# OpenEdge 12.0 Database Performance and Server Side Joins

**Richard Banville**
Fellow, OpenEdge Development
October 26, 2018

# Data Access Performance Enhancements

- Increasing overall throughput
  - Provide more concurrency
  - More efficient use of resources
    - Speed vs space
    - Sharing, caching, optimize I/O, etc.
- Mechanisms
  - Improve algorithms (or make better guesses)
  - Limit contention
    - Asynchronous operations
    - Decrease time blocking others
    - Limit Time blocked
    - Eliminate need to block altogether

*Database development perspective*

# Data Access Performance Enhancements

**BHT Enhancements**

- Random data access for large deployments
- Concurrency for table scans of small tables

**Threaded DB Server**

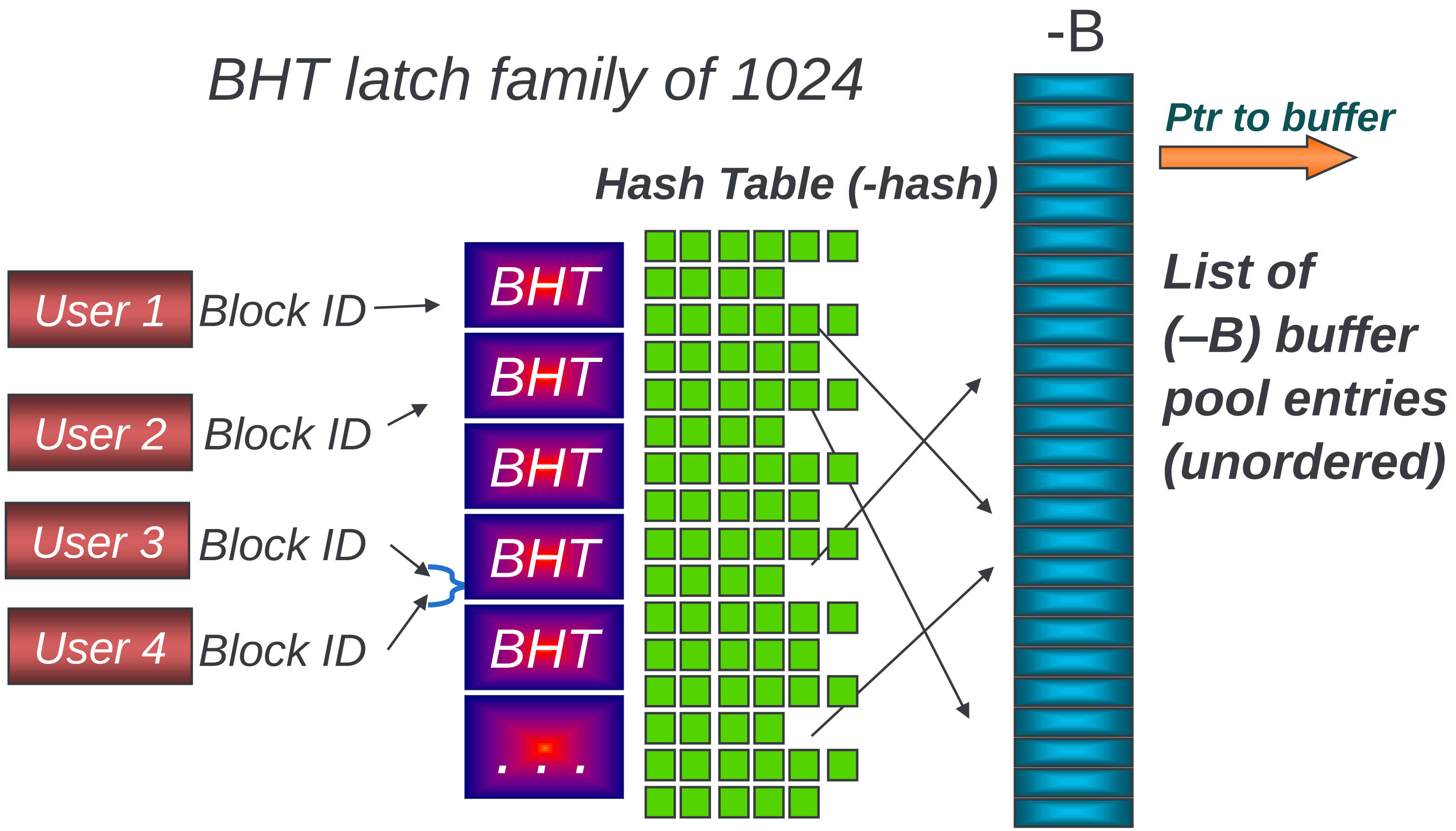
- Concurrent processing of remote client requests
- Not parallel statement execution

