Dramatic scalability for data intensive architectures

Mike Stolz

VP of Architecture

GemStone Systems

GEMSTONE

"NETWORK IS THE DATABASE"

Agenda

- Need for a single Data + events platform
  - Example in financial trading
- Distributed Data fabric/GRID features
- Data Scalability artifacts
  - Partitioning – data and application behavior
    - Challenges, solutions, benchmark
  - Replication – pros and cons
  - Scaling for Grid environments
- Scaling Events
  - ‘Continuous querying’ engine
  - Distributed event processing with colocated data

GEMFIRE™
THE ENTERPRISE DATA FABRIC

Background on GemStone Systems

- Leader in Object Database technology since 1982
- Now specializes in memory-oriented distributed data management
  - 12 pending patents
- Over 200 installed customers in global 2000

- Main-memory data management focus driven by:
  - ***Clustering with cheap commodity servers***
  - ***Data colocation in app space has value***
  - ***Demand for very high performance with predictable throughput, latency and availability***

    - Capital markets – risk analytics, pricing, etc
    - Large e-commerce portals – real time fraud
    - Federal intelligence

GEMFIRE™
THE ENTERPRISE DATA FABRIC

Single Data + Events platform?

- Traditional database technology design is centralized and optimized for disk IO
  - Everything is ACID
  - Designed for Pull oriented apps
- PUSH oriented data management
  - High perf SOA, Web 2.0, Everything is Event driven (EDA)
  - Classic solution - DB with traditional messaging
    - Many problems

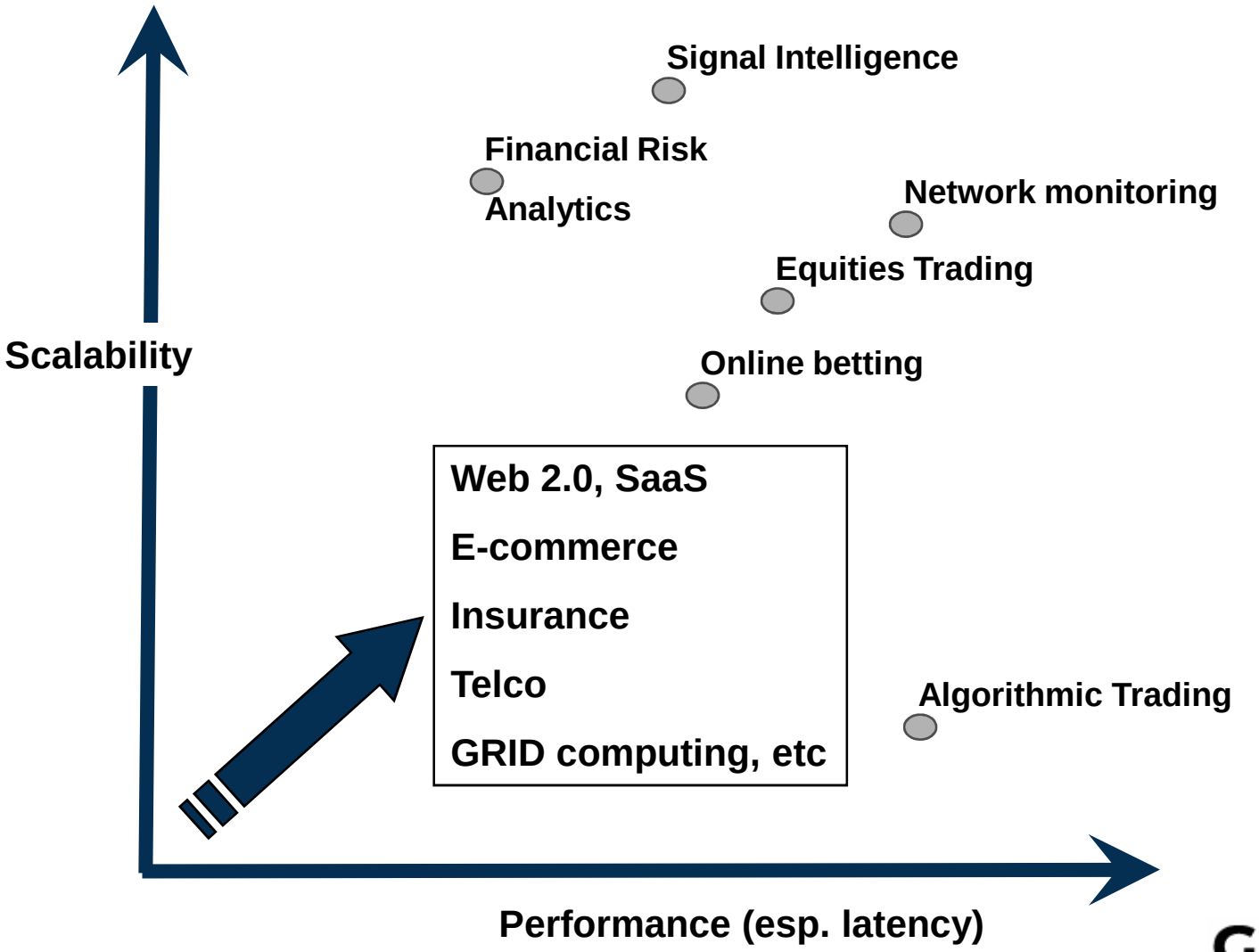
Data Fabric/Grid

Distributed main-memory oriented

Express complex interest in moving data and get notified when the data of interest changes

