BigWorld Technology Training

Server

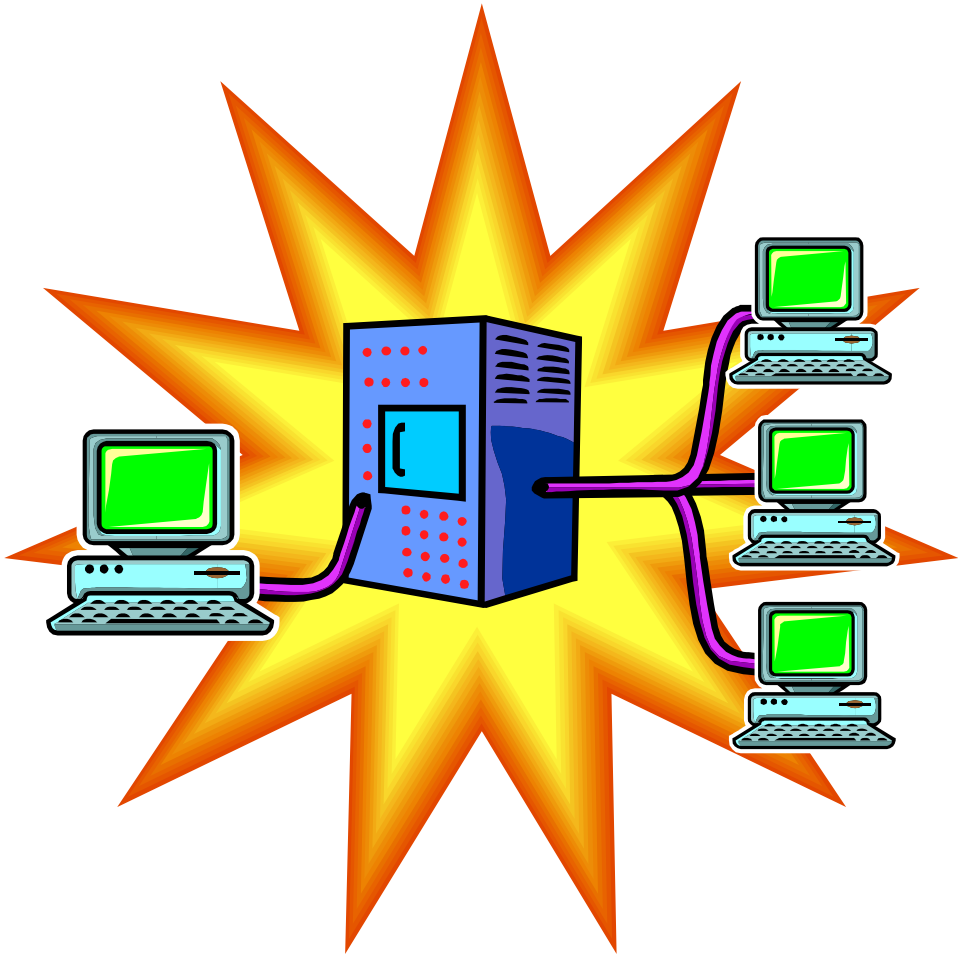
BigWorld Technology
The Complete MMOG Solution

Outline

- BigWorld Server Overview
- Implementing an Entity
- Entity Communication
- Core Entity Components
- Cell Functionality
- Server Setup and Maintenance
- Server Profiling and Stress Testing

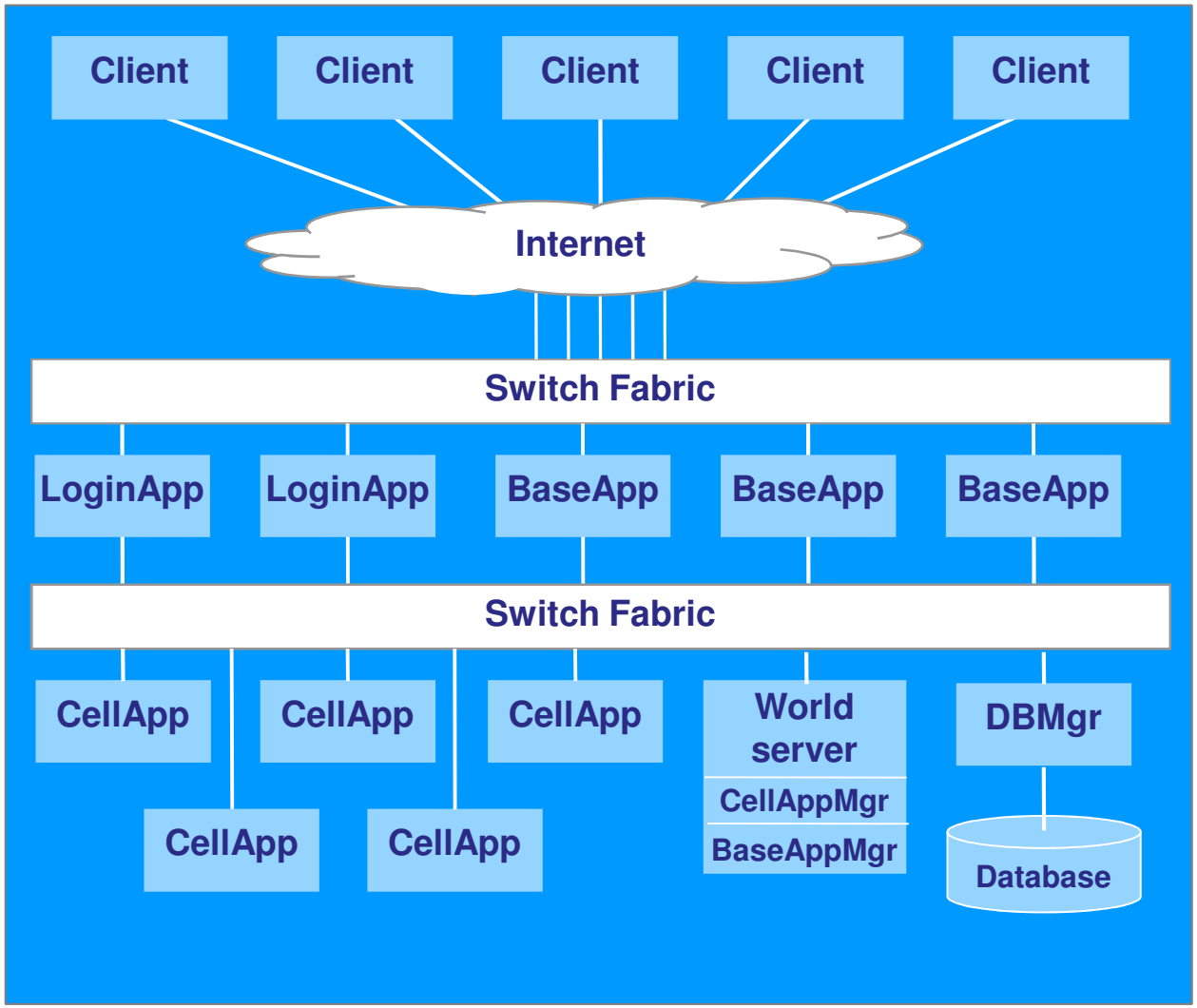
bigWORLD™ TECHNOLOGY

Session 1
BigWorld Server Overview



bigWORLD™ TECHNOLOGY

BigWorld Server



bigWORLD TECHNOLOGY

LoginApp

- First connection point for clients
- Fixed port
- Initial communication encrypted
  - Public key pair (arbitrary key size)
  - Username / password security
- Multiple LoginApps for load balancing
  - DNS round robin

bigWORLD™ TECHNOLOGY

BaseApp

- Fixed communication point for clients
- Proxies communication to CellApps
- Load balancing mechanism for client connections
- Used for processing non-spatial entities
  - Auction House
  - Guild Managers
  - Instance Managers
- Fault tolerance for other BaseApps
- One BaseApp process per CPU / core

bigWORLD™ TECHNOLOGY

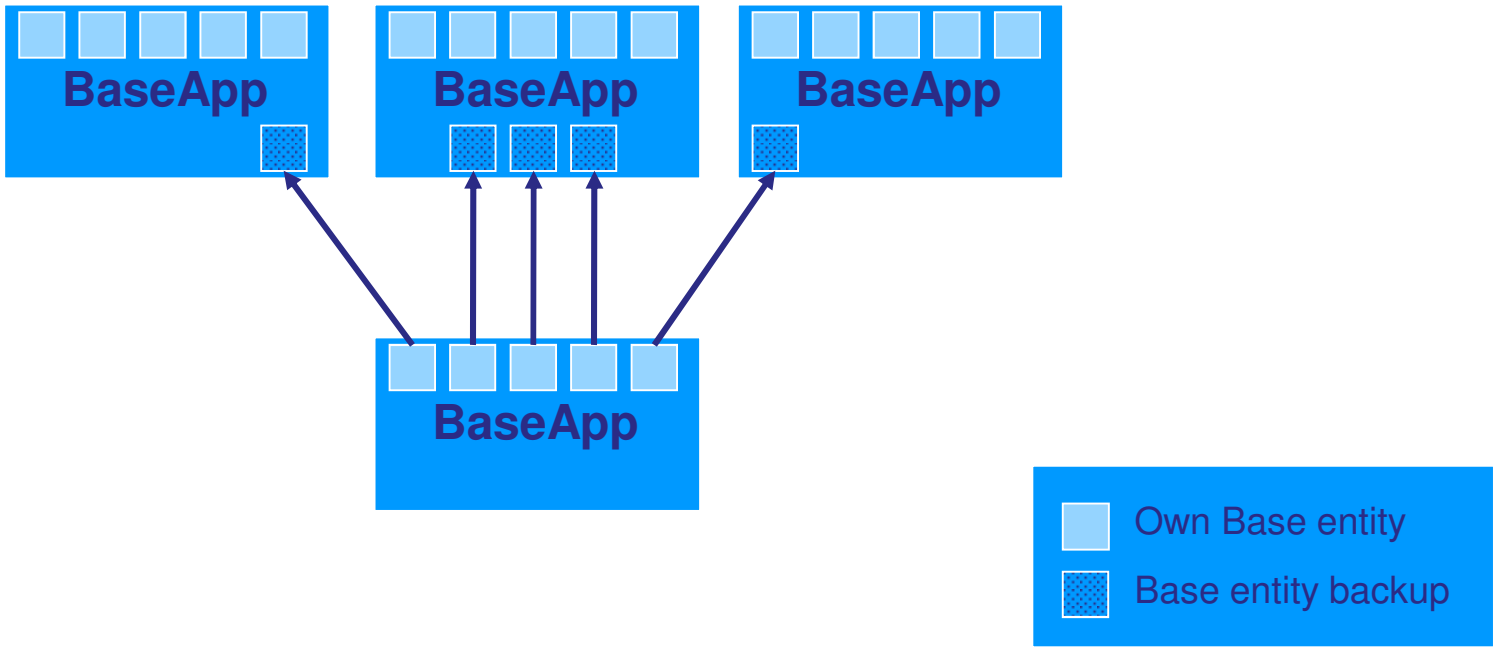
Base Entities

- Two types of Base entities
  - Base
  - Proxy
- Base
  - Regular game entity
  - Eg: persistent NPC, Auction house…
- Proxy
  - Client connection
  - Specialisation of Base

bigWORLD™ TECHNOLOGY

BaseApp Fault Tolerance

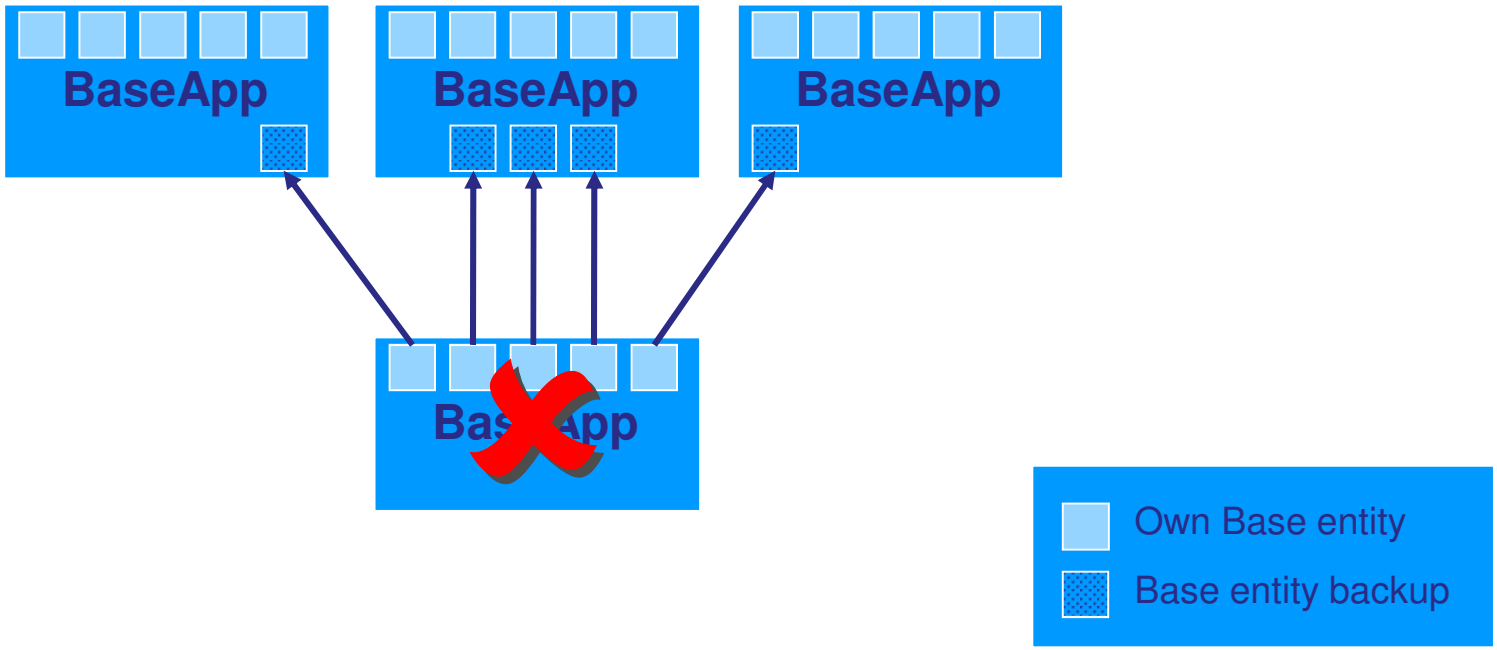
- Backup entities onto other BaseApps



bigWORLD™ TECHNOLOGY

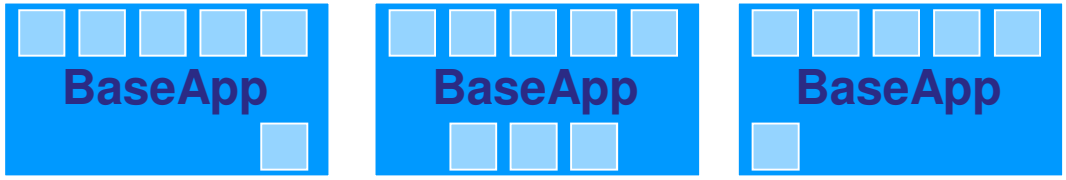
BaseApp Fault Tolerance

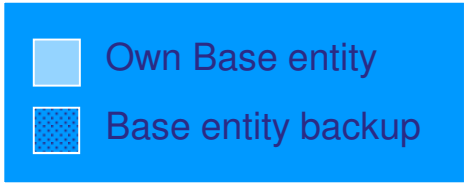
- BaseApp becomes unavailable



BaseApp Fault Tolerance

- Entities of the dead BaseApp are resurrected on their backups





bigWORLD™ TECHNOLOGY

BaseApp Fault Tolerance

- Clients connected to an unavailable BaseApp will be disconnected
  - All data is saved
  - Upon reconnection they are handed to their old entity (providing it hasn't timed out)

BaseApp Manager (BaseAppMgr)

- Manages load balancing for BaseApps
- Monitors BaseApps for fault tolerance
- Primarily used during login and new entity creation
- 1 instance per server
- Fault tolerance with Reviver
- Shuts the server down if 2 BaseApps fail
  - Configurable, but unsafe

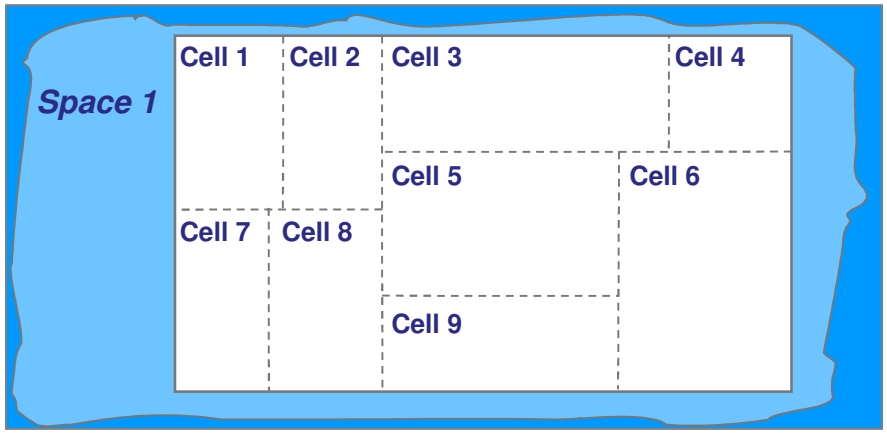
bigWORLD™ TECHNOLOGY

CellApp

- Spatial process
  - Deals with the game space players interact in
- Processes entities that exist in spaces
- Processes a region of a space (Cell)
  - Only 1 cell per space
- Potentially manages many spaces
- One CellApp process per CPU / core

Cells and Spaces

- Spaces are load balanced via cells
- A space must contain at least 1 cell
- Each cell processes an area of a space
- Cell boundaries shift depending on load
- Cells don’t affect client game play



bigWORLD™ TECHNOLOGY

CellApp Load

- Total number of entities being managed
- Frequency of entity communication
  - Explicit: method calls
  - Implicit: property updates
  - Entity density
- Entity script
- Entity data size

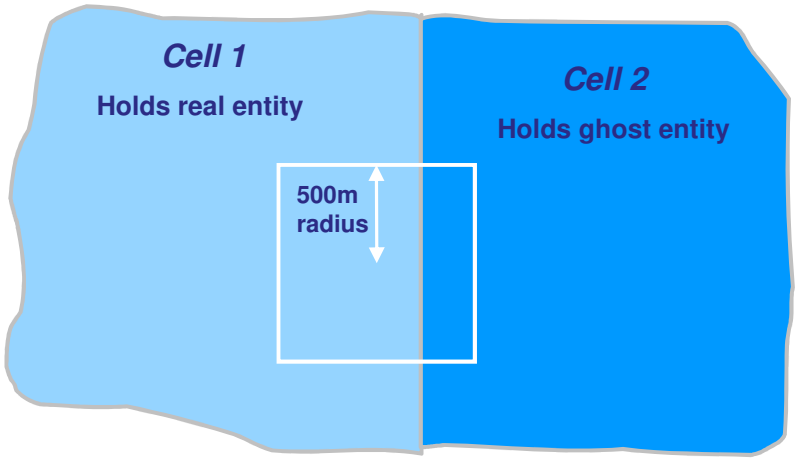
bigWORLD™ TECHNOLOGY

Entities and Cells

- Each space must contain at least 1 entity
  - Initial space is an exception
- CellApp client entity has a Witness object
  - Witness tracks surrounding entities
- Entity Area of Interest defaults to 500m
  - Can be modified, a lot of dependencies