**Server Side Joins**

- Join operations performed server side
- Improved performance via decreased network traffic

# Additional BHT improvements

- What are BHTs?
  - Buffer pool hash table latches protecting –B look ups
    - bucket values
    - hash chains
      (value collisions)
  - Growing family
    Progression of 1, 4, 256, 1024 and still contention is seen; Why?




- *Buffer pool location lookup multi-threaded*
- *High activity, typically few naps*

# Additional BHT improvements

## Two main reasons for BHT contention

| | |
|---|---|
| 1 | **Larger database deployments**<br><br>• Running run with larger –B<br>  – Each BHT protects more hash buckets<br>• # concurrent users increasing |
| 2 | **Applications with data contention issues**<br><br>• Access to small tables are not locally cached. |

# 1 Larger database deployments

- For example
  - -B 6,000,000  with default –hash of 1471
  - BHT @ 1024, = ~1.4 buckets per latch
  - Avg 4 hash chain entries per bucket
    - ~5,860 hash entries locked per BHT latch
    - Contention chances increased
  - Increase –hash?
    - Fewer hash collisions and therefore shorter chain length
      - May decrease time the BHT is held
    - Does nothing to change # entries protected by each latch.

# 1 Larger database deployments

Resolution:  (OE 11.7.3 & OE 12.0)

- -hashLatchFactor default 10%
  - Percentage of hash buckets per –B hash latch (BHT)
  - Increase –hash “automatically” increases # BHT latches
  - Helps improve random data access BHT contention
- Why not always 100%?
  - -B 6,000,000 = ~ 1,500,000 latches  = ~ 23 MB
  - Page out / page in may require 2 BHT latches
    - Increased likelihood with higher % of latches
- At 100% can I still see BHT waits?

# 2 Applications with data contention issues

- Frequent scanning of small tables
  - Few blocks accessed frequently - not really random access
  - Not helped much by -hashLatchFactor
  - Could be locally cached by the application

- Typical data access:
  - Records: random except for table scan
    - Accessed in some indexed order
    - Sequential access limited by “rec per block” setting
  - Indexes: Sequential
    - Indexes are highly compressed
    - Many entries in one index block

# 2 Applications with data contention issues

Resolution:

- Optimistic buffer pool lookups
  - Remember not only last block accessed,
    but remember where in the -B the buffer resided last
  - Eliminates need for many BHT requests
  - Helps both random and small table data access
  - Index scan and “true” table scan only (sequential access)
- Result?
  - 50% reduction in hash table lookups (higher for “true” table scans)

# Multi-threaded DB Server

## *Isn’t the database already multi-threaded?*

# The OE DB Storage Engine is indeed thread safe

- The Storage Engine provides threaded access to data for

- PASOE accesses the database via threads
  - Uses a thread pooling technique
- OE SQL accesses the database via threads
  - Employs one thread per connection
- Certain DB utilities utilize threads for data access