GEMFIRE™
THE ENTERPRISE DATA FABRIC

Motivating examples



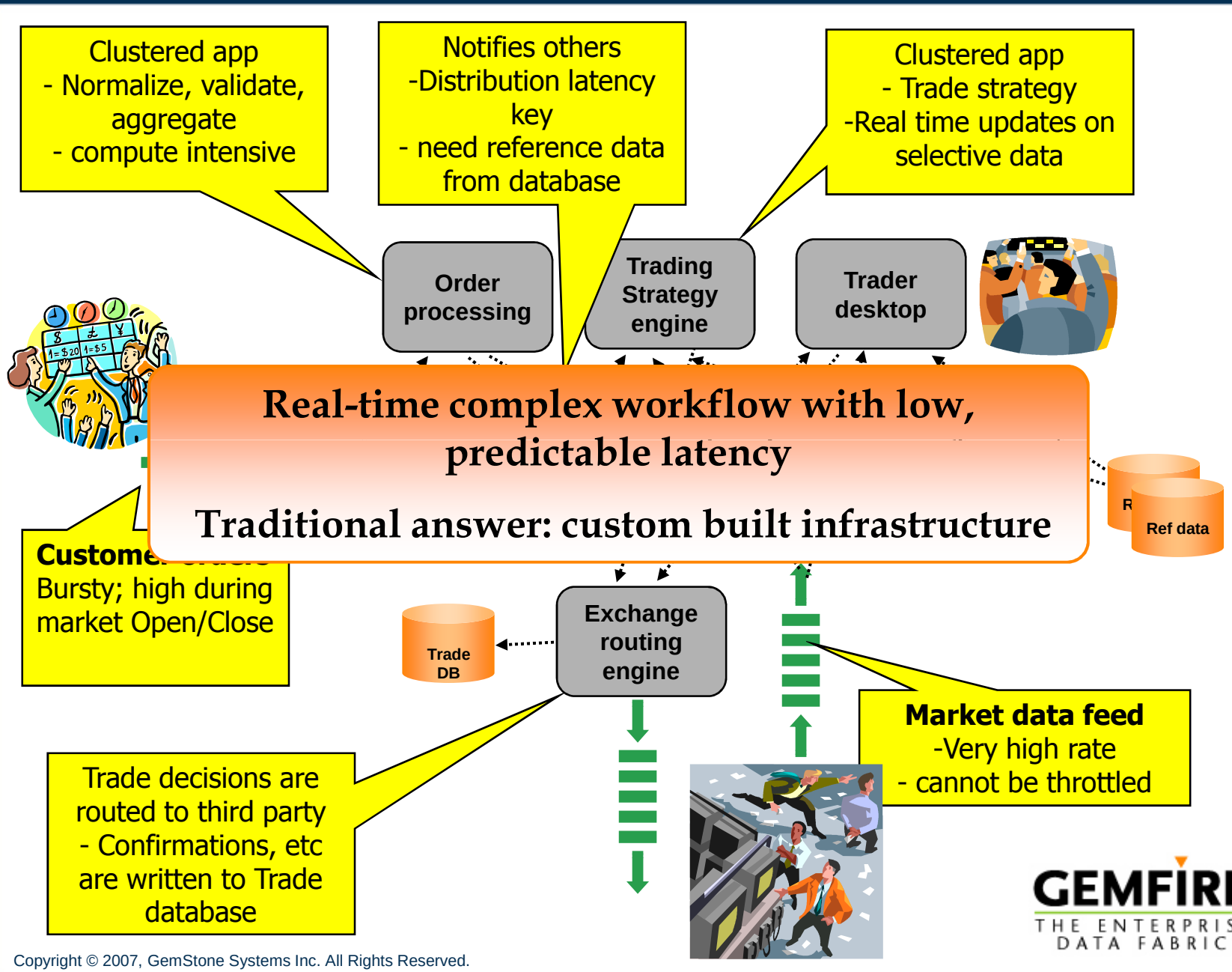
GEMFIRE™
THE ENTERPRISE DATA FABRIC

Extreme event/data architectures

- Events pushed at high rates
  - applications have to process and drive workflow at very high rate
- What is so extreme?
  - 1000's of messages per second and app dependent on other state to act
  - Derived data distributed to many processes
  - nothing can fail

GEMFIRE™
THE ENTERPRISE DATA FABRIC

Electronic Trade Order Management example



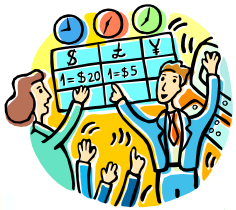
GEMFIRE™
THE ENTERPRISE DATA FABRIC

How does the Data Fabric help?

load balances the events to cluster for normalization and publish as objects into database

Hierarchical namespace allows updates of objects without having to serialize/deserialize the entire object

Allows apps to register interest on the database
- Propagates the events in a highly available manner



Order processing

Tra Strat engin

Trader desktop

DATA FABRIC

Ref data

Ref data

Customer orders **Distributed Event processing**

Trade DB

Exchange routing engine

apriori parallel load of database data spread across memory with redundancy

asynchronous write through to the database in order

Native C++ caching, very low latency

GEMFIRE
THE ENTERPRISE DATA FABRIC

Architect Dimensions

Want speed

- Pool distributed memory and manage as a single unit
- Replicate slow moving data and provide concurrent access
- Partition fast moving or large data sets and linearly scale
- Reliable Distribution with very low latencies

Want to do more with less (scale)

- Add nodes to dynamically rebalance data
- Add nodes to dynamically rebalance behavior
- I give you more memory, CPU, network
- want more throughput, more data

GEMFIRE™
THE ENTERPRISE DATA FABRIC

Architect Dimensions

Never Fail

- At least one redundant copy
- Load balance : manage CPU effectively across cluster/grid
- move data as well as processing: No more OOM
- Persistence using shared nothing disk architecture

Tell me when data of interest changes

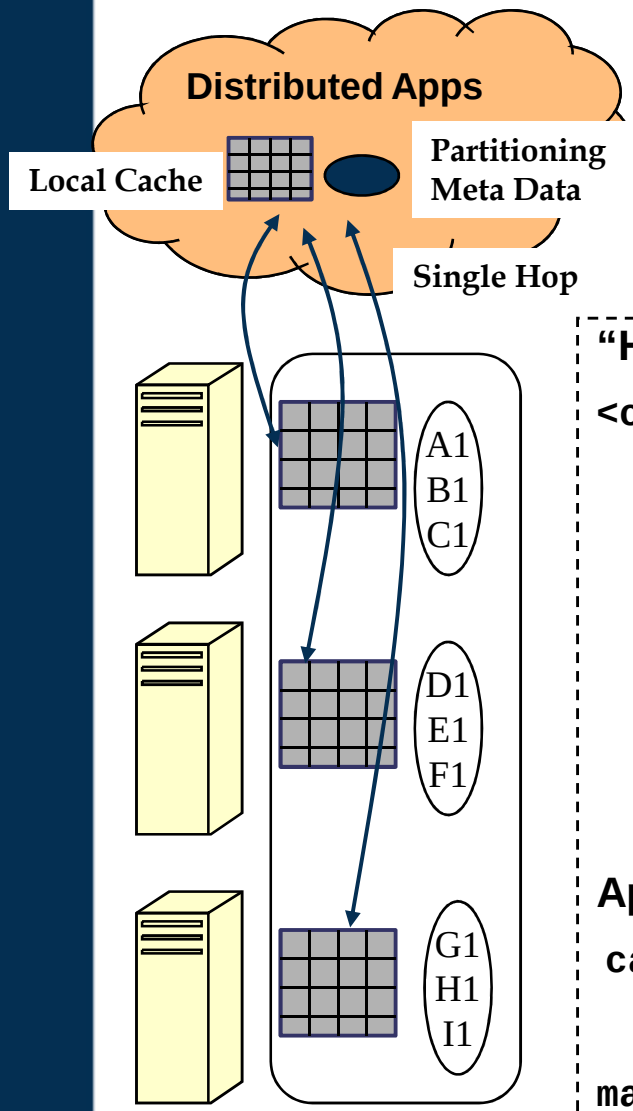
- Distribution infrastructure built for reliable pub-sub
- Multiple transports - TCP, UDP, reliable UDP Mcast
- event can be synch or async delivered, always in order
- takes care of duplicate events
- Sophisticated CQ engine

