DXGraph: Large-scale Graph Processing Based on a Distributed In-memory Key Value Store

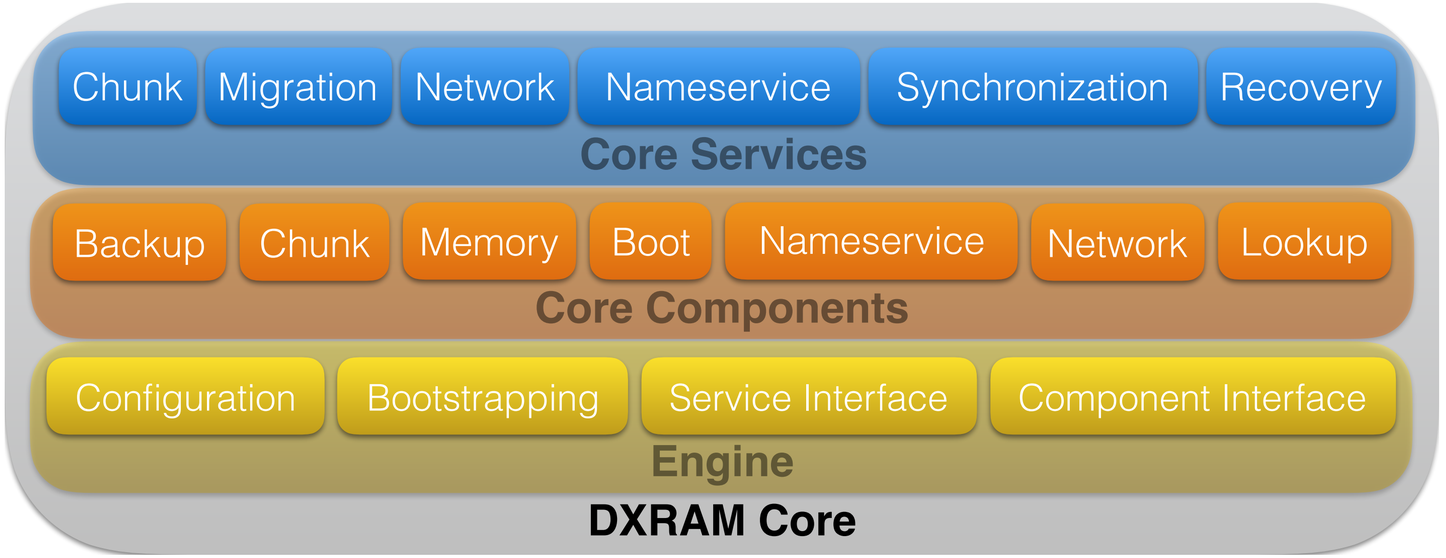
Stefan Nothaas, Kevin Beineke, Michael Schöttner
Department of Computer Science, Heinrich-Heine-University Düsseldorf, Germany

March 3rd 2017, Reisensburg

DXRAM - Architecture

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

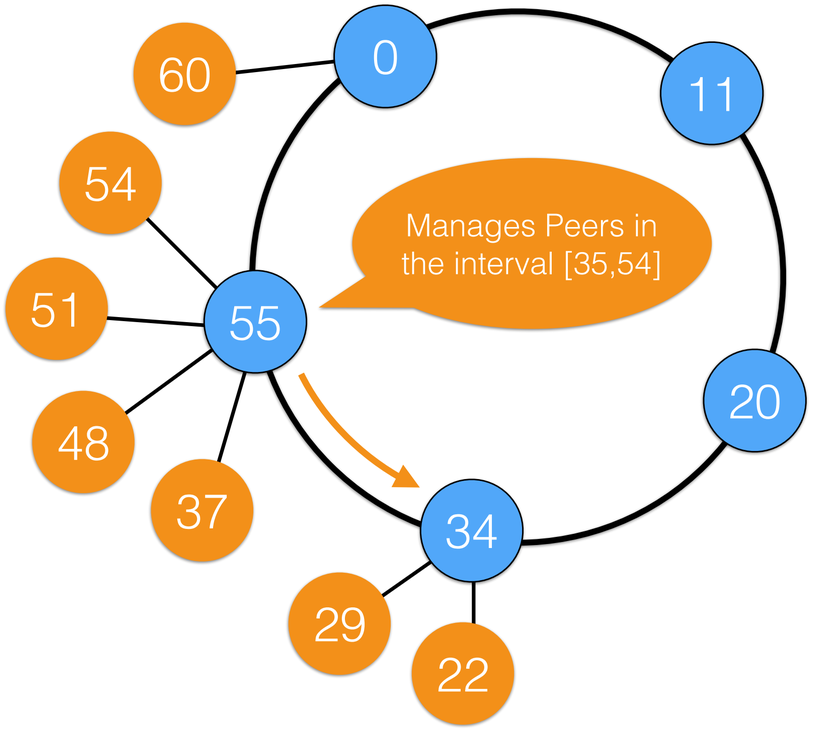
- DXRAM Core
  - Engine
  - Components
  - Services (API)
- Custom applications



