

# Parallel Replication



© Continuent 2011

**continuent**  
Open. Always Available.

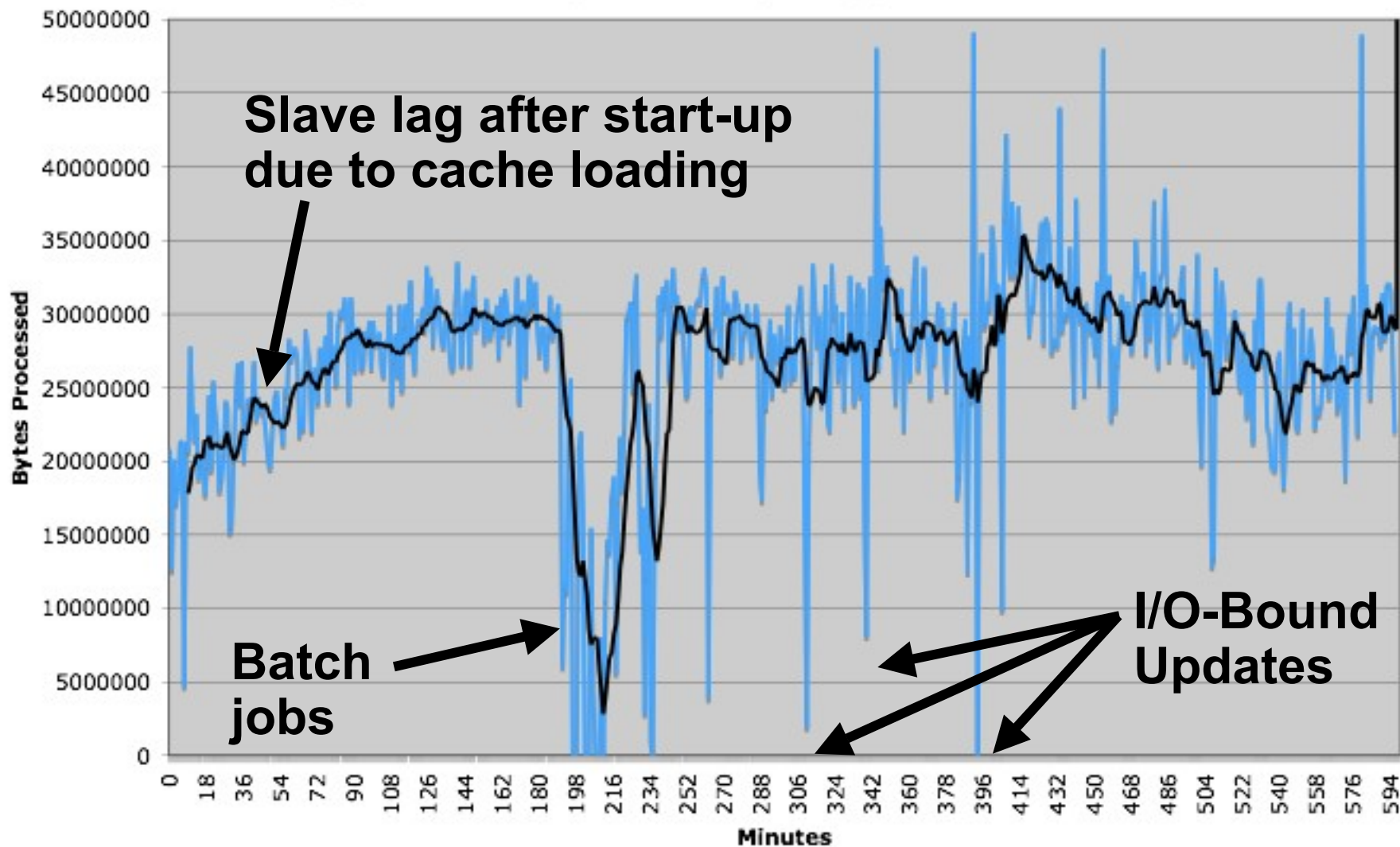
# Getting Started...

