



Esri International Developer Summit  
Palm Springs, CA

# Effective Geodatabase Programming

Colin Zwicker

Erik Hoel

# Purpose

- Cover material that is important to master in order for you to be an effective Geodatabase programmer
- Provide additional insight regarding how we (the Geodatabase development team) conceptualize the architecture
- General focus areas:
  - Best practices
  - Common questions
  - New programming patterns

# Assumptions

- Good working knowledge of the Geodatabase and experience in programming against the GDB API
- Code examples will use Java, C#, or C++
- Lots of technical content, very little time (60 minutes)
  - Please save questions for the end
- Slides intended to be later used by you as a reference
  - Extra bonus material in the downloadable version

# Outline

- Unique Instancing of Objects
- Cursors
- Caching Geodatabase Data
  - Spatial and Schema Caches
- Schema Locks
- Top Developer Mistakes

# Outline

- Unique Instancing of Objects
- Cursors
- Caching Geodatabase Data
  - Spatial and Schema Caches
- Schema Locks
- Top Developer Mistakes

# Unique Instancing of Objects

- Geodatabase objects that will have at most one instance instantiated
  - Similar to COM singletons except they are not cocreateable
  - Examples:
    - Datasets (tables, feature classes, feature datasets)
    - Workspaces and versions
- Regardless of the API that handed out the reference, it is the same reference
- Changes to the object affect all holders of the reference

# Unique Instancing of Objects

```
public void run(Workspace workspace) throws IOException, AutomationException {  
  
    IVersionedWorkspace vw = new IVersionedWorkspaceProxy(workspace);  
    IVersion default1 = vw.getDefaultVersion();  
    IVersion default2 = vw.findVersion("DEFAULT");  
  
    System.out.println(default2.equals(default1)); <<- true  
}
```

```
public void run(Workspace workspace) throws IOException, AutomationException {  
  
    IFeatureClass fc1 = workspace.openFeatureClass("StateBoundaries");  
    IFeatureClass fc2 = workspace.openFeatureClass("gdb.DBO.StateBoundaries");  
  
    IFeatureDataset fd3 = workspace.openFeatureDataset("UnitedStates");  
    IFeatureClassContainer fcc = new IFeatureClassContainerProxy(fd3);  
    IFeatureClass fc3 = fcc.getClassByName("StateBoundaries");  
  
    System.out.println(fc1.equals(fc2)); <<- true  
    System.out.println(fc1.equals(fc3)); <<- true  
    System.out.println(fc2.equals(fc3)); <<- true  
}
```

# Unique Instancing of Objects

```
public void run(Workspace workspace) throws IOException, AutomationException {  
  
    IVersionedWorkspace vw = new IVersionedWorkspaceProxy(workspace);  
    IVersion default1 = vw.getDefaultVersion();  
    IVersion default2 = vw.findVersion("DEFAULT");  
  
    System.out.println(default2.equals(default1)); <<- true  
}
```

```
public void run(Workspace workspace) throws IOException, AutomationException {  
  
    IFeatureClass fc1 = workspace.openFeatureClass("StateBoundaries");  
    IFeatureClass fc2 = workspace.openFeatureClass("gdb.DBO.StateBoundaries");  
  
    IFeatureDataset fd3 = workspace.openFeatureDataset("UnitedStates");  
    IFeatureClassContainer fcc = new IFeatureClassContainerProxy(fd3);  
    IFeatureClass fc3 = fcc.getClassByName("StateBoundaries");  
  
    System.out.println(fc1.equals(fc2)); <<- true  
    System.out.println(fc1.equals(fc3)); <<- true  
    System.out.println(fc2.equals(fc3)); <<- true  
}
```



# Unique Instancing of Objects

```
public void run(Workspace workspace) throws IOException, AutomationException {  
  
    IVersionedWorkspace vw = new IVersionedWorkspaceProxy(workspace);  
    IVersion default1 = vw.getDefaultVersion();  
    IVersion default2 = vw.findVersion("DEFAULT");  
  
    System.out.println(default2.equals(default1)); <<- true  
}
```

```
public void run(Workspace workspace) throws IOException, AutomationException {  
  
    IFeatureClass fc1 = workspace.openFeatureClass("StateBoundaries");  
    IFeatureClass fc2 = workspace.openFeatureClass("gdb.DBO.StateBoundaries");  
  
    IFeatureDataset fd3 = workspace.openFeatureDataset("UnitedStates");  
    IFeatureClassContainer fcc = new IFeatureClassContainerProxy(fd3);  
    IFeatureClass fc3 = fcc.getClassByName("StateBoundaries");  
  
    System.out.println(fc1.equals(fc2)); <<- true  
    System.out.println(fc1.equals(fc3)); <<- true  
    System.out.println(fc2.equals(fc3)); <<- true  
}
```

# Unique Instancing of Objects – Special Case

Rows / features uniquely instanced only within an edit session

```
public void run(Workspace workspace) throws IOException, AutomationException {

    IFeatureClass fc1 = workspace.openFeatureClass("StateBoundaries");

    IFeature f1 = fc1.getFeature(1);
    IFeature f2 = fc1.getFeature(1);
    System.out.println(f2.equals(f1)); <<- false

    IMultiuserWorkspaceEdit workspaceEdit = new IMultiuserWorkspaceEditProxy(workspace);
    workspaceEdit.startMultiuserEditing(esriMultiuserEditMode.esriMESMNonVersioned);

    IFeature fe1 = fc1.getFeature(1);
    IFeature fe2 = fc1.getFeature(1);
    System.out.println(fe2.equals(fe1)); <<- true

    workspace.stopEditing(false);
}
```

# Unique Instancing of Objects – Special Case

Rows / features uniquely instanced only within an edit session

```
public void run(Workspace workspace) throws IOException, AutomationException {  
  
    IFeatureClass fc1 = workspace.openFeatureClass("StateBoundaries");  
  
    IFeature f1 = fc1.getFeature(1);  
    IFeature f2 = fc1.getFeature(1);  
    System.out.println(f2.equals(f1)); <<- false  
  
    IMultiuserWorkspaceEdit workspaceEdit = new IMultiuserWorkspaceEditProxy(workspace);  
    workspaceEdit.startMultiuserEditing(esriMultiuserEditMode.esriMESMNonVersioned);  
  
    IFeature fe1 = fc1.getFeature(1);  
    IFeature fe2 = fc1.getFeature(1);  
    System.out.println(fe2.equals(fe1)); <<- true  
  
    workspace.stopEditing(false);  
}
```

# Unique Instancing of Objects – Special Case

Rows / features uniquely instanced only within an edit session

```
public void run(Workspace workspace) throws IOException, AutomationException {  
  
    IFeatureClass fc1 = workspace.openFeatureClass("StateBoundaries");  
  
    IFeature f1 = fc1.getFeature(1);  
    IFeature f2 = fc1.getFeature(1);  
    System.out.println(f2.equals(f1)); <<- false  
  
    IMultiuserWorkspaceEdit workspaceEdit = new IMultiuserWorkspaceEditProxy(workspace);  
    workspaceEdit.startMultiuserEditing(esriMultiuserEditMode.esriMESMNonVersioned);  
  
    IFeature fe1 = fc1.getFeature(1);  
    IFeature fe2 = fc1.getFeature(1);  
    System.out.println(fe2.equals(fe1)); <<- true  
  
    workspace.stopEditing(false);  
}
```

# Outline

- Unique Instancing of Objects
- Cursors
- Caching Geodatabase Data
  - Spatial and Schema Caches
- Schema Locks
- Top Developer Mistakes

# Cursor Types

- Three class cursors
  - Search (general query cursor)
  - Update (positioned update cursor)
  - Insert (bulk inserts)
- One QueryDef cursor
  - Defined query (e.g., `IQueryDef.evaluate`)
- What's the difference?
  - Rows created by class cursors are bound to the class which created the cursor
  - Rows created by a QueryDef cursor are not bound to a class

## Cursor Types - Example

```
public void run(Workspace workspace)
    throws IOException, AutomationException {

    ITable testTable = workspace.openTable("parcels");
    ICursor classCursor = testTable.ITable_search(null, true);
    IRow classRow = classCursor.nextRow();
    ITable rowTable = classRow.getTable();
    System.out.println(rowTable.equals testTable)); <<-- PRINTS true

    IQueryDef qdef = workspace.createQueryDef();
    qdef.setTables("parcels");
    ICursor queryCursor = qdef.evaluate();
    IRow queryRow = queryCursor.nextRow();

    try {
        ITable queryTable = queryRow.getTable();
    }
    catch (AutomationException ax) {
        System.out.println(ax.getMessage()); <<-- PRINTS AutomationException Message
    }
}
```