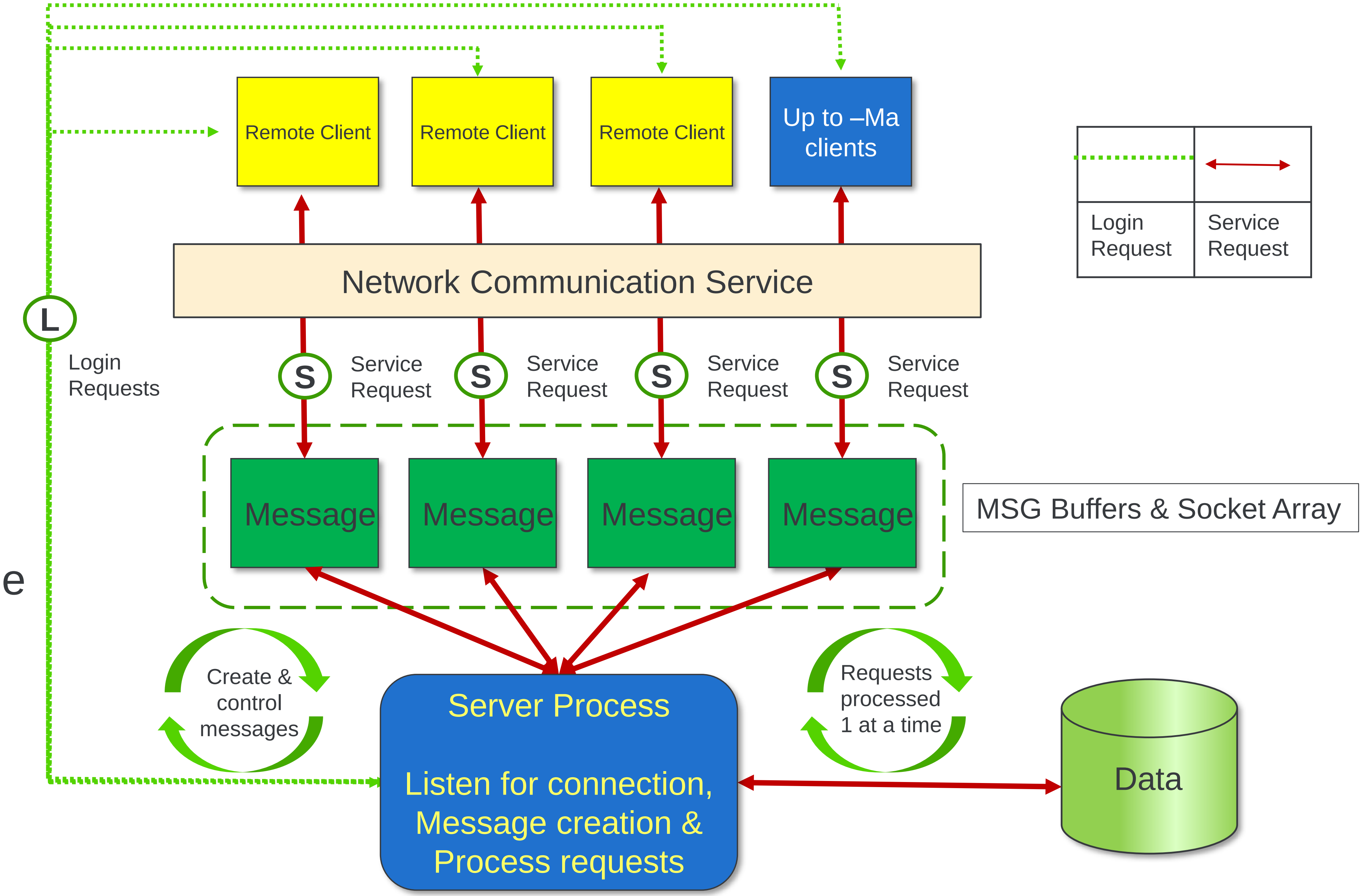
- ABL Database Server is not multi-threaded
  - Each server process handles data requests for multiple connections one at a time.