Entities and Cells

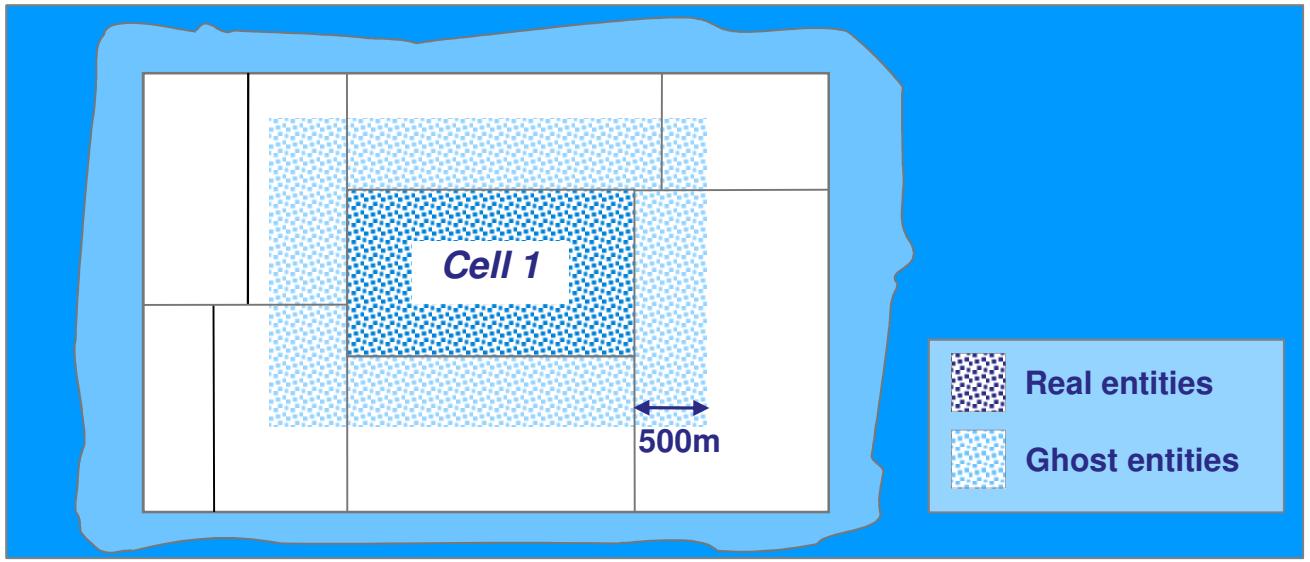
- Entities seamlessly cross cell borders
  - Client has no knowledge of cells
- Cells maintain a list of entities outside their borders
  - Ghost entities
  - 500m (same as AoI)



bigWORLD™ TECHNOLOGY

Entities: Reals and Ghosts

- A *real* entity is authoritative
- A *ghost* is a partial copy from a nearby cell



bigWORLD™ TECHNOLOGY

Ghost entities

- Solve entity interaction across cell boundaries
- Method calls
  - Forwarded to the *real* entity
- Properties
  - Can be made *real* only
    - Ie: Will never exist on the Ghost
  - Must be ghosted if visible to the client
    - Current weapon
    - Armour type
    - Name
  - Are read-only
    - Use method calls to update the *real*

bigWORLD™ TECHNOLOGY

Entity Updates

- Clients implement Level of Detail to speed up rendering
- CellApps implement LoD to reduce:
  - bandwidth consumption
  - Per-entity CPU utilisation
- LoD on CellApps work as with a Client
  - Detail is relative to the active entity
- Client entity methods can implement LoD
- Entity properties implement LoD to avoid unnecessary communication to the Client
  - Active weapon (not visible from long range)

CellApp Manager (CellAppMgr)

- Has knowledge of:
  - All CellApps (and their load)
  - All cell boundaries
  - Spaces
- Manages CellApp load balancing
  - Tells CellApps where their cell boundaries should be
- Adds new entities to the correct cell
- 1 instance per server
- Fault tolerance with Reviver
  - Server can continue operating (no load balancing)

bigWORLD™ TECHNOLOGY

Database Manager (DBMgr)

- Persistent entity storage manager
- Communicates entity information to and from DB to the rest of the server
- DB types supported:
  - XML (rapid prototyping)
  - MySQL (production)
  - … your own (full source to DBMgr)
- Hooks into your billing system
- Separate machine

bigWORLD™ TECHNOLOGY

Entity Backups

- Archiving
  - Round robin procedure across BaseApps
  - BaseApps request data from Cell
  - Pass back to DBMgr

Reviver

- Respawns unavailable processes
- Not required but useful for production
- Dormant process
- Receives notification of process death
- Restarts process then dies
  - Can be configured to stay active
- Primarily used to revive Managers

bigWORLD™ TECHNOLOGY

BigWorld Machine Daemon (bwmachined)

- Daemon for managing server processes
- Run on every server machine
- Starts / stops processes
- Informs cluster of server process health
  - Eg: Notifies Revivers on process death
- Monitors machine utilisation
  - CPU / Memory / Bandwidth

bigWORLD™ TECHNOLOGY

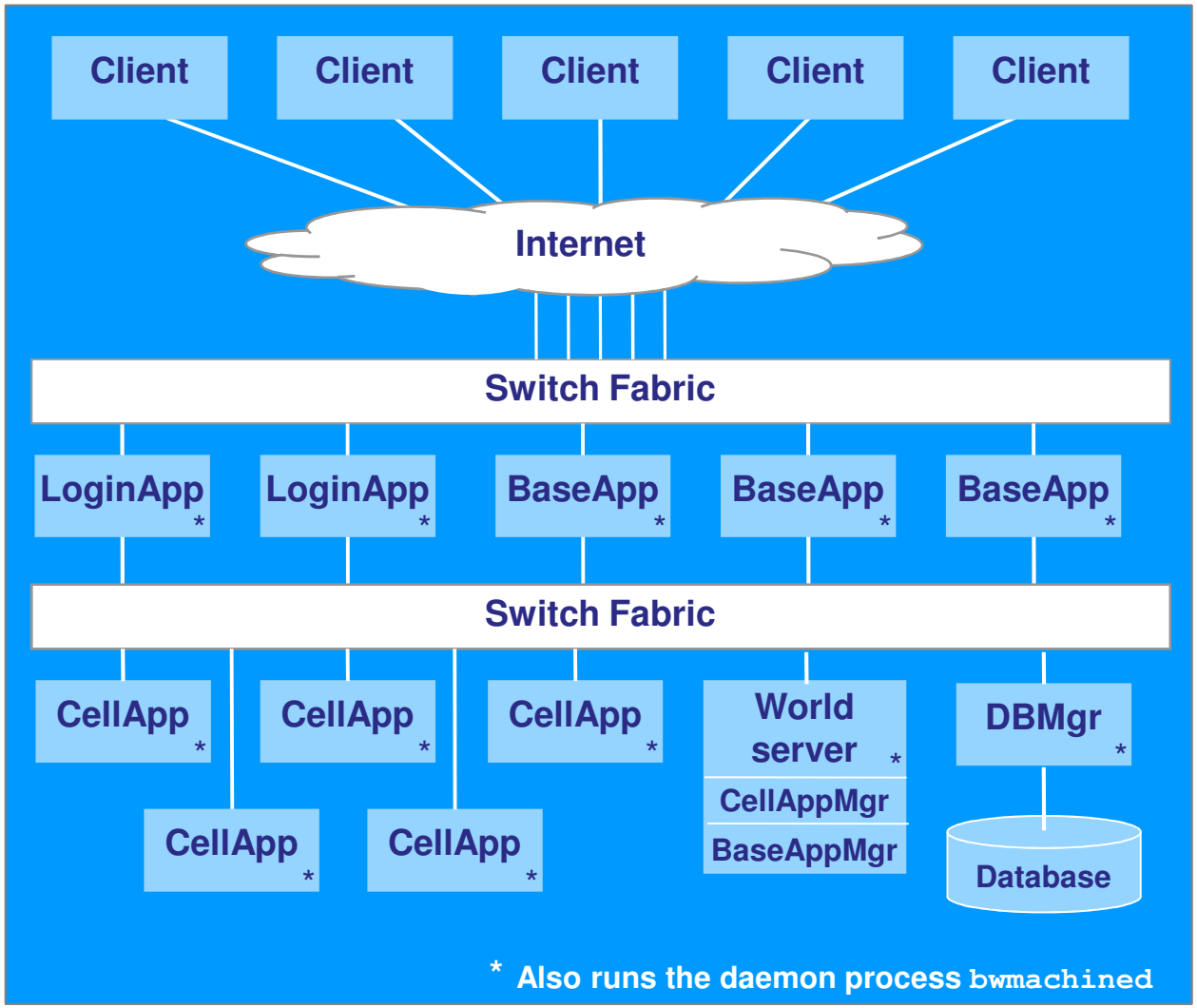
Process Communication

- Mercury
  - RPC protocol over UDP
  - Reliable communication

- Some terms you might hear:
  - Bundle
    - Collection of messages to be sent
  - Channel
    - Ongoing communication stream between 2 components
    - Eg: Client / Proxy channel

bigWORLD™ TECHNOLOGY

BigWorld Server



bigWORLD™ TECHNOLOGY

General Operation

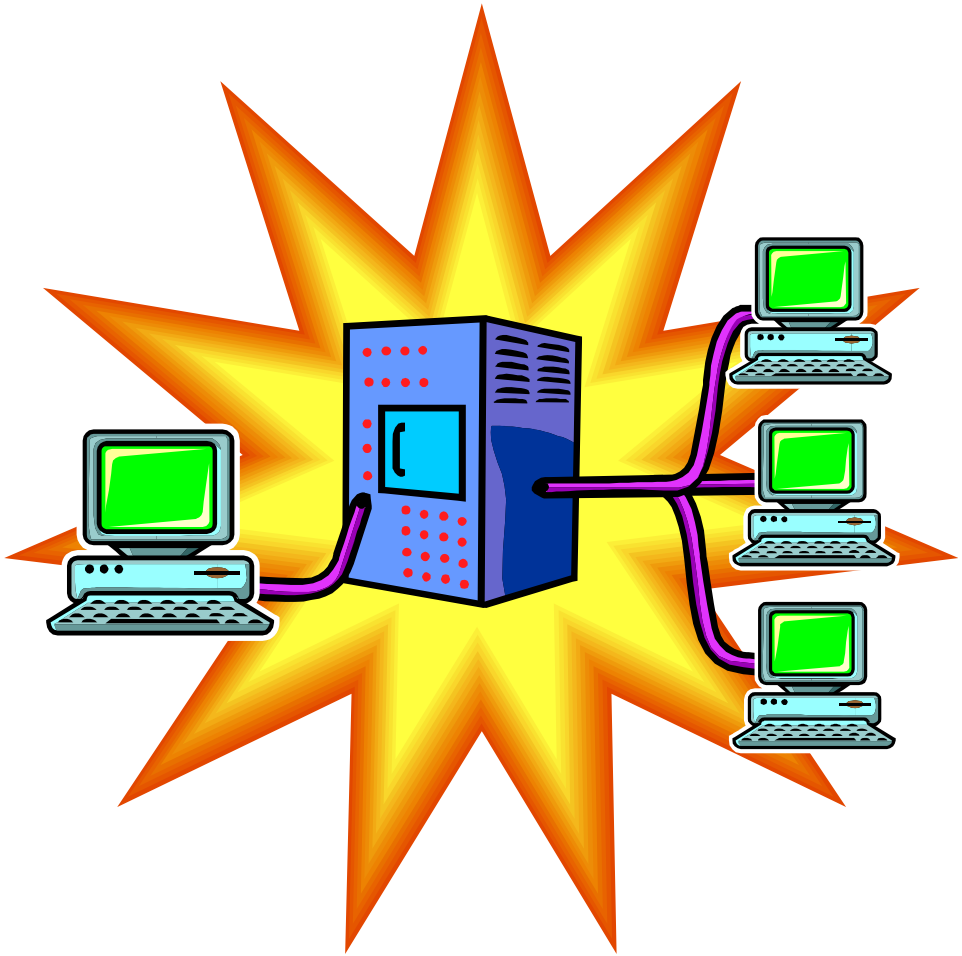
- 2 CellApps per BaseApp
  - Rule of thumb, differs for every game
  - Profile early / consistently
- Separate machine for:
  - DBMgr
  - Server Tools

Login Procedure

- Client sends login request
  - Known hostname / port
- LoginApp receives login request
  - Decrypts request
- LoginApp forwards request to DBMgr
- DBMgr validates username / password
  - Queries the DB
- Valid requests forwarded to BaseAppMgr
- BaseAppMgr forwards player entity creation to least loaded BaseApp
- BaseApp creates a new Proxy
  - This may in turn create a new Cell entity
- UDP port of Proxy is returned to the Client
  - via BaseAppMgr, DBMgr, LoginApp

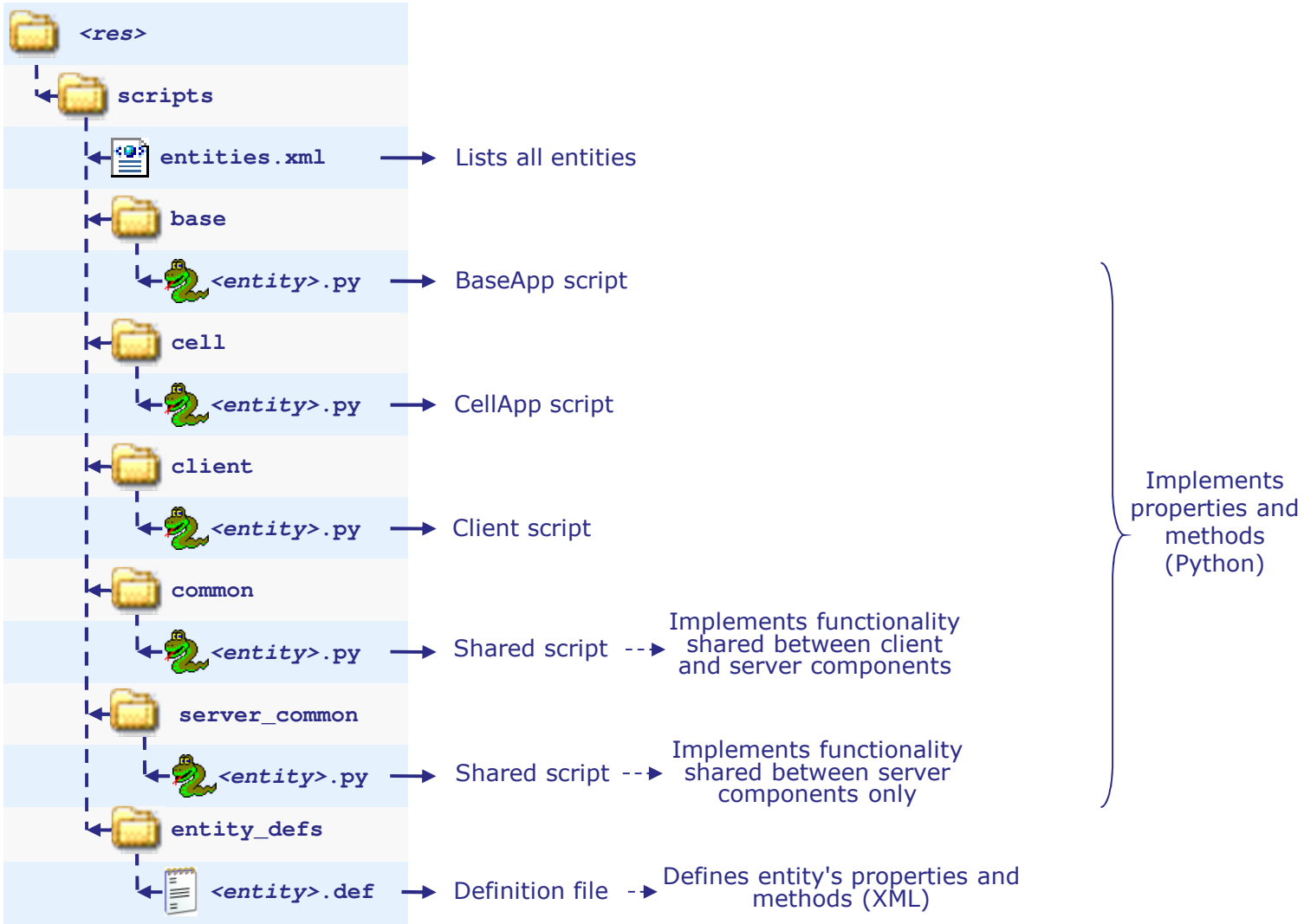
bigWORLD™ TECHNOLOGY

Session 2
Implementing an Entity



bigWORLD™ TECHNOLOGY

Entity Implementation Files

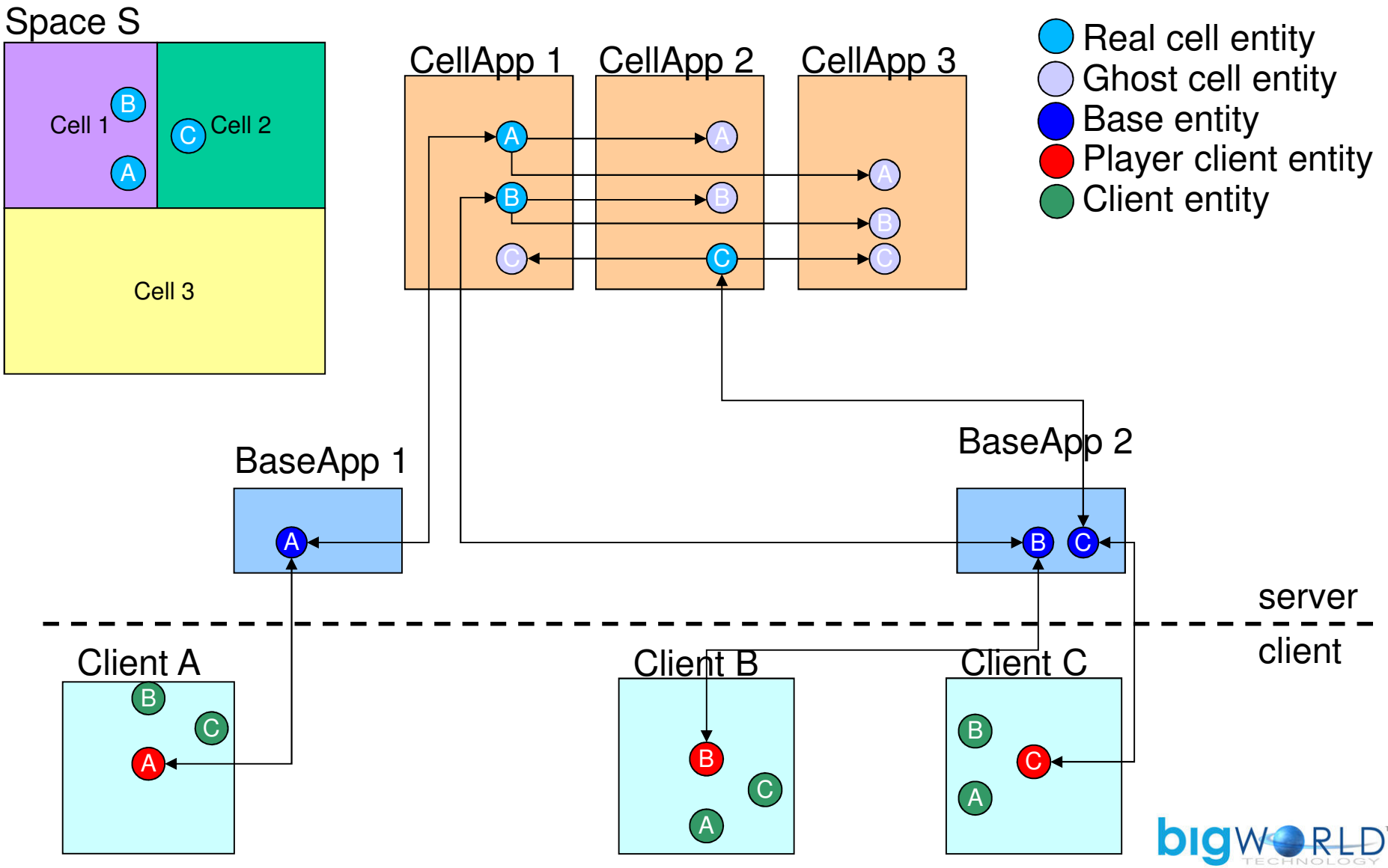


Entity Implementation

- Each entity must:
  - Be listed in `entities.xml`
  - Must have an `<Entity_name>.def`
- Each entity can:
  - Have up to 3 implementations (client/cell/base)
  - Re-use shared code from `common`
- Client / Server definition files must match

