Emerson: Scripting for Federated Virtual Worlds

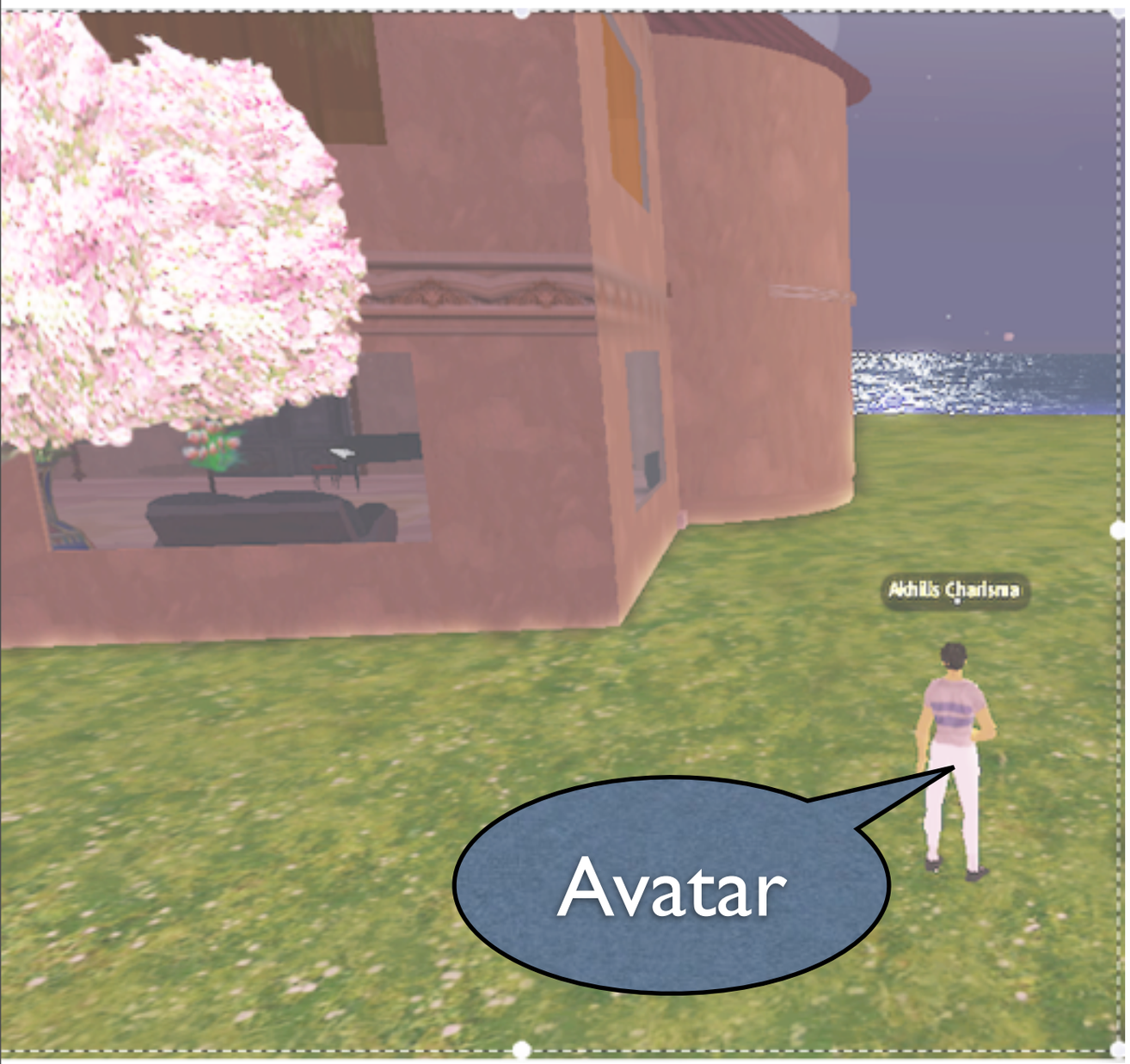
Bhupesh Chandra*, Ewen Cheslack-Postava*, Behram F.T. Mistree*, Philip Levis*, David Gay†
*Stanford University
†Intel Labs

