Coding with Identity Management & Security

Part 2 of Identity Management with Progress OpenEdge

Peter Judge
OpenEdge Development
pjudge@progress.com

PROGRESS EXCHANGE 2013
DISCOVER. DEVELOP. DELIVER.

What Is Identity Management?

Identity management is all about trust relationships

It’s about protecting your business data

You make security decisions on behalf of your customers…understand the maximum loss they might suffer

What Is Identity Management?

It's about protecting your business data by

- Controlling and verifying who accesses your data
- Controlling what they can do with your data
- Reviewing what they did with your data
- Maintaining information about your users

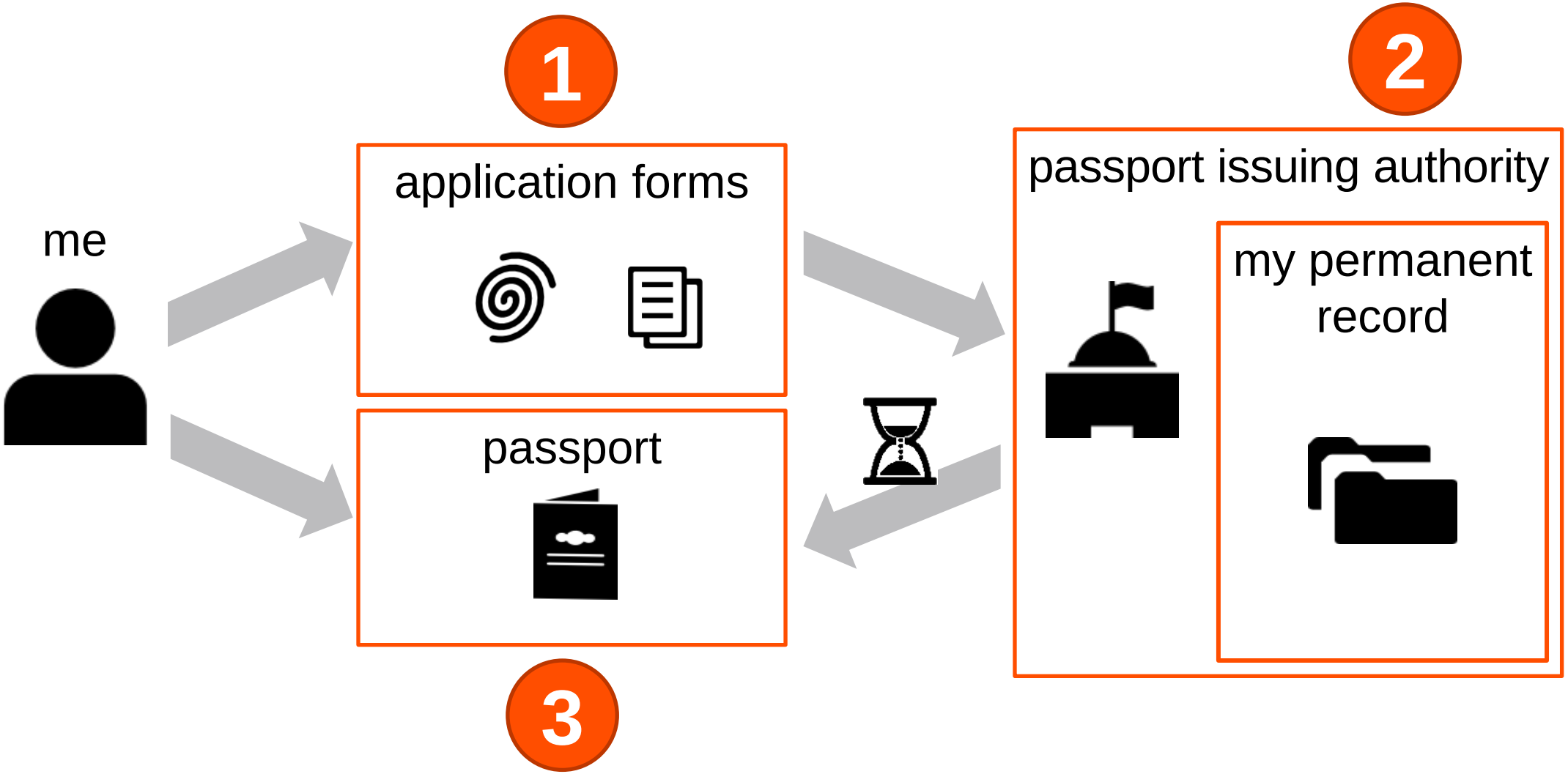
Authentication

Authorisation

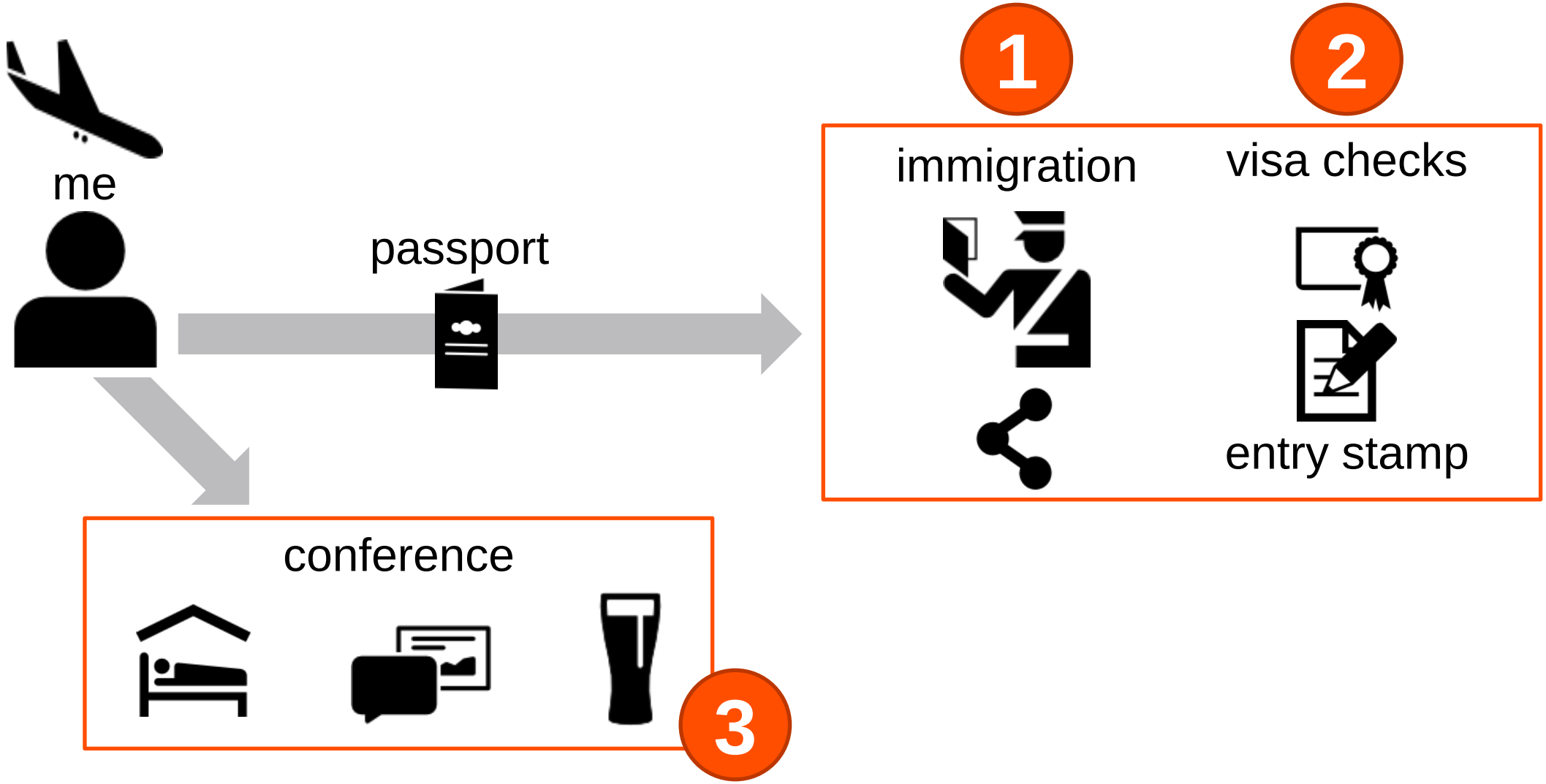
Auditing

Administration

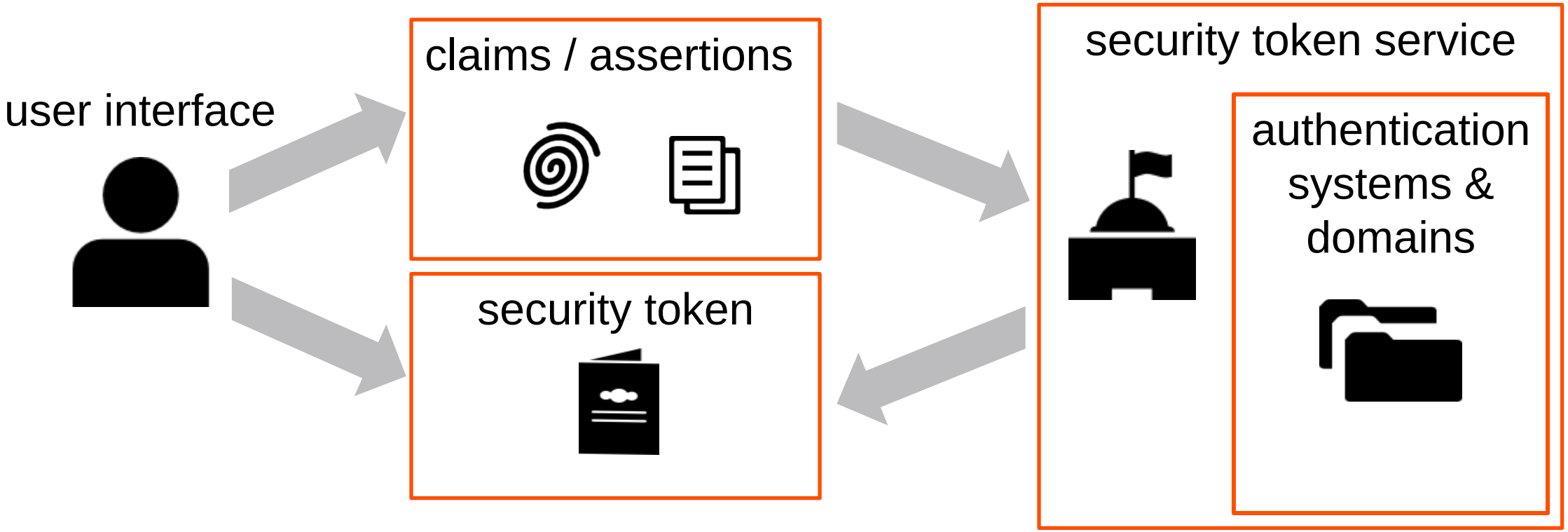
Getting a Passport



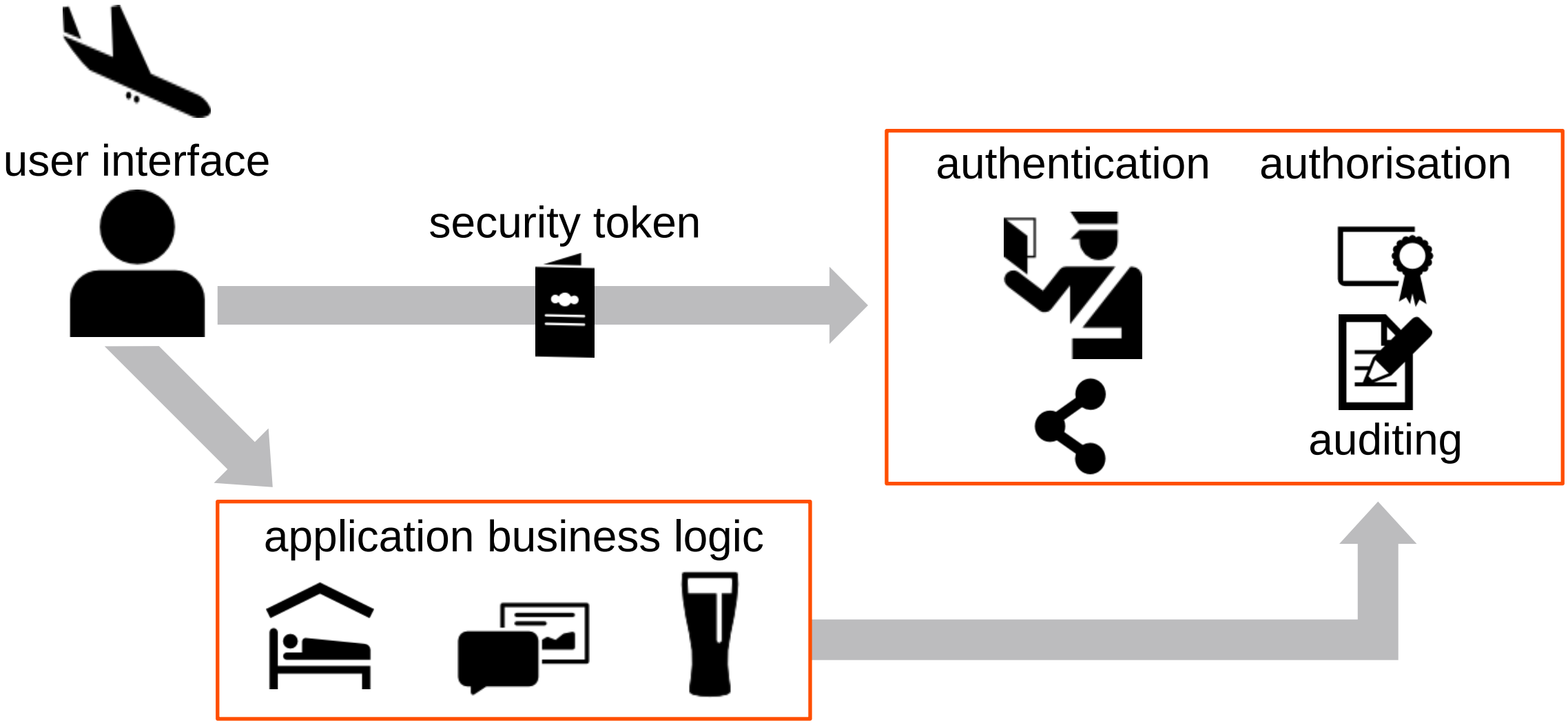
Using a Passport



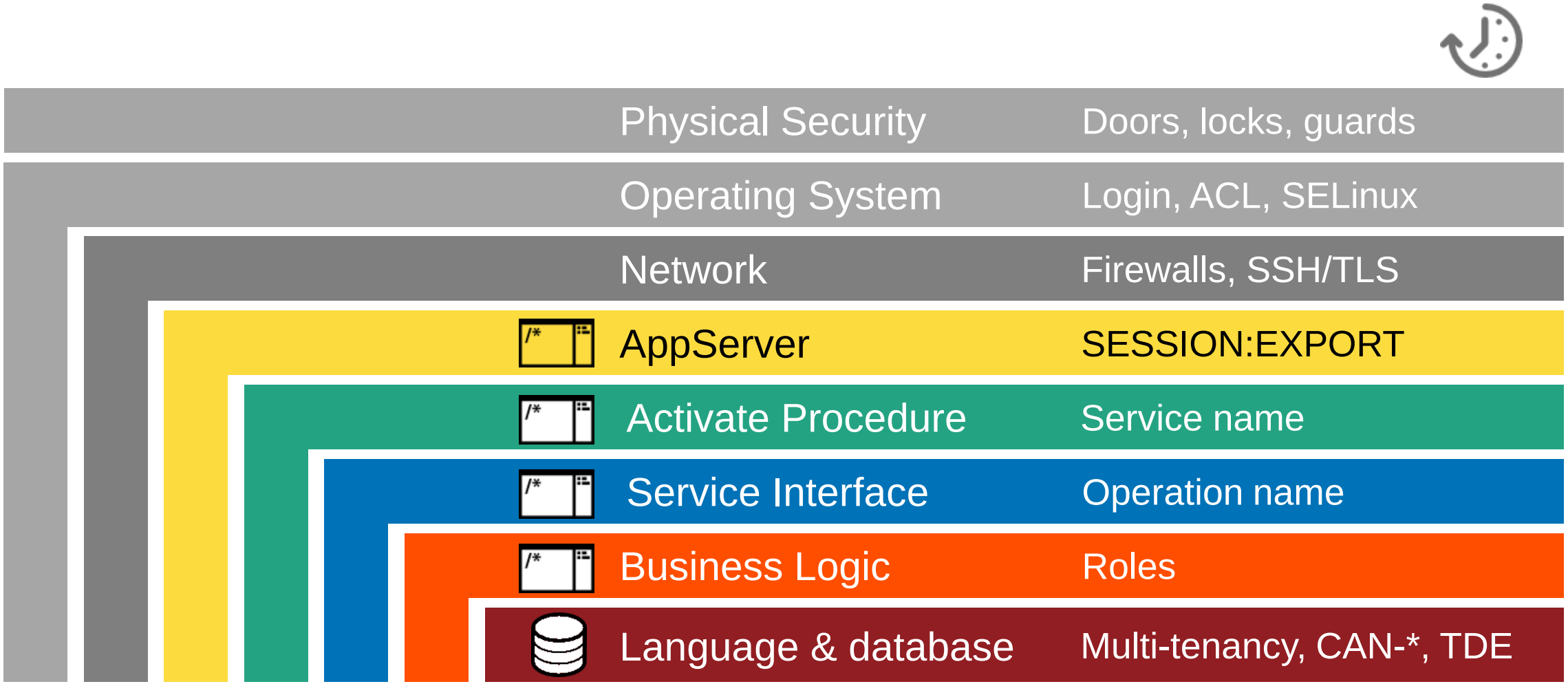
Application Flow: Login



Application Flow: Business Logic



Authorisation: Defence in Depth



When Authorisation Fails

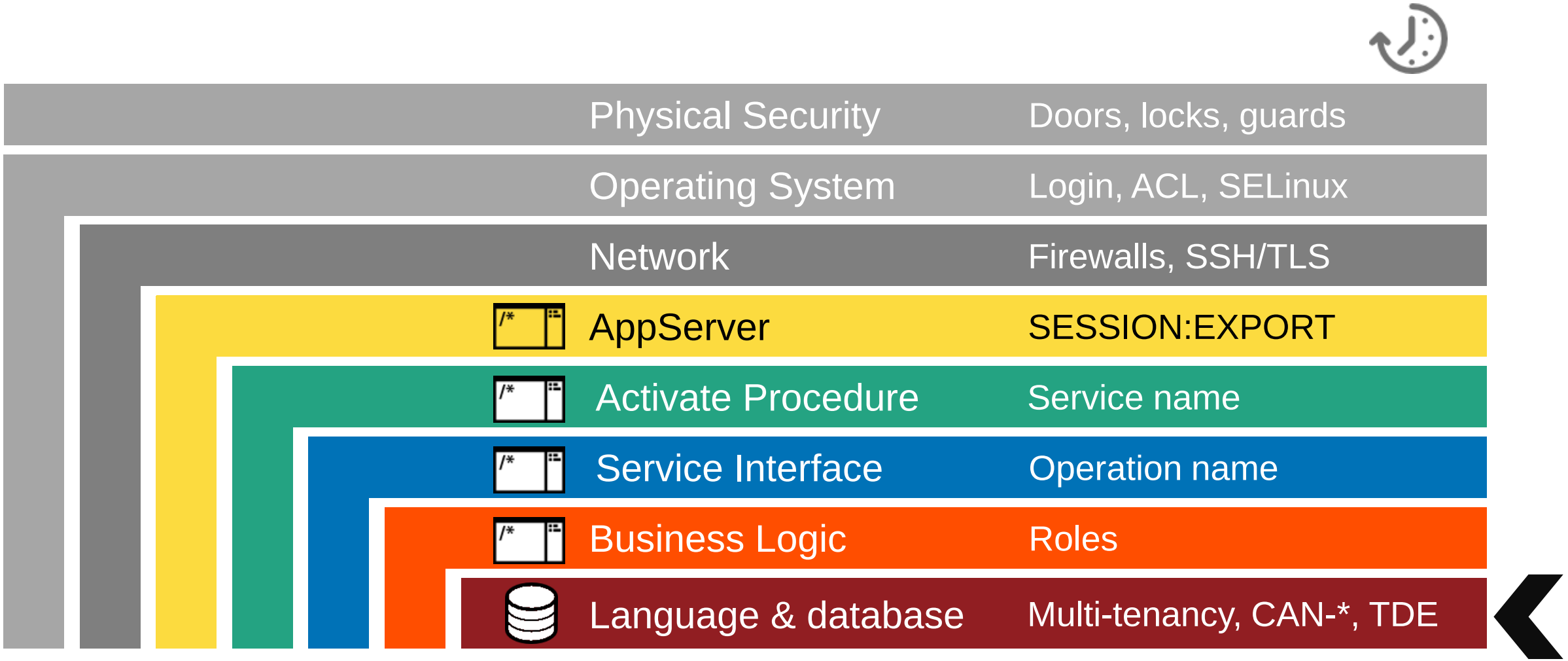
- Record
- Rewind
  - The deeper you are in the stack, the harder it is to unwind
  - The deeper you are in the stack, the less info you have
- Return
  - Nondescript error messages

```
undo, throw new AppError(
  substitute("User &1 not authorised for service &2",
             phCP:qualified-user-id, pcServiceName)).

undo, throw new AppError("User not authorised for service").
```

Roles & Responsibilities

	Anonymous	Customer	Employee	System