# Let's do a demo!



# Real-Life Replication Performance

Single-Threaded Replication -- Bytes Applied Per Minute



# Worst Scenarios for Slave Lag

## / **Slave catch-up**

- After provisioning from backup
- After maintenance/upgrade
- After mysqld restart

## / **I/O-bound slave workloads**

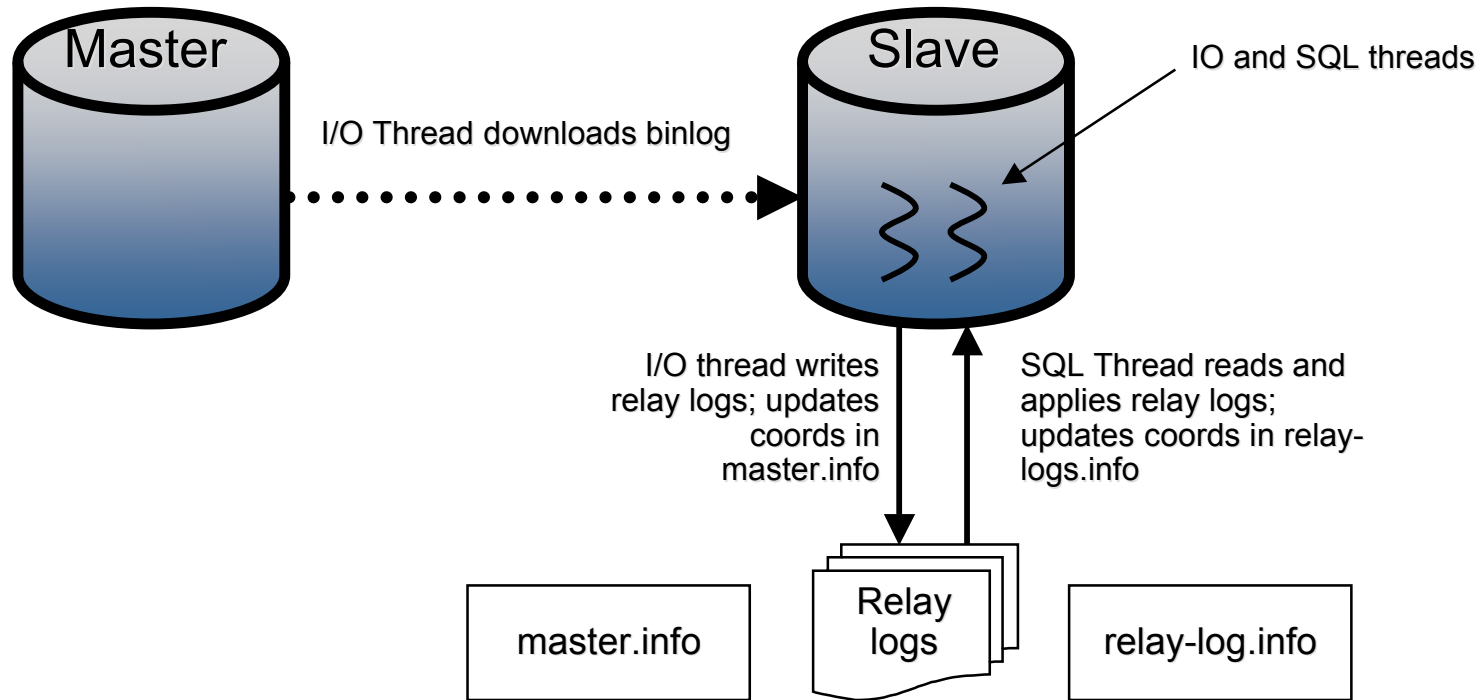
- Statements with WHERE clauses
- Row updates on large datasets
- DDL statements