GEMFIRE™
THE ENTERPRISE
DATA FABRIC

Artifacts for Data Scalability

GEMSTONE

Artifact 1 – data partitioning



By keeping data spread across many nodes in memory, we can exploit the CPU and network capacity on each node simultaneously to provide linear scalability

"Hello World" Example

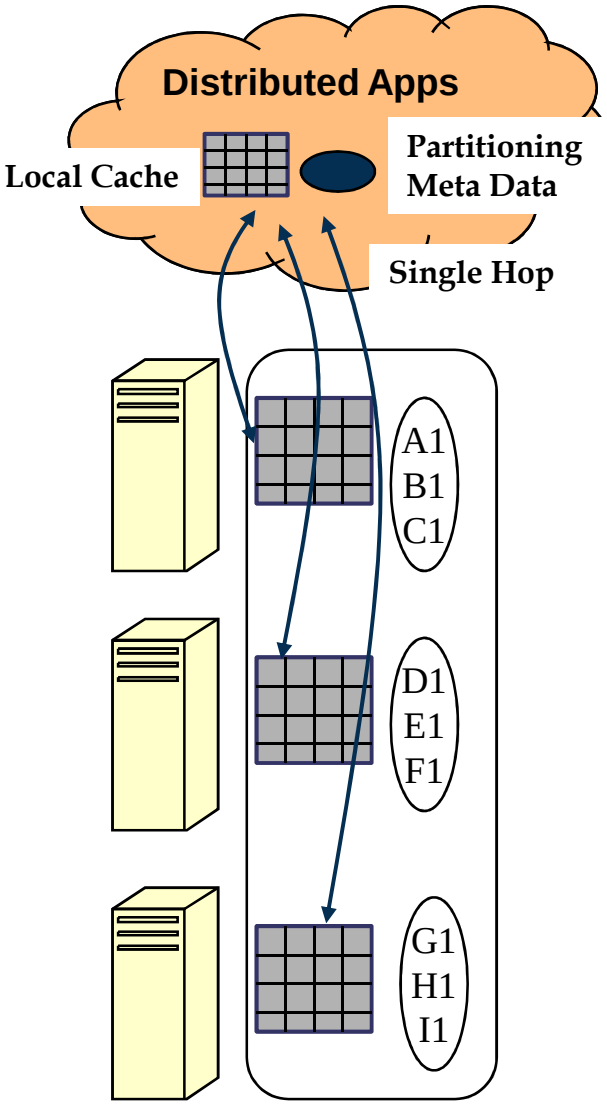
```
<cache>
   <region>  // Data region definition
       <region name="PartitionedOrders">
       <partition-attributes redundant-copies="1">
         <LOCAL_MAX_MEMORY> 2000MB ..
  </region>
  </cache>
```

Application Code ....

```
cache = CacheFactory.create
    (DistributedSystem.connect(<prop>));
map = (Map)cache.getRegion("PartitionedOrders");
 map.get(key) or map.put(key, data)
```

GEMFIRE™
THE ENTERPRISE DATA FABRIC

Data Partitioning Policies



- Data buckets distributed with redundancy
- Single network hop at most

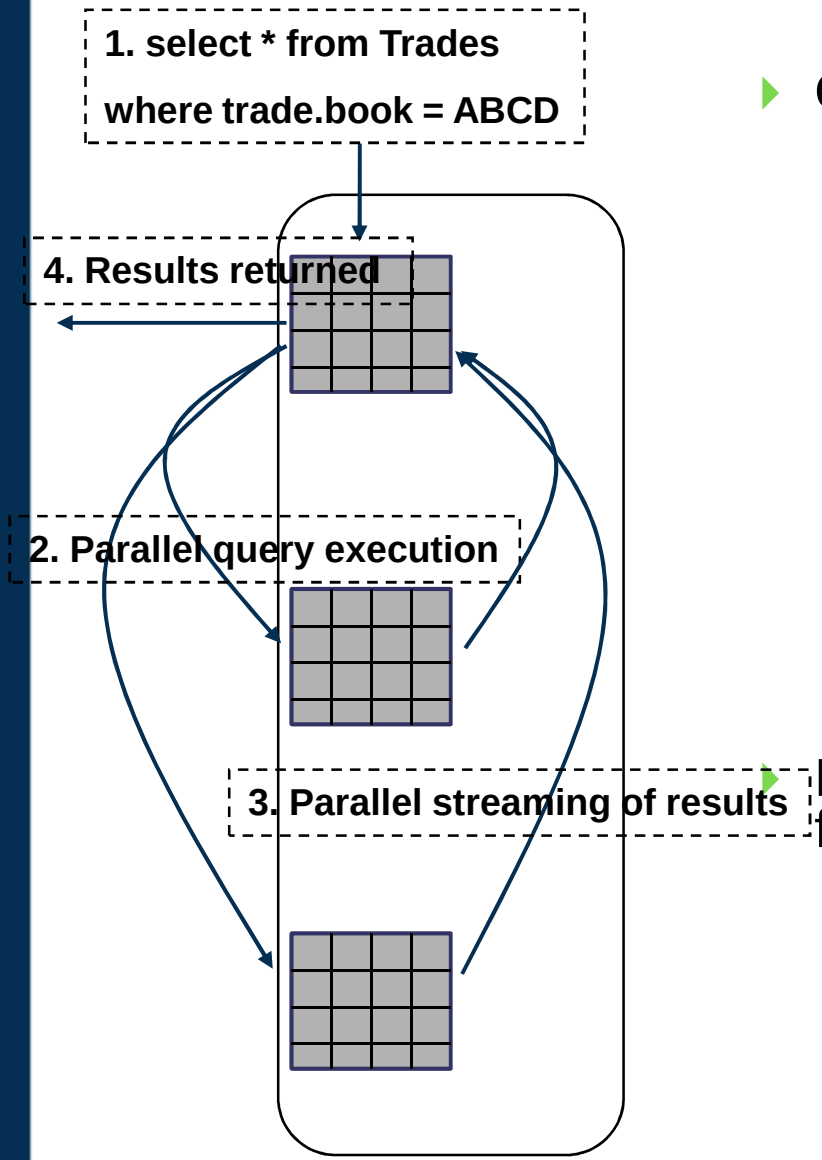
- Different Partitioning policies
- Policy 1: Hash partitioning
  - Suitable for key based access
  - Uniform random hashing

