Identity Management Basics

Part 1 of Identity Management with OpenEdge

Peter Judge
OpenEdge Development
pjudge@progress.com

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

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

This is Nothing New

Forces aligned against you are more prevalent, and they have more, and more sophisticated weapons

And you’ve given people a door and invitation via the internet

So now the things you used to do are no longer adequate

What is Identity Management?

It’s about protecting your business data by

▪ Controlling and verifying who accesses your data	Authentication
▪ Controlling what they can do with your data	Authorisation
▪ Reviewing what they did with your data	Auditing
▪ Maintaining information about your users	Administration

Authentication

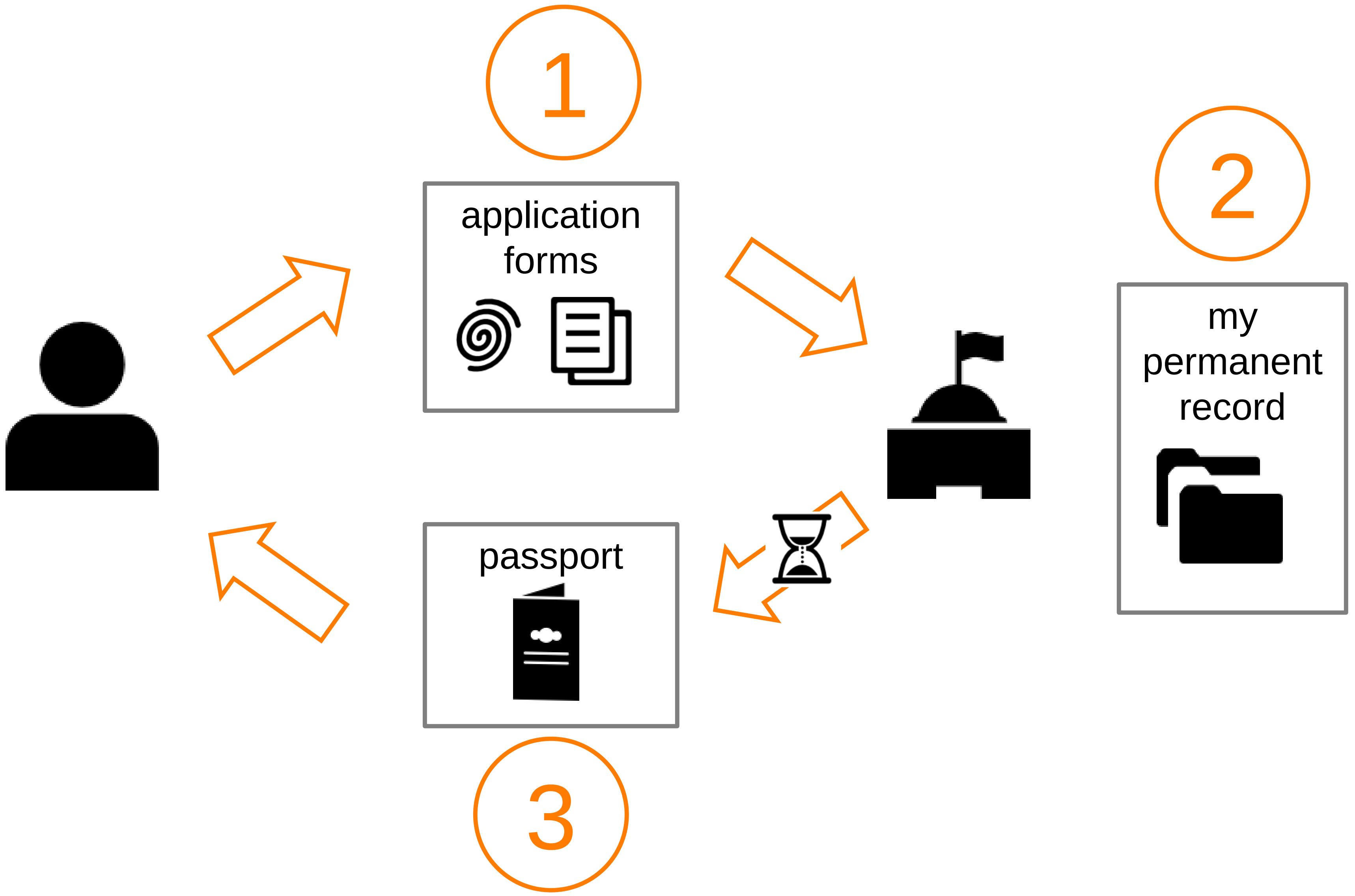
- Identifies a user, using factors
  - Something the user knows (e.g. password)
  - Something the user has (e.g. security token)
  - Something of the user (e.g. biometric)
- Verify that users are who they say they are
  - We need to be able to trust this fact, as do others

Authorisation & Auditing

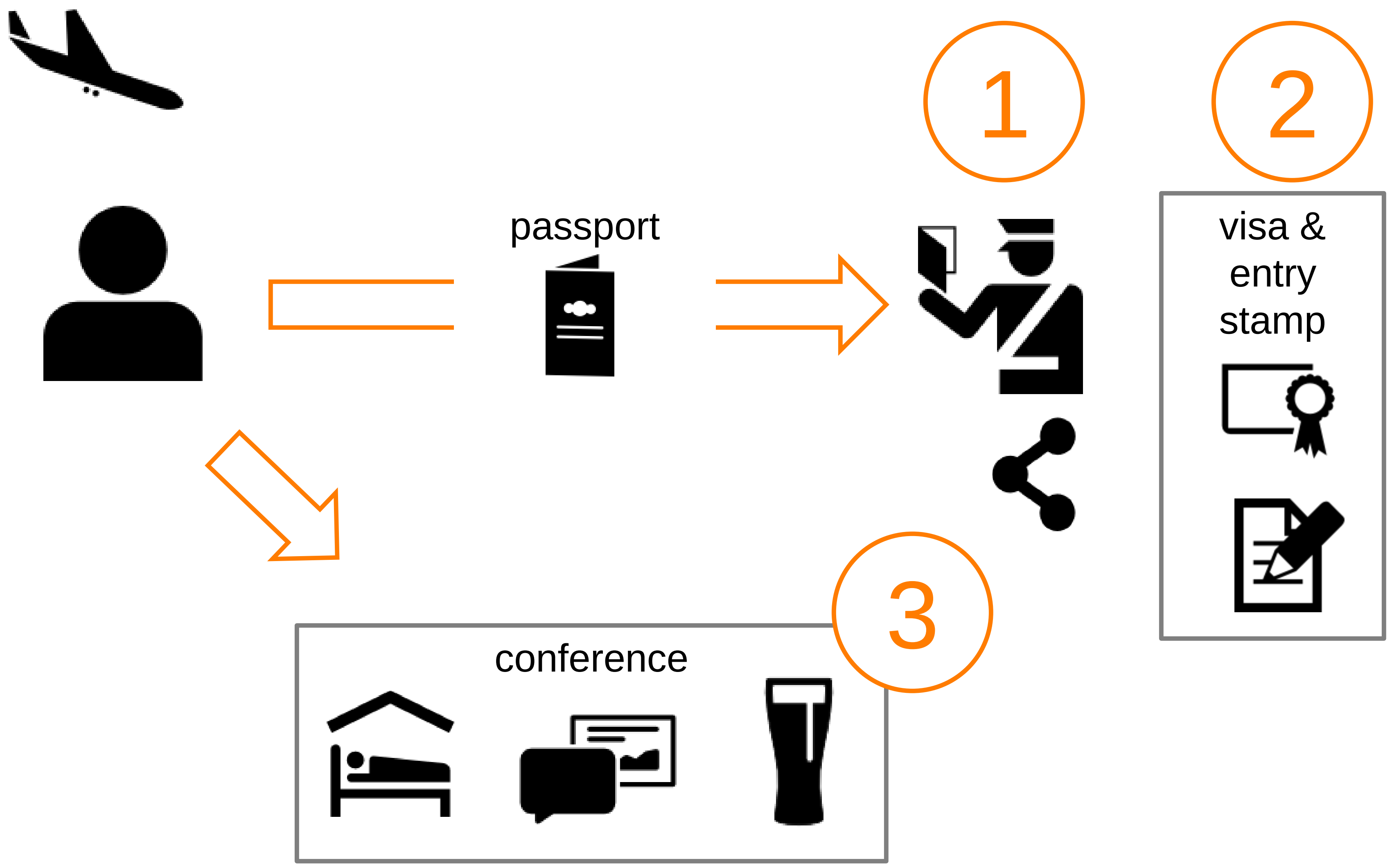
- Authorisation
  - What services can the user access?
  - What data can the user see and/or modify?
    - Multi-tenancy
    - Record-level, field-level

- Auditing
  - Verifiable trace of a user’s actions

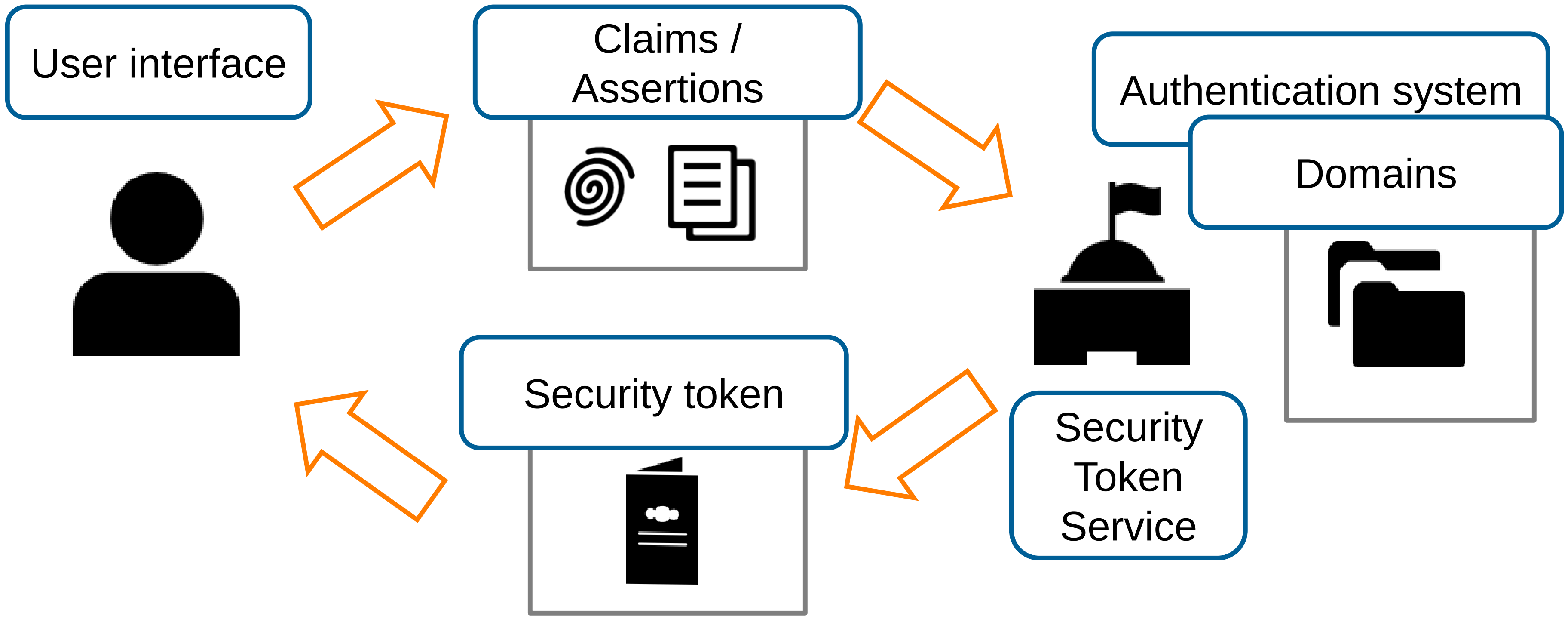
Getting a Passport



Using a Passport



Application Architecture



What is a Security Token?

- A transportable block of data that can be used as proof of user identity by any systems or applications that have a trust relationship with the originator of the security token
  - Exists for same reason passports do: so that a gatekeeper doesn’t have to ask you for everything every time you want to pass
- Enables Single Sign On (SSO)
  - Authenticate once and allow access many times across (ABL) processes
- Secure, time sensitive and data-integrity protected

The ABL CLIENT-PRINCIPAL

- CLIENT-PRINCIPAL = ABL security token

```
CREATE CLIENT-PRINCIPAL hCP.
hCP:INITIALIZE(<args>)
```

- Sets current identity in any connected db or AVM session

```
SECURITY-POLICY:SET-CLIENT(hCP).
SET-DB-CLIENT(<dbname>, hCP).
```

- AVM creates if not created explicitly

```
SETUSERID(<userid>, <pass>, <dbname>).
cmd> $PROEXE –U <userid> -P <pass>
```

- Manage a user's login session

```
hCP = SECURITY-POLICY:GET-CLIENT().
rCP = hCP:EXPORT-PRINCIPAL.
hCP:LOGOUT().
```

OE 10.1A+

What Are Domains?

- A group of users with a common set of
  - Roles and responsibilities
  - Level of security
  - Data access privileges

- Configured in db meta-schema

```
_sec-authentication-domain

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

Authentication Systems (aka Plug-ins)

- Validates requesting entity’s claims
  - Full user login (i.e. user authentication), or
  - Single Sign-On (SSO)
- Specifies actual means of performing authentication
  - ABL callbacks available for user-defined systems

OE 11.1+

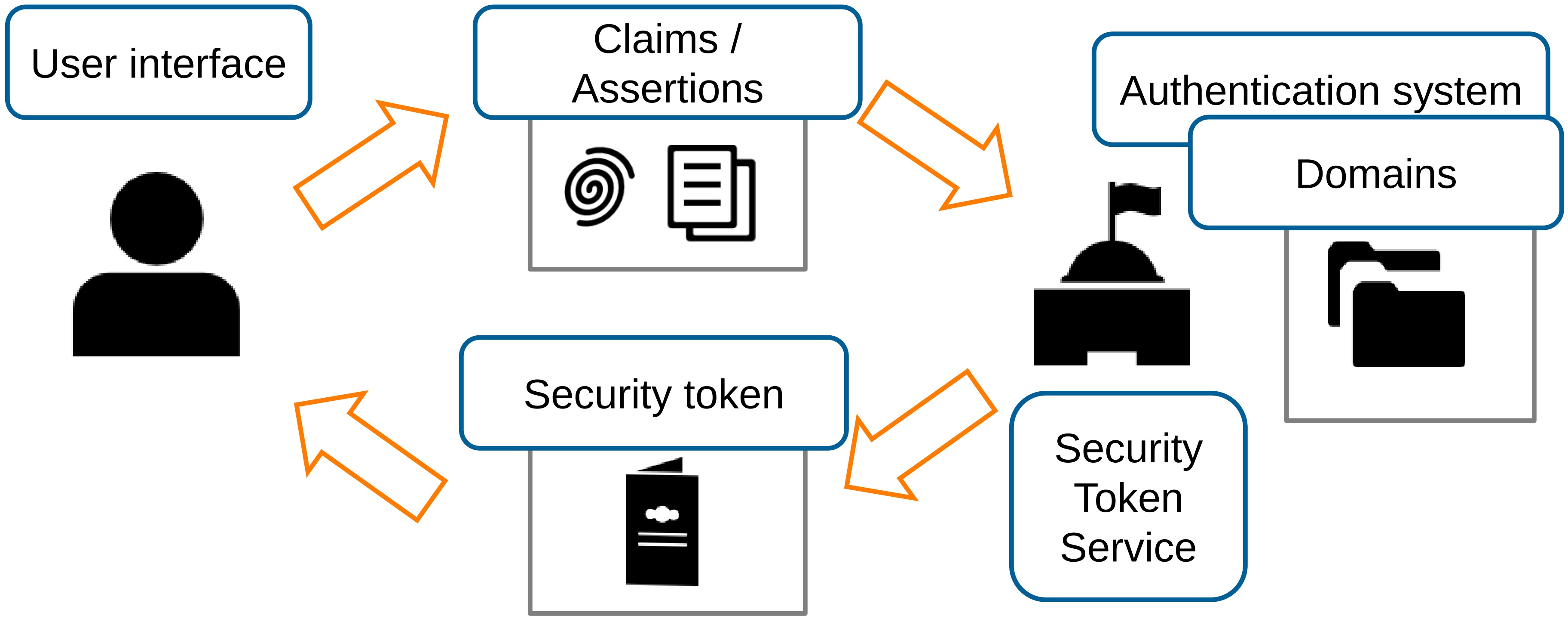
```
_sec-authentication-system