## / **CPU-bound slave workloads**

## / **Bigger data sets have bigger problems**



# Root Cause: Single Threaded Replication

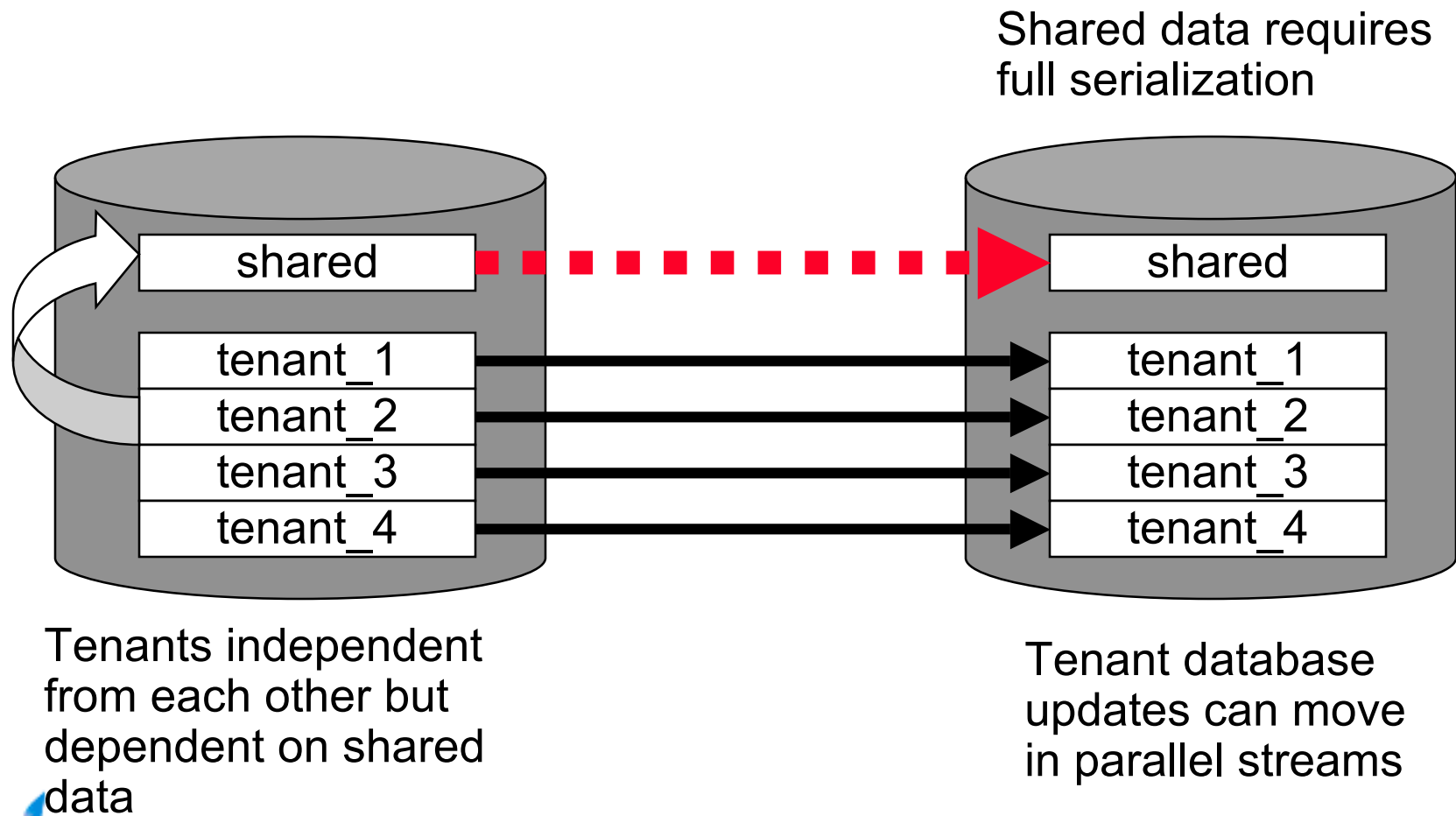


MySQL SQL thread is overwhelmed by CPU- or IO-bound queries



# Solution: Serialized Shards

- / Divide workload up-front into independent streams of updates (“shards”)



# Serialized Shards: Benefits/Drawbacks

## / On the good side

- Many workloads are naturally sharded (SaaS, social media apps, ISP shared DBMS, etc.)
- Solves problems of restart, deadlocks, conflicts, and transaction variability
- Can handle shard dependencies if declared beforehand

## / On the bad side

- Not all workloads are logically sharded
- Shard dependencies kill performance
- May require application changes