- Policy 2 – Relationship based
  - *Orders hash partitioned but associated line items are collocated*

- Policy 3 – Application managed
  - Grouped on data object field(s)
  - Customize what is collocated
  - Example: *'Manage all fills associated with an order in one data partition'*

GEMFIRE™
THE ENTERPRISE DATA FABRIC

Parallel “scatter gather”

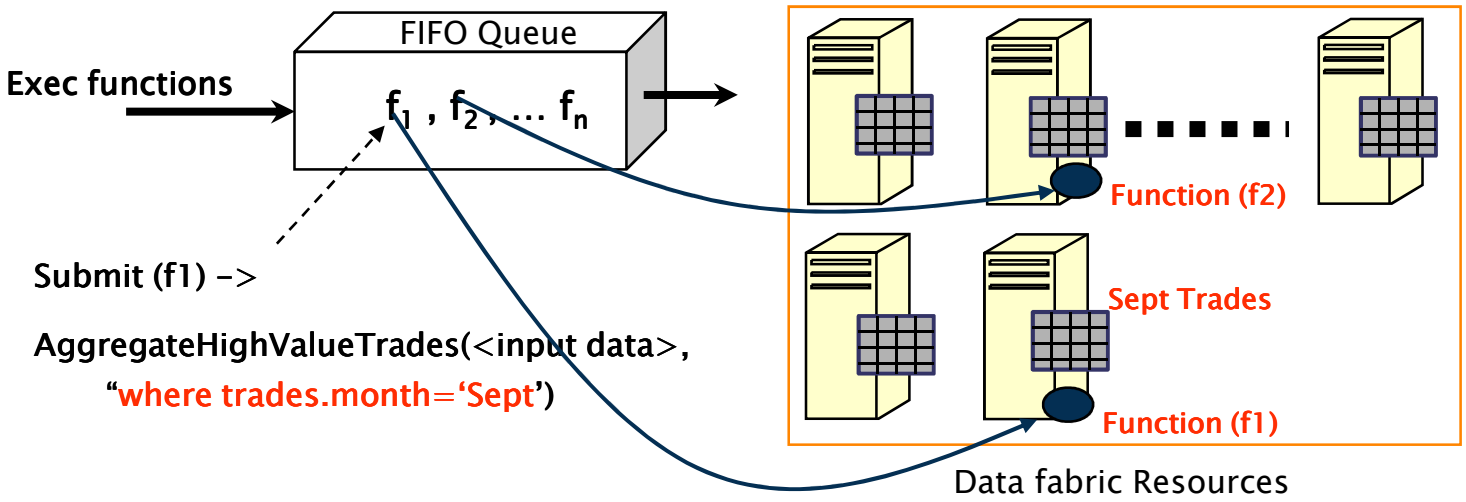


- Query execution for Hash policy
  - Parallelize query to each relevant node
  - Each node executes query in parallel using local indexes on data subset
  - Query result is streamed to coordinating node
  - Individual results are unioned for final result set
  - This “scatter-gather” algorithm can waste CPU cycles
- Partition the data on the common filter
  - For instance, most queries are filtered on a Trading book
  - Query predicate can be analyzed to prune partitions

GEMFIRE™
THE ENTERPRISE DATA FABRIC

Co-locate behavior with data

- Principle: Move task to computational resource with most of the relevant data before considering other nodes where data transfer becomes necessary
- Fabric function execution service
- Data dependency hints
  - Routing key, collection of keys, “where clause(s)”
- Serial or parallel execution
  - “Map Reduce”



GEMFIRE™
THE ENTERPRISE DATA FABRIC

Challenges with Partitioned data management

- What if allocated memory capacity is insufficient?
  - Shed load, or run Out-of-Memory
- What if data access pattern is non-uniform?
  - Hotspot: only small percentage of CPUs busy
- What if functions are dependent on small subset of data?
  - All functions are routed to small percentage of nodes

GEMFIRE™
THE ENTERPRISE DATA FABRIC

Dealing with them

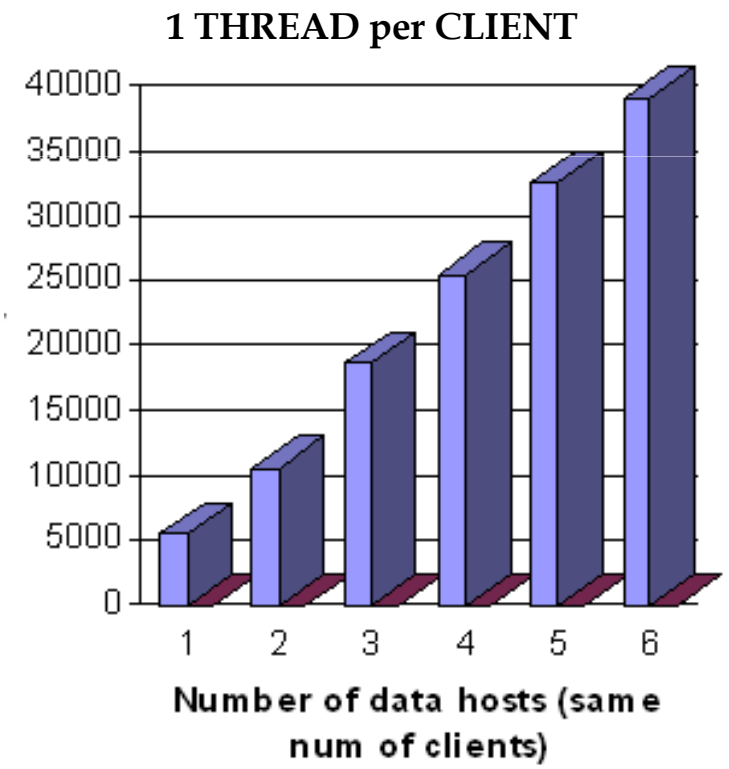
- Dynamic bucket migration
  - Act of transparently migrating data without blocking
  - Triggers
    - additional capacity is detected
    - Non-uniform distribution of data
      - memory utilization hits threshold (keep GC stress low)
    - Non-uniform distribution of collocated behavior or data access
      - *“one sector's transactions/trades are more frequently accessed than another sector’s transactions”*
  - Additional capacity automatically to maintain uniform distribution
- Automatic overflow to disk