  _Domain-type
  _Domain-type-description
  _PAM-plug-in
  _PAM-callback-procedure
```

- Single authentication system can support multiple domains
  - One domain has one authentication system

User Credentials Example Schema

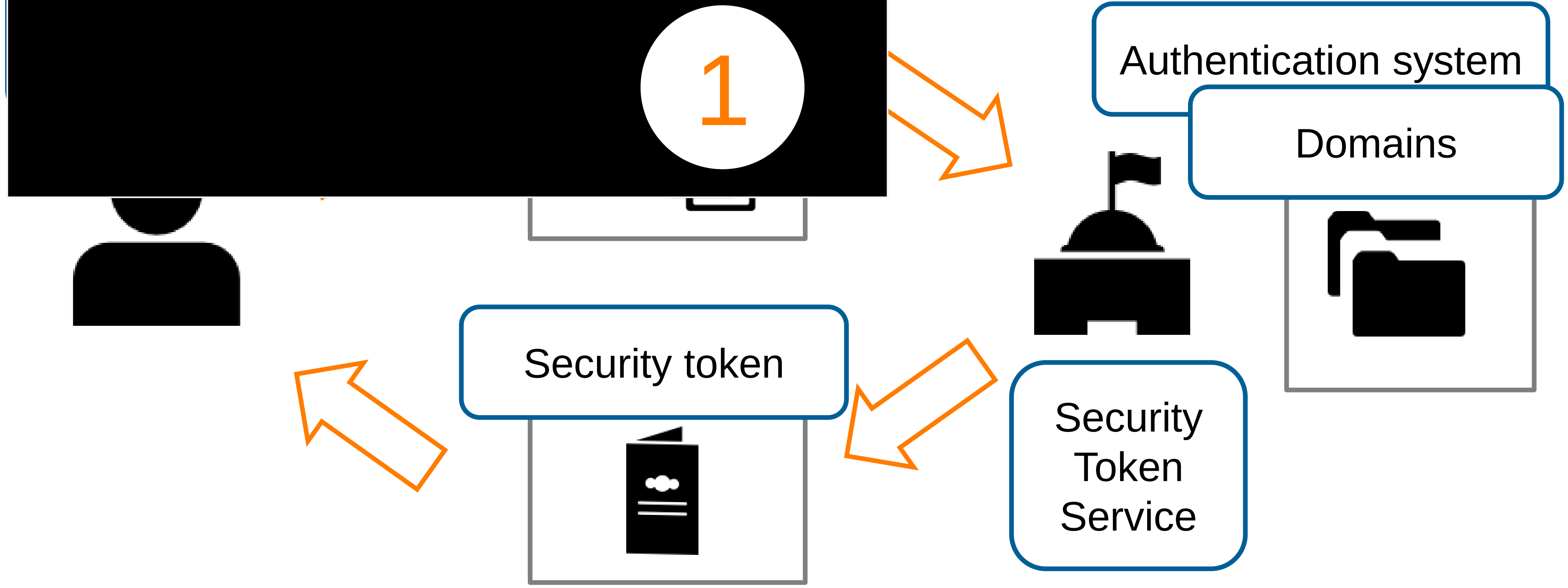
```
ADD TABLE "ApplicationUser"
  AREA "Data"
  DESCRIPTION "The application's user table. Contains login names,
passwords and mappings to login domains."
  DUMP-NAME "applicationuser"


ADD FIELD "LoginName" AS character
/* Domain necessary for re-use */
ADD FIELD "LoginDomain" AS character
ADD FIELD "Password" AS character
ADD FIELD "LastLoginDate" AS datetime-tz
/* Last login IP address / host  */
ADD FIELD "LastLoginLocation" AS character

ADD INDEX "Login" ON "ApplicationUser"
  AREA "Indexes"
  UNIQUE
  INDEX-FIELD "LoginName" ASCENDING
  INDEX-FIELD "LoginDomain" ASCENDING
```

Application Architecture: Login

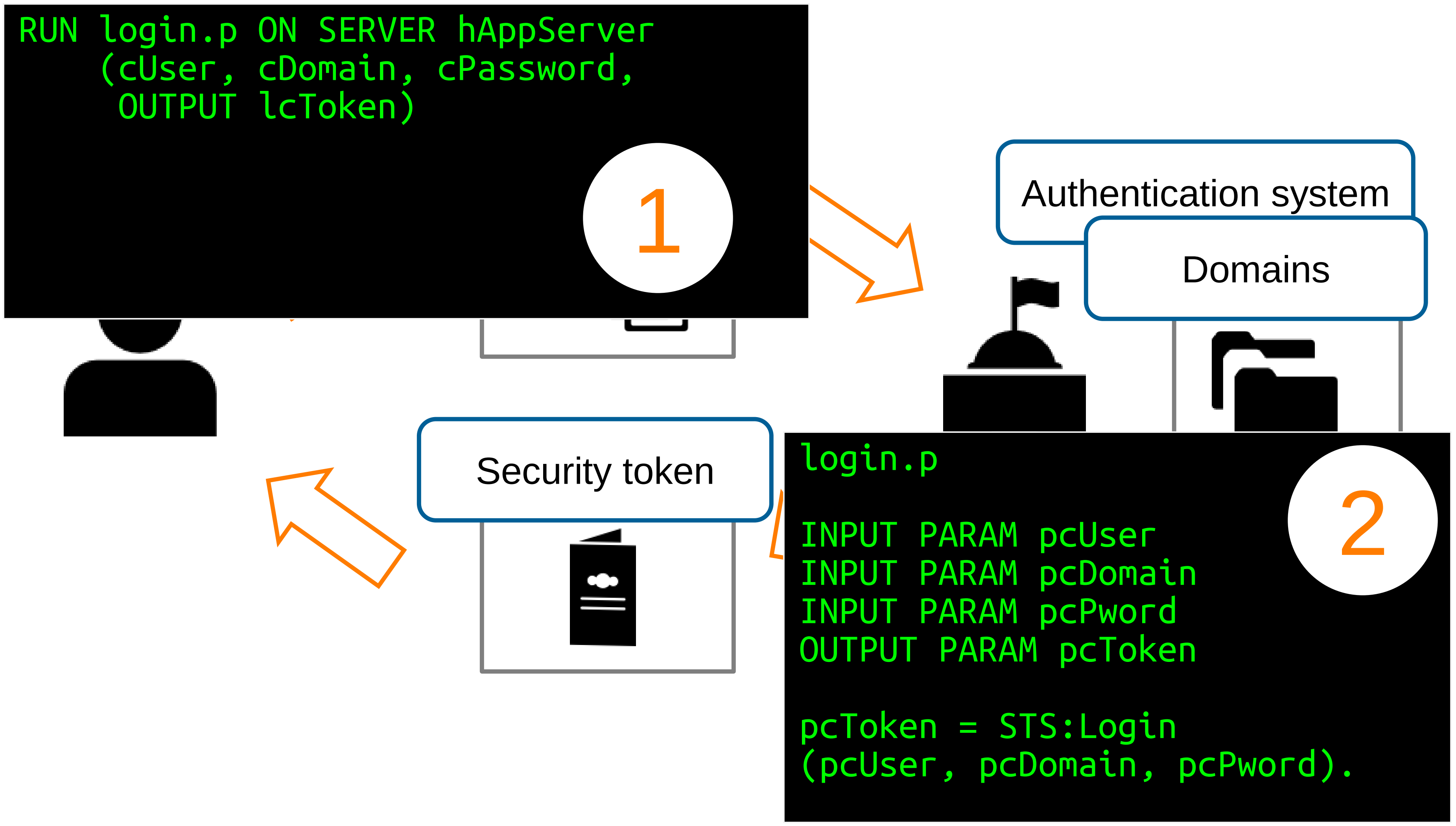


Application Architecture: Login

```
RUN login.p ON SERVER hAppServer
    (cUser, cDomain, cPassword,
     OUTPUT lcToken)
```



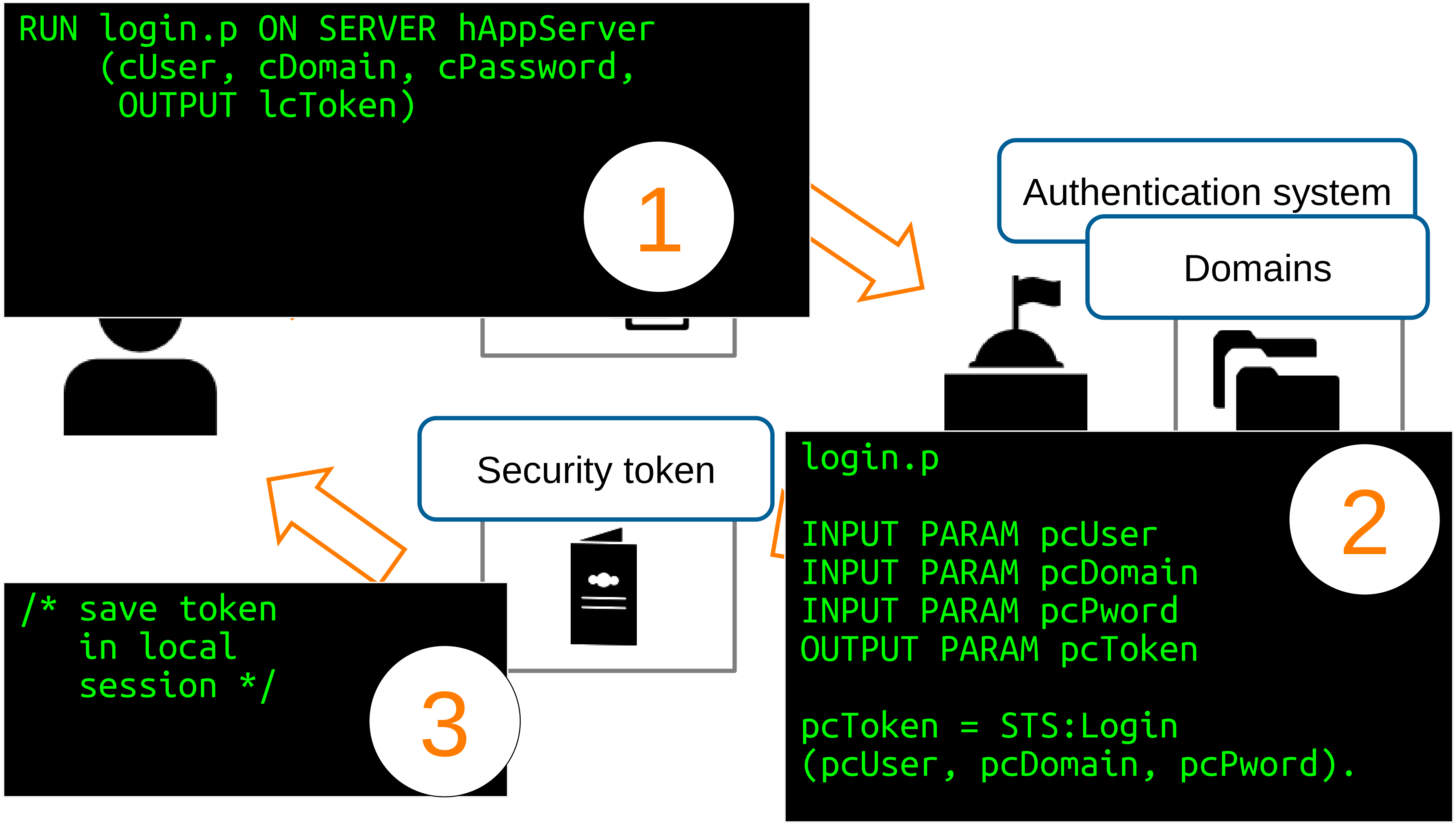
Application Architecture: Login



```
RUN login.p ON SERVER hAppServer
    (cUser, cDomain, cPassword,
     OUTPUT lcToken)
```

```
login.p

INPUT  PARAM pcUser
INPUT  PARAM pcDomain
INPUT  PARAM pcPword
OUTPUT PARAM pcToken

pcToken = STS:Login
(pcUser, pcDomain, pcPword).
```

Application Architecture: Login

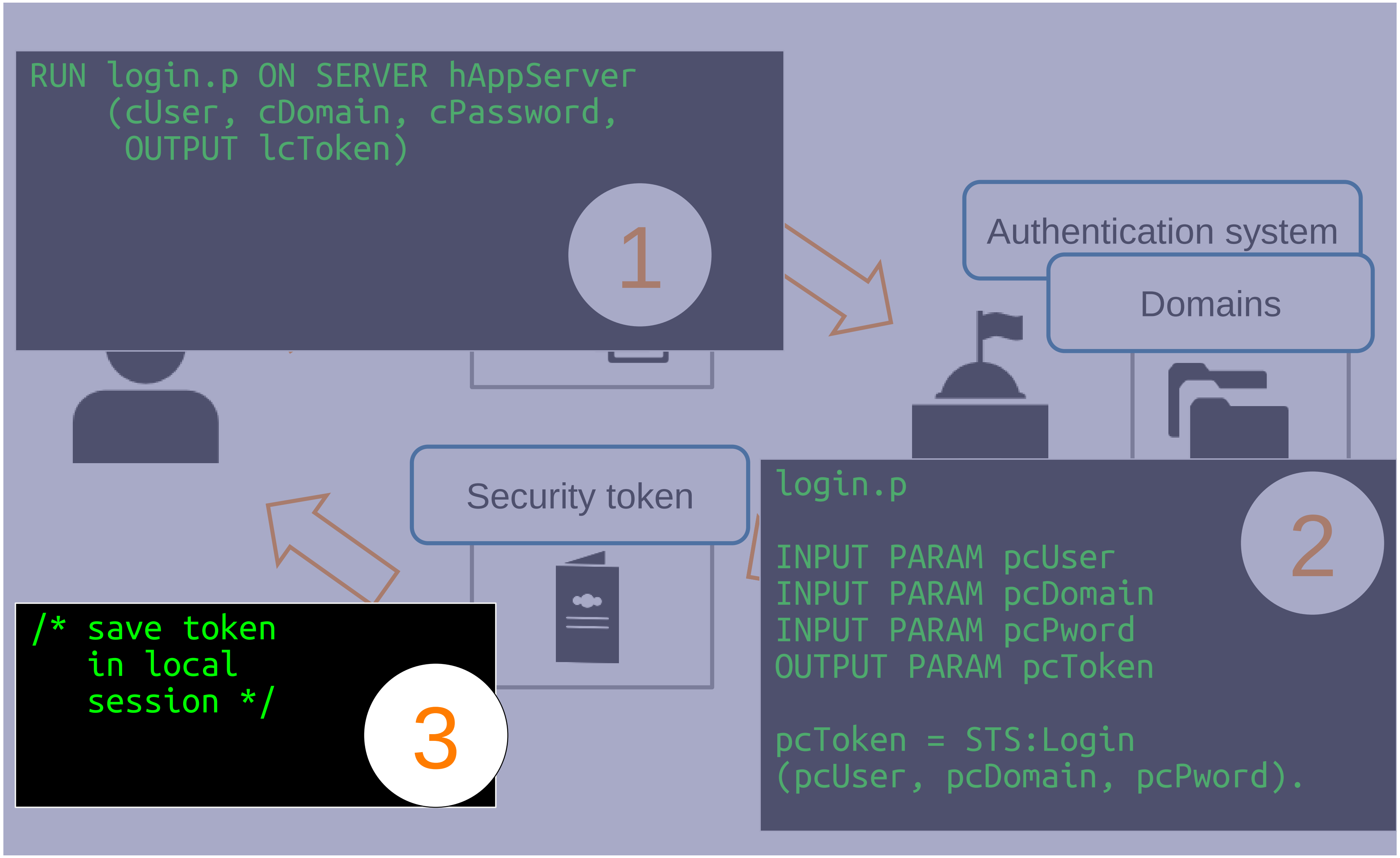


```
RUN login.p ON SERVER hAppServer
     (cUser, cDomain, cPassword,
      OUTPUT lcToken)
```

```
login.p

INPUT PARAM pcUser
INPUT PARAM pcDomain
INPUT PARAM pcPword
OUTPUT PARAM pcToken

pcToken = STS:Login
(pcUser, pcDomain, pcPword).
```

```
/* save token
   in local
   session */
```

Application Architecture: Login



```
RUN login.p ON SERVER hAppServer
    (cUser, cDomain, cPassword,
     OUTPUT lcToken)
```

```
login.p

INPUT PARAM pcUser
INPUT PARAM pcDomain
INPUT PARAM pcPword
OUTPUT PARAM pcToken

pcToken = STS:Login
(pcUser, pcDomain, pcPword).
```

```
/* save token
   in local
   session */
```

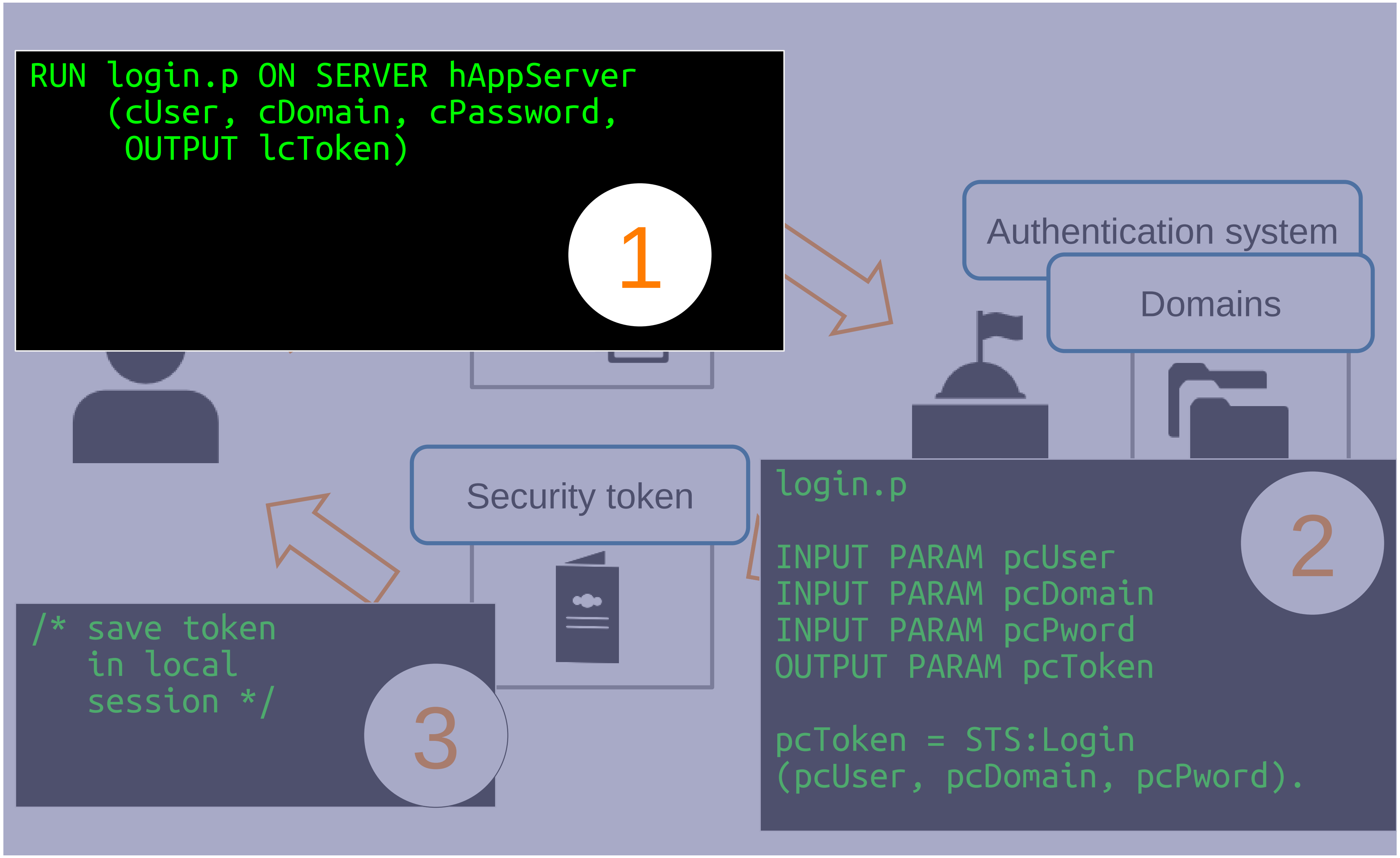
Managing Security Context

Client Context (OE 11.2+)	Server Context (OE 11.0+)
Entire security context for session in sealed C-P	Entire security context for session stored on server, using C-P's SESSION-ID as key ("CCID")
Sealed C-P moves between server and client	CCID moves between server and client. CCID used to find context in cache & rehydrate C-P
Server validates C-P & uses it to establish security context	Server validates C-P & uses it to establish security context
Used in stateful apps that run in stateless server environments	Used in stateful applications
More data transmitted per call = more overhead	Less data transmitted = lower overhead
Less secure, unless C-P encrypted or in SSL session	More secure, since C-P not at risk of exposure