# How Does Tungsten Parallel Replication Work?



© Continuent 2011

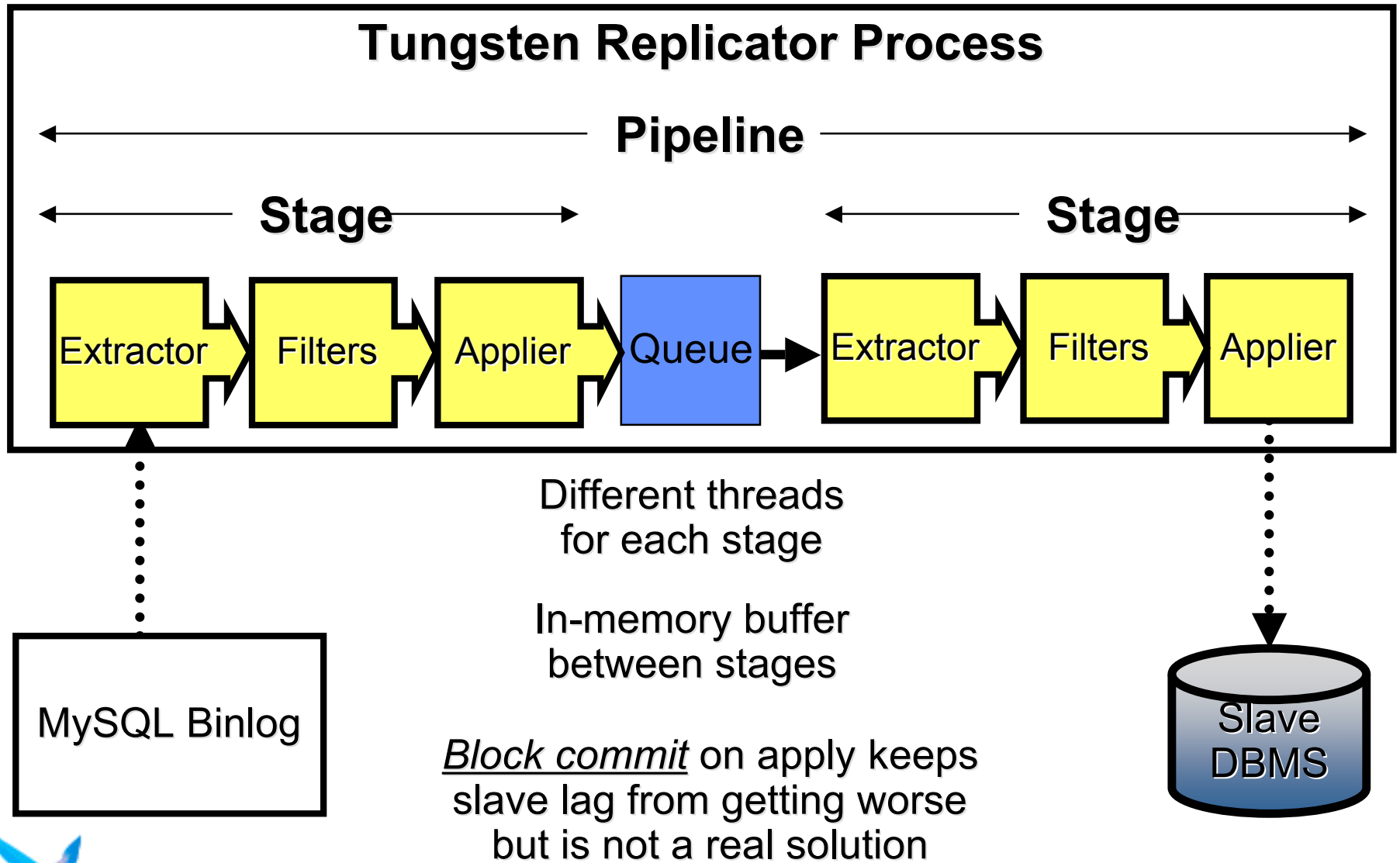
**continuent**  
Open. Always Available.

# Introducing Parallel Replication

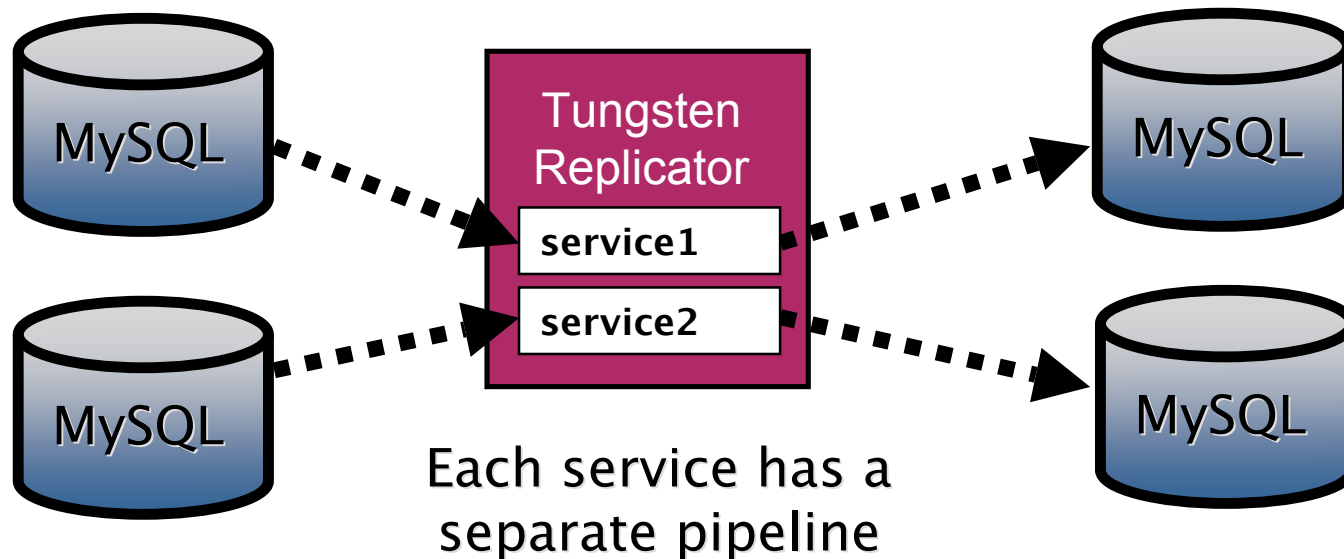
- / Tungsten optimizes resource utilization for large-scale data transfer
- / Pipeline parallelism - Spread work across stages
- / Service parallelism -- Spread work across different replicators and replication services within replicators
- / Parallel apply -- Spread apply operations across multiple threads