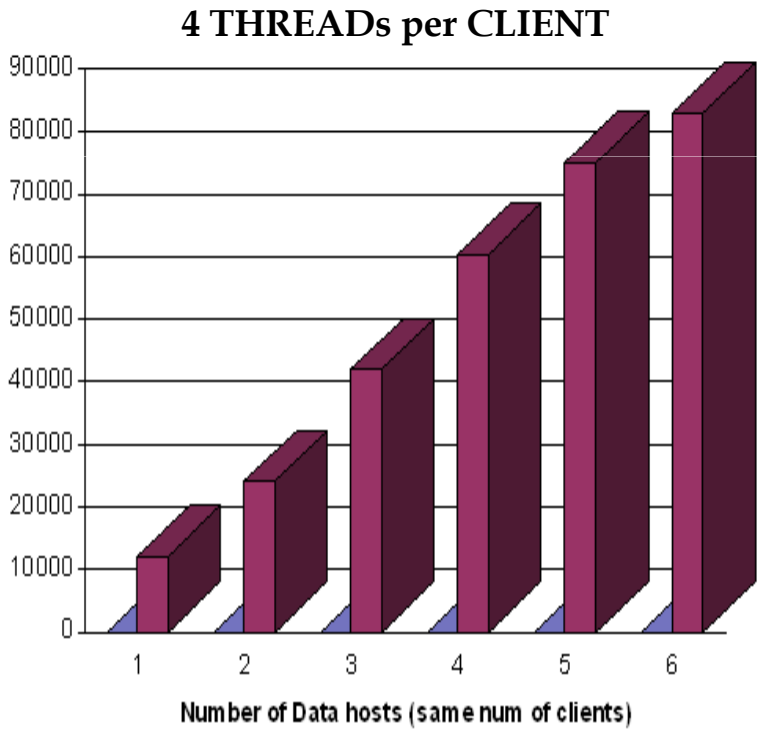
Sense-n-Respond based on load, throughput, latency

GEMFIRE™ THE ENTERPRISE DATA FABRIC

Simple throughput benchmark

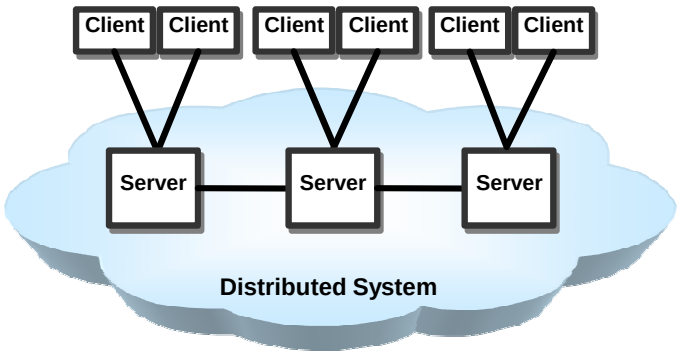
- Equal number of data nodes (partitions) and client nodes
- Client nodes do get/put of 2KB object
- Linux RH xx, 2 CPU * 3.2 Ghz
- Linear increase in throughput (26K for 2 → 60K for 4)

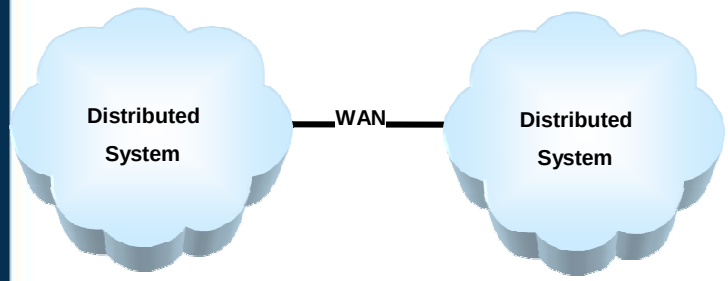




GEMFIRE™
THE ENTERPRISE DATA FABRIC

Artifact 2: Data replication





- When?
  - Data access patterns are too random.. hotspots cannot be avoided
  - or, data set is small and doesn't change much
  - or, data access across widely distributed networks
- Sync vs. Async replication
  - Async - potential for data loss, inconsistencies with client failover
    - More appropriate between applications or data centers
    - Data Servers pushing to clients

GEMFIRE™
THE ENTERPRISE DATA FABRIC

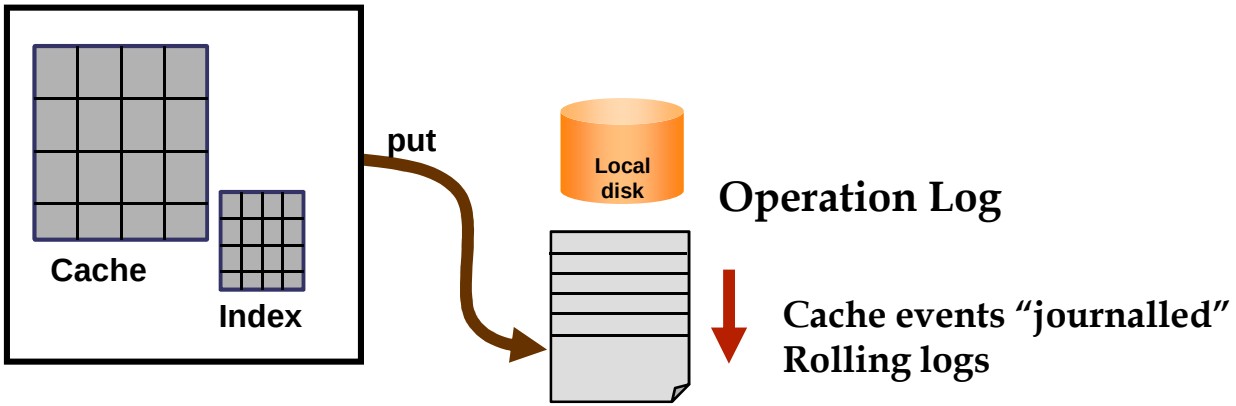
Broadcast based replication may not work

- Practical lessons tell us
  - Networks are often not tuned
  - broadcast can storm the network resulting in collisions
    - retransmissions can be expensive
  - ACKs are still unicast and will slow things down
  - Mcast traffic can be unfair to TCP