# Multi-threaded DB Server – Why?

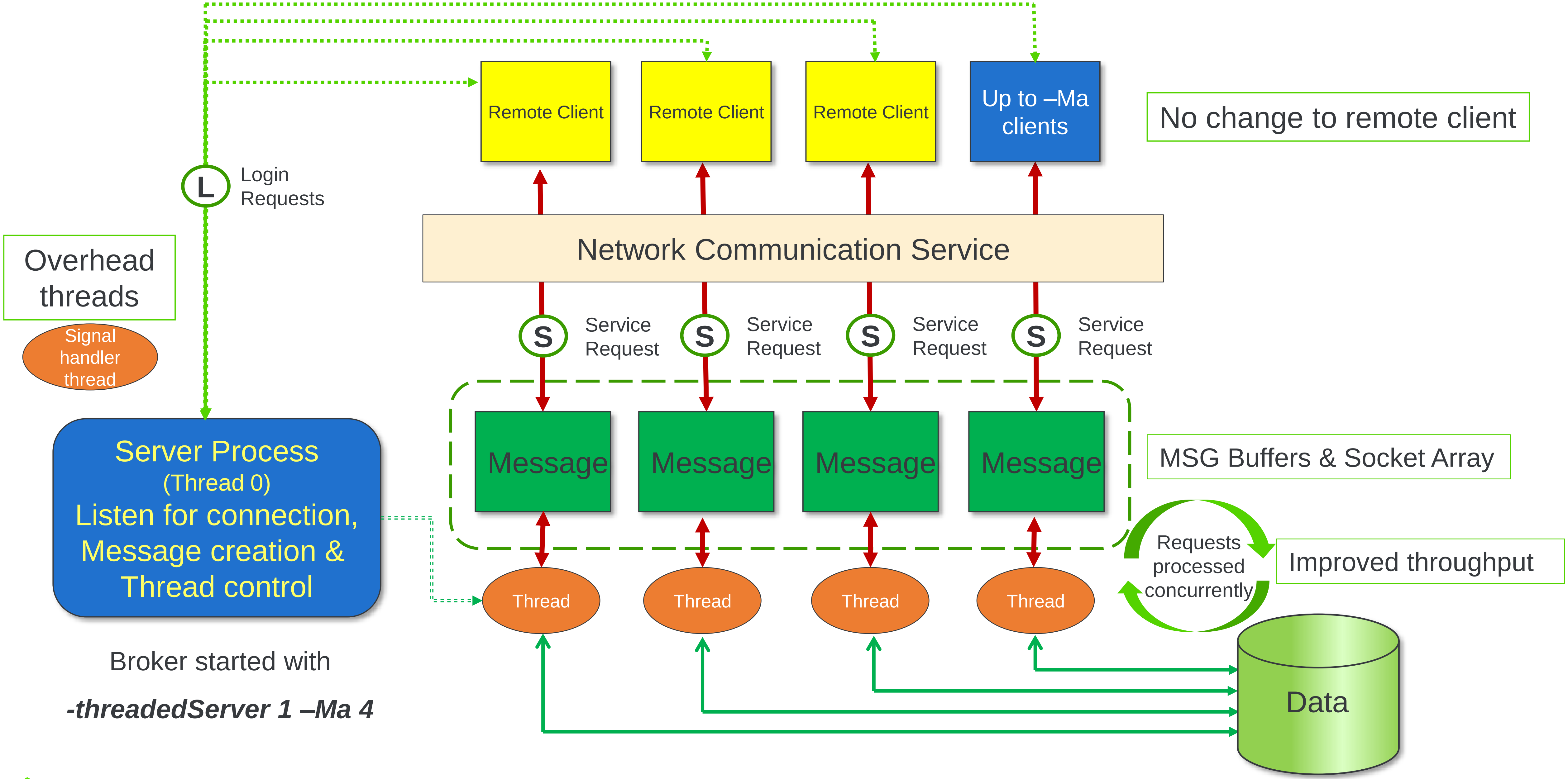
- Improved performance
  - Processing requests in parallel improves remote client performance
  - Enhanced lock wait processing
  - Connection processing separated from OLTP
  - Decreases context switching costs
- Continuous availability
  - Kill of remote client can’t crash a database
    - Remote client process never executes in a database critical section
- Enabler for Server Side Join project
  - Served clients don’t need to wait another’s completion

