Code Performance Optimization

Presented at Adobe ColdFusion Summit 2023

by Denard Springle

Agenda

- What Is Concurrency?
- Resource-Efficient Coding Patterns
- Caching Strategies (Native CF and CacheBox)
- Database Performance Optimizations
- Q&A

DENARD SPRINGLE

LOOKING FOR MY NEXT CHALLENGE!

- denard.s@3dgeeksolutions.com
- CF Developer since v3.1
- Polyglot (Java, Node, etc.)
- in @denardspringleiv

What Is Concurrency?

- A measure of how many active requests can be serviced at the same time
- The more concurrent requests you have the more resources (RAM, CPU and I/O) will be required
- Horizontal Scaling
  - Maximizing the available RAM, CPU and I/O in a single server (tall server)
  - Configuring OS, application servers, DBs and containers for best resource utilization
  - **Optimizing application code to use as few resources as possible**
- Vertical Scaling
  - Load balancing two or more horizontally scaled servers (and/or containers) to achieve higher concurrency

Resource-Efficient Coding Patterns

- Use modern coding techniques (OO, (H)MVC, FP, etc.)
- Use script vs tags (except in views) – faster compilation and execution
- Avoid scope cascading
- Implement the fast exit strategy
- Break (out of loops) and Continue (over loops)
- Avoid the string processor (`variable` vs. `#variable#` / `"#variable#"`)
- Avoid creating unnecessary variables
- Use threads (and parallel, in Lucee and ACF2023+)
- Implement Java directly
- Use caching strategies

Avoid Scope Cascading

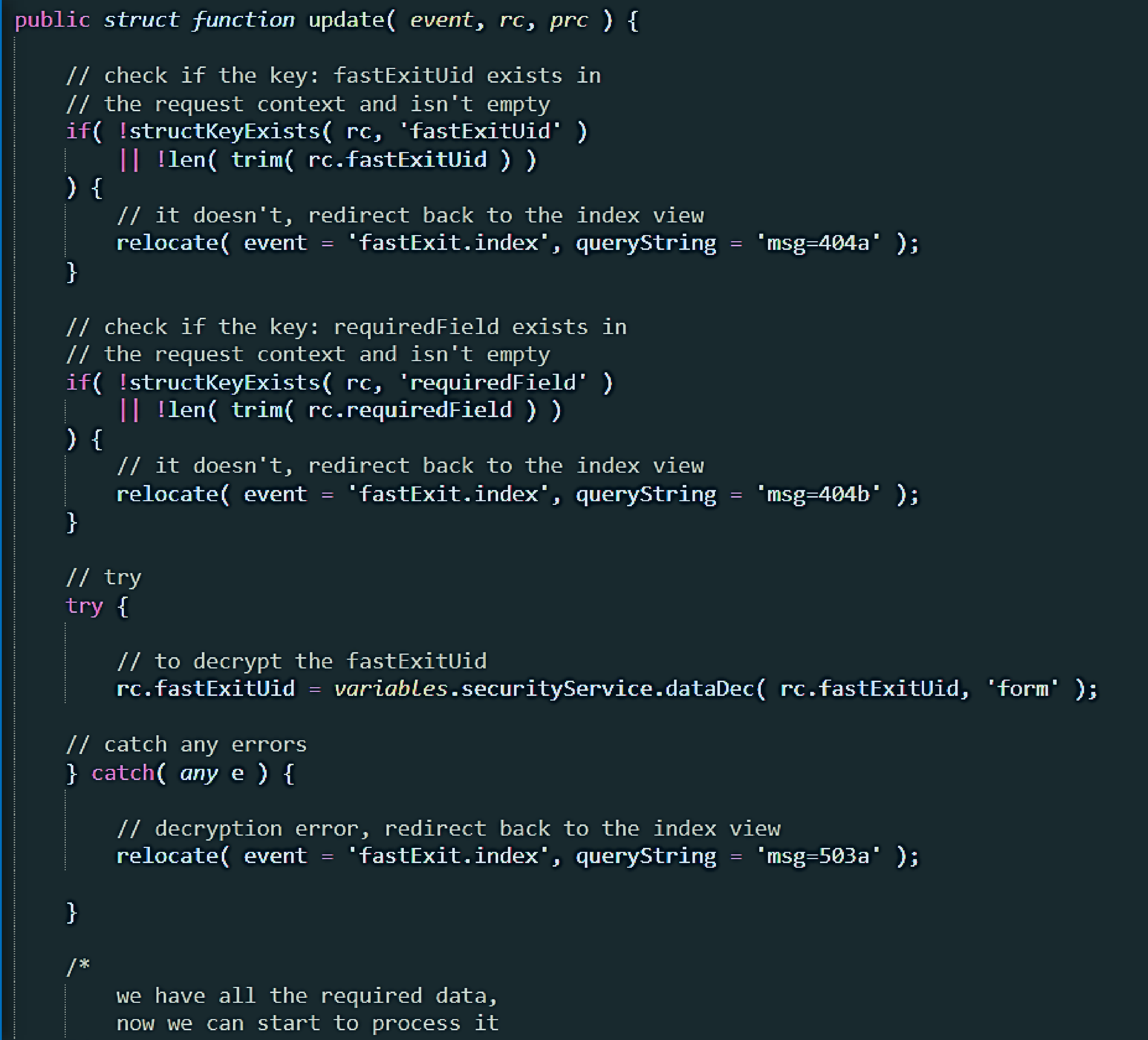
- Always SCOPE your variables (e.g. `form.myVariable VS myVariable`)
- local, arguments, thread local (inside threads), query (inside query loop), thread, variables, cgi, cffile, url, form, cookie, client – When no scope is provided, ColdFusion has to search through scopes sequentially until it finds a matching variable. This is time consuming and potentially unsafe.
- To speed up execution, as well as eliminate ambiguity and potential security threats, always fully scope the variables you use. The only exception is the variables scope, which ColdFusion always scans first and is implied when defining any variable outside of a function.
- var scope variables inside of functions. `var myArray = [];`