bigWORLD™ TECHNOLOGY

Distributed Entities



Simple Character Entity

```
<root>
   <Properties>
      <name>
         <Type>         STRING        </Type>
         <Flags>        ALL_CLIENTS   </Flags>
         <Persistent>   true          </Persistent>
      </name>
   </Properties>

   <ClientMethods>
   </ClientMethods>

   <BaseMethods>
   </BaseMethods>

   <CellMethods>
      <setName>
         <Exposed/>
      </setName>
   </CellMethods>
</root>
```

bigWORLD™ TECHNOLOGY

Entity Inheritance

- Entity definitions files support inheritance
  - `<res>/scripts/entity_defs/interfaces`
- Two inheritance mechanisms
  - `<Parent>`
    - Inherits everything
      - Properties / Methods
      - Volatile property specification
      - LoD Levels
    - Single level of inheritance
  - `<Implements>`
    - Inherits properties and methods
    - Multiple levels of inheritance

bigWORLD™ TECHNOLOGY

Player Entity

```
<root>
   <Implements>
      <Interface>  SimpleCharacter  </Interface>
   </Implements>

   ...

</root>
```

bigWORLD™ TECHNOLOGY

Entity Properties

- Type
  - As with most languages
  - Standardised for network communication / DB storage
- Default Value
  - Determined by type
  - Can be overridden in entity definition
- Distribution Flag
- Detail Level
- Volatile Information
- Persistence

bigWORLD™ TECHNOLOGY

Entity Definition Data Types

- Simple types
  - `INT8 / UINT8`
  - `FLOAT32 / FLOAT64`
  - `STRING`
  - `VECTOR3`
  - …
- Sequence Types
  - `ARRAY`
  - `TUPLE`

bigWORLD™ TECHNOLOGY

Entity Definition Data Types

```
<root>
   <Properties>
      <name>
         <Type>     STRING     </Type>
      </name>

      <armorColours>
         <Type>    TUPLE
           <of>    UINT8 </of>
           <size> 4      </size>
         </Type>
      <armorColours>
   </Properties>

   ...

</root>
```

bigWORLD™ TECHNOLOGY

Entity Definition Data Types

- Complex Types
  - `FIXED_DICT`
    - Dictionary like object
    - Fixed set of keys
  - `PYTHON`
    - Less efficient than `FIXED_DICT`
    - Rapid prototyping
    - Security Issues
      (Streaming Python objects from Client)
    - Uses Python's `pickle` module

bigWORLD™ TECHNOLOGY

Entity Definition Data Types

```
<root>
   <Properties>

      <characterInfo>
         <Type>  FIXED_DICT
            <Properties>
               <name>
                  <Type> STRING </Type>
               </name>

               <class>
                  <Type> UINT8  </Type>
               </class>
            </Properties>
         </Type>
      </characterInfo>

</Properties>

   ...

</root>
```

bigWORLD™ TECHNOLOGY

Type Aliases

- Re-useable custom type definitions
  - scripts/entity_defs/alias.xml

```
<root>
   <CHARACTER_INFO> FIXED_DICT
      <name> <Type> STRING </Type>
      </name>
      <class> <Type> UINT8 </Type>
      </class>
   </CHARACTER_INFO>
</root>
```

```
<root>
   <Properties>
      <CharacterInfo>
         <Type> CHARACTER_INFO </Type>
      </CharacterInfo>
</Properties>
   ...
</root>
```

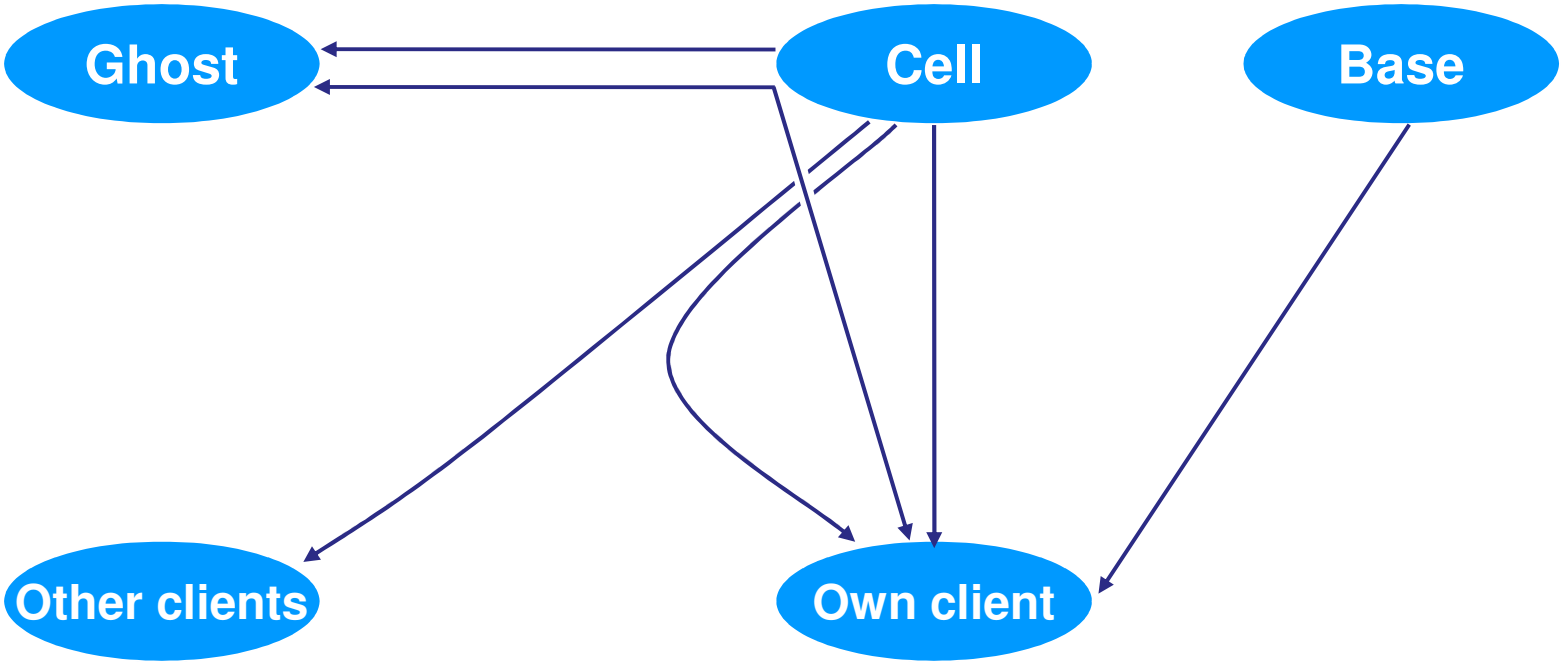
bigWORLD™ TECHNOLOGY

Entity Property Distribution

```
<root>
   <Properties>

      <name>
         <Type>   STRING       </Type>
         <Flags>  ??  </Flags>

      </name>

   </Properties>

   ...
</root>
```

bigWORLD™ TECHNOLOGY

Entity Property Distribution



Property Distribution – BASE

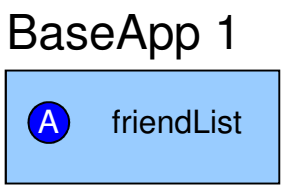
- Owned by: Base
- Available to: Base

- Examples:
  - List of people in a chat room
  - Items in a characters bag

bigWORLD™ TECHNOLOGY

BASE properties

- BASE properties do not propagate updates.
- Declaring them in the .def file means their values will be backed up and archived periodically.



Property Distribution – BASE_AND_CLIENT

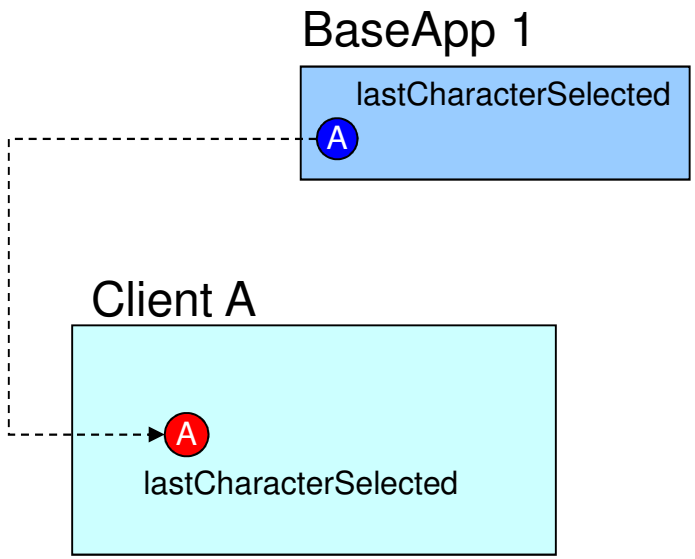
- Owned by: Base
- Available to: Base, Own Client

- Only synchronised when client entity is created.
- Subsequent changes must be propagated with explicit method calls
- Examples:
  - Same as `BASE`
  - Rarely used

bigWORLD™ TECHNOLOGY

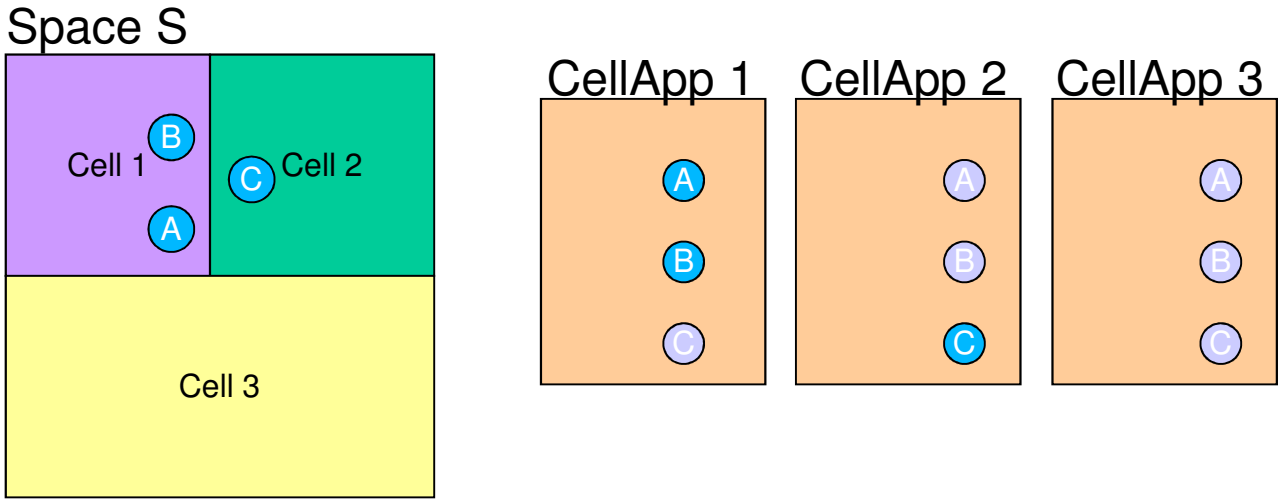
BASE_AND_CLIENT Properties

- Properties propagate their value **once** when the client entity is created.
- Client receives no further updates.



Cell Entity Layout Example

- CellApp 1,2 and 3 each have a cell in Space S
- 3 entities A, B and C in Space S.

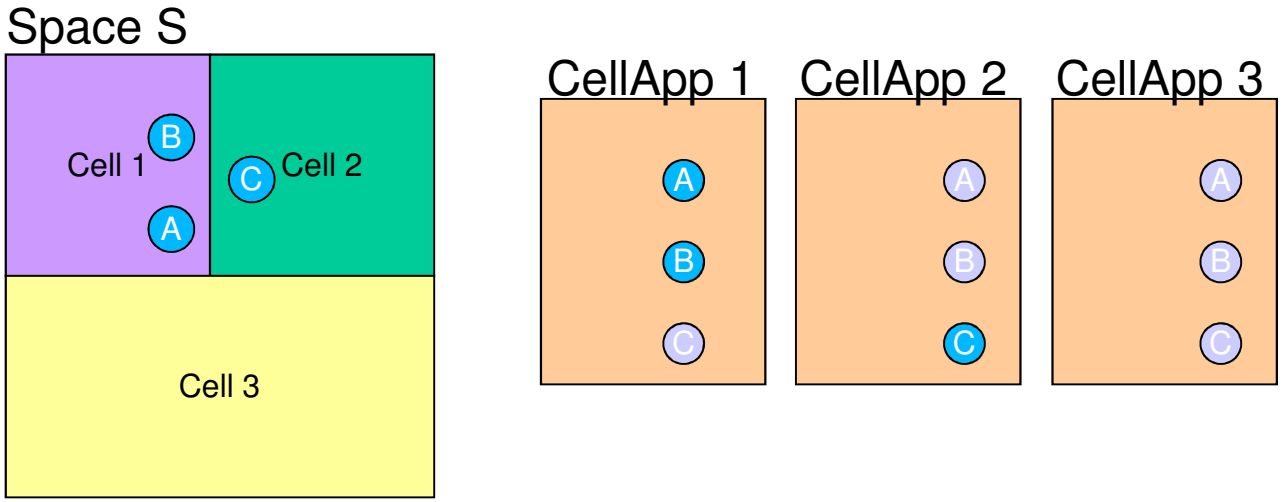


bigWORLD™ TECHNOLOGY

CellApp 1's Perspective

On CellApp 1's cell in space S:

- A and B are real entities.
- C is a ghost entity, ghosted from CellApp 2.

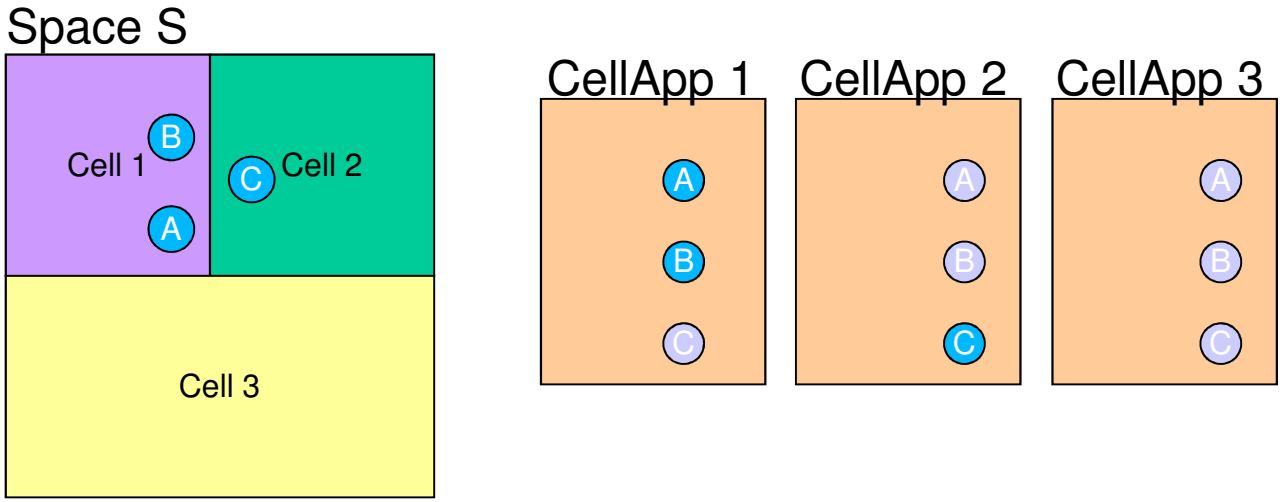


bigWORLD™ TECHNOLOGY

CellApp 2's Perspective

On CellApp 2's cell in space S:

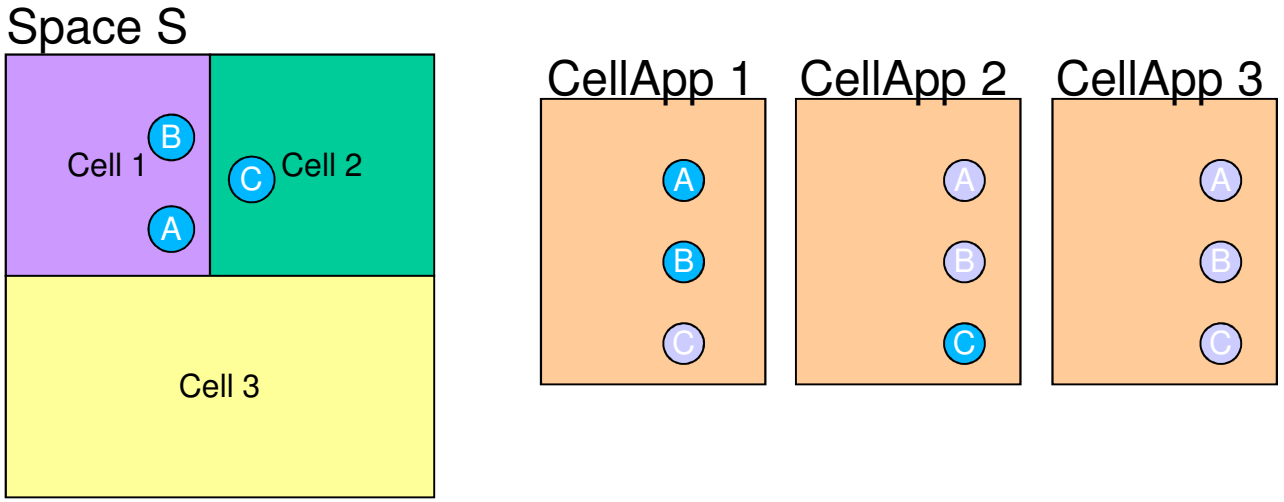
- C is a real entity.
- A and B are ghost entities, ghosted from CellApp 1.



CellApp 3's Perspective

On CellApp 3's cell in space S:

- A and B are ghost entities, ghosted from CellApp 1.
- C is a ghost entity, ghosted from CellApp 2.