DXRAM - Node Types

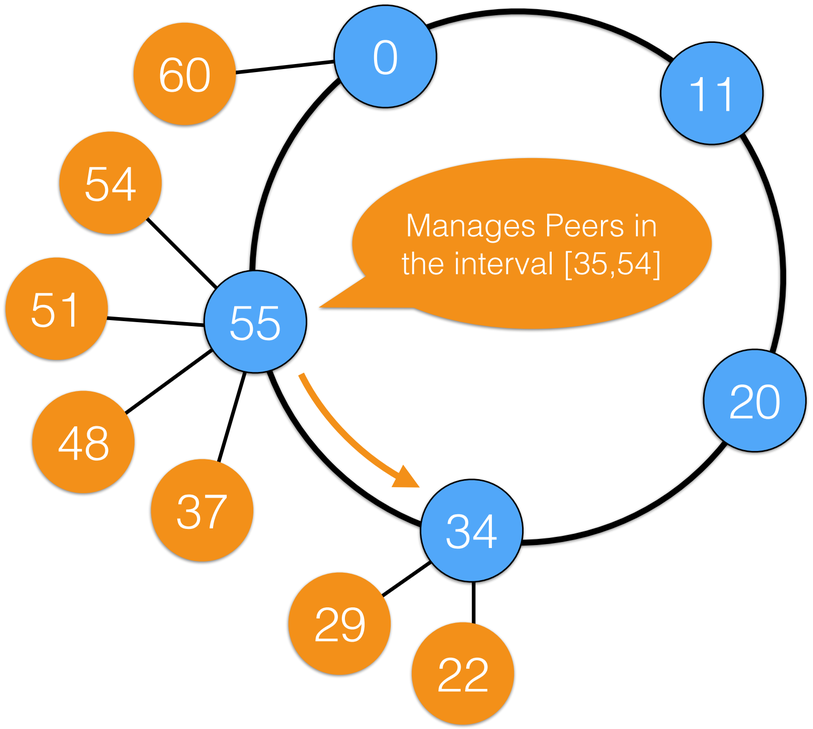
HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

- **Superpeer** Overlay
  - Fast node lookup with custom Chord-like overlay
  - Superpeers do not store chunks but all global meta-data (modified B-Tree)
  - Meta-data replicated on successors
  - 5 to 10% of all nodes are superpeers
  - Every superpeer knows every other superpeer