# Pipelines and Stages



# Multiple Services within One Replicator



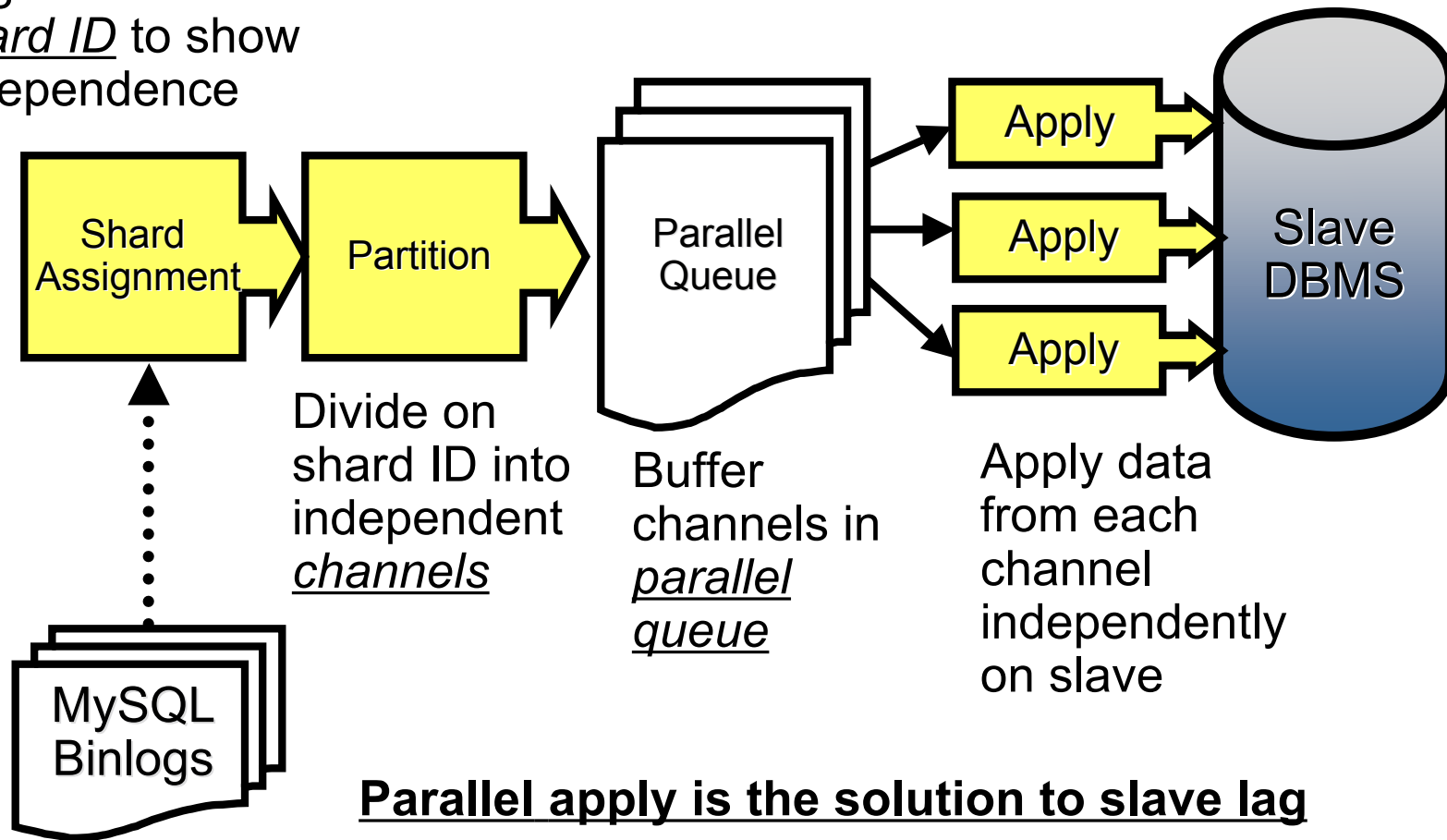
Multiple services  
amortize Java foot print