Virtual Worlds

Second Life

WoW

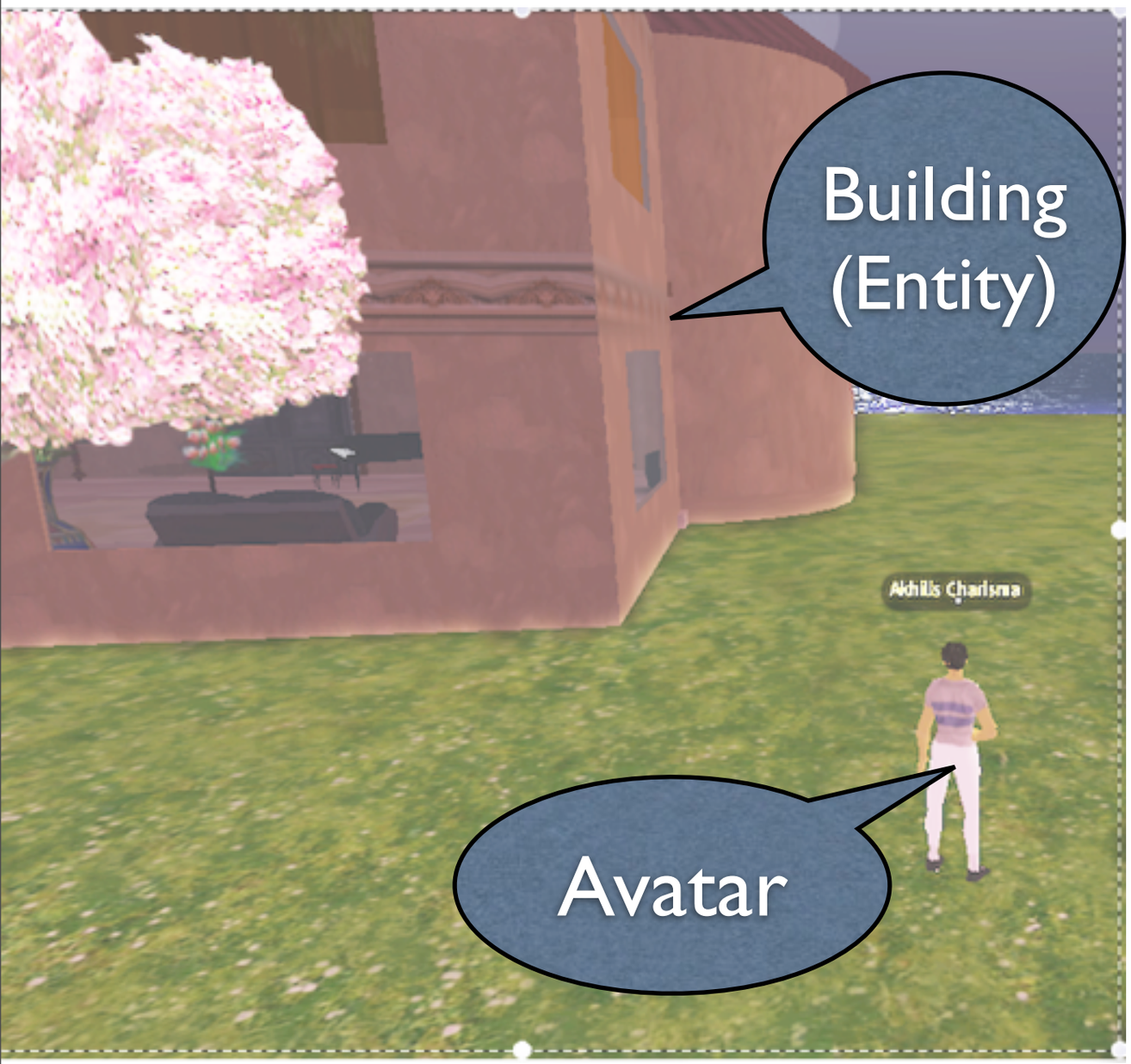
Virtual Worlds



Second Life

WoW

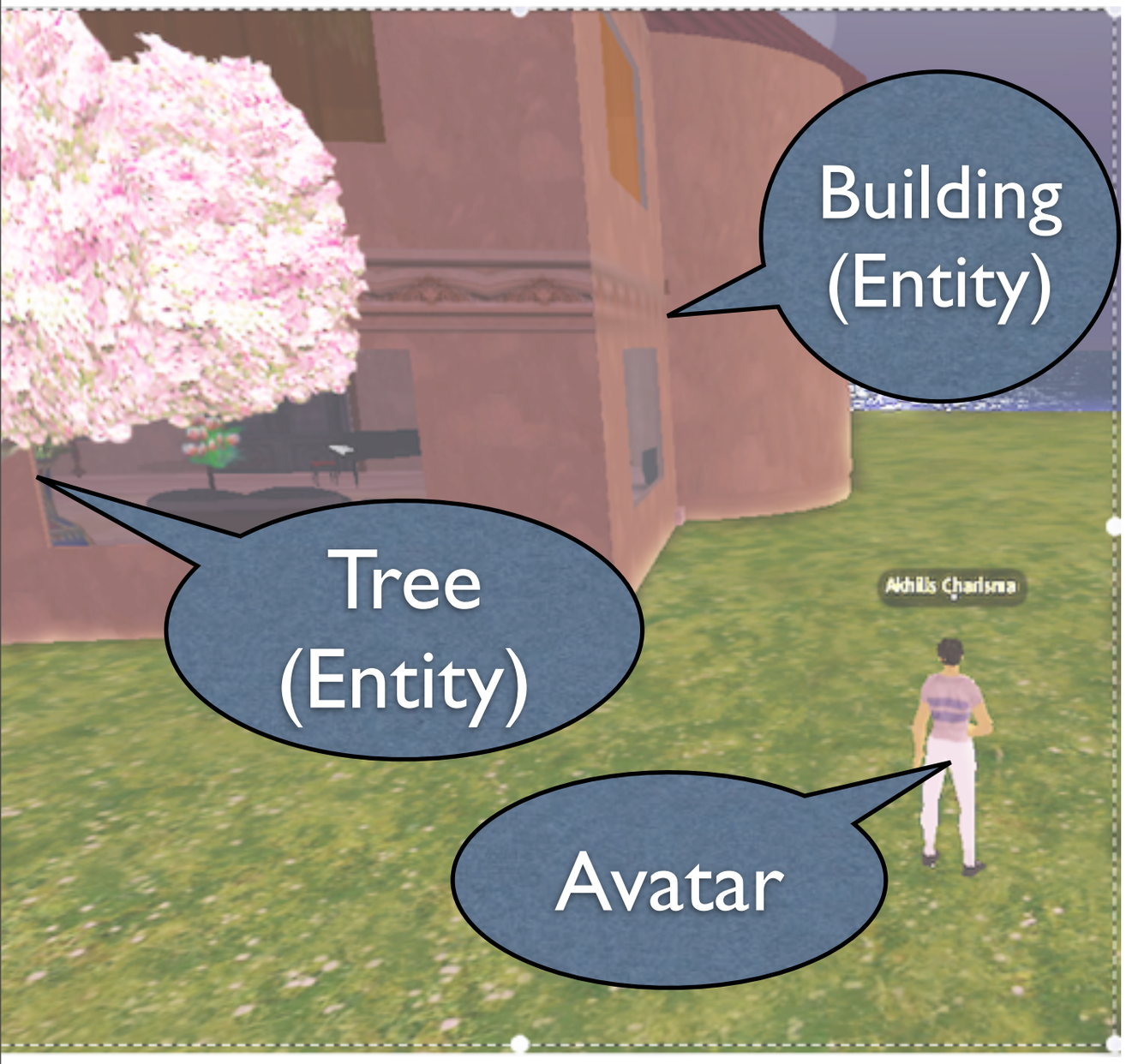
Virtual Worlds



Second Life

WoW

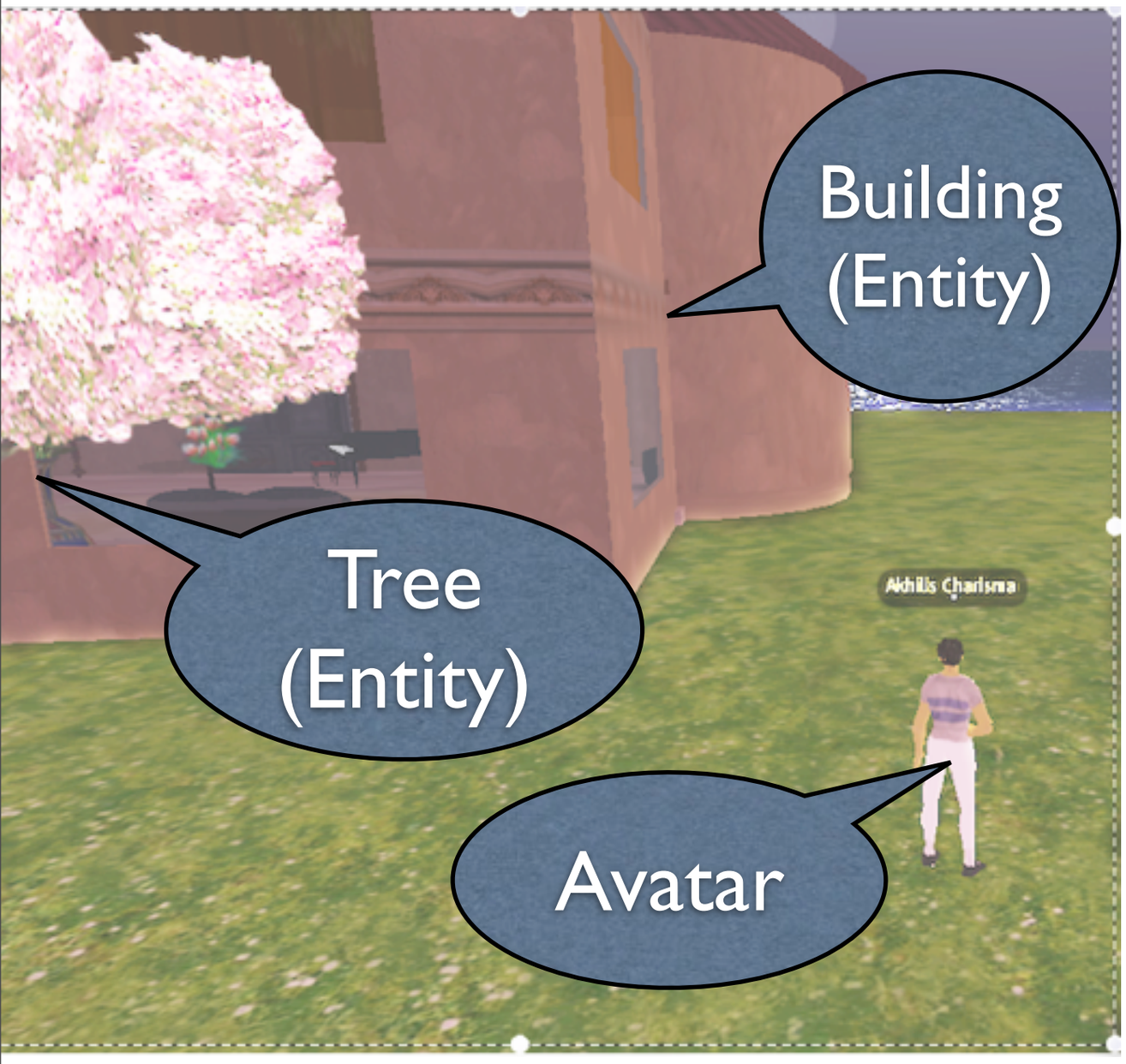
Virtual Worlds



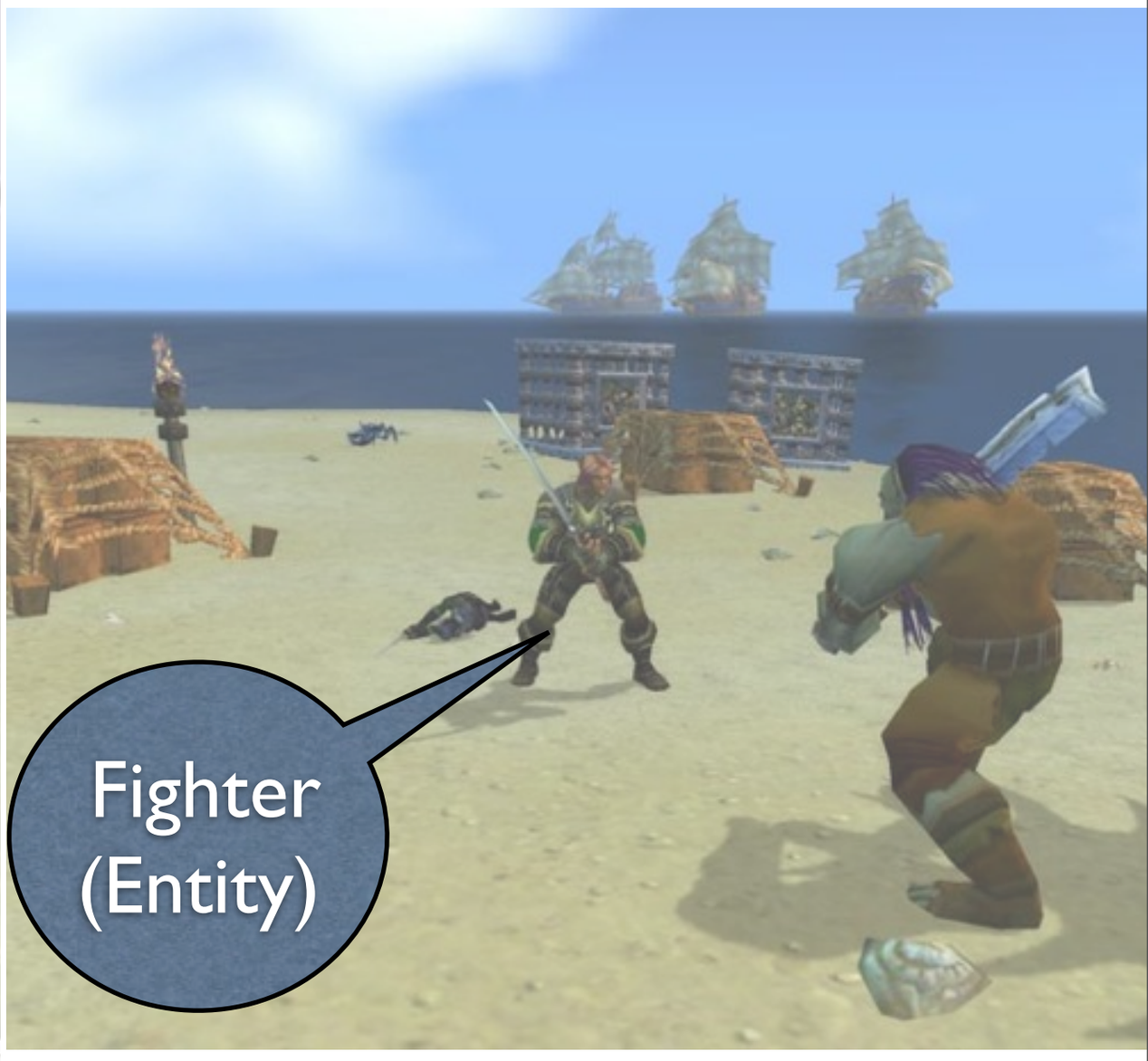
Second Life

WoW

Virtual Worlds

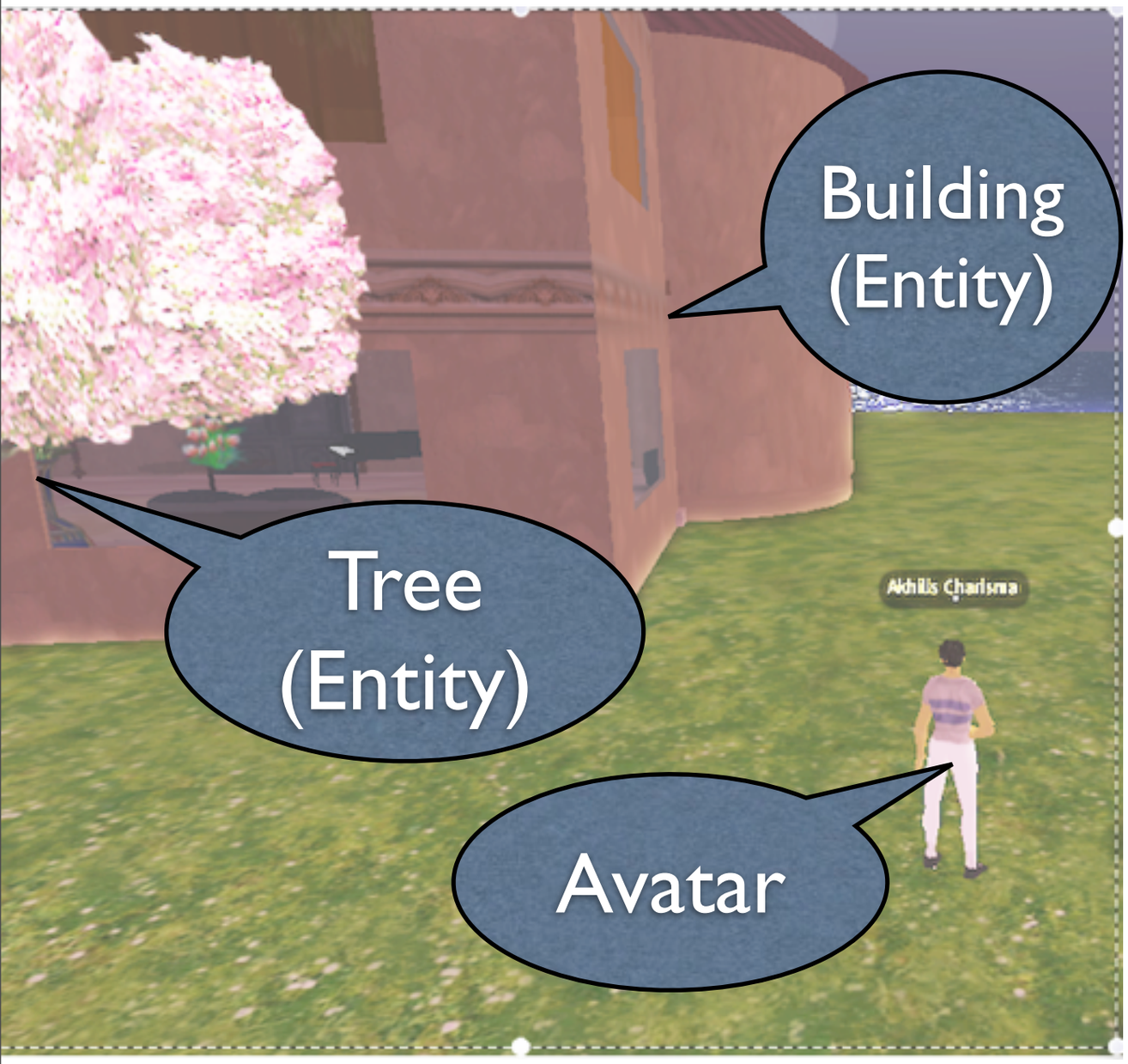


Second Life

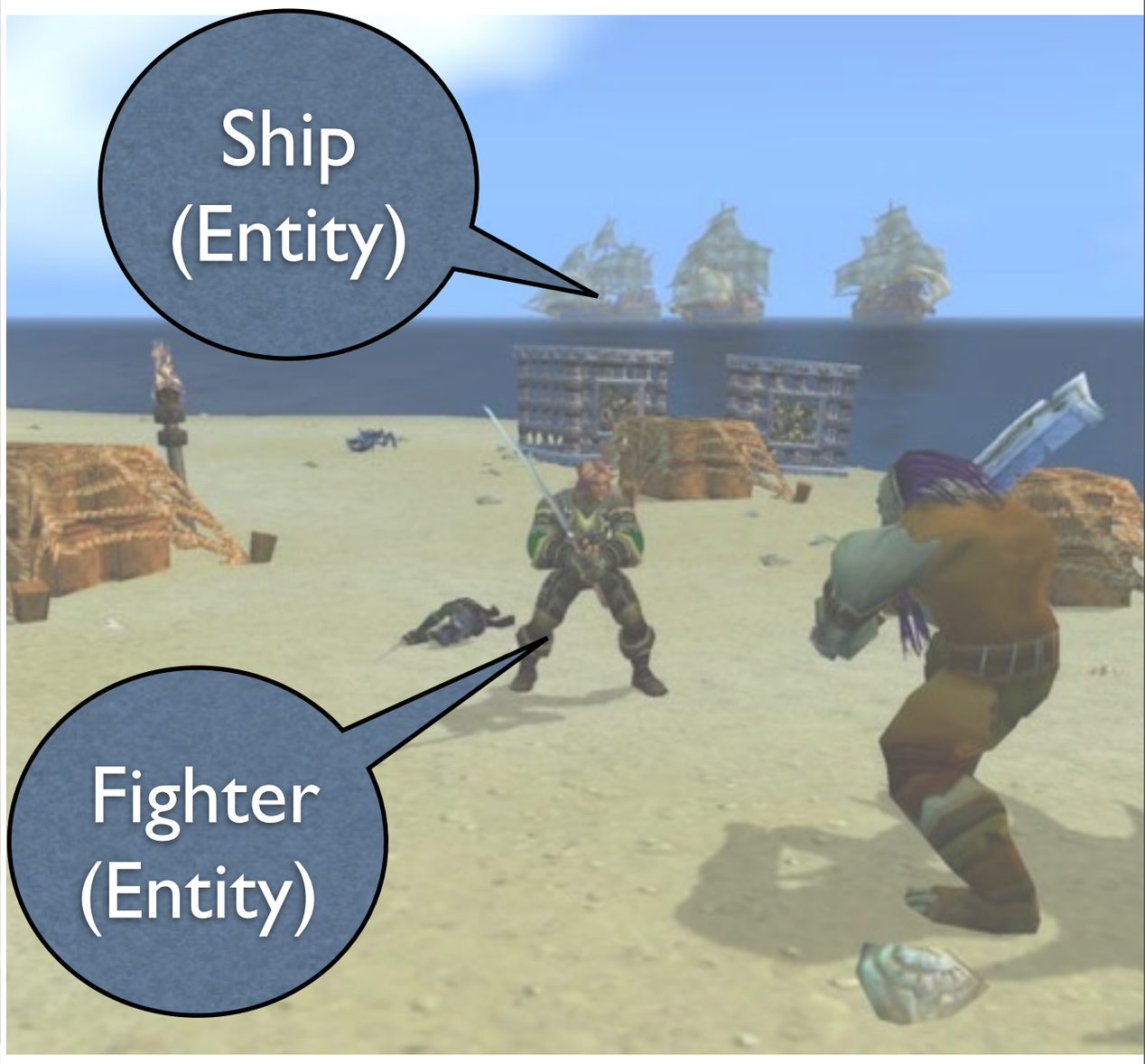


WoW

Virtual Worlds

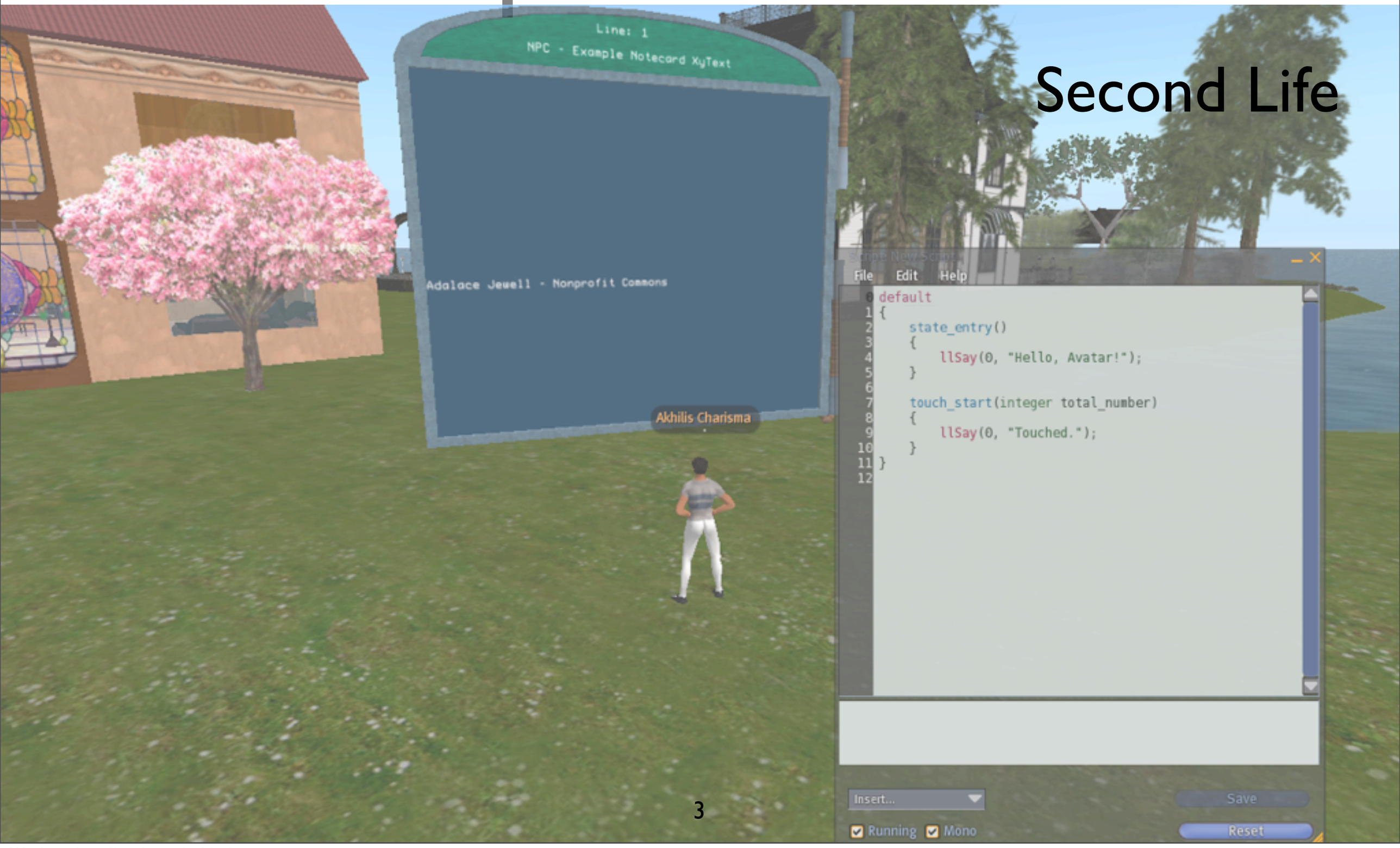


Second Life



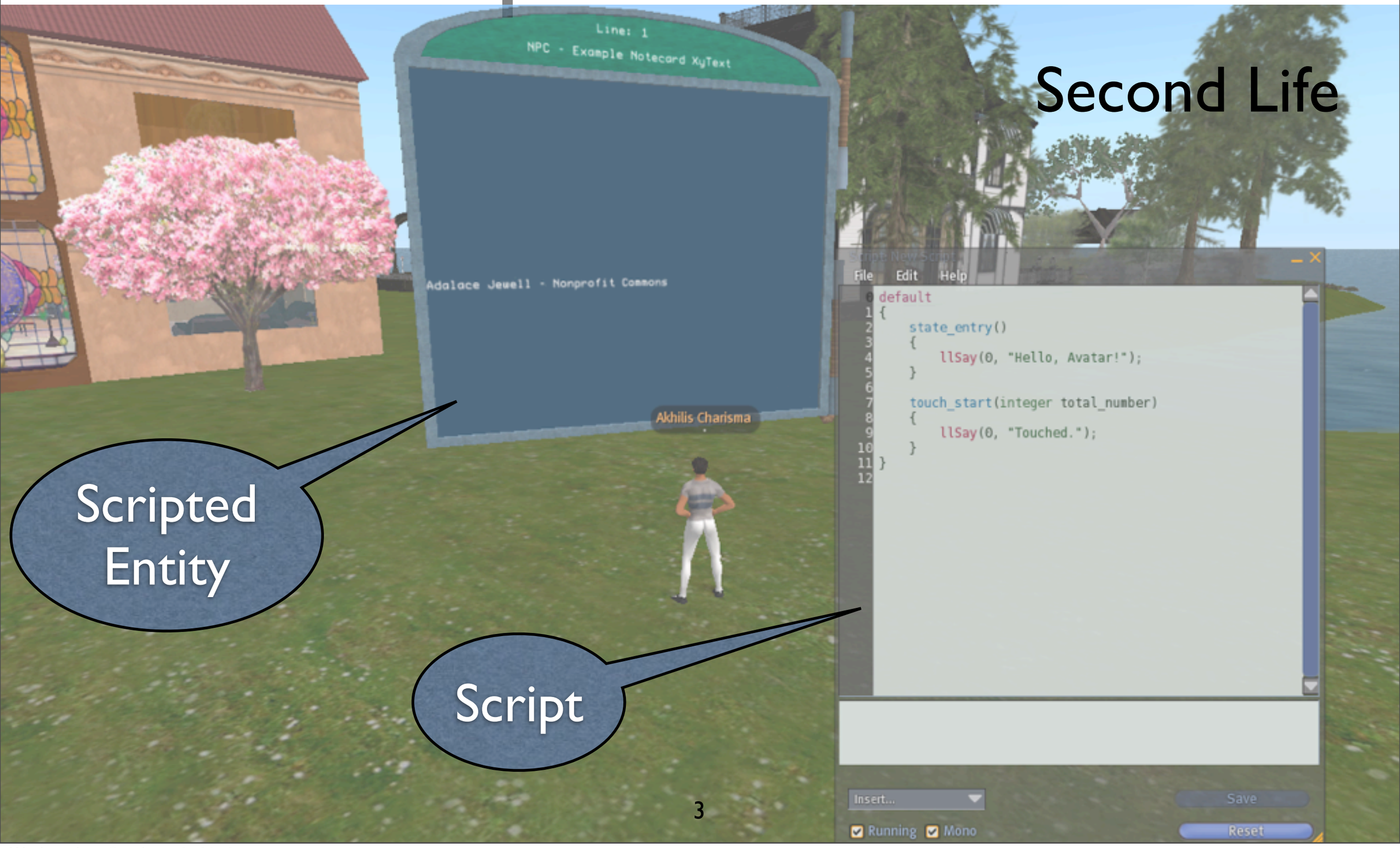
WoW

Scripted Entities



Scripted Entities

Scripted Entities



Scripting in VW

Scripting in VW

- Lua (WoW), LSL (Second Life), UScript (Unreal)

Scripting in VW

- Lua (WoW), LSL (Second Life), UScript (Unreal)
- Emerson
  - Scripting language for future VW
  - Easy to Script

Future Virtual Worlds

Future Virtual Worlds

- **Federation**
  - Multiple parties cooperate to run the world
  - Web users can not only create but host their own content => extensible, flexible

Future Virtual Worlds

- **Federation**
  - Multiple parties cooperate to run the world
  - Web users can not only create but host their own content => extensible, flexible
- **Seamless and Scalable**
  - Distributed simulation of billions of entities
  - Entities must interact over the network

Ease of Scripting

Ease of Scripting

- **Opportunistic programming**
  - Copy-paste and modify, code reuse

Ease of Scripting

- **Opportunistic programming**
  - Copy-paste and modify, code reuse
- **Iterative Development**
  - Continuously running world
  - Modify entity without terminating execution

Design Challenges

Design Challenges

- Lack of trust between entities
  - Protect against untrusted operations

Design Challenges

- Lack of trust between entities
  - Protect against untrusted operations
- Distributed Simulation of entities
  - Large latencies, packet losses and node failures

Design Challenges

- Lack of trust between entities
  - Protect against untrusted operations
- Distributed Simulation of entities
  - Large latencies, packet losses and node failures
- Live, incremental scripting

Emerson: Main Features

Emerson: Main Features

- **Entity, Presence, Object**
  - Federation and distributed simulation

Emerson: Main Features

- **Entity, Presence, Object**
  - Federation and distributed simulation
- **Prototyping**
  - Code reuse through prototypes

Emerson: Main Features

- **Entity, Presence, Object**
  - Federation and distributed simulation
- **Prototyping**
  - Code reuse through prototypes
- **Live Programming**
  - Execute code dynamically to modify behavior (more in the paper)

Emerson: Main Features

- **Entity, Presence, Object**
  - Federation and distributed simulation
- **Prototyping**
  - Code reuse through prototypes
- **Live Programming**
  - Execute code dynamically to modify behavior (more in the paper)
- **Event Driven Pattern Matching**
  - Address sphagetti if-else problem

Fundamentals

Fundamentals

- **Objects** encapsulate state and functionality

Fundamentals

- **Objects** encapsulate state and functionality
- **Entities** contain objects, event handlers
  - Obtain presences
  - Communicate with entities in the same world
  - Objects exist and are addressable within entity

Fundamentals

- **Objects** encapsulate state and functionality
- **Entities** contain objects, event handlers
  - Obtain presences
  - Communicate with entities in the same world
  - Objects exist and are addressable within entity
- **Presence** is connection of entity in the world
  - Geometry, communication

Art Gallery

Art Gallery

VirtualWorld

Art Gallery

LONDON

Entity Host

Artist Entity Host

Art Gallery

LONDON

Artist
Entity

Artist Entity Host

Art Gallery

VirtualWorld

LONDON

Entity

Entity Host

Artist
Entity

Artist Entity Host

Art Gallery

VirtualWorld

LONDON

Artist
Avatar
(Presence)

Entity

Entity Host

Artist
Entity

Artist Entity Host

Art Gallery

VirtualWorld

LONDON

Artist
Avatar
(Presence)

Entity

Entity

Entity Host

Artist
Entity

Artist Entity Host

Art Gallery Entity

Art Gallery

VirtualWorld

LONDON

Artist
Avatar
(Presence)

Entity

Entity

Entity Host

Artist
Entity

Artist Entity Host

Art Gallery Entity

Art Gallery

VirtualWorld

LONDON

Artist
Avatar
(Presence)

Art Gallery
(Presence)

Entity

Entity

Entity Host

Artist
Entity

Artist Entity Host

Art Gallery Entity

Art Gallery

VirtualWorld

LONDON

Artist Avatar (Presence)

Art Gallery (Presence)

Entity

Artist Entity

Artist Entity Host

Object

Entity

root

inventory

getArtDetails()

Art Gallery Entity

Entity Host

Art Gallery

VirtualWorld

LONDON

Artist Avatar (Presence)

Art Gallery (Presence)

Entity

Entity

Entity Host

Object

Artist Entity

Artist Entity Host

root

getArtDetails()

inventory

handleLoan()

Art Gallery Entity

handler

action:borrow
item_id

Entities

Entities

- Communicate by sending asynchronous messages over the network
  - Short event handlers; don't block other entities (Helps Seamless Scaling)

Entities

- Communicate by sending asynchronous messages over the network
  - Short event handlers; don't block other entities (Helps Seamless Scaling)
- Boundary of trust (Helps Federation)
  - Exclusive right to change their state/ behavior

Code Reuse

Code Reuse

- Important for easy scripting

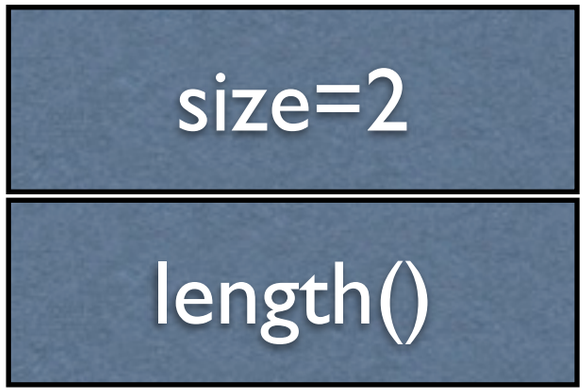
Code Reuse

- Important for easy scripting
- Class based Inheritance (Java)
  - Subclassing

Code Reuse

- Important for easy scripting
- Class based Inheritance (Java)
  - Subclassing
- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects

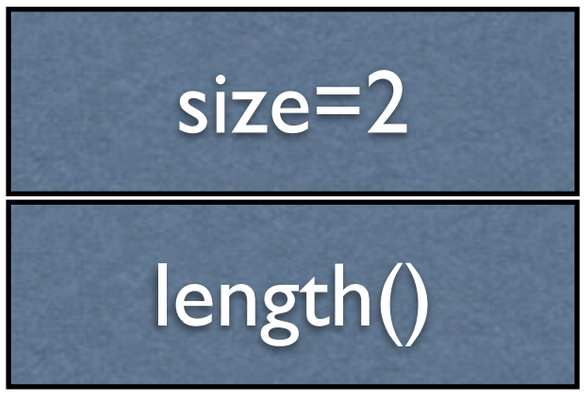
Code Reuse

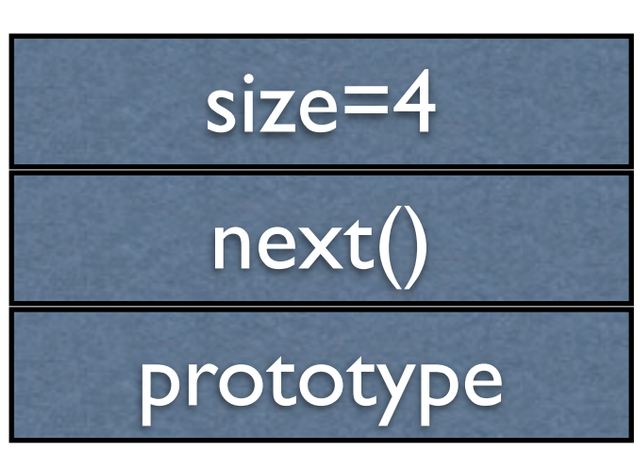


- Important for easy scripting
- Class based Inheritance (Java)
  - Subclassing
- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects

Code Reuse

- Important for easy scripting
- Class based Inheritance (Java)
  - Subclassing
- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects

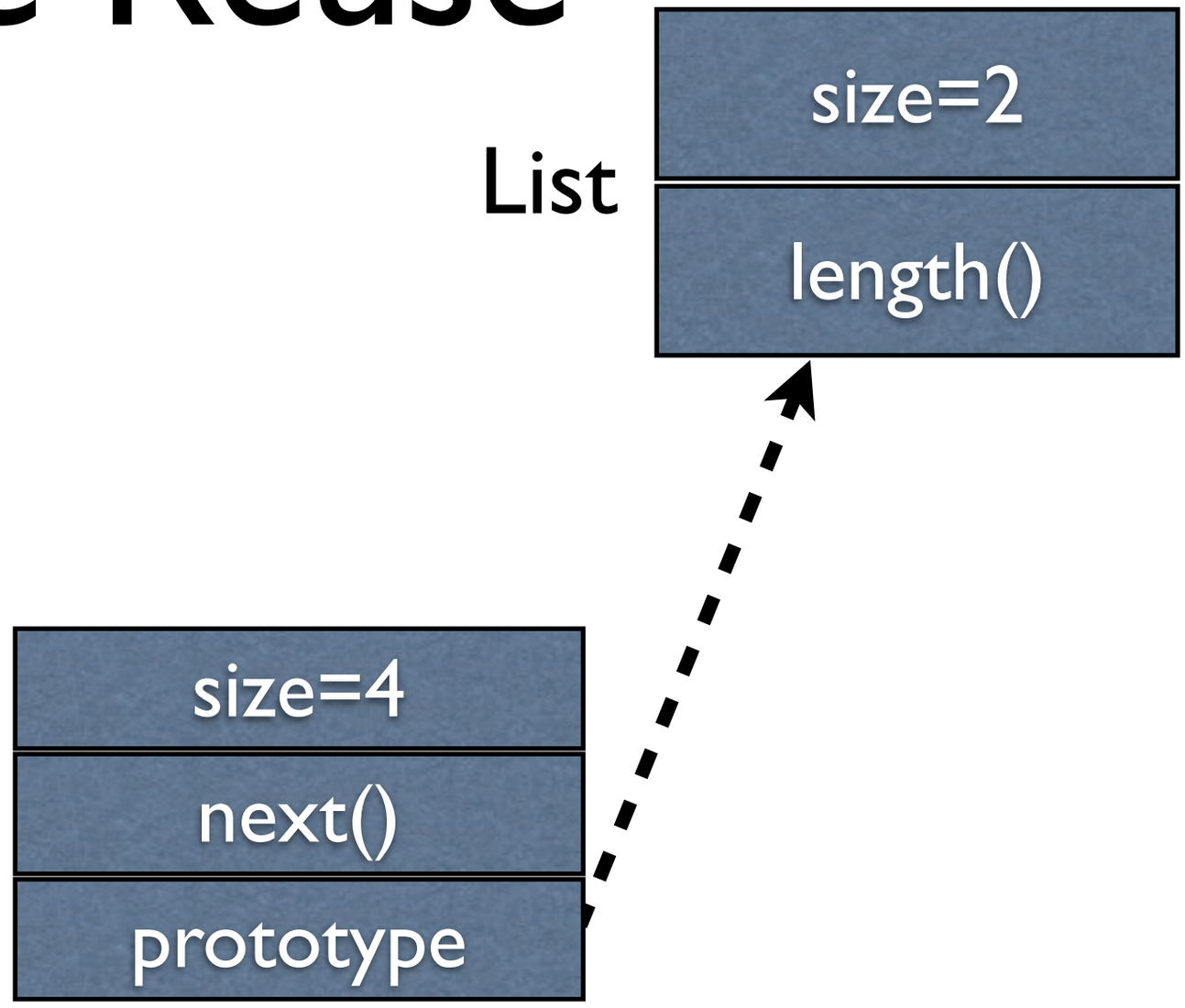




CustomerQueue

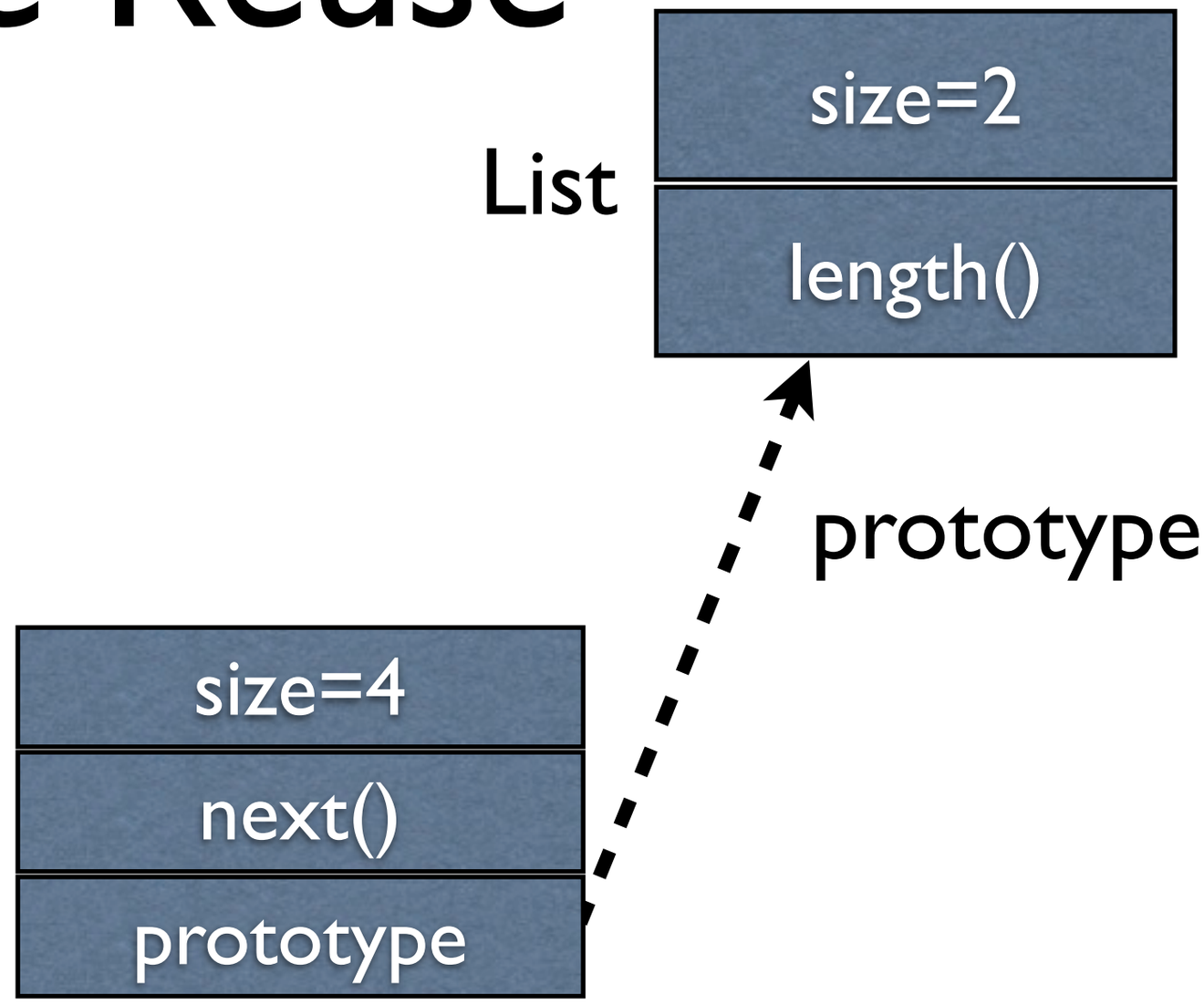
Code Reuse

- Important for easy scripting
- Class based Inheritance (Java)
  - Subclassing
- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects



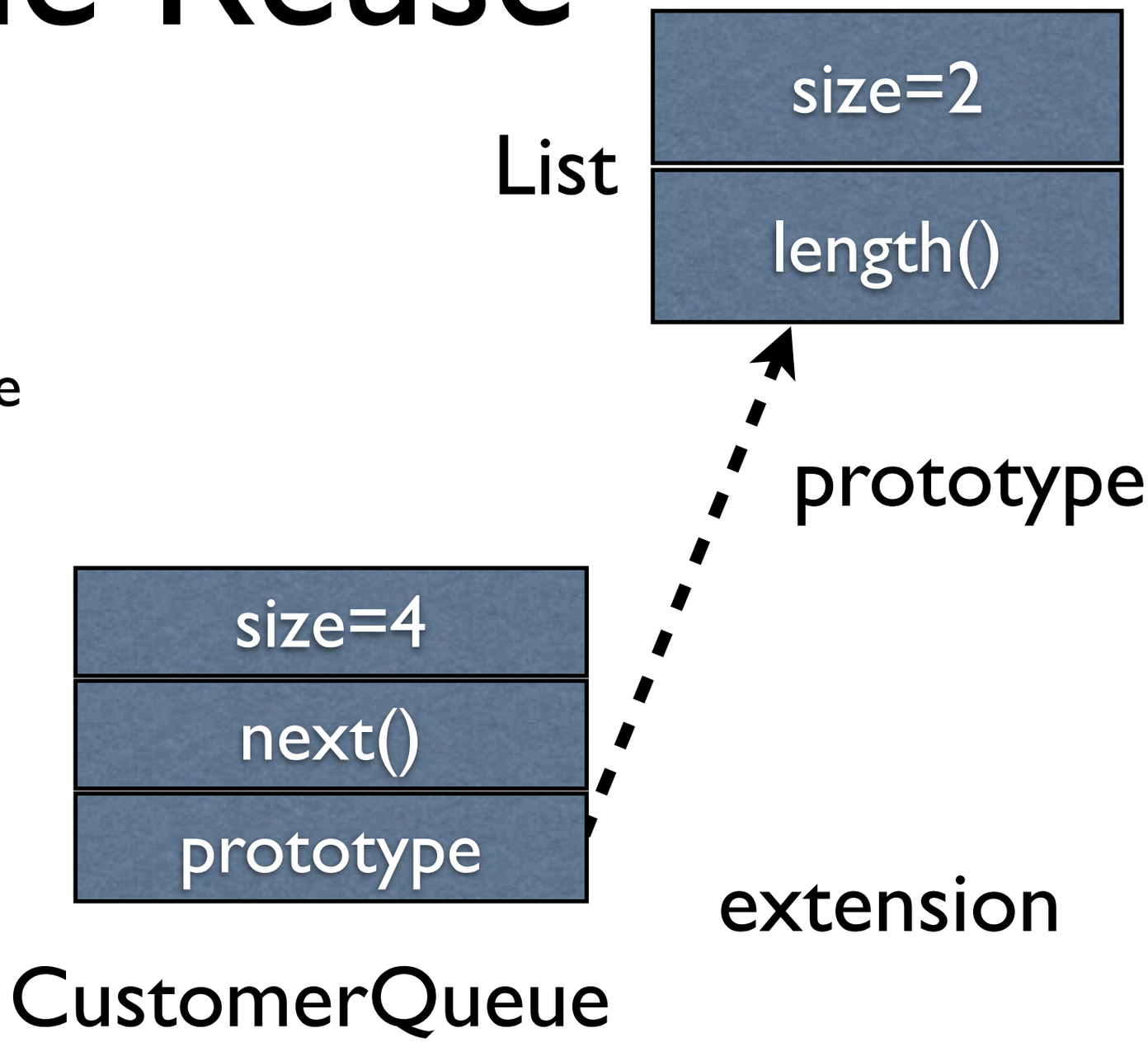
Code Reuse

- Important for easy scripting
- Class based Inheritance (Java)
  - Subclassing
- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects



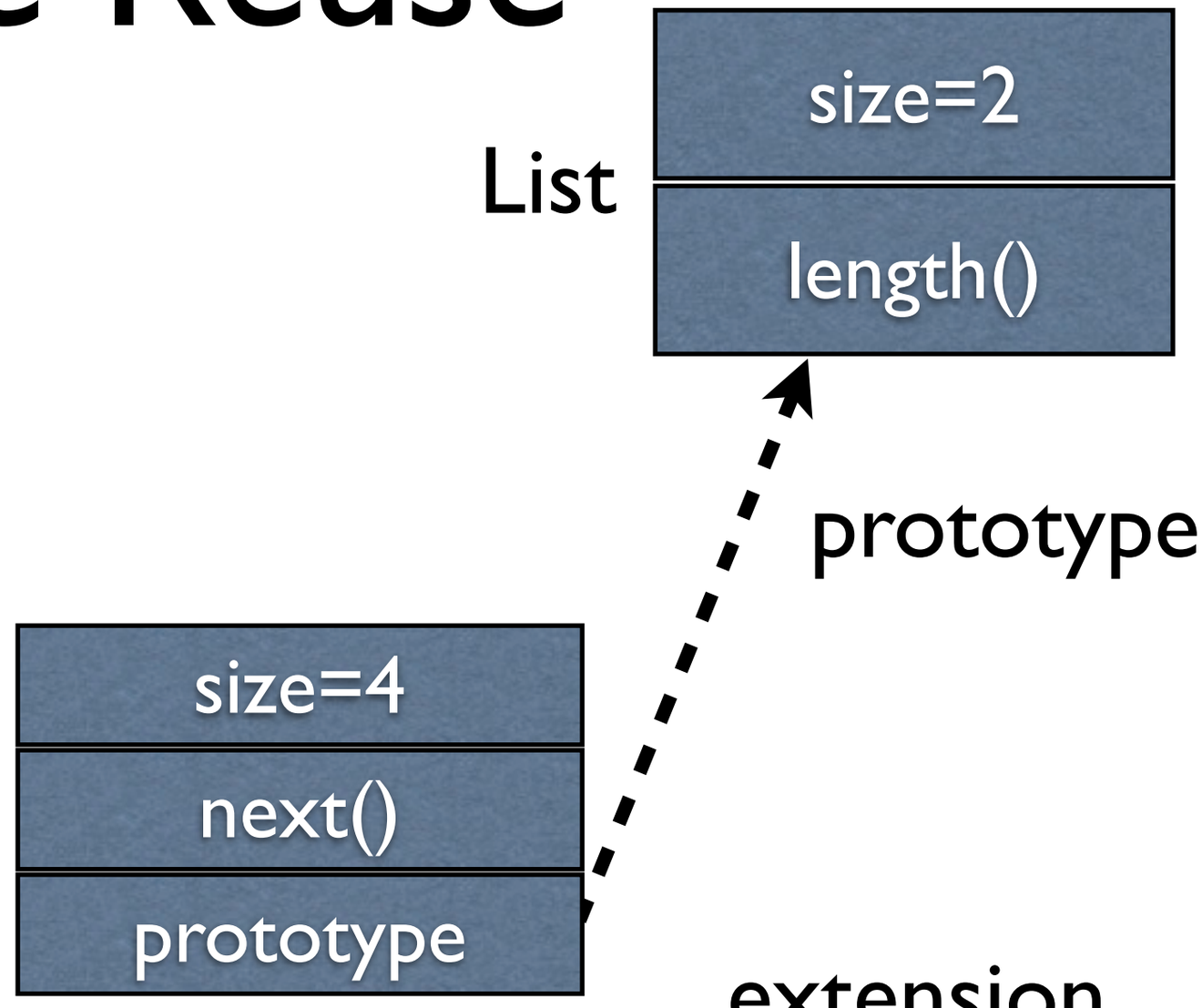
Code Reuse

- Important for easy scripting
- Class based Inheritance (Java)
  - Subclassing
- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects



Code Reuse

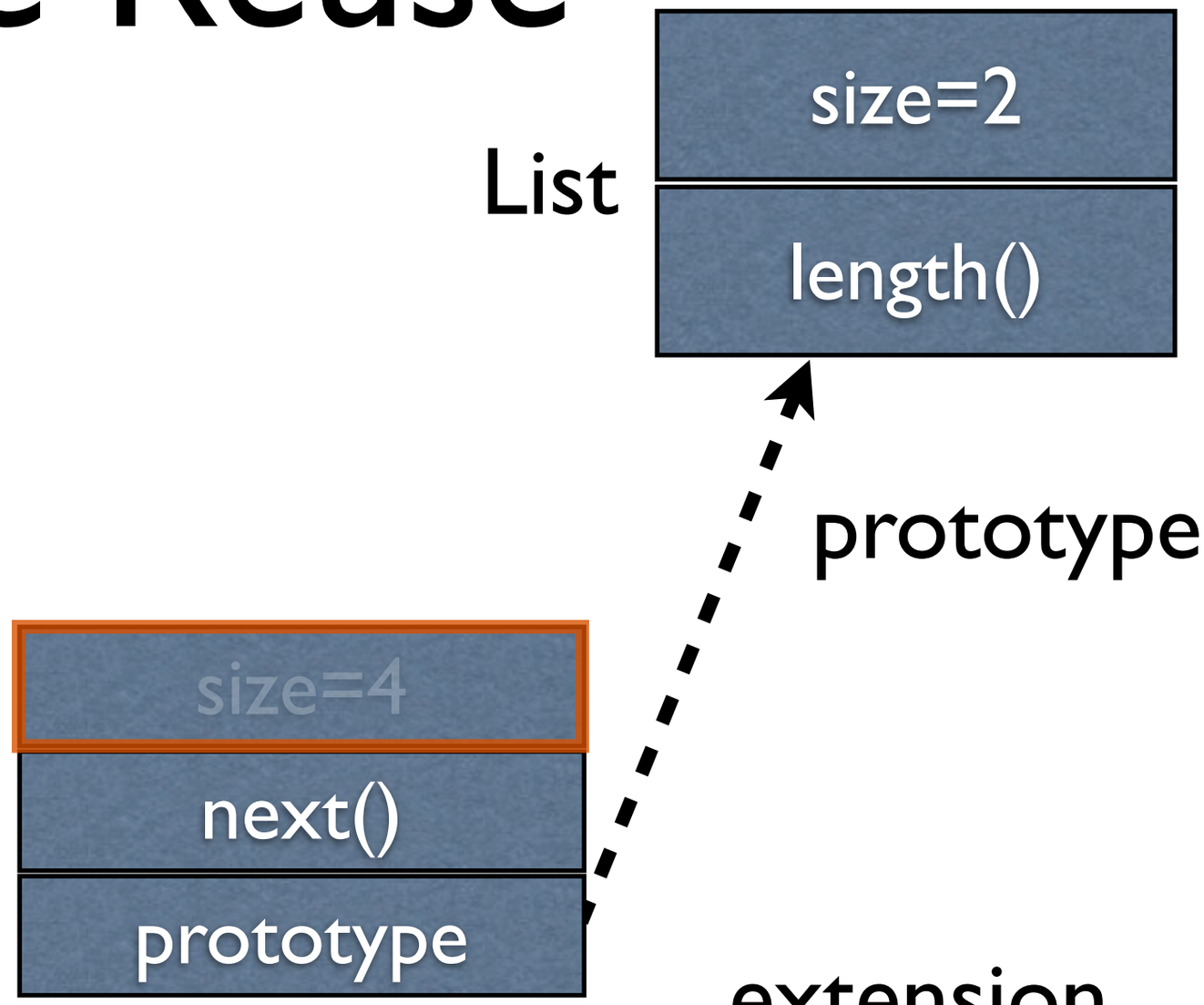
- Important for easy scripting
- Class based Inheritance (Java)
  - Subclassing
- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects



CustomerQueue.length() ?

Code Reuse

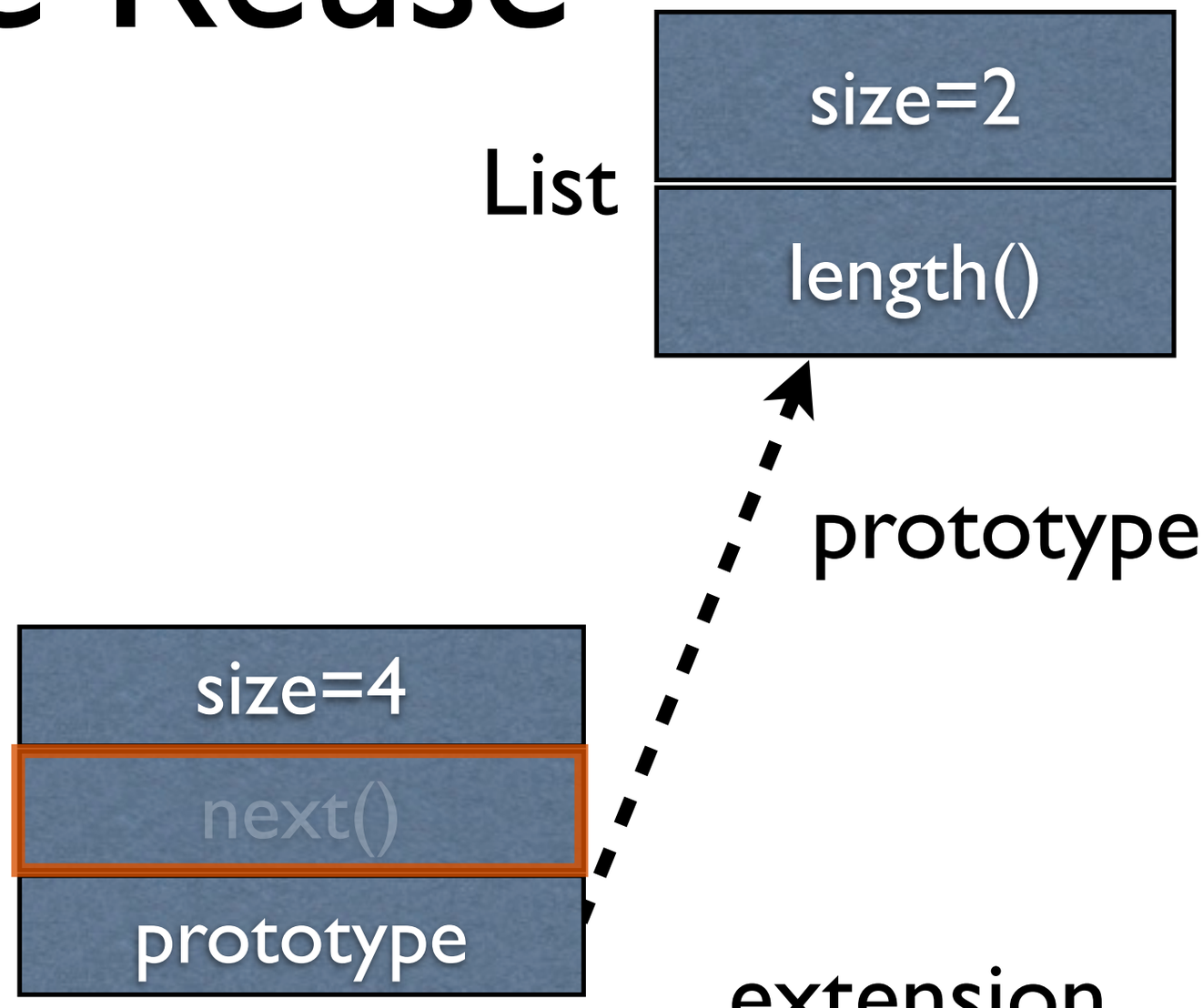
- Important for easy scripting
- Class based Inheritance (Java)
  - Subclassing
- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects



CustomerQueue.length() ?

Code Reuse

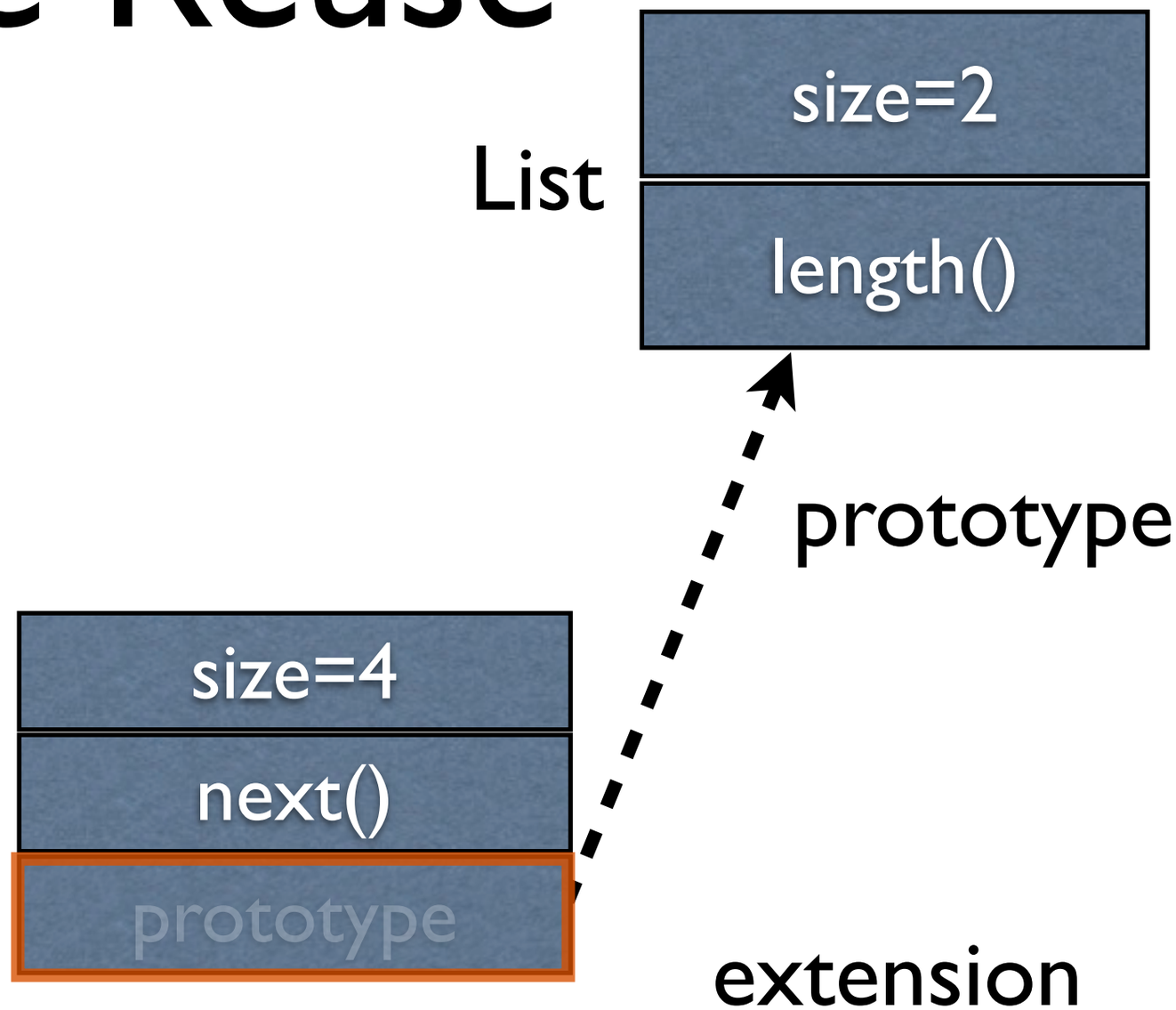
- Important for easy scripting
- Class based Inheritance (Java)
  - Subclassing
- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects



CustomerQueue.length() ?