Delayed Exits Hold Requests Open Longer

```
if( structKeyExists( url, ‘bob’ ) ) {
    // do something with url.bob
    if( structKeyExists( url, ‘jane’ ) ) {
        // do something with url.jane
        if( structKeyExists( url, ‘zed’ ) {
            // do something with url.bob, url.jane, url.zed
        } else {
            // exit
    } else {
        // exit
} else {
    // exit
}
```

Fast Exit Strategy

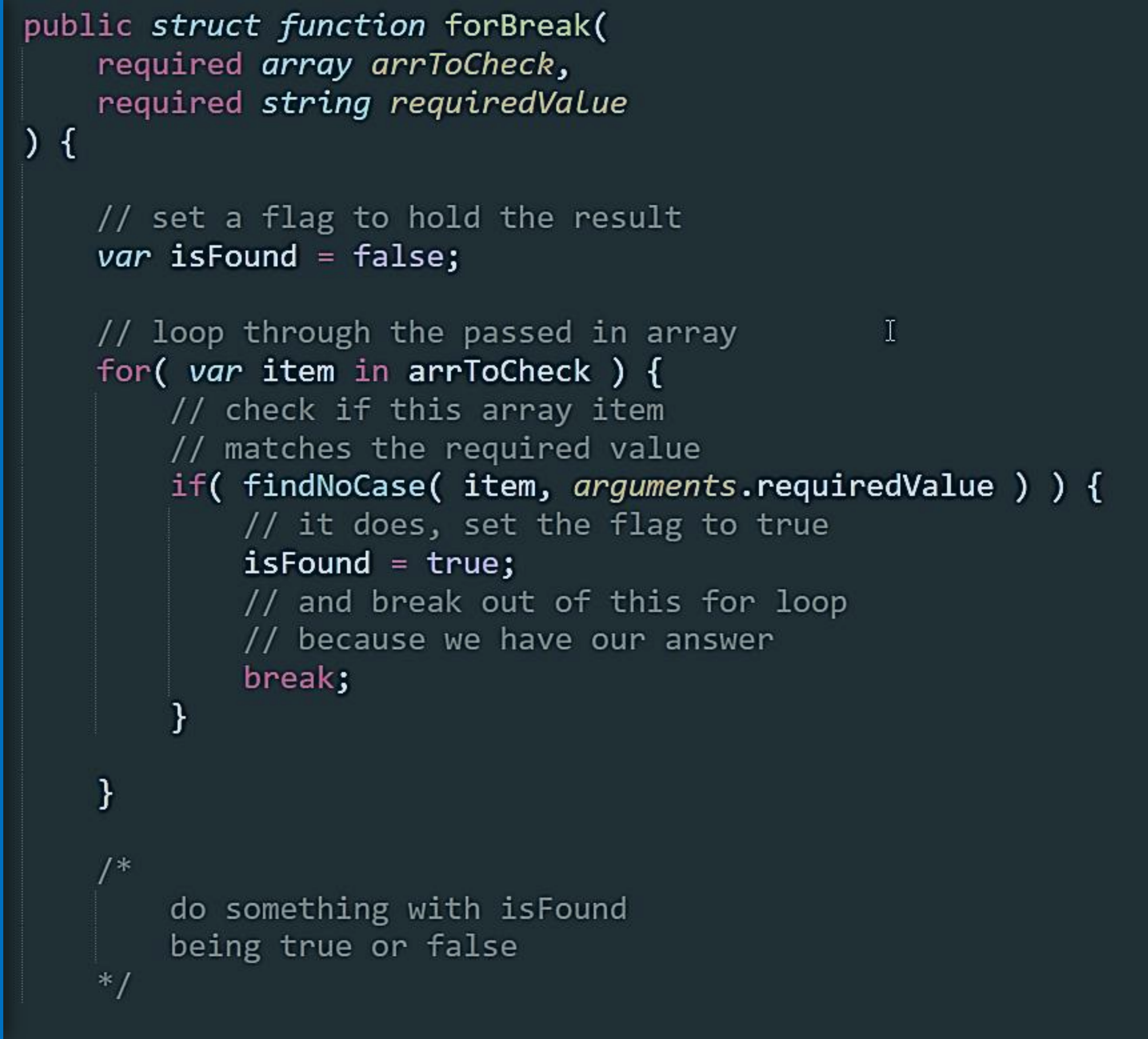
- End requests that cannot be processed as quickly as possible to allow other pending requests to be processed



```
public struct function update( event, rc, prc ) {

    // check if the key: fastExitUid exists in
    // the request context and isn't empty
    if( !structKeyExists( rc, 'fastExitUid' )
        || !len( trim( rc.fastExitUid ) )
    ) {
        // it doesn't, redirect back to the index view
        relocate( event = 'fastExit.index', queryString = 'msg=404a' );
    }

    // check if the key: requiredField exists in
    // the request context and isn't empty
    if( !structKeyExists( rc, 'requiredField' )
        || !len( trim( rc.requiredField ) )
    ) {
        // it doesn't, redirect back to the index view
        relocate( event = 'fastExit.index', queryString = 'msg=404b' );
    }

    // try
    try {

        // to decrypt the fastExitUid
        rc.fastExitUid = variables.securityService.dataDec( rc.fastExitUid, 'form' );

    // catch any errors
    } catch( any e ) {

        // decryption error, redirect back to the index view
        relocate( event = 'fastExit.index', queryString = 'msg=503a' );

    }

    /*
        we have all the required data,
        now we can start to process it
```