GEMFIRE™
THE ENTERPRISE DATA FABRIC

Artifact 3: Replication with Disk overflow

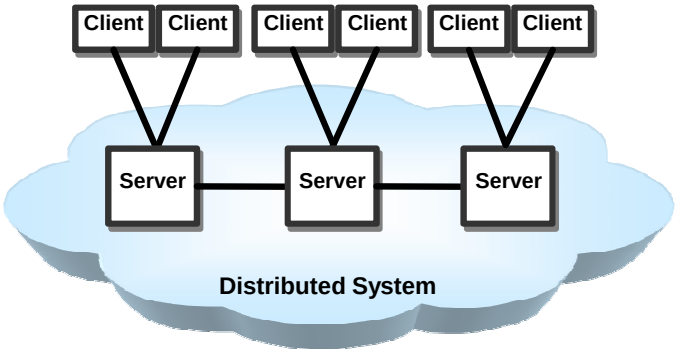
- Manage a large data size with small cluster
- Surprisingly, can be faster than partitioning for some scenarios
  - 16K cache puts/sec with async overflow (90% on disk)
- GemFire "Operations logging"
  - Basic notion: Avoid disk seek time
  - Sync writes to disk files with flush only to disk driver
  - Like a transaction log... continuous append mode
  - All entries are indexed in memory



GEMFIRE™
THE ENTERPRISE DATA FABRIC

Artifact 4: Scaling for GRID deployments

- Data fabric supports thousands of concurrent clients
- Large data volume sometimes changing often
- Super peer architecture
  - Large pool of servers pools memory, disk (peers)
  - Clients load balance to server farm
  - Load conditioning and Load Shedding



GEMFIRE™
THE ENTERPRISE DATA FABRIC

GEMFIRE™
THE ENTERPRISE
DATA FABRIC

Scaling Events

GEMSTONE

Premise

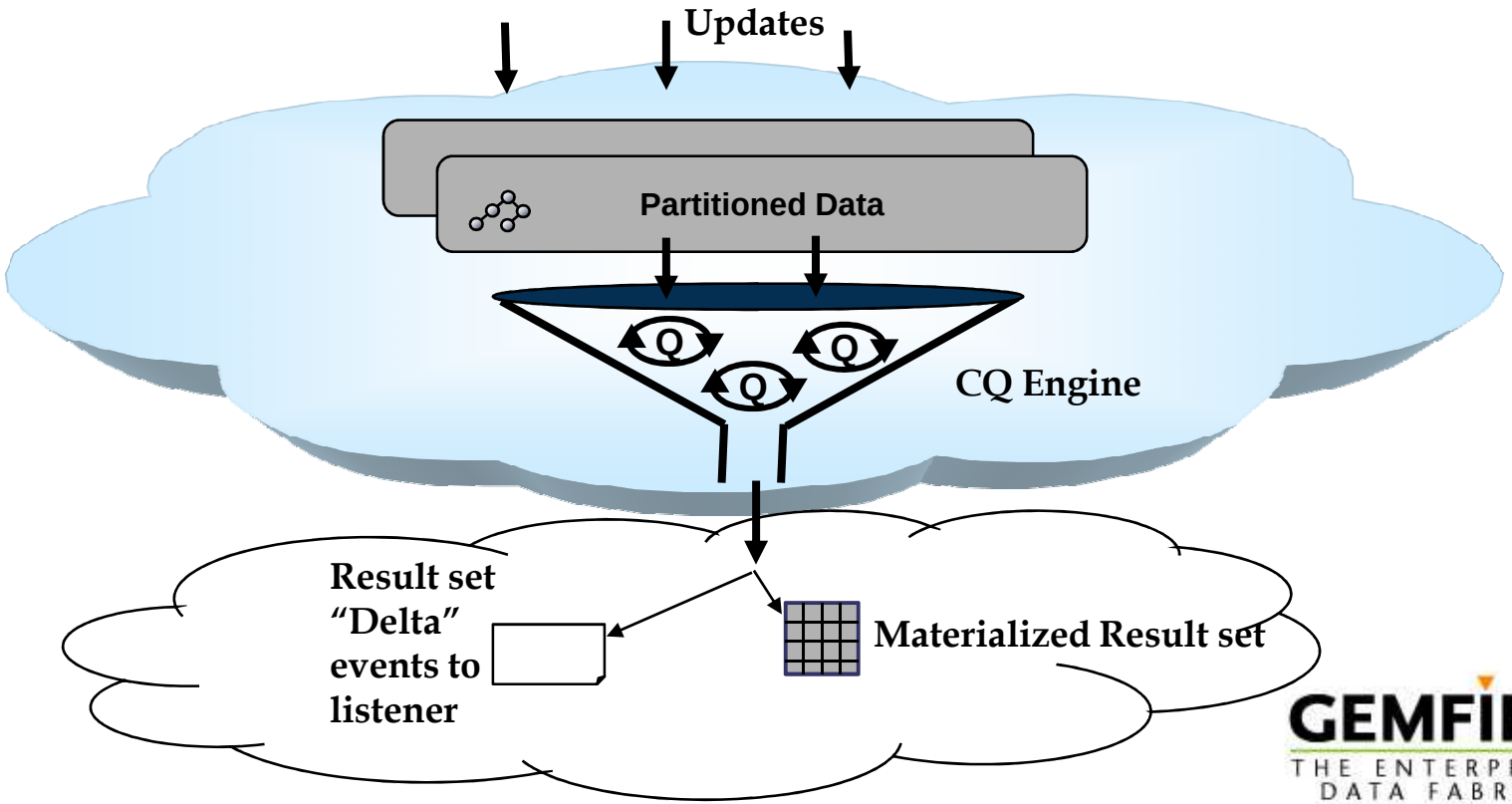
- All Data with relationships in distributed memory
- Applications directly modify objects and relationships
- Low latency distribution with data consistency
  - Only the “delta” propagates
- Subscribers get direct reference to app object and all context to act

- Benefits
  - No need to construct explicit messages: headers, content, identifiers for related data
  - When messages arrive, no need to fetch related data from the database
  - Don't have to deal with data inconsistencies or be throttled by the speed of the database