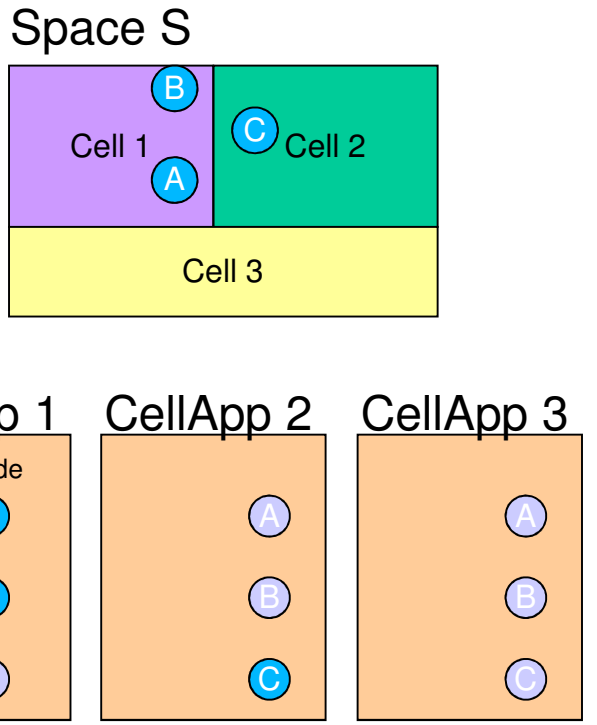
Property Distribution – CELL_PRIVATE

- Owned by: Real Entity
- Available to: Real Entity

- Examples:
  - NPC AI ‘thoughts’
  - Player properties relevant to game play that others shouldn’t see

CELL_PRIVATE Example

- Properties are owned by the real cell entity.
- Properties do not propagate from the real entity.
- Declaring them in the .def file means they will be offloaded when the cell entity changes cells. Additionally, the property will be backed up to the base entity periodically.
- A's nextPatrolNode property is not propagated to ghosts of A on CellApp 2 and CellApp 3.



bigWORLD™ TECHNOLOGY

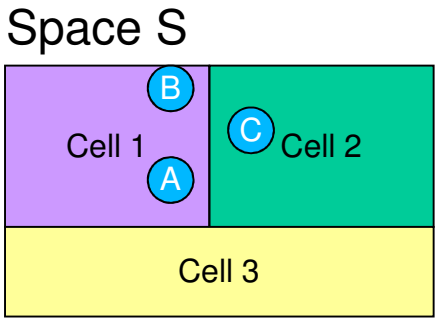
Property Distribution – CELL_PUBLIC

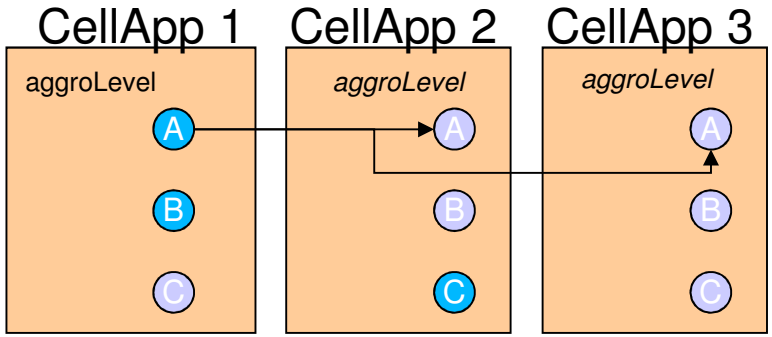
- Owned by: Real Entity
- Available to: Real Entity and its Ghost Entities

- Examples:
  - Aggro level of a creature (can be seen by other creatures but not other players)
  - Group name of an NPC

CELL_PUBLIC Example

- Properties are owned by the real cell entity.
- Updates propagated to ghost entities. Properties are available as read-only attributes on the ghost entity.
- A's aggroLevel property is propagated to CellApp 2 and 3's ghost entity of A.





bigWORLD™ TECHNOLOGY

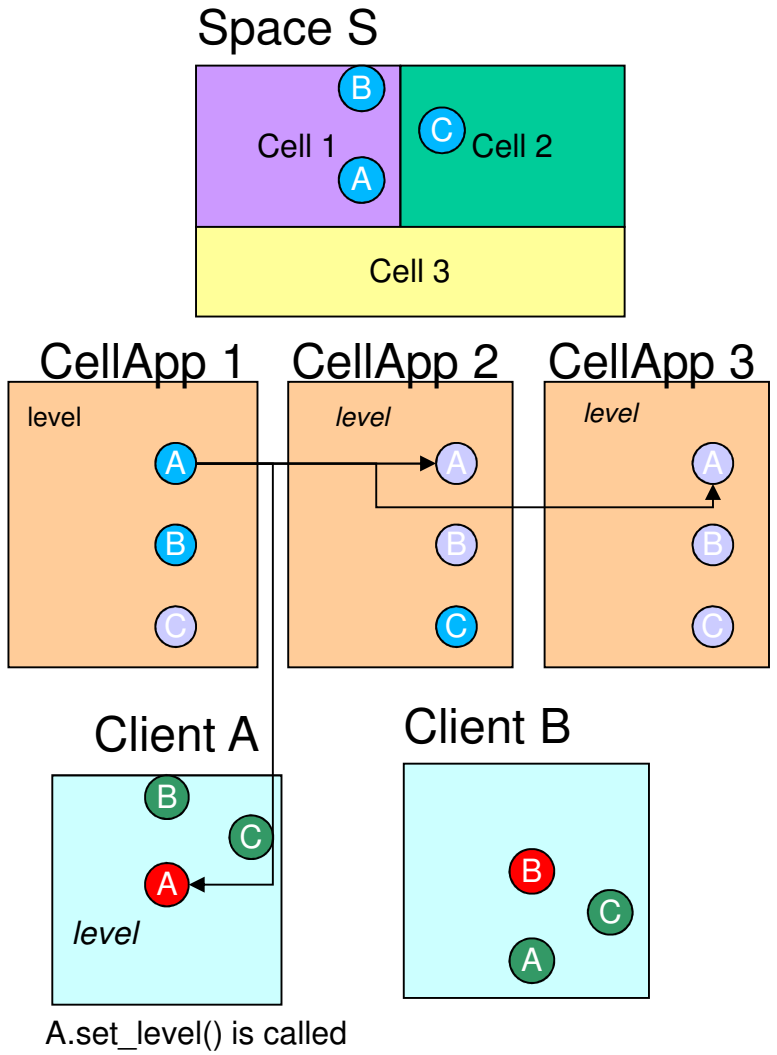
Property Distribution – CELL_PUBLIC_AND_OWN

- Owned by: Real Entity
- Available to:
  - Real Entity, Ghost Entities, and Own Client
- Examples:
  - Player debuff triggered by an NPC
    - Player uses property for interface visualisation
    - NPCs use property when deciding how to attack

bigWORLD™ TECHNOLOGY

CELL_PUBLIC_AND_OWN Example

- Properties are owned by the real cell entity.
- Updates propagated to ghost entities. Properties are available as read-only attributes on the ghost entity.
- Updates propagated to their own client entity. Script callback called when properties change.
- A's level property is propagated to CellApp 2 and 3, as well as the client of A.
- Client B does not have knowledge of the level property, and does not receive updates for it.



bigWORLD TECHNOLOGY

Property Distribution – ALL_CLIENTS

- Owned by: Real Entity
- Available to:
  - Real Entity, Ghost Entities, Own Client, Other Clients

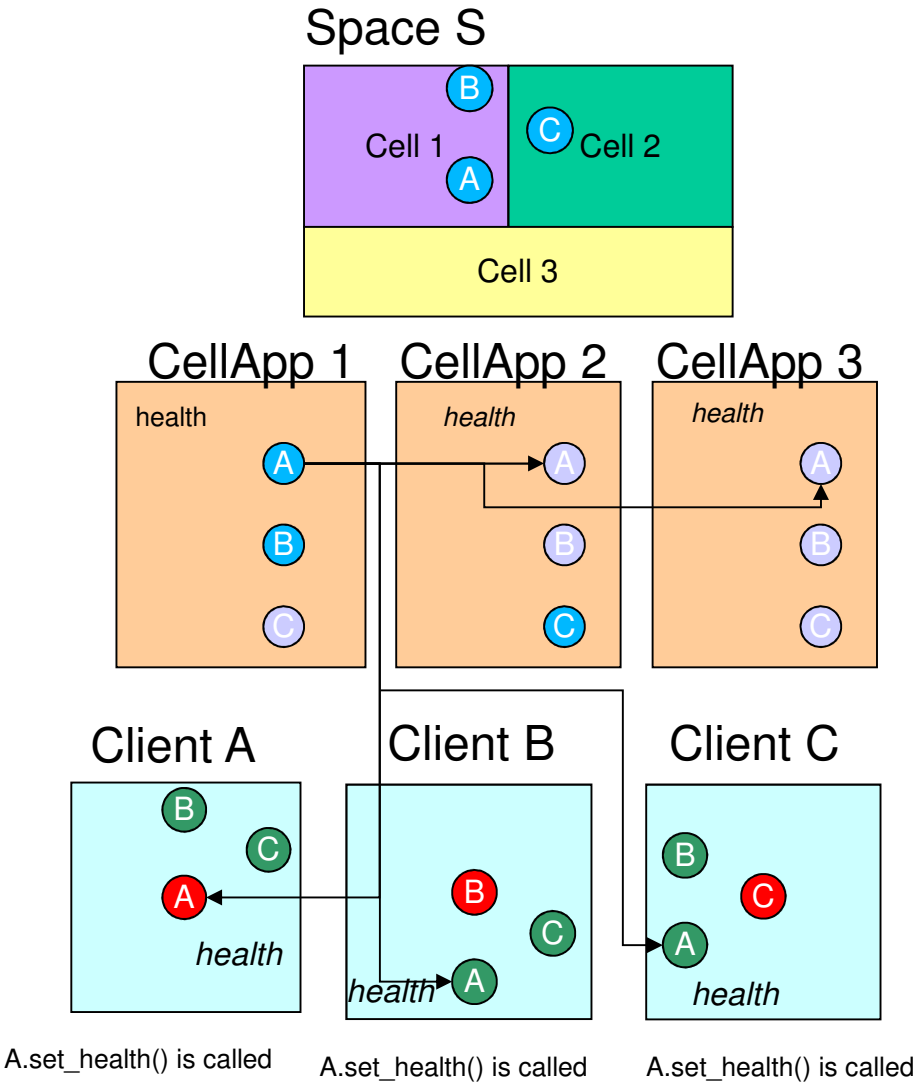
- Examples:
  - Name of the player
  - Health of a player / creature

Cell property updates will trigger `set_<property_name>()` on the client

bigWORLD™ TECHNOLOGY

ALL_CLIENTS Example

- Properties are owned by the real cell entity.
- Updates propagated to ghost entities. Properties are available as read-only attributes on the ghost entity.
- Updates propagated to their own client entity. Script callback called when properties change.
- Updates are propagated to other clients that have that entity in their AoI.
- A's health property is propagated to A's ghosts on CellApp 2 and 3.
- A's health property is propagated to the clients of A, B and C, and the A.set_health() callback is called.



Property Distribution – OWN_CLIENT

- Owned by: Real Entity
- Available to: Real Entity, Own Client

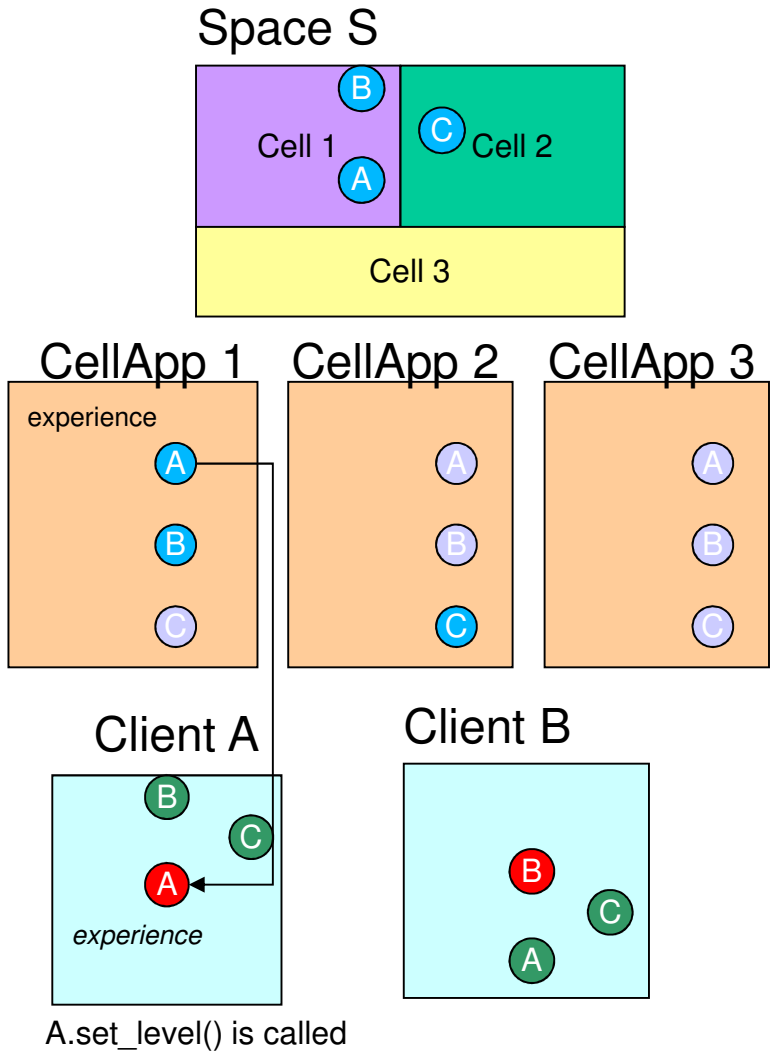
- Examples:
  - Character class of a player
  - XP of a player

Cell property updates will trigger `set_<property_name>()` on the client

bigWORLD™ TECHNOLOGY

OWN_CLIENT Example

- Properties are owned by the real cell entity.
- Updates propagated to their own client entity. Script callback called when properties change.
- Entity A's experience property is propagated to A's client.



Property Distribution – OTHER_CLIENTS

- Owned by: Real Entity
- Available to:
  - Real Entity, Ghost Entities, Other Clients

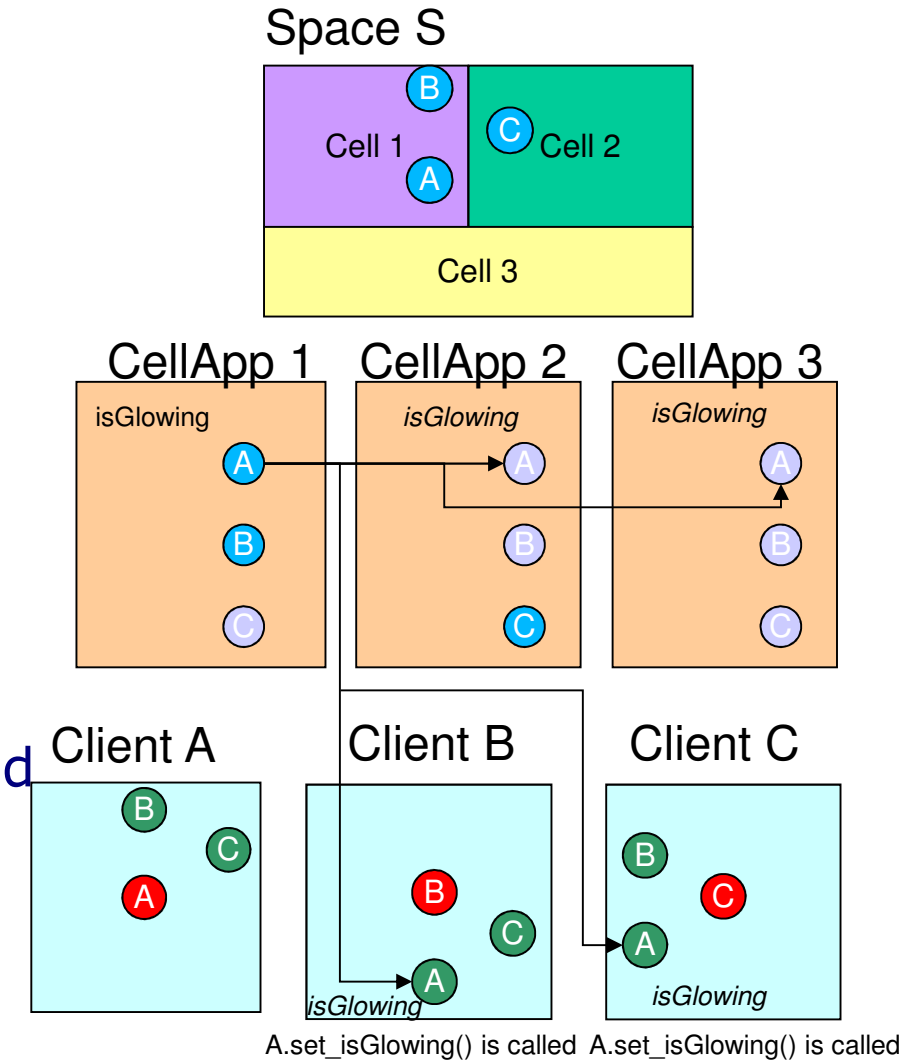
- Examples:
  - State of dynamic world items (eg: doors, buttons, loot items)
  - Particle system effect type
  - Player that is sitting on a seat

Cell property updates will trigger `set_<property_name>()` on the client

bigWORLD™ TECHNOLOGY

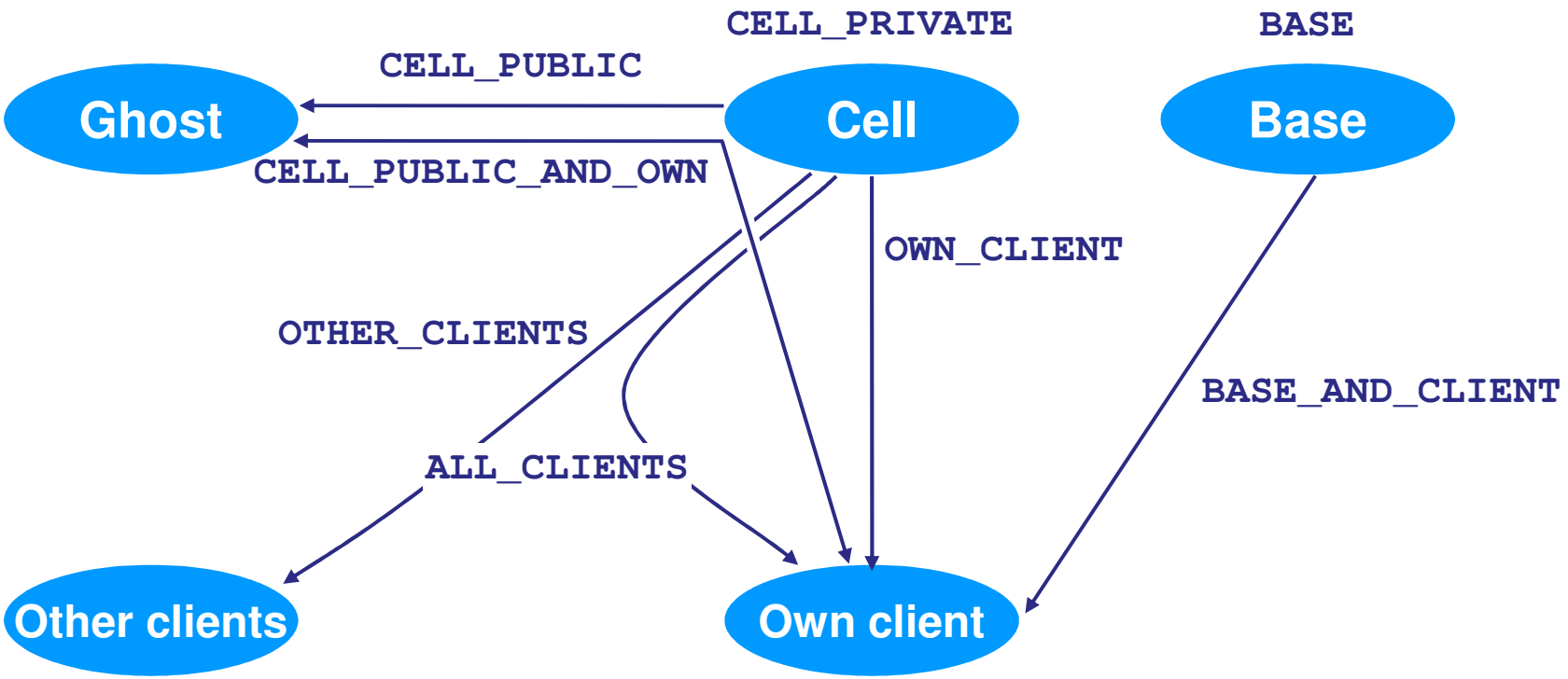
OTHER_CLIENTS Example

- Properties are owned by the real cell entity.
- Updates propagated to ghost entities. Properties are available as read-only attributes on the ghost entity.
- A's isGlowing property is propagated to A's ghosts on CellApp 2 and 3.
- A's isGlowing property is propagated to the clients of B and C.