## Cursor Types - Example

```
public void run(Workspace workspace)
    throws IOException, AutomationException {

    ITable testTable = workspace.openTable("parcels");
    ICursor classCursor = testTable.ITable_search(null, true);
    IRow classRow = classCursor.nextRow();
    ITable rowTable = classRow.getTable();
    System.out.println(rowTable.equals testTable)); <<-- PRINTS true

    IQueryDef qdef = workspace.createQueryDef();
    qdef.setTables("parcels");
    ICursor queryCursor = qdef.evaluate();
    IRow queryRow = queryCursor.nextRow();

    try {
        ITable queryTable = queryRow.getTable();
    }
    catch (AutomationException ax) {
        System.out.println(ax.getMessage()); <<-- PRINTS AutomationException Message
    }
}
```



## Cursor Types - Example

```
public void run(Workspace workspace)
    throws IOException, AutomationException {

    ITable testTable = workspace.openTable("parcels");
    ICursor classCursor = testTable.ITable_search(null, true);
    IRow classRow = classCursor.nextRow();
    ITable rowTable = classRow.getTable();
    System.out.println(rowTable.equals testTable)); <<-- PRINTS true

    IQueryDef qdef = workspace.createQueryDef();
    qdef.setTables("parcels");
    ICursor queryCursor = qdef.evaluate();
    IRow queryRow = queryCursor.nextRow();

    try {
        ITable queryTable = queryRow.getTable();
    }
    catch (AutomationException ax) {
        System.out.println(ax.getMessage()); <<-- PRINTS AutomationException Message
    }
}
```

# Search Cursors

- All purpose class-based query API
  - Common APIs which create search cursors
    - `ITable.Search`, `GetRow`, `GetRows`, `ISelectionSet.Search`
- When in an edit session, the query may be satisfied by a cache (spatial cache, object pool)
- When used within an edit session will only flush the class' cached rows
  - Could result in a database write
- Resulting rows can be modified
  - `Store` / `Delete` supported

# Update Cursors

- Positional update cursor
  - NextRow->UpdateRow ... NextRow->UpdateRow
  - Update the row at the current cursor position
- Common APIs which create update cursors
  - ITable.Update, ISelectionSet2.Update
- Query is never satisfied by a cache
- Use within an edit session will only flush the class' cached rows
- Resulting rows can be modified using ICursor.UpdateRow Or ICursor.DeleteRow
  - Should not be combined with Store and Delete

# Update Cursors

- If the class supports `Store` events
  - An internal search cursor is created and `UpdateRow` and `DeleteRow` become equivalent to `Row.Store` and `Row.Delete`
- As a developer how do you know if the class supports `Store` events?
  - Non-simple feature types (! `esri.FTSimple`)
  - Class participates in geometric network or relationships that require messaging
  - Custom features (`ObjectClassExtension` overrides)
- `UpdateRow` method behaviors
  - Error if called with a row that was not retrieved from the update cursor
  - Error if called with a row not at the current position

## Update Cursors – Best Practices

- Do not use an update cursor across edit operations when editing
  - Very important when editing versioned data
  - Will throw an error
- Why? Because Update cursors point to a specific state in the database
  - This state could get trimmed during a edit sessions
  - Updates could be lost!
- Best Practice – Always scope the cursor to the edit operation
  - If you start a new edit operation you should get a new update cursor

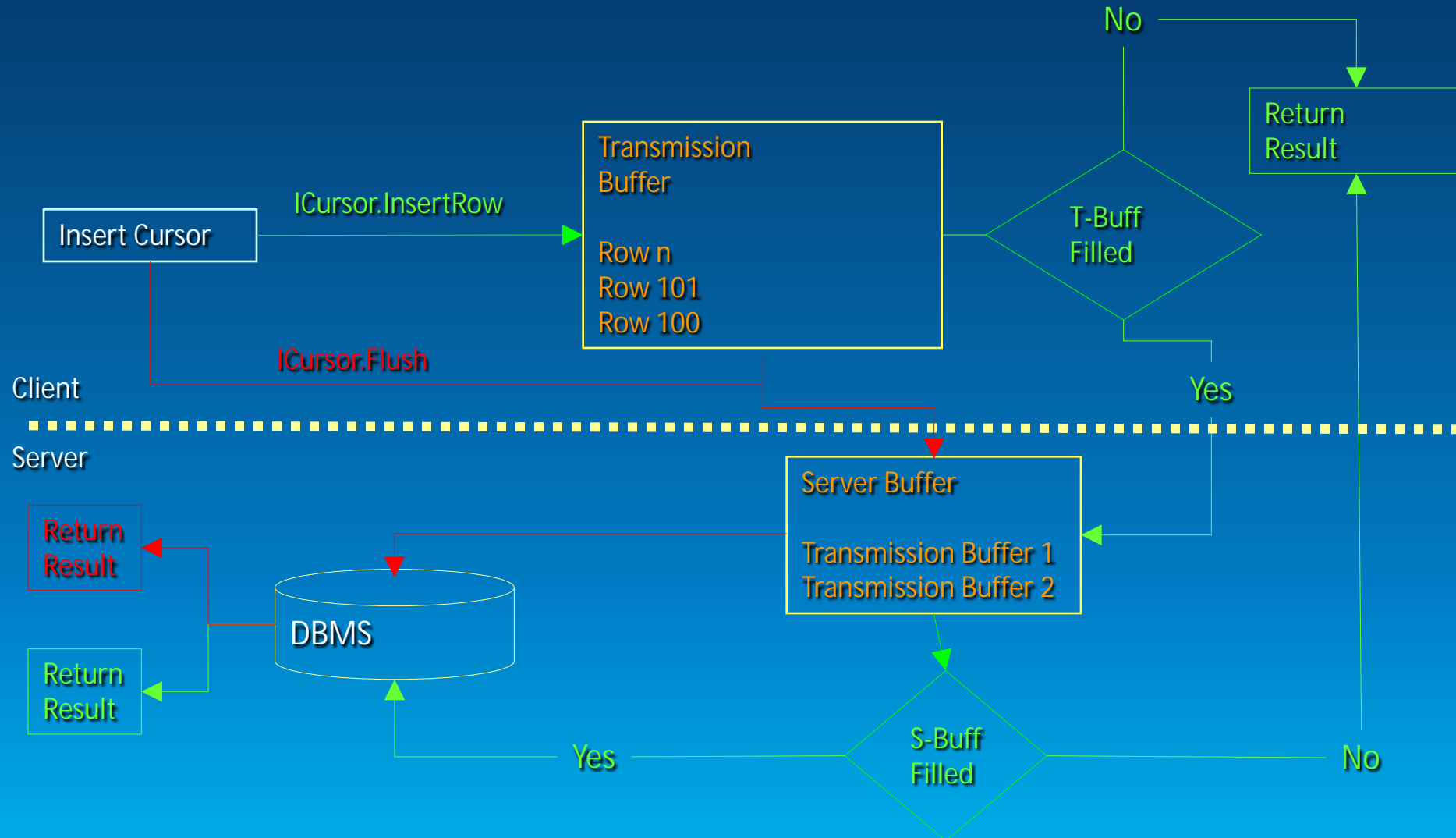
# Insert Cursors

- Primary use is for bulk inserts
- API which creates an insert cursor
  - `ITable.Insert`
- Best performance when using buffering and proper flushing
- If the class supports `Store` events, the search cursor will be simulated
  - `InsertRow` becomes `CreateRow` and `Store`