They are very useful  
but do not solve slave  
lag problems

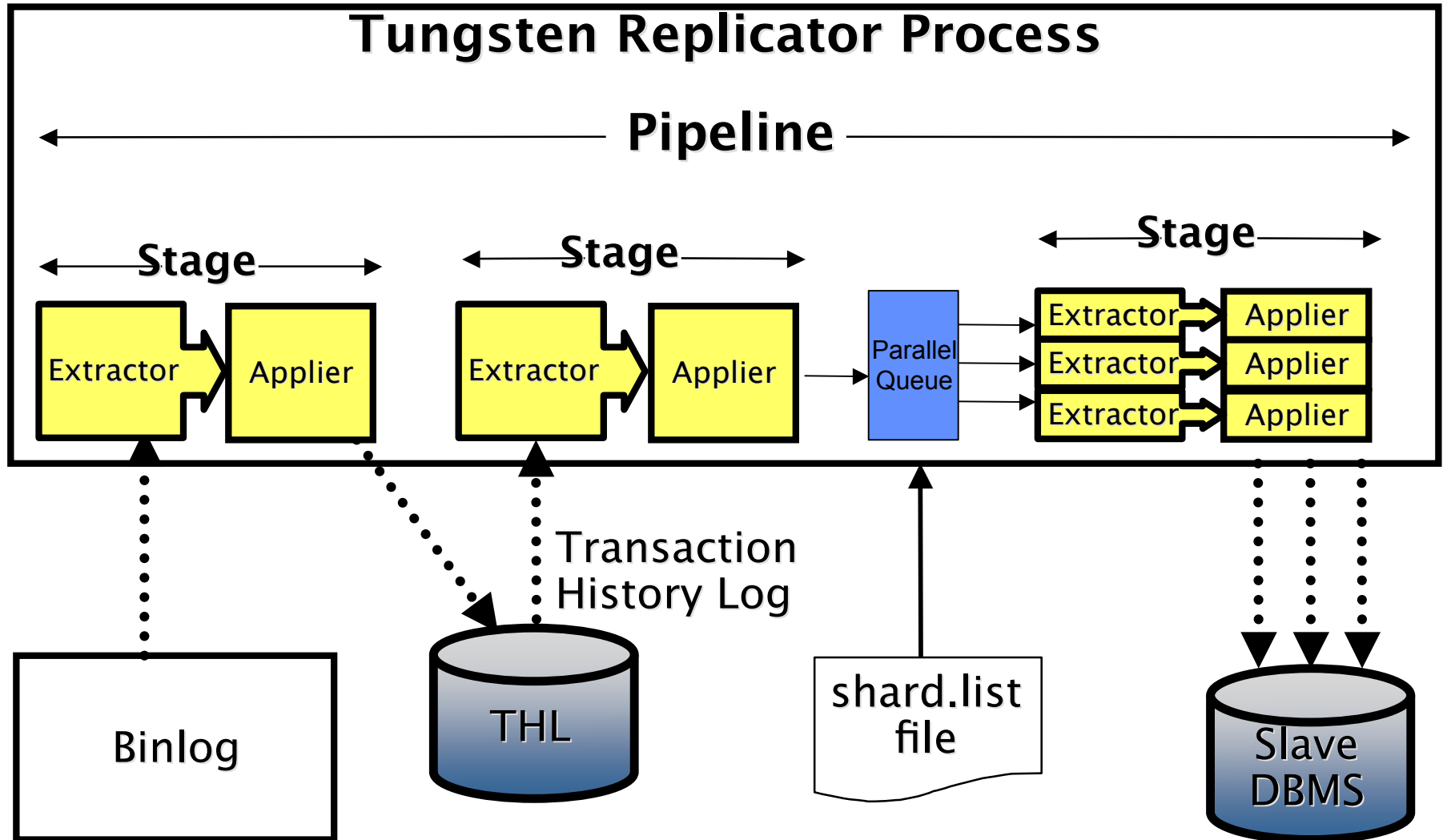