bigWORLD TECHNOLOGY

Entity Property Distribution



bigWORLD™ TECHNOLOGY

Entity Property Distribution

```
<root>
   <Properties>

      <name>
         <Type>    STRING         </Type>
         <Flags>   ALL_CLIENTS    </Flags>

      </name>

   </Properties>

   ...
</root>
```

bigWORLD™ TECHNOLOGY

Property Detail Levels

- Influences client property updates
- Typically applied to visualised properties
- Bandwidth saving mechanism
- Use if required, not necessary
- Specified with `<DetailLevel>`
- Detail levels aliased with `<LodLevels>`

bigWORLD™ TECHNOLOGY

Property Detail Levels

```
<root>

   <LoDLevels>
      <level>  20  <label> NEAR    </label>  </level>
      <level> 100  <label> MEDIUM  </label>  </level>
      <level> 250  <label> FAR     </label>  </level>
   </LodLevels>

   <Properties>

      <name>
         <Type>           STRING       </Type>
         <Flags>          ALL_CLIENTS  </Flags>
         <DetailLevel>    NEAR   </DetailLevel>
      </name>

   </Properties>

   ...
</root>
```

bigWORLD™ TECHNOLOGY

Volatile Properties

- Optimised protocol
- Only interested in the most recent value
- Position (x,y,z)
- Yaw, Pitch, Roll

Entity Persistence

- Some entities and their properties may need to persist across server restarts
- Defined on a per-property basis
- Causes entities to be written to DB
- Generates `self.databaseID` when in DB

```
<root>
   <Properties>
      <name>
         <Type>          STRING         </Type>
         <Flags>         ALL_CLIENTS    </Flags>
         <Persistent>    true           </Persistent>
      </name>
   </Properties>

   ...
</root>
```

bigWORLD™ TECHNOLOGY

Entity Properties

- Cell
  - Entity data is frequently accessed
  - Data copied when crossing cell boundaries
  - Data backed up to the base
  - Notifies Client of state change:
    - Property changes
    - When an entity enters the player's AoI
- Base
  - More complex / less frequently accessed
  - Not automatically propagated to client

bigWORLD™ TECHNOLOGY

Entity Properties

- Client
  - Accesses a subset of server properties
  - Properties propagated from the cell
  - Cell property changes invoke `set_<property>()`
  - Example:

```
def set_health( self, oldHealth ):
  if self.health == 0 and oldHealth > 0:
    self.doDeath()

def setNested_inventory( path, oldValue ):
  print "Inventory slot %d changed" % (path[-1],)

def setSlice_inventory( path, oldValues ):
  print "%d added. %d removed" % \
    (path[-1][1] - path[-1][0], len( oldValues ))
```

bigWORLD™ TECHNOLOGY

Entity Methods

- Definitions separated
  - Client / Cell / Base
- Arguments must be defined
- Base / Cell methods can be exposed to the client
- Client methods can specify a maximum callable distance
- Must be in entity definition file for remote invocation

bigWORLD™ TECHNOLOGY

Entity Methods

```
<root>
   <Properties>
      ...
   </Properties>

   <ClientMethods>
     ...
   </ClientMethods>

   <BaseMethods>
      <addToFriendsList>
         <!-- Entity ID -->
         <Arg>  INT32  </Arg>

         <!-- Expose to client -->
         <Exposed />
      <addToFriendsList>
   </BaseMethods>

   <CellMethods>
      ...
   </CellMethods>
</root>
```

bigWORLD™ TECHNOLOGY

Exposed Server Methods

- Not all server methods are exposed
- Explicitly expose with `<Exposed />`
- Exposed CellMethods
  - Automatically receive EntityID of the caller
  - Generally checks whether
    `self.id == callerID`
- Exposed BaseMethods
  - Can only be called by their own Client

bigWORLD™ TECHNOLOGY

Entity Methods

- Client method LoD
  - Helps reduce client bandwidth usage
  - Produces a visual effect on a distant entity
  - Useful when broadcasting client messages

```
<root>
   ...

   <ClientMethods>
     ...
     <smile>
       <DetailDistance> 30 </DetailDistance>
     </smile>
     ...
   </ClientMethods>

   ...
</root>
```

bigWORLD™ TECHNOLOGY

Entity Implementation

- Entities existence is contextual
- If no entity is required in the context, neither is a Python script

Entity Presence Examples

Base	Cell	Client
`SpawnPoint`		
Chat room	Spawned wildlife*	
Player entity		
Server AI/NPC's		

* Entities without a base part are not fault tolerant

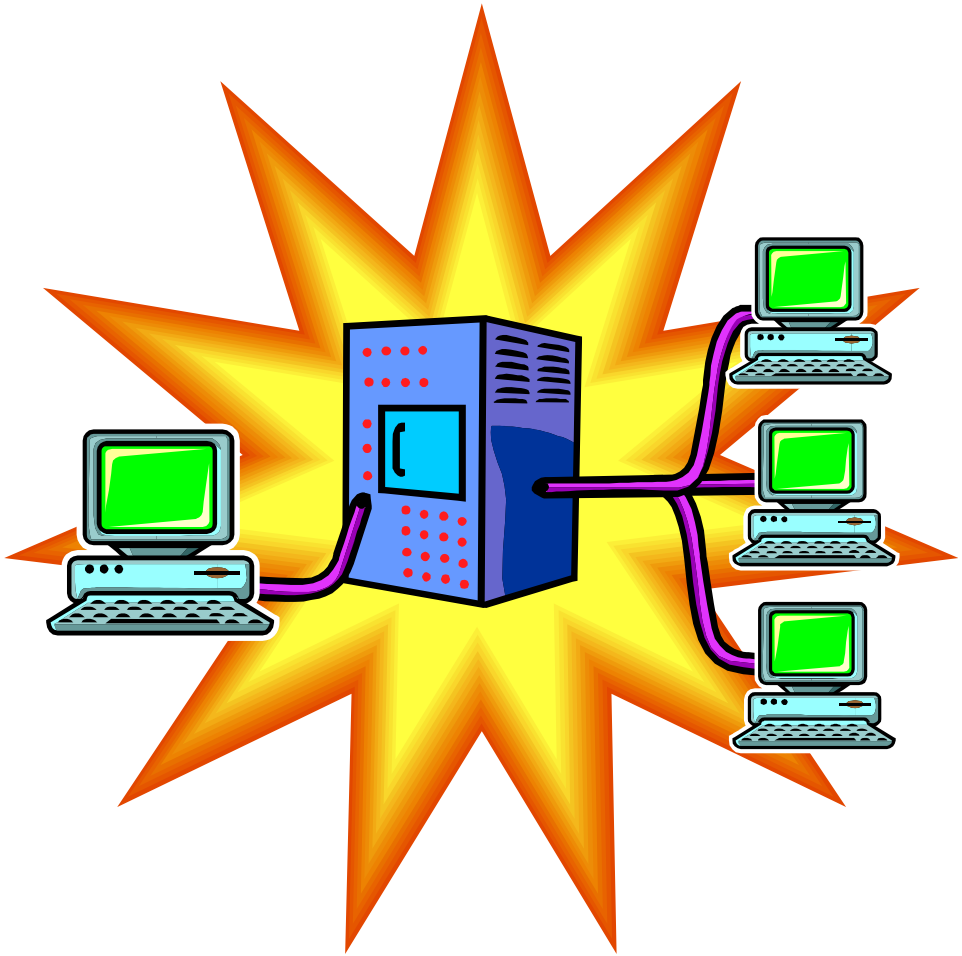
bigWORLD™ TECHNOLOGY

Guidelines for script development

- Offload functionality to BaseApp when possible
- Keep persistent entity properties to a minimum
- Avoid calling `writeToDB()` too much
- Try to avoid nested data types
  - Eg: Arrays of arrays
- Script exceeding 1 game tick can negatively impact server performance

bigWORLD™ TECHNOLOGY

Session 3
Entity Communication



bigWORLD™ TECHNOLOGY

Mailboxes

- Reference to remote entity
  - e.g.: Cell part of an entity
- Allows remote method calls
  - e.g.: `mb` is a cell entity mailbox

    `mb.someMethod( a, b )`

    will invoke `someMethod()` wherever the real cell entity exists.
- Intra-entity communication
  - e.g.: cell part to base part
- Inter-entity communiction
  - e.g.: cell part of entity A to base part of entity B

bigWORLD™ TECHNOLOGY

Mailboxes

- Different types
  - Base
  - Cell
  - Client
  - Single-hop
  - Multi-hop
    - Cell via Base
- Some BigWorld methods may only accept certain types
  - See the Python API documentation for details

Mailboxes

- Entities have mailbox members
  - Client entities: `self.cell`, `self.base` (for players)
  - Base entities: `self.cell`
  - Proxy entities: `self.cell`, `self.client`
  - Cell entities:
    - `self.base`
    - `self.ownClient`
    - `self.allClients`
    - `self.otherClients`

bigWORLD™ TECHNOLOGY

Mailboxes

- Mailbox is automatically created when passing an entity object to a server method with a `MAILBOX` argument
- Example:
  - Cell method `talkToMe()` has a `MAILBOX` argument
  - On a cell, entityA calls:
    ```
    entityB.talkToMe( self )
    ```
  - Entity A's mailbox is passed to Entity B
    ```
    def talkToMe( self, mailbox ):
        mailbox.sendMsg( "hello" )
    ```
  - Entity A's `sendMsg()` method is called with `"hello"`

bigWORLD™ TECHNOLOGY

Storing Mailboxes

- Base mailboxes are valid for the life of the entity
  - Base entities never change BaseApps
  - Can be used for long-term inter-entity communication
  - Must implement a notification mechanism if storing base mailboxes
- Cell mailboxes are only guaranteed for a short time
  - Cell entities may change CellApps
  - Do not store Cell **`MailBox`**'s as properties.
  - Use immediately, then discard

bigWORLD™ TECHNOLOGY

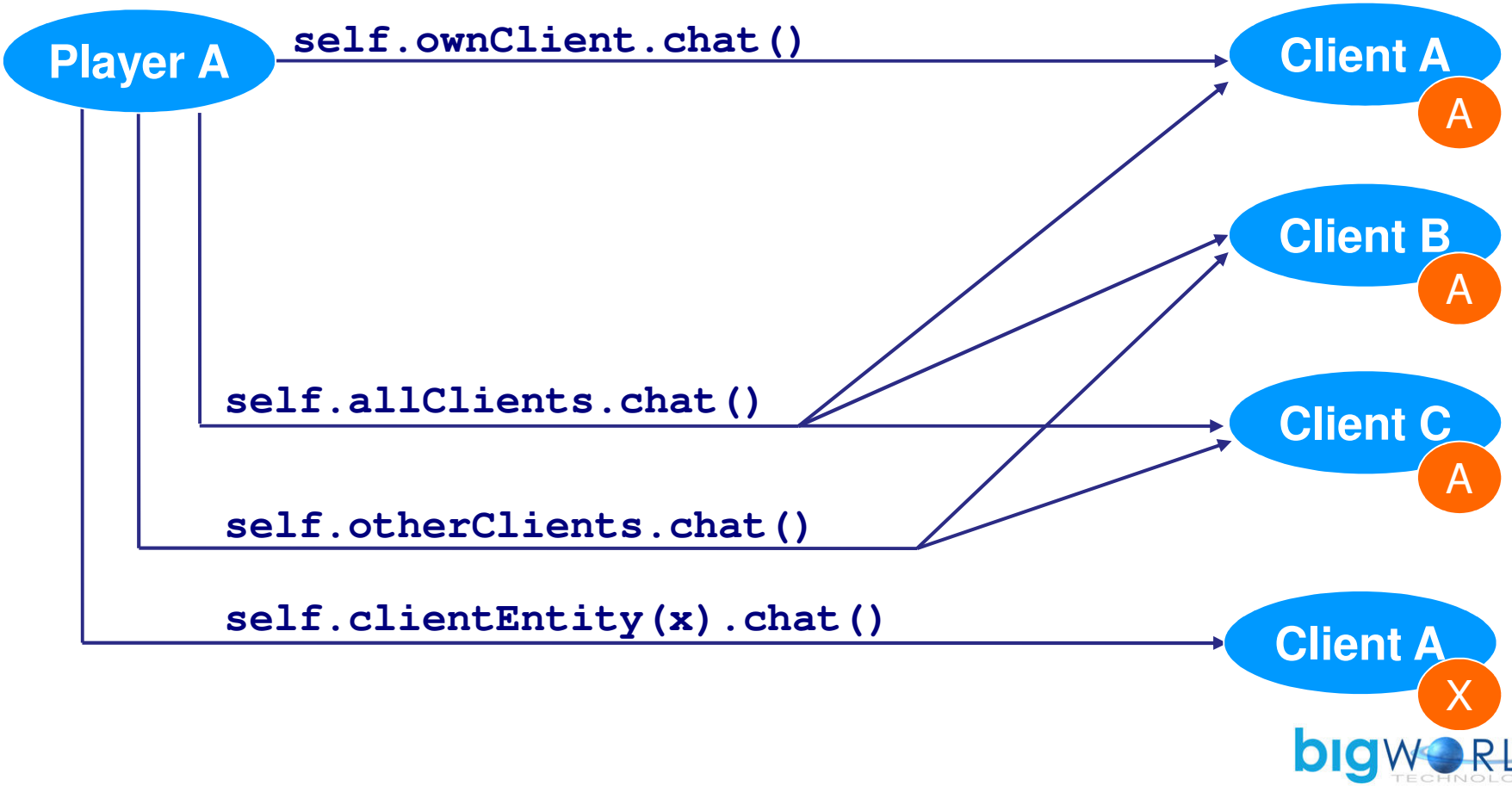
Storing Mailboxes

- Cannot pass mailboxes to or from clients
  - Cannot trust the client
  - Use an entity ID instead
- Mailboxes cannot be stored in the database
  - IP addresses will change on server restart

bigWORLD™ TECHNOLOGY

Cell to Client Communication

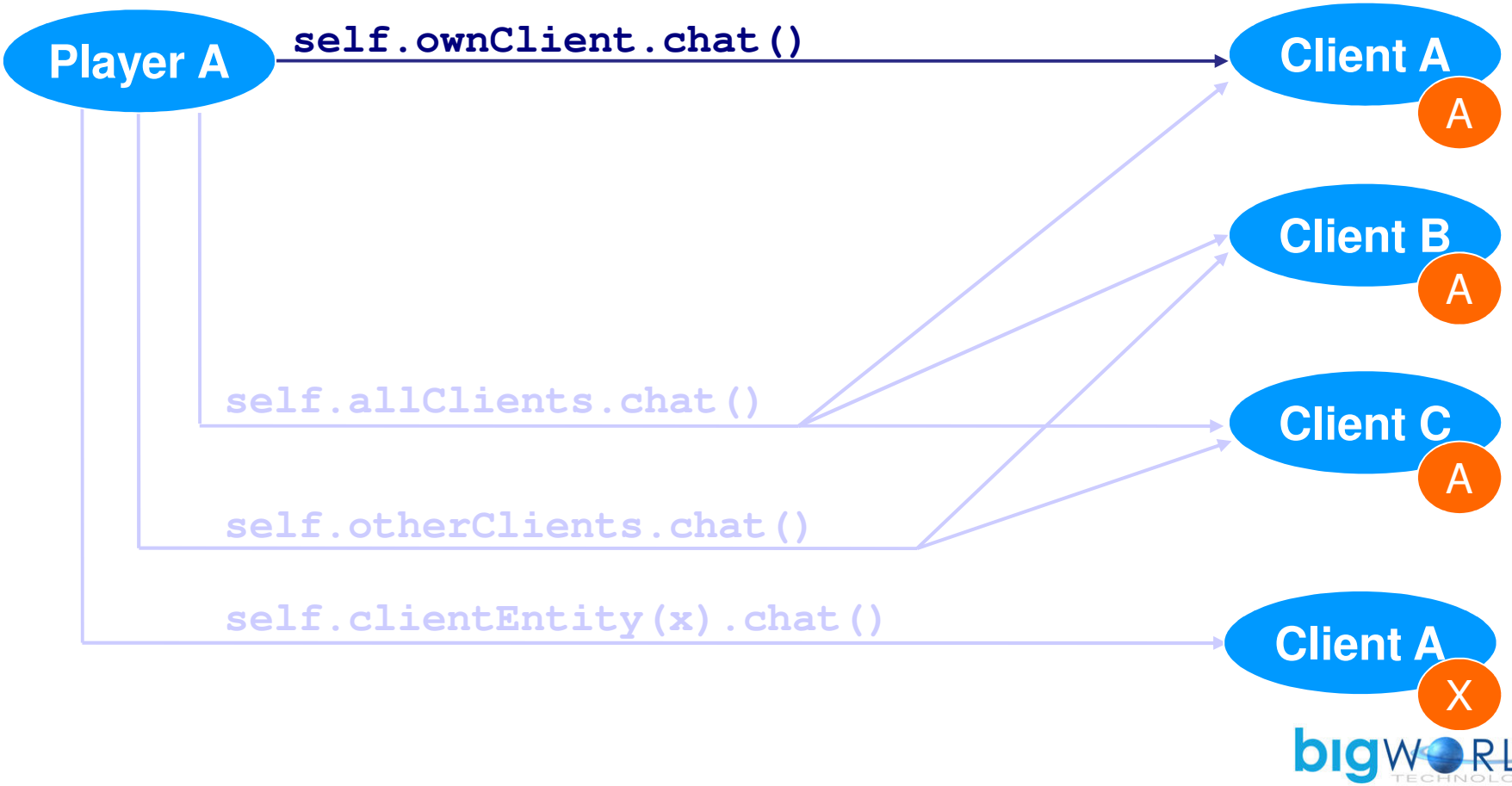
- **`self` is `Player A`**
- Player must be a proxy on the BaseApp
- These **`MailBox`**'s cannot be passed around