# Requests of Server – Classic Model

Unused CPU
power on
server machine



# Requests of Server – Threaded Model




Broker started with

***-threadedServer 1 –Ma 4***

Progress

# Parameters

- Broker specific configuration (not database wide)
  - Primary vs secondary brokers
  - -ServerType (ABL, SQL, BOTH)
    - Sql only Brokers – has no effect
- -threadedServer 1  -S <service> -H <hostname>
  - On by default
    - (19151) Threaded database server (-threadedServer): Enabled
- -threadedServerStack 512
  - Reserved stack space for each thread
    - (19159) Threaded stack size for threaded database servers (-threadedServerStack): 512k

# More on Parameters

- -Mi, -Ma, -Mn
- Checking parameter settings
  - _dbparams, _servers parameter array
  - .lg and promon
- ulimits
  - “max user processes” (threads),  “stack size”, “virtual memory”
  - No additional file handles required
    - Threads share file handles
    - Operating system deals with thread consistency
  - One open socket per connection (same as non-threaded)

# Debugging

- Promon / vst identification
  - Type: “TSRV”
  - New connection information:
    - TID: thread Id
    - SPID: Server PID
    - STID: Server TID
- Executables spawned by “preserve” broker process
  - -threadedServer 1: _m**t**prosrv
  - -threadedServer 0: _mprosrv
- .lg file: P-301988 T-301989 I TSRV
  - Thread id changed to OS’s LWP tid

# Debugging

- Debugging
  - ps –eflyT to see light weight processes
  - Stack trace information
    - Location information recorded in .lg file
  - kill –SIGUSR1
    - Remote client: TSRV: Protrace location: /usr1/richb/12/protrace.13573
    - Threaded server: Protrace.<pid>.<tid>
      protrace.301988.301988
      protrace.301988.301989 (…)
  - On SIGSEGV, thread causing the error will dump core & protrace
    - Server process exits; Same as non-threaded servers

# Light Weight Processes: 2 remote client example

**ps –eflyT**

# Light Weight Processes: 2 remote client example

**ps –eflyT**

| | UID | PID | SPID | PPID | CMD |
|---|---|---|---|---|---|
| B: | psc | 301939 | 301939 | 1 | _mprosrvx -S 6988 -threadedServer 1 |
| T0: | psc | 301988 | 301988 | 1 | _mtprosrvx -m1 -threadedServer 1 -threadedServerStack 512 |
| T1: | psc | 301988 | 301989 | 1 | _mtprosrvx -m1 -threadedServer 1 -threadedServerStack 512 |
| T2: | psc | 301988 | 301990 | 1 | _mtprosrvx -m1 -threadedServer 1 -threadedServerStack 512 |
| T3: | psc | 301988 | 301991 | 1 | _mtprosrvx -m1 -threadedServer 1 -threadedServerStack 512 |

- PPID: parent process ID
- SPID: LWP or thread ID

- Thread spawned on 1st connection request
- Threads re-used after client disconnects

# Tuning

- Performance profile mimics self service
  - Tune for self-service
- You can overwhelm your server machine faster
  - Improved performance requires more resource
- Broker centric
  - One broker can spawn threaded servers
  - A different broker can spawn non-threaded servers
- Latch contention increases – there are more concurrent requests
  - MTX, TXQ, BHT, BUF
  - BHT improvements help
  - General recovery subsystem tuning (ai/bi bufs, checkpoints…)

# Performance

- Typical high read work load
  - 250kB record reads/sec for 100 concurrent users, 8 DB Servers
- Query information
  - 7 table join
  - Local loopback
  - 25% record presentation
  - 75% record filtering

```
FOR EACH Table1 NO-LOCK,
    EACH Table2 NO-LOCK OF Table1
  , EACH Table3 NO-LOCK WHERE Table3.Percent_100 = Table2.Num_Key2
  , EACH Table4 NO-LOCK OF Table3
  , EACH Table5 NO-LOCK WHERE Table5.Percent_75 = Table4.Num_Key4
  , EACH Table6 NO-LOCK OF Table5
  , EACH Table7 NO-LOCK WHERE Table7.Percent_50 = Table6.Num_Key6
```