Code Reuse

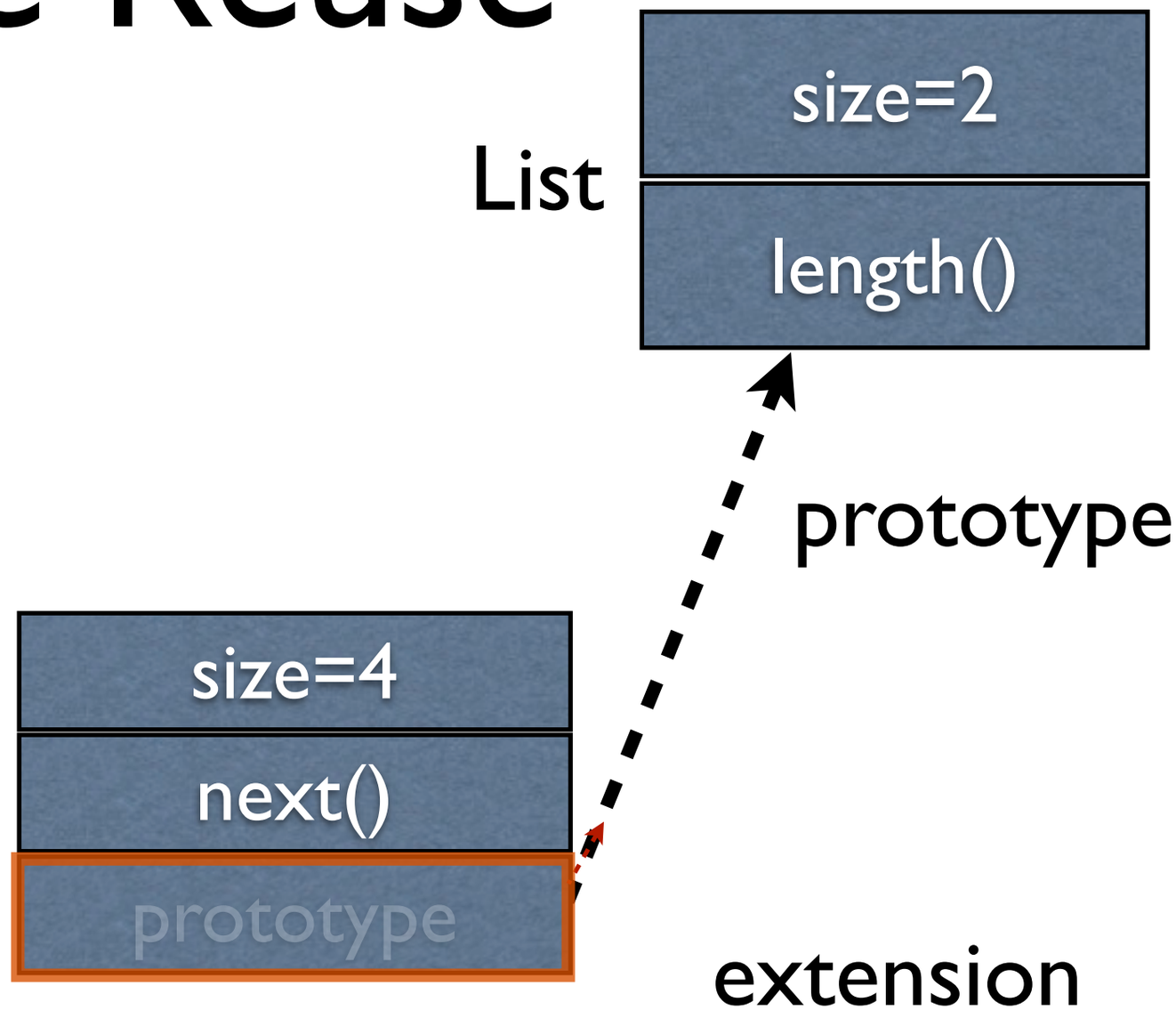
- Important for easy scripting
- Class based Inheritance (Java)
  - Subclassing
- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects



CustomerQueue.length() ?

Code Reuse

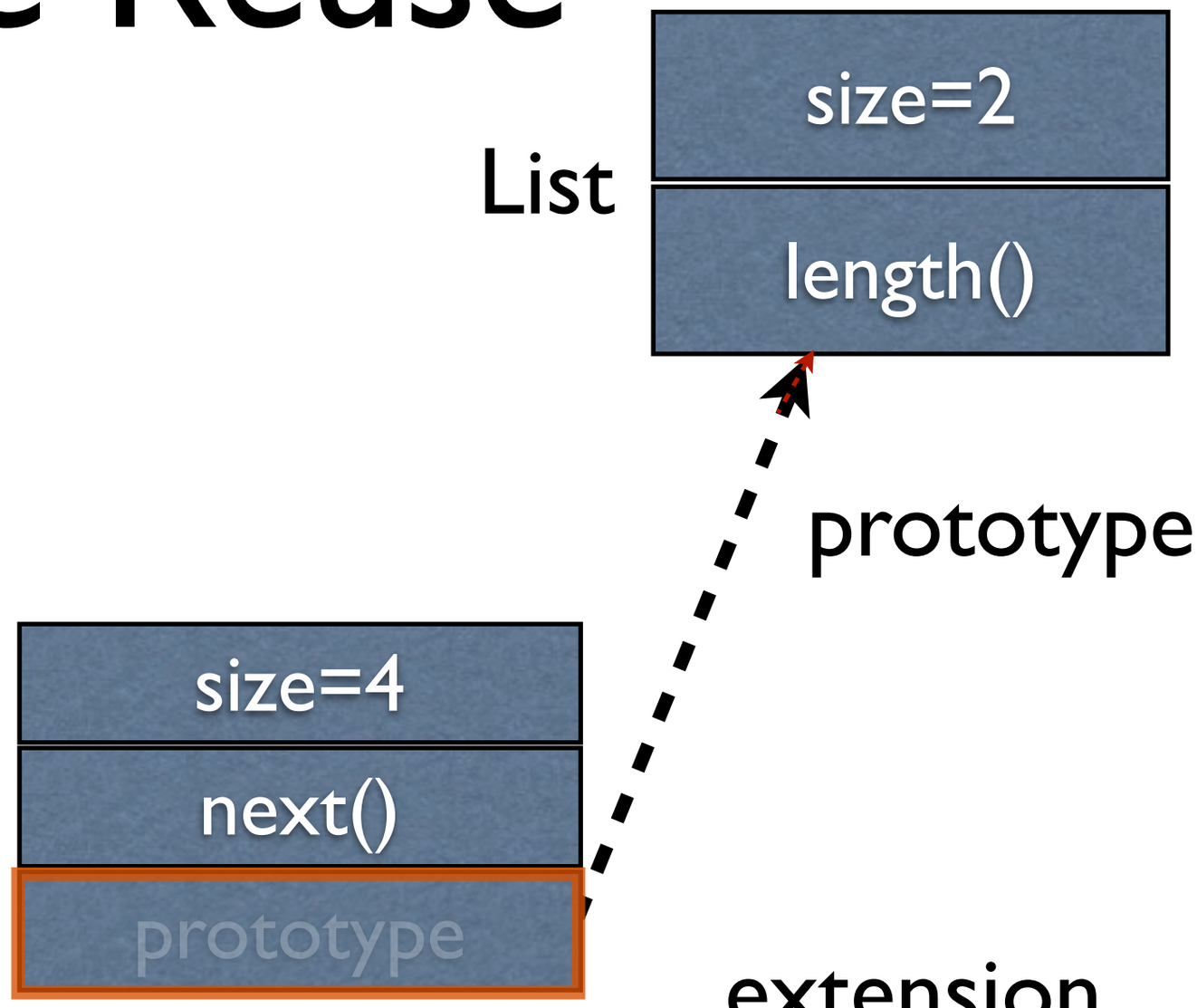
- Important for easy scripting
- Class based Inheritance (Java)
  - Subclassing
- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects



CustomerQueue.length() ?

Code Reuse

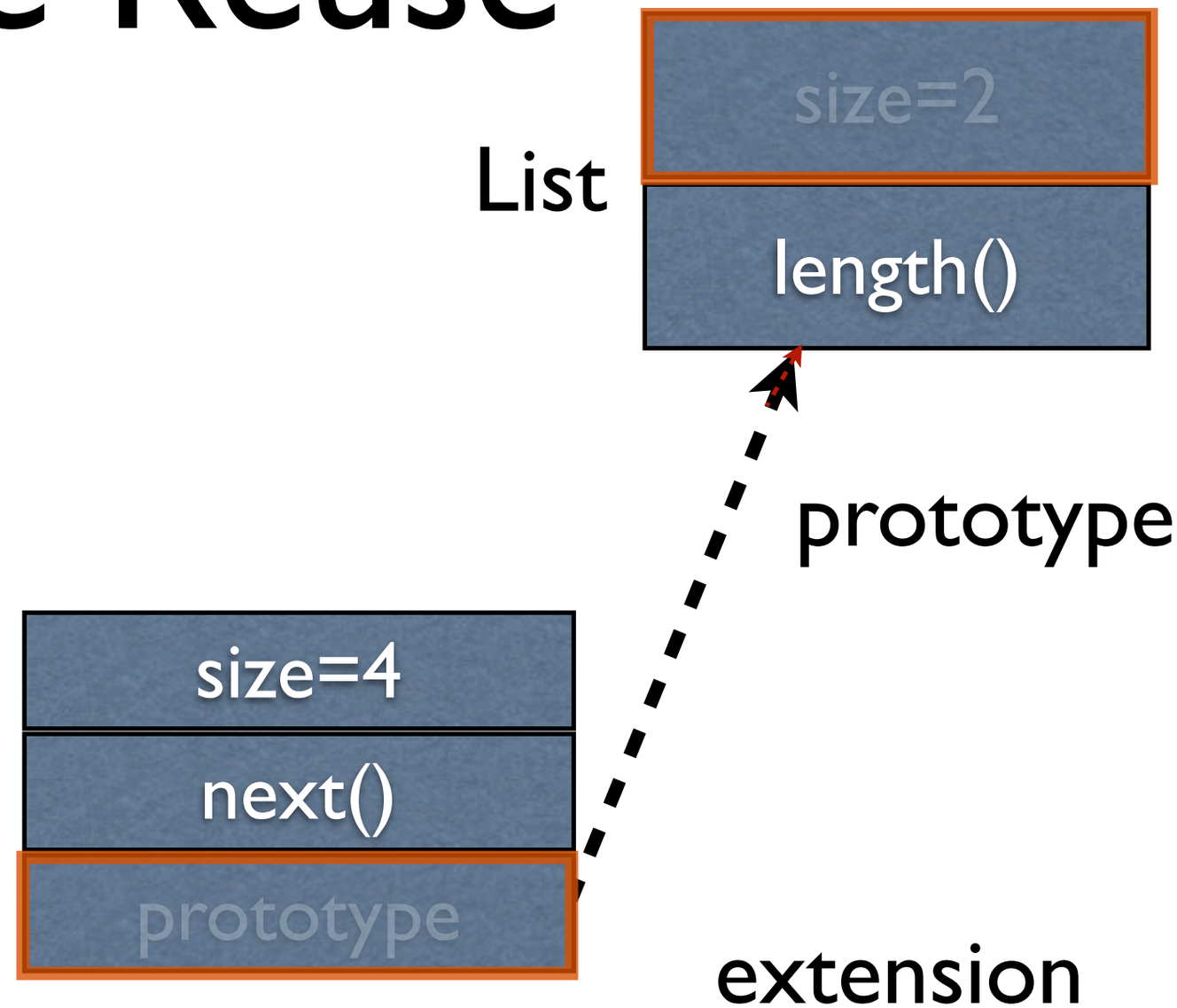
- Important for easy scripting
- Class based Inheritance (Java)
  - Subclassing
- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects



CustomerQueue.length() ?

Code Reuse

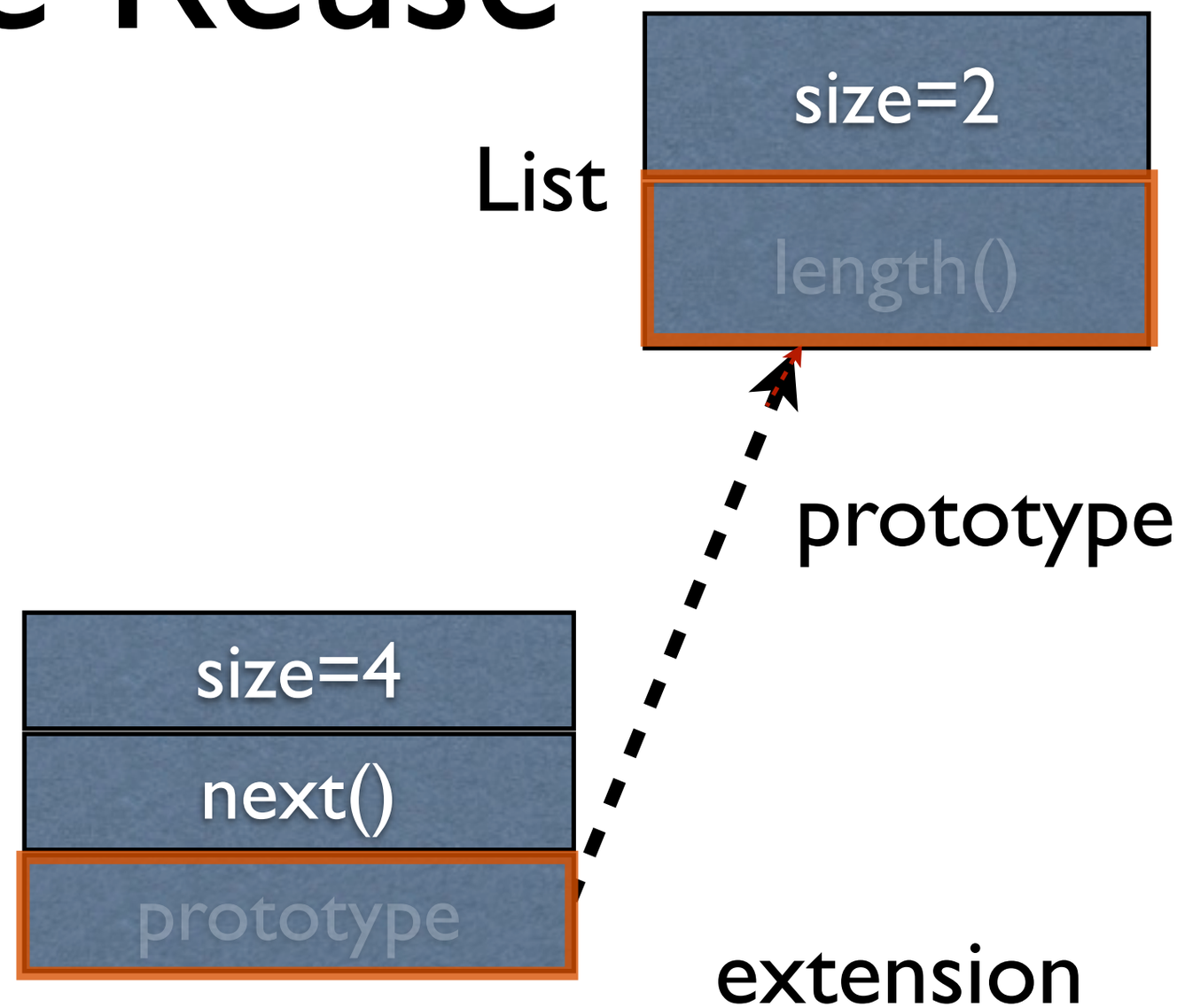
- Important for easy scripting
- Class based Inheritance (Java)
  - Subclassing
- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects



CustomerQueue.length() ?

Code Reuse

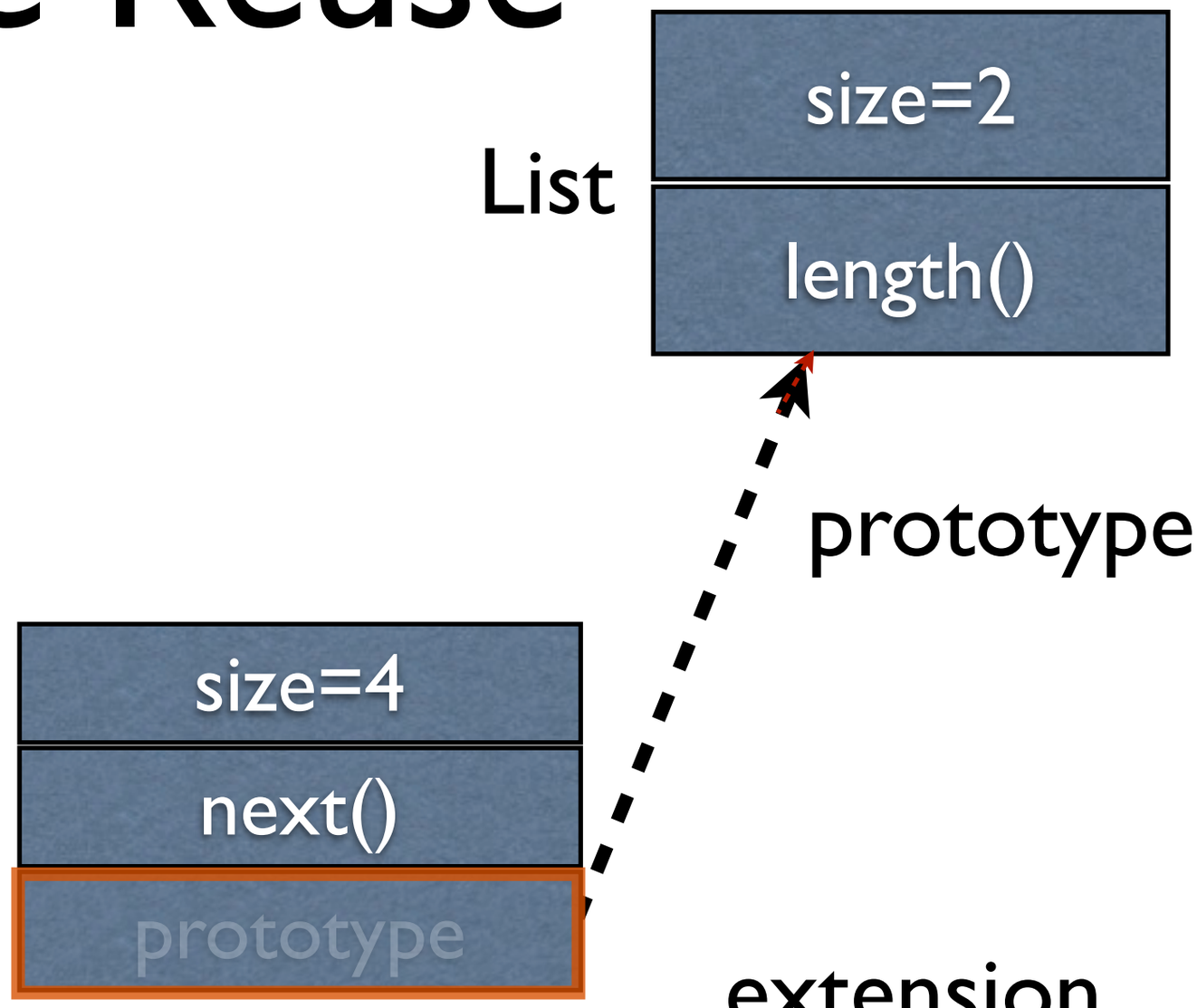
- Important for easy scripting
- Class based Inheritance (Java)
  - Subclassing
- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects



CustomerQueue.length() ?

Code Reuse

- Important for easy scripting
- Class based Inheritance (Java)
  - Subclassing
- Prototype based (JavaScript)
  - No Classes
  - Objects inherit from objects



CustomerQueue.length() ?