bigWORLD™ TECHNOLOGY

Cell to Client Communication

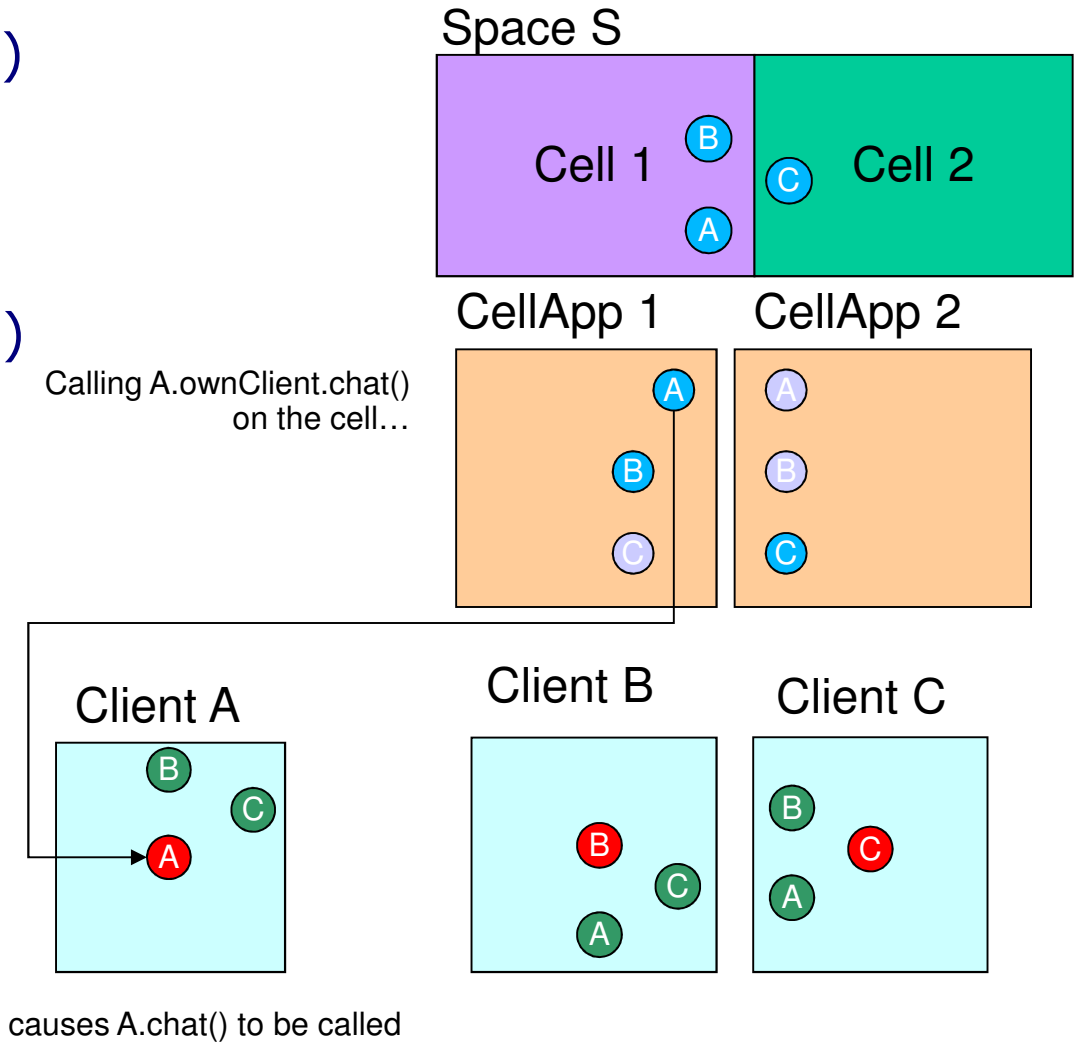
- **`self`** is **`Player A`**
- Player must be a proxy on the BaseApp
- These **`MailBox`**'s cannot be passed around



bigWORLD™ TECHNOLOGY

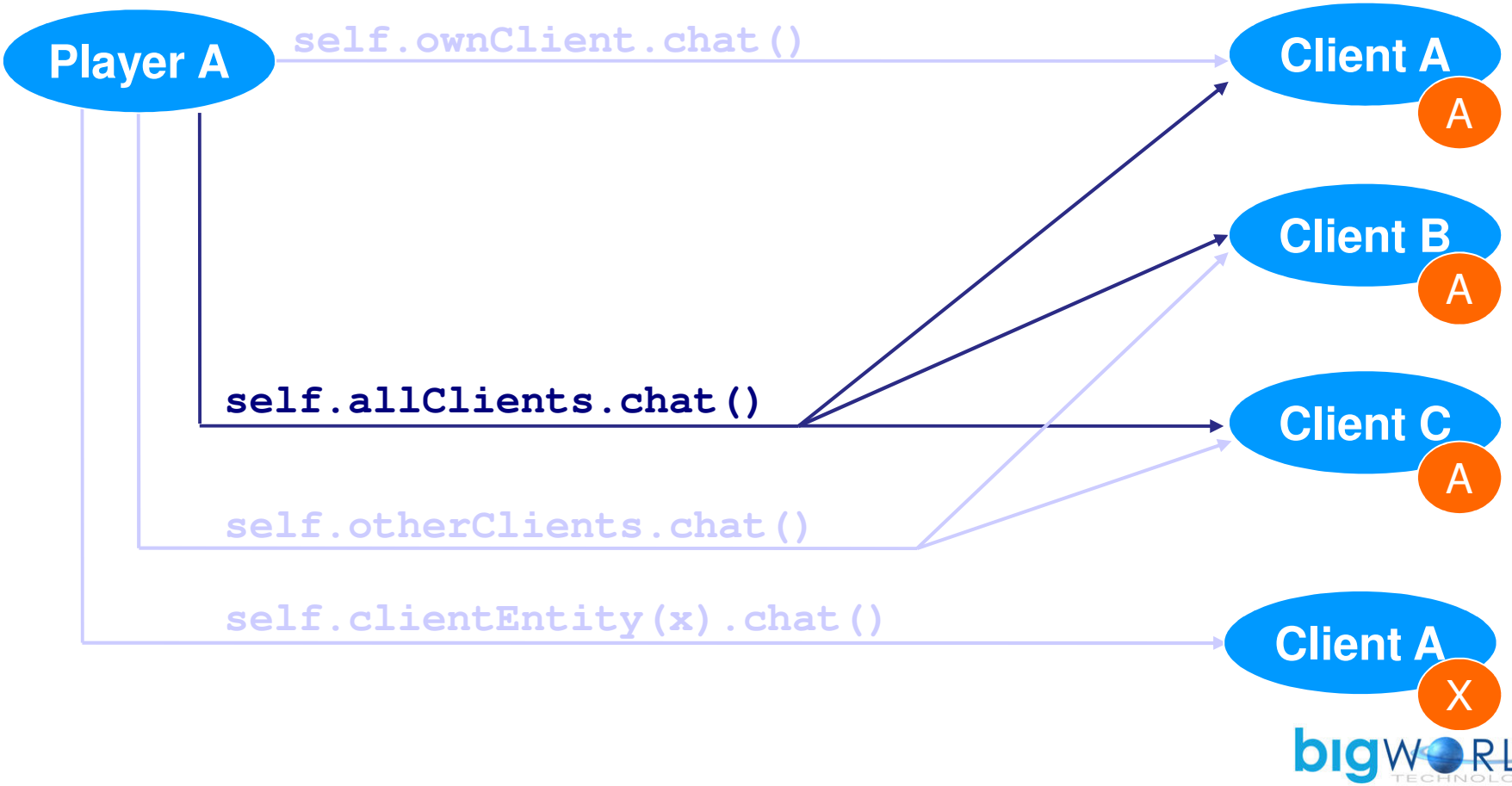
Entity.ownClient Method Call Example

- Calling self.ownClient.chat() will call chat on Entity A in Client A.
- No other client that can observe A will have A.chat() called.



Cell to Client Communication

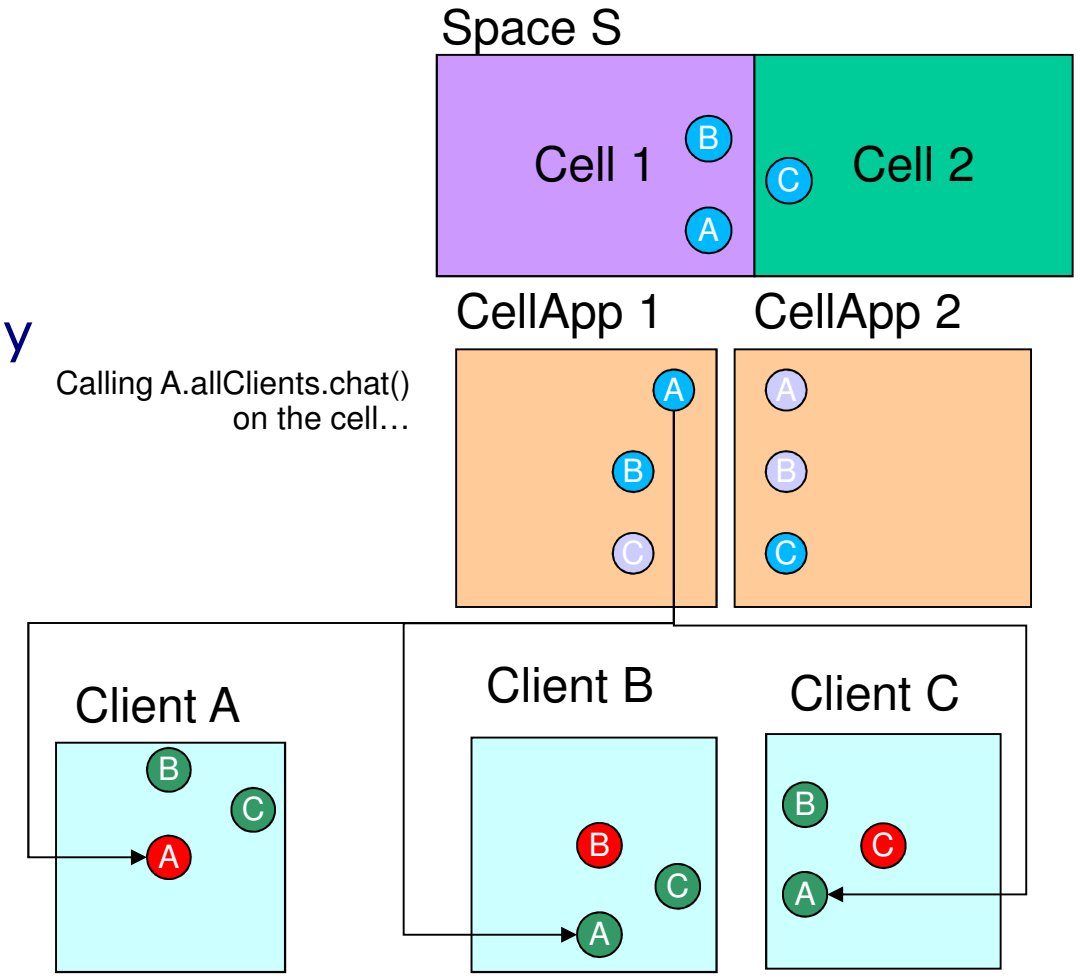
- **`self`** is **`Player A`**
- Player must be a proxy on the BaseApp
- These **`MailBox`**'s cannot be passed around



bigWORLD™ TECHNOLOGY

Entity.allClients Method Call Example

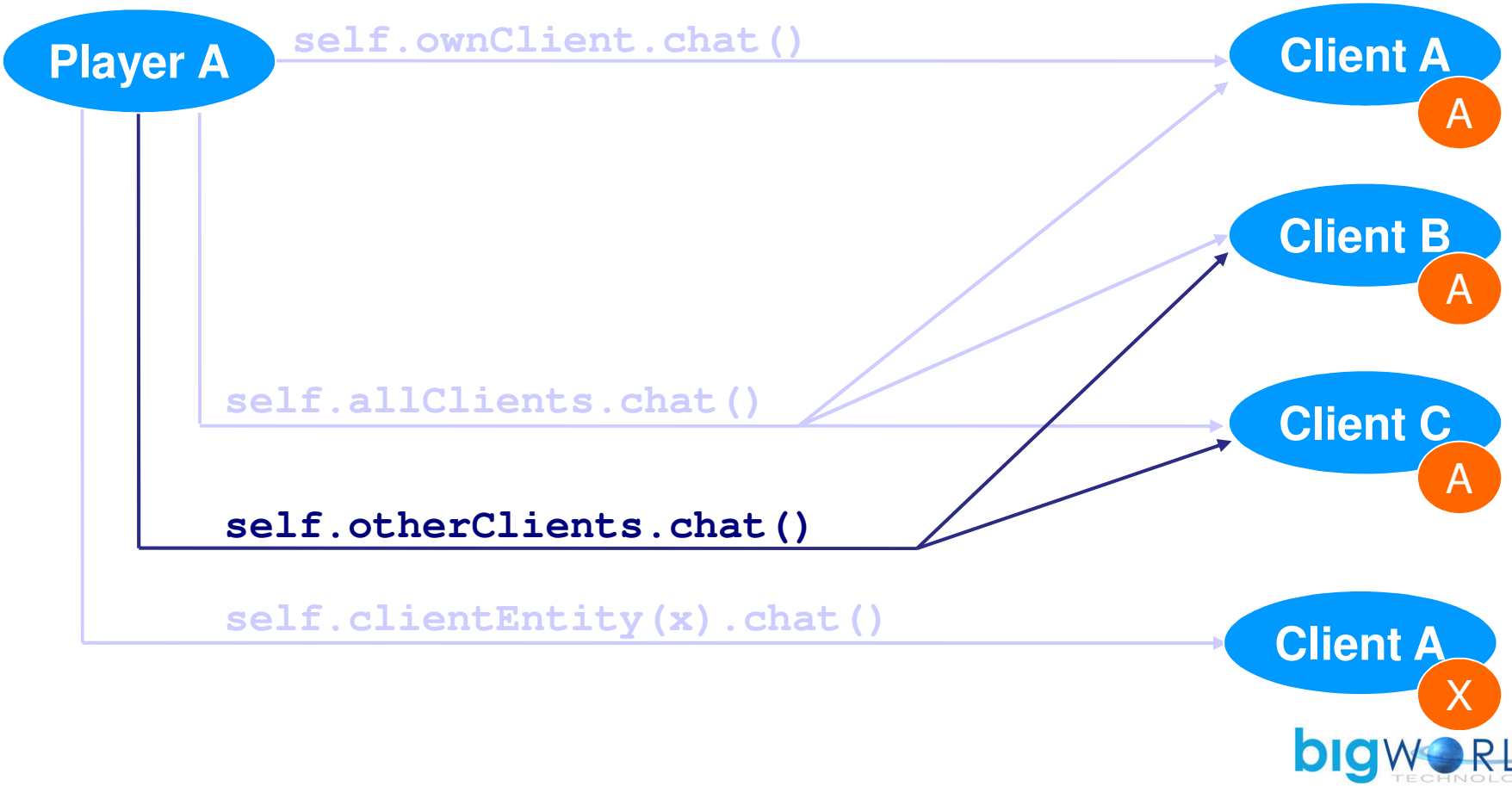
- Calling self.allClients.chat() will call chat() on Entity A for all Clients that can observe A.
- Clients can observe A if they are in the same space and they are within the AoI distance.



bigWORLD TECHNOLOGY

Cell to Client Communication

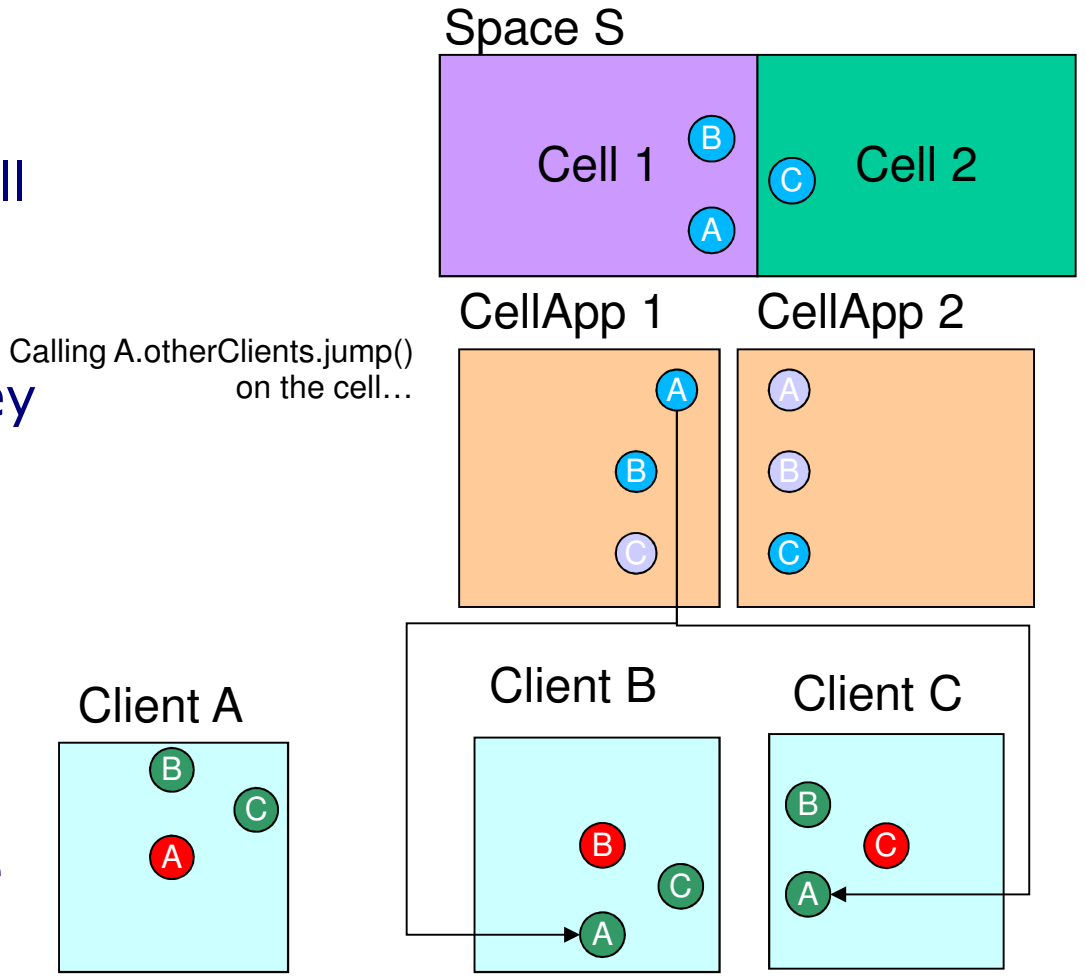
- **`self` is `Player A`**
- Player must be a proxy on the BaseApp
- These **`MailBox`**'s cannot be passed around