See catalogue	X	X	X	X
Modify catalogue			X	X
Update shopping cart		X	X	X
Add users			X	X
Dump & load data				X
Provision services				X
Level of trust				

Authorisation: Defence in Depth



What Are Domains?

A group of users with a common set of

- Roles and responsibilities
- Level of security
- Data access privileges

Also configured in db meta-schema

- Authentication systems
- Tenants

```
_sec-authentication-domain

   _Domain-name
   _Domain-type
   _Domain-description
   _Domain-access-code
   _Domain-runtime-options
   _Tenant-name

   _Domain-enabled
```

OE 10.1A+ | OE 11.0+ | OE 11.1+

Roles & Responsibilities: By Business Role

	System
See catalogue	X
Modify catalogue	X
Update shopping cart	X
Add users	X
Dump & load data	X
Provision services	X
Domain name	@system

Roles & Responsibilities: By Business Role

	System	Application Admin	DB Admin
See catalogue	X	X	X
Modify catalogue	X	X	
Update shopping cart	X	X	
Add users	X	X	
Dump & load data	X		X
Provision services	X		X
Domain name	@system	@app-admin.system	@db-admin.system

Roles & Responsibilities: By Business Role

	Employee
See catalogue	X
Modify catalogue	X
Update shopping cart	X
Add users	X
Dump & load data	
Provision services	
Domain name	@employee

Roles & Responsibilities: By Business Role

	Employee		
	Customer Service	HR	Sales
See catalogue	X	X	X
Modify catalogue	X		X
Update shopping cart	X		
Add users	X		
Dump & load data			
Provision services			
Domain name	@cs.employee	@hr.employee	@sales.employee

Roles & Responsibilities: By Tenancy

	Employee		
	Tenant Aye	Tenant Bee	Tenant Sea
See catalogue	X	X	X
Modify catalogue	X	X	X
