Progress®

Let’s Take a Closer Look

Powerful – Performance & Scale

Progress®

OpenEdge Database:
Buffer Hash Table (BHT) Locking Delays

OpenEdge 12.0 – Performance and Scale

OpenEdge Database Server: ***Buffer Hash Table Locking***

- OpenEdge Database is conceptually two components: **Storage Engine** and **DB Server**
- Storage Engine and DB Server communicate through Shared Memory
- Contention issues between the two creates scale problems
- Larger -B creates more potential for contention
- **Issue comes down to the Buffer Hash Table (BHT) and the latching mechanics**
- **Two Main Problems with BHT Contention**
  - Random data access with larger DBs
  - Concurrency of table scans for small tables

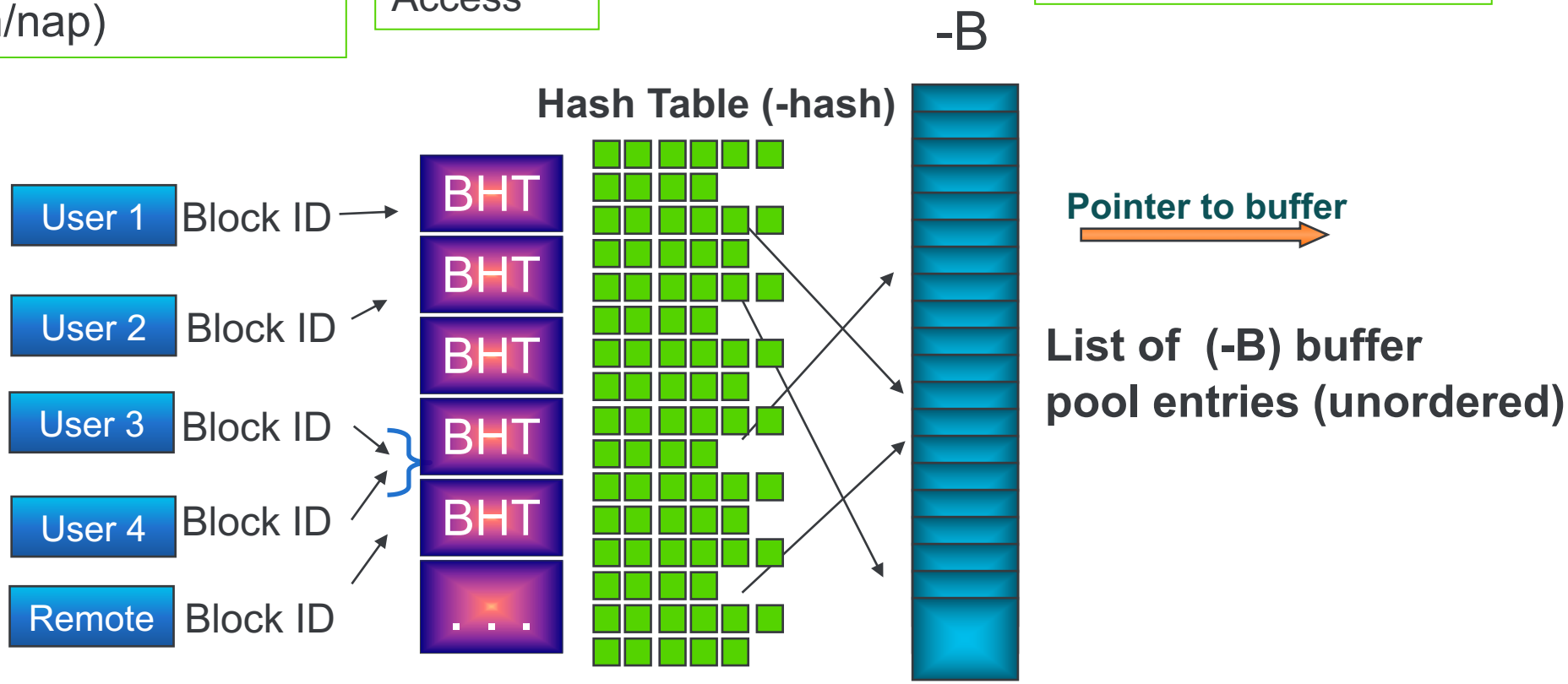
How Does Database Access Work?

Users wait for access to hash table (spin/nap)

Latch that grants Access

Controls Access

Bigger is Better
Tune for 99% hit rate



OpenEdge 12.0 – Performance and Scale

OpenEdge Database Server: ***Buffer Hash Table Locking***

- BHT Latches are now configured as a percentage of Hash Table (-hash) parameter
- No longer rely on making sure you have the correct setting
- Optimistic Buffer Pool Lookups
  - Remember what was looked up and where it is
  - Subsequent requests do not have to go through the buffer hash table
- Up to **28%** performance improvement
- **50%** less BHT latching conflicts in the buffer hash table
- **No application code changes required!**

Progress®

OpenEdge Database: Multi-Threaded Database Server

OpenEdge 12.0 – Performance and Scale

OpenEdge Database Server: ***Multi-threaded Processing***

Today, DB access requests are handled one-at-a-time

OpenEdge 12.0 – Performance and Scale

OpenEdge Database Server: ***Multi-threaded Processing***

Today, DB access requests are handled one-at-a-time

In 12.0, DB access requests are handled ***Concurrently*** for Remote Clients

OpenEdge 12.0 – Performance and Scale

OpenEdge Database Server: ***Multi-threaded Processing***

- Access from Remote Clients will now be multi-threaded
- Reduces the bottleneck on the Server process for Remote Client requests
- Spreads the requests across multiple threads
- **Up to 100% performance improvement**
- Better concurrency, parallelism and resource utilization
- Client/Server benefits start to outweigh costs compared with shared memory
- **No application code changes required!**