```
rawToken = hCP:EXPORT-PRINCIPAL
```

```
charToken = hCP:SESSION-ID
```

Application Architecture: Login



```
RUN login.p ON SERVER hAppServer
    (cUser, cDomain, cPassword,
     OUTPUT lcToken)
```

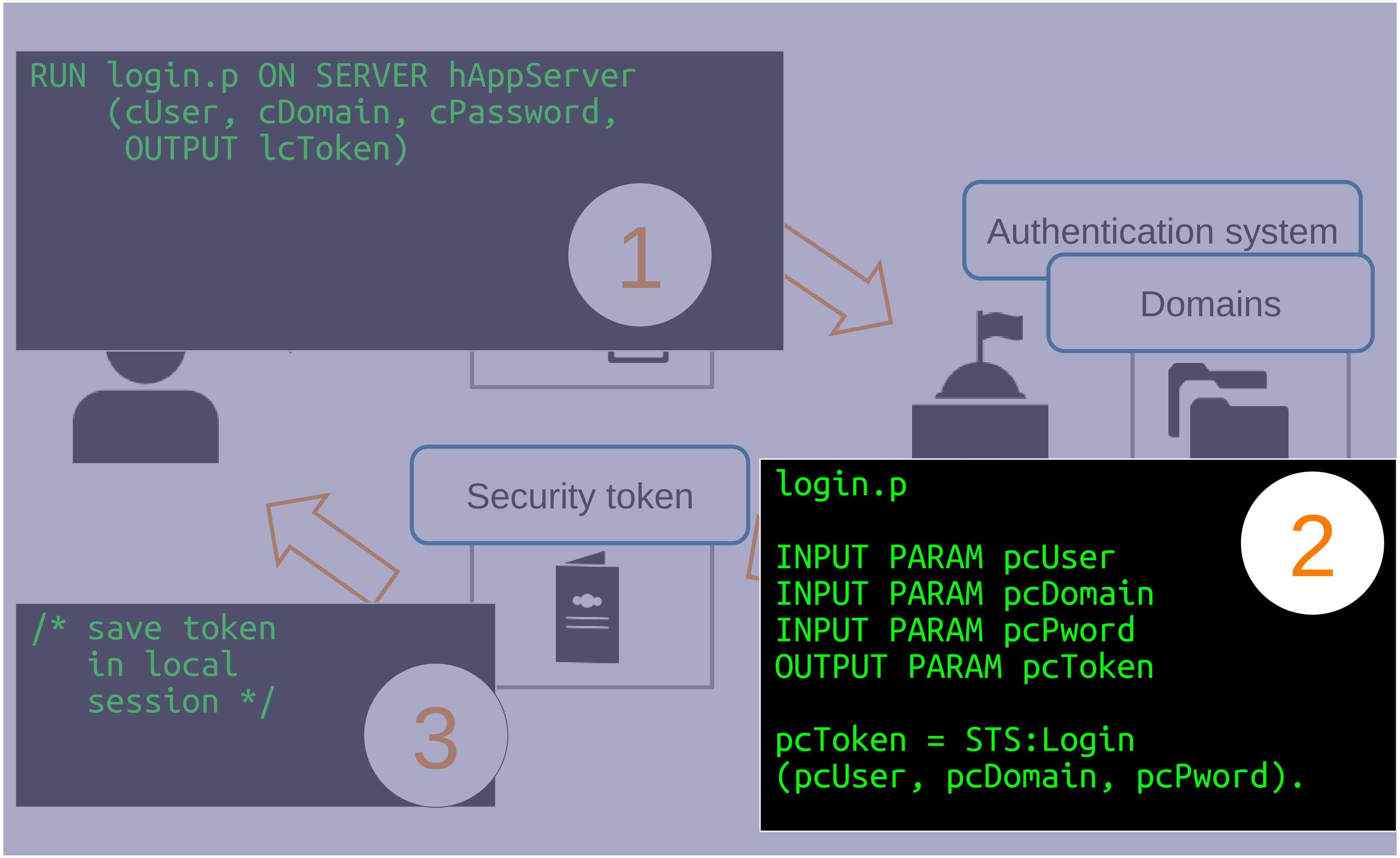
```
login.p

INPUT PARAM pcUser
INPUT PARAM pcDomain
INPUT PARAM pcPword
OUTPUT PARAM pcToken

pcToken = STS:Login
(pcUser, pcDomain, pcPword).
```

```
/* save token
   in local
   session */
```

Desktop.MainForm.cls

```
method public logical LoginUser(
              input pcUserName as char,
              input pcDomain as char,
              input pcPassword as char):

  run Security/Login.p on hAppServer (
          pcUserName, pcDomain, pcPassword,
          output cUserContextId).
  if cUserContextId eq '' then return false.

  /* set the CCID on the business logic server so that it's
     transported with every request. */
  hAppServer:request-info:ClientContextId = cUserContextId.

  return true.
end method.
```

Application Architecture: Login



```
RUN login.p ON SERVER hAppServer
    (cUser, cDomain, cPassword,
     OUTPUT lcToken)
```

```
login.p

INPUT PARAM pcUser
INPUT PARAM pcDomain
INPUT PARAM pcPword
OUTPUT PARAM pcToken

pcToken = STS:Login
(pcUser, pcDomain, pcPword).
```

```
/* save token
   in local
   session */
```

Security/Login.p

```
define input  parameter pcUser as character no-undo.
define input  parameter pcDomain as character no-undo.
define input  parameter pcPassword as character no-undo.
define output parameter pcToken as character no-undo.

pcToken = Security.SecurityTokenService:Instance
            :LoginUser(pcUser, pcDomain, pcPassword).
```

Security.SecurityTokenService.cls

```
method public char LoginUser(input pcUserName as char,
                             input pcUserDomain as char,
                             input pcPassword as char):
  define variable hClientPrincipal as handle no-undo.

  create client-principal hClientPrincipal.
  hClientPrincipal:initialize(
     substitute('&1@&2', pcUserName, pcUserDomain),
     ?,   /* unique session id */
     add-interval(now, 8, 'hours'),  /* login expiration */
     pcPassword).

  /* passes authentication work off to
     authentication system */
  security-policy:set-client(hClientPrincipal).

  /* writes security context into DB */
  WriteClientPrincipalToStore(hClientPrincipal).

  /* return character value */
  return hClientPrincipal:session-id.
end method.
```

Security.SecurityTokenService.cls

```
method public char LoginUser(input pcUserName as char,
                             input pcUserDomain as char,
                             input pcPassword as char):
  define variable hClientPrincipal as handle no-undo.

  create client-principal hClientPrincipal.
  hClientPrincipal:initialize(
     substitute('&1@&2', pcUserName, pcUserDomain),
     ?,   /* unique session id */
     add-interval(now, 8, 'hours'),  /* login expiration */
     pcPassword).

  /* passes authentication work off to
     authentication system */
  security-policy:set-client(hClientPrincipal).

  /* writes security context into DB */
  WriteClientPrincipalToStore(hClientPrincipal).

  /* return character value */
  return hClientPrincipal:session-id.
end method.
```

_sec-authentication-system

```
create _sec-authentication-system.
_Domain-type                 = 'TABLE-ApplicationUser'.
_Domain-type-description =
         'The ApplicationUser table serves as
          the authentication domain'.
_PAM-plug-in                 = true.

_PAM-callback-procedure   =
         'Security/AppUserAuthenticate.p'.
```

Security/AppUserAuthenticate.p

```
procedure AuthenticateUser:
  def input  param phClientPrincipal  as handle no-undo.
  def input  param pcSystemOptions as character extent no-undo.
  def output param piPAMStatus as integer init ? no-undo.
  def output param pcErrorMsg as character no-undo.

  find ApplicationUser where
       ApplicationUser.LoginName eq phCP:user-id and
       ApplicationUser.LoginDomain eq phCP:domain-name
       no-lock no-error.

  if not available ApplicationUser then
    piPAMStatus = Progress.Lang.PAMStatus:UnknownUser.
  else
  if ApplicationUser.Password ne
     encode(phCP:primary-passphrase) then
    piPAMStatus = Progress.Lang.PAMStatus:AuthenticationFailed.
  else
    /* we're good to go */
    piPAMStatus = Progress.Lang.PAMStatus:Success.

  return.
end procedure.
```

Security.SecurityTokenService.cls

```
method public char LoginUser(input pcUserName as char,
                             input pcUserDomain as char,
                             input pcPassword as char):
  define variable hClientPrincipal as handle no-undo.

  create client-principal hClientPrincipal.
  hClientPrincipal:initialize(
     substitute('&1@&2', pcUserName, pcUserDomain),
     ?,   /* unique session id */
     add-interval(now, 8, 'hours'),  /* login expiration */
     pcPassword).

  /* passes authentication work off to
     authentication system */
  security-policy:set-client(hClientPrincipal).

  /* writes security context into DB */
  WriteClientPrincipalToStore(hClientPrincipal).

  /* return character value */
  return hClientPrincipal:session-id.
end method.
```

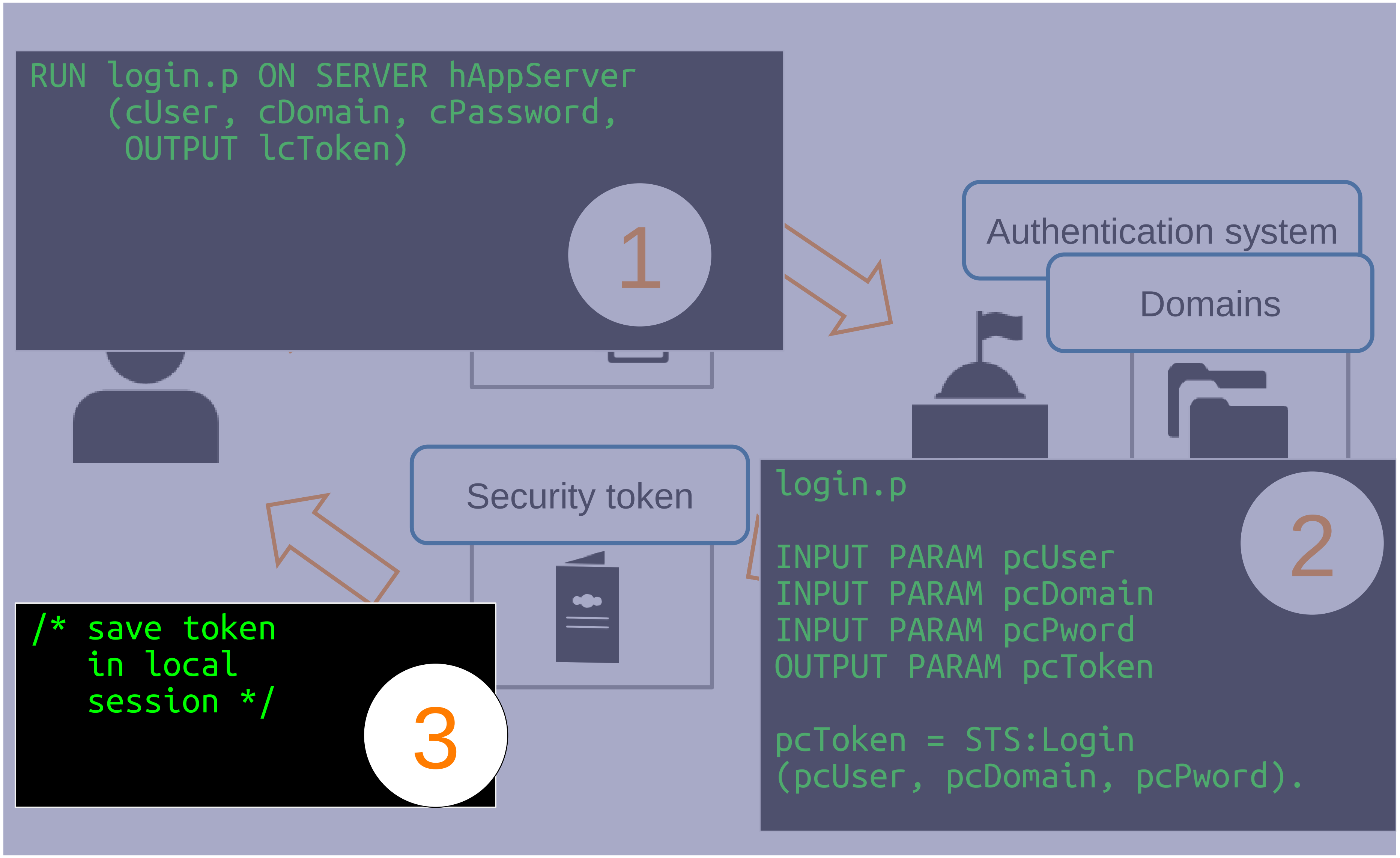
Security.SecurityTokenService.cls

```
method protected void WriteClientPrincipalToStore(
                                   input phClientPrincipal as handle):
  define buffer lbSecurityContext for SecurityContext.

  /* scope this transaction as small as possible */
  do for lbSecurityContext transaction:
    find lbSecurityContext where
         lbSecurityContext.SessionId eq phClientPrincipal:session-id
         exclusive-lock no-wait no-error.
    if not available lbSecurityContext then
    do:
      create lbSecurityContext.
      lbSecurityContext.SessionId = phClientPrincipal:session-id.
    end.
    lbSecurityContext.ClientPrincipal =
                        phClientPrincipal:export-principal().
    lbSecurityContext.LastAccess = now.
  end.
end method.
```

Security.SecurityTokenService.cls

```
method public char LoginUser(input pcUserName as char,
                             input pcUserDomain as char,
                             input pcPassword as char):
  define variable hClientPrincipal as handle no-undo.

  create client-principal hClientPrincipal.
  hClientPrincipal:initialize(
     substitute('&1@&2', pcUserName, pcUserDomain),
     ?,   /* unique session id */
     add-interval(now, 8, 'hours'),  /* login expiration */
     pcPassword).

  /* passes authentication work off to
     authentication system */
  security-policy:set-client(hClientPrincipal).

  /* writes security context into DB */
  WriteClientPrincipalToStore(hClientPrincipal).