## Insert Cursors - Buffering

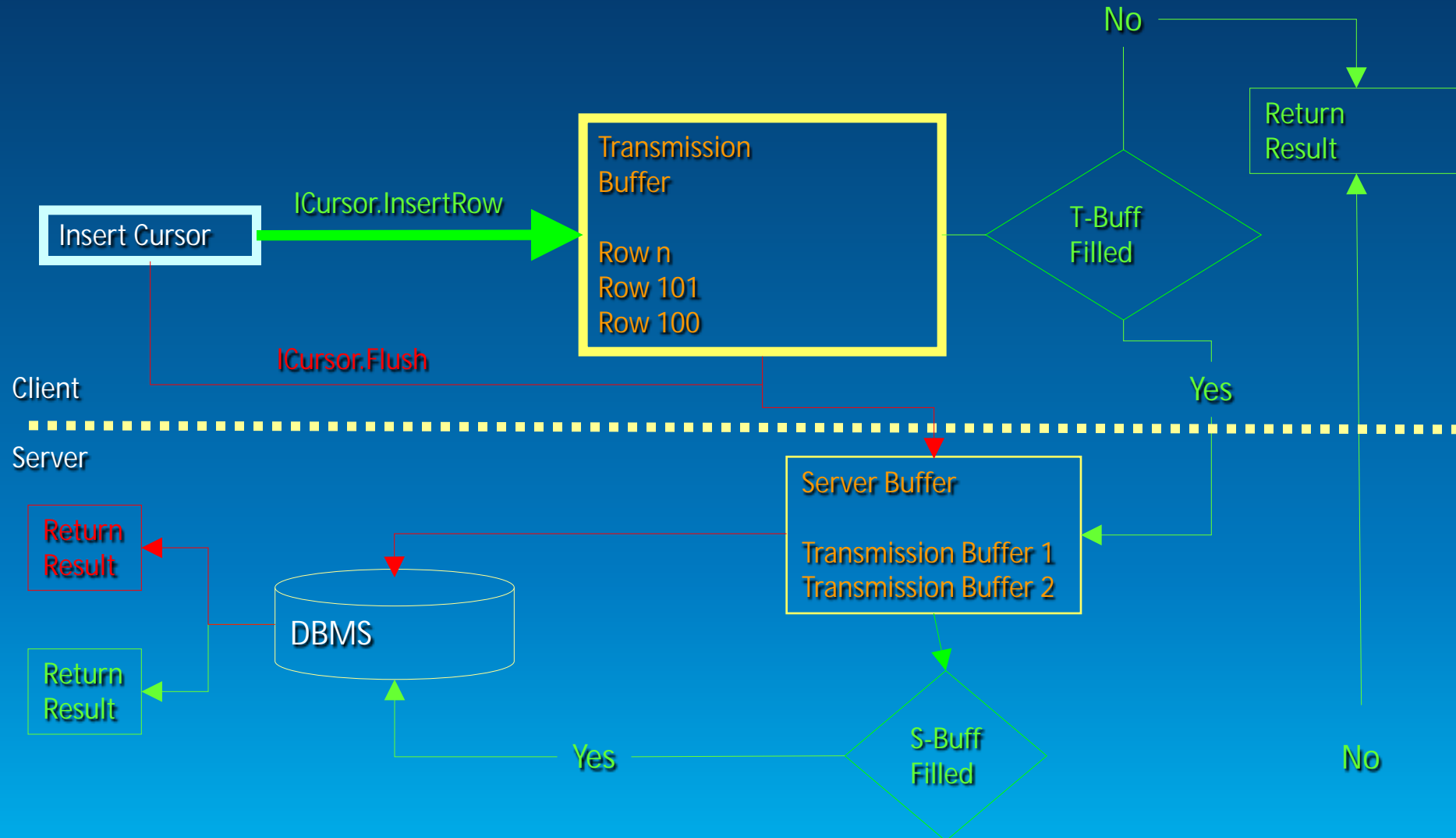
- Call the `ITable.Insert` method with the argument “useBuffering” set to “true”
- Periodically call `Flush`
  - 1000 rows per flush is a good starting number
  - The higher the flush interval the less network traffic (but more re-insertion)
- Try to ensure that the class has no spatial cache – extra processing is required to keep the cache in synch
- Crucial that calls to `InsertRow` and `Flush` have the proper error handling since both can result in rows written to the database