DXRAM - Node Types

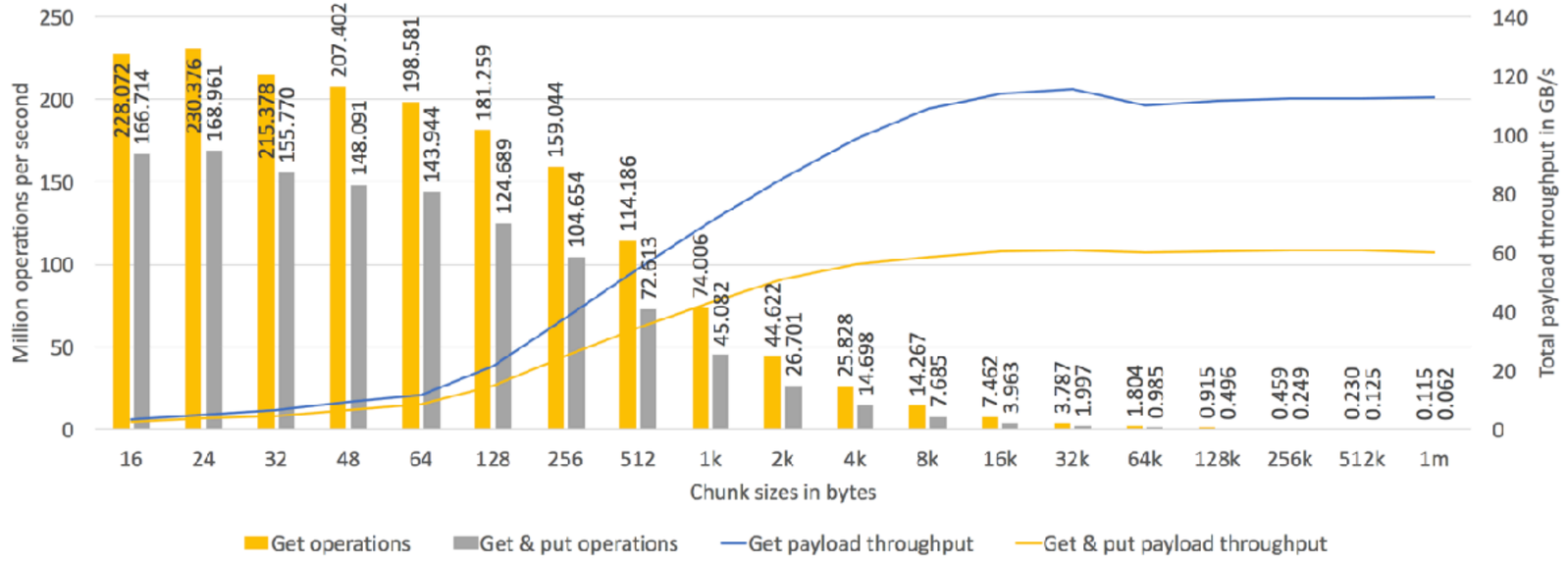
- **Peers** store chunks
  - Every peer is assigned to one superpeer
  - Key: 64 bit globally unique sequential chunk ID (CID)
  - Value: Byte buffer



DXRAM - Memory Management

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

- Custom allocator designed for many small chunks
- Operations: Create, get, put, remove
- Optimized for high concurrency
  - get (16 byte): 228 million ops/sec ⇒ 3.4 GB/sec
  - get & put (16 byte): 116 million ops/sec ⇒ 2.5 GB/sec