  /* return character value */
  return hClientPrincipal:session-id.
end method.
```

Application Architecture: Login



```
RUN login.p ON SERVER hAppServer
    (cUser, cDomain, cPassword,
     OUTPUT lcToken)
```

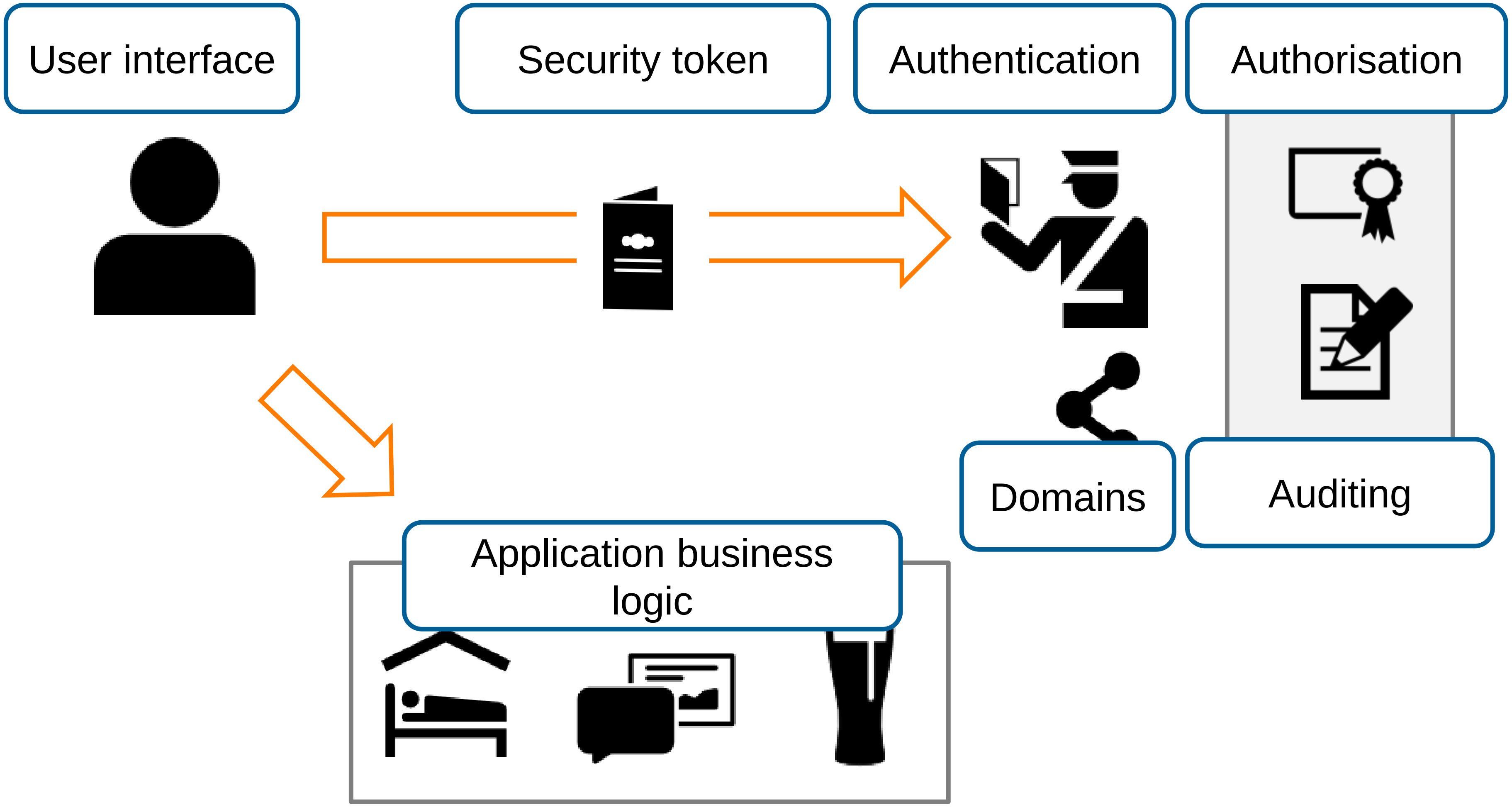
```
login.p

INPUT PARAM pcUser
INPUT PARAM pcDomain
INPUT PARAM pcPword
OUTPUT PARAM pcToken

pcToken = STS:Login
(pcUser, pcDomain, pcPword).
```

```
/* save token
   in local
   session */
```

Desktop.MainForm.cls

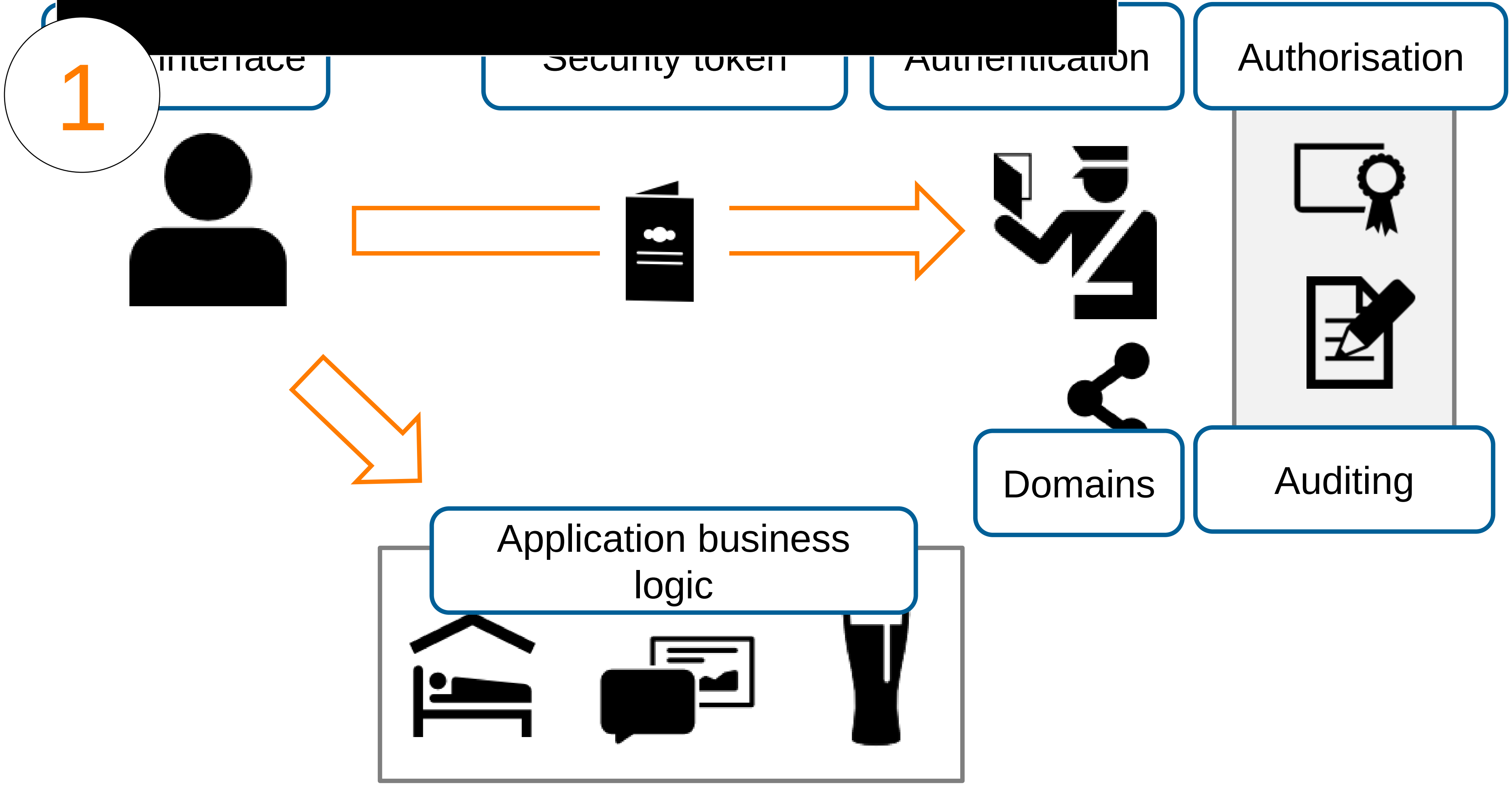
```
method public logical LoginUser(
              input pcUserName as char,
              input pcDomain as char,
              input pcPassword as char):

  run Security/Login.p on hAppServer (
          pcUserName, pcDomain, pcPassword,
          output cUserContextId).
  if cUserContextId eq '' then return false.

  /* set the CCID on the business logic server so that it's
     transported with every request. */
  hAppServer:request-info:ClientContextId = cUserContextId.

  return true.
end method.
```

Application Architecture: Business Logic

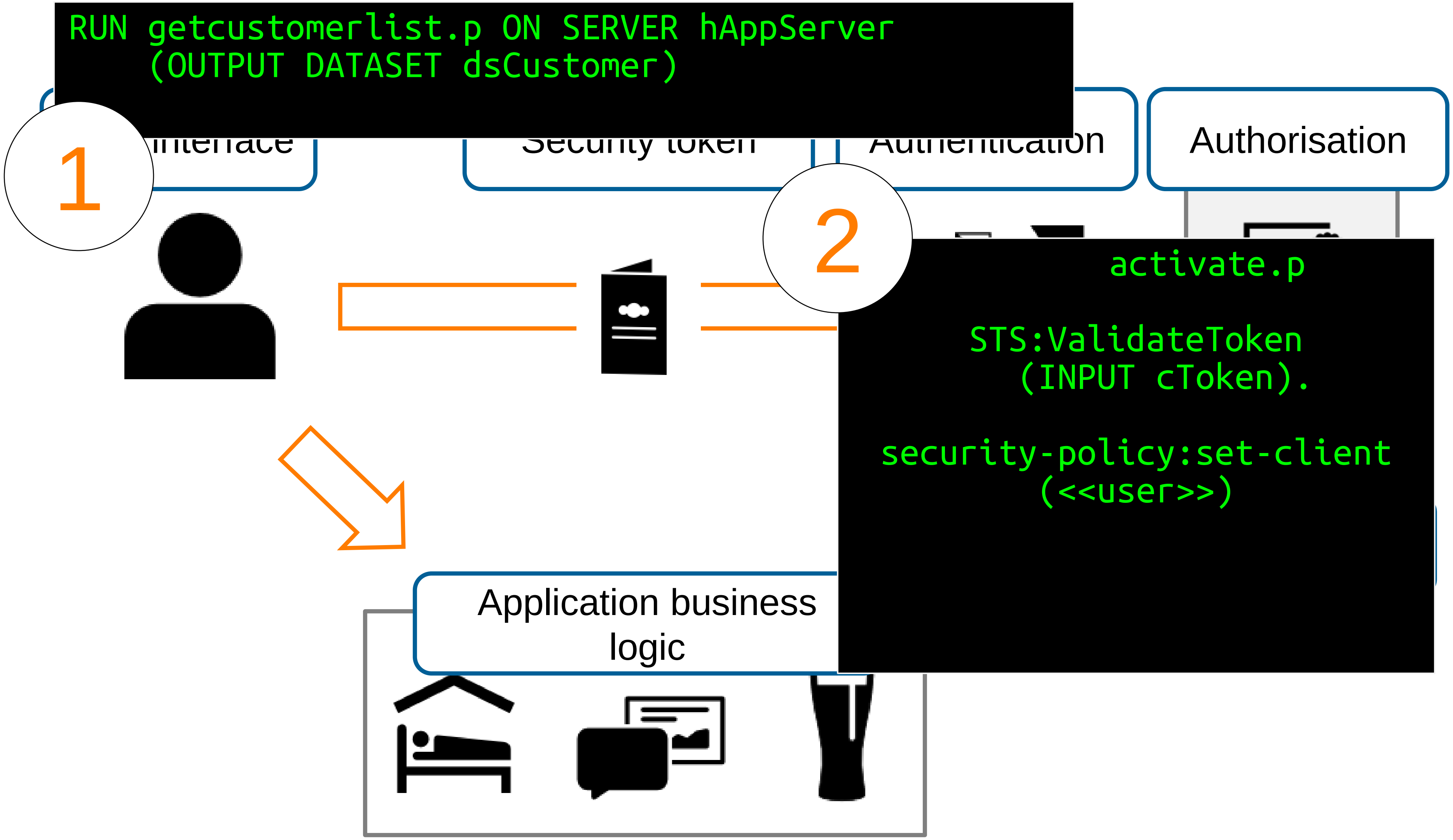


Application Architecture: Business Logic

```
RUN getcustomerlist.p ON SERVER hAppServer
    (OUTPUT DATASET dsCustomer)
```



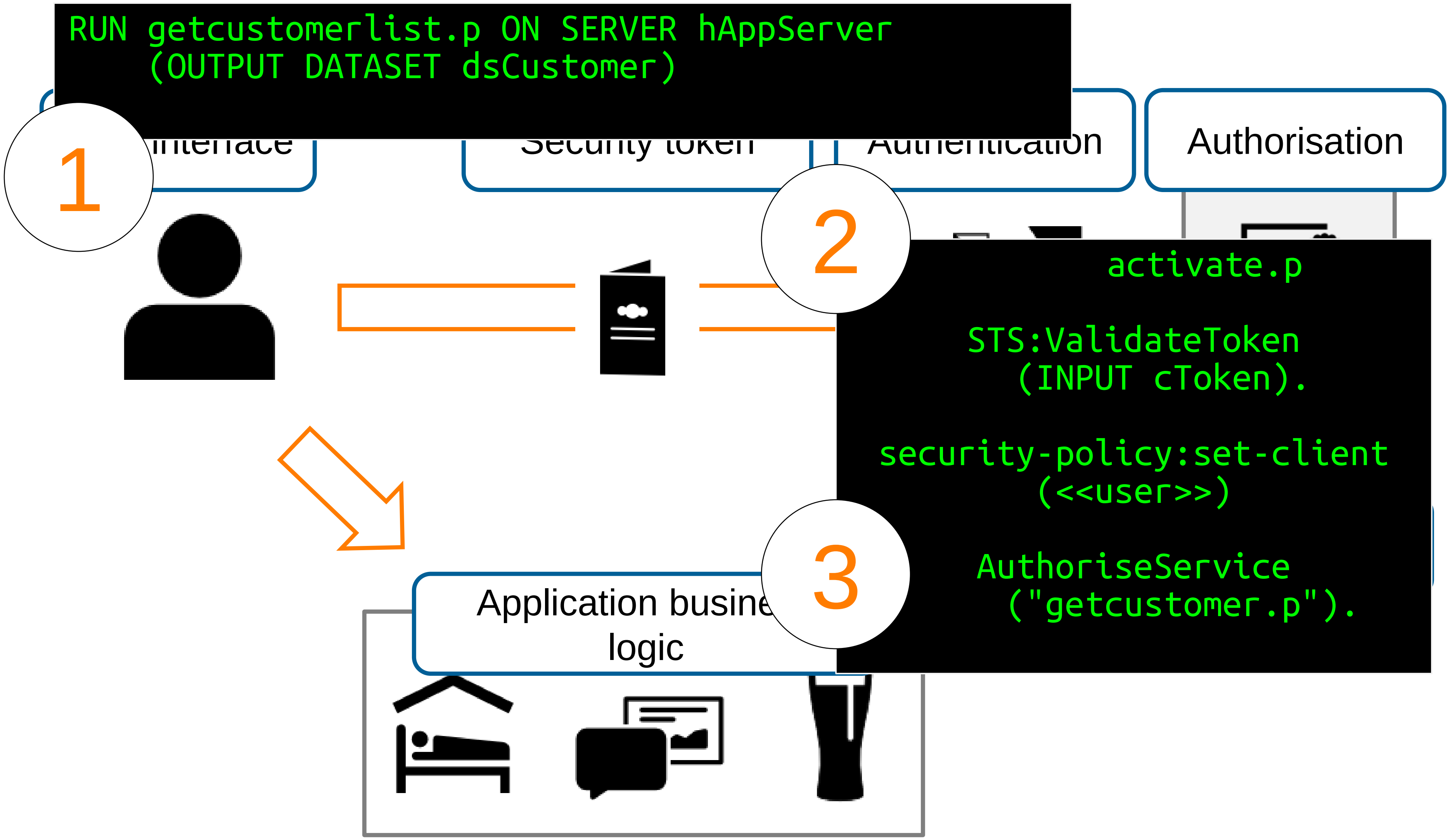
Application Architecture: Business Logic



```
RUN getcustomerlist.p ON SERVER hAppServer
    (OUTPUT DATASET dsCustomer)
```

```
activate.p

STS:ValidateToken
    (INPUT cToken).

security-policy:set-client
    (<<user>>)
```

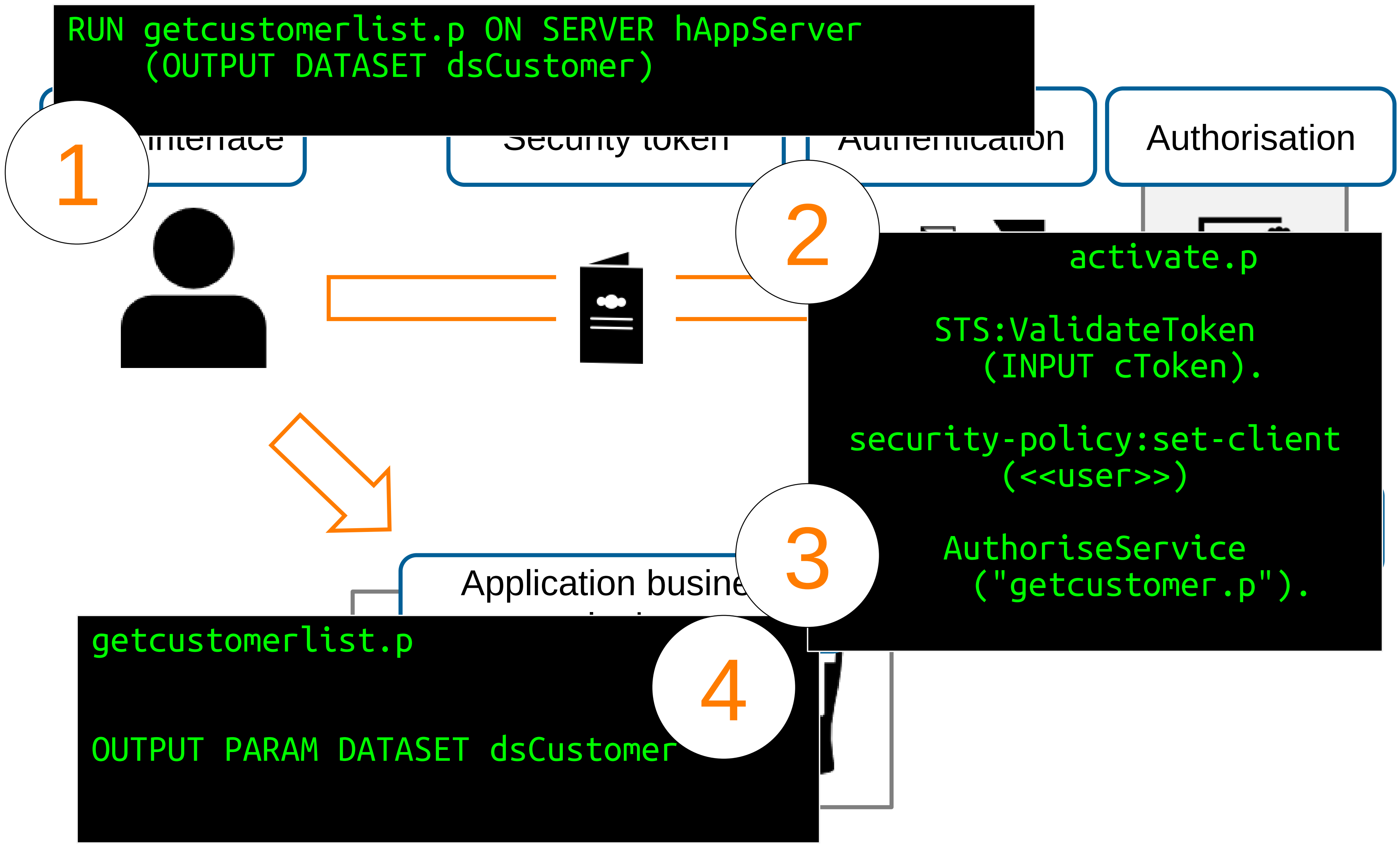
Application Architecture: Business Logic