# Use of Buffering with Enterprise Geodatabases

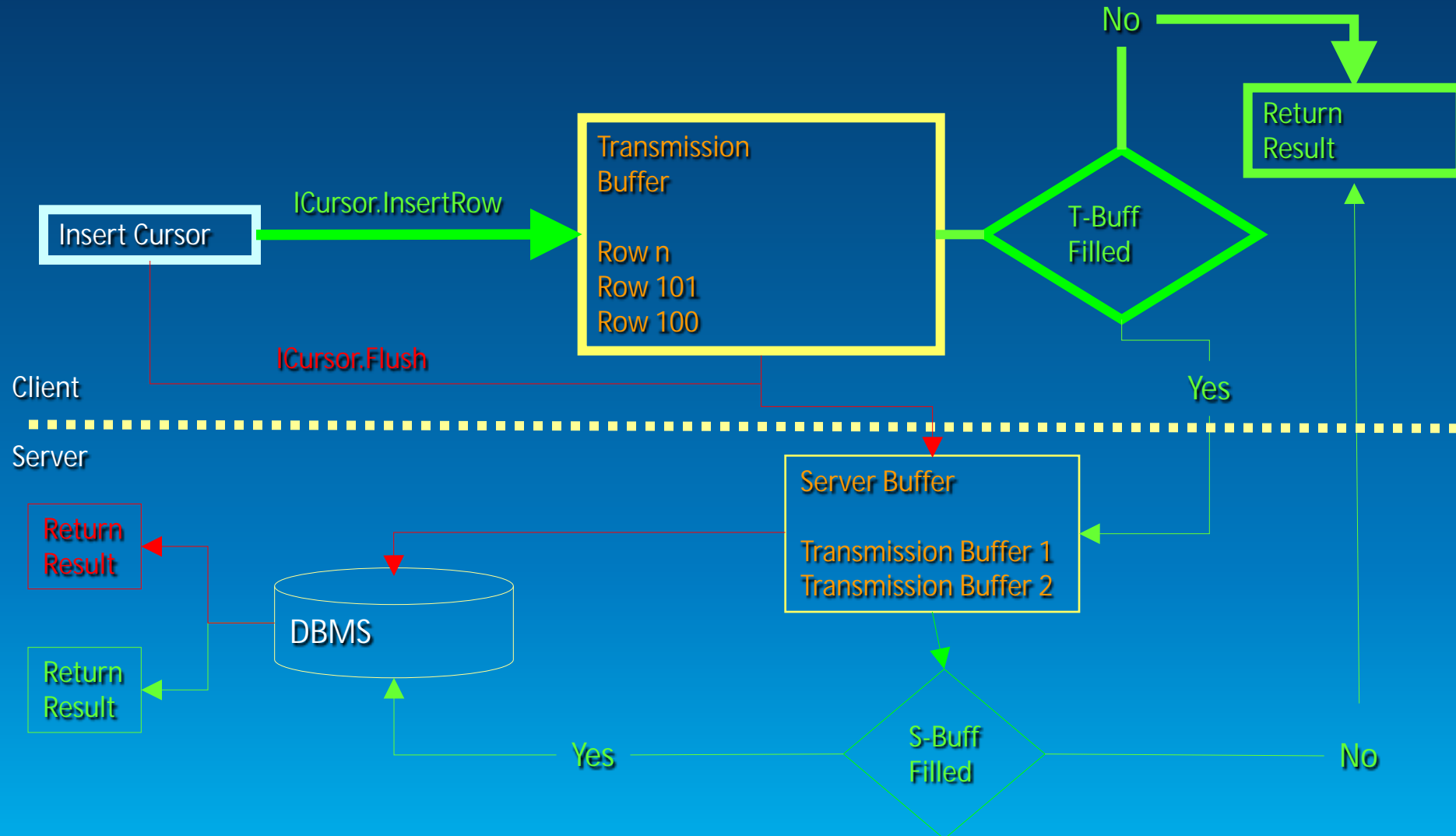




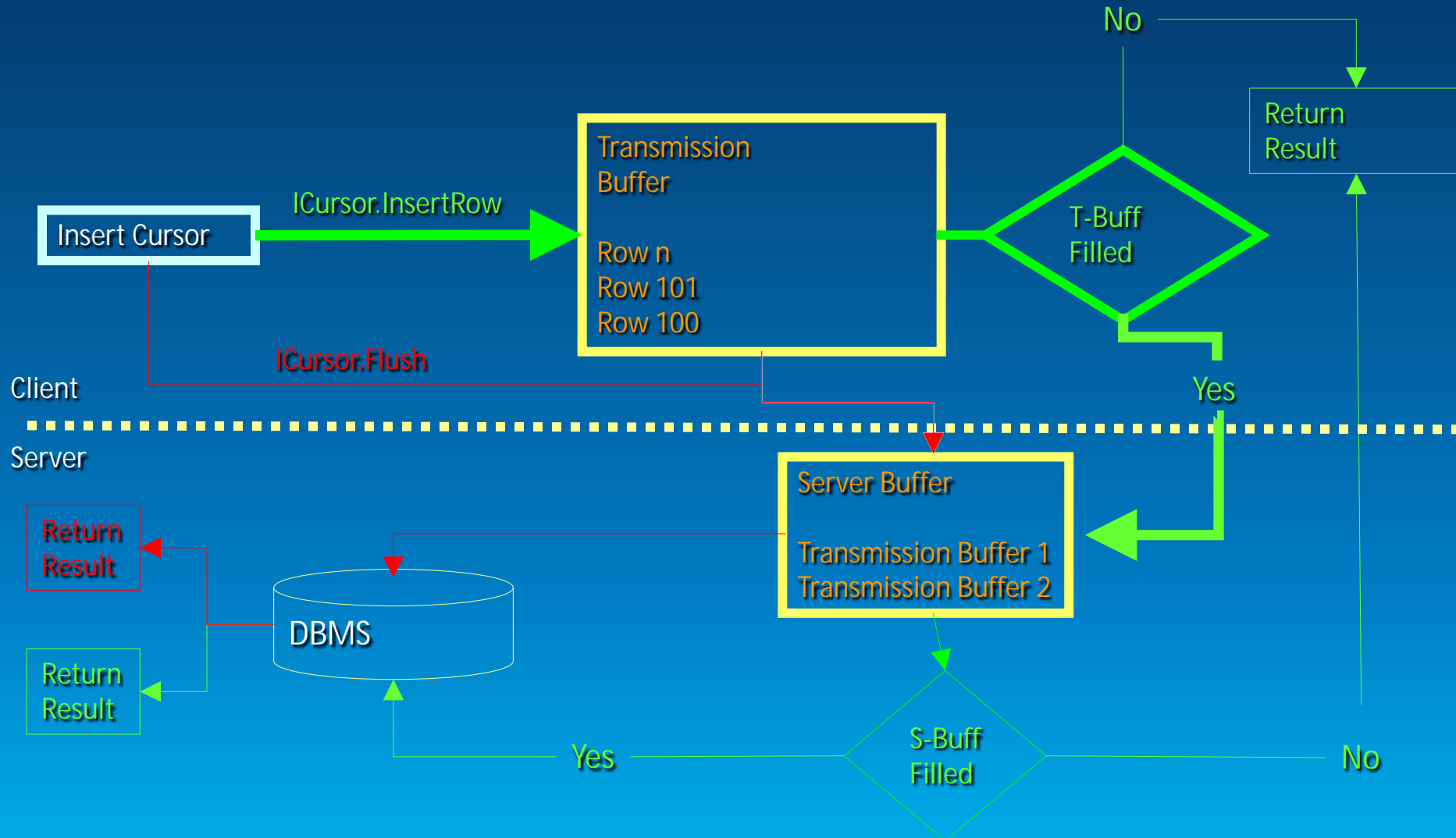
# Use of Buffering with Enterprise Geodatabases



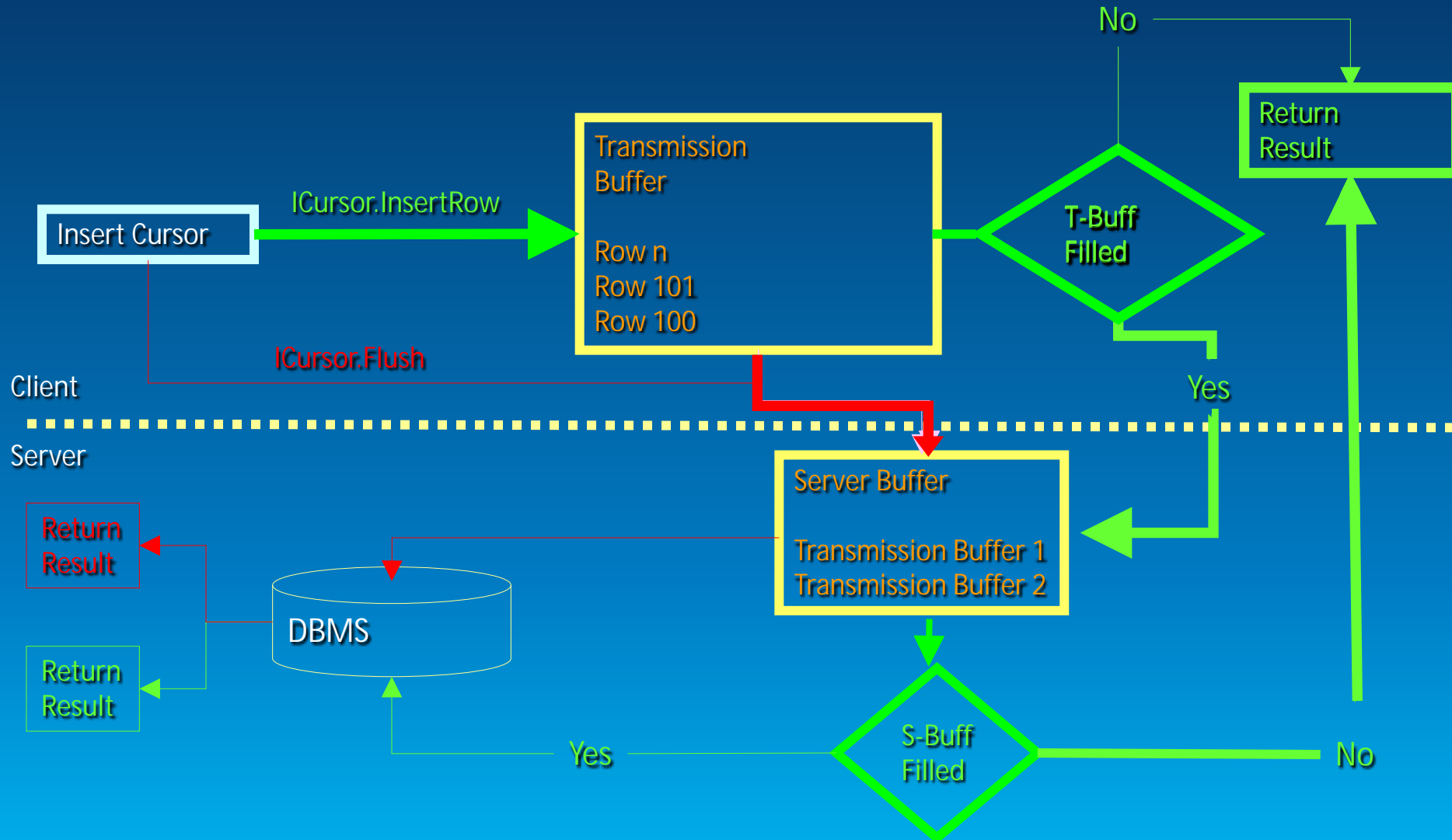
# Use of Buffering with Enterprise Geodatabases