Prototypes for Objects

Prototypes for Objects

- Singleton classes are wasteful

Prototypes for Objects

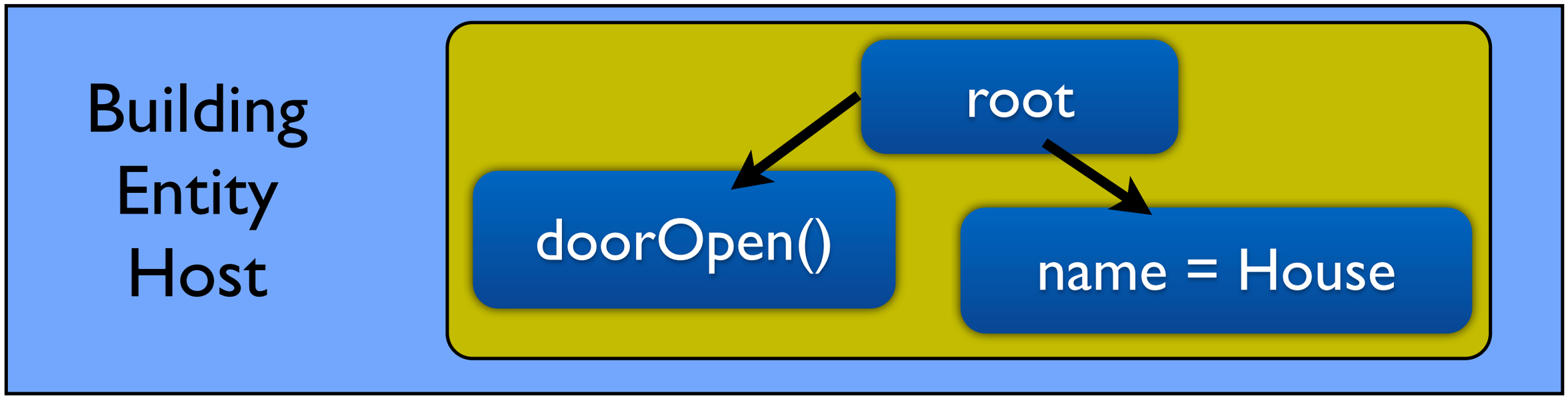
- Singleton classes are wasteful
- Modify class then re-instantiate in some languages

Prototypes for Objects

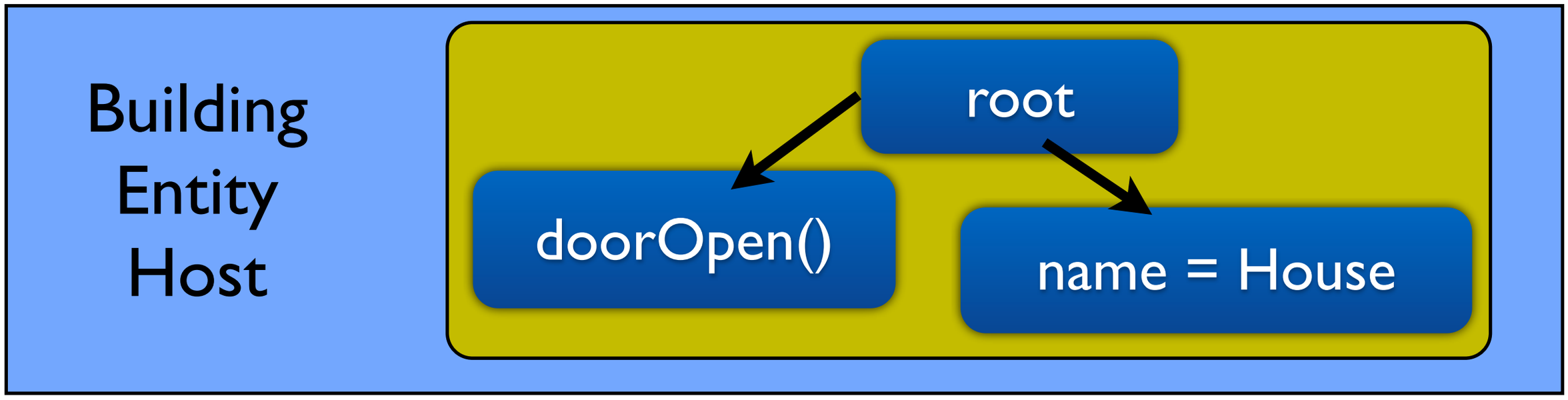
- Singleton classes are wasteful
- Modify class then re-instantiate in some languages
- Emerson objects inherit "live" modifications to their prototype