# Performance (As always, YMMV)

BHT

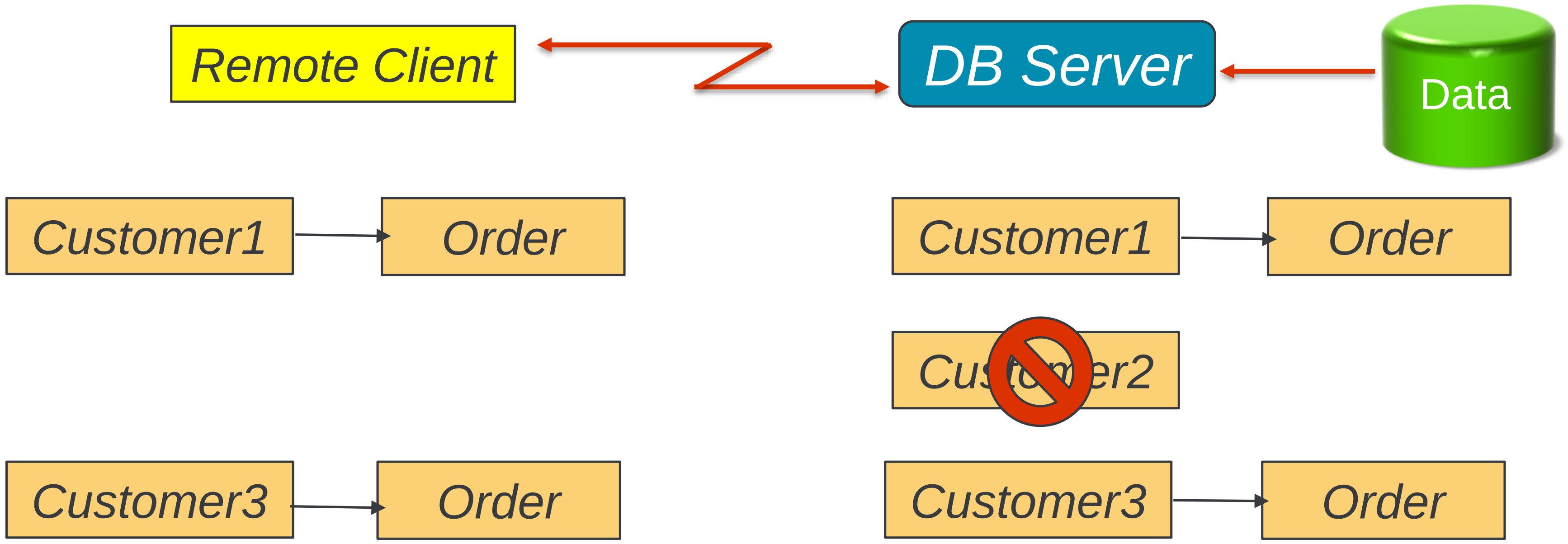
- At 100 users, little contention
- At 150 users, contention grows and BHT really shows a difference
- Bottom line:
  - If #users and read rates low, no change
  - Otherwise ~10% improvement

BHT & Threaded DB Server

- key factors: Configuration, lock conflicts & network latency
  - 1.8x to 2x performance improvement should be typical

# Server Side Joins

# Server query resolution model




*FOR EACH Customer, EACH Order of Customer WHERE ...*

- Client now only asks for the next set of data
  - In the past, Client tells Server what to do
- Reduces # records sent
- Reduces TCP communication requests

# SSJ OE 12.0 Functionality

- In the first release of the Server Side Join feature
  - Support of “for each” statements for joins up to 10 tables
    - no open query or dynamic query operations
- Requires multi-threaded database server
  - -ssj on by default if –threadedServer 1
    - (19329) Database server side join support (-ssj): Enabled
  - -ssj setting lasts for the life of the connection
  - -ssj can be changed online (currently primary broker only)
- Broker Specific Configurations
  - -threadedServer 1 and -ssj 1

# Realizing SSJ

- No changes to the application code
- Client logging
  - -logentrytypes QryInfo, -logginglevel 3
  - Monitor the change in
    - DB Reads:
    - Records from server:
  - Type: FOR Statement, Server-side join

# When does SSJ matter?

- # records filtered client side
  - Fewer records filtered clients side improves performance
- Cost of TCP I/O
  - Fewer network messages means fewer costly operations.