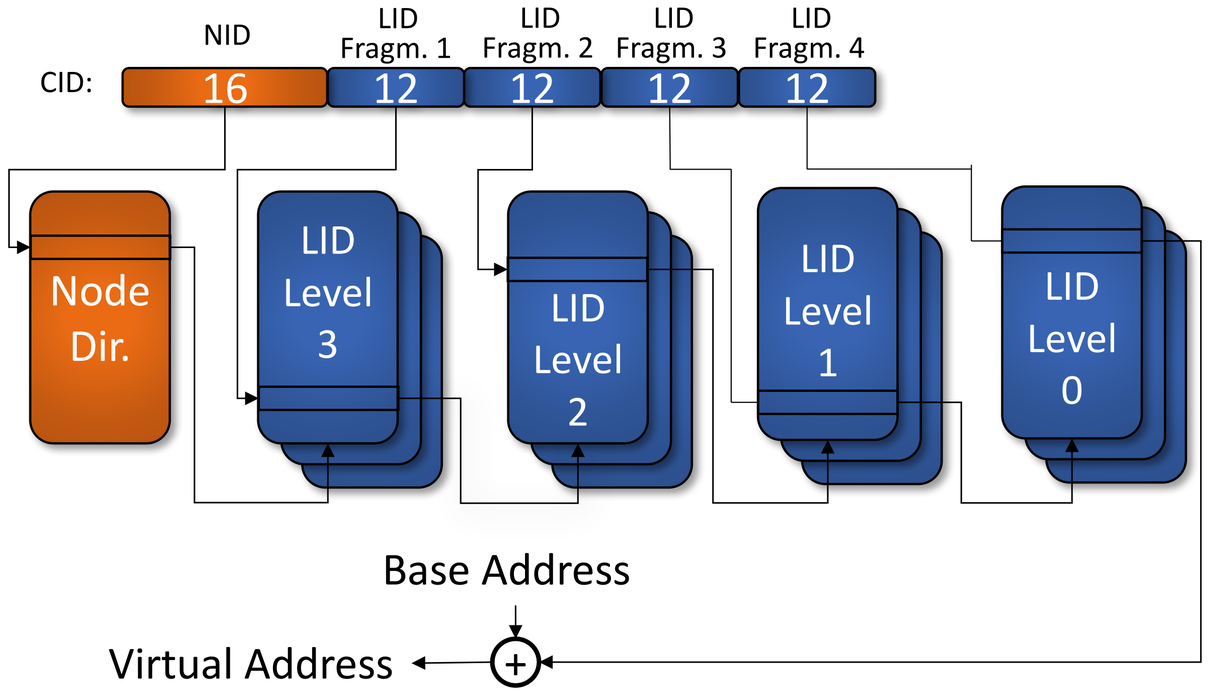


DXRAM - Address Translation

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Paging like address translation

- Chunk location lookup in O(1)
- Tables created on demand



- Average metadata overhead ~5% (avg. payload size: 64 bytes)
- Example: 64 GB for key-value store ⇒ ~1 billion chunks per node

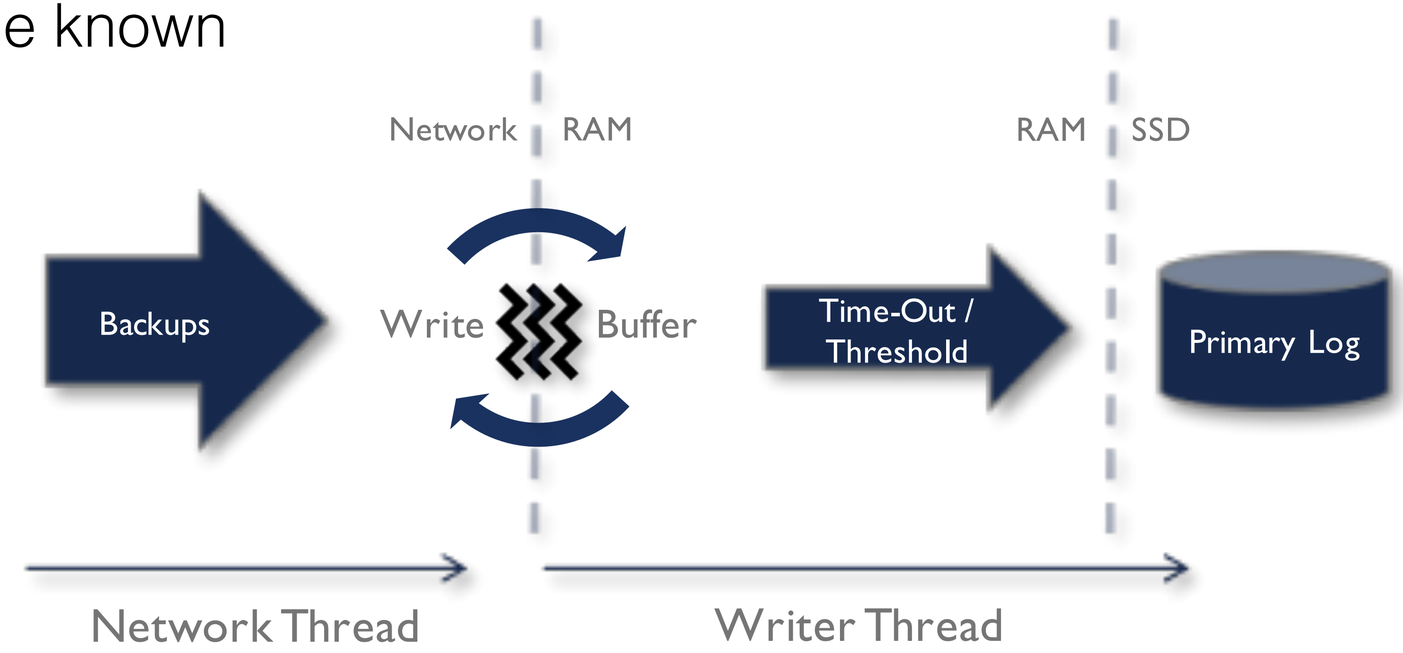
DXRAM - Logging

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Storing replicas on remote peers, challenges