Break

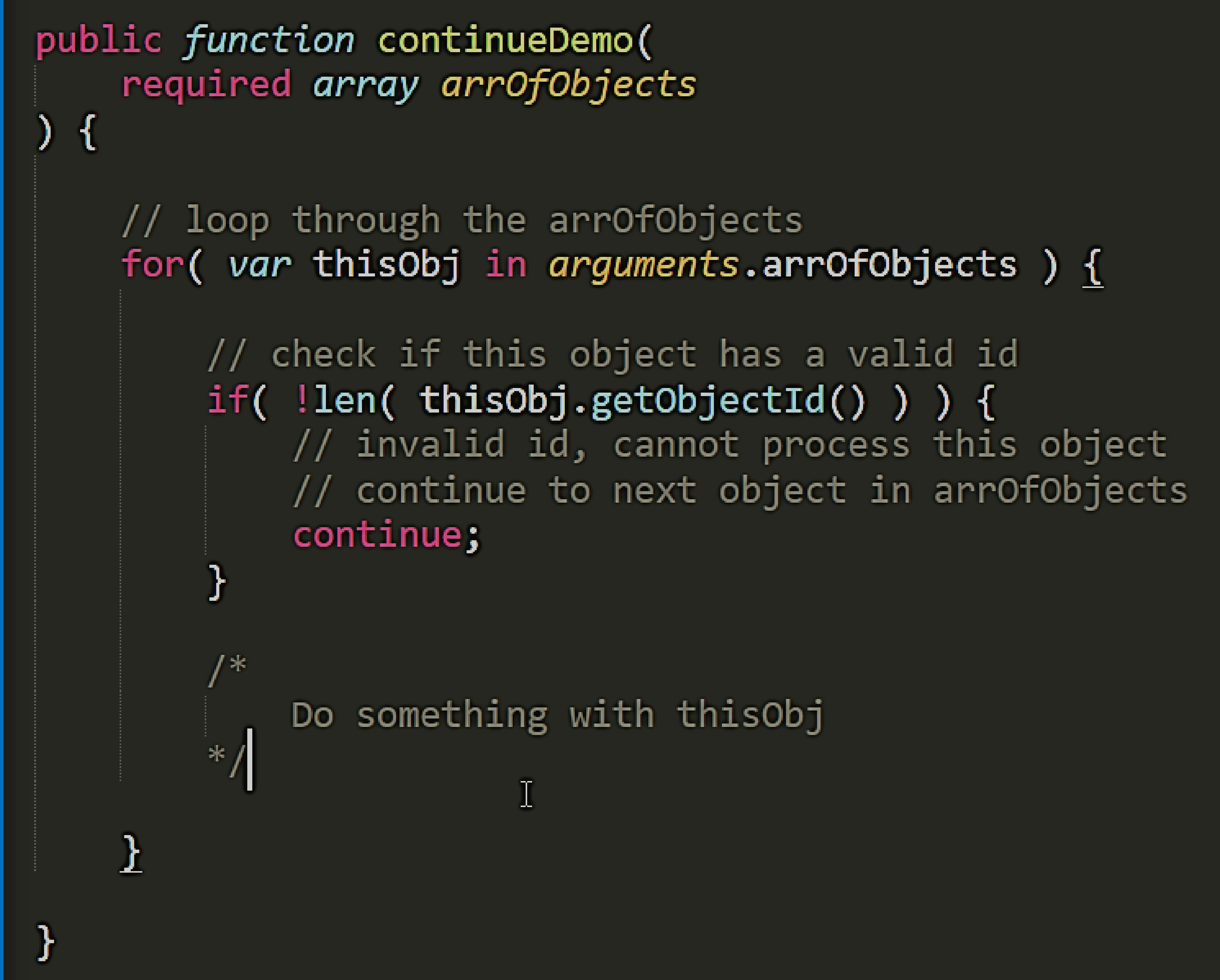
- break; out of loops (for, while) and decision trees (switch { case }) once you have satisfied a condition



```
public struct function forBreak(
    required array arrToCheck,
    required string requiredValue
) {

    // set a flag to hold the result
    var isFound = false;

    // loop through the passed in array
    for( var item in arrToCheck ) {
        // check if this array item
        // matches the required value
        if( findNoCase( item, arguments.requiredValue ) ) {
            // it does, set the flag to true
            isFound = true;
            // and break out of this for loop
            // because we have our answer
            break;
        }

    }

    /*
        do something with isFound
        being true or false
    */
```

Continue

- `continue;` over loops when data cannot be processed



```
public function continueDemo(
    required array arrOfObjects
) {

    // loop through the arrOfObjects
    for( var thisObj in arguments.arrOfObjects ) {

        // check if this object has a valid id
        if( !len( thisObj.getObjectId() ) ) {
            // invalid id, cannot process this object
            // continue to next object in arrOfObjects
            continue;
        }

        /*
            Do something with thisObj
        */

    }

}
```

Avoid Variable Creation

- Variable construction consumes RAM and CPU cycles, avoid creating them wherever possible

```
public string function getTempFilePath(
    required string filename
) {

    var tempDir = getTempDirectory();
    var tempFilename = replace( arguments.filename, ' ', '_', 'all' );
    var returnValue = tempDir & tempFilename;

    return returnValue;

}

// versus


public string function getTempFilePath(
    required string filename
) {

    return getTempDirectory() & replace( arguments.filename, ' ', '_', 'all' );

}
```

Know your language

- Avoid CPU/RAM intensive functions when alternatives exist

structKeyExists() vs structFind()

a++ vs incrementValue()

?: ternary operator vs iif()

- Use functional programming techniques e.g. map(), reduce(), filter(), each() – they're typically faster than loops

- ColdFusion is a tool in your toolbelt. You should master it as you would any other tool.