```
RUN getcustomerlist.p ON SERVER hAppServer
    (OUTPUT DATASET dsCustomer)
```

```
           activate.p

       STS:ValidateToken
         (INPUT cToken).

security-policy:set-client
        (<<user>>)

      AuthoriseService
        ("getcustomer.p").
```

Application Architecture: Business Logic

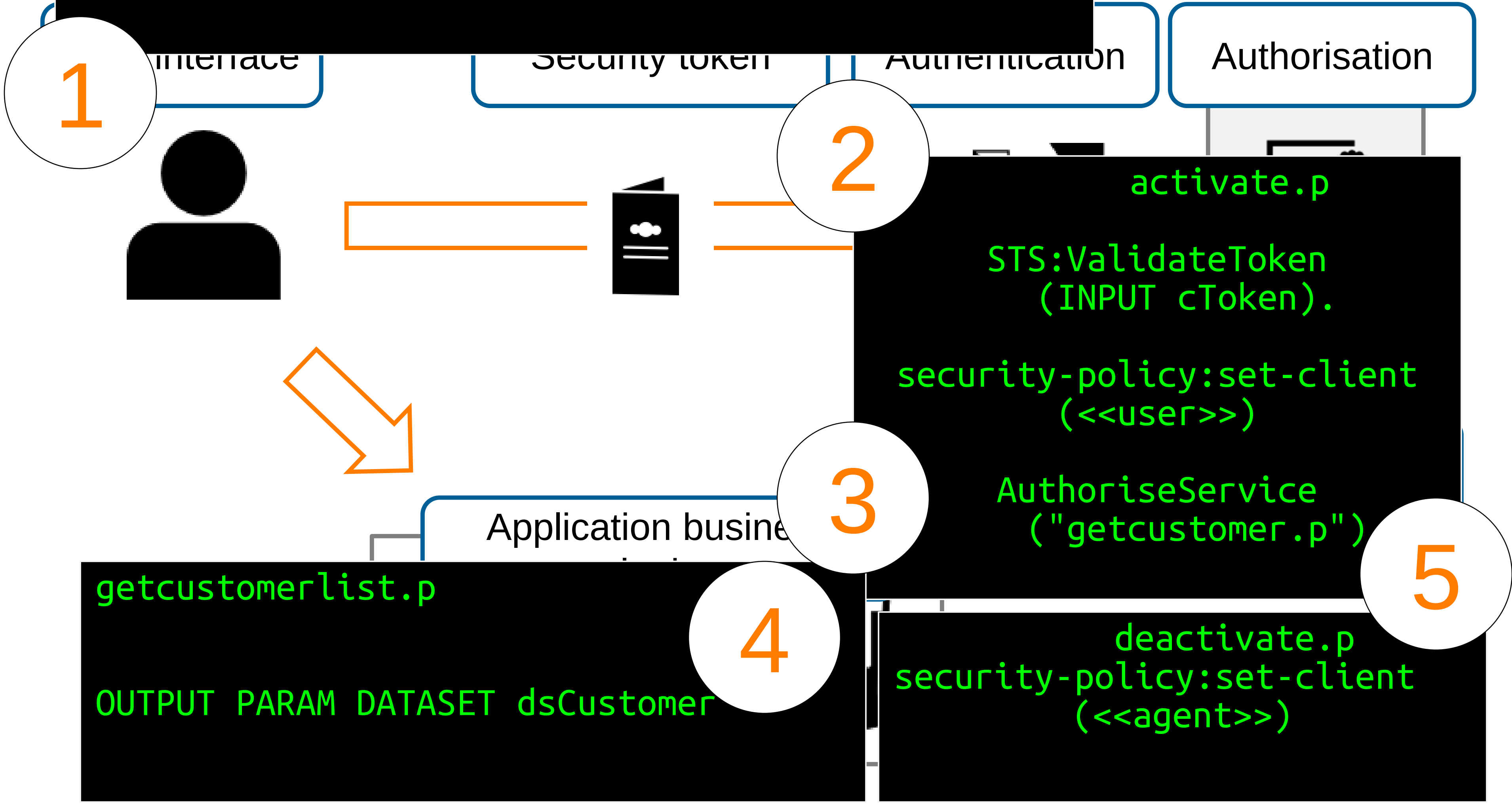


```
RUN getcustomerlist.p ON SERVER hAppServer
    (OUTPUT DATASET dsCustomer)
```

```
activate.p

STS:ValidateToken
   (INPUT cToken).

security-policy:set-client
  (<<user>>)

AuthoriseService
  ("getcustomer.p").
```

```
getcustomerlist.p


OUTPUT PARAM DATASET dsCustomer
```

Application Architecture: Business Logic



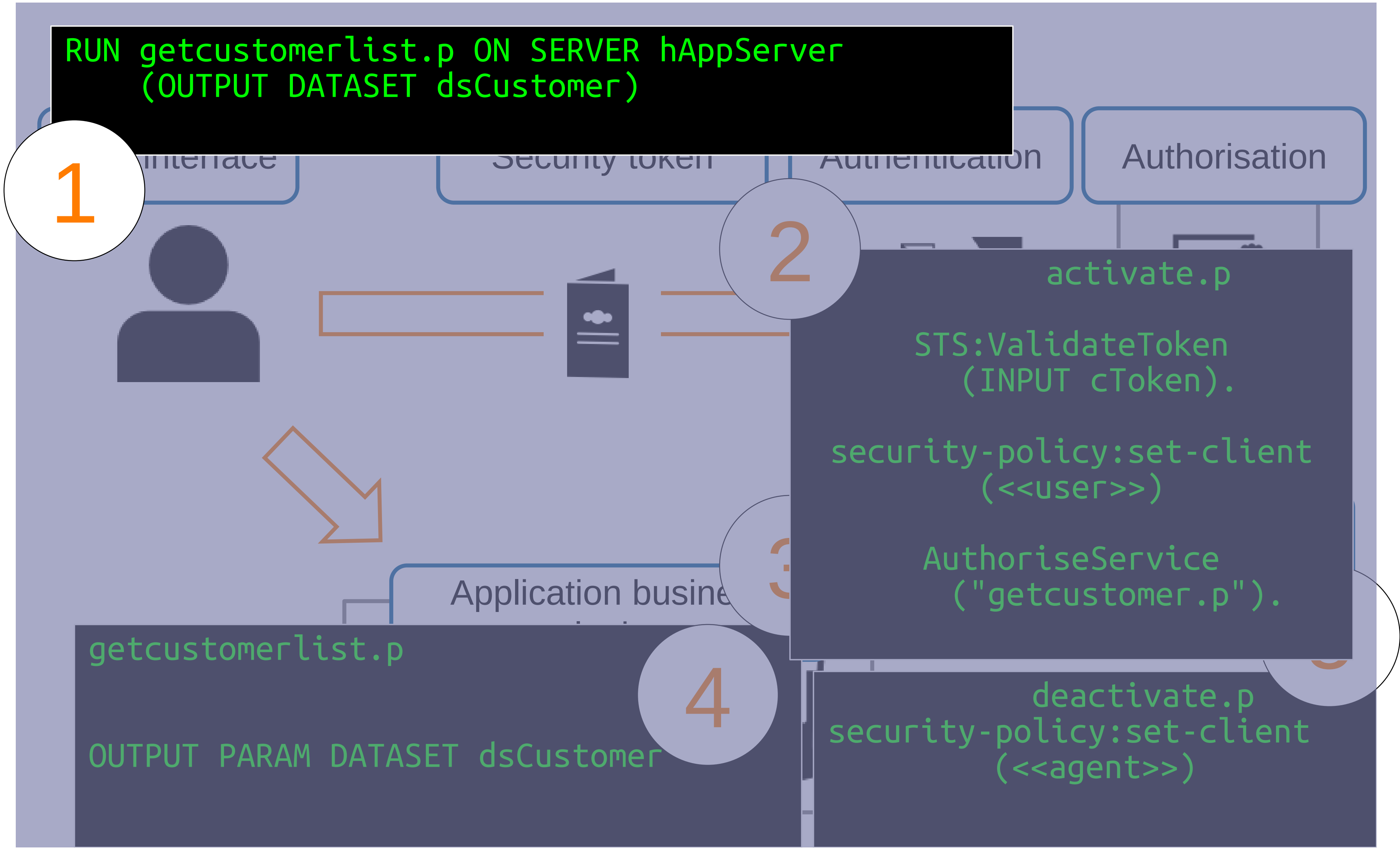
```
RUN getcustomerlist.p ON SERVER hAppServer
    (OUTPUT DATASET dsCustomer)
```

```
activate.p

STS:ValidateToken
  (INPUT cToken).

security-policy:set-client
  (<<user>>)

AuthoriseService
  ("getcustomer.p")
```

```
getcustomerlist.p


OUTPUT PARAM DATASET dsCustomer
```

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

Application Architecture: Business Logic



```
RUN getcustomerlist.p ON SERVER hAppServer
    (OUTPUT DATASET dsCustomer)
```

```
activate.p

STS:ValidateToken
  (INPUT cToken).

security-policy:set-client
  (<<user>>)

AuthoriseService
  ("getcustomer.p").
```

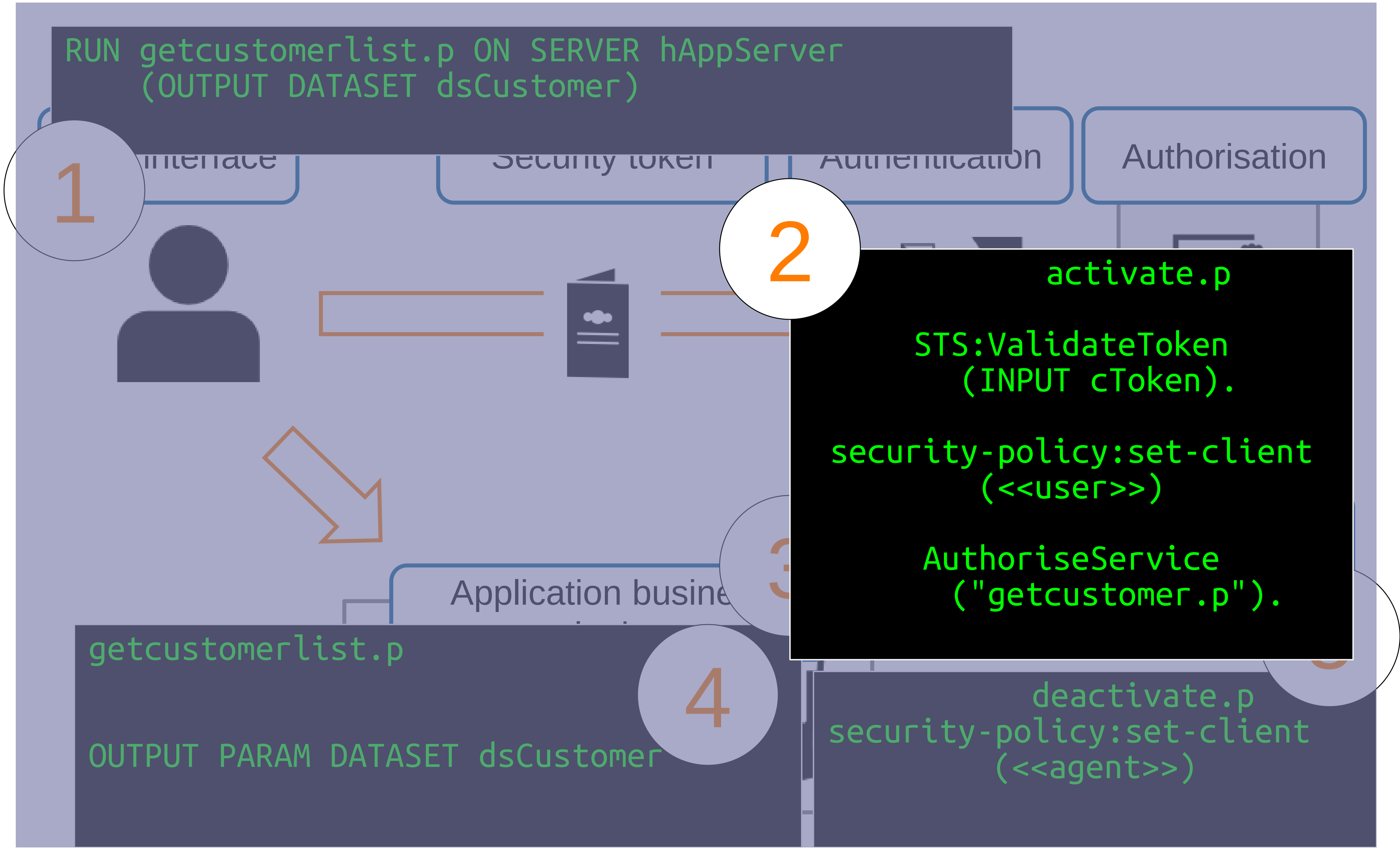
```
getcustomerlist.p


OUTPUT PARAM DATASET dsCustomer
```

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

Desktop.MainForm.cls

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

  run BusinessLogic/GetCustomerList.p on hAppServer
                          (output dataset dsCustomerOrder).

  open query qryCustomer preselect
     each ttCustomer by ttCustomer.CustNum.

  bsCustomer:Handle = query qryCustomer:handle.

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

Application Architecture: Business Logic



```
RUN getcustomerlist.p ON SERVER hAppServer
    (OUTPUT DATASET dsCustomer)
```

```
activate.p

STS:ValidateToken
  (INPUT cToken).

security-policy:set-client
  (<<user>>)

AuthoriseService
  ("getcustomer.p").
```

```
getcustomerlist.p


OUTPUT PARAM DATASET dsCustomer
```

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

Security/Activate.p

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

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

Security.SecurityTokenService.cls

```
method public handle GetClientPrincipal(input pcContextId as char):
  define variable hClientPrincipal as handle no-undo.
  define variable rClientPrincipal as raw no-undo.

  define buffer lbSecurityContext for SecurityContext.

  /* scope this transaction as small as possible */
  do for lbSecurityContext transaction:
    find lbSecurityContext where
         lbSecurityContext.SessionId eq pcContextId
         exclusive-lock no-wait no-error.
    if not available lbSecurityContext then
      undo, throw new AppError('Context does not exist').
    assign rClientPrincipal = lbSecurityContext.ClientPrincipal
           lbSecurityContext.LastAccess = now.
  end.

  create client-principal hClientPrincipal.
  hClientPrincipal:import-principal(rClientPrincipal).

  return hClientPrincipal.
end method.
```

Security/Activate.p

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

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

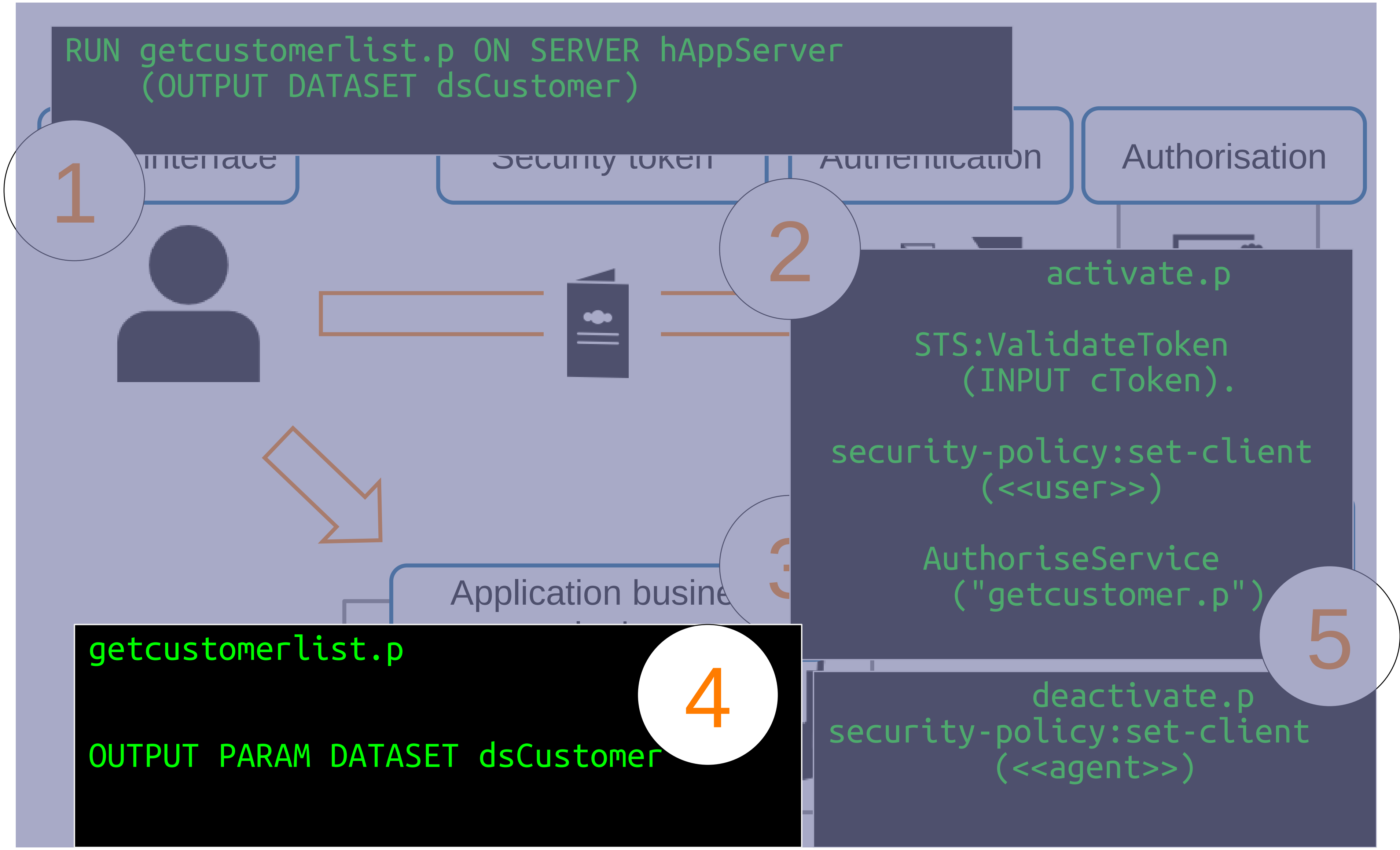
_sec-authentication-system

```
create _sec-authentication-system.
_Domain-type                 = 'TABLE-ApplicationUser'.
_Domain-type-description =
         'The ApplicationUser table serves as
          the authentication domain'.
_PAM-plug-in                 = true.