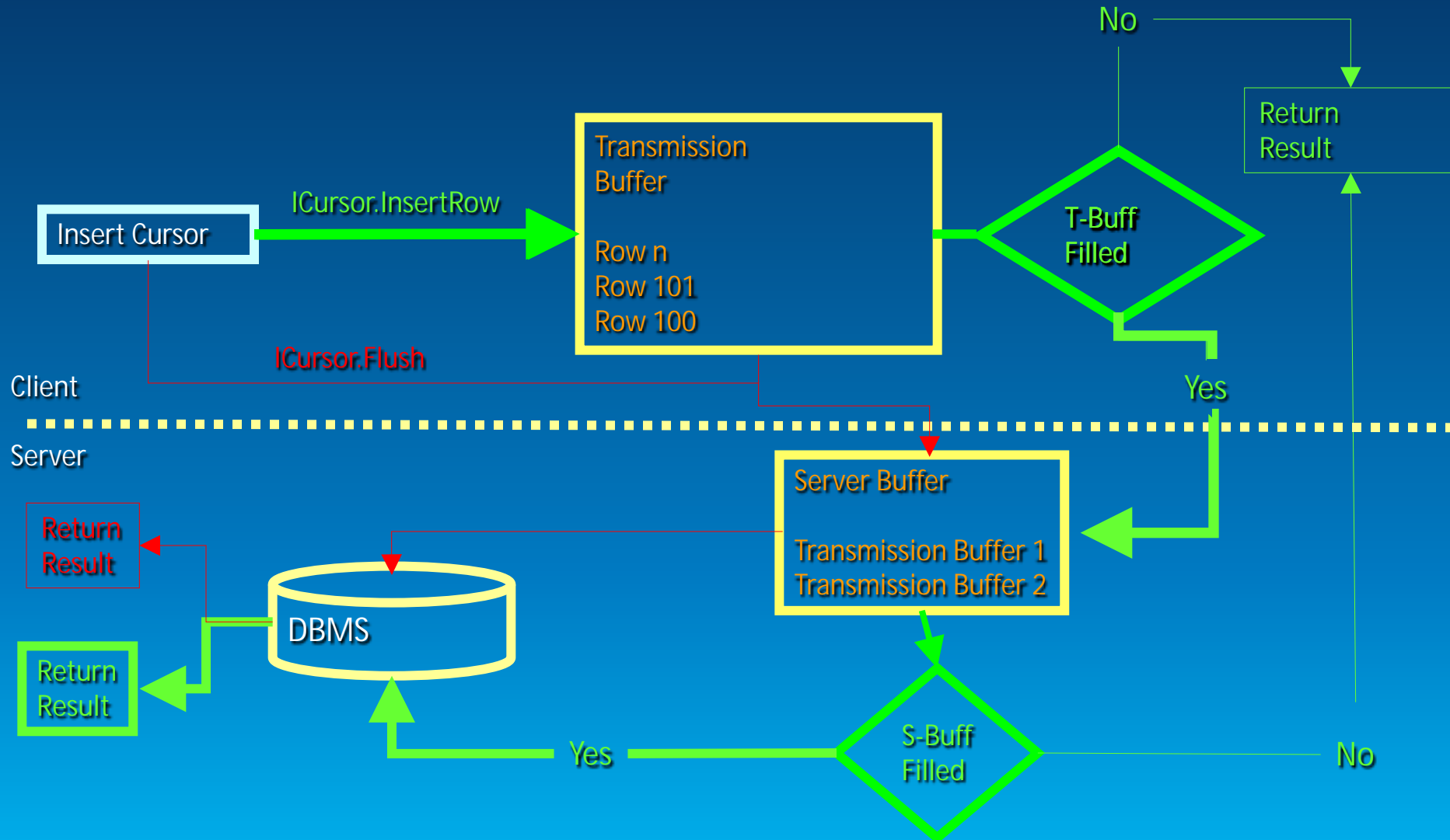
# Use of Buffering with Enterprise Geodatabases



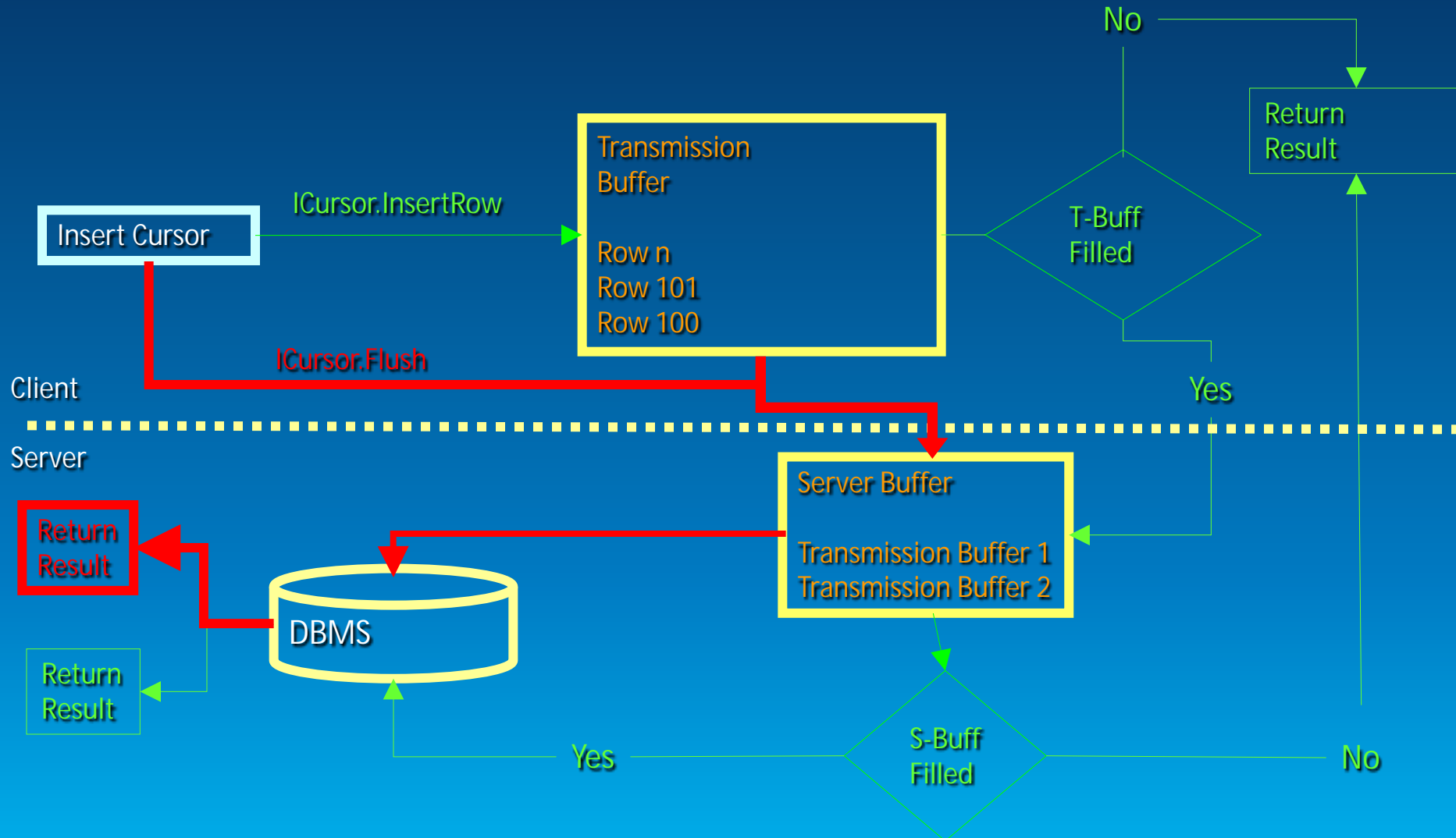
# Use of Buffering with Enterprise Geodatabases



# Use of Buffering with Enterprise Geodatabases



# Use of Buffering with Enterprise Geodatabases



# QueryDef Cursors

- Executes user defined query (not bound to class)
- QueryDef cursors always bypass any row cache held by the class / workspace
- IQueryDef. Evaluate within an edit session will cause all cached rows to be flushed
- Rows from QueryDef cursors do not support APIs which modify the row
  - Store not supported
  - Delete not supported

## Recycling Cursors – What are they?

- A **recycling cursor** is a cursor that does not create a new client side row object for each row retrieved from the database
- Internal data structures and objects will be re-used
  - Memory
  - Object instances (e.g., Geometry)
- Geodatabase APIs which support the creation of recycling cursors have a boolean method argument
  - `recycling = true` creates a "recycling cursor"



# Recycling Cursors

- Interfaces with methods that can create recycling cursors
- `ITable` / `IFeatureClass`
  - `GetRows` / `GetFeatures`
  - `Search`
  - `Update`
- `ISet` / `ISet2`
  - `Search`
  - `Update`
- `ITableWrite`
  - `UpdateRows`

## Recycling Cursors - Example

- Calls to `NextRow` result in the same row instance

```
public void run(Workspace workspace)
    throws IOException, AutomationException {

    ITable testTable = workspace.openTable("road_node");
    ICursor cursor = testTable.ITable_search(null, true); <<-- recycling

    IRow firstRow = cursor.nextRow();
    IRow secondRow = cursor.nextRow();

    if (firstRow != null && secondRow != null) {
        System.out.println(secondRow.equals(firstRow)); <<-- PRINTS true
    }
}
```