Caching Strategies

- What is cache?
  - A way to store (in RAM, on disk, etc.) and retrieve frequently accessed but rarely changing data instead of generating that data every time, especially data that increases resource utilization on the application or database server

- Why use cache?
  - Reduces load on application and database resources, allowing greater throughput for requests that require just-in-time data rendering and handling
  - Increases the performance and responsiveness of your application by caching complex calculated results, or even just the queries that make up a frequently requested but not time sensitive pages of data

Native Cache Functions

```
// get your cached data
var myData = cacheGet( 'myCachedData' );

// check if your cached data still exists, or if we're forcing a reload
if( isNull( myData ) || arguments.clearCache == true ) {

   // it doesn't exist, or we are reloading, get the data into myData
   myData = doSomeComplexProcess();


   // put it in the cache for an hour
   cachePut( 'myCachedData', myData, createTimeSpan( 0, 1, 0, 0 ) );

}

// do something with myData
```

User specific caching pattern

```
// get your cached data
var myData = cacheGet( 'myCachedData_' & arguments.userId );

// check if your cached data still exists, or if we're forcing a reload
if( isNull( myData ) || arguments.clearCache == true ) {

   // it doesn't exist, or we are reloading, get the data into myData
   myData = doSomeComplexProcess();
   // put it in the cache for an hour
   cachePut(
        'myCachedData_' & arguments.userId,
        myData,
        createTimeSpan( 0, 1, 0, 0 ) );

}

// do something with myData
```

Using a cache service layer

```
// get your cached data
var myData = cacheService.get( 'myCachedData' );

// check if your cached data still exists, or if we’re forcing a reload
if( isNull( myData ) || arguments.clearCache == true ) {

   // it doesn’t exist, or we are reloading, get the data into myData
   myData = doSomeComplexProcess();


   // put it in the cache for an hour
   cacheService.set( 'myCachedData', myData, '1h' );

}

// do something with myData
```

CacheBox

- CacheBox by Ortus provides multiple advanced features:
  - Cache Aggregators – Ability to aggregate multiple caching engines
  - Simple API – use a common set of functions `get()`, `set()` across aggregate engines
  - Fully Configurable – convention and runtime configuration options
  - Cache Monitoring – Dashboard and commands panel
  - Use with or without ColdBox and other products in the *Box ecosystem

Database Performance Optimization

- Use Stored Procs
- Index Relationship Keys
- Utilize Dirty Reads

Use Stored Procedures

- Take advantage of database server CPU/RAM resources
- Stored procedures, on average, execute faster than queries – q stored procedure is an execution plan
- In DAO the Create, Read, Update and Delete (CRUD) functions should use stored procedures
- You can use tools liks SSMS Tools Pack to generate CRUD stored procedures quickly

  https://www.ssmstoolspack.com/Download

Index Relationship Keys

- Tables should use integer primary and relationship (foreign) keys
- Integer indexes are faster than alternatives like GUID and suffer less fragmentation
- By indexing relationship keys you expedite the process of finding relational data when doing JOINs

Utilize Dirty Reads

- When having the most recent data is inconsequential to the result of a read, use a dirty read to prevent locking
- In MSSQL use `WITH  (NOLOCK)`
- The WITH (NOLOCK) table hint is used to override the default transaction isolation level of the table or the tables within the view in a specific query. The query will consume less memory in holding locks against that data and no deadlock will occur against the queries that are requesting the same data from that table.

GoFundMe Donors

- **Thomas Long**

- Aaron DeRenard
- Daniel Fredericks
- Kevin Wright
- Igal Sapir

Thank you for helping get me to Vegas to speak today!

Thank you for attending! Questions?