- If all records satisfy the query (no client side filtering), then there is no expected advantage.
  - True? **FALSE!**

# SSJ Example

- Report customers and their order information for orders promised tomorrow.

*For each customer*

*, each order of customer where*

*promise-date = (today + 1)*

*, each order-line of order*

# Threaded DB Server & SSJ Test Case

- Client log stats

| | DB Reads | | Recs from server | |
|---|---|---|---|---|
| **Server Activity** | **-ssj 0** | **-ssj 1** | **-ssj 0** | **-ssj1** |
| DB Blocks accessed: | 789 | 718 | | |
| Customer | 82 | 81 | 83 | 8 |
| Order | 202 | 22 | 8 | 8 |
| Order-line | 24 | 24 | 27 | 27 |

*For each customer*

*, each order of customer where*

*promise-date = 03/15/1993*

*, each order-line of order*

# Easing Network Traffic – Orders of magnitude!

| Server Activity | -ssj 0 | -ssj 1 | Description |
|---|---:|---:|---|
| Messages received | 375 | 53 | 7x fewer messages received |
| Bytes received | 63,420 | 5,432 | 11x less data received |
| Messages sent | 191 | 36 | 5x fewer messages sent |
| Bytes sent | 34,336 | 6,768 | 5x less data sent |
| “Records” received | 0 | 0 | |
| “Records” sent | *118 | *45 | 2.5x fewer records to client |
| Queries received | 191 | 34 | 5.5x fewer query requests |
| Result Count | 27 | 27 | Entities realized |

*For each customer*

*, each order of customer where*

*promise-date = 03/15/1993*

*, each order-line of order*

# Threaded DB Server & SSJ Test Case

- Client log stats (7 table join)

| | DB Reads | | Recs from server | |
|---|---|---|---|---|
| **Server Activity** | **-ssj 0** | **-ssj 1** | **-ssj 0** | **-ssj1** |
| DB Blocks accessed: | 30,375 | 21,578 | | |
| Table1 | 98 | 1 | 100 | 50 |
| Table2 | 198 | 228 | 200 | 100 |
| Table3 | 398 | 199 | 400 | 200 |
| Table4 | 798 | 398 | 800 | 400 |
| Table5 | 1,200 | 799 | 1,200 | 8,00 |
| Table6 | 2,398 | 1,598 | 2,400 | 1,600 |
| Table7 | 3,200 | 3,198 | 3,200 | 3,200 |
| Totals | | | 8,300 | 6,350 |

# Performance of Easing Network Traffic

| Server Activity | -ssj 0 | -ssj 1 | Description |
|---|---:|---:|---|
| Messages received | 14,364 | 4,711 | 3X fewer messages received |
| Bytes received | 2,490,192 | 510,132 | 4x less data received |
| Messages sent | 9,322 | 4,739 | 50% fewer messages sent |
| Bytes sent | 1,729,899 | 1,092,886 | 63% less data sent |
| “Records” received | 0 | 0 | |
| “Records” sent | *8,300 | *6,356 | 25% less filtering |
| Queries received | 9,262 | 4,703 | 50% fewer query requests |
| Result Count | 3,200 | 3,200 | Entities realized |

- Performance of the test case described
  - An additional 30% performance improvement
  - ~3X overall improvement (using localhost network access)
  - Expect even greater improvement with “true” remote access

# Factors Affecting Performance Enhancements

- Current concurrency conditions
- Data access patterns
- Configuration
  - # clients per server
    - More server processes increase context switching cost
  - 1 client per server
    - high concurrency, bad at record lock resolution)
- Network latency
- Amount of client side filtering
- Query type

# Performance,
# performance,
# performance.

BHT Improved concurrency

Multi-threaded ABL DB Server

Server-Side Joins

## Any Questions?

Progress®