Relaxed Synchronization and Eager Scheduling for Map-Reduce

Ananth Grama
Suresh Jagannathan

Dept. of Computer Science
Purdue University

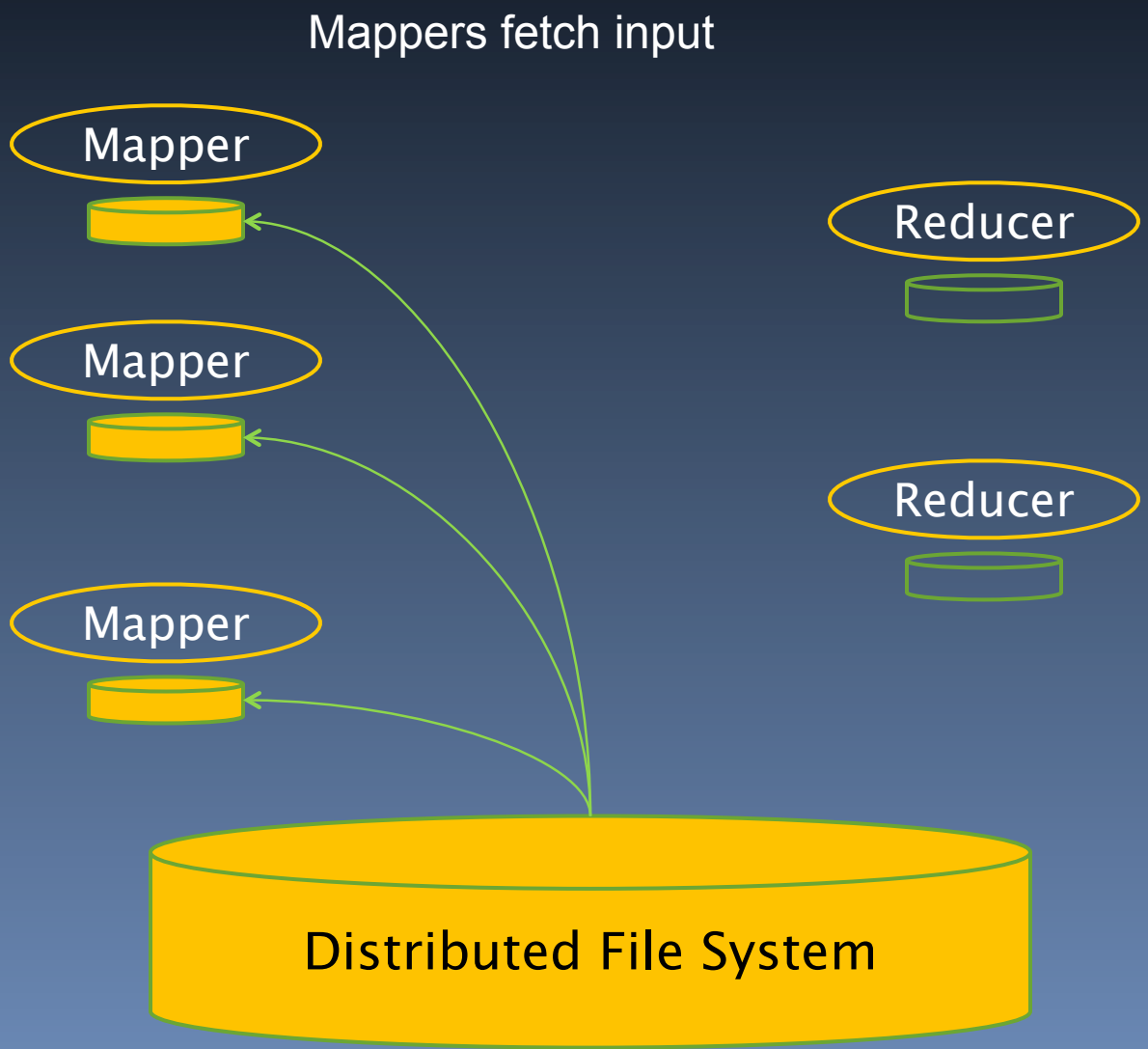
Goals of this project

- How can MapReduce be used to
  - support non-data parallel applications
  - specifically, **unstructured sparse graph algorithms**
    - social and biological networks
    - scientific datasets
- Enhancements to improve performance and scalability
  - without compromising
    - programmability
    - fault-tolerance