bigWORLD™ TECHNOLOGY

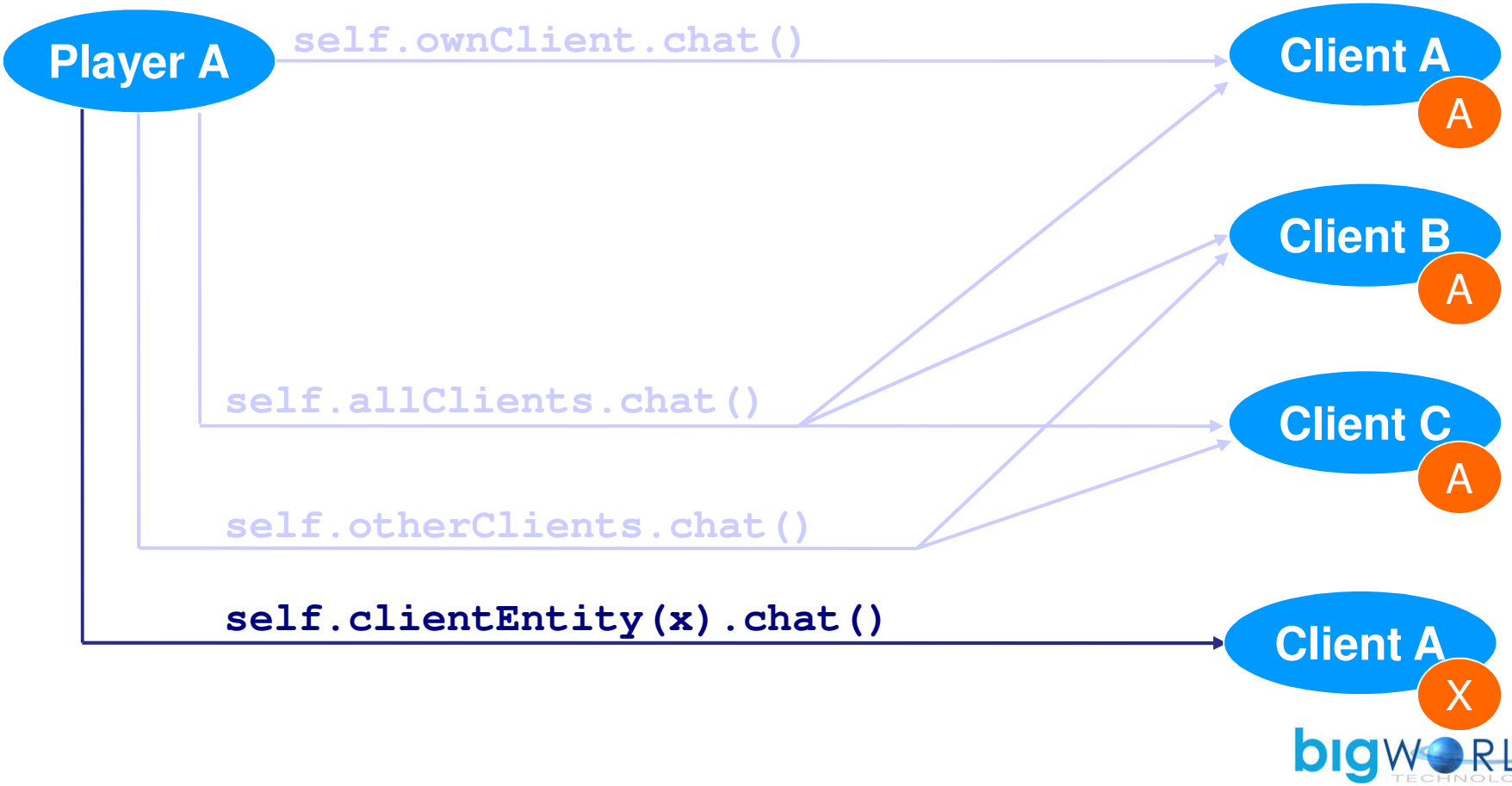
Entity.otherClients Method Call Example

- Calling self.otherClients.chat() will call chat() on Entity A for all Clients that can observe A, except A itself.
- Clients can observe A if they are in the same space and they are within the AoI distance.
- Often used for client-initiated actions that have an immediate effect on the player's client, but must be broadcast to other players. For example, jumping.



Cell to Client Communication

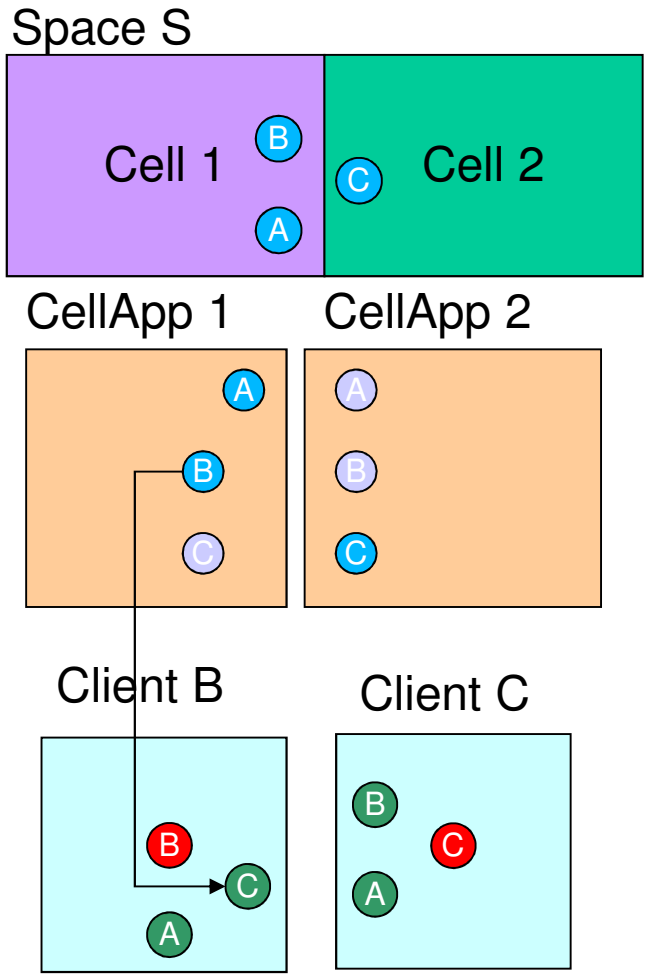
- **`self`** is **`Player A`**
- Player must be a proxy on the BaseApp
- These **`MailBox`**'s cannot be passed around



bigWORLD™ TECHNOLOGY

Entity.clientEntity(id) Method Call Example

- Specific entities on specific clients can be targeted for remote client method calls.

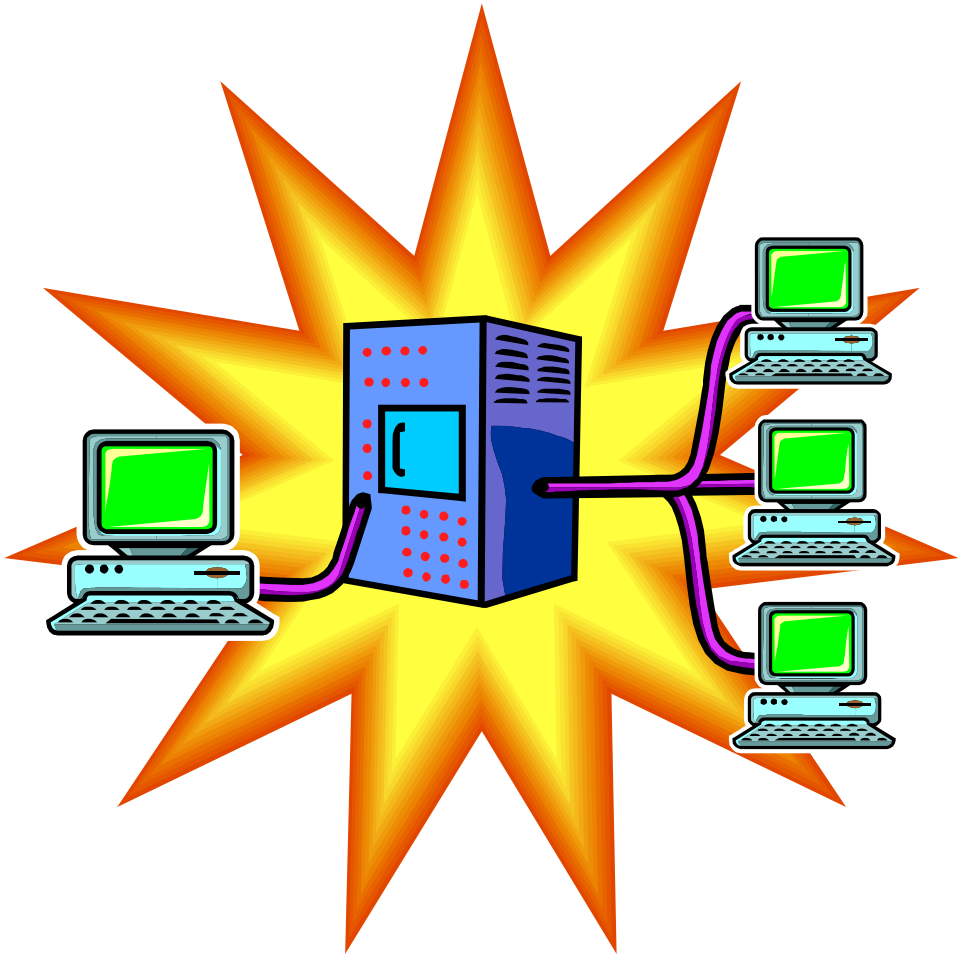


Calling B.clientEntity(C.id).wave() on the cell…

… causes C.wave() to be called on client B.

bigWORLD™ TECHNOLOGY

Session 4
Core Entity Components



bigWORLD™ TECHNOLOGY

Base Entity Types

- Base
  - Python script derives from `BigWorld.Base`
  - Store large / complex data
    - Helps reduce system load when cell entity moves across a cell boundary
  - Fixed mailbox for receiving method calls
- Proxy
  - Python script derives from `BigWorld.Proxy`
    - `BigWorld.Proxy` internally derives from `BigWorld.Base`
  - Communication point for the client
  - Clients can attach and detach as needed

bigWORLD™ TECHNOLOGY

Base Entity Attributes

- Attributes of entities derived from `BigWorld.Base`

Attribute	Description
`id`	Unique entity identifier. Shared across cell, base and client.
`databaseID`	Entity's persistent ID in the DB. Zero if not persistent. 64 bits
`cell`	MailBox for cell entity, if it exists
`cellData`	Dictionary-like object containing the cell properties if the cell entity does not exist

Base Proxy Attributes

- **`BigWorld.Proxy`** derives from **`BigWorld.Base`**, and is used as a parent class for base entities that have a proxy

- Additional attributes

Attribute	Description
`client`	MailBox for communicating with this Entity on the Client process
`clientAddr`	Address and port of the client machine
`bandwidthPerSecond`	Amount of information to send to client *
`wards`	List of entity ID's that client is controlling. This affects how information is passed between client and cell, and how that information is processed. Read-only

* **`Rarely used, limits volatile property updates`**

bigWORLD™ TECHNOLOGY

Base Entity Methods

Method	Description
`addTimer(initOffset [,repeatOffset, userData] )`	▪ Adds a new timer (offsets are in seconds), returning its ID ▪ Entity must implement `onTimer(self, timerID, userData)`
`delTimer( timerID )`	▪ Removes specified timer
`createCellEntity( [cellMailBox])` *	▪ Creates the cell entity on the cell that the mailbox refers to ▪ Useful to instantiate an entity on cell when it is first created on base ▪ If `cellMailBox` is not passed, then `Base.cellData[spaceID]` is used
`createInNewSpace()` *	▪ Creates the cell representation of an entity in a new space (including a new cell to manage it) ▪ Useful when creating an entity to control new space **(*e.g.*, Mission Manager)**
`destroyCellEntity()`	▪ Destroys cell Entity, retaining base counterpart ▪ Use 'teleport' on CellApp to move between spaces, rather than destroying and recreating the cell entity ▪ Base gets `onLoseCell` called, and `Base.cellData` property is set with the cell entity properties
`destroy()`	▪ Destroys the Base part of this Entity ▪ Cell Entity must have already been destroyed ▪ Useful when removing Entity from game ▪ Often used in the onLoseCell callback

* Cell entity properties are transferred from `Base.cellData` and it becomes inaccessible

bigWORLD™ TECHNOLOGY

Cell Entity Attributes

- Some attributes defined by **`BigWorld.Entity`**

Attribute	Description
`id`	Unique entity identifier. Shared across cell, base and client.
`spaceID`	BigWorld space where entity is located
`vehicle`	Entity's current vehicle. **`None`** if entity is not on a vehicle.
`position`	Entity's position in world
`roll`	Entity's orientation
`pitch`	
`yaw`	
`direction`	Entity's facing direction. Composed of **`roll`**, **`pitch`**, **`yaw`**.
`volatileInfo`	Determines when each volatile element is updated. Defaults to values defined by the **`.def`** file
`topSpeed`	Entity's maximum speed. For physics checking.

bigWORLD™ TECHNOLOGY

Cell Entity Methods

Method	Description
`destroySpace()`	▪ Destroys all entities in the space, effectively destroying the space
`destroy()`	▪ Destroys the Cell part of this Entity ▪ Removes entity from the space
`entitiesInRange( range [,entityType, position] )`	▪ Finds all entities within given range ▪ Can find entities outside of AoI but not outside of cell ▪ Spherical test
`isReal()`	▪ Returns whether the entity is a real entity or ghost entity
`setAoIRadius( radius [, hysteresis] )`	▪ Changes the AoI radius from default of 500m ▪ Must be less than ghost distance (default 500m)
`teleport( nearbyEntityMBRef, position, direction )`	▪ Changes the position of the entity in the same space ▪ Puts entity into another space – same space as the entity referred to by `nearbyEntityMBRef`

- All entity attributes and methods can be found in Python API Documents:
  - For BaseApp: bigworld/doc/api_python/python_baseapp.chm
  - For CellApp: bigworld/doc/api_python/python_cellapp.chm
  - For client: bigworld/doc/api_python/python_client.chm

bigWORLD™ TECHNOLOGY

Entity Life Cycle (typical)

- Base part is created first
  - From database, chunks or in code
  - Can exist without cell part - cellData property
  - Cannot be destroyed while cell part exists
  - Usually decide to self-destruct in **`onLoseCell()`** callback
- Cell part is created by base part
  - Cell-only entities can be created using script
- Client part is created when it enters the AoI of player entity
  - onEnterWorld()/onLeaveWorld() callbacks should be used instead of __init__() method

bigWORLD™ TECHNOLOGY

Entity Creation

- Entity instantiation on a cell propagates to appropriate clients with the next network update
- Recommended methods for creation:
  - Base Entities:

    `BigWorld.createBaseAnywhere()`

    - Alternatives:

      `createBaseLocally()`

      `createBaseRemotely()`

      `createBase...FromDB()`

      `createEntityOnBaseApp()`

bigWORLD™ TECHNOLOGY

Entity Creation

- Recommended methods for creation:
  - Cell Entities:
    `createCellEntity()`
    `createInNewSpace()`
  - Cell entity properties can modified after being loaded from DB but prior to entity creation.
    - See Base API doc: `BigWorld.Base.cellData`
  - Cell Only Entities:
    `createEntity()`
    - Called from the cell
    - No fault tolerance

bigWORLD™ TECHNOLOGY

Entity Destruction

- Cell entities are destroyed as part of game logic
- Base entity can't be destroyed unless cell no longer exists
- Destroying the cell component:
  - On cell: `Entity.destroy()`
  - On base: `Base.destroyCellEntity()`
  - `Base.onLoseCell()` called on cell component death
- Destroying the base component:
  - `Base.destroy()`
  - Causes a `writeToDB()` if persistent

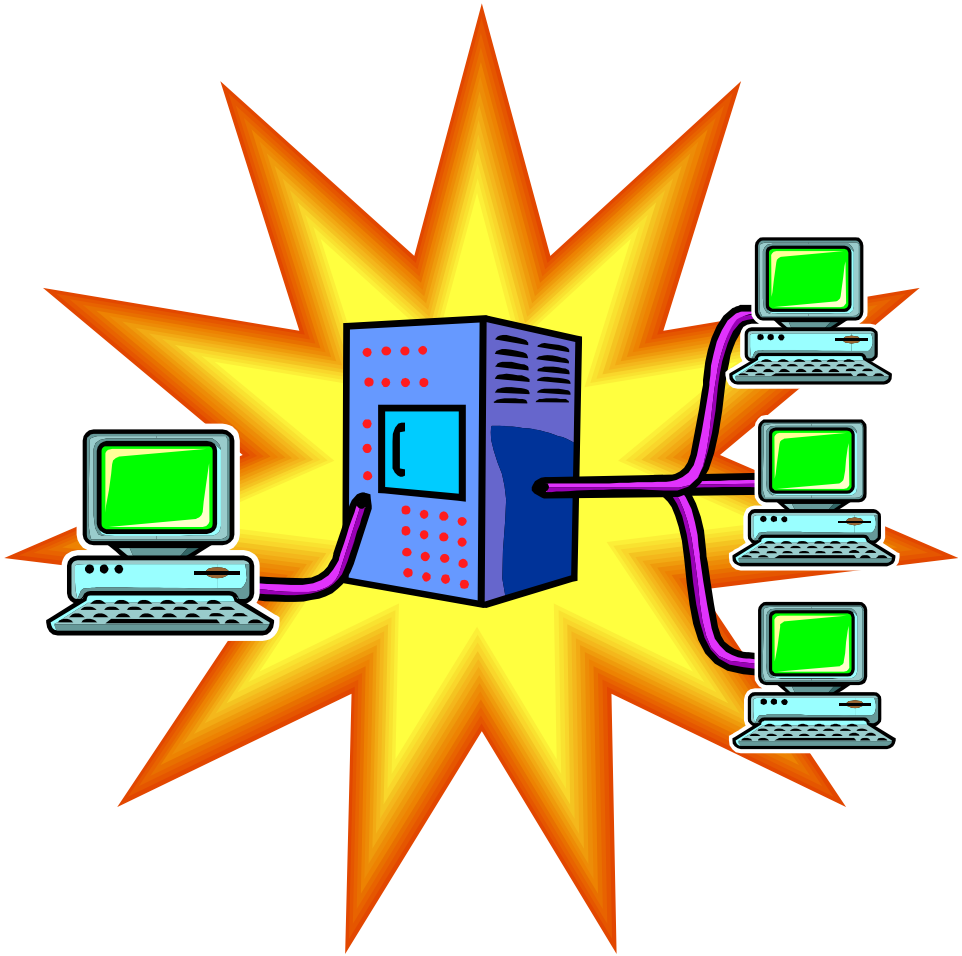
bigWORLD™ TECHNOLOGY

Fault Tolerance

- Cell properties backed up on base
- Base properties backed up on another BaseApp
- Persistent properties backed up on database
  - Archiving: Continuous round-robin
- Fault tolerance vs. Disaster recovery
  - Disaster = multiple simultaneous component failure

bigWORLD™ TECHNOLOGY

Session 5
Cell Functionality



bigWORLD™ TECHNOLOGY

Controllers and Entity Extras

- Extend cell entities using C++
- Used when performance is critical
- Often used together
- Controllers
  - Implement functionality that requires background processing over many ticks
  - Calls back Python script when done
- EntityExtras
  - Implement accessors to internal state or derived state
  - Generally used for querying / modifying Controller parameters
- Example in **`bigworld/src/server/egextra`**

bigWORLD™ TECHNOLOGY

Controller Overview

- Used to implement processing heavy logic
- C/C++ for performance (as opposed to script)
- Copied as entity crosses cell boundaries
- No limitation on number of controllers per entity
- Each instance returns a Controller ID
  - Destroy with `Entity.cancel( id )`
- Can invoke a callbacks in their entity's Python script

bigWORLD™ TECHNOLOGY

EntityExtra Overview

- Conceptually an extension of the Entity class
- Only one EntityExtra of the same type per entity
- Exposed to Python as normal entity methods / properties
- Created only as needed, upon access from either C++ or Python
  - Saves on data storage
- Not moved with entity between cells
  - Saves on data transfer
- Create your own:
  - Inherit from **`EntityExtra`** in `src/server/cellapp/entity_extra.hpp`

bigWORLD™ TECHNOLOGY

Entity Cell Functionality

- There are many spatially-pertinent functions available to the Cell part of an entity
  - Navigation
  - Proximity (traps)
  - Vision
  - Rotators
- These functions are implemented as Controllers and EntityExtras