Inheritance: Bad For Entities

Inheritance: Bad For Entities

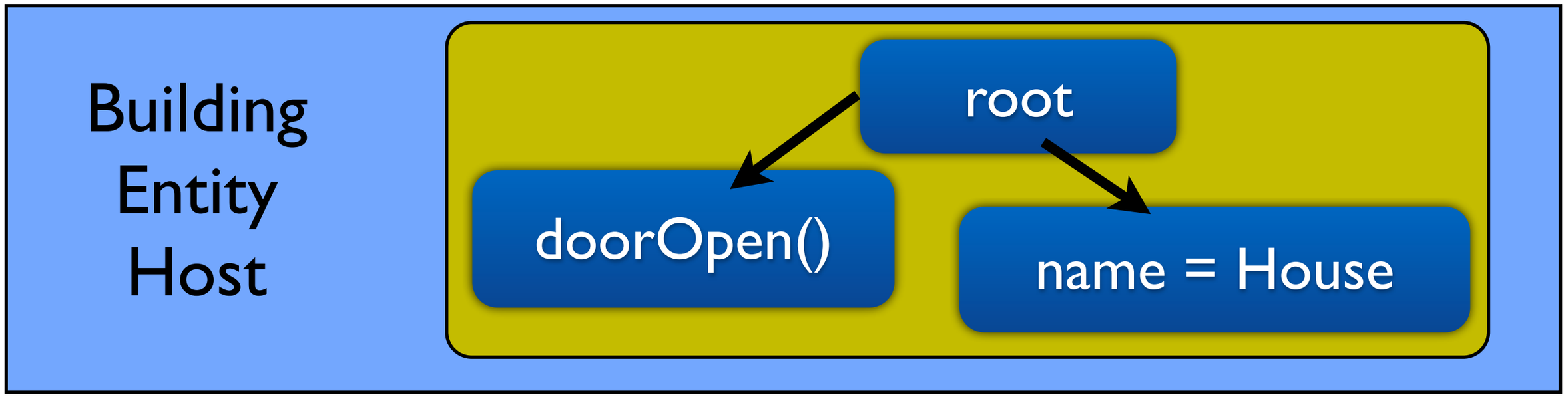


Inheritance: Bad For Entities

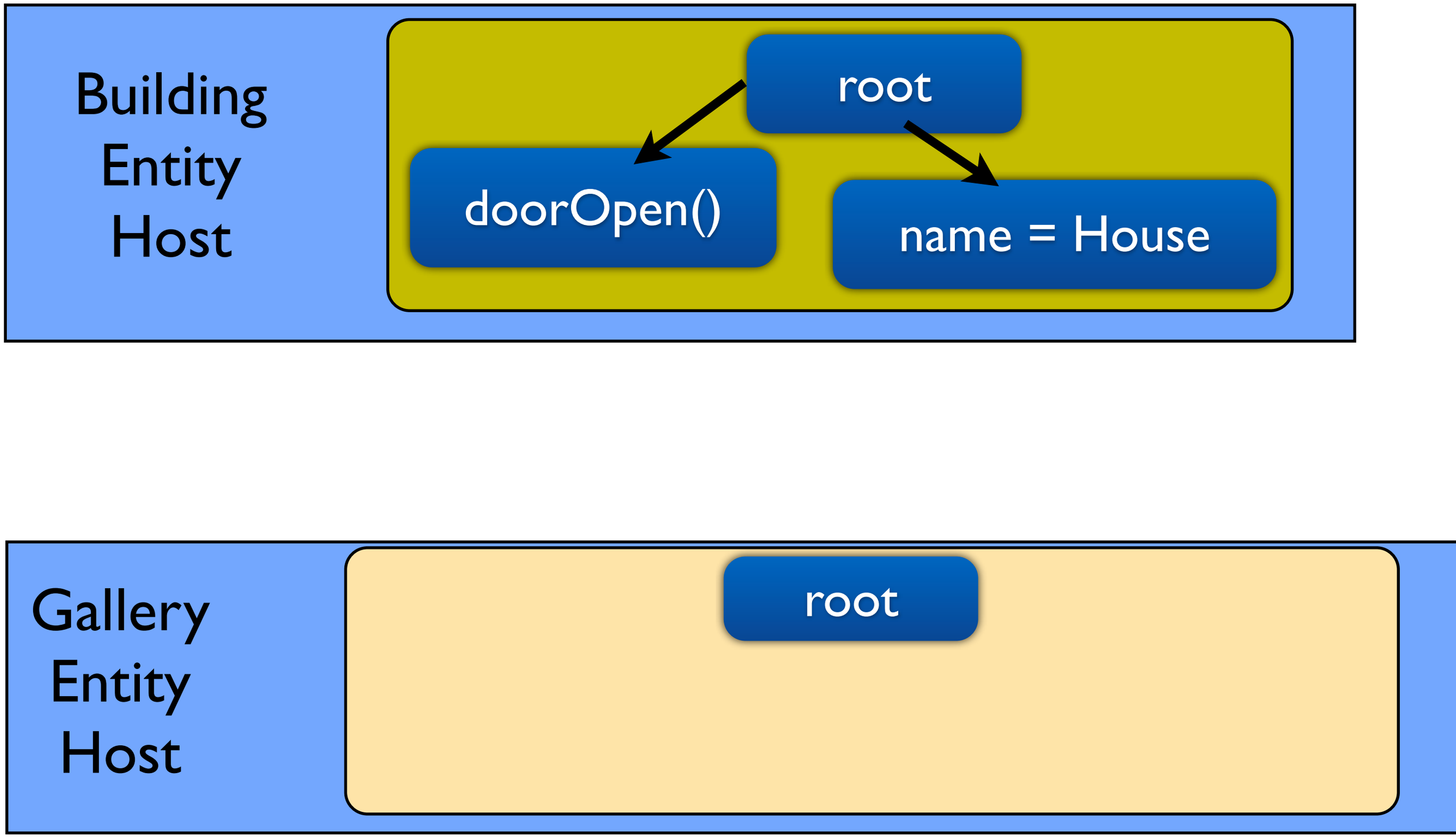


Gallery
Entity
Host

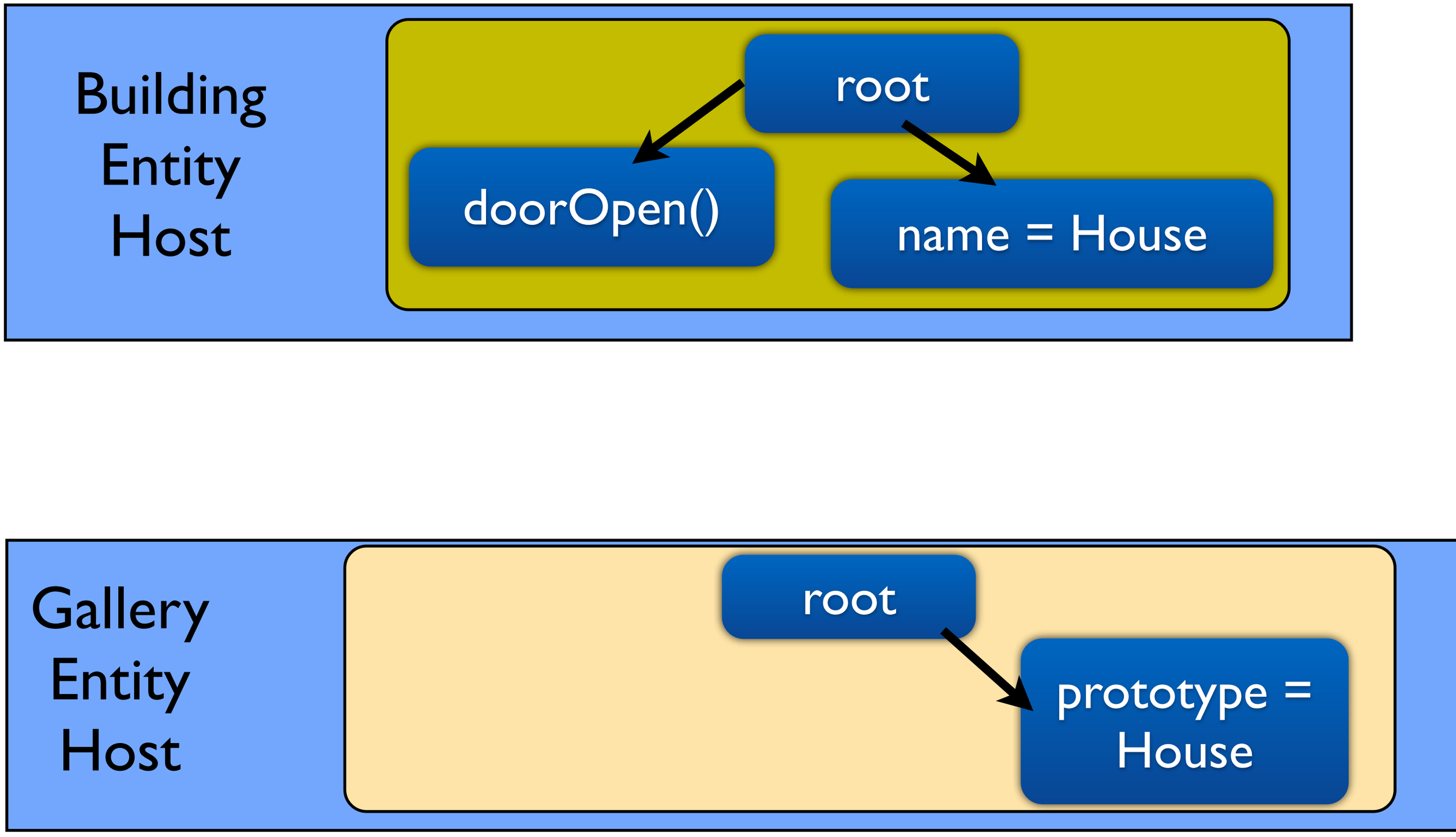
Inheritance: Bad For Entities



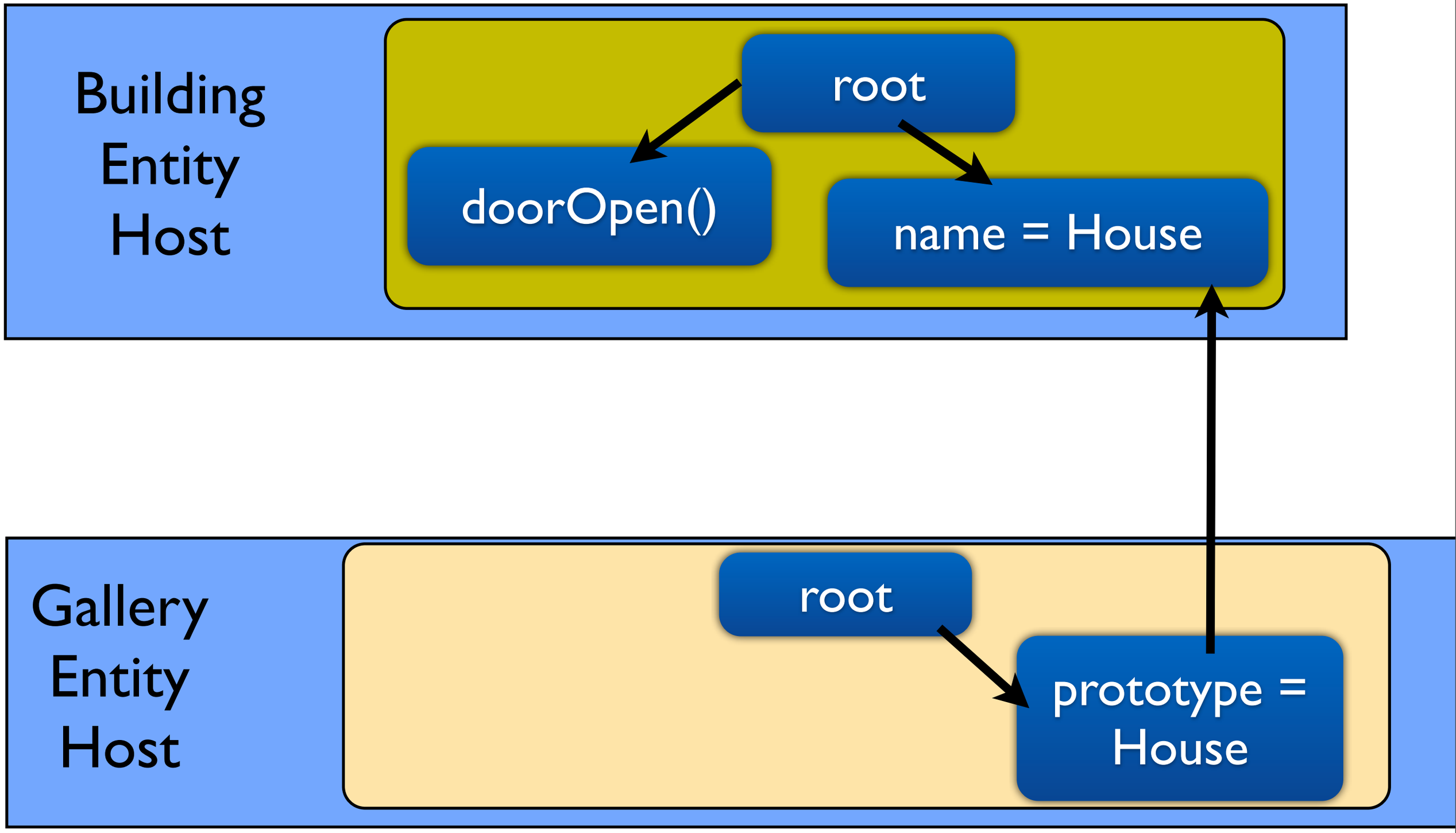
Inheritance: Bad For Entities



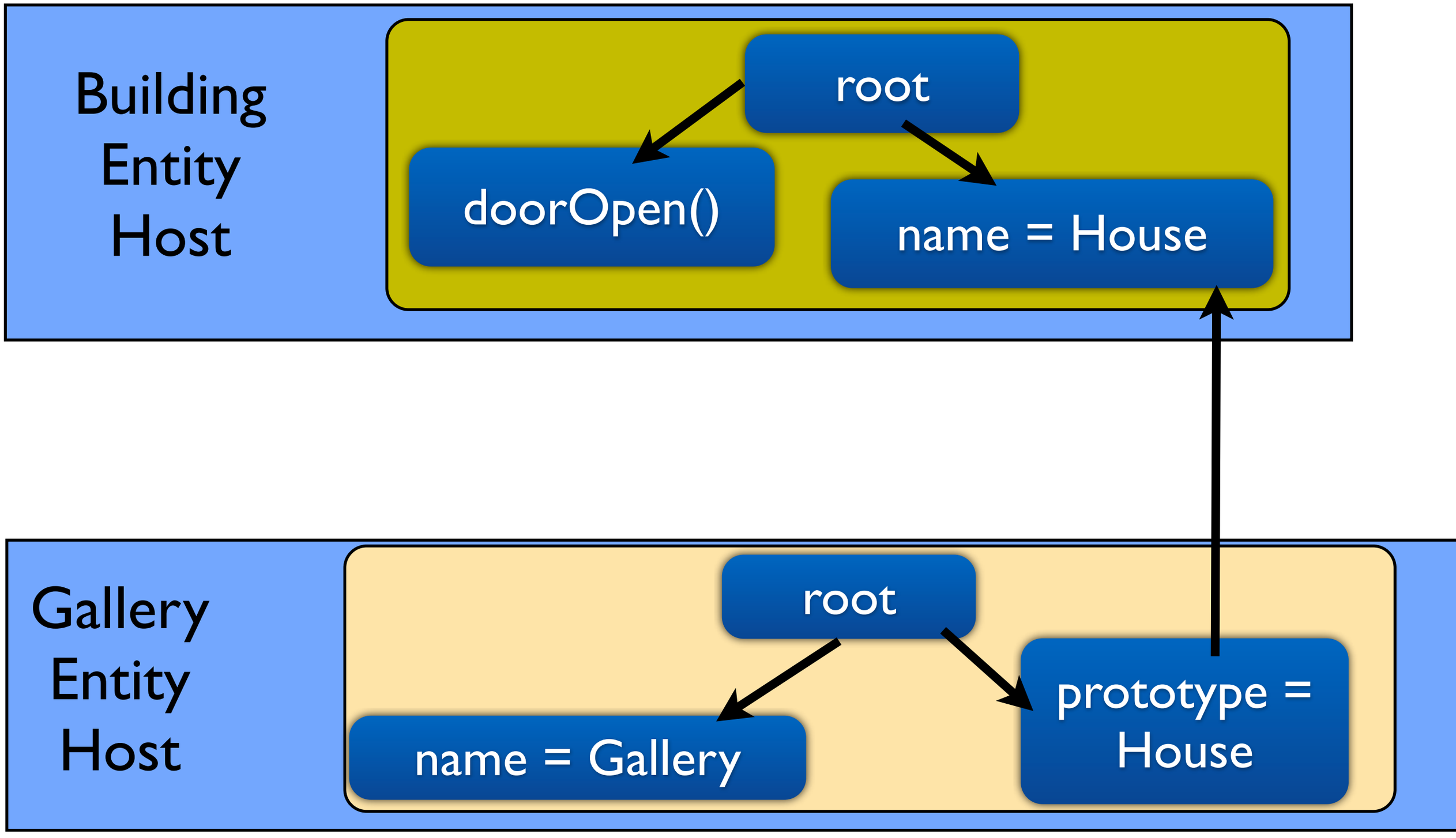
Inheritance: Bad For Entities



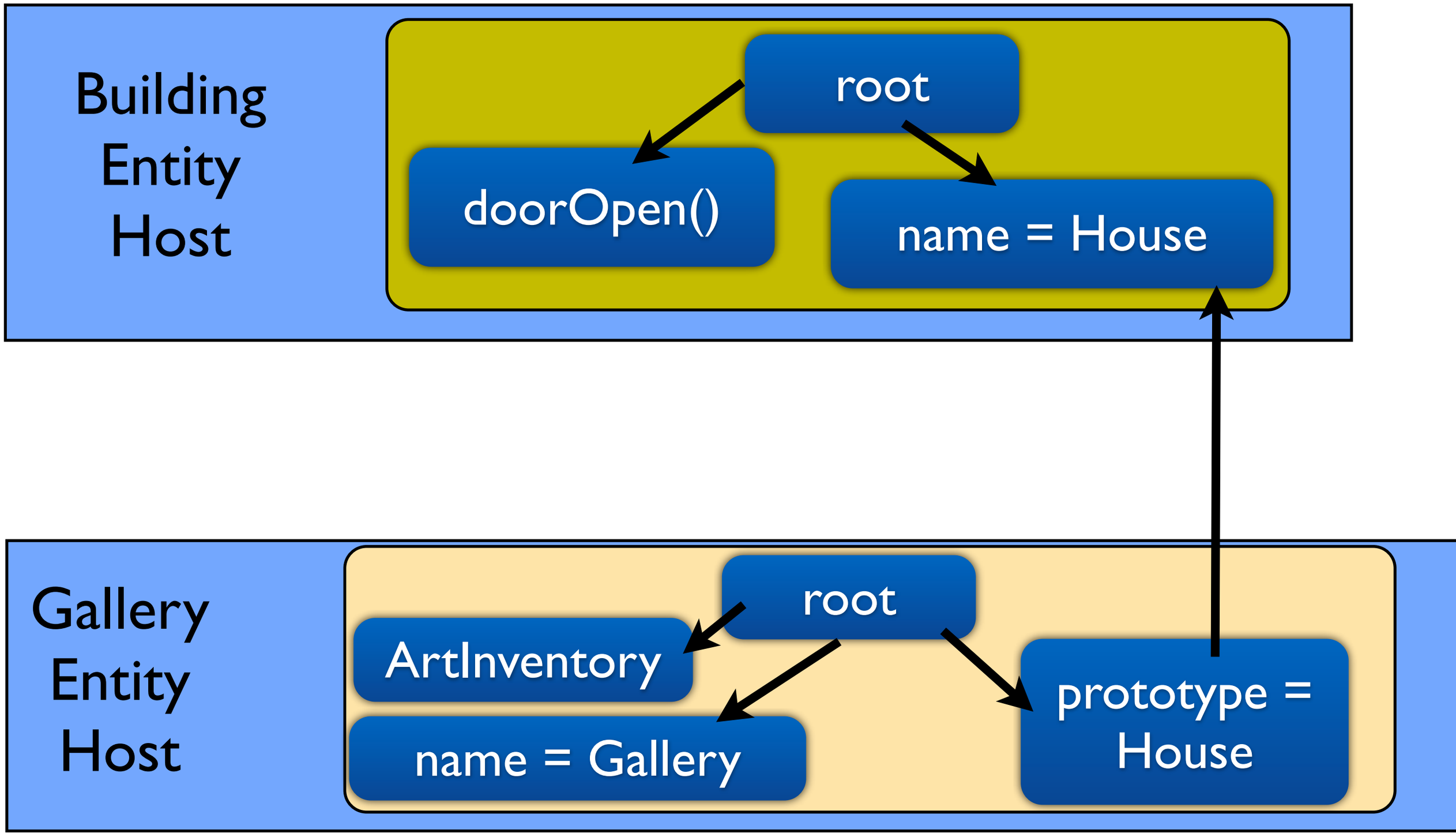
Inheritance: Bad For Entities



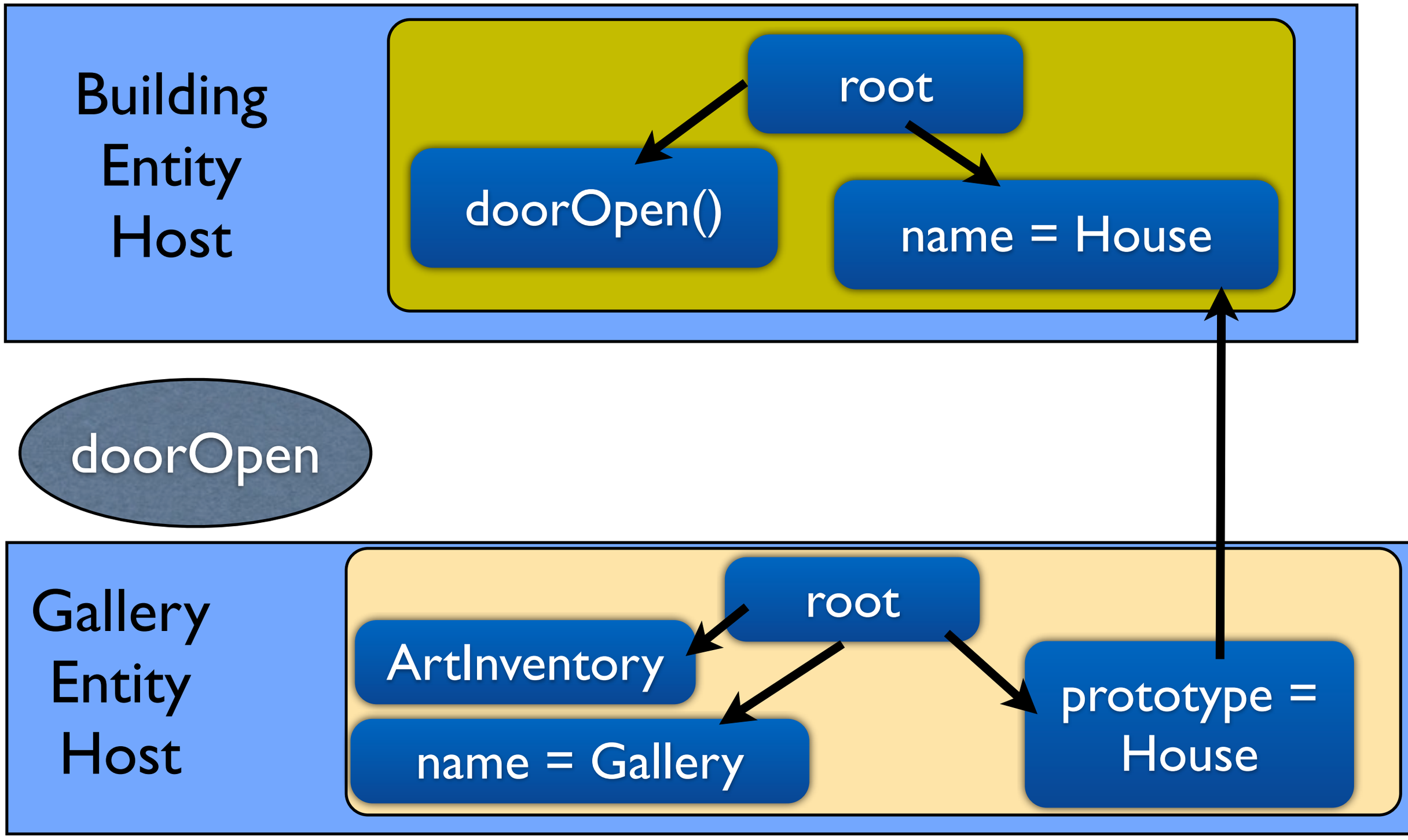
Inheritance: Bad For Entities



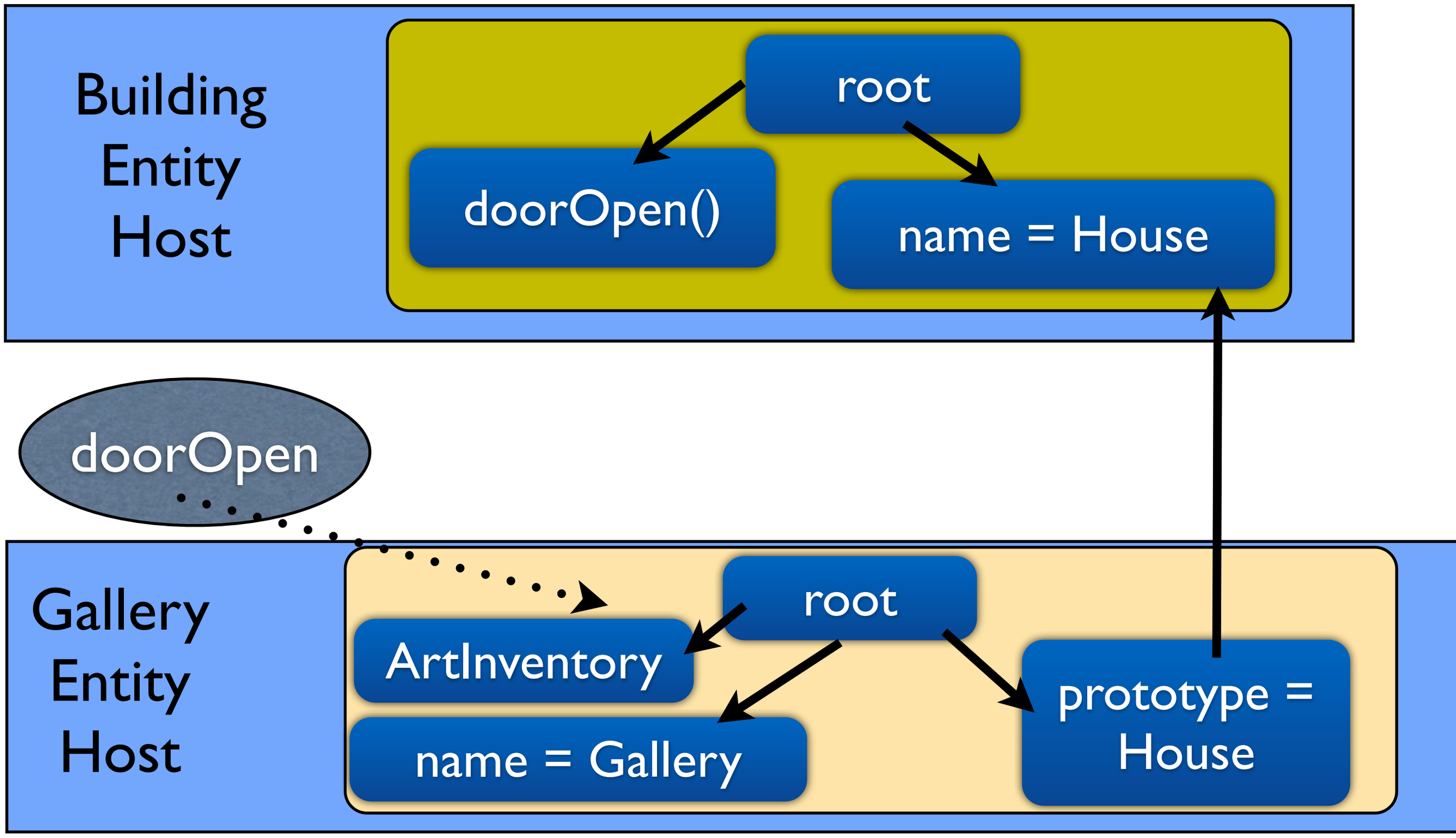
Inheritance: Bad For Entities



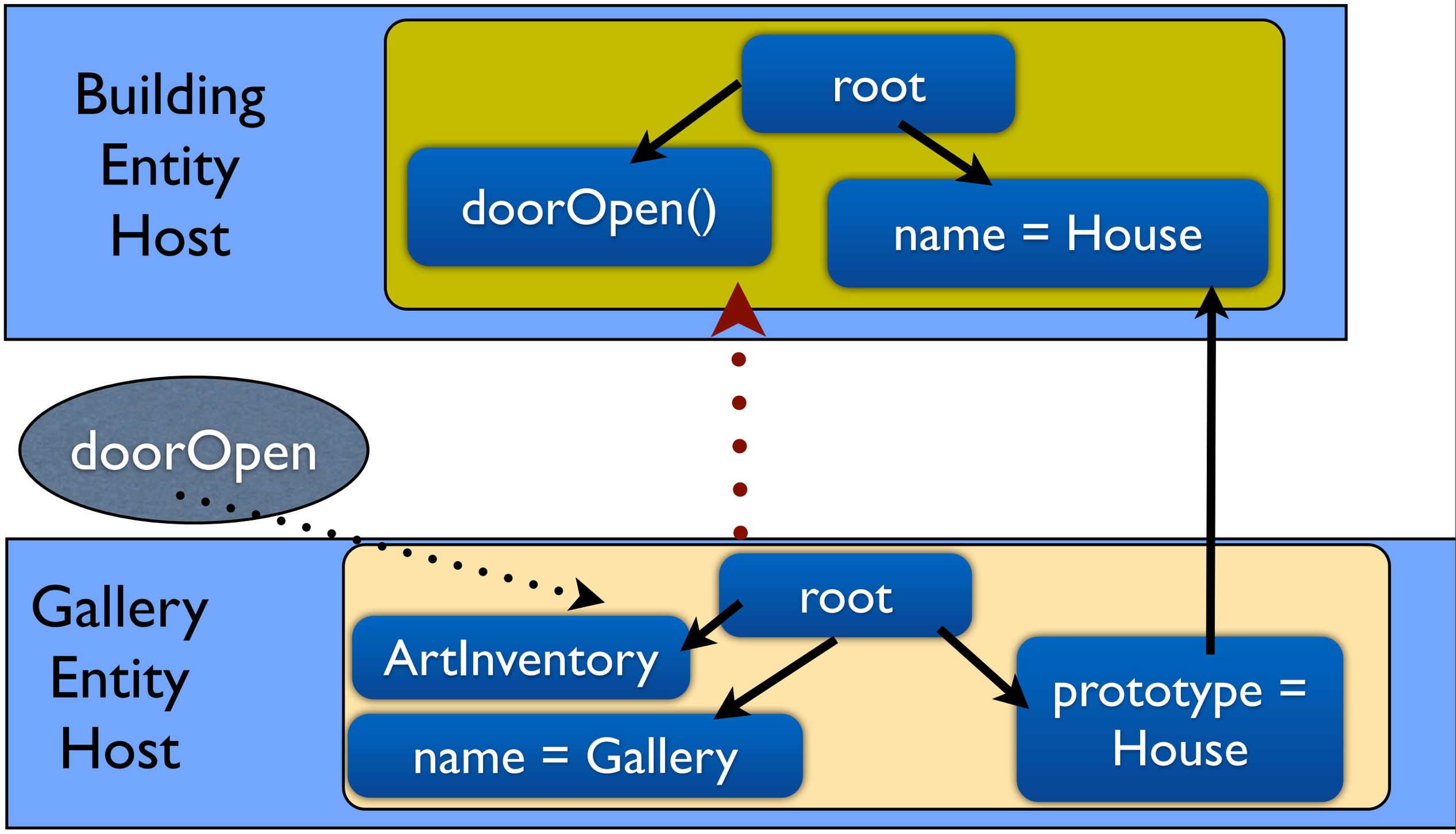
Inheritance: Bad For Entities



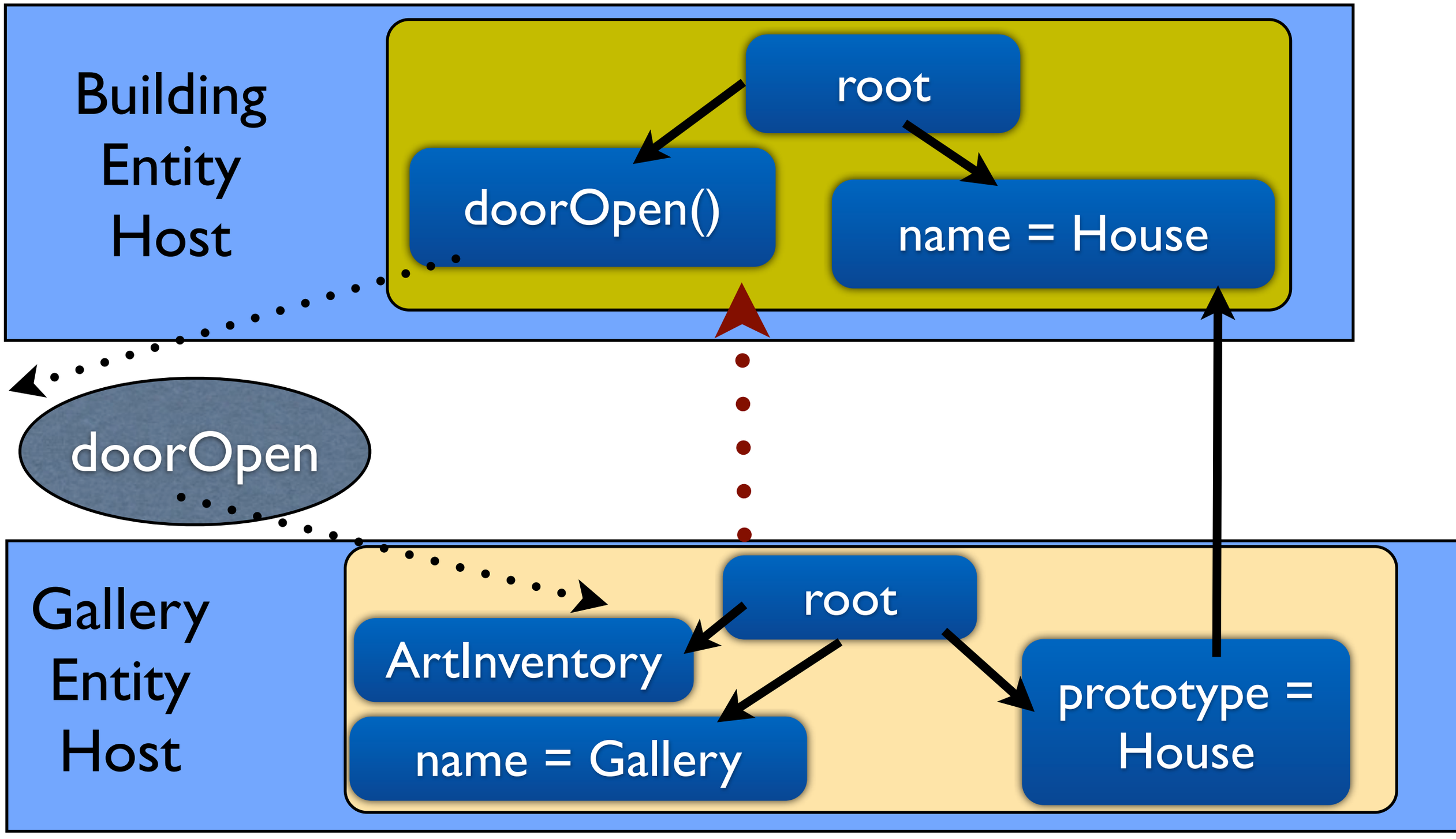
Inheritance: Bad For Entities



Inheritance: Bad For Entities



Inheritance: Bad For Entities



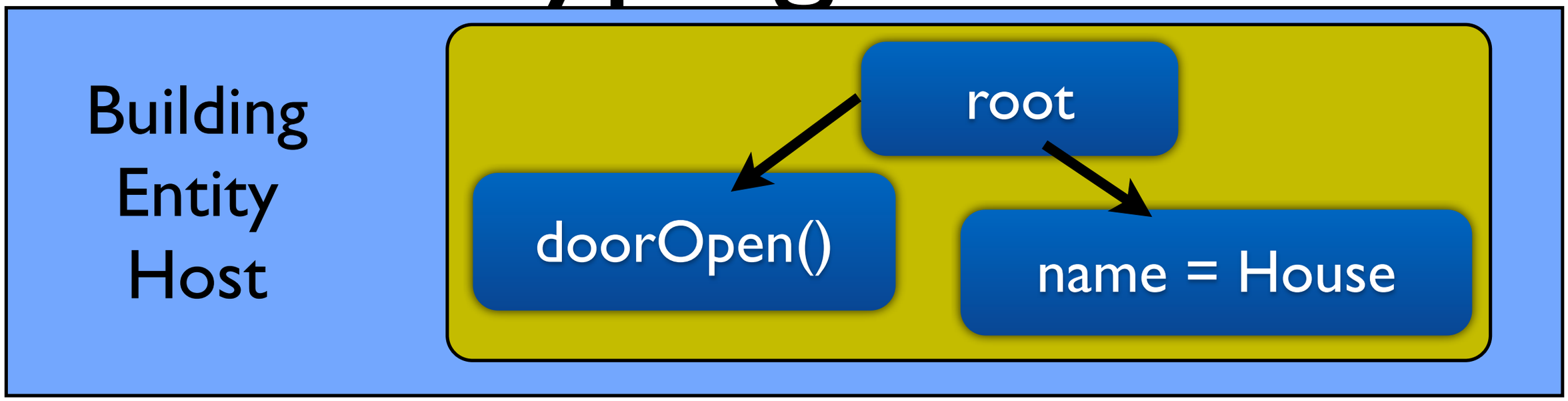
Entity Prototyping

Entity Prototyping

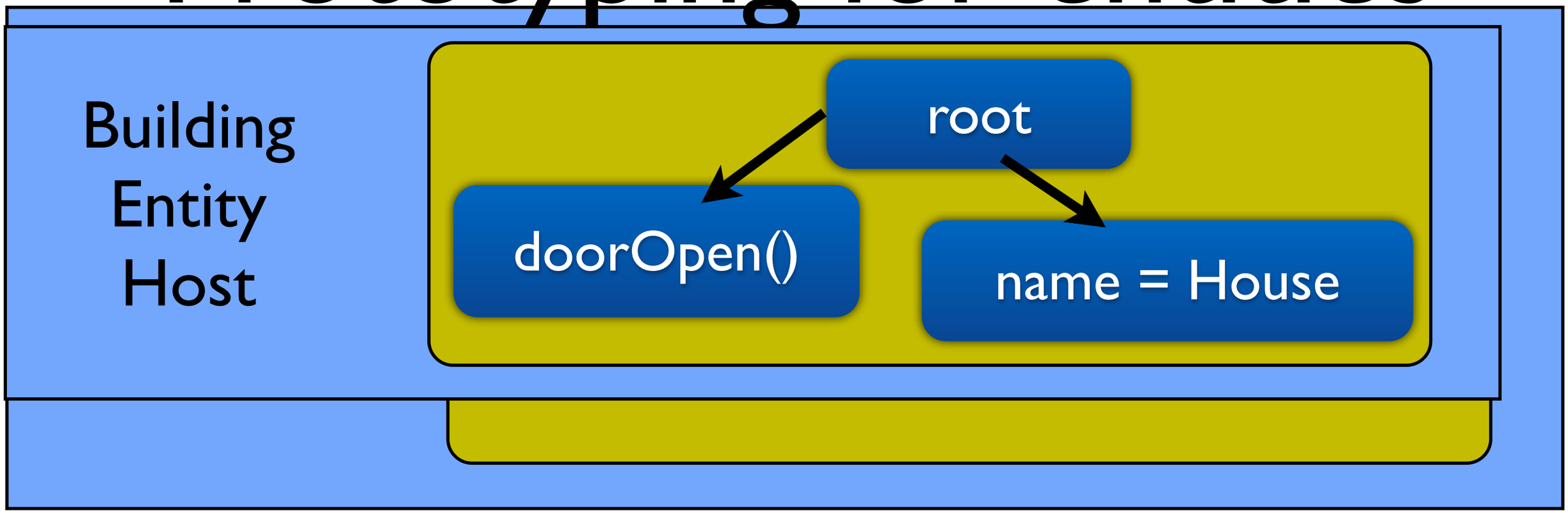
- **Copy Based Prototyping**
  - No prototype lookup
    - Prototype may be on different entity host
    - Look up requires network messaging
  - Copy existing entity and modify
    - State gets copied too