Update shopping cart	X	X	X
Add users	X	X	X
Dump & load data			
Provision services			
Domain name	@employee.aye	@employee.bee	@employee.sea

Roles & Responsibilities: By Location

	Employee		
	EMEA	Americas	APAC
See catalogue	X	X	X
Modify catalogue	X	X	X
Update shopping cart	X	X	X
Add users	X	X	X
Dump & load data			
Provision services			
Domain name	@emea.employee	@na.employee	@apac.employee

Roles & Responsibilities: By Tenancy and Business Role

	Customer Service Employee		
	Tenant Aye	Tenant Bee	Tenant Sea
See catalogue	X	X	X
Modify catalogue	X	X	X
Update shopping cart	X	X	X
Add users	X	X	X
Dump & load data			
Provision services			
Domain name	@cs.employee.aye	@cs.employee.bee	@cs.employee.sea

_File Operations

```
find _File where _File-Name eq "Customer"
_Desc       : "Customer master data"
_Can-Create : "*@cs.employee.*, *@app-admin.system, !*@customer"
_Can-Write  : "*@cs.employee.*, *@app-admin.system, !*@customer"
_Can-Delete : "*@cs.employee.*, *@app-admin.system, !*@customer"
_Can-Read   : "*"

find _File where _File-Name eq "ShoppingCart"
_Desc       : "Customer Shopping Cart data"
_Can-Create : "*@customer, *@cs.employee.*, *@app-admin.system"
_Can-Write  : "*@customer, *@cs.employee.*, *@app-admin.system"
_Can-Delete : "*@customer, *@cs.employee.*, *@app-admin.system"
_Can-Read   : "*"

find _File where _File-Name eq "ApplicationUser"
_Desc       : "Application login data"
_Can-Create : "*@cs.employee.*, *@jane@app-admin.system"
_Can-Write  : "*@cs.employee.*, *@jane@app-admin.system"
_Can-Delete : "*@cs.employee.*, *@jane@app-admin.system"
_Can-Read   : "*"
```