- Replication in RAM too expensive
- Update in-place on SSD
  - Writing small objects randomly is very slow
  - Locations of all objects must be known

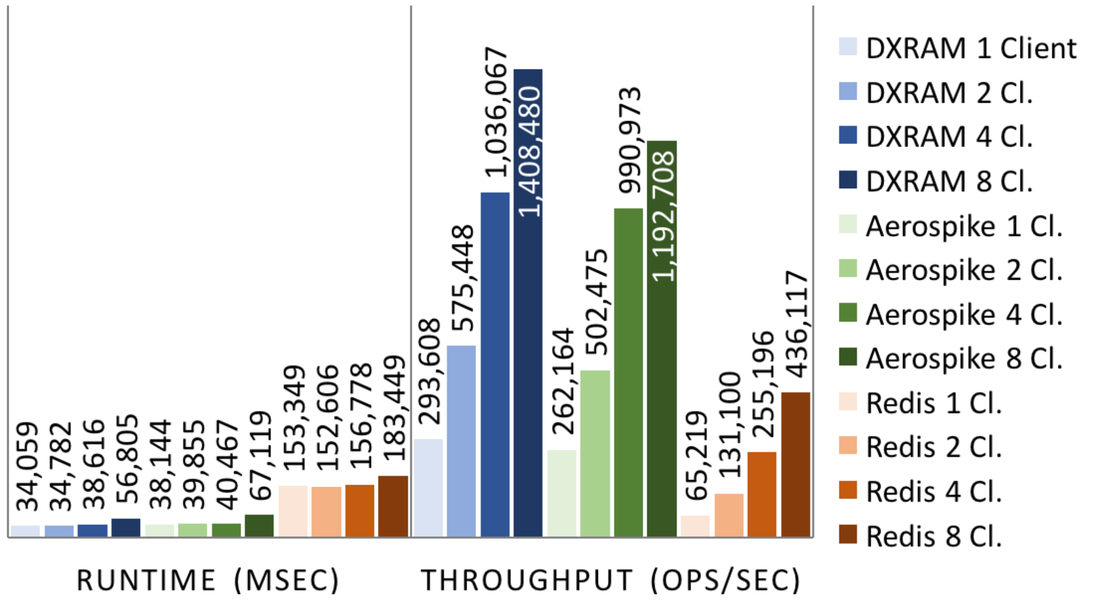
- Append data to a log
  - Best SSD utilization
  - Low RAM consumption
  - Requires reorganization and version control



DXRAM - Logging

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

- Reorganization is necessary to free space for further updates
  - Novel version control
    - Epoch based (combining caching and writing to SSD)
- 2-level logging
  - Low memory footprint and high throughput
- Fast parallel recovery ⇒ < 1 sec
- Yahoo! Cloud Serving Benchmark
  - Comparing DXRAM to Aerospike and Redis