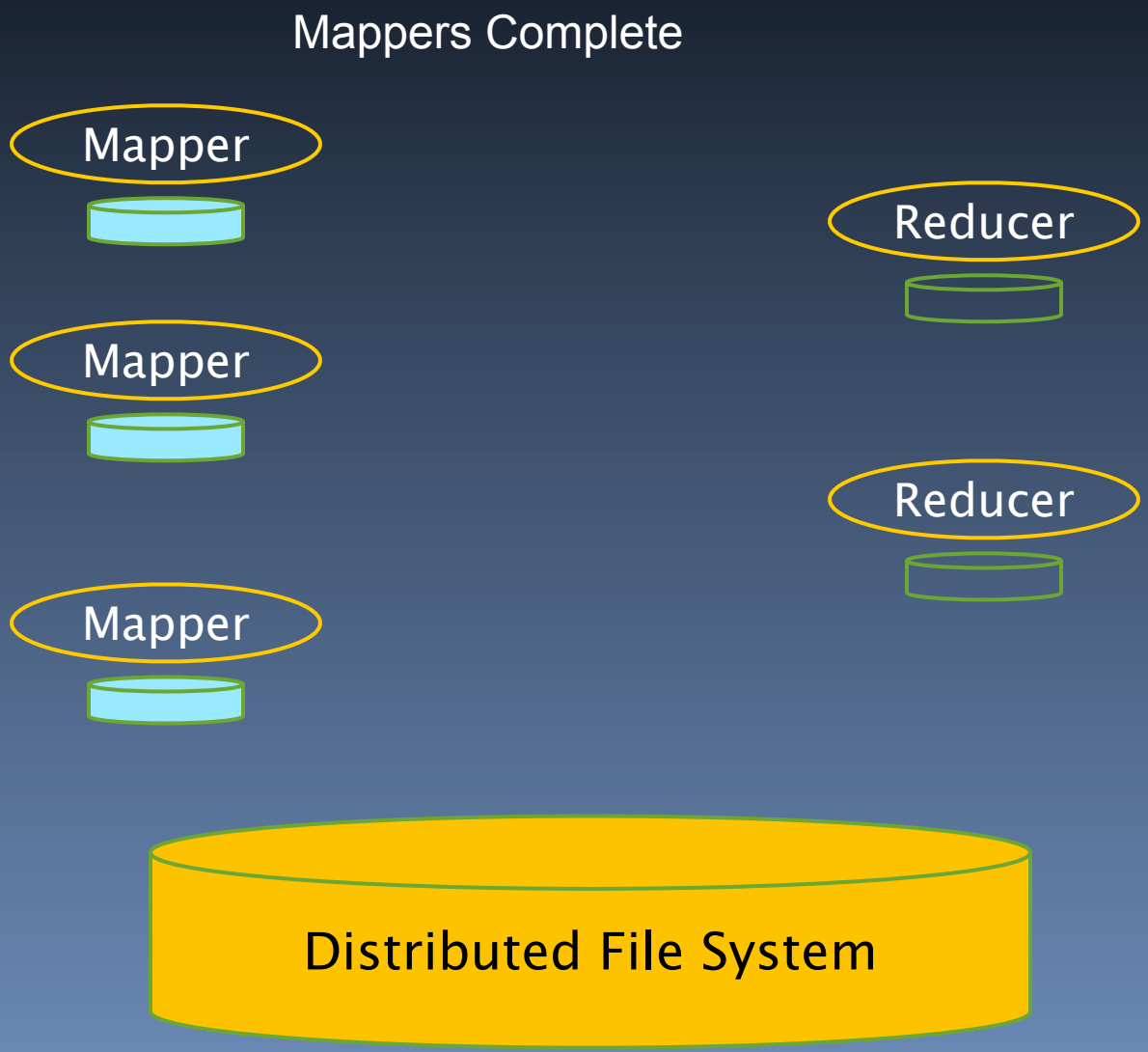
Observations

- Global reduction is a bottleneck for non-uniform algorithms
  - Often not necessary for correctness
- Tradeoff serial operation counts for partial reductions
  - locality enhancing mechanisms
    - natural graph partitioning
  - consistent with application semantics
- Show notable performance gains