OE 3.0+ OE 11.0+

Roles & Responsibilities

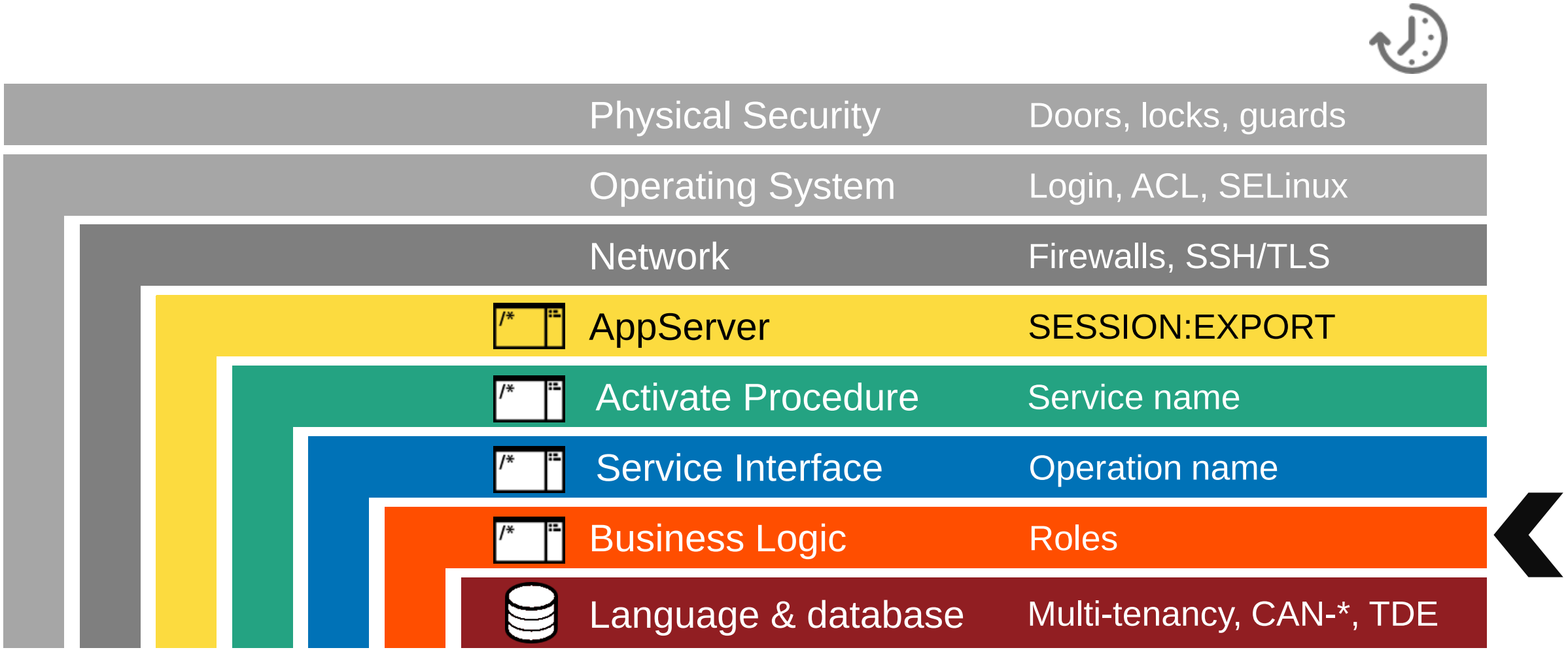
	Anonymous	Customer	Employee	System
See catalogue	X	X	X	
Modify catalogue			X	
Update shopping cart		X	X	
Add users			X	
Dump & load data				X
Provision services				X
Level of trust	■ □ □	■ ■ □	■ ■ □	■ ■ ■

Roles

- Roles a way of mapping sets of capabilities to classes of users
- May not serve the principle of least privilege
  - (which states that one should have the minimal privileges necessary, and no more)
- On the other end of the spectrum, one can define one role for every set of resource capabilities one might want to allow
- Map roles to static sets of capabilities

Role definition from OWASP https://www.owasp.org/index

Authorisation: Defence in Depth



Configuration: Roles

```
create SecurityRole.
Name        = 'ShoppingCart.Data.Create'.
Creator     = 'jane@app-admin.system'.
Description = 'Allows create access to the shopping cart table'.

create SecurityRole.
Name        = 'ShoppingCart.Data.Write'.
Creator     = 'jane@app-admin.system'.
Description = 'Allows write access to the shopping cart table'.

create SecurityRole.
Name        = 'ShoppingCart.Data.Delete'.
Creator     = 'jane@app-admin.system'.
Description = 'Allows delete access to the shopping cart table'.
```

Configuration: Granting Access

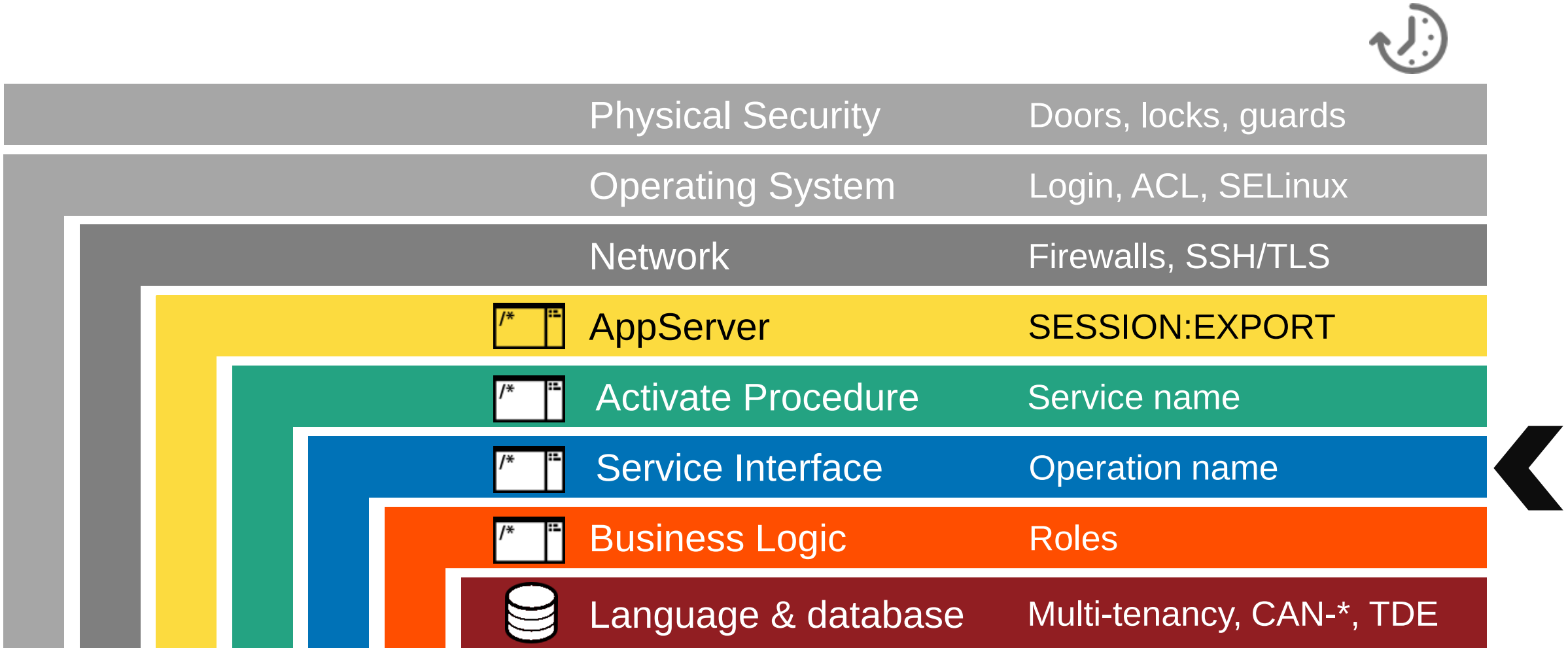
```
create GrantedRole.
Name    = 'ShoppingCart.Data.Create'.
Grantee = '*@customer'    /* one record per domain */
Grantor = 'jane@app-admin.system'.

create GrantedRole.
Name    = 'ShoppingCart.Data.Write'.
Grantee = '*@customer'    /* one record per domain */
Grantor = 'jane@app-admin.system'.

create GrantedRole.
Name    = 'ShoppingCart.Data.Read'.
Grantee = '*@customer'    /* one record per domain */
Grantor = 'jane@app-admin.system'.

create GrantedRole.
Name    = 'ShoppingCart.Data.Delete'.
Grantee = 'supervisor@app-admin.system'.  /* one record per user */
Grantor = 'jane@app-admin.system'.
```

Using Roles

```
/* runs after the user has been successfully authenticated  */

for each GrantedRole where
   Grantee = hCP:qualified-user-id or   /* amy@customer   */
   Grantee = '*@' + hCP:domain-name:    /*   *@customer   */

  hCP:roles = GrantedRole.Name + ',' + hCP:roles .
end.

/* Roles: ShoppingCart.Data.Write, ShoppingCart.Data.Create ... */
```

```
/* later, when the user attempts access */
pcOperation = 'ShoppingCart.Data.Create'.
if not can-do(hCP:roles, pcOperation) then
    undo, throw new AppError('User not authorised for operation').
```

Authorisation: Defence in Depth



Configuration: Operation Access

```
create SecurityRole.
Name        = 'ShoppingCart.Service.UpdateCart'.
Creator     = 'jane@app-admin.system'.
Description = 'Allows access to Update Shopping Cart operation'.

create SecurityRole.
Name        = 'ShoppingCart.Service.CreateCart'.
Creator     = 'jane@app-admin.system'.
Description = 'Allows access to Create Shopping Cart operation'.

create SecurityRole.
Name        = 'ShoppingCart.Service.DeleteCart'.
Creator     = 'jane@app-admin.system'.
Description = 'Allows access to Delete Shopping Cart operation'.

create SecurityRole.
Name        = 'ShoppingCart.Service.ReadCart'.
Creator     = 'jane@app-admin.system'.
Description = 'Allows access to Read Shopping Cart operation'.
```