DXRAM - Foundation for Graph Computation

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

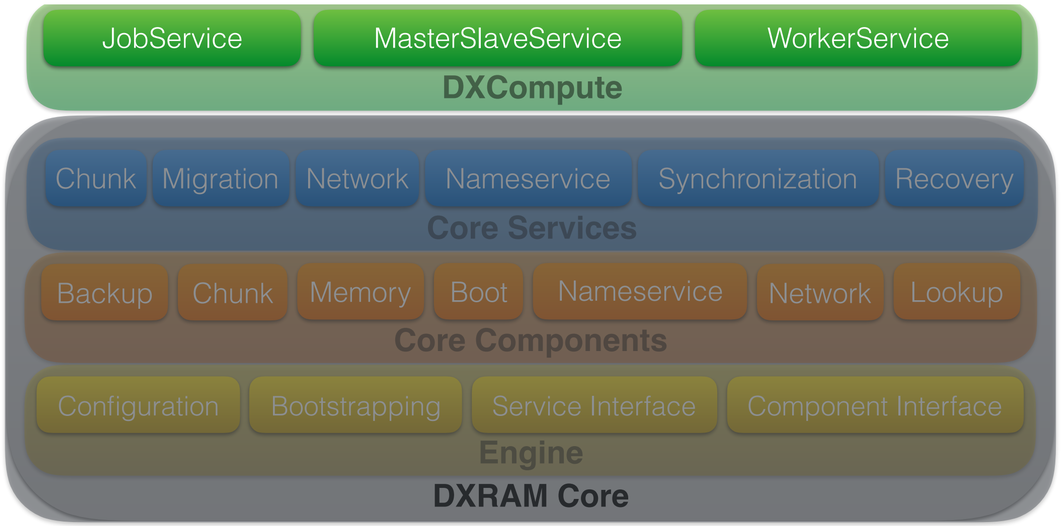
- DXRAM provides
  - Low latency
  - Scalability
  - Efficient handling of small objects

⇒ Foundation for graph processing

- What else do we need for graph processing?
  - Utilize CPU resources on storage nodes
  - Move computations to data ⇒ locality

DXCompute

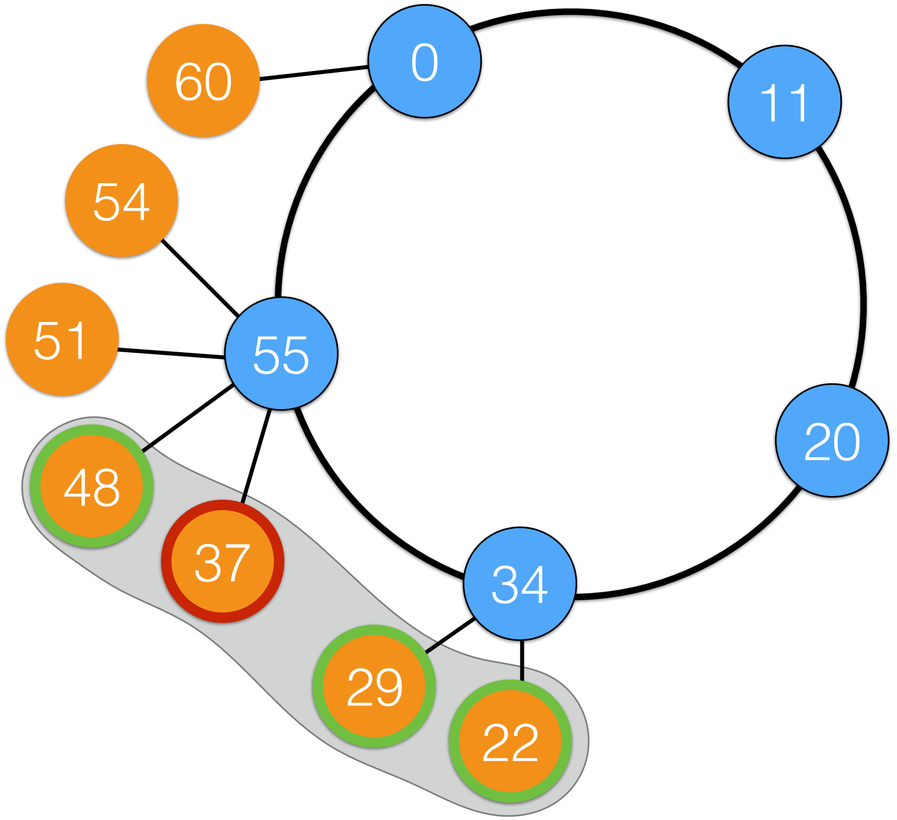
- Extends DXRAM Core
- Services to run computations on peers
- Benefit from locally stored chunks
- **JobService**
  - Deploy light weight jobs to single nodes
  - Scheduling by work stealing
- **MasterSlaveService**
  - Aggregate nodes to compute groups
  - Deploy compute tasks to group