_PAM-callback-procedure  =
         'Security/AppUserAuthenticate.p'.
```

Security/AppUserAuthenticate.p

```
procedure AfterSetIdentity:
  def input param phClientPrincipal  as handle no-undo.
  def input param pcSystemOptions as character extent no-undo.

  /* At this point the CLIENT-PRINCIPAL is sealed and the
     user authenticated */

  /* Load user/application (as opposed to security)
     context here */

  return.
end procedure.
```

Application Architecture: Business Logic



```
RUN getcustomerlist.p ON SERVER hAppServer
    (OUTPUT DATASET dsCustomer)
```

```
activate.p

STS:ValidateToken
  (INPUT cToken).

security-policy:set-client
  (<<user>>)

AuthoriseService
  ("getcustomer.p")
```

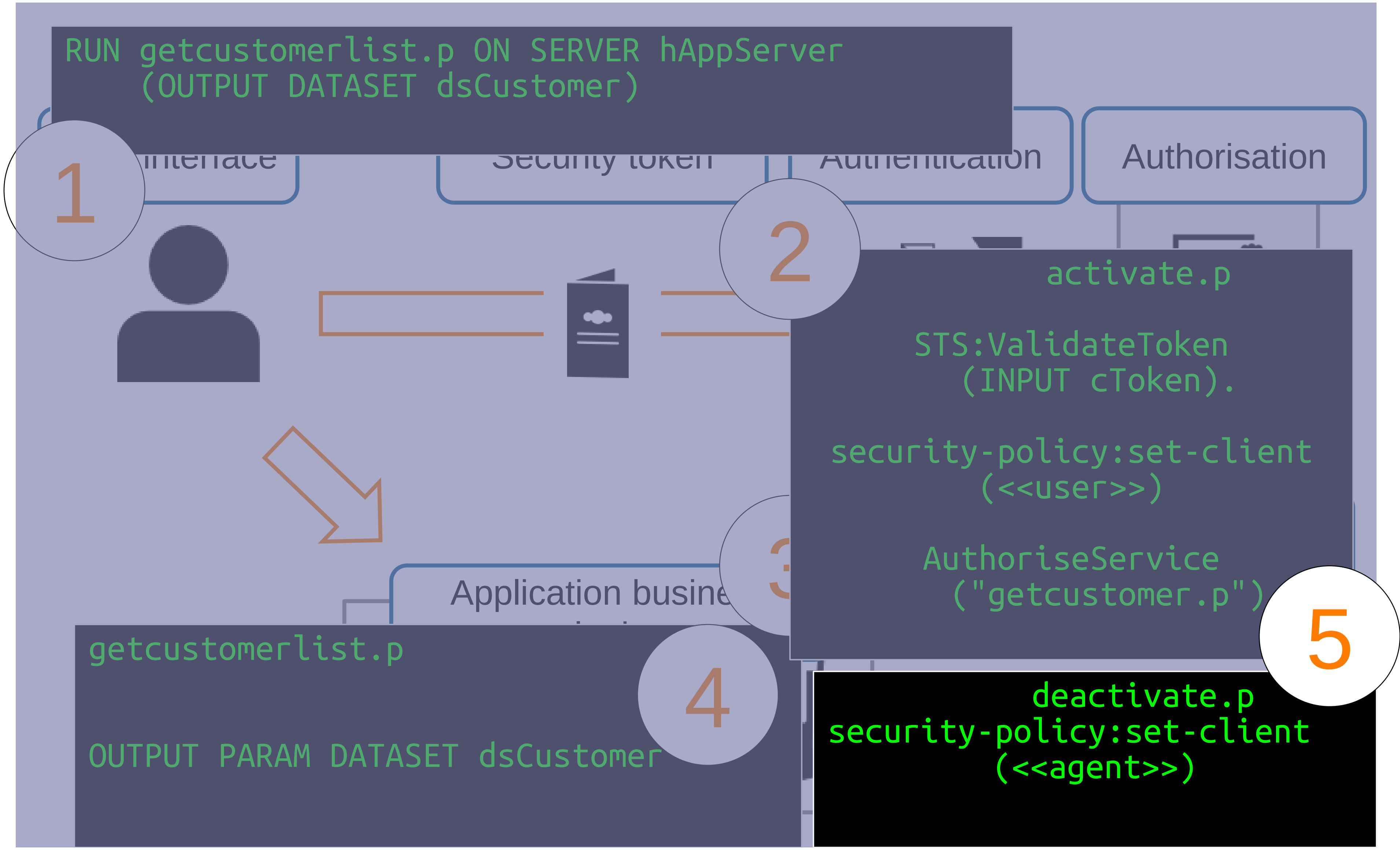
```
getcustomerlist.p


OUTPUT PARAM DATASET dsCustomer
```

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

BusinessLogic/GetCustomerList.p

```
{BusinessLogic/dsCustomerOrder.i}

define output parameter dataset for dsCustomerOrder.

define variable oBusinessEntity as CustomerOrderBE no-undo.

oBusinessEntity  = new CustomerOrderBE().

oBusinessEntity:GetCustomers(output dataset dsCustomerOrder).

/* eof */
```

Application Architecture: Business Logic



```
RUN getcustomerlist.p ON SERVER hAppServer
    (OUTPUT DATASET dsCustomer)
```

```
activate.p

STS:ValidateToken
  (INPUT cToken).

security-policy:set-client
  (<<user>>)

AuthoriseService
  ("getcustomer.p")
```

```
getcustomerlist.p


OUTPUT PARAM DATASET dsCustomer
```

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

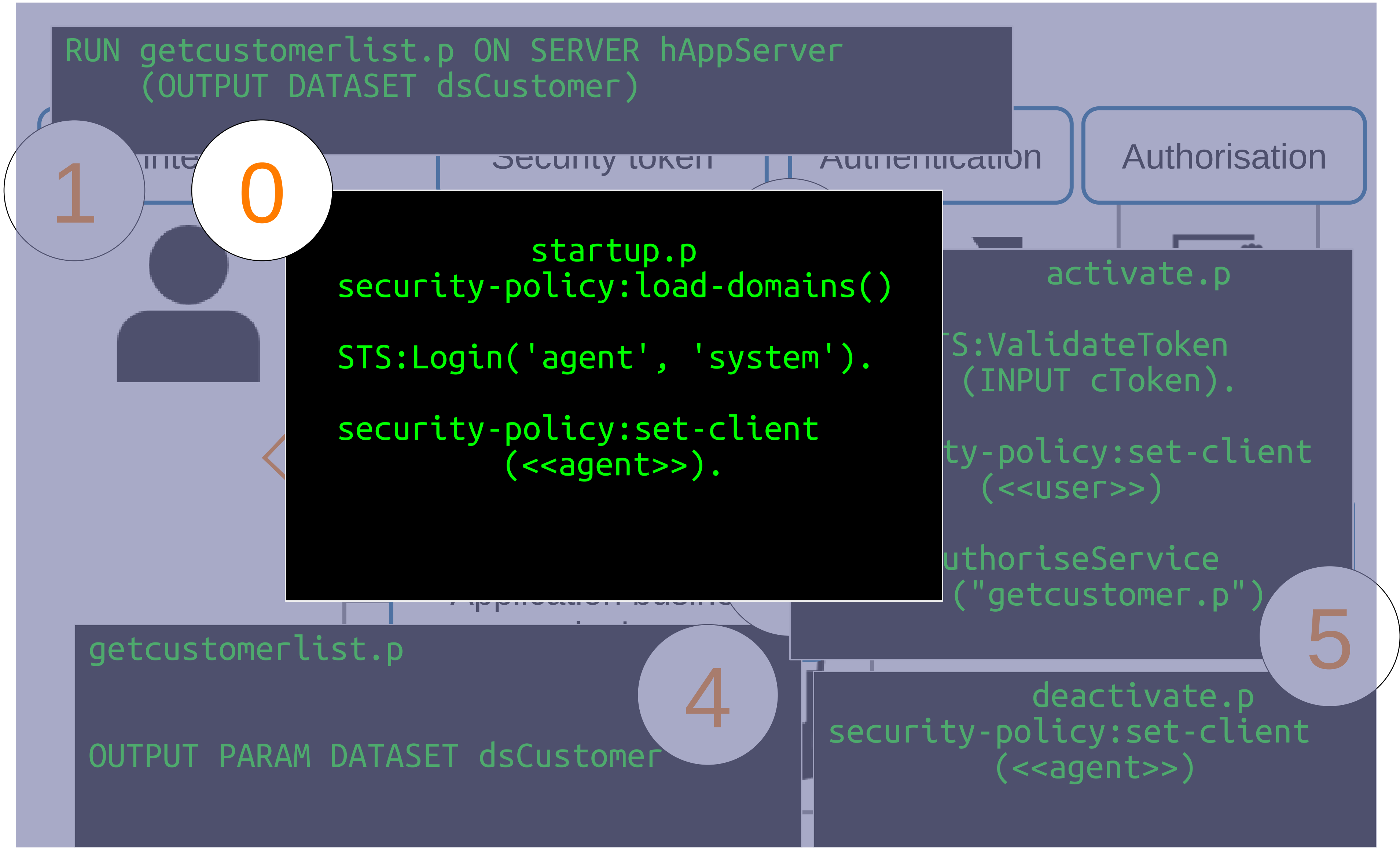
Security/Deactivate.p

```
define variable hClientPrincipal as handle no-undo.

hClientPrincipal = dynamic-function(
                      'GetAgentClientPrincipal' in hStartupProc)

security-policy:set-client(hClientPrincipal).

/* eof */
```

Desktop.MainForm.cls

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

  run BusinessLogic/GetCustomerList.p on hAppServer
                          (output dataset dsCustomerOrder).

  open query qryCustomer preselect
     each ttCustomer by ttCustomer.CustNum.

  bsCustomer:Handle = query qryCustomer:handle.

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

Application Architecture: Business Logic



```
RUN getcustomerlist.p ON SERVER hAppServer
    (OUTPUT DATASET dsCustomer)
```

```
startup.p
security-policy:load-domains()

STS:Login('agent', 'system').

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

```
activate.p

TS:ValidateToken
  (INPUT cToken).

ty-policy:set-client
  (<<user>>)

uthoriseService
 ("getcustomer.p")
```

```
getcustomerlist.p


OUTPUT PARAM DATASET dsCustomer
```

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

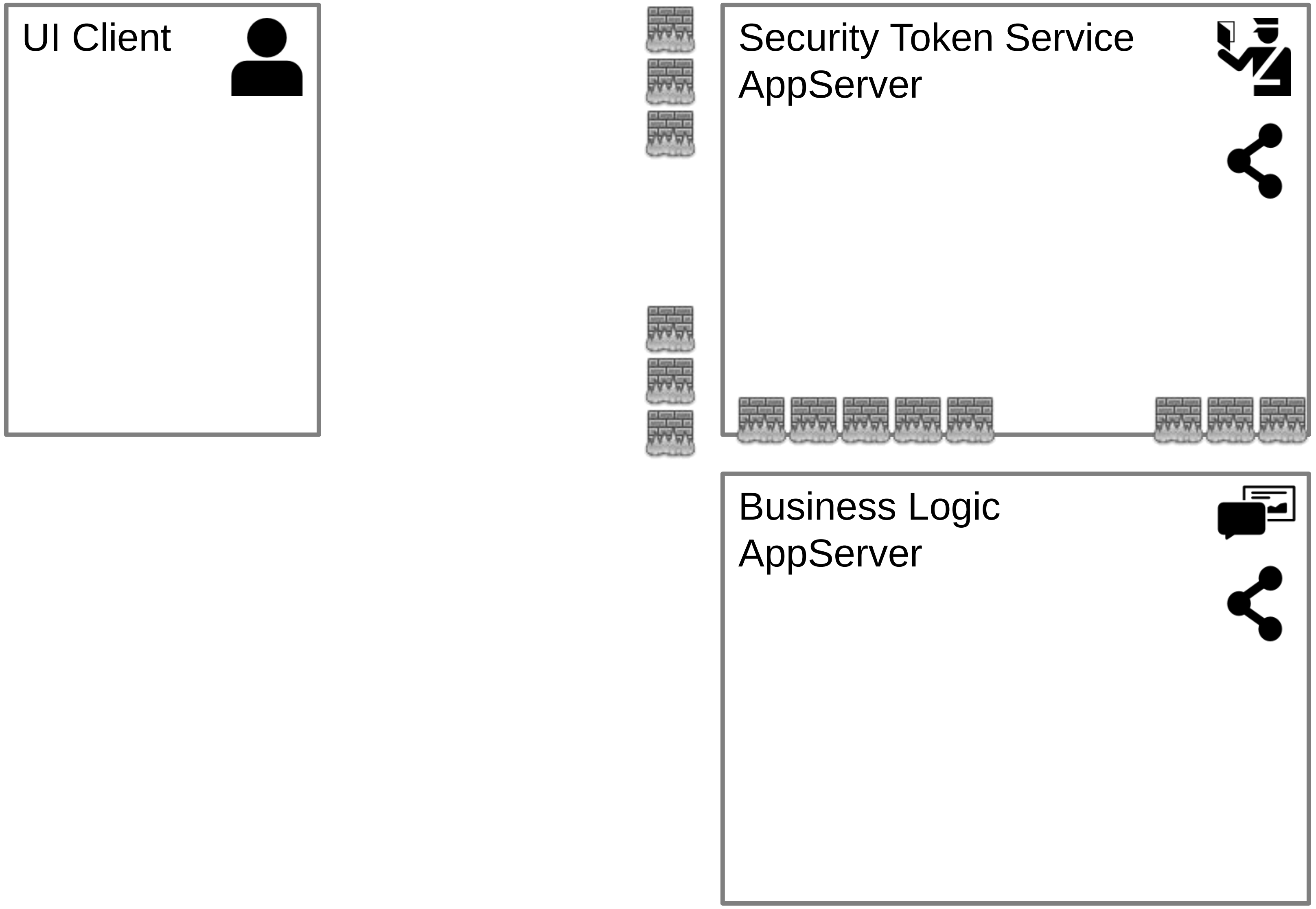
Security/Startup.p

```
define input parameter pcStartupData as character no-undo.

define variable cAgentSessionId as character no-undo.
define variable hClientPrincipal as handle no-undo.

/* load domains */
security-policy:load-domains('sports2000').

/* immediately  set session user to a low-privilege agent user */
cAgentSessionId = Security.SecurityTokenService:Instance

             :LoginUser('agent', 'system','oech1::3c373b2a372c3d').

hClientPrincipal = Security.SecurityTokenService:Instance
                      :GetClientPrincipal(cAgentSessionId).

security-policy:set-client (hClientPrincipal).

function GetAgentSessionId returns character ():
    return cAgentSessionId.
end function.

function GetAgentClientPrincipal returns handle():
    return hClientPrincipal.
end function.
/* eof */
```

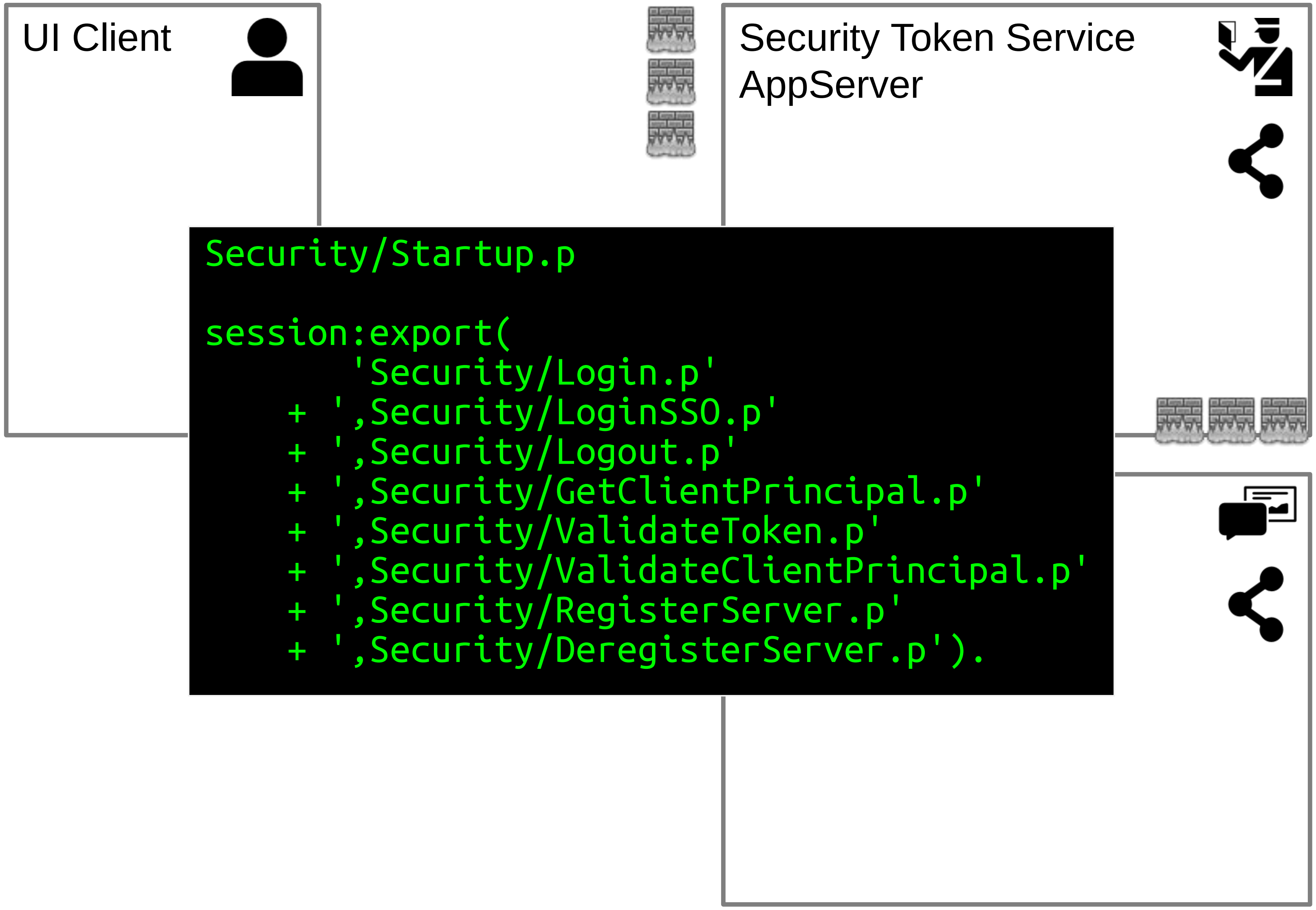
Security/Shutdown.p

```
Security.SecurityTokenService:Instance
    :LogoutUser(
        dynamic-function('GetAgentSessionId' in hStartupProc)).

/* eof */
```