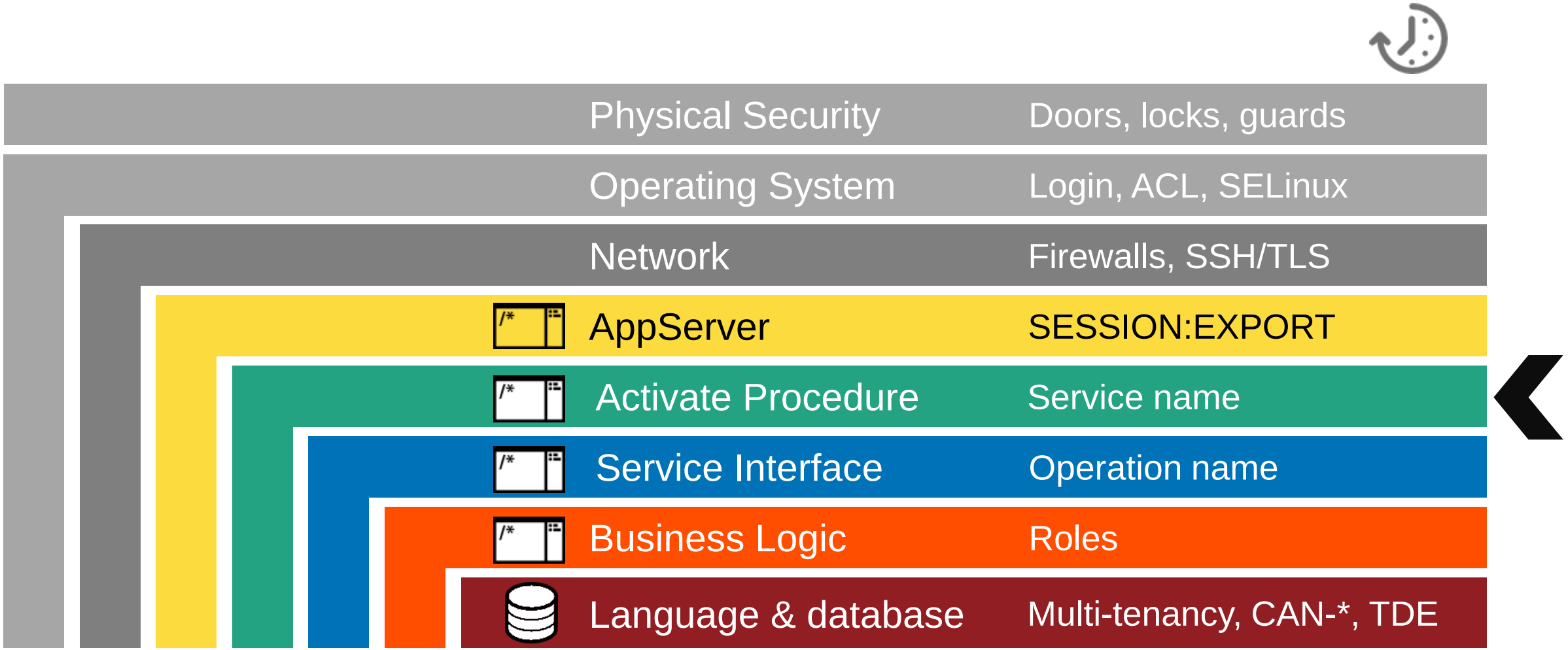
Using Operation Access

```
/* BusinessLogic/GetShoppingCart.p */
routine-level on error undo, throw.

{dsShoppingCart.i}

procedure UpdateShoppingCartService:
  define input-output parameter dataset for dsShoppingCart.

  Security.AuthorisationService:AuthoriseOperation(
        'ShoppingCart.Service.UpdateCart').

  ShoppingCartService:Instance:UpdateShoppingCartService(
      input-output dataset dsShoppingCart by-reference).
end procedure.
```

Roles & Responsibilities

	Anonymous	Customer	Employee	System
See catalogue	X	X	X	
Modify catalogue			X	
Update shopping cart		X	X	
Add users			X	
Dump & load data				X
Provision services				X
Level of trust	■ □ □	■ ■ □	■ ■ □	■ ■ ■

Authorisation: Defence in Depth

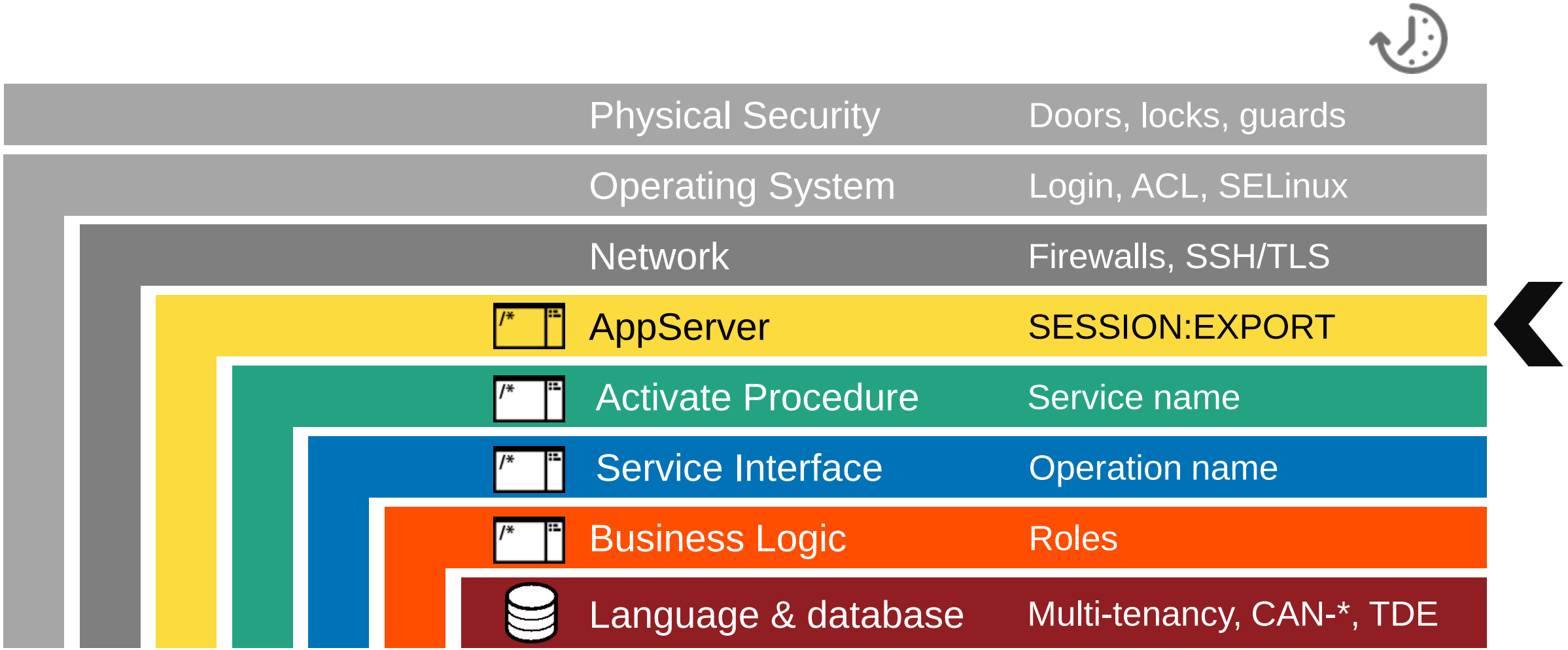


Configuration: Service Names

```
SecurityRole.Name        = 'ShoppingCart.Service.Access'.
SecurityRole.Description = 'Allows access to the ShoppingCart service'.

SecurityRole.Name        = 'Customer.Service.Access'.
SecurityRole.Description = 'Allows access to the Customer service'.
```

```
define private temp-table ttService no-undo
  field Service as character
  field Role    as character

create ttService.
Service = "BusinessLogic/GetShoppingCart.p".
Role    = "ShoppingCart.Service.Access".

create ttService.
Service = "BusinessLogic/GenericFetchData.p".
Role    = "ShoppingCart.Service.Access".
```