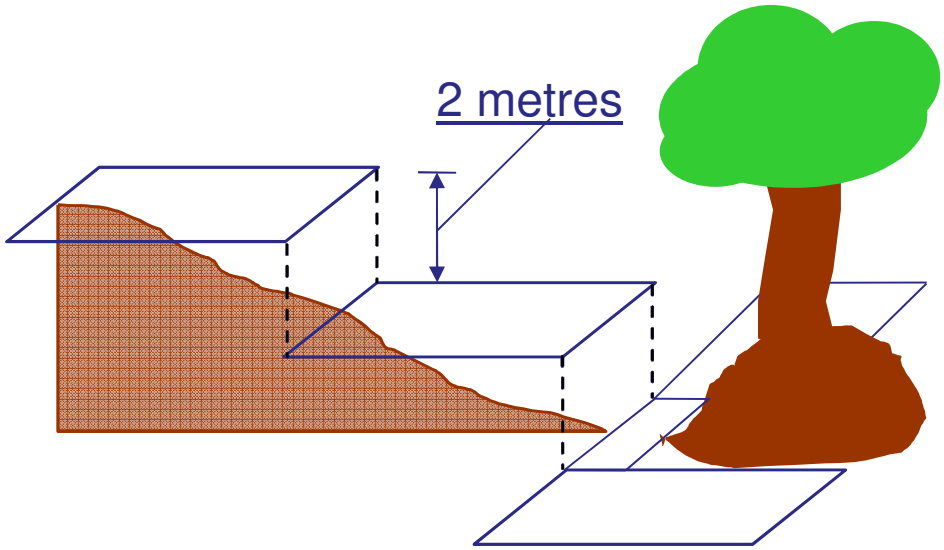
bigWORLD™ TECHNOLOGY

Entity Navigation

- Provides functionality for entity movement and path-finding
- Navigation Controller finds paths using a NavMesh of NavPolys pre-generated with NavGen from the static collision scene

NavMesh and NavPolys

- NavMesh is a graph of horizontal polygons called NavPolys
- Entity traverses the top of each NavPoly, then jumps up or down to the next
- The client drops the entities up to 2 metres to place them correctly on the scene



Entity Navigation Methods

- Straight line movement
  - `moveToPoint()`
  - `moveToEntity()`
- Navigation (using the NavMesh)
  - `navigate()`
  - `navigateStep()`
  - `navigateFollow()`
- General
  - `canNavigateTo()`
  - `getStopPoint()`

bigWORLD™ TECHNOLOGY

Entity Proximity

- The `ProximityController` implements an infinitely high, axis-aligned, square column trap
- Y-axis checking should be performed after trap notification
- An Entity can have multiple proximity traps
- Add a proximity trap with: `Entity.addProximity()`

bigWORLD™ TECHNOLOGY

Entity Vision

- Server side entities need to see too
  - Wild boars charge when you get too close
- 2 components
  - Vision
  - Visibility
- Entities with Vision can only see entities with Visibility

Entity Vision

- Methods
  - `addVision()`
  - `addScanVision()`
  - `setVisionRange()`
  - `entitiesInView()`
- Properties
  - `seeingHeight`
  - `visibleHeight`
  - `canBeSeen`
  - `shouldDropVision`

bigWORLD™ TECHNOLOGY

Vehicles Overview

- Any entity can be used as a vehicle
- A vehicle is any entity that will move another one
- Examples: Moving platform, car, horse, battleship
  - Moving platform
    - Player entity should be scripted on client side to automatically board and alight from vehicle, based on position, and still be free to move
  - Car
    - Player entity should select, then enter vehicle and transfer movement control to move it, all on client-side
    - Player may ride on or in vehicle, depending on client-side animation, or be made invisible to 'transform' into that entity
- Vehicles are recursive
  - Player might ride a horse on a moving platform in a battleship at sea

bigWORLD™ TECHNOLOGY

Vehicle Methods

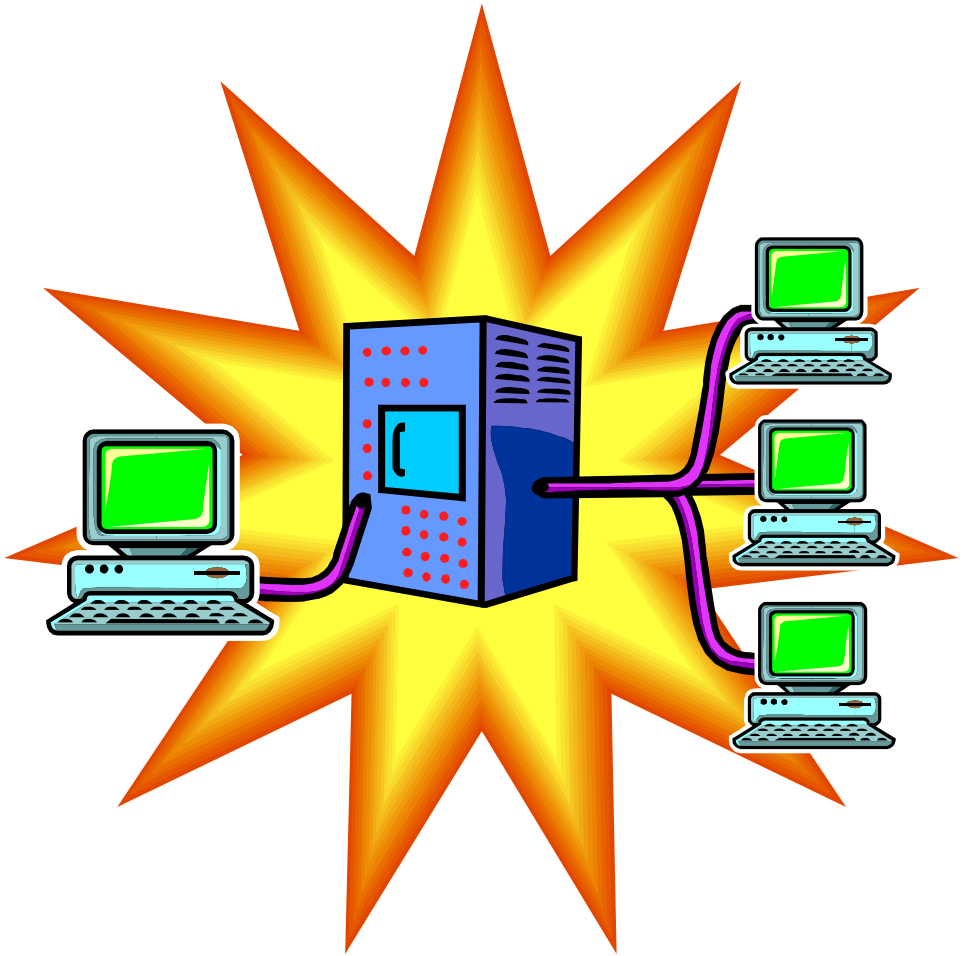
- Passenger EntityExtra provides interface to vehicles for cell entities
  - **`Entity.boardVehicle(`*`vehicleEntityID`*`)`**
    - Associates entity with vehicle entity
  - **`Entity.alightVehicle()`**
    - Disassociates entity from current vehicle entity
    - Must be called before boarding another vehicle entity
- Being on a vehicle just links the entities together to translate position and direction
  - Therefore, animation, changes in response to input, velocity, etc, must all be performed on the client
- In their movement messages, player entities sends the innermost vehicle ID they are on, and position and direction updates relative to that vehicle

bigWORLD™ TECHNOLOGY

Controlling Another Entity

- Consists of two parts:
  - Client sending position updates to new entity: `BigWorld.controlEntity()`
  - Server accepting position updates for entity: `Entity.controlledBy`
    - Set to the base mailbox of the player controlling the entity
- Cannot go outside of AoI of player (Proxy entity)
  - Therefore, only really suitable for vehicles that player is on
- Alternatively, can transfer control from one player to another (both must have Proxy base part)
  - `Proxy.giveClientTo()`
    - `Entity.controlledBy` is automatically set to the new player
  - Disruptive – AoI is destroyed and recreated, space is reloaded

Session 6
BigWorld Server Setup



bigWORLD™ TECHNOLOGY

Server Configuration

- **`bw.xml`** – Server configuration file
  - Specifies many server runtime parameters
  - Located in server resource folder
  - Fully documented in Server Operations Guide
- Personality Scripts
  - Implements global callbacks
  - Uses BigWorld Python Interface to handle specific system-wide events
    - Examples: Startup, recovery, shutdown
  - Separate CellApp and BaseApp scripts is default
  - Script name specified in **`bw.xml`**
    - Default: **`BWPersonality`**

bigWORLD™ TECHNOLOGY

Server Personality Scripts

- CellApp Personality script can set up game in **`onCellAppReady`**
  - See example
  - Imports BigWorld to get access to BigWorld functions
  - **`BigWorld.addSpaceGeometryMapping(1, None, "spaces/main")`**
    - **`1`** is default space, only present if **`useDefaultSpace`** is enabled in **`bw.xml`**
    - **`None`** is optional geometry transform matrix to alter geometry mapping
    - Folder path leads to a BigWorld **`space.settings`** file describing chunks
- BaseApp Personality script can set up game in **`onBaseAppReady`**
  - Good place to create any global bases
  - Should create a new space here, if not using the default one
- Both scripts should perform cleanup
  - Either when **`onBaseAppShuttingDown`** or **`onCellAppShuttingDown`** is called
  - BaseApps also receive **`onBaseAppShutDown`** when shutdown is near completion
- Game state may have been recovered from an interrupted game, so take care not to overwrite it or create duplicate global entities
- Personality scripts can perform other tasks as required
  - Good as a global game script, but do not put everything into it
  - Use a modular approach, with separate script files for each logical unit

Loading Entities

- Entities are loaded from:
  - WorldEditor – place entities in WorldEditor, these can be loaded using `BigWorld.fetchEntitiesFromChunk` in Python
  - Personality scripts – typically create singleton entities
  - Database – entities can be marked as auto-loaded in a previous run of the game
  - RunScript/PythonConsole – load entities by executing script in an already-running server process
- BigWorld.fetchEntitiesFromChunk
  - Used on the BaseApp for loading entities that are defined in chunk files using WorldEditor
  - When the base part of an entity is created, the base script should create the cell part

bigWORLD™ TECHNOLOGY

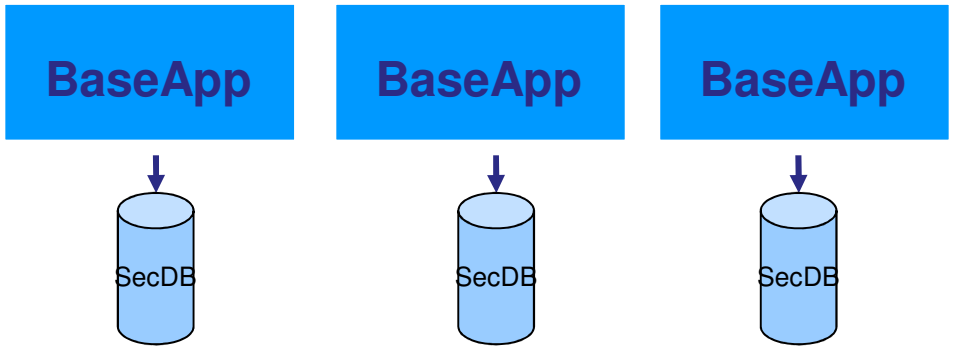
Secondary Databases

- Scalability feature
- DB is normally the first bottleneck
- Vast majority of DB operations are writes

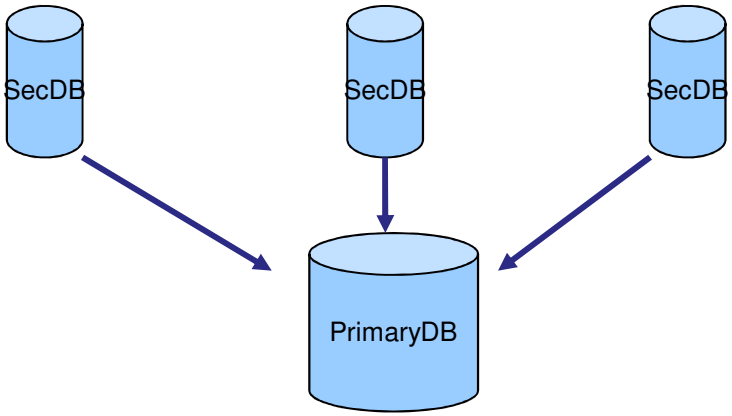
bigWORLD™ TECHNOLOGY

Secondary Databases

- BaseApp write request goes to local Secondary DB.



  - Secondary DBs are consolidated on server shutdown (or next startup).



bigWORLD™ TECHNOLOGY

Using the Server

- WebConsole
  - Web-based monitoring interface
    - ClusterControl – server process administration; view of processes, users, machines
    - LogViewer – view and search log output
    - StatGrapher – view graphs of process and machine statistics collected by StatLogger
    - PythonConsole – connect to running processes' Python interpreter
    - Commands – execute predefined scripts
- ControlCluster (CLI)
  - Swiss army knife tool for starting/stopping servers, cluster queries, watcher queries
- SpaceViewer
  - For monitoring spaces, cells and their entities for a running server
  - Cross-platform (Win32 & Linux)
- For more details on these and other tools, see Server Operations Guide

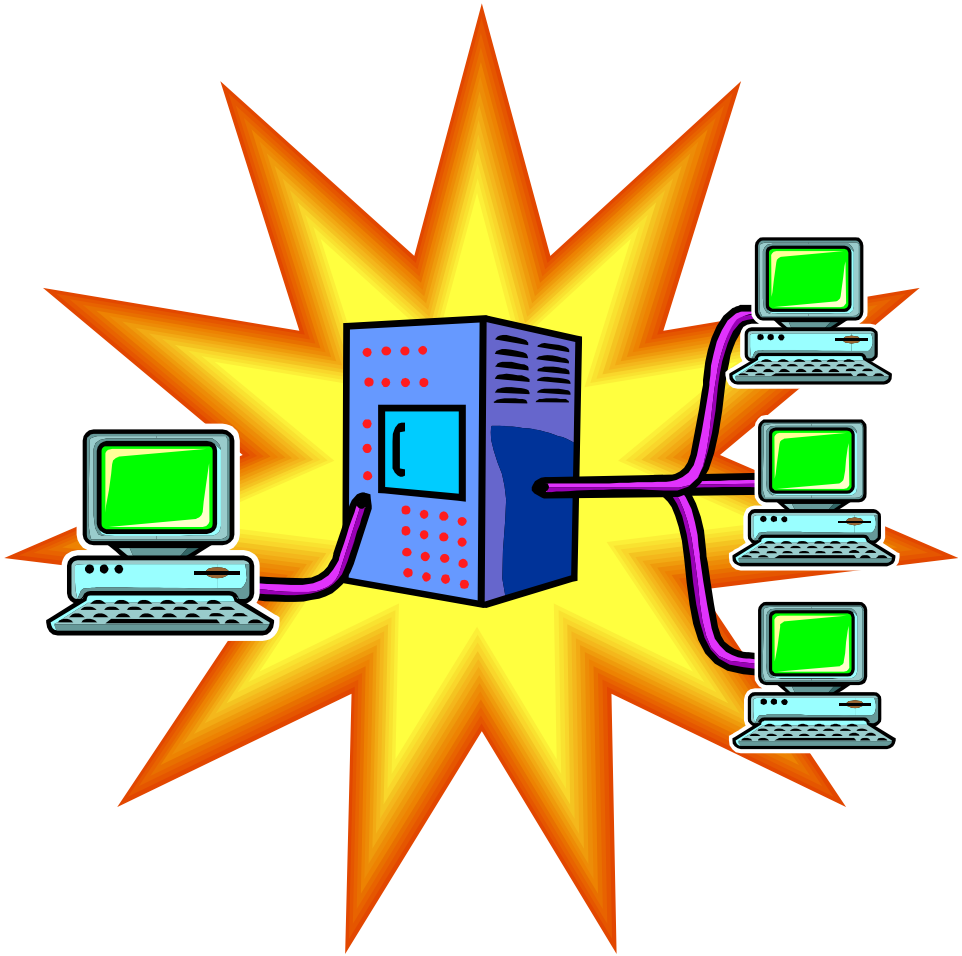
bigWORLD™ TECHNOLOGY

Interacting with the Server

- **runScript** to alter content
- **telnet** to query Python at run-time using **control_cluster.py**
  - **control_cluster.py pyconsole [process] [--entity {cell|base}:{id}]**
- Use BigWorld Python Interface to interact (in this case on BaseApp)
  - **>>> g = BigWorld.createBase("Guard", position = (2, 3, 5))**
    - → **Guard.Guard** instance at ***<address>***
    - → **EntityName**
    - → Default attribute values
  - **>>> g.id**
  - **1234**
  - On CellApp:
    - **>>> g = BigWorld.entities[1234]**
    - **>>> g.position**
    - (2.000000, 3.000000, 5.000000)
      - Note that **y** is vertical height in BigWorld
    - **>>> dir(g)**
      - Many built-in properties, methods, some from EntityExtras
      - Also entity-specific properties and methods from entity definition
    - **>>> g.destroy()**
      - **Base.onLoseCell()** is called on the BaseApp, and so entity base can destroy self

bigWORLD™ TECHNOLOGY

Session 7
BigWorld Profiling and Stress Testing



bigWORLD™ TECHNOLOGY

Stress Testing Using Bots

- Simulates many players
- Highly recommended prior to large scale player testing
- No terrain loading
- No navigation

Bot Scripting

- Each entity type requires a Python script in **`res/scripts/bot`**
  - Bot scripts implement the client part of the entity
    - But simply copying the client scripts usually doesn't work since client scripts reference many UI and 3D objects that do not exist in bots
  - For most entity types, implementing an empty class will do
  - For account and player entities, custom scripts are needed to log in and simulate a player
  - A.I. programming to simulate a player can be extensive

Adding bots

- Run bot process and use WebConsole to add bots
- Alternatively use **`bigworld/tools/server/bot_op.py`** to automatically distribute bots amongst many bot processes

Profiling Tools

- ControlCluster has many CLI commands to profile various aspects of a running server.
- StatGrapher can show you the load of each server component.
- Pay attention to profiling early, ensure your internal and external bandwidth capacity is not being exceeded by expensive method calls.
  - Also recommend having your own network hardware performance monitoring, to identify when the network is saturated.
- Use profiling data to guide optimisations.

bigWORLD™ TECHNOLOGY

Profiling Commands

- eventprofile - Identify expensive method calls and property updates.
- mercuryprofile - Identify high traffic Mercury-level messages.
- pyprofile - Identify high CPU-time Python calls.
- cprofile - Identify high CPU-time engine C++ calls for CellApps.

bigWORLD™ TECHNOLOGY

Further Reading

- Server Overview has a detailed explanation of concepts
- Server Programming Guide is a guide for C++ programming and Python scripting
- Server Operations Guide is a guide for performing server software operations

Conclusion

- Client-side slides build on an understanding of server concepts
- Thanks for attending

bigWORLD™ TECHNOLOGY