Separate AppServers for STS & Business Logic

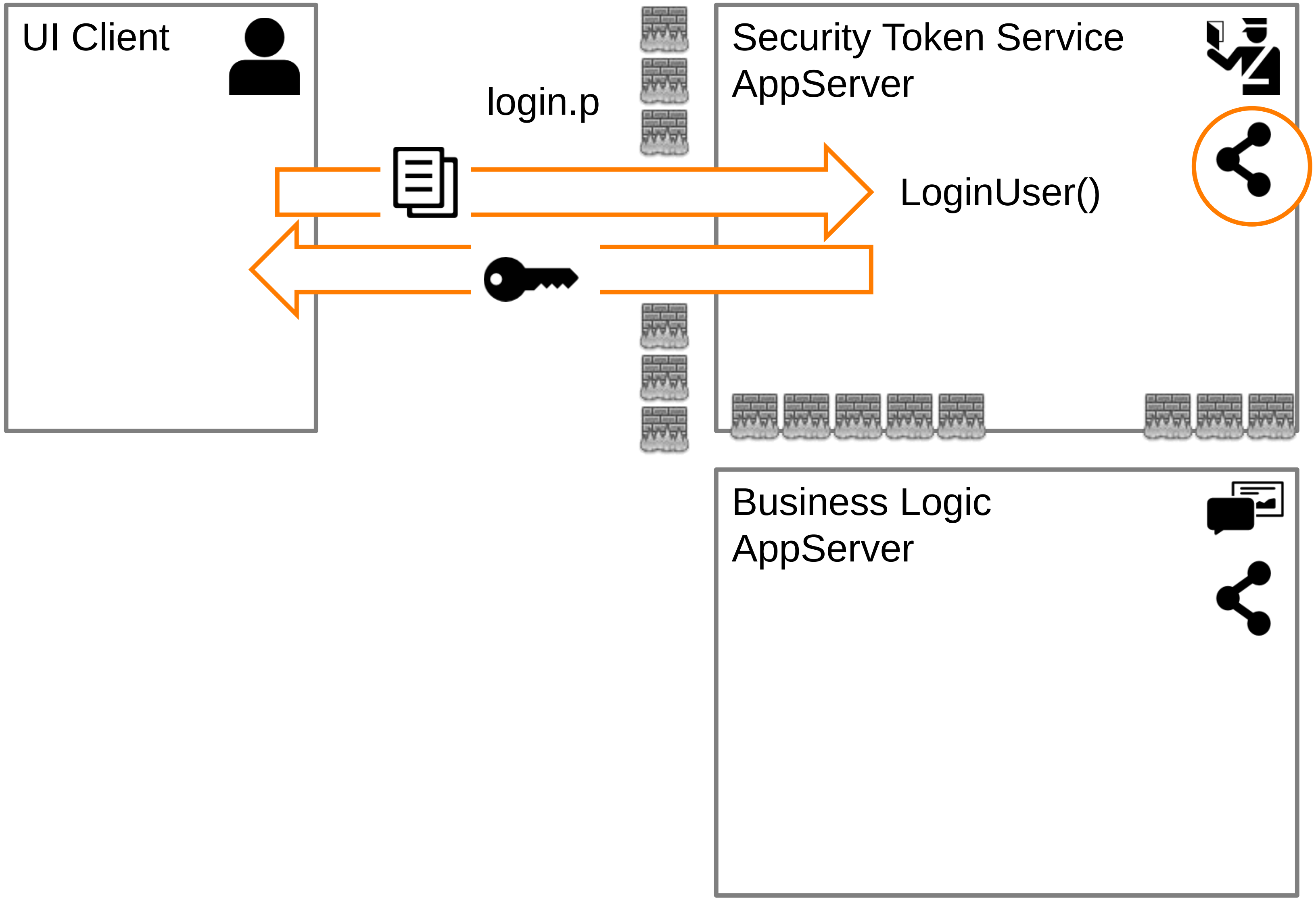


Separate AppServers for STS & Business Logic

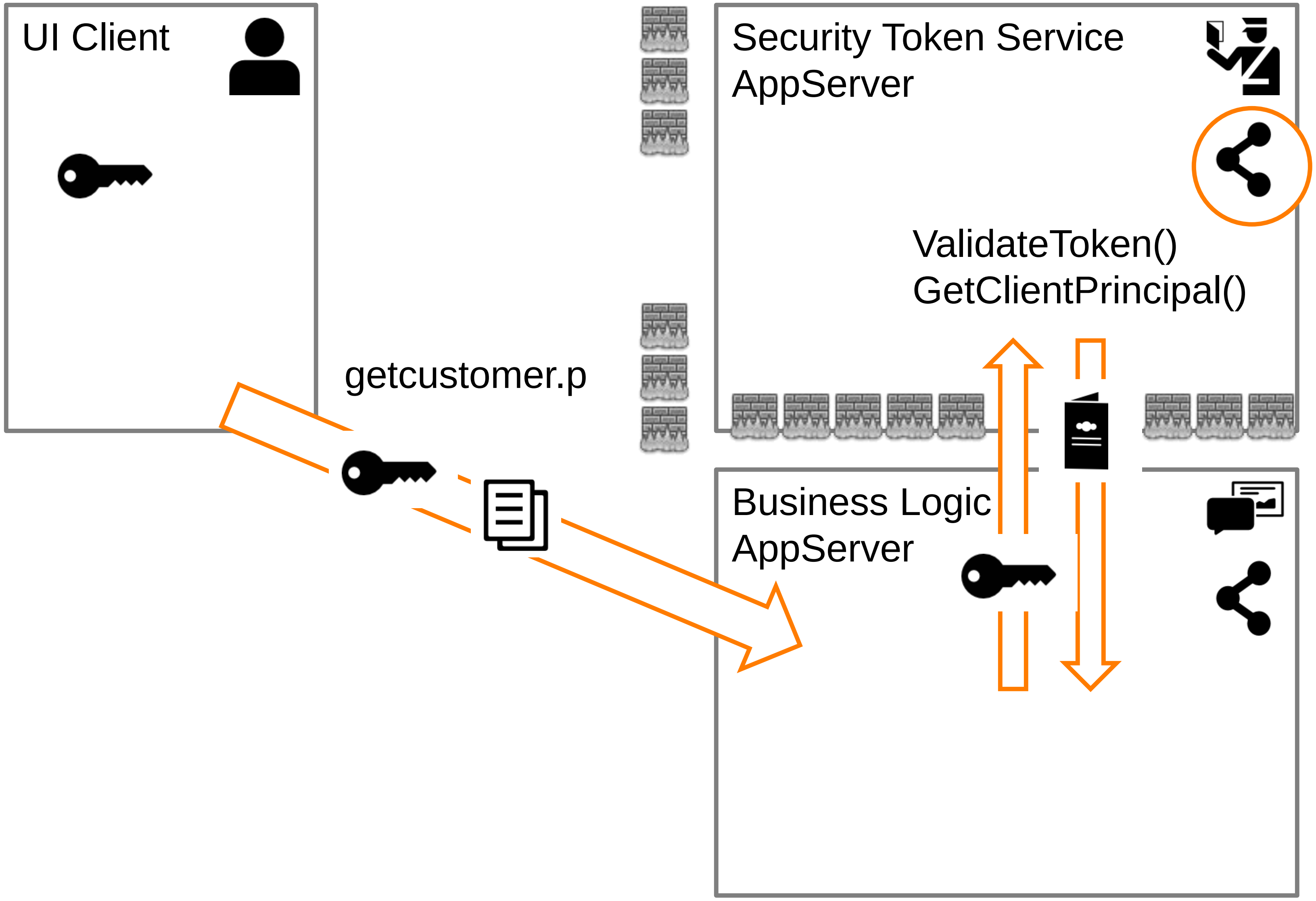


```
Security/Startup.p

session:export(
        'Security/Login.p'
    + ',Security/LoginSSO.p'
    + ',Security/Logout.p'
    + ',Security/GetClientPrincipal.p'
    + ',Security/ValidateToken.p'
    + ',Security/ValidateClientPrincipal.p'
    + ',Security/RegisterServer.p'
    + ',Security/DeregisterServer.p').
```

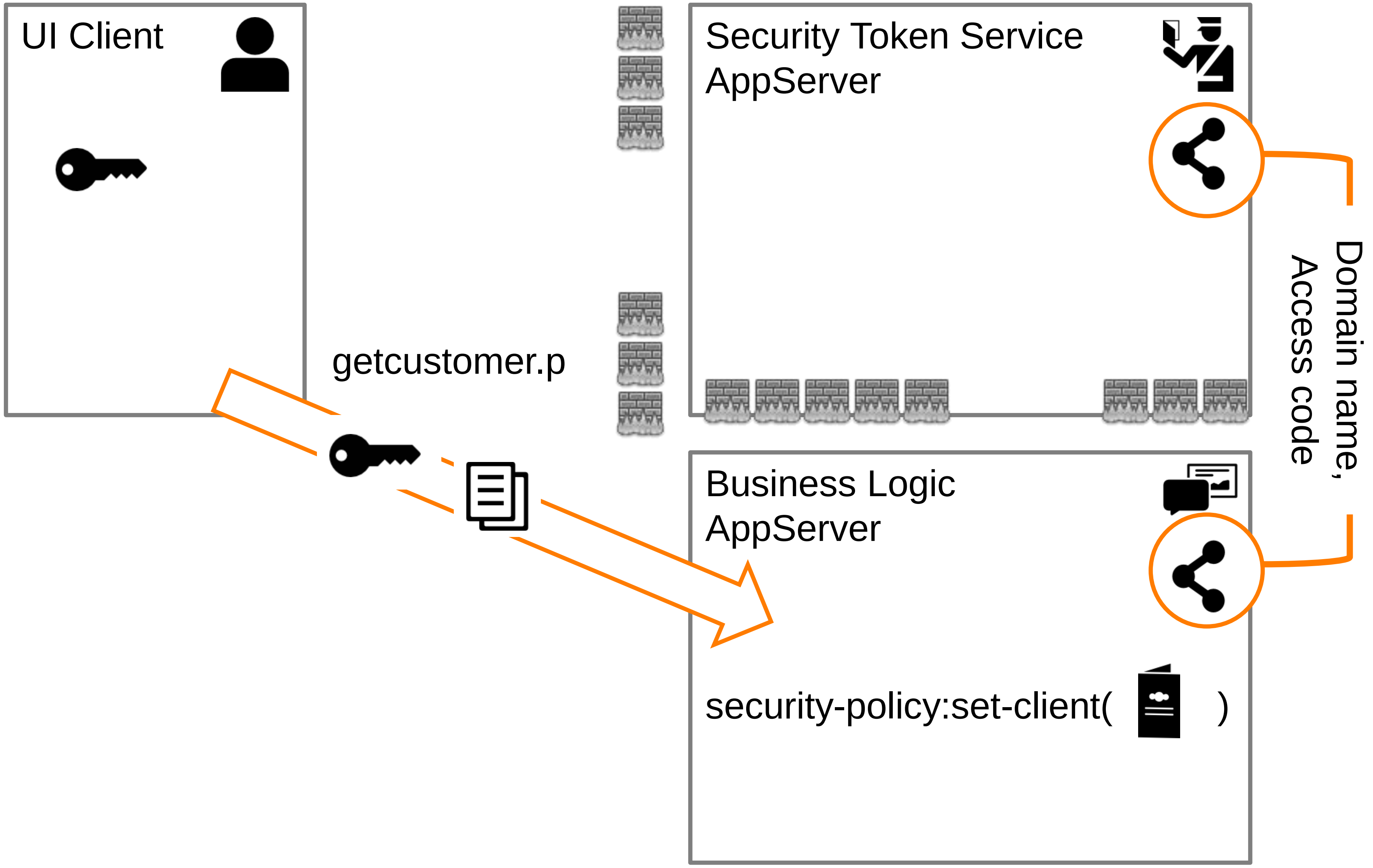
Separate AppServers for STS & Business Logic



Separate AppServers for STS & Business Logic



Separate AppServers for STS & Business Logic



_sec-authentication-system & -domain

Security Token Service

```
create _sec-authentication-system.
_Domain-type                  = 'TABLE-ApplicationUser'.
_PAM-plug-in                  = true.
_PAM-callback-procedure  =
         'Security/AppUserAuthenticate.p'.
```

Business Logic Service

```
create _sec-authentication-system.
_Domain-type                  = 'TABLE-ApplicationUser'.
_PAM-plug-in                  = true.
_PAM-callback-procedure  =
         'Security/NoLoginAuthenticate.p'.
```

Common

```
create _sec-authentication-domain.
_Domain-name             = 'employee'.
_Domain-type             = 'TABLE-ApplicationUser'.
_Domain-access-code =
    audit-policy:encrypt-audit-mac-key(
                        ‘s00perSecr1tK3y4EMPLOYEE').
_Domain-enabled          = true.
```

_PAM-callback-procedure

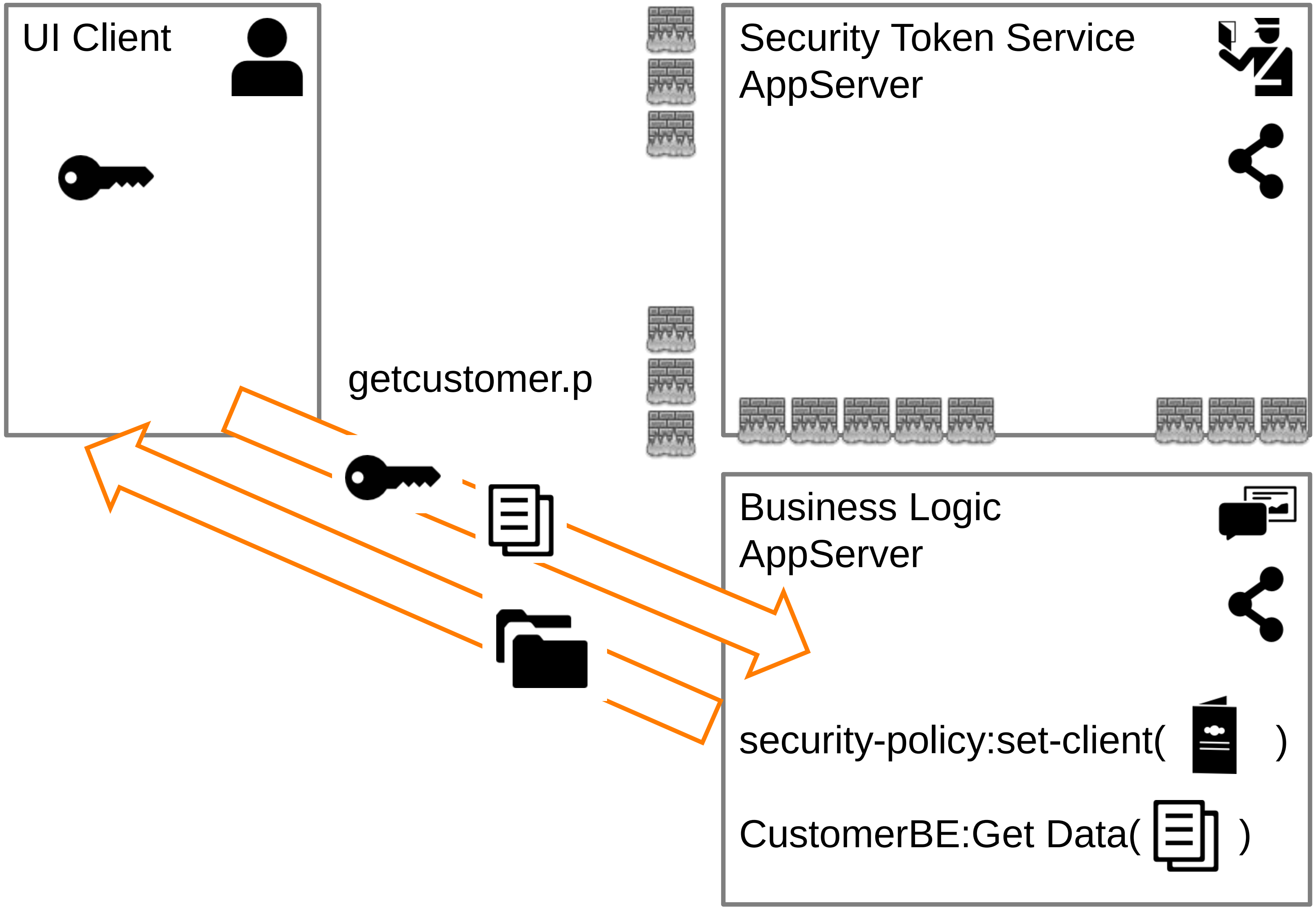
Security Token Service

```
procedure AuthenticateUser:
   /* snipped parameters*/
   find ApplicationUser where
          ApplicationUser.LoginName eq phCP:user-id and
          ApplicationUser.LoginDomain eq phCP:domain-name
          no-lock no-error.

   if not available ApplicationUser then
     piPAMStatus = Progress.Lang.PAMStatus:UnknownUser.
   else
   if ApplicationUser.Password ne
      encode(phCP:primary-passphrase) then
     piPAMStatus = Progress.Lang.PAMStatus:AuthenticationFailed.
   else
     /* we're good to go */
     piPAMStatus = Progress.Lang.PAMStatus:Success.

   return.
end procedure.
```