## Recycling Cursors - Example

- Calls to `NextRow` result in the same row instance

```
public void run(Workspace workspace)
    throws IOException, AutomationException {

    ITable testTable = workspace.openTable("road_node");
    ICursor cursor = testTable.ITable_search(null, true); <<-- recycling

    IRow firstRow = cursor.nextRow();
    IRow secondRow = cursor.nextRow();

    if (firstRow != null && secondRow != null) {
        System.out.println(secondRow.equals(firstRow)); <<-- PRINTS true
    }
}
```

## Recycling Cursors - Example

```
public void run(Workspace workspace)
    throws IOException, AutomationException {

    ITable testTable = workspace.openTable("road_node");
    ICursor cursor = testTable.ITable_search(null, true); <!-- recycling

    int gidx = cursor.findField("location");

    IRow firstRow = cursor.nextRow();
    IGeometry firstGeometry = new IGeometryProxy(firstRow.getValue(gidx));

    IRow secondRow = cursor.nextRow();
    IGeometry secondGeometry = new IGeometryProxy(secondRow.getValue(gidx));

    if (firstGeometry != null && secondGeometry != null) {
        System.out.println(secondGeometry.equals(firstGeometry)); <!-- true
    }
}
```

## Recycling Cursors - Example

```
public void run(Workspace workspace)
    throws IOException, AutomationException {

    ITable testTable = workspace.openTable("road_node");
    ICursor cursor = testTable.ITable_search(null, true); <<-- recycling

    int gidx = cursor.findField("location");

    IRow firstRow = cursor.nextRow();
    IGeometry firstGeometry = new IGeometryProxy(firstRow.getValue(gidx));

    IRow secondRow = cursor.nextRow();
    IGeometry secondGeometry = new IGeometryProxy(secondRow.getValue(gidx));

    if (firstGeometry != null && secondGeometry != null) {
        System.out.println(secondGeometry.equals(firstGeometry)); <<-- true
    }
}
```

## Recycling Cursors – **When they should be used**

- When you don't need to persist a reference to a row
- Don't pass it around
  - Isolate use of references to the local method which created the recycling cursor to minimize potential bugs
  - Do not pass the references around as some other method may decide to hold it
- **Never directly edit a recycled row**
- Proper use within an edit session can dramatically reduce resource consumption

## Recycling Cursor - Reduced resource consumption

```
public void run(Workspace workspace, boolean recycling)
    throws IOException, AutomationException {

    ITable testTable = workspace.openTable("road_node_small");
    IMultiuserWorkspaceEdit workspaceEdit = new
    IMultiuserWorkspaceEditProxy(workspace);
    workspaceEdit.startMultiuserEditing(esriMESMNonVersioned);

    ICursor cursor = testTable.ITable_search(null, recycling);
    IRow row = cursor.nextRow();
    while (row != null) {
        System.out.println("OID: " + row.getOID());
        row = cursor.nextRow();
    }
    workspace.stopEditing(false); <<-- BREAKPOINT
}
```

- Test data: 50,000 rows with ~75 fields
  - recycling = true     ~60MB memory
  - recycling = false    ~185MB memory

## Non-Recycling Cursors - **What are they?**

- A cursor that creates a new client side row object for each row retrieved from the database
- New internal data structures and objects will be created for each row
  - Memory
  - Object instances (e.g., Geometry)
- Geodatabase APIs which support the creation of non-recycling cursors have a boolean method argument
  - `recycling = false` creates a "non-recycling cursor"



## Non-Recycling Cursors - **When to use?**

- When references to the current row and its values need to be persisted
- Commonly used to cache sets of rows (long lived references)
- Some Geodatabase APIs require sets of rows – should be retrieved as non-recycled rows
- **Always edit non-recycled rows**

# Cursor FAQs

## Question:

When should I release a reference to a cursor?

## Answer:

Do not hold cursor references if they are not needed.

- Release ASAP after fetching is completed
- Alternatively, release after application is done with the cursor

## Cursor FAQs

### Question:

If I need to use a cursor inside an edit session, where should I create the cursor?

### Answer:

Inside the edit session, scope to edit operation

```
// AVOID THIS PATTERN
workspace.startEditOperation();
ICursor cursor = testTable.ITable_search(null, true);
IRow row = cursor.nextRow();
workspace.stopEditOperation();

workspace.startEditOperation();
    row = cursor.nextRow();
    System.out.println(row.getOID());
workspace.stopEditOperation();
```

## Cursor FAQs

### Question:

Should I use a search cursor to update rows?

### Answer:

Yes. In fact, using search cursors within an edit session is the recommended way to update rows (see next slide)

# Cursor FAQs

If I am editing, what type of cursor should I use?

	ArcMap	Engine Simple Data	Engine Complex Data
Inside Edit Session	Search	Search	Search
Outside Edit Session	Search	Update / ITableWrite	Search

- ArcMap – possibility that an active cache can satisfy the query and no DBMS query is required
- Engine Simple Data
  - Inside ES – take advantage of batched updates for edit operations
  - Outside ES – performance, can handle errors on a per row basis
- Engine Complex Data – the system will emulate Search regardless

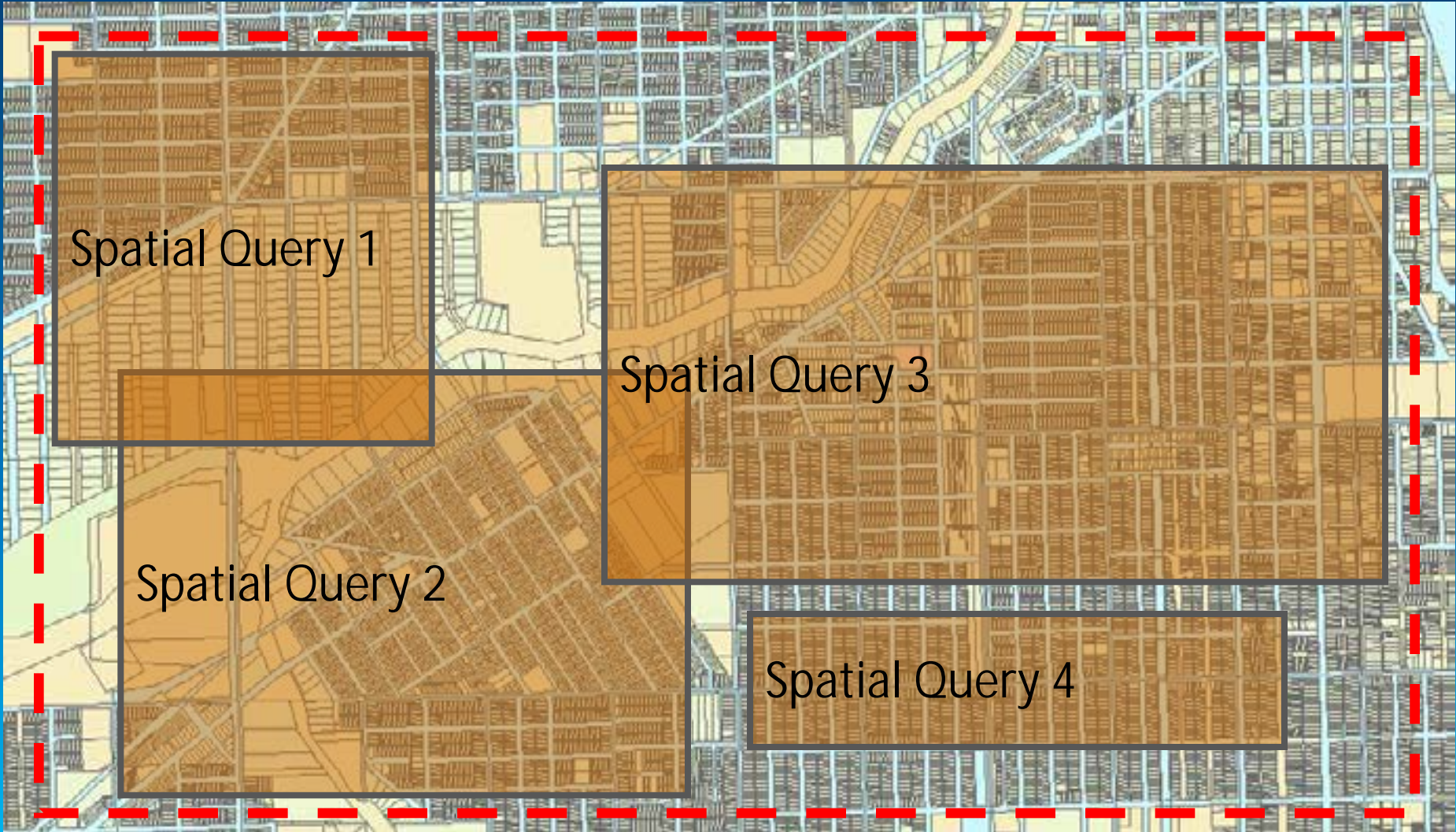
# Outline

- Unique Instancing of Objects
- Cursors
- Caching Geodatabase Data
  - Spatial and Schema Caches
- Schema Locks
- Top Developer Mistakes

