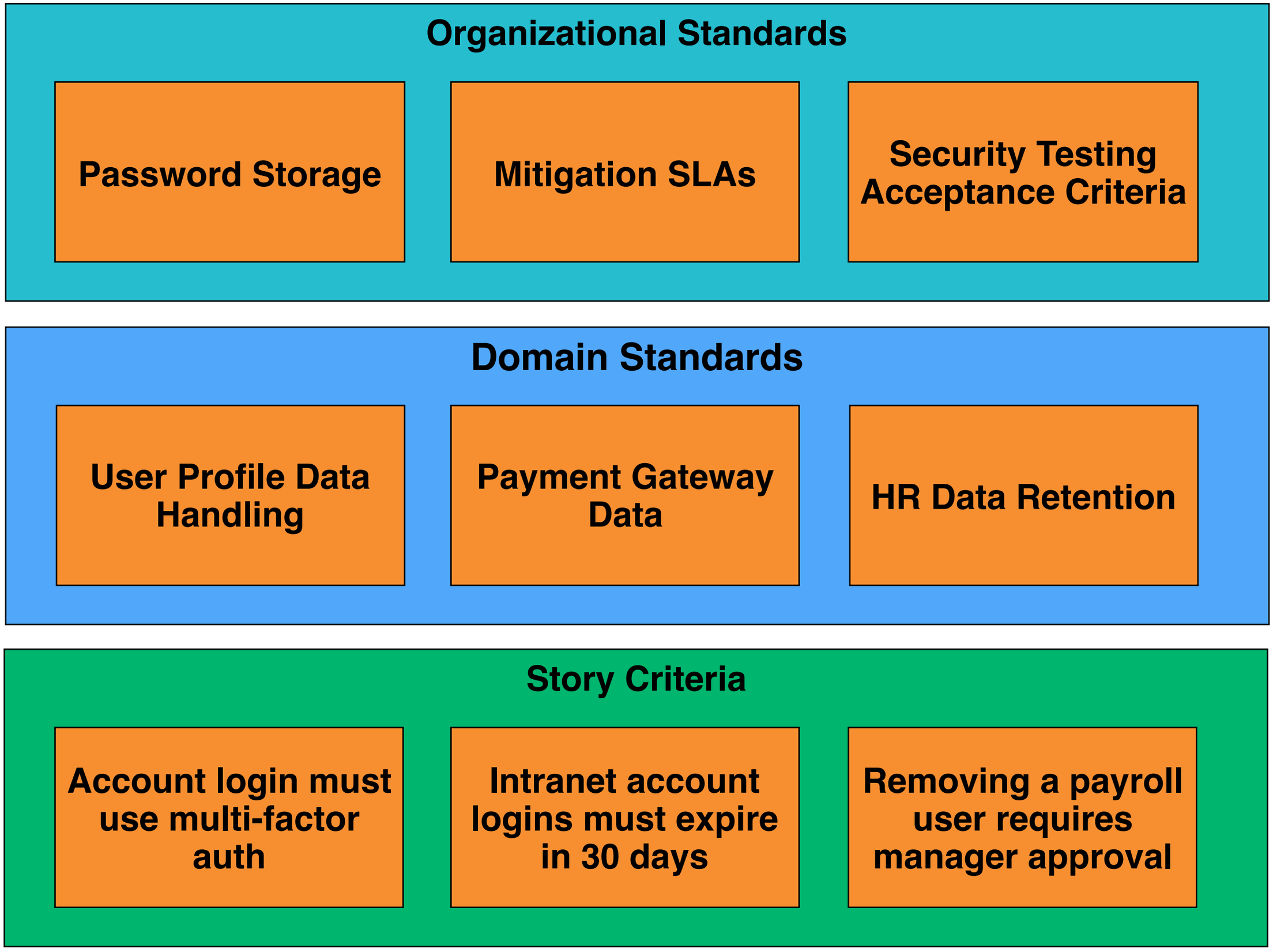
ORGANIZATIONAL PATTERN #2

Name: Layer Cake

AKA: Tiered Security Standards

Benefits: Clear business-driven definitions that drive implementation choices. Investment in securing resources that are most important to the interests of the organization.

ORGANIZATIONAL PATTERN: LAYER CAKE



LAYER CAKE VERSION 1

"The Business"

Brand is our most important asset

Customer data must be protected, even from a disgruntled employee.