GEMFIRE™
THE ENTERPRISE DATA FABRIC

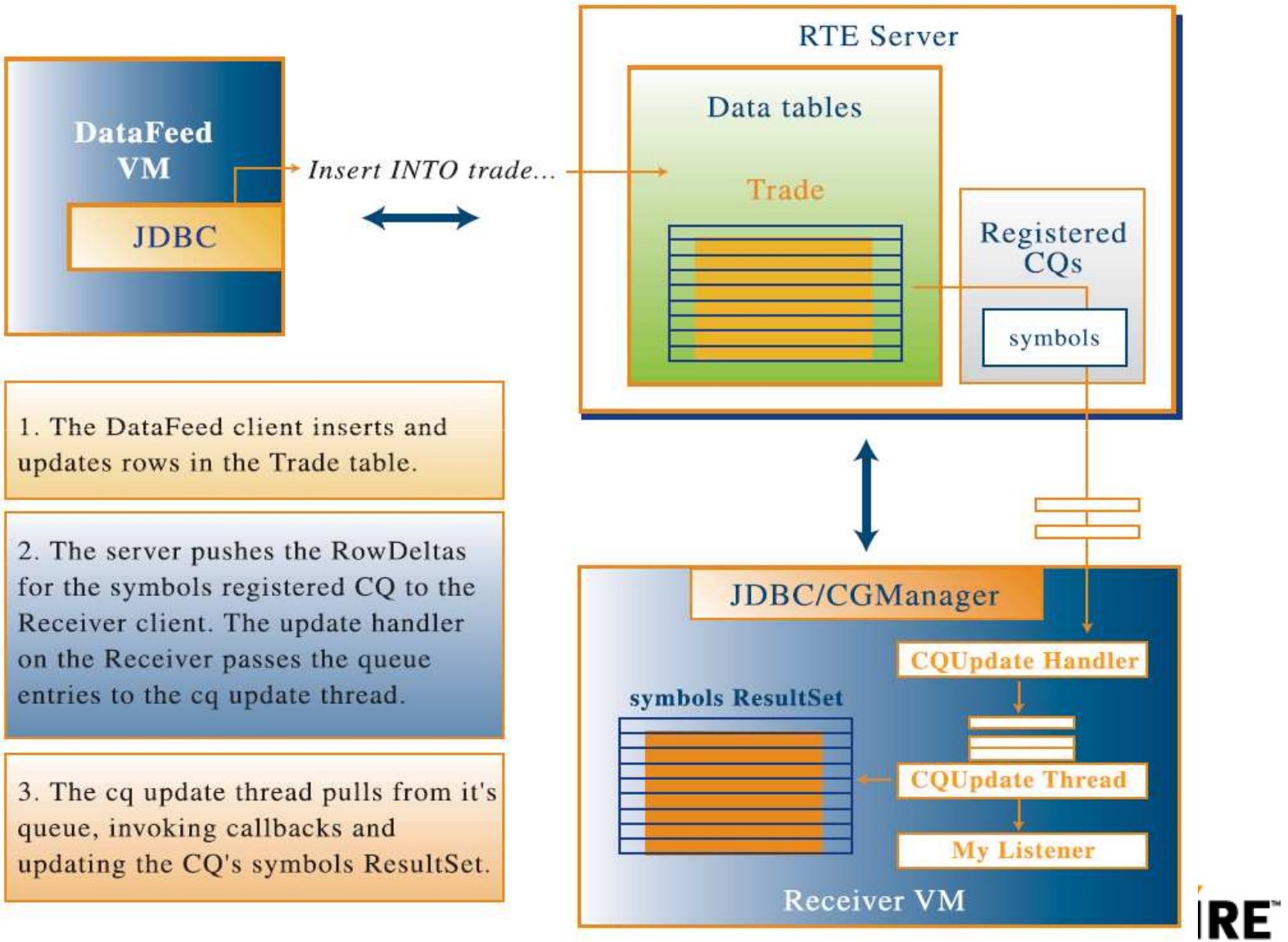
Subscribe using Continuous Queries (CQ)

- Queries resident and active
  - as if they are continuously running
- Client side view refreshed with configured latency
- *Maintain a continuous view of all Intel and Dell orders placed today and notify me when AMD moves up or down by 5%*



GEMFIRE
THE ENTERPRISE DATA FABRIC

Demo: Trades feeder, CQ receivers



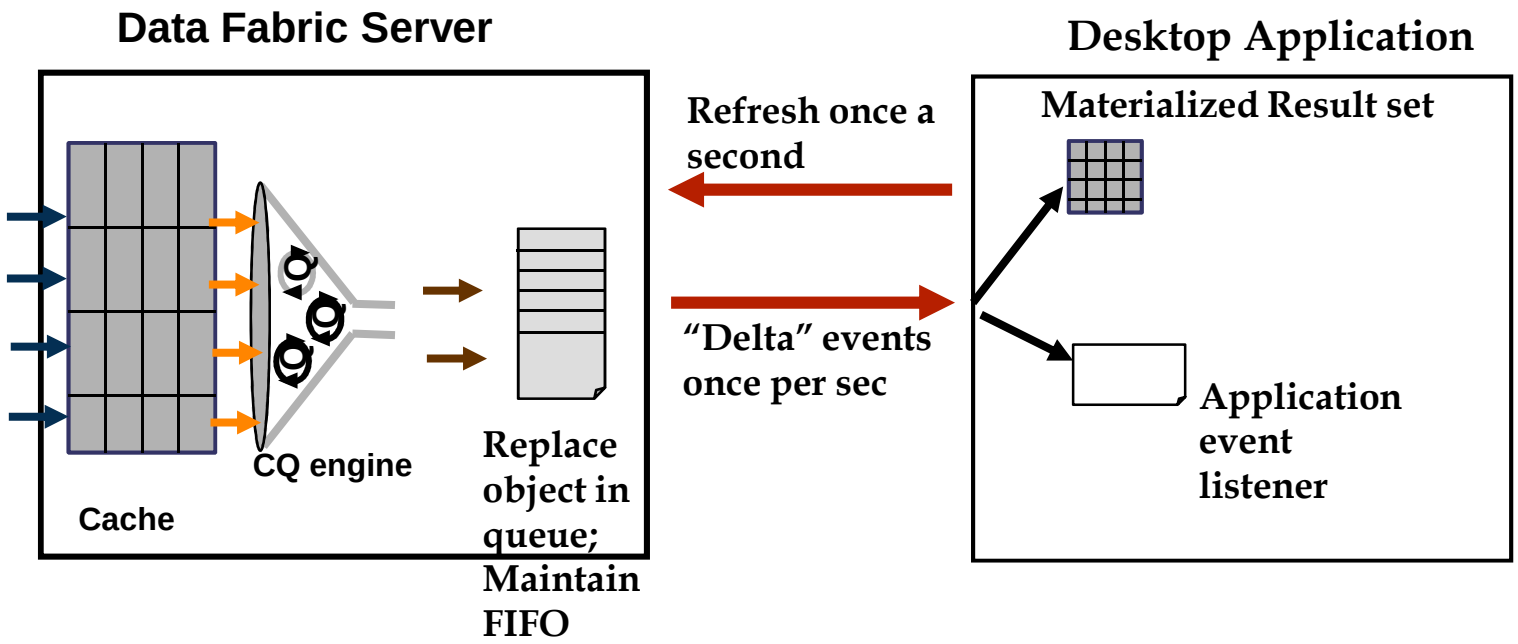
CQ Subscriber scalability

- Say, 1000 events pushed per sec with 100 queries registered
- Brute force: 100,000 queries per second is not possible
- Grouped predicate filtering
- View materialization – optimize natural joins
- Cluster scaling - queries distributed
- Async client dispatch, with batching and conflation