# What is the Spatial Cache?

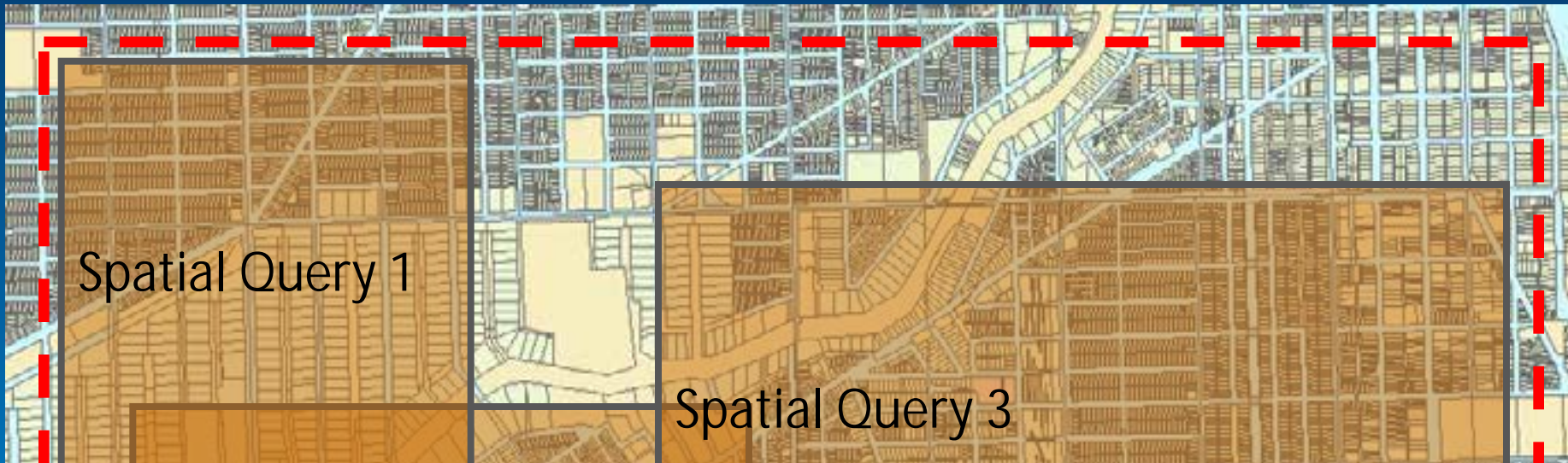
- Client side caching of feature values over a given spatial extent (Map Cache in ArcMap)
- Can speed up queries
  - Reduces roundtrips to the database
- `ISpatialCacheManager2`
  - `FillCache` / `EmptyCache`
  - `CacheExtent`
- When to use?
  - If making many spatial queries within a common extent

# Using the Spatial Cache





# Using the Spatial Cache



The image shows an aerial photograph of a city grid. Two overlapping rectangular areas are highlighted with solid black borders and labeled 'Spatial Query 1' and 'Spatial Query 3'. The area covered by both queries is shaded in a darker brown color. A red dashed border surrounds the entire map area.

```
//open feature class(es) before the cache is activated
IFeatureClass featureClass = fWorkspace.OpenFeatureClass("ParcelClass");

ISpatialCacheManager spCacheManager = (ISpatialCacheManager)fWorkspace;
spCacheManager.FillCache(queryEnvelope);

if (spCacheManager.CacheIsFull)
{
    //execute multiple spatial queries within the active cache extent
}
```

# What is the Schema Cache?

- A cached snapshot of the geodatabase schema
  - Used by ArcGIS (opening Map Documents, Reconcile)
  - Requires a static data model
    - GDB schema changes will not be reflected in the cache
- Can improve performance when opening datasets by reducing database round trips
- APIs to access the schema cache
  - `IWorkspaceFactorySchemaCache`

## When to use the Schema Cache

- Beneficial when working with large **static** data models
  - Tables, fields, domains, subtypes, and relationships are well defined and will not change
- If the application opens and uses many classes
  - These should be opened at the start of the application and references maintained throughout the lifetime of the application

# How to use the Schema Cache

- Enable schema cache before tables are opened
  - Calls to `OpenFeatureClass`, `OpenTable`, `IName.Open` will be optimized
  - Can enable schema caching at the factory level
- Cache needs to be “fresh”
  - If in dynamic environments schema changes will not be visible until cache is refreshed.

```
if (sCache.IsSchemaCacheStale(workspace))  
    sCache.RefreshSchemaCache(workspace);
```

- Your responsibility to disable the cache
  - Must be disabled before releasing the workspace object that is cached

# Using Schema Caching

- Ⓔ Cast to `IWorkspaceFactorySchemaCache` from the workspace factory and open the workspace

1

```
IWorkspaceFactorySchemaCache sCache =  
(IWorkspaceFactorySchemaCache)workspaceFactory;  
  
IWorkspace workspace = workspaceFactory.Open(propset, 0);  
  
sCache.EnableSchemaCache(workspace);  
  
IFeatureWorkspace featureWorkspace = (IFeatureWorkspace)workspace;  
IFeatureClass parcel = featureWorkspace.OpenFeatureClass("ParcelClass");  
//...open all feature classes used in application  
  
sCache.DisableSchemaCache(workspace);
```

# Using Schema Caching

- Enable the schema cache on the workspace
  - Alternatively `EnableSchemaCaching` will enable caching on all workspace passed out by the factory

```
IWorkspaceFactorySchemaCache sCache =  
(IWorkspaceFactorySchemaCache)workspaceFactory;  
  
IWorkspace workspace = workspaceFactory.Open(propset, 0);  
  
2 sCache.EnableSchemaCache(workspace);  
  
IFeatureWorkspace featureWorkspace = (IFeatureWorkspace)workspace;  
IFeatureClass parcel = featureWorkspace.OpenFeatureClass("ParcelClass");  
//...open all feature classes used in application  
  
sCache.DisableSchemaCache(workspace);
```

# Using Schema Caching

Ž Open all feature classes used in application

Ÿ Largest benefit for stand alone feature classes and tables

```
IWorkspaceFactorySchemaCache sCache =  
(IWorkspaceFactorySchemaCache)workspaceFactory;  
  
IWorkspace workspace = workspaceFactory.Open(propset, 0);  
  
sCache.EnableSchemaCache(workspace);  
  
IFeatureWorkspace featureWorkspace = (IFeatureWorkspace)workspace;  
IFeatureClass parcel = featureWorkspace.OpenFeatureClass("ParcelClass");  
//...open all feature classes used in application  
  
sCache.DisableSchemaCache(workspace);
```

# Using Schema Caching

- Disable the schema cache after the classes have been opened

```
IWorkspaceFactorySchemaCache sCache =  
(IWorkspaceFactorySchemaCache)workspaceFactory;  
  
IWorkspace workspace = workspaceFactory.Open(propset, 0);  
  
sCache.EnableSchemaCache(workspace);  
  
IFeatureWorkspace featureWorkspace = (IFeatureWorkspace)workspace;  
IFeatureClass parcel = featureWorkspace.OpenFeatureClass("ParcelClass");  
//...open all feature classes used in application  
  
4 sCache.DisableSchemaCache(workspace);
```



## Using Schema Caching - Gotchas

- Stale schema cached
  - Geodatabase schema changes will not be visible until cache is refreshed

```
IWorkspaceFactorySchemaCache sCache =  
(IWorkspaceFactorySchemaCache)workspaceFactory;  
  
IWorkspace workspace = workspaceFactory.Open(propset, 0);  
  
sCache.EnableSchemaCache(workspace);  
  
IFeatureWorkspace featureWorkspace = (IFeatureWorkspace)workspace;  
IFeatureClass parcel = featureWorkspace.OpenFeatureClass("ParcelClass");  
//...open all feature classes used in application  
  
sCache.DisableSchemaCache(workspace);
```