# Parallel Slave Apply - Concepts

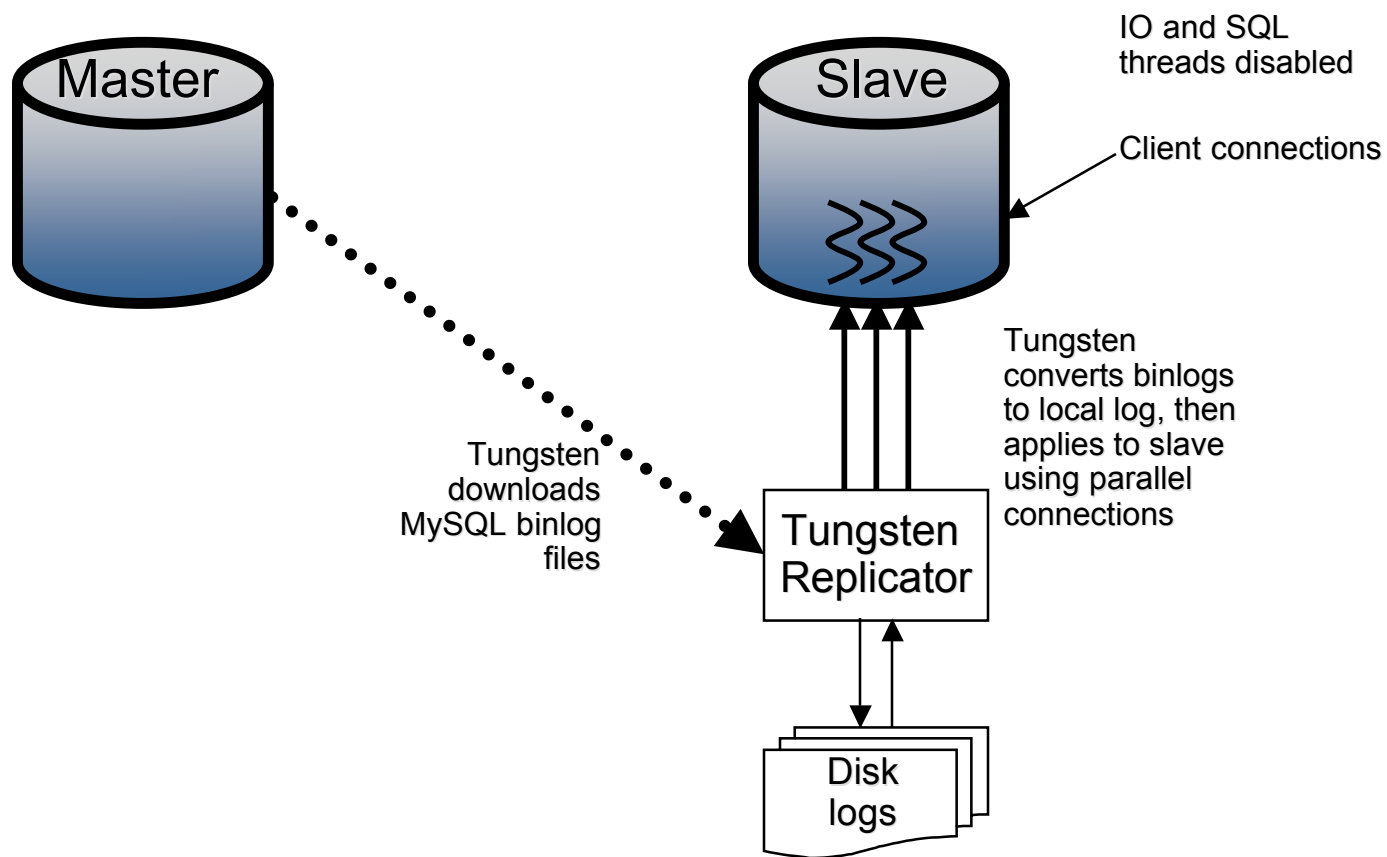
Tag transactions with  
shard ID to show  
independence