Authorisation: Defence in Depth



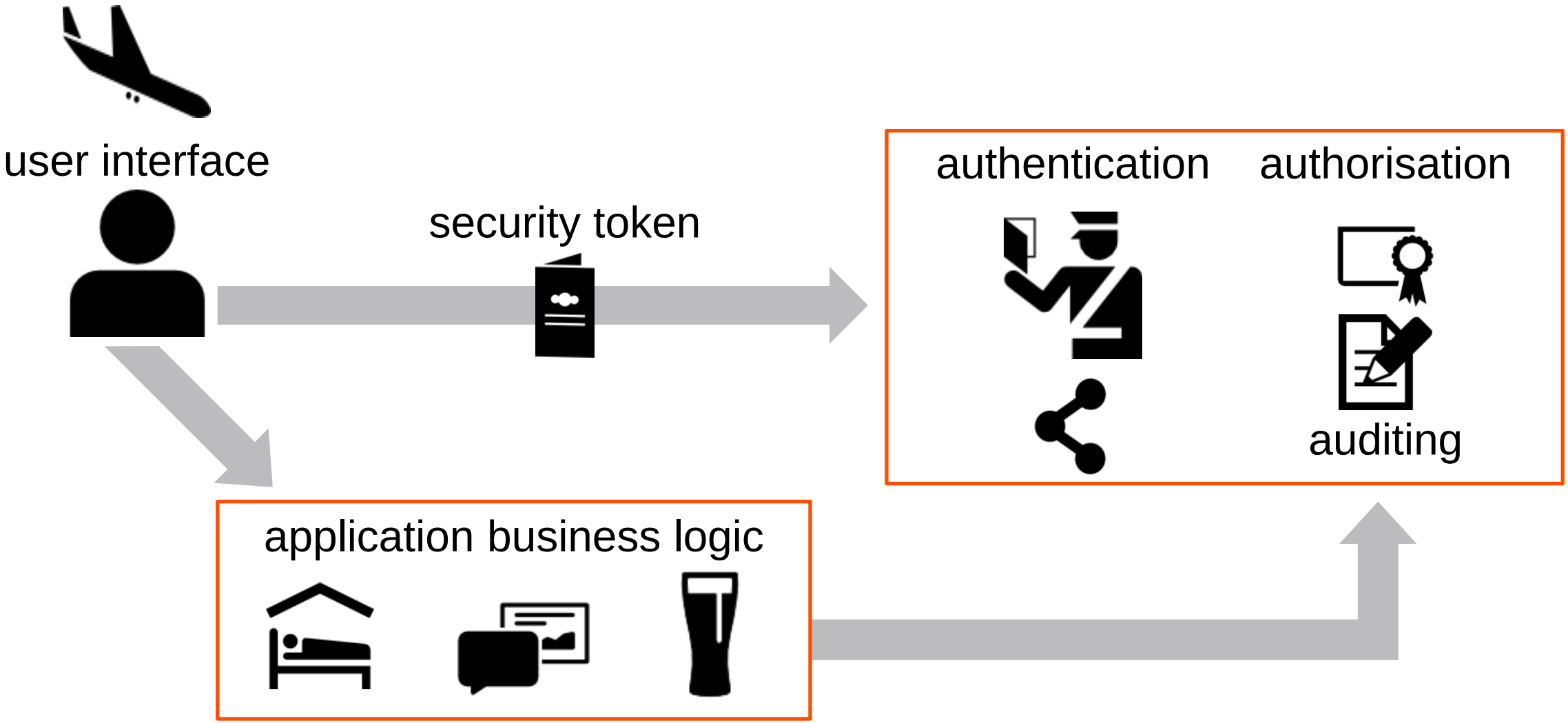
AppServer Access

```
ttService.Service = "BusinessLogic/GetShoppingCart.p".
ttService.Role    = "ShoppingCart.Service.Access".

ttService.Service = "BusinessLogic/GenericFetchData.p".
ttService.Role    = "ShoppingCart.Service.Access".
```

```
for each ttService break by Service:
    if first-of(ttService.Service) then
        assign cExportList = cExportList
                           + ','
                           + ttService.Service.
end.

session:export(cExportList).
```