MapReduce: Data Flow

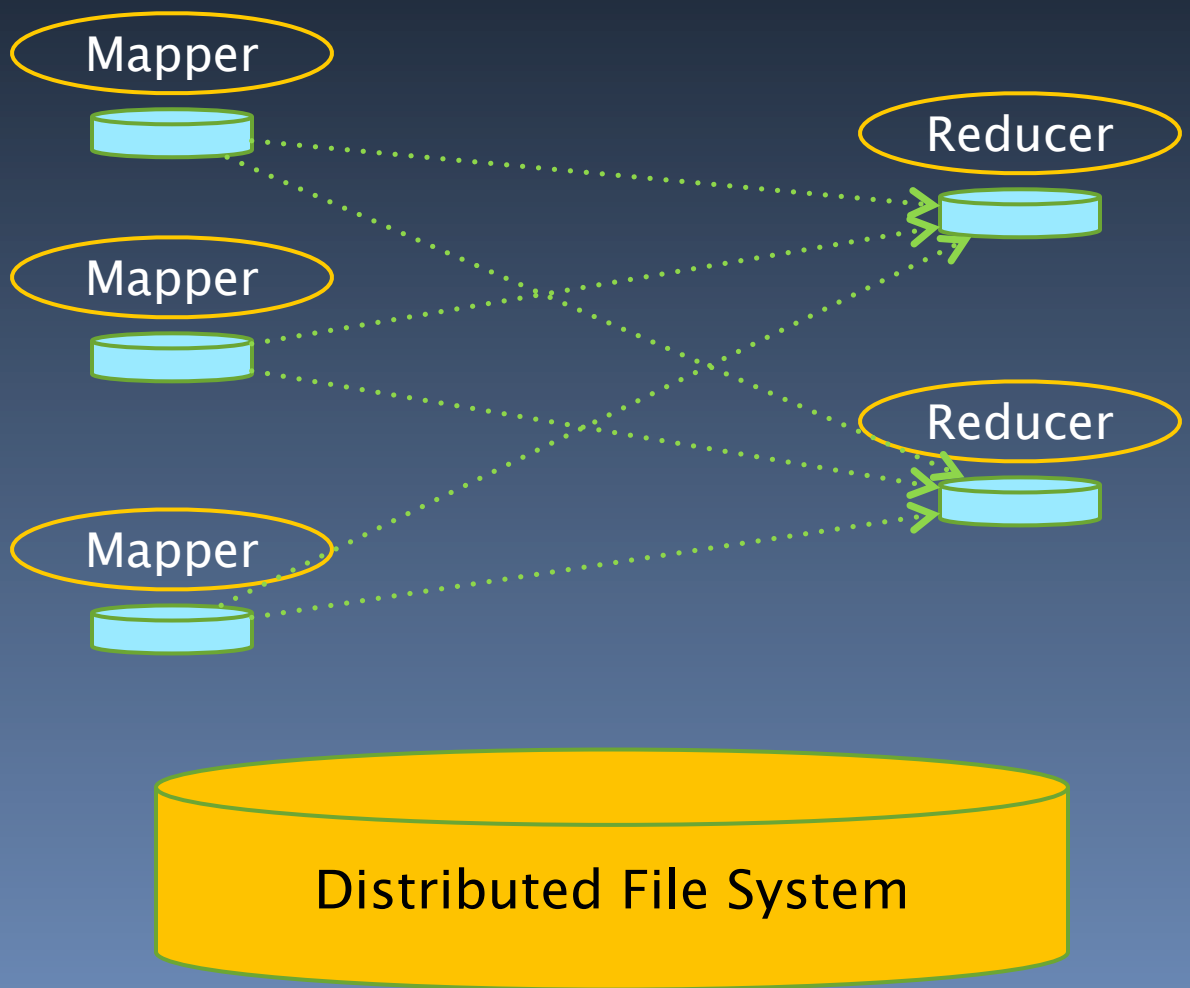


MapReduce: Data Flow

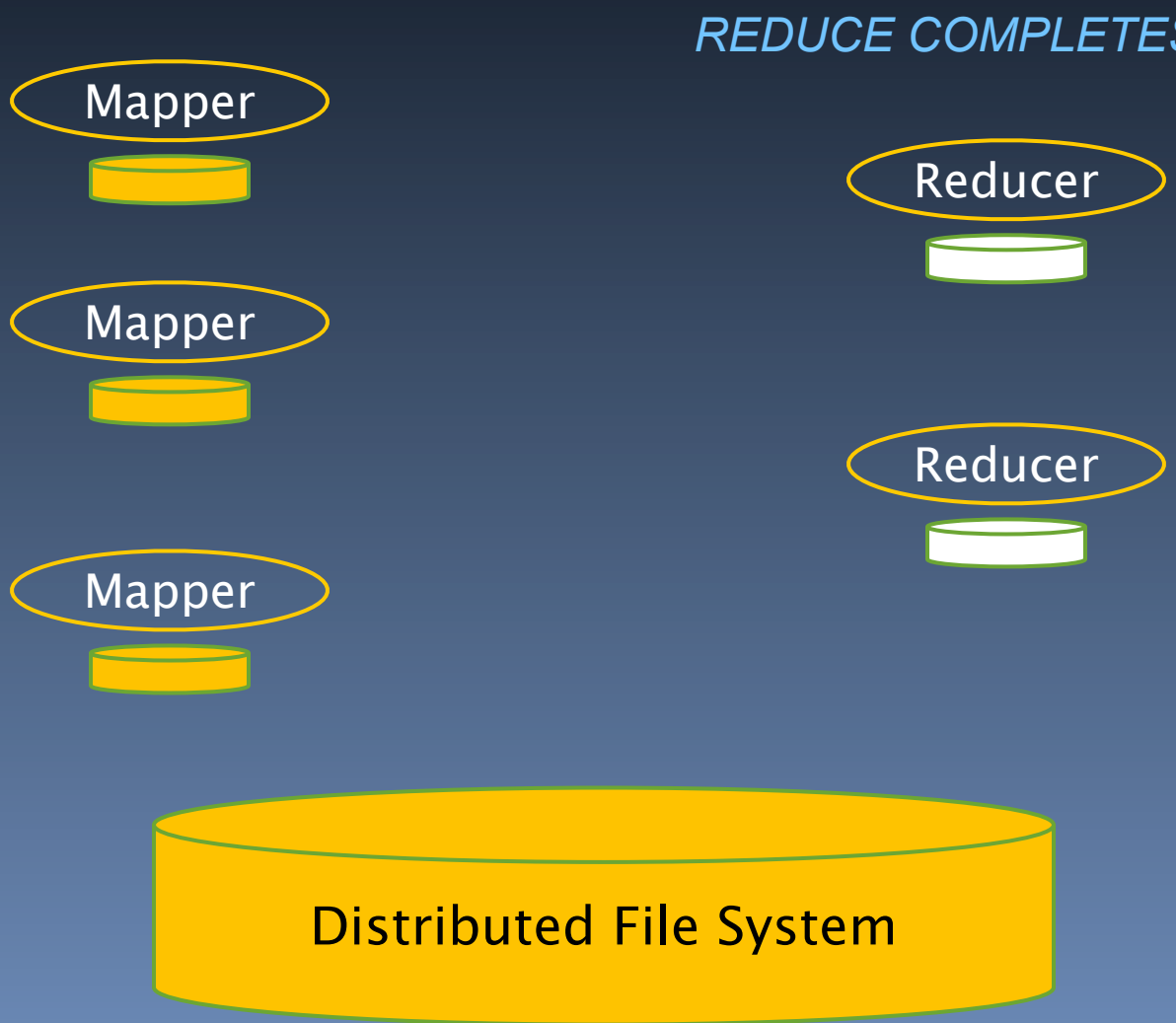


MapReduce: Data Flow

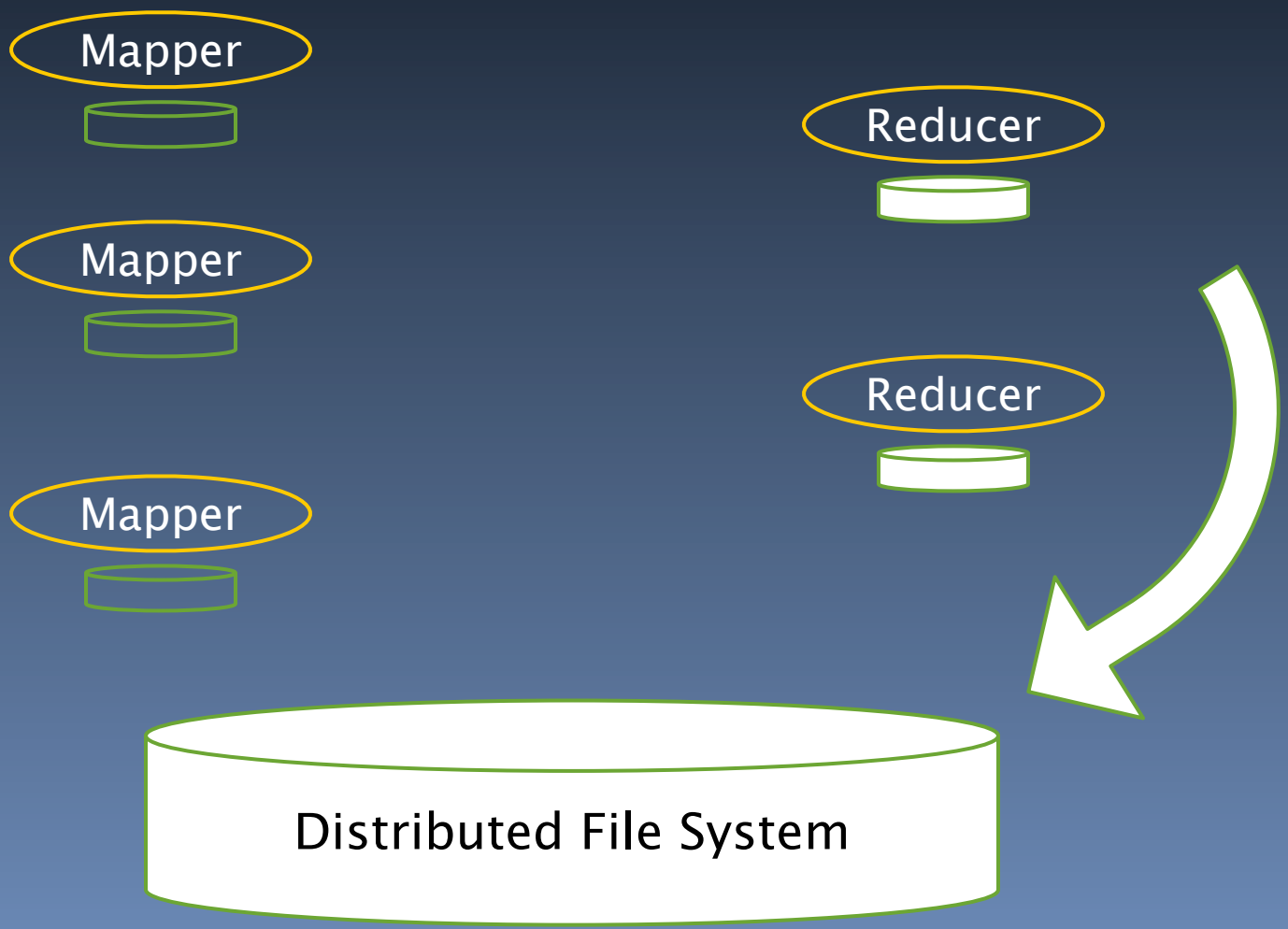
Reducers pull data from Mappers