DXCompute - MasterSlaveService

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

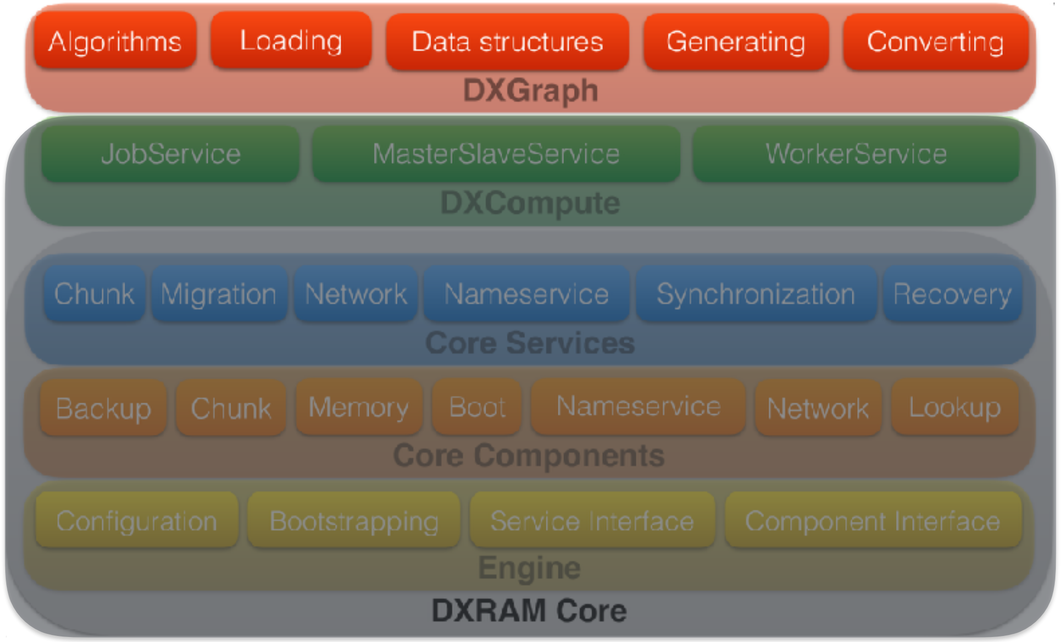
- Peers form a compute group
- Master: one peer as coordinator
- Slaves: further peers as distributed workers
- Tasks are submitted to compute groups
- Groups can grow
- Access to other nodes outside group (storage)
- Task context on execution
  - Compute group ID
  - Own slave ID
  - List of node IDs of every other slave
  - Total number of slaves



DXGraph

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

- DXGraph extends DXCompute
- Uses JobService or MasterSlaveService
- Algorithms for graph processing
- Graph data loading
- Natural representation of graph data as objects: Vertex, Edge, Attribute



DXGraph - Breadth-First-Search

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

- Implementation as specified by the Graph500 benchmark
- Stress test for system: Highly random access
- Standard top-down combined with bottom-up approach (reducing number of visited vertices)
- Compute task: Implements BFS
  - Distributed and multithreaded implementation
  - Delegates processing of non local vertices to owner node
  - Lock-free bitmap based frontier data structure
  - Low overhead synchronization between BFS levels