We really need to prevent the site from being defaced

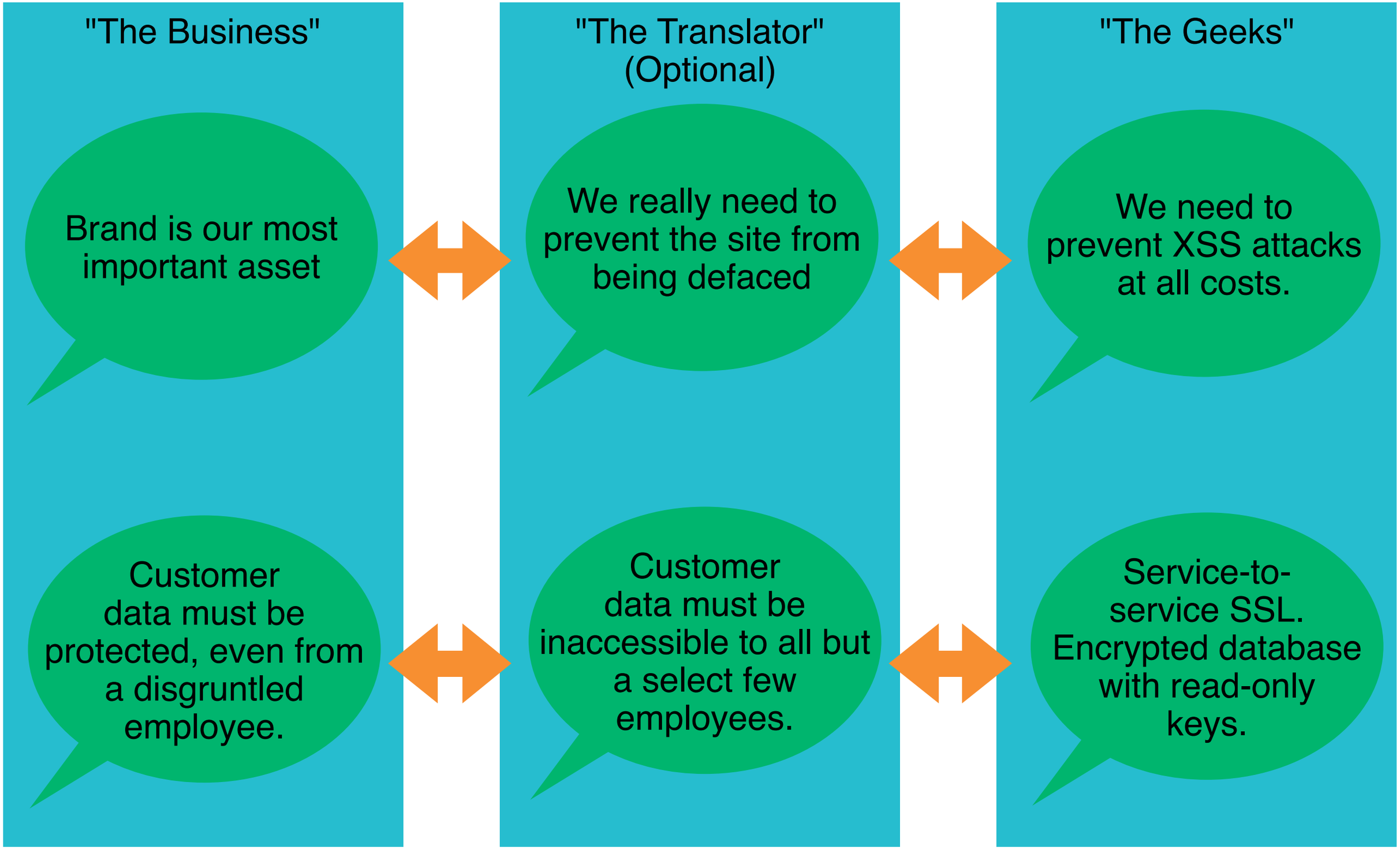
Customer data must be inaccessible to all but a select few employees.

"The Geeks"

We need to prevent XSS attacks at all costs.

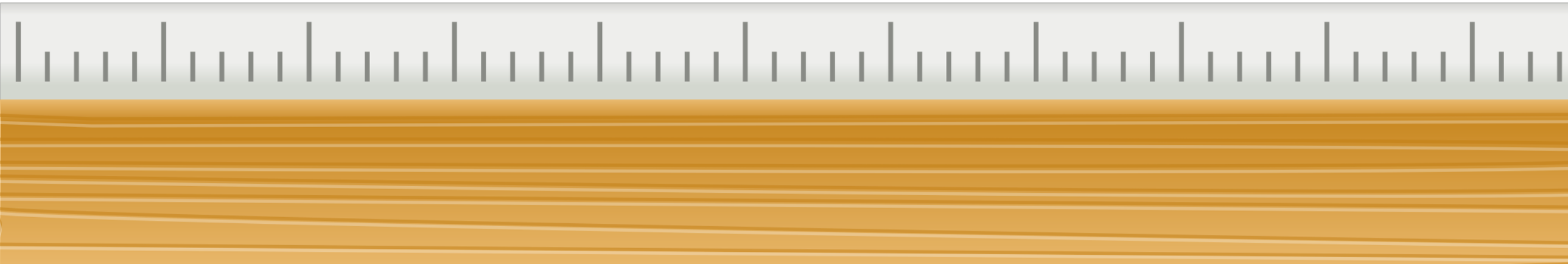
Service-to-service SSL. Encrypted database with read-only keys.

LAYER CAKE VERSION 2



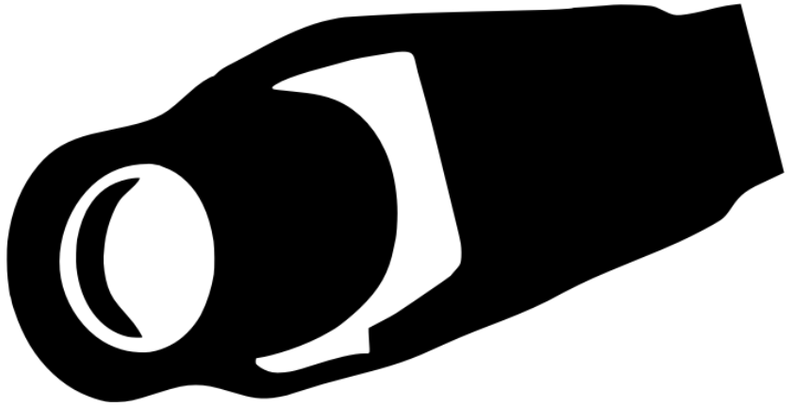
ThoughtWorks®

Driving security investments from measurement.



HOLISTIC MONITORING

Monitoring should be something that encompasses more than just your network.



THANK YOU

Daniel Somerfield
dsomerfi@thoughtworks.com
https://continuoussecurity.wordpress.com/
@D_Somerfield

ThoughtWorks®