# Parallel Slave Apply - Pipeline

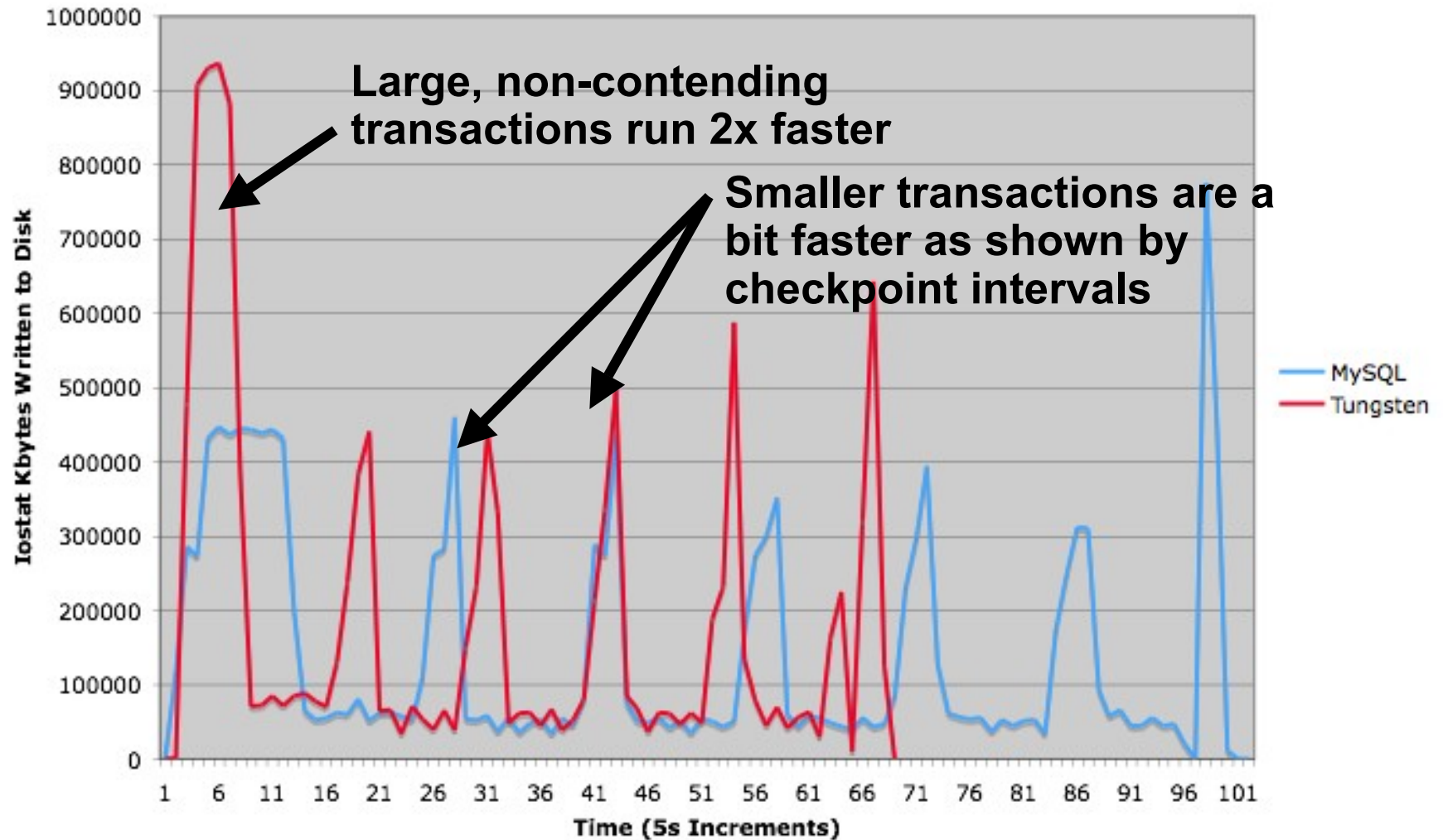


# Example of Use: MySQL Parallel Slave



# Sysbench Performance Results

## MySQL vs. Tungsten Parallel Replication



# Parallel Apply Gotchas

## / **Too little serialization causes errors**

- Dependent updates must go in the same channel or be serialized
- Declare shared databases in shards.list
- Lock wait timeouts are key sign of serialization problems

## / **Too much serialization kills performance**

- Check serialization count in parallel queue

## / **Poor shard distribution lowers performance**

- Look for even distribution across parallel queues
- Look for even distribution of shard event counts
- Hash channel assignment can be uneven
- Assign shards explicitly to channels using shards.list



# Managing and Tuning Parallel Replication

## More Demonstrations



# Parallel Replication Worst Cases

## / **Single large database**

- Difficult to parallelize transactions easily

## / **Multi-tenant app with lots of shared data**

- Have to serialize shared database updates

## / **Small databases with fast transactions**

- Data size  $\leq$  buffer cache
- Few I/O bound updates

## / **“Economy-class” hardware or VMs**

- Adding more threads does not help if one thread blocks the disk

**Minimal or no performance gain**



# Parallel Replication Best Cases

## / **Multi-tenant applications**

- Independent customer databases
- Minimal or no shared data
- Uniform distribution of updates across schemas

## / **Large data sets**

- Data size  $\gg$  buffer cache
- Lots of I/O bound queries

## / **Big iron**

- Fast I/O subsystems that can absorb parallel updates

**Look for 2-3X performance  
increases in real workloads**



# Parallel Replication Roadmap

- / **Fix bugs**
- / **Buffered I/O for THL and MySQL extractor**
- / **Implement ParallelStore directly on THL**
- / **Round-robin partitioning using database name**
- / **Improve SQL parsing and shard detection**
- / **Handle restart corner cases better**