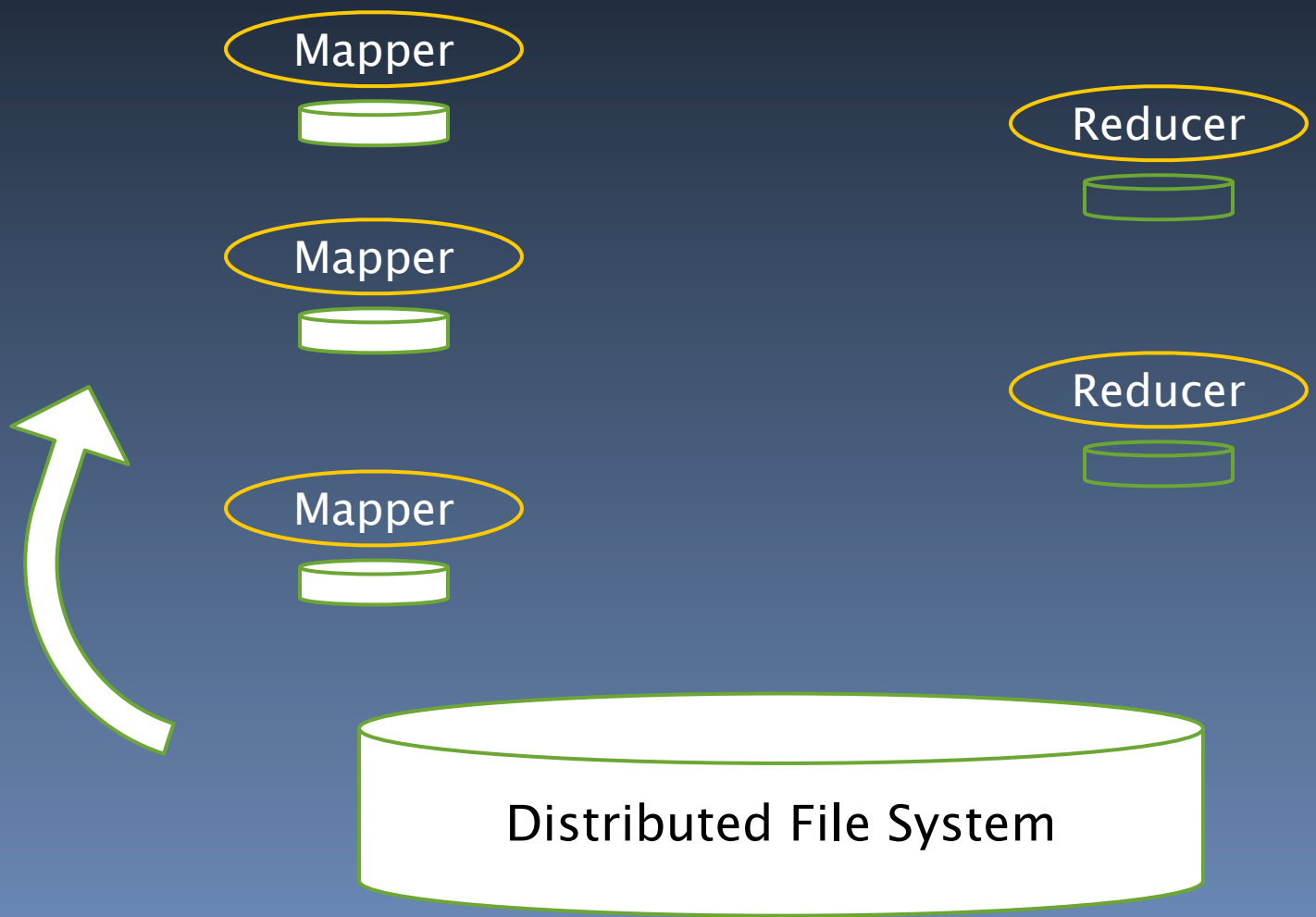
MapReduce: Data Flow



MapReduce: Data Flow



MapReduce: Data Flow



Characteristics

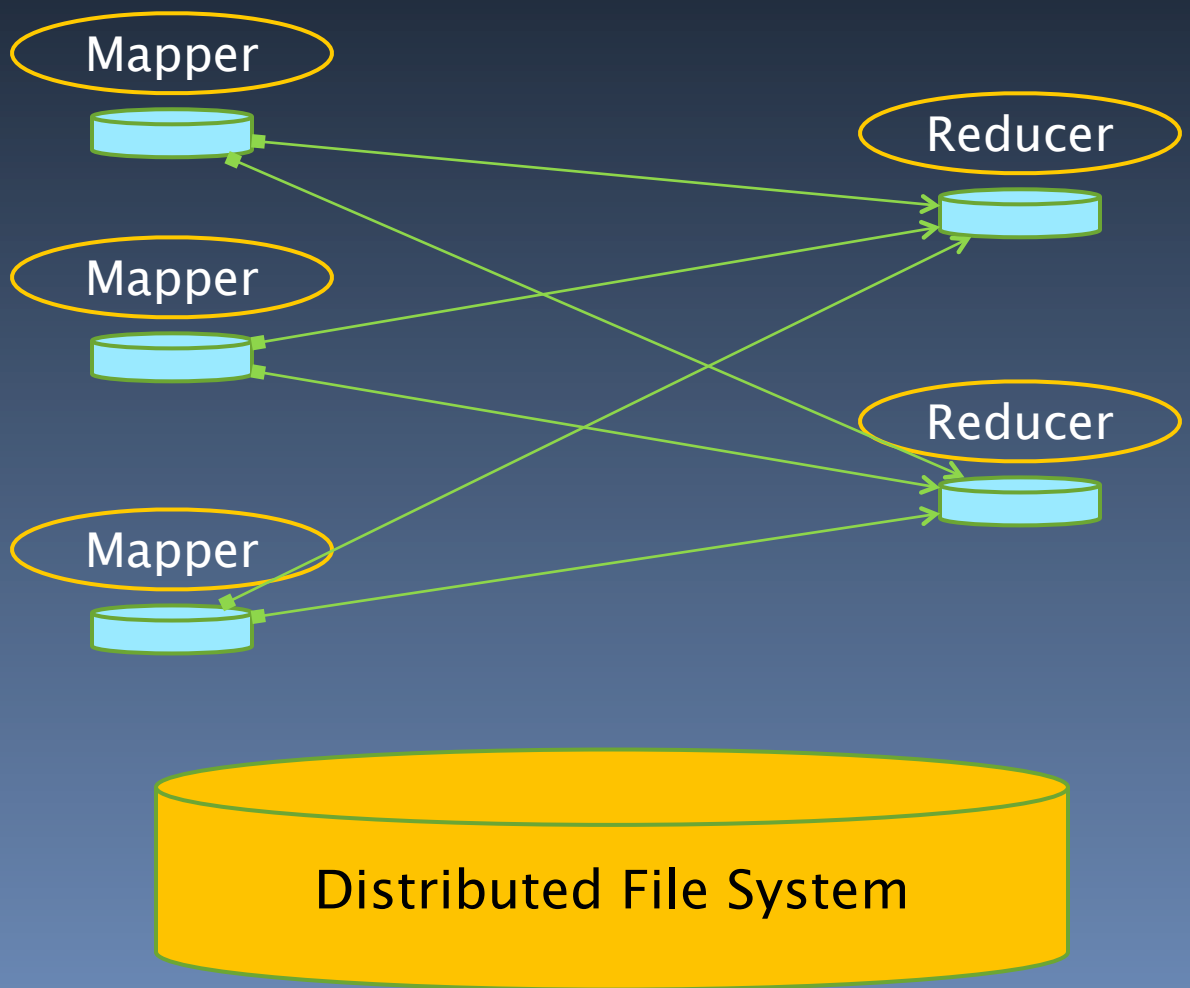
- Non data-parallel applications
  - E.g., search and prune, branch-and-bound
- Communication > Computation
  - Substantial non-local dataflow
  - More pronounced in iterative algorithms
    - how much does the input change across iterations?