# Outline

- Unique Instancing of Objects
- Cursors
- Caching Geodatabase Data
  - Spatial and Schema Caches
- **Schema Locks**
- Top Developer Mistakes

# Schema Locks

- Prevent clashes with other users when changing the geodatabase structure
- Exclusive and shared
  - ISchemaLock primarily used for establishing exclusive lock
  - Shared locks are applied when accessing the object
  - Promote a shared lock to an exclusive lock
  - Only one exclusive lock allowed
- Exclusive locks are not applied or removed automatically

## Schema Locks – When to use

- You must promote a shared lock to exclusive when performing schema modification such as:
  - Modifications to attribute domains; coded or range
  - Adding or deleting a field to a feature or object class
  - Associating a class extension with a feature class
  - Creating a topology, geometric network, network dataset, terrain, schematic dataset, representation or cadastral fabric on a set of feature classes
  - Any use of the `IClassSchemaEdit` interfaces
  - `IFeatureClassLoad.LoadOnlyMode`
  - Rebuilding spatial and attribute indexes

# Schema Locks

- Demote exclusive lock to shared lock when the modification is complete
  - Includes when errors are raised during the schema modification
- Keep your use of exclusive schema locks tight
  - Prevents clashes with other applications and users
- If your application keeps a reference to an object with an exclusive schema lock
  - You will need to handle the exclusive schema lock when the object is used

# Outline

- Unique Instancing of Objects
- Cursors
- Caching Geodatabase Data
  - Spatial and Schema Caches
- Schema Locks
- **Top Developer Mistakes**

## Top Mistakes – Misusing recycling cursors

- If recycling is enabled, a cursor only allocates memory for a single row regardless of how many rows are returned from the cursor
  - Performance benefits in terms of memory usage and running time, but has drawbacks for certain workflows
  - E.g., comparing the first two geometries from a feature cursor to see if they are equal

## Top Mistakes – Misusing recycling cursors

```
public static void RecyclingInappropriateExample(IFeatureClass featureClass, Boolean
    enableRecycling)
{
    using(ComReleaser comReleaser = new ComReleaser())
    {
        // Create a search cursor.
        IFeatureCursor featureCursor = featureClass.Search(null, enableRecycling);
        comReleaser.ManageLifetime(featureCursor);

        // Get the first two geometries and see if they intersect.
        IFeature feature1 = featureCursor.NextFeature();
        IFeature feature2 = featureCursor.NextFeature();
        IRelationalOperator relationalOperator = (IRelationalOperator)feature1.Shape;
        Boolean geometriesEqual = relationalOperator.Equals(feature2.Shape);
        Console.WriteLine("Geometries are equal: {0}", geometriesEqual);
    }
}
```



## Top Mistakes – Calling FindField in a loop

- Relying on FindField as opposed to hard-coded field positions is a good practice but overusing FindField can hinder performance

```
public static void ExcessiveFindFieldCalls(IFeatureClass featureClass)
{
    using(ComReleaser comReleaser = new ComReleaser())
    {
        // Open a cursor on the feature class.
        IFeatureCursor featureCursor = featureClass.Search(null, true);
        comReleaser.ManageLifetime(featureCursor);

        // Display the NAME value from each feature.
        IFeature feature = null;
        while ((feature = featureCursor.NextFeature()) != null)
        {
            Console.WriteLine(feature.get_Value(featureClass.FindField("NAME")));
        }
    }
}
```

## Top Mistakes – Cursors across edit operations

- Documented as a “don’t do this”
- Blocked at 10.0

## Top Mistakes – DDL in edit sessions

- Methods that trigger DDL commands, such as `IFeatureWorkspace.CreateTable` or `IClass.AddField`, should never be called inside an edit session
  - DDL commands will commit any transactions that are currently open, making it impossible to rollback any unwanted edits if an error occurs
  - E.g., a custom editing application that adds new values to a coded value domain based on a user's edits, then fails unexpectedly when the application tries to commit the edits

## Top Mistakes – Calling Store in Store-triggered events

- Calling Store on the object again triggers the event model from within the model, leading to unexpected behavior
  - In some cases, this results in infinite recursion causing an application to hang, while in others, errors are returned with messages that might be difficult to interpret

## Top Mistakes – OnCreateFeature event

- Modifying the shape of the feature passed into the OnCreateFeature event, leading to unexpected behavior

```
void events_OnCreateFeature(ESRI.ArcGIS.Geodatabase.IObject obj)
{
    try
    {
        IGeometry newShape;
        //While creating a new feature,
        //don't handle it's on create event by modifying it's shape
        IFeature feature = (ESRI.ArcGIS.Geodatabase.IFeature)obj;
        feature.Shape = newShape;

        //do not call Store() on the feature which is being handled in the event
        feature.Store();
    }
    catch (Exception exc) {}
}
```

## Top Mistakes – Using GetFeature versus GetFeatures

- For performance purposes, anytime more than one feature is being retrieved using a known object ID, always use the GetFeatures method

```
private static void GetFeatureExample(IFeatureClass featureClass, int[] oidList)
{
    int nameFieldIndex = featureClass.FindField("NAME");
    foreach (int oid in oidList)
    {
        IFeature feature = featureClass.GetFeature(oid);
        Console.WriteLine("NAME: {0}", feature.get_Value(nameFieldIndex));
    }
}
```

## Top Mistakes – Relying on name objects for caching

- Calling Open on a name object does not auto cache the reference to the returned dataset
- Relying only on the name object causes the underlying dbms to open the fully table reference each time
- Cache the open dataset reference at the start of the application

## Top Mistakes – Careless reuse of variables

```
private static IFields FieldSetCreation()
{
    // Create a field collection and a field.
    IFields fields = new FieldsClass();
    IFieldsEdit fieldsEdit = (IFieldsEdit)fields;
    IField field = new FieldClass();
    IFieldEdit fieldEdit = (IFieldEdit)field;

    // Add an ObjectID field.
    fieldEdit.Name_2 = "OBJECTID";
    fieldEdit.Type_2 = esriFieldType.esriFieldTypeOID;
    fieldsEdit.AddField(field);

    // Add a text field.
    fieldEdit.Name_2 = "NAME";
    fieldEdit.Type_2 = esriFieldType.esriFieldTypeString;
    fieldsEdit.AddField(field);

    return fields;
}
```



## Top Mistakes – Change non-persisting schema objects

- When modifying geodatabase objects – e.g., datasets, domains, fields, etc., you should be aware that these classes fall into two categories of the following behaviors:
  - Those that automatically persist schema changes in the geodatabase, that is, tables
  - Those that do not – fields, domains, indexes
- A classic example of this are the methods, `IClass: AddField` and `IFieldsEdit: AddField`
  - When the former is called, the API adds a field to the database table
  - When the latter is called, a field is added to the field collection in memory but no change is made to the actual table

## Top Mistakes – Inserts and relationship class notification

- Notification (also known as messaging) ensures the proper behavior of composite relationships, feature-linked annotation, and custom class extensions
  - This behavior is not free
  - Edits and inserts to datasets that trigger notification is noticeably slower than the same operation on datasets that do not trigger any notification
- This performance hit can be mitigated by ensuring that all notified classes are opened before any inserts taking place

# Top Developer Mistakes

- Misusing recycling cursors
- Calling `FindField` in a loop
- Using cursors across edit operations
- DDL in edit sessions
- Calling `Store` inside of store triggered events
- Using `GetFeature` when `GetFeatures` could be used
- Name objects and dataset caching
- Careless reuse of variables
- Modifying schema objects that don't persist
- Inserts and relationship class notification

# Topic Summary

- Unique Instancing of Objects
- Cursors
- Caching Geodatabase Data
  - Spatial and Schema Caches
- Schema Locks
- Top Developer Mistakes

# Developer Resources

- ESRI blogs ([blogs.esri.com](http://blogs.esri.com))
  - [Inside the Geodatabase](#)
- Geodatabase “How To” articles
  - [Developers Guide to the Geodatabase](#)
  - [Many more articles in Geodatabase SDK](#)
    - Working with ArcGIS Components > Geodatabase Management
- Instructor lead training ([www.esri.com/training](http://www.esri.com/training))

# Final Notes

- Please fill out session surveys
  - Helps us improve the session
  - Rate a Session: Effective Geodatabase Programming
- All sessions are recorded and will be available on the Resource Center
  - Slides and code samples will also be available
- Still have questions?
  - Desktop Island in the Showcase



Understanding our world.