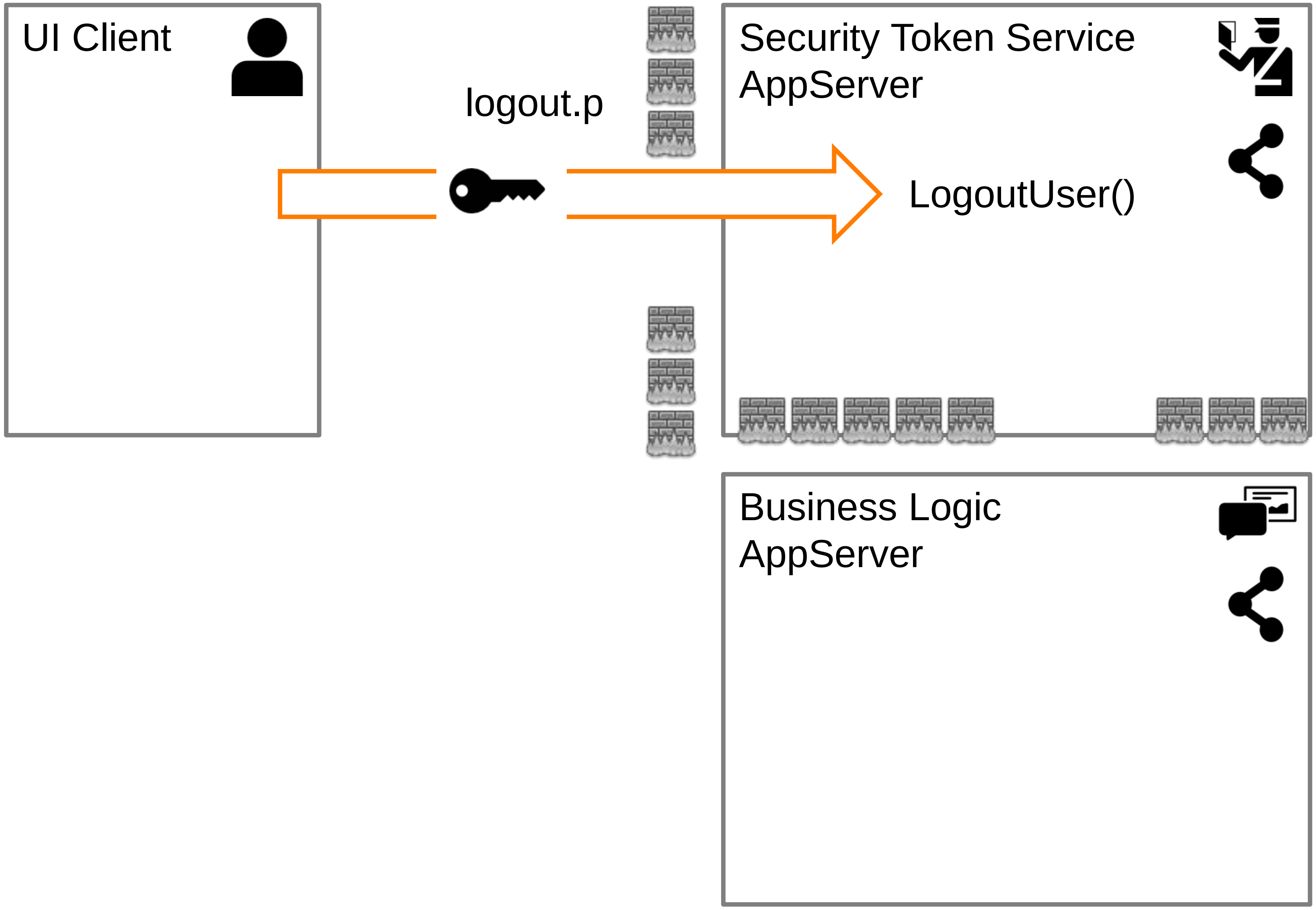
Business Logic Service

```
procedure AuthenticateUser:
  /* snipped parameters*/
  /* we're not allowed to do any logins here */
  piPAMStatus = PAMStatus:InvalidConfiguration.
  return.
end.
```

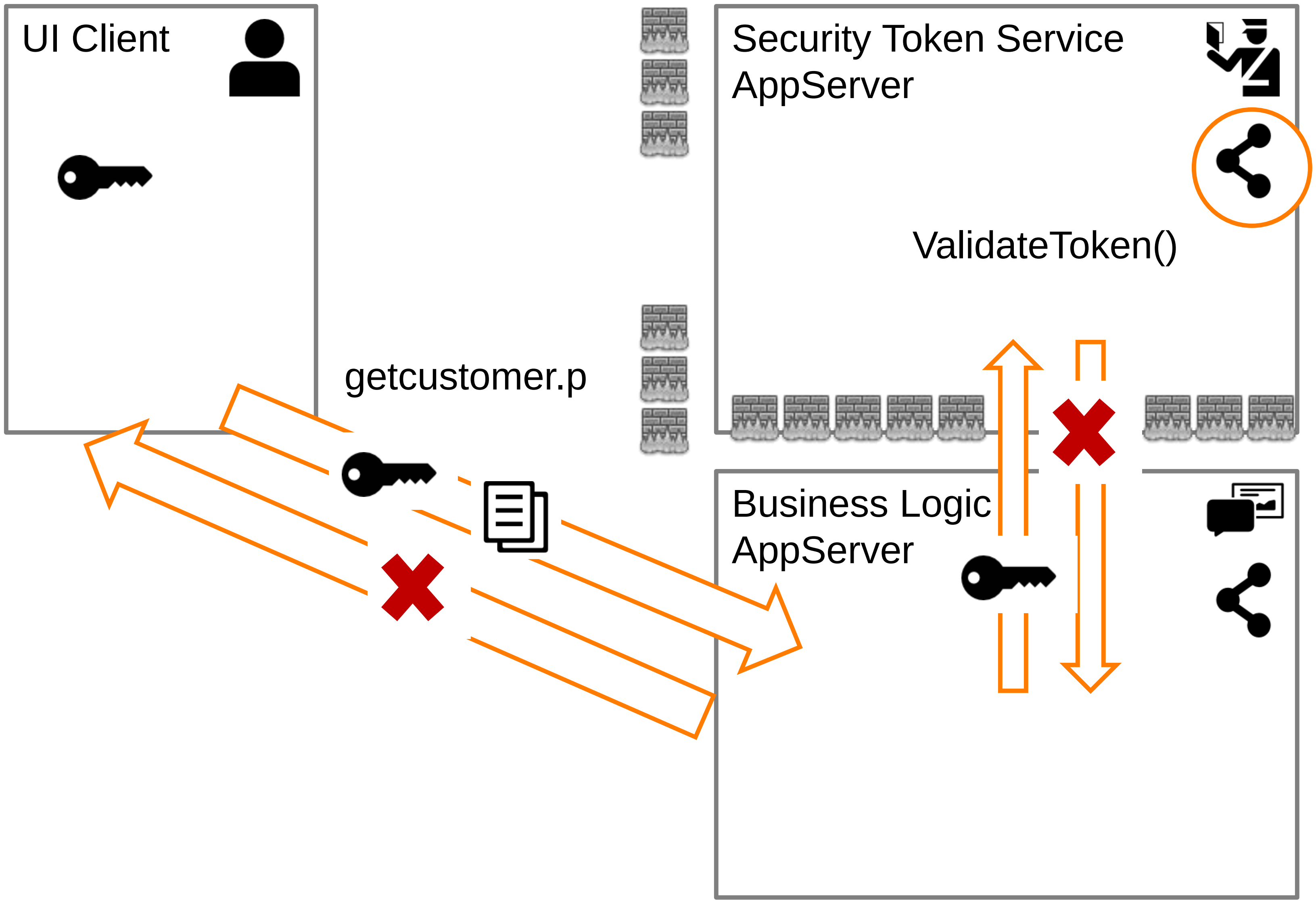
Separate AppServers for STS & Business Logic



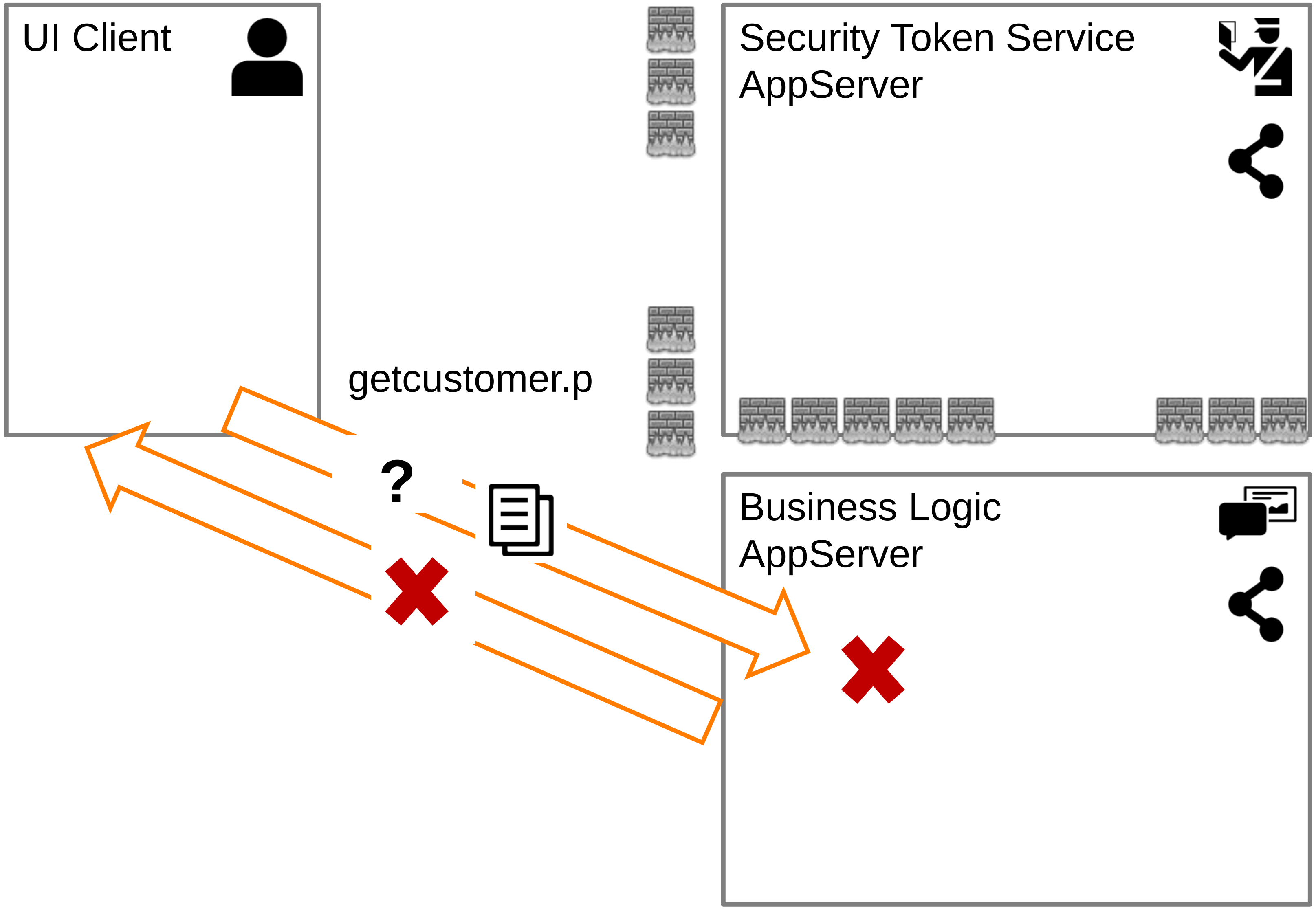
Separate AppServers for STS & Business Logic



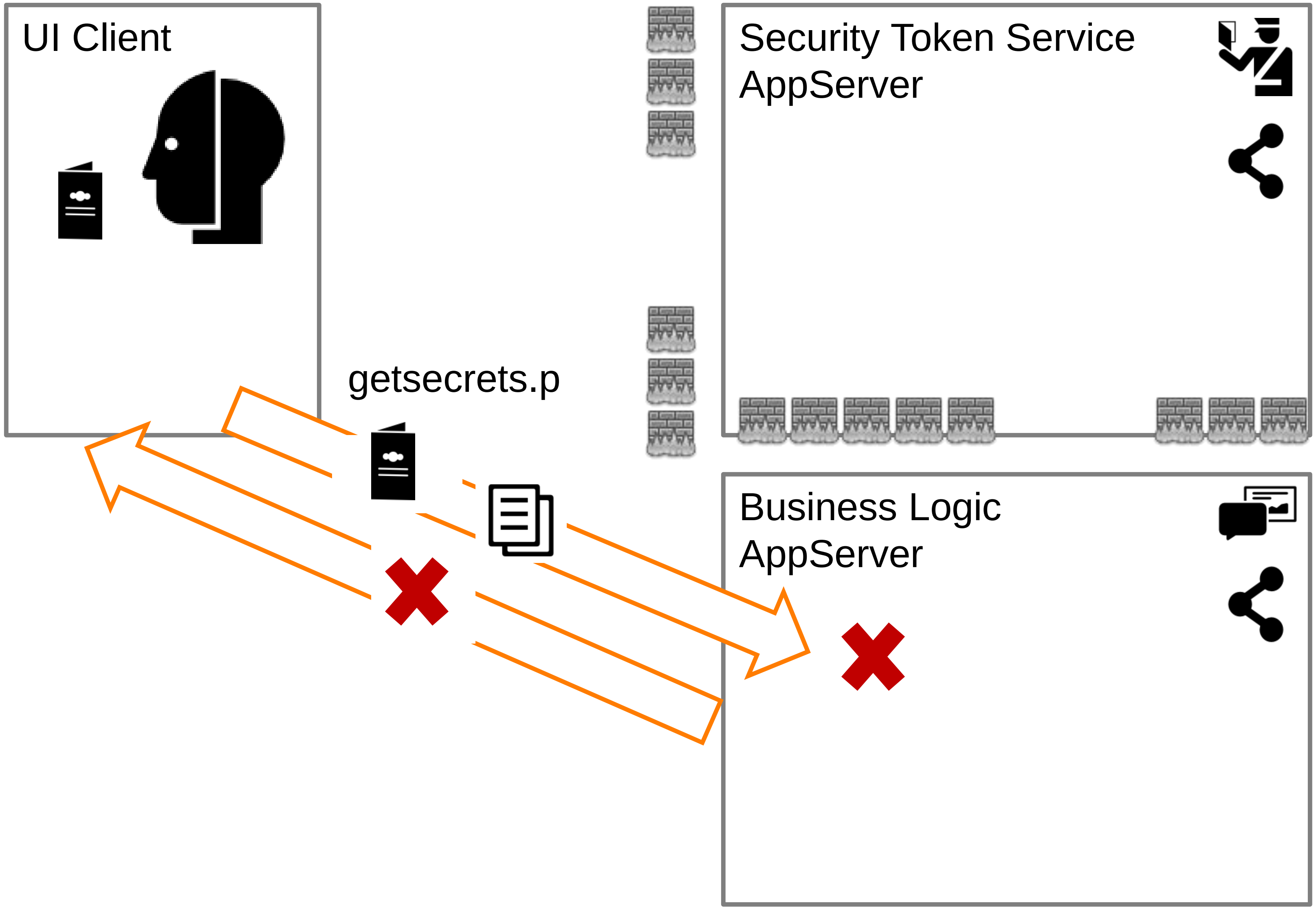
Separate AppServers for STS & Business Logic



Separate AppServers for STS & Business Logic



Separate AppServers for STS & Business Logic



OpenEdge Provides …

- A security token
  - CLIENT-PRINCIPAL available in multiple clients
  - Automatic creation in some cases
- Token available in activate procedure
- PAM modules
  - Configurable, plug-in architecture
  - Guaranteed, consistent, trusted code-paths

OpenEdge Does Not …

- Have a prescriptive model
- Manage security context for an entire application
- Automatic import of external systems’ tokens
  - For example, SAML for federated authentication

Coming Soon … {std/disclaimer.i}

- More authentication systems / PAM modules
  - Better SSO support (Windows workstation)
  - LDAP
  - ActiveDirectory
- Upgraded security for _User
- OpenEdge realm for BPM & REST

```
Progress.Security.Realm.IHybridRealm
```

Summary

- Identity management is a process that helps protect your business data

- Applications must have security designed in
  - Delegation of responsibility
  - Multiple layers

- OpenEdge provides components of identity management
  - CLIENT-PRINCIPAL
  - Authentication Systems
  - Transportation of security token

Extra materials

- This session
  - Slides to be posted on PUG Challenge site
  - Supporting code at https://github.com/nwahmaet/IdM_Sample
- Other PUG Challenge sessions
  - Coding with Identity Management & Security (Part 2)
    Peter Judge, PSC
  - Advanced OpenEdge REST/Mobile Security
    Mike Jacobs, PSC
  - Programming with the Client-Principal Object
    Chris Longo, BravePoint

Image Credits:
Passport designed by Catia G, Time designed by wayne25uk, Database designed by Anton Outkine, Code designed by Nikhil Dev, 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 EXCHANGE 2013

DISCOVER. DEVELOP. DELIVER.

October 6–9, 2013 • Boston

#PRGS13

www.progress.com/exchange-pug

Special **low rate of $495** for PUG Challenge attendees with the code **PUGAM**

And visit the Progress booth to learn more about the Progress App Dev Challenge!

PROGRESS
software