Investigation

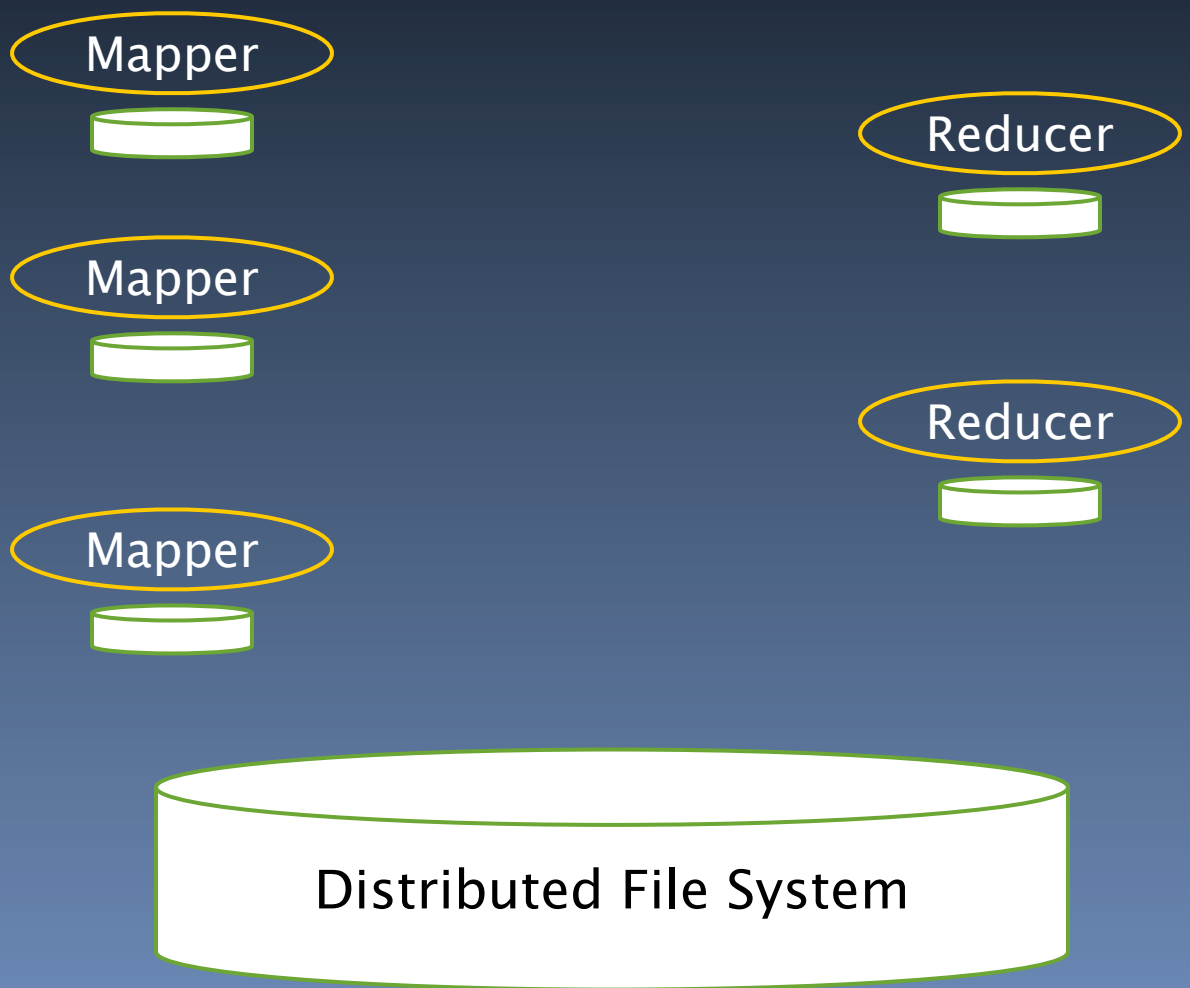
- Implemented sparse graph kernels/ apps
  - Matrix-matrix, matrix-vector products
  - Node accumulation, Similarity etc.
- Observation
  - Iterative structure
  - High synchronization costs under multiple iterations
    - Data shuffling, copying and hashing

Asynchronous data movement

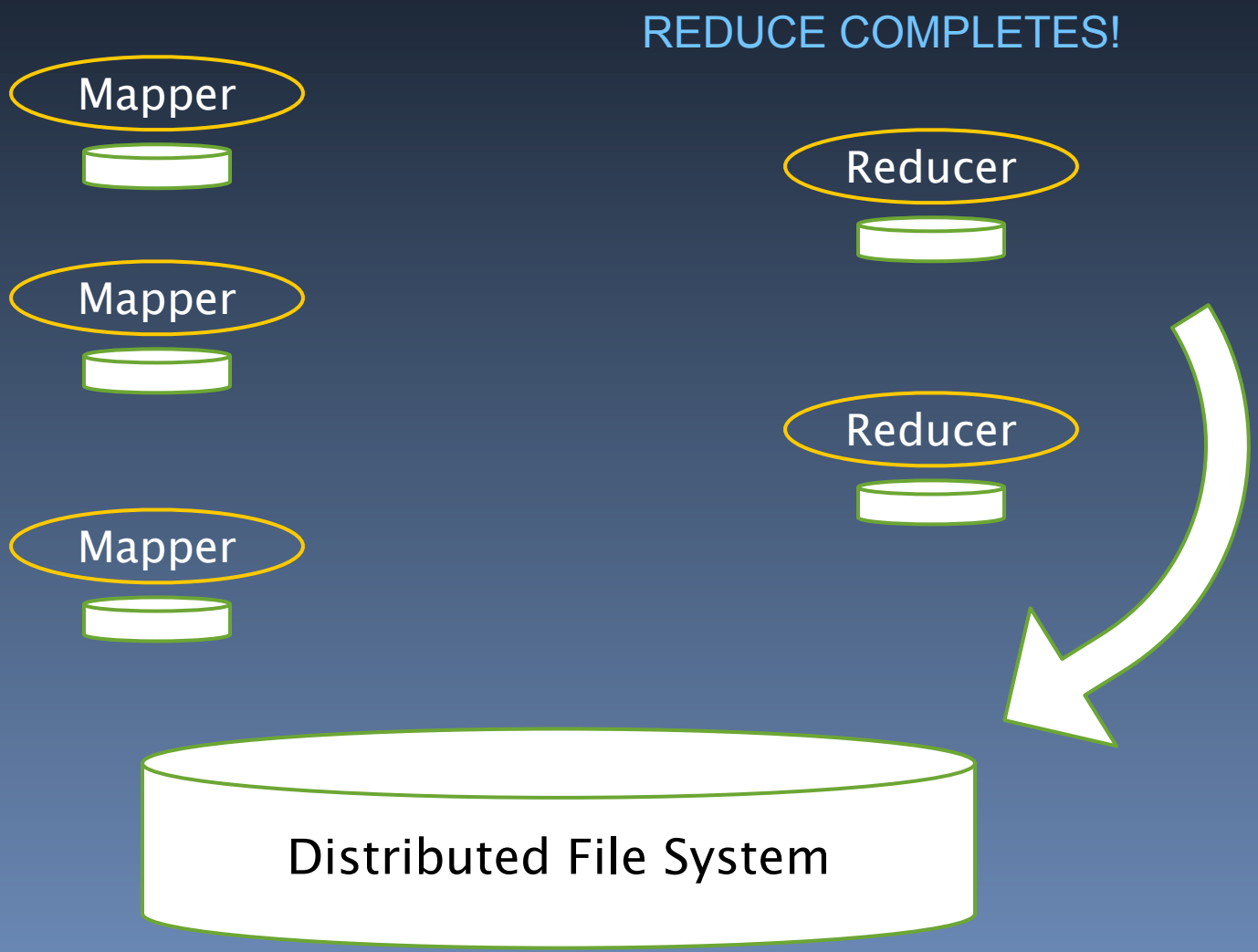
Mappers complete and asynchronously push data to reducers