Prototyping for entities

Prototyping for entities

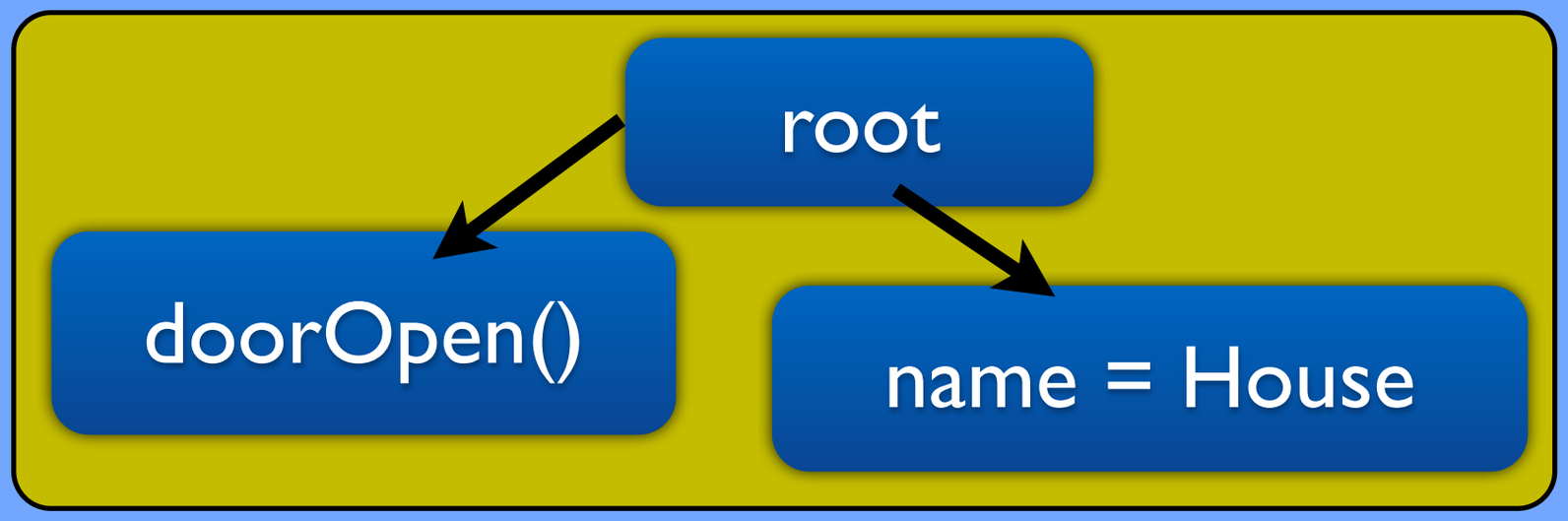


Prototyping for entities

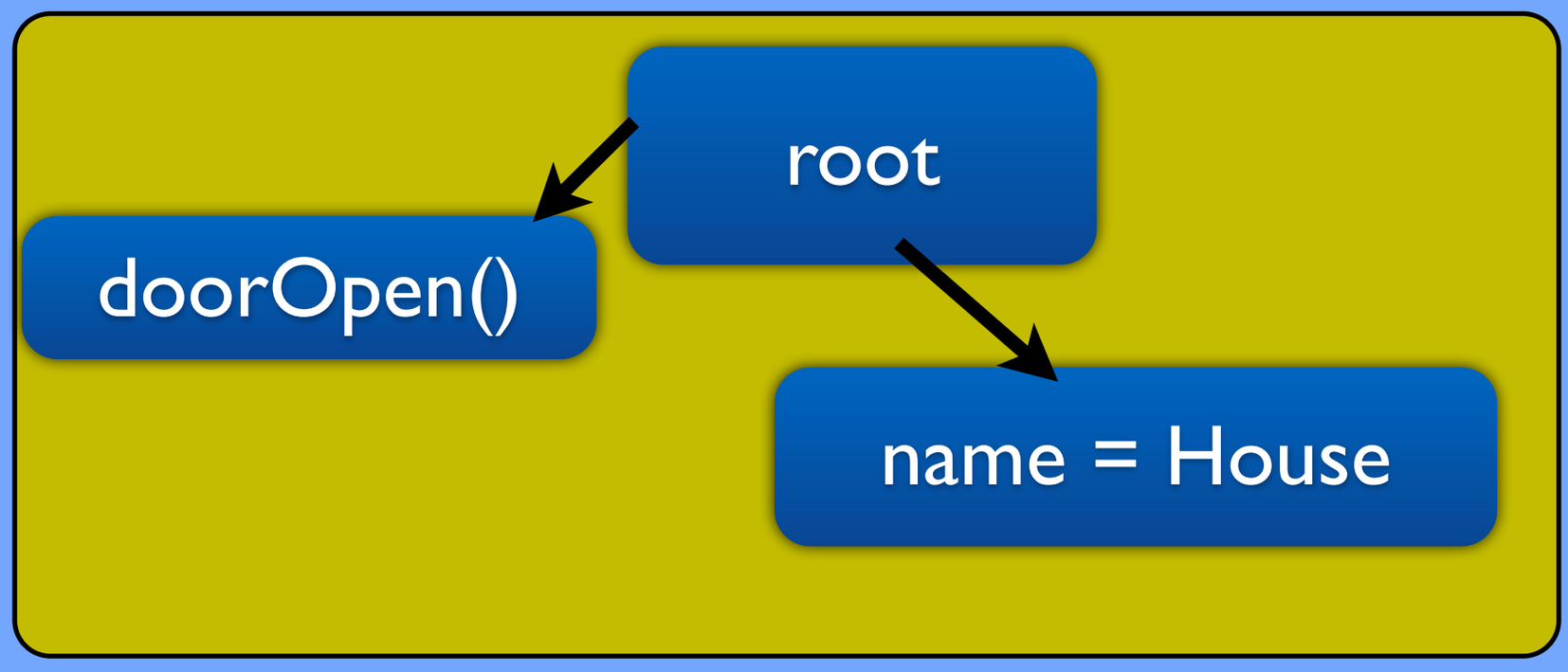


Prototyping for entities

Building Entity Host

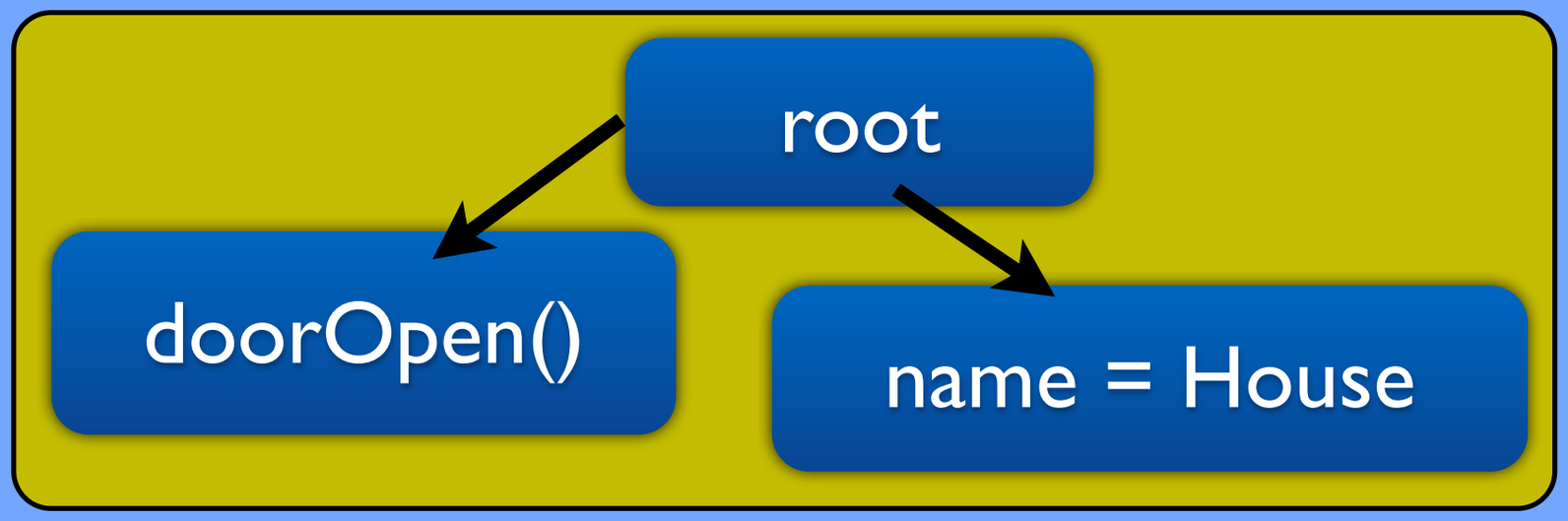


Gallery Entity Host

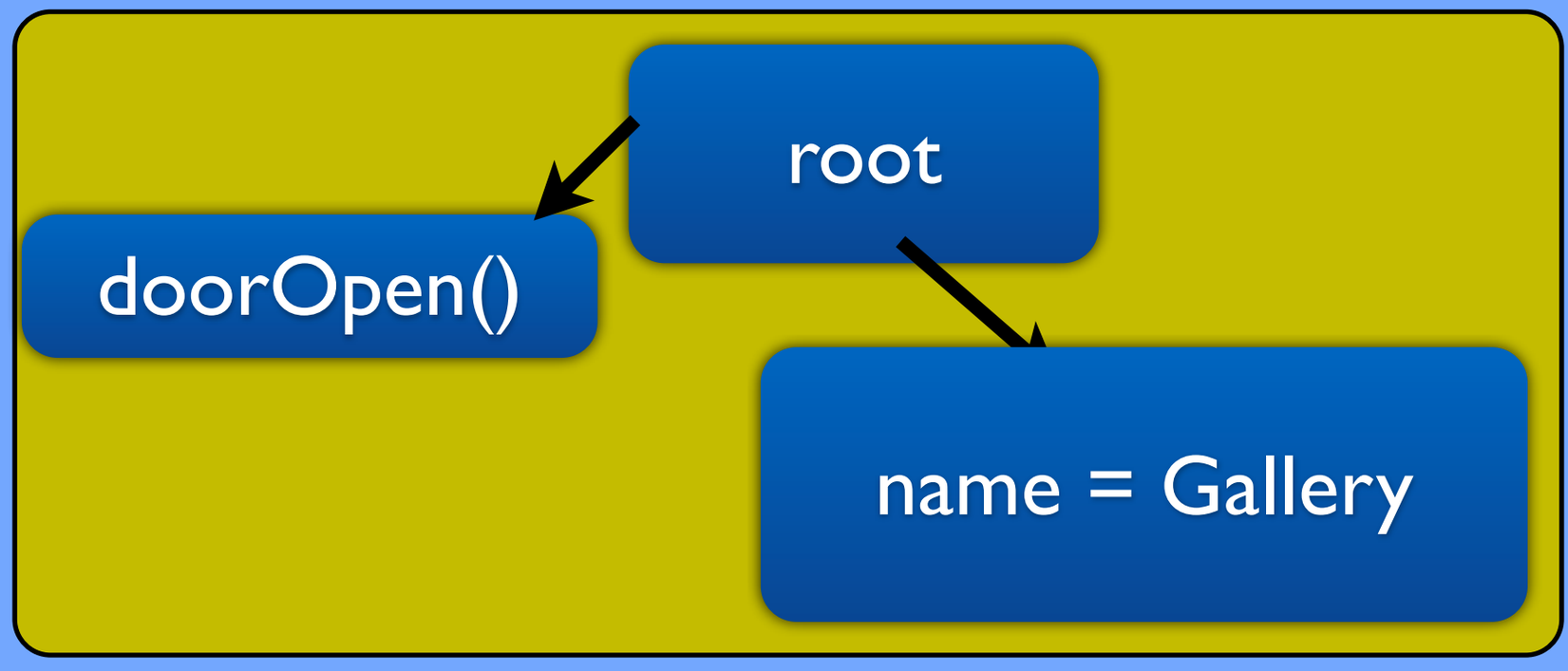


Prototyping for entities

Building Entity Host

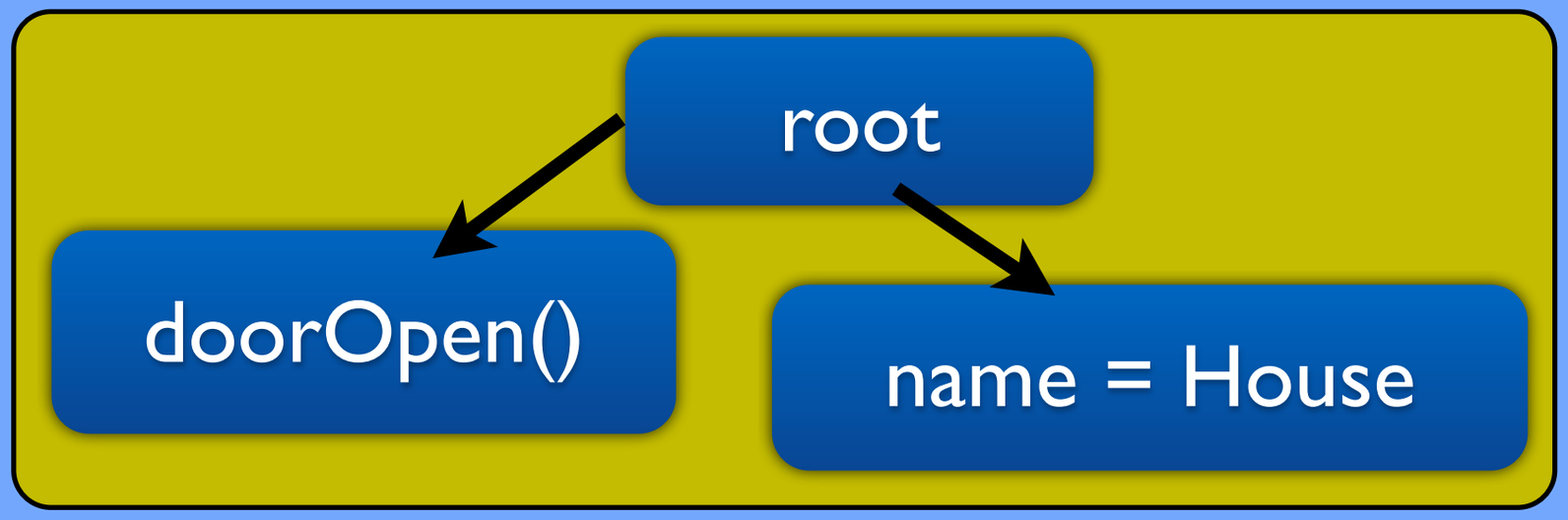


Gallery Entity Host

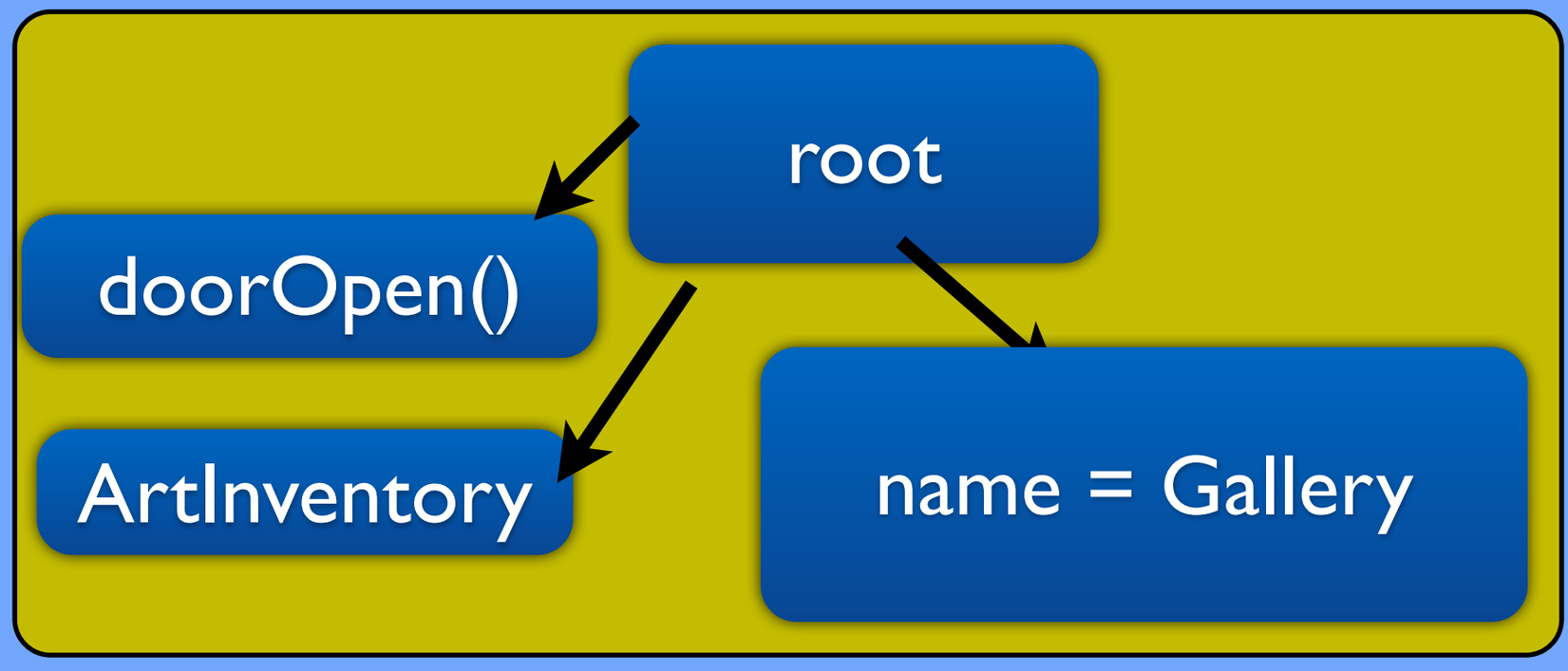


Prototyping for entities

Building Entity Host



Gallery Entity Host



Events

- Multiple event types in virtual worlds
  - Message send/recv, timers, proximity
- Register callbacks for specific events
- Can be painful to handle

Event handling example

- Single Event handler for all message types
- Lots of if-else
- Ugly to fit in incremental model

```
listen_for_messages(message_handler);

fun message_handler(sender_id, msg)
{
    if( msg.name == "loan" )
    {
      loan_art(sender_id, msg);
    }
    else if( msg.name == "buy" )
    {
      buy_art(sender_id, msg);
    }
    else if( msg.name == "take_money" )
    {
      take_money(sender_id, msg);
    }

}
```

Events in Emerson

- Events are described by patterns
- Patterns are objects matched by field
  - name, value and prototype
  - Similar to patterns in Erlang
- [proto] field[.subfield[...]] [: value]
  - `(action:borrow,item_id)`

Messaging Example

Messaging Example

```
handleLoan <- (action:borrow, <-customer
                item_id)
```

Messaging Example

```
handleLoan <- (action:borrow, <-customer
                item_id)

// handleLoan
```

Messaging Example

```
handleLoan <- (action:borrow, <-customer
                item_id)

 // handleLoan
receipt = new Receipt(status = 'OK')
```

Messaging Example

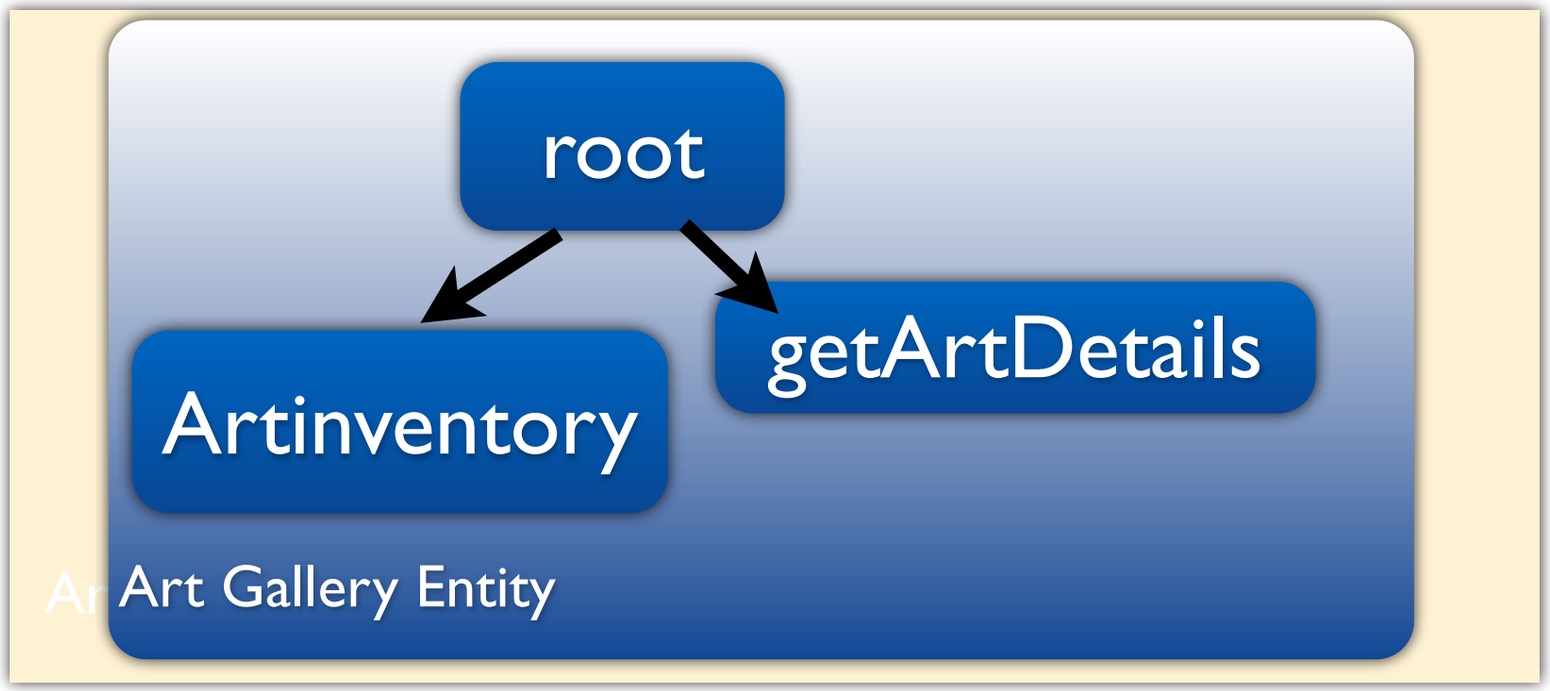
```
handleLoan <- (action:borrow, <-customer
                item_id)

 // handleLoan
receipt = new Receipt(status = ‘OK’)
receipt ->  customer
```