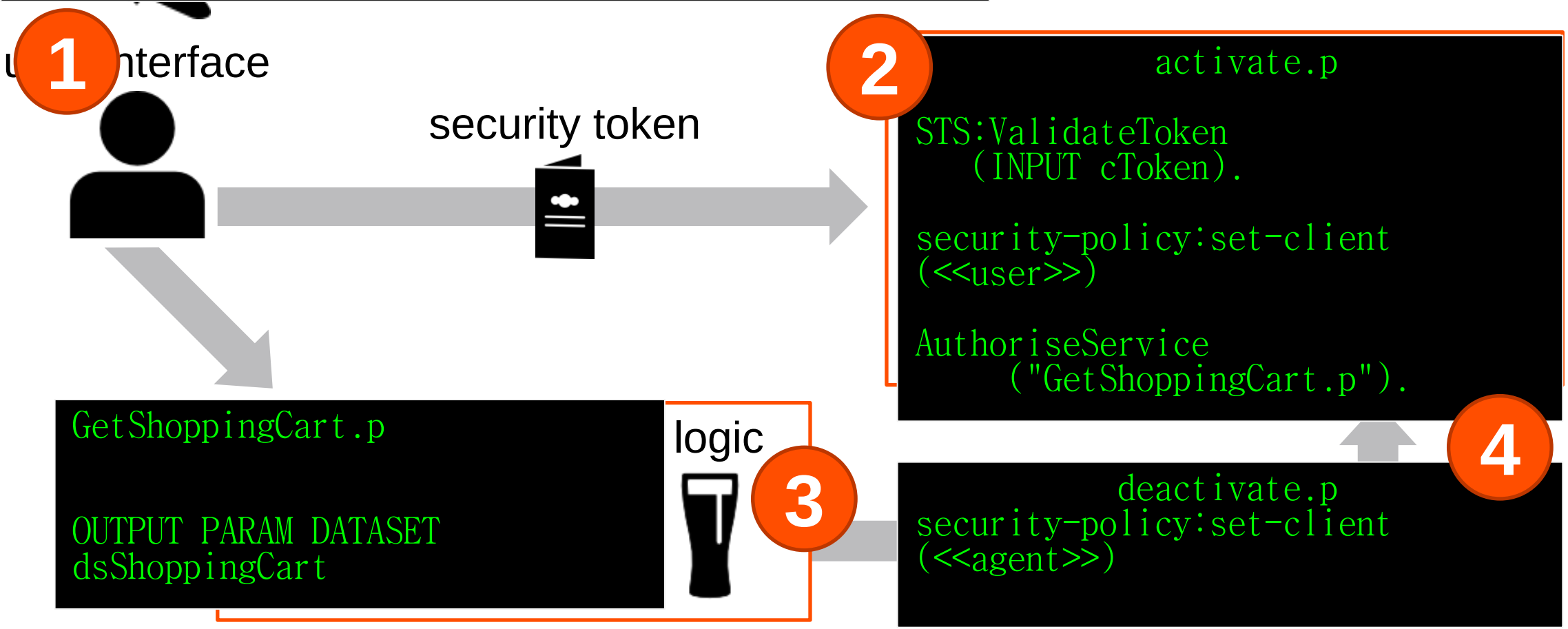
OE ~9.0+ | OE 11.3+

Application Flow: Business Logic

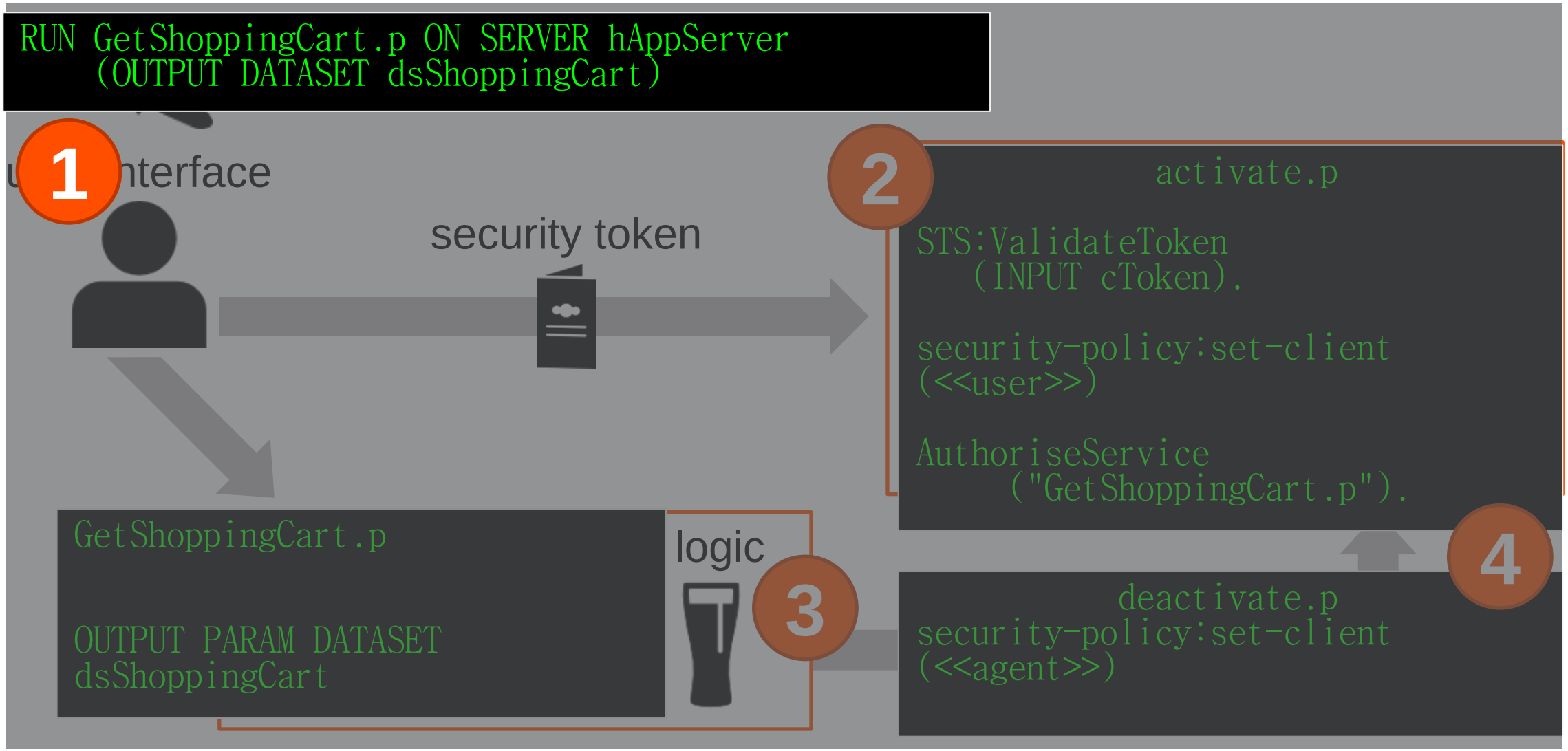


Application Flow: Business Logic

```
RUN GetShoppingCart.p ON SERVER hAppServer
    (OUTPUT DATASET dsShoppingCart)
```



Application Flow: Business Logic



Desktop.MainForm.cls

```
method protected void RefreshCart():
  define variable hAppServer as handle no-undo.

  run BusinessLogic/GetShoppingCart.p on hAppServer
                          (input this-object:CustNum,
                           output dataset dsShoppingCart).

  open query qryShoppingCart preselect
     each ttShoppingCart.

  bsShoppingCart:Handle = query qryShoppingCart:handle.

  query qryShoppingCart:reposition-to-row(1).
end method.
```

Application Flow: Business Logic

```
RUN GetShoppingCart.p ON SERVER hAppServer
    (OUTPUT DATASET dsShoppingCart)
```

1

nterface

2

```
activate.p
```

```
/* pseudo-code as run by AppServer agent */
if not can-do(session:export,
          request-info:ProcedureName) then
    undo, throw new Error().
```

```
set-client
```

```
AuthoriseService
    ("GetShoppingCart.p").
```

```
GetShoppingCart.p

OUTPUT PARAM DATASET
dsShoppingCart
```

logic

3

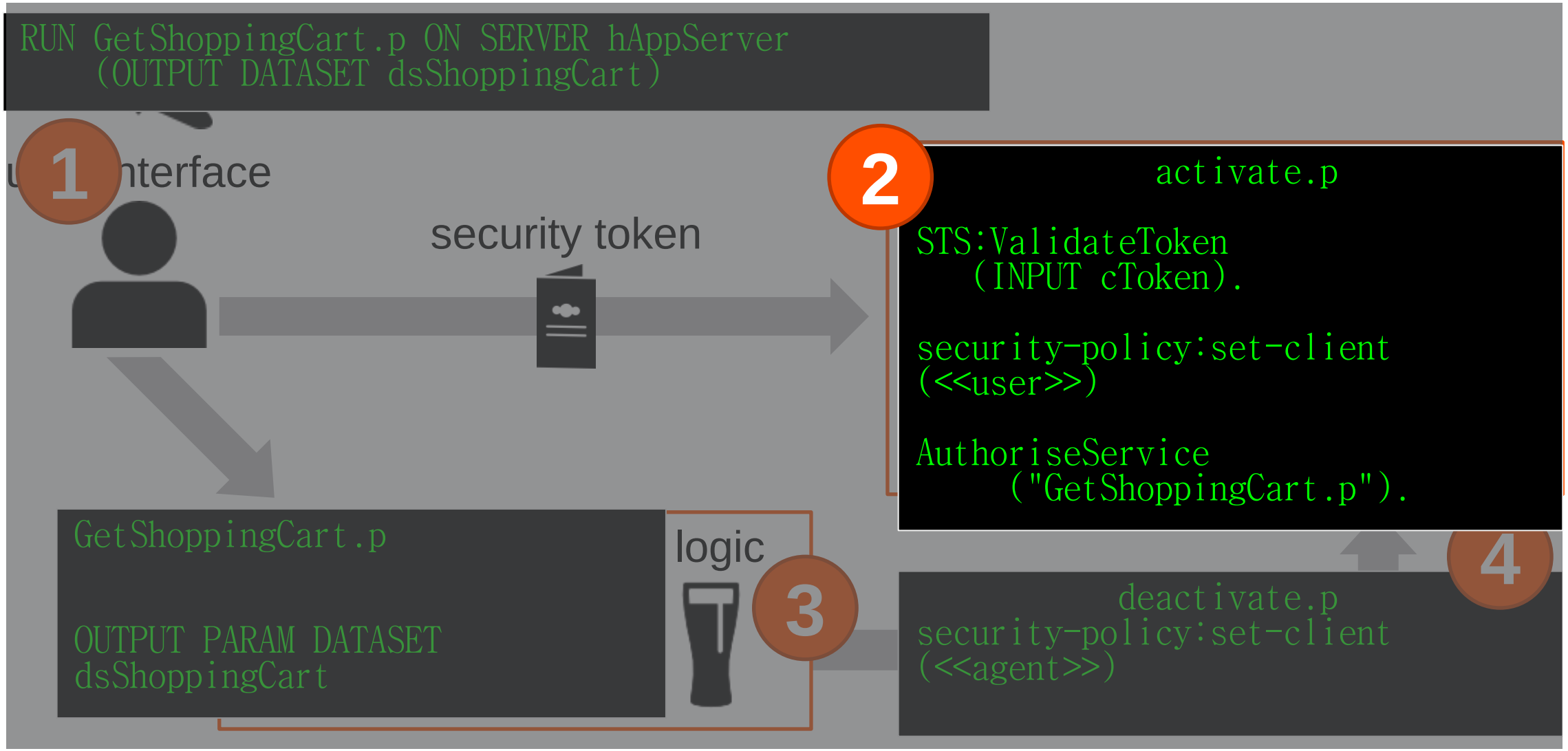
4

```
deactivate.p
security-policy:set-client
(<<agent>>)
```

AppServer

SESSION:EXPORT

Application Flow: Business Logic



Security/Activate.p

```
hClientPrincipal = Security.SecurityTokenService:Instance:
    GetClientPrincipal(
            session:current-request-info:ClientContextId).

/* authenticate client-principal */
security-policy:set-client(hClientPrincipal).

/* authorise service access */
Security.AuthorisationService:Instance
    :AuthoriseService(
        hClientPrincipal,
        session:current-request-info:ProcedureName).
```

Security/Activate.p

```
hClientPrincipal = Security.SecurityTokenService:Instance:
    GetClientPrincipal(
         session:current-request-info:ClientContextId).

/* authenticate client-principal */
security-policy:set-client(hClientPrincipal).

/* authorise service access */
Security.AuthorisationService:Instance
    :AuthoriseService(
        hClientPrincipal,
        session:current-request-info:ProcedureName).
```

Security/Activate.p

```
hClientPrincipal = Security.SecurityTokenService:Instance:
    GetClientPrincipal(
         session:current-request-info:ClientContextId).

/* authenticate client-principal */
security-policy:set-client(hClientPrincipal).

/* authorise service access */
Security.AuthorisationService:Instance
    :AuthoriseService(
        hClientPrincipal,
        session:current-request-info:ProcedureName).
```