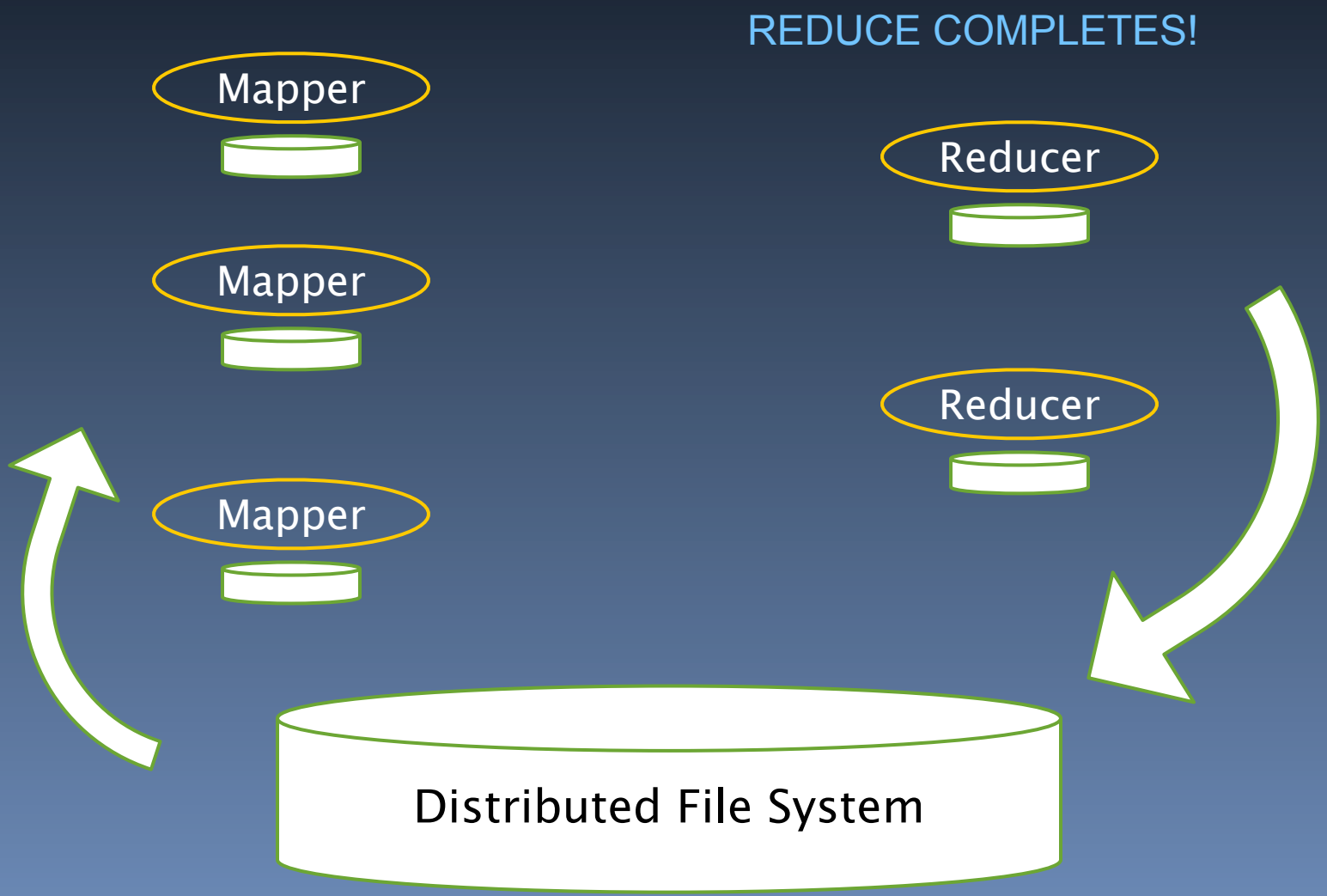
Asynchronous data movement



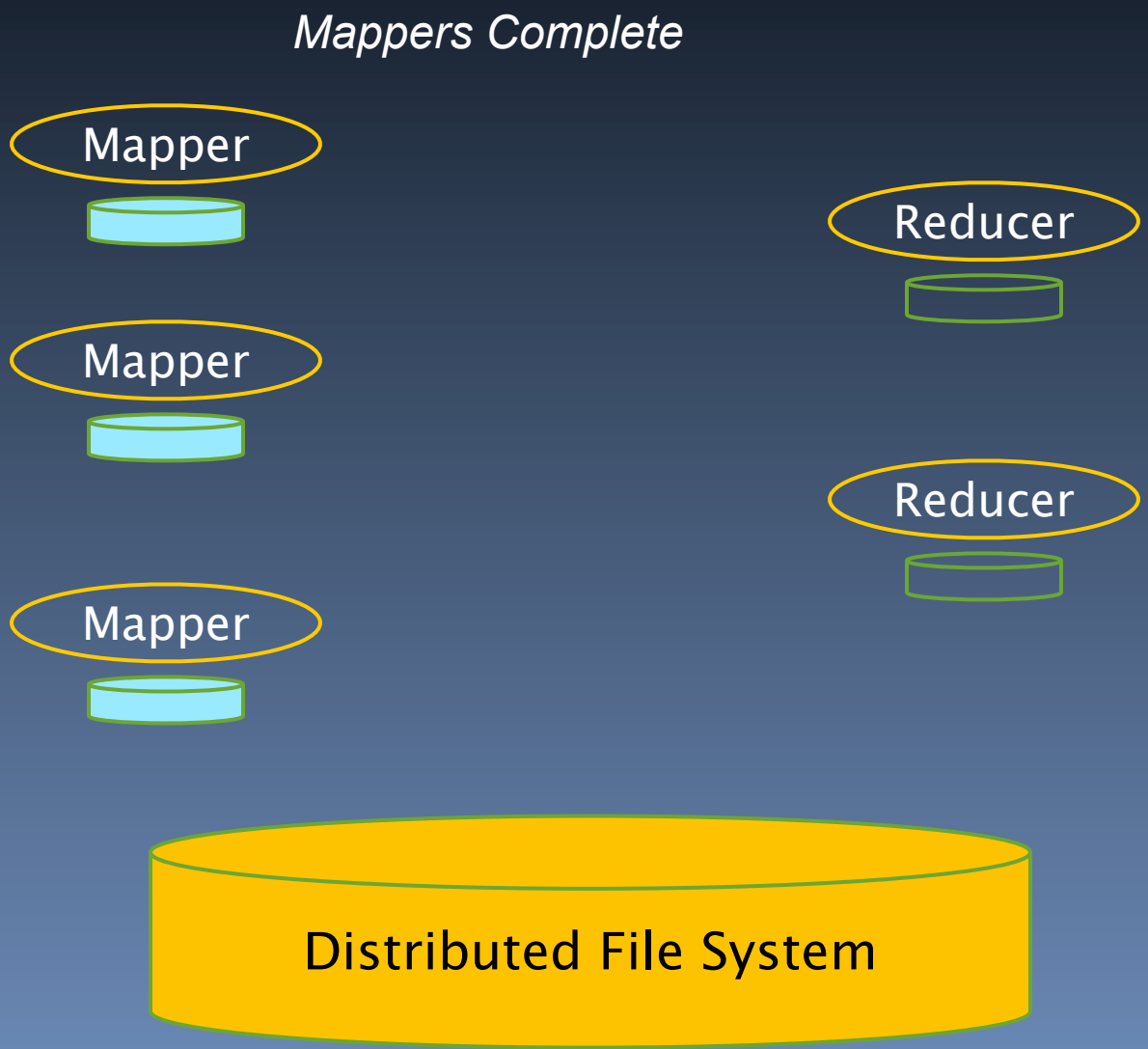