GEMFIRE™
THE ENTERPRISE DATA FABRIC

Event conflation

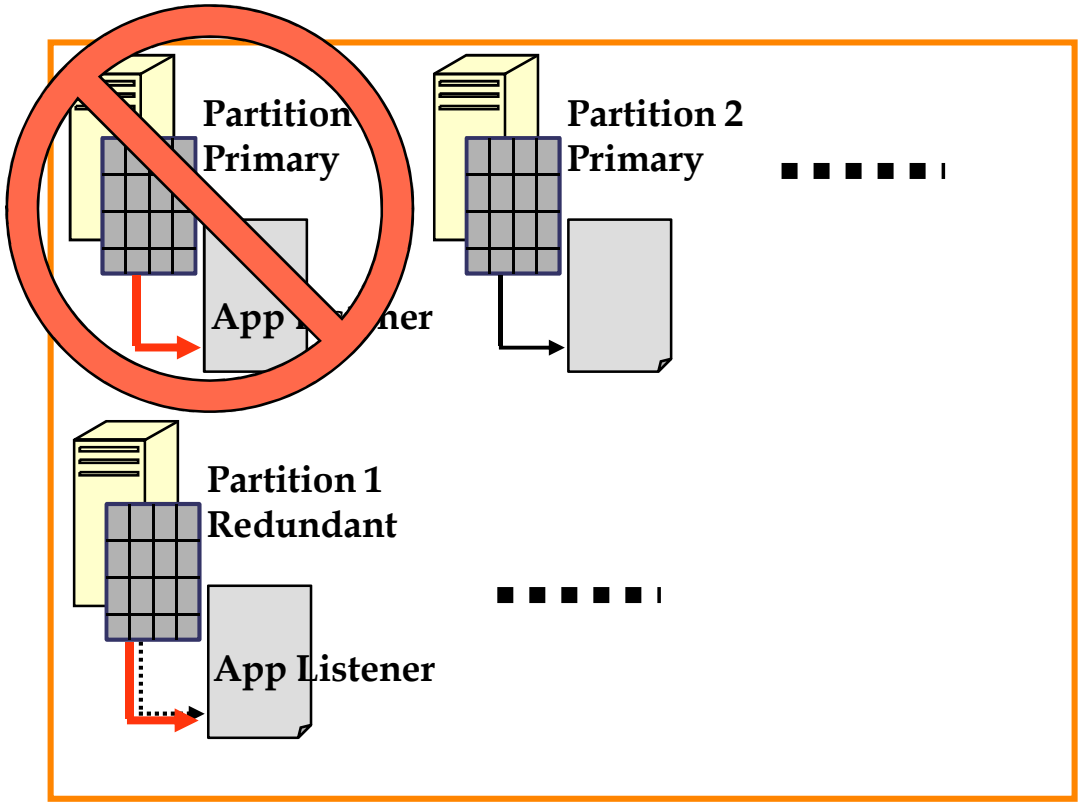
- Object state changes rapidly
- Clients only care about the latest object state
- But, need the view refreshed within say one second
  - e.g. any real-time desktop application



GEMFIRE™
THE ENTERPRISE DATA FABRIC

Distributed Event Processing with HA

- Events fired on nodes with primary data buckets – distributed
- Linearly scales with data collocation
- Event on redundant if primary fails
  - Event Listener callback indicates if event is possible duplicate



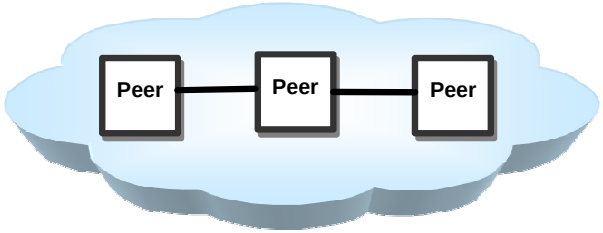
GEMFIRE™
THE ENTERPRISE DATA FABRIC

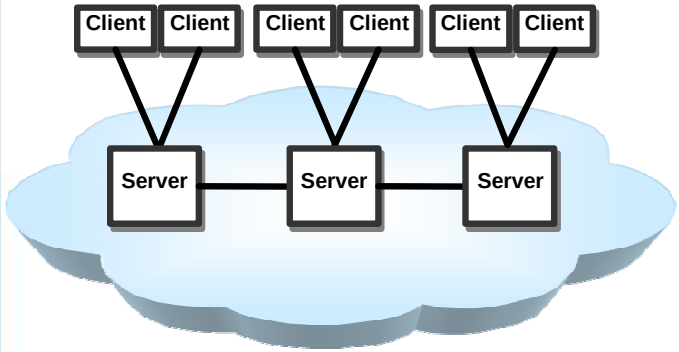
GEMFIRE™
THE ENTERPRISE
DATA FABRIC

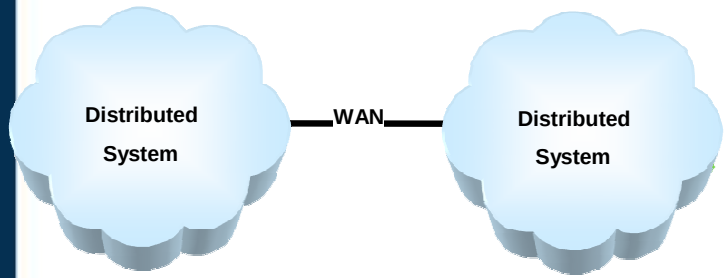
Q & A

GEMSTONE

Common deployment topologies







- Data fabric and compute engines are co-located (peer-2-peer data network)
  - When: limited parallelism, highly iterative tasks that operate on the same data set over and over
- Data managed on fabric servers
  - When: Many Jobs with unpredictable data access requirements, large data volume, data life cycle is independent of compute job lifecycle, data is changing constantly and data updates need to be synchronized to back-end databases, etc
  - Super peer architecture

Loosely coupled distributed systems

- partial or full replication
- data sets are partitioned across data centers

GEMFIRE™
THE ENTERPRISE DATA FABRIC