DXGraph's BFS on Hilbert

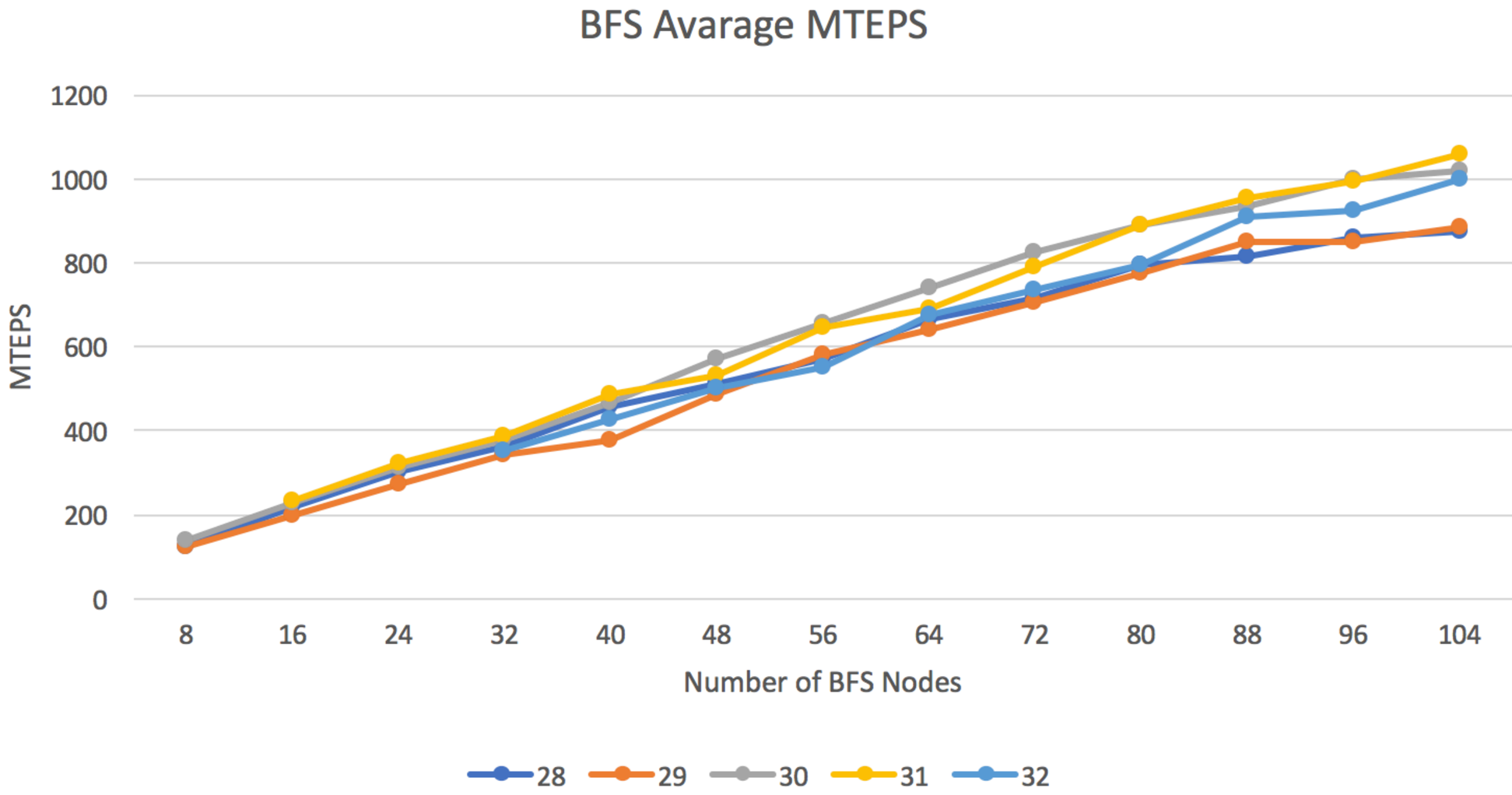
HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

- HPC system of our university:
  - BULL: Cluster architecture, 112 nodes with 24 cores and 128 GB RAM each
- Running DXGraph's BFS implementation on BULL cluster
- Goals: Scalability, Low memory overhead ⇒ storing many small objects

- Graph sizes tested: Scale 28 (64 GB) to 32 (1 TB)
- Random but equally distributed to 8 to 104 compute nodes

DXGraph's BFS on Hilbert - Results

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF



Conclusions & Outlook

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

- Conclusions
  - DXRAM: Distributed in-memory key-value store for many small objects
    - ~ 5% metadata overhead, get (16 byte): 228 million ops/sec ⇒ 3.4 GB/sec
    - Outstanding logging performance, especially with objects <= 100 bytes
  - DXGraph: Fast and scalable BFS implementation on 104 nodes
    - Graph: 1 TB, ~4.3 billion vertices, ~137 billion edges
    - Double the nodes ⟹ Half the execution time
    - Up to 1 billion traversed edges per second
- Outlook
  - InfiniBand