Activate Procedure

Service name

Security.AuthorisationService

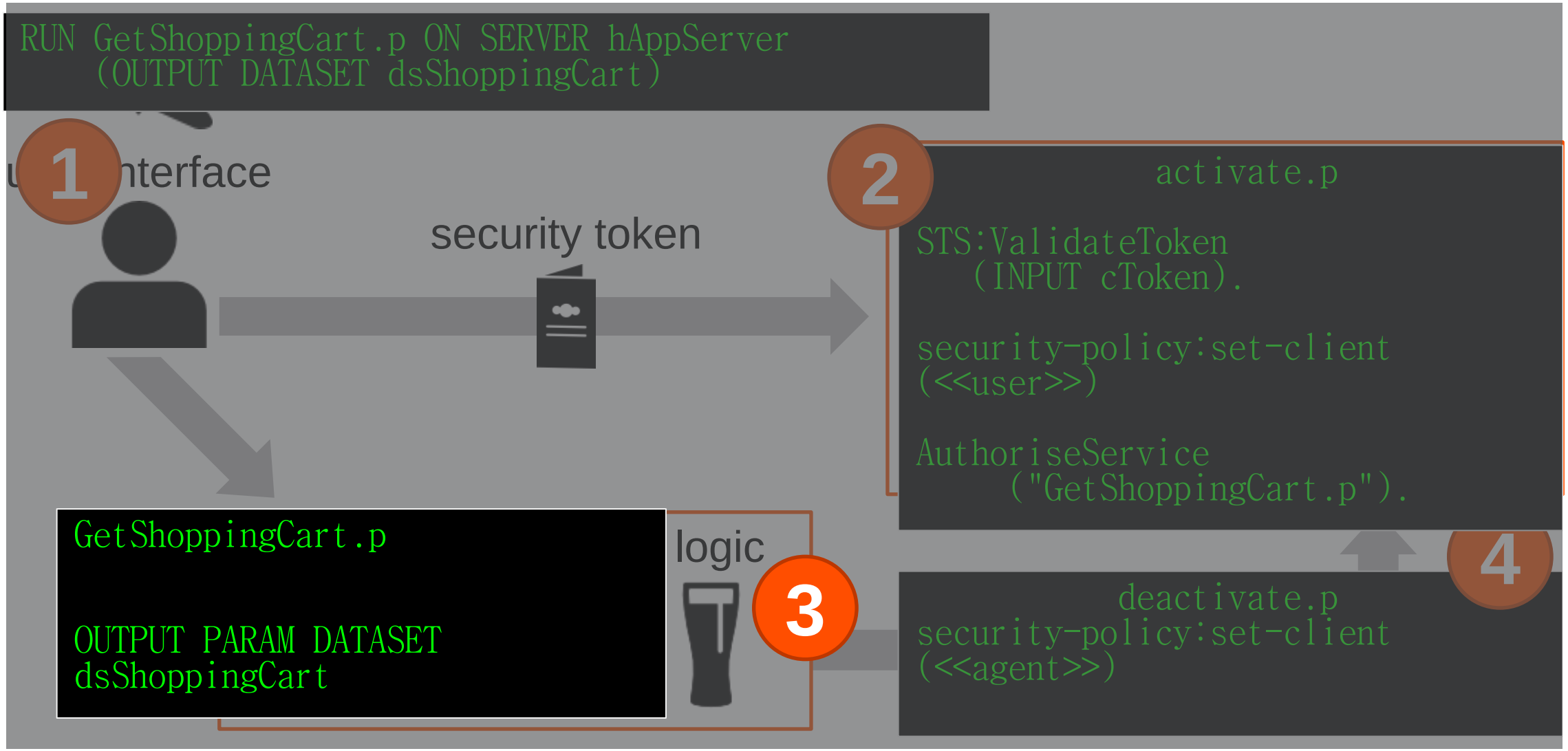
```
ttService.Service = "BusinessLogic/GetShoppingCart.p".
ttService.Role    = "ShoppingCart.Service.Access".
ttService.Service = "BusinessLogic/GenericFetchData.p".
ttService.Role    = "ShoppingCart.Service.Access".
```

```
method public void AuthoriseService(
      input phCP as handle,      /* client-principal */
      input pcService as char): /* BusinessLogic/GetShoppingCart.p */
  define variable lIsAuthorised as logical no-undo.

  lIsAuthorised = can-find(
      first ttService where ttService.Service eq pcServiceName).

  for each ttService where ttService.Service eq pcServiceName
      while lIsAuthorised:
    lIsAuthorised = not can-do(phCP:roles, ttService.Role).
  end.

  if not lIsAuthorised then
    undo, throw new AppError("User not authorised for service").
end method
```

Activate Procedure

Service name

Application Flow: Business Logic



BusinessLogic/GetShoppingCart.p

```
{BusinessLogic/dsShoppingCart.i}

define input parameter piCustNum as integer.
define output parameter dataset for dsShoppingCart.

define variable oBusinessEntity as ShoppingCartBE no-undo.

Security.AuthorisationService:Instance
    :AuthoriseOperation("ShoppingCart.Service.ReadCart").

oBusinessEntity  = new ShoppingCartBE().

oBusinessEntity:GetCart(
    input piCustNum, output dataset dsShoppingCart).

/* eof */
```

Service Interface

Operation name

Security.AuthorisationService

```
method public void AuthoriseOperation(
                                input pcOperation as character):
  define variable hCP as handle no-undo.

  hCP = security-policy:get-client().

  if not can-do(hCP:roles, pcOperation) then
    undo, throw new AppError("User not authorised for service").
end method.
```

Service Interface

Operation name

BusinessLogic/GetShoppingCart.p

```
{BusinessLogic/dsShoppingCart.i}

define input parameter piCustNum as integer.
define output parameter dataset for dsShoppingCart.

define variable oBusinessEntity as ShoppingCartBE no-undo.

Security.AuthorisationService:Instance
    :AuthoriseOperation("ShoppingCart.Service.ReadCart").

oBusinessEntity  = new ShoppingCartBE().

oBusinessEntity:GetCart(
    input piCustNum, output dataset dsShoppingCart).

/* eof */
```

ShoppingCartBE.cls

```
{BusinessLogic/dsShoppingCart.i}

method public void GetCart(
             input parameter piCustNum as integer,
             output parameter dataset dsShoppingCart):
  define data-source srcCart for ShoppingCart.

  Security.AuthorisationService:Instance
    :AuthoriseOperation("ShoppingCart.Data.Read").

  data-source srcCart:fill-where-string =
        'where CustNum eq ' + quoter(piCustNum).

  buffer ttShoppingCart:attach-data-source(data-source srcCart:handle).

  /* multi-tenancy magic happens here. CAN-READ too.

     Based on the asserted user via SECURITY-POLICY:GET-CLIENT */
  dataset dsShoppingCart:fill().

  buffer ttSh
end method.
```

Business Logic

Roles

ShoppingCartBE.cls

```
{BusinessLogic/dsShoppingCart.i}

method public void GetCart(
             input parameter piCustNum as integer,
             output parameter dataset dsShoppingCart):
  define data-source srcCart for ShoppingCart.

  Security.AuthorisationService:Instance
    :AuthoriseOperation("ShoppingCart.Data.Read").

  data-source srcCart:fill-where-string =
        'where CustNum eq ' + quoter(piCustNum).

  buffer ttShoppingCart:attach-data-source(data-source srcCart:handle).

  /* multi-tenancy magic happens here. CAN-READ too.

     Based on the asserted user via SECURITY-POLICY:GET-CLIENT */
  dataset dsShoppingCart:fill().

  buffer ttShoppingCart:detach-data-source()
end method.
```

Language & database

Multi-tenancy, CAN-*, TDE

Desktop.MainForm.cls

```
method protected void RefreshCart():
  define variable hAppServer as handle no-undo.

  run BusinessLogic/GetShoppingCart.p on hAppServer
                          (input this-object:CustNum,
                           output dataset dsShoppingCart).

  open query qryShoppingCart preselect
     each ttShoppingCart.

  bsShoppingCart:Handle = query qryShoppingCart:handle.

  query qryShoppingCart:reposition-to-row(1).
end method.
```

Application Security Principles

Applications must have security designed in. Some proven application security principles

1. **Identify and secure the weakest link**
2. **Practice defense in depth**
3. Be reluctant to trust
4. Remember that hiding secrets is hard
5. **Follow the principle of least privilege**
6. **Fail and recover securely**
7. Compartmentalize
8. Keep it simple, stupid
9. Keep trust to yourself
10. **Assume nothing**

Gary McGraw's 10 steps to secure software

Summary

- Identity management is a process that helps protect your business data
  - Strength in depth
- OpenEdge provides components of identity management
  - CLIENT-PRINCIPAL
  - Multi-tenancy, Domains & Authentication Systems
- Run with least privilege
  - Use Domains and Roles to keep privileges ‘tight’
  - Reset to lower privileges when done
- Configuration > Code
  - Code is the weakest link

Extra Materials

This session

- Slides to be posted on Exchange website

Other Exchange sessions

- Identity Management Basics (Part 1)
  Peter Judge
- Workshop: Progress OpenEdge Security
  Brian Bowman, Rob Marshall et al
- Transparent Data Encryption
  Doug Vanek
- Introduction to Multi-tenancy
  Gus Bjorklund
- Security and Session Management with Mobile Devices
  Mike Jacobs & Wayne Henshaw

Image Credits:
Passport designed by Catia G, Time designed by wayne25uk, Database designed by Anton Outkine, Code designed by Nikhil Dev, Ninja designed by John O'Shea, Imposter designed by Luis Prado, User designed by T. Weber, Fingerprint designed by Andrew Forrester, Document designed by Samuel Green, Certificate designed by VuWorks, Network designed by Ben Rex Furneaux, Beer designed by Leigh Scholten; all from The Noun Project

PROGRESS