Message Handlers

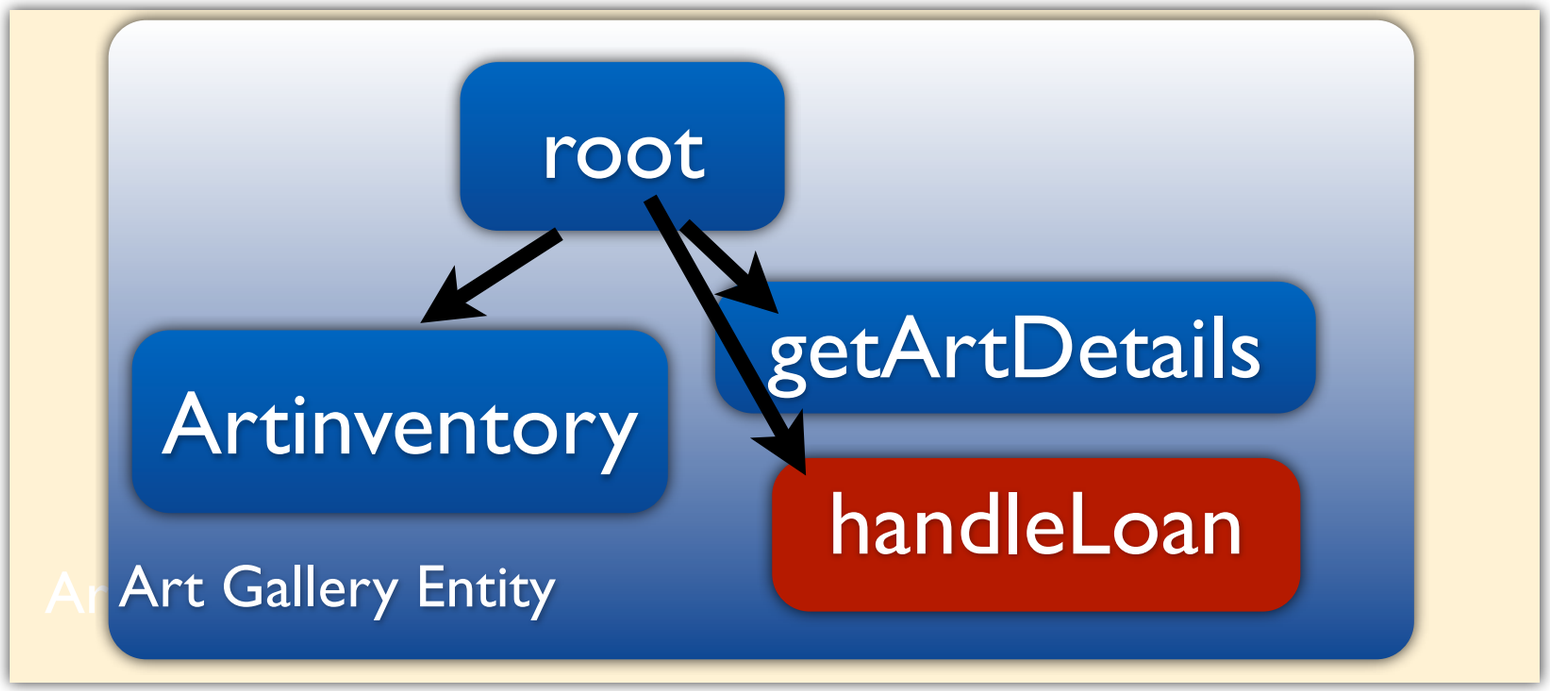
Message Handlers

LONDON



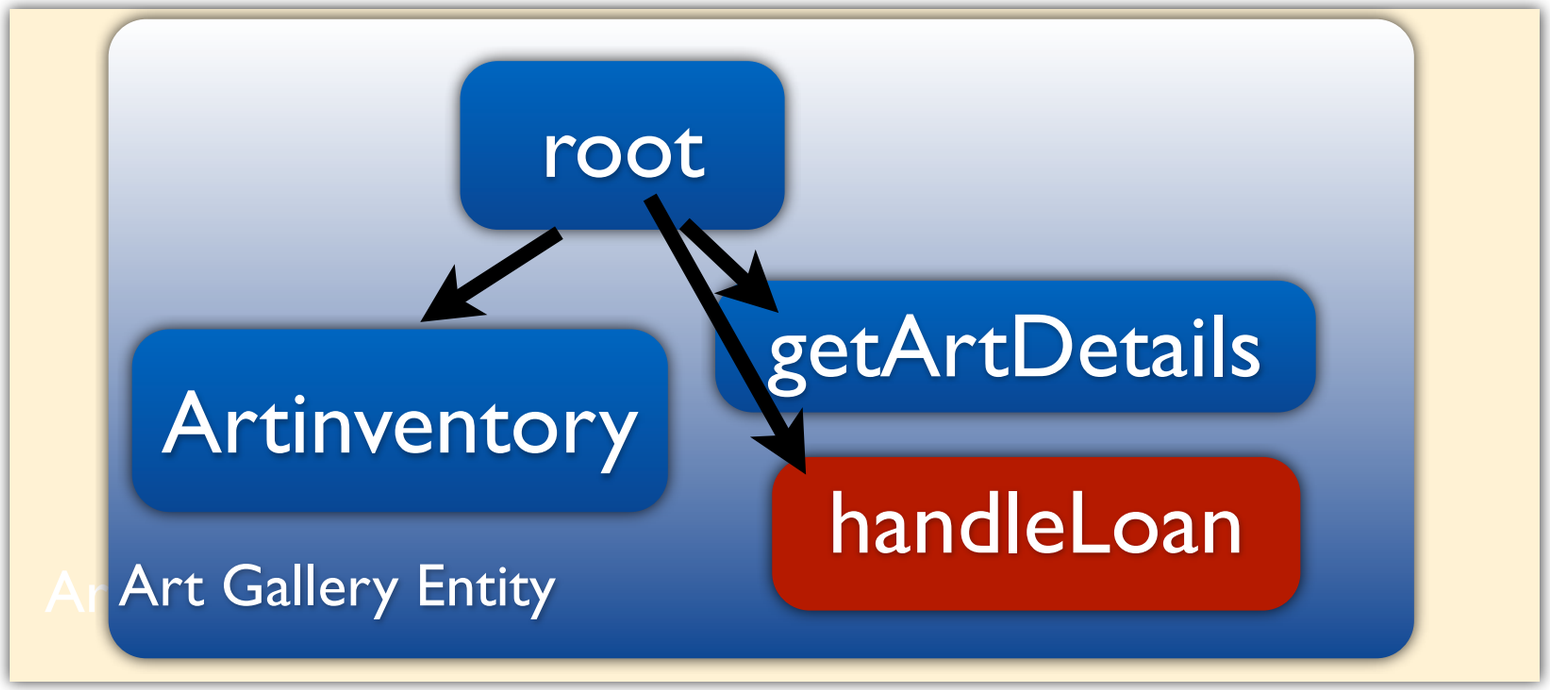
Message Handlers

LONDON



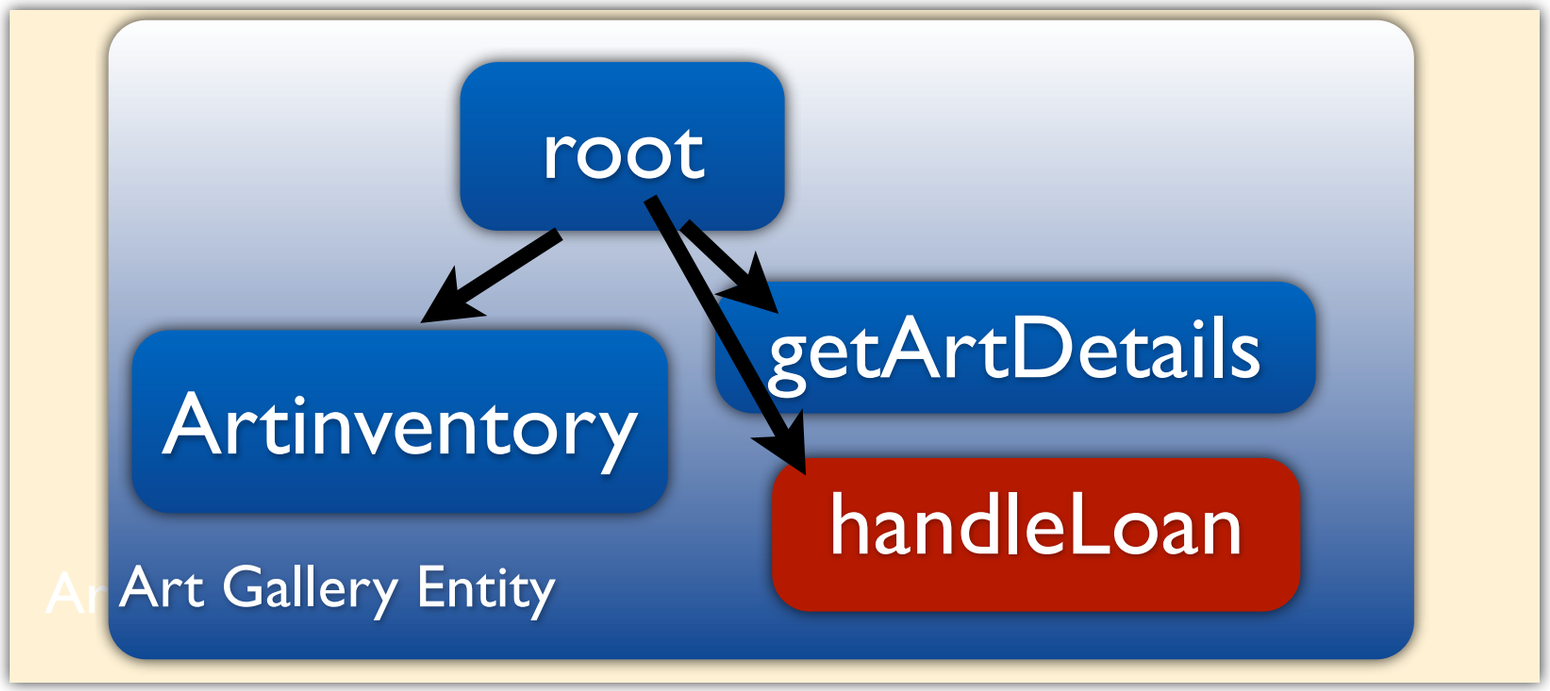
Message Handlers

LONDON



Message Handlers

LONDON



Message Handlers

LONDON

action=borrow
item_id = 65

root

Artinventory

getArtDetails

handleLoan

Art Gallery Entity

Customer Entity

Customer Entity Host

Message Handlers

LONDON

action=borrow
item_id = 65

Customer Entity

Customer Entity Host

root

Artinventory

getArtDetails

handleLoan

Art Gallery Entity

Message Handlers

LONDON

action=borrow
item_id = 65

Customer Entity

Customer Entity Host

root

Artinventory

getArtDetails

handleLoan

Art Gallery Entity

Message Handlers

LONDON

action=borrow
item_id = 65

root

Artinventory

getArtDetails

handleLoan

Art Gallery Entity

Customer Entity

Customer Entity Host

Summary

Summary

- Emerson: Scripting for Federated, Seamless and Scalable VW
  - Federation: Entity Isolation
  - Scalability: Asynchronous Messaging

Summary

- Emerson: Scripting for Federated, Seamless and Scalable VW
  - Federation: Entity Isolation
    - Scalability: Asynchronous Messaging
- Easy scripting
  - Reuse prototypes, Incremental Scripting

Summary

- Emerson: Scripting for Federated, Seamless and Scalable VW
  - Federation: Entity Isolation
    - Scalability: Asynchronous Messaging
- Easy scripting
  - Reuse prototypes, Incremental Scripting
- Event Handling: Pattern based

Current State

- Prototype based on V8 JavaScript interpreter
- Sirikata virtual world (www.sirikata.com)
- Language Library

Future

- Need to extend Emerson for
  - Persistence features
  - Transactions
  - Access control and ownership issues

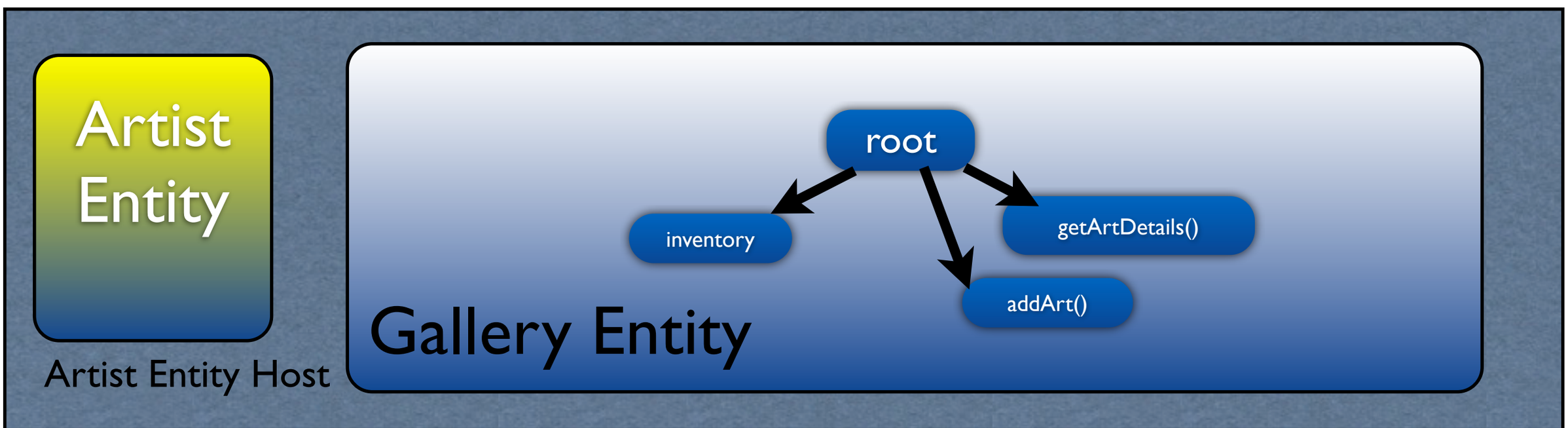
Thank You

Live Programming

- Entities can dynamically execute scripts
  - `eval` as in JavaScript
- Modification of state and behavior without termination
- Access control to prevent executing arbitrary scripts

Live modification

LONDON



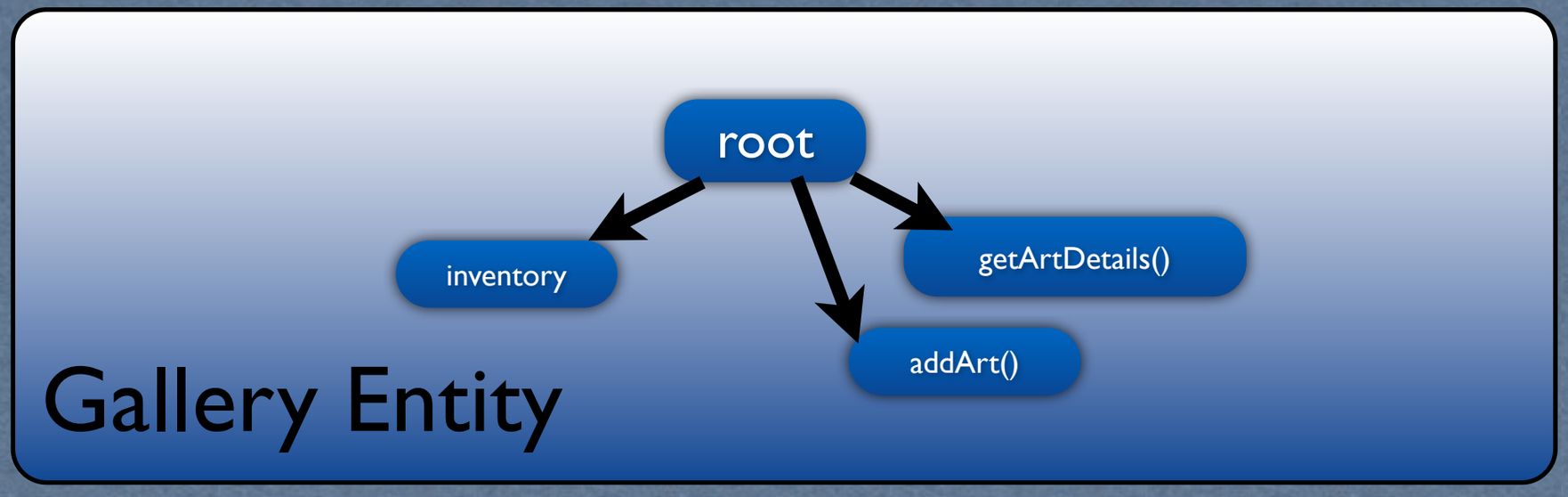
Live modification

function deleteArt()

LONDON

Artist Entity

Artist Entity Host



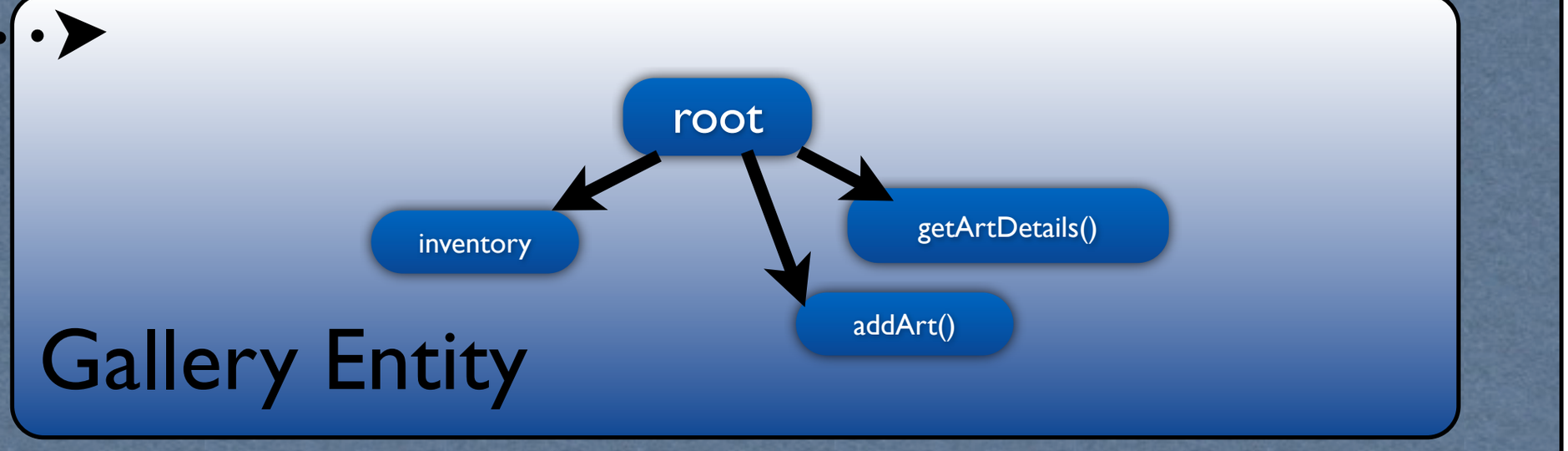
Live modification

function deleteArt()

LONDON

Artist Entity

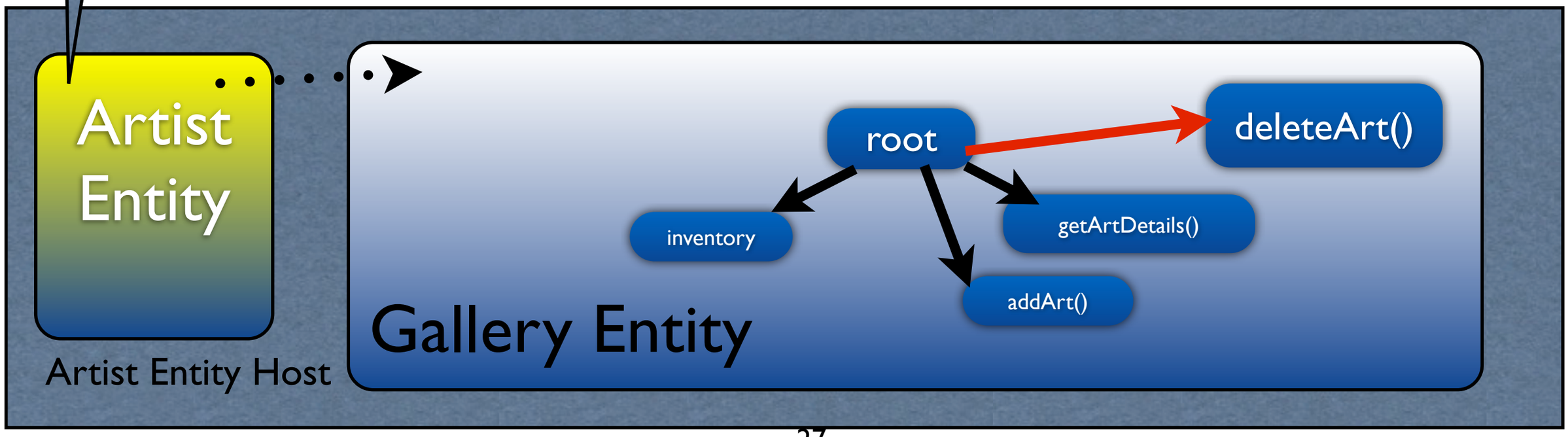
Artist Entity Host



Live modification

function deleteArt()

LONDON



Still evolving...

- developing more of the syntactic features
- Writing programs to find common cases and embed these in the language as syntax
- Exposing more of the underlying system functionality into the language
- Language Library

Presences

Presences

- Entities hold references to presences of their own and other entities

Presences

- Entities hold references to presences of their own and other entities
- Communication through presences

Presences

- Entities hold references to presences of their own and other entities
- Communication through presences
- Multiple presences to bridge worlds
  - same entity can service multiple worlds

Emerson

Emerson

- Interpreted language
  - similar to JavaScript

Emerson

- Interpreted language
  - similar to JavaScript
- Event-driven execution model
  - Each entity executes single script
  - Script consists of short event handlers

Live Programming

- Entities can dynamically execute scripts
- More in the paper

Scripting in VW

Scripting in VW

- Add behavior to graphical entities
  - Entities execute a program (Scripted Entities)
  - Eg. Lua (WoW), LSL (Second Life), UScript (Unreal)
  - Bulletin Boards, Intelligent Fighters

Summary

- Entities, Presences, Objects
- Code Reuse
  - prototypes for objects
  - copy and modify for entities
- Incremental development by executing arbitrary scripts
- Patterns for events with failure callbacks

Art Gallery

Art Gallery

Art Gallery

VirtualWorld

LONDON

Entity Host

Artist Entity Host

Art Gallery

VirtualWorld

LONDON

Entity

Artist
Entity

Artist Entity Host

Entity Host

Art Gallery

VirtualWorld

LONDON

Entity

Artist
Entity

Artist Entity Host

Entity Host

Art Gallery

VirtualWorld

LONDON

Artist
Avatar
(Presence)

Entity

Artist
Entity

Entity Host

Artist Entity Host

Art Gallery

VirtualWorld

LONDON

Artist
Avatar
(Presence)

Entity

Entity

Entity Host

Artist
Entity

Artist Entity Host

Art Gallery Entity

Art Gallery

VirtualWorld

LONDON

Artist
Avatar
(Presence)

Entity

Entity

Entity Host

Artist
Entity

Artist Entity Host

Art Gallery Entity

Art Gallery

VirtualWorld

LONDON

Artist Avatar (Presence)

Art Gallery (Presence)

Entity

Entity

Entity Host

Artist Entity

Artist Entity Host

Art Gallery Entity

Art Gallery

VirtualWorld

LONDON

Artist Avatar (Presence)

Art Gallery (Presence)

Entity

Entity

Entity Host

Artist Entity

Artist Entity Host

root

getArtDetails()

inventory

Art Gallery Entity

Object

Art Gallery

VirtualWorld

LONDON

Artist Avatar (Presence)

Art Gallery (Presence)

Entity

Artist Entity

Artist Entity Host

Entity

root

inventory

getArtDetails()

handleLoan()

Art Gallery Entity

Object

handler

Entity Host

action:borrow
item_id