Asynchronous data movement



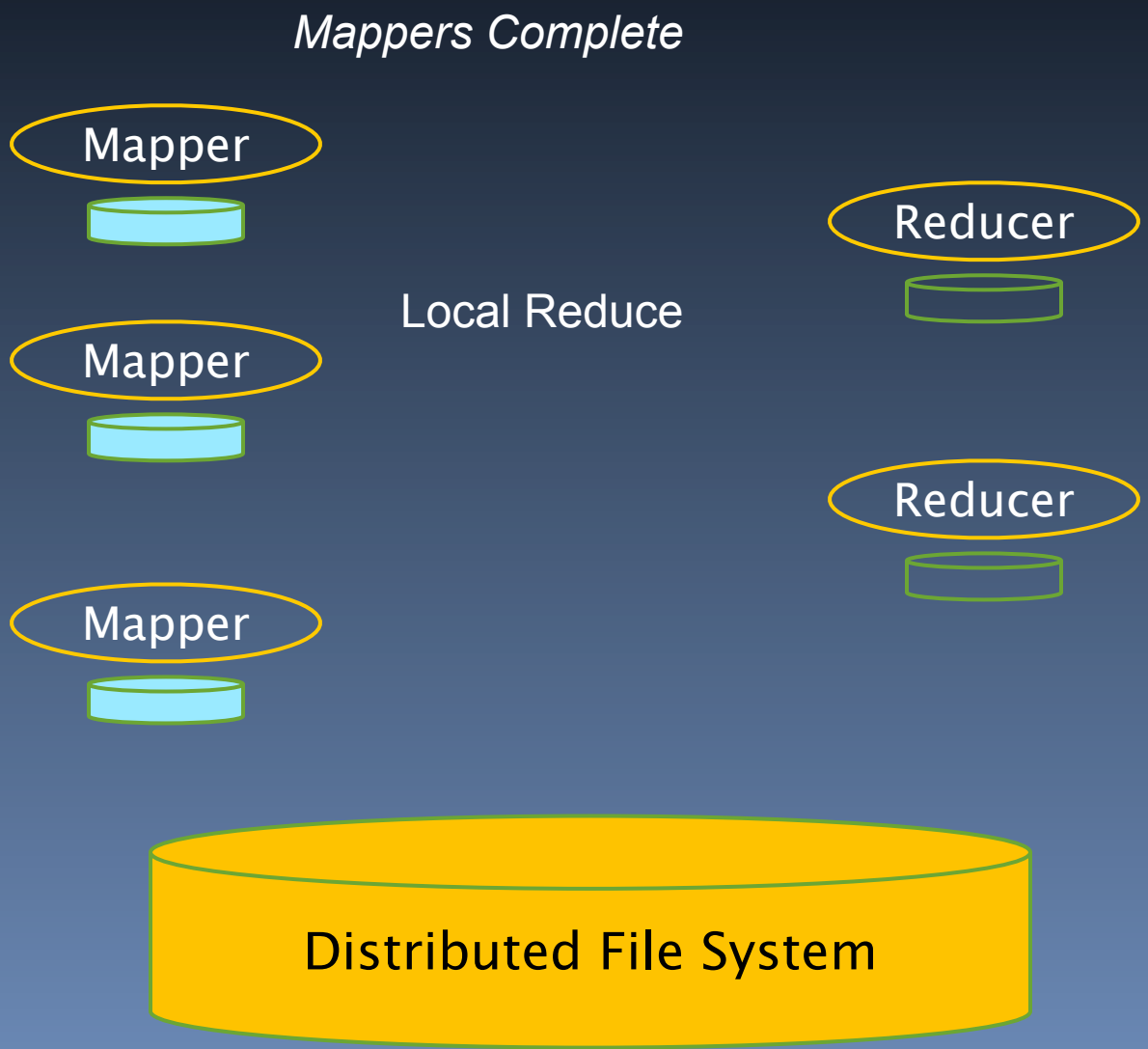
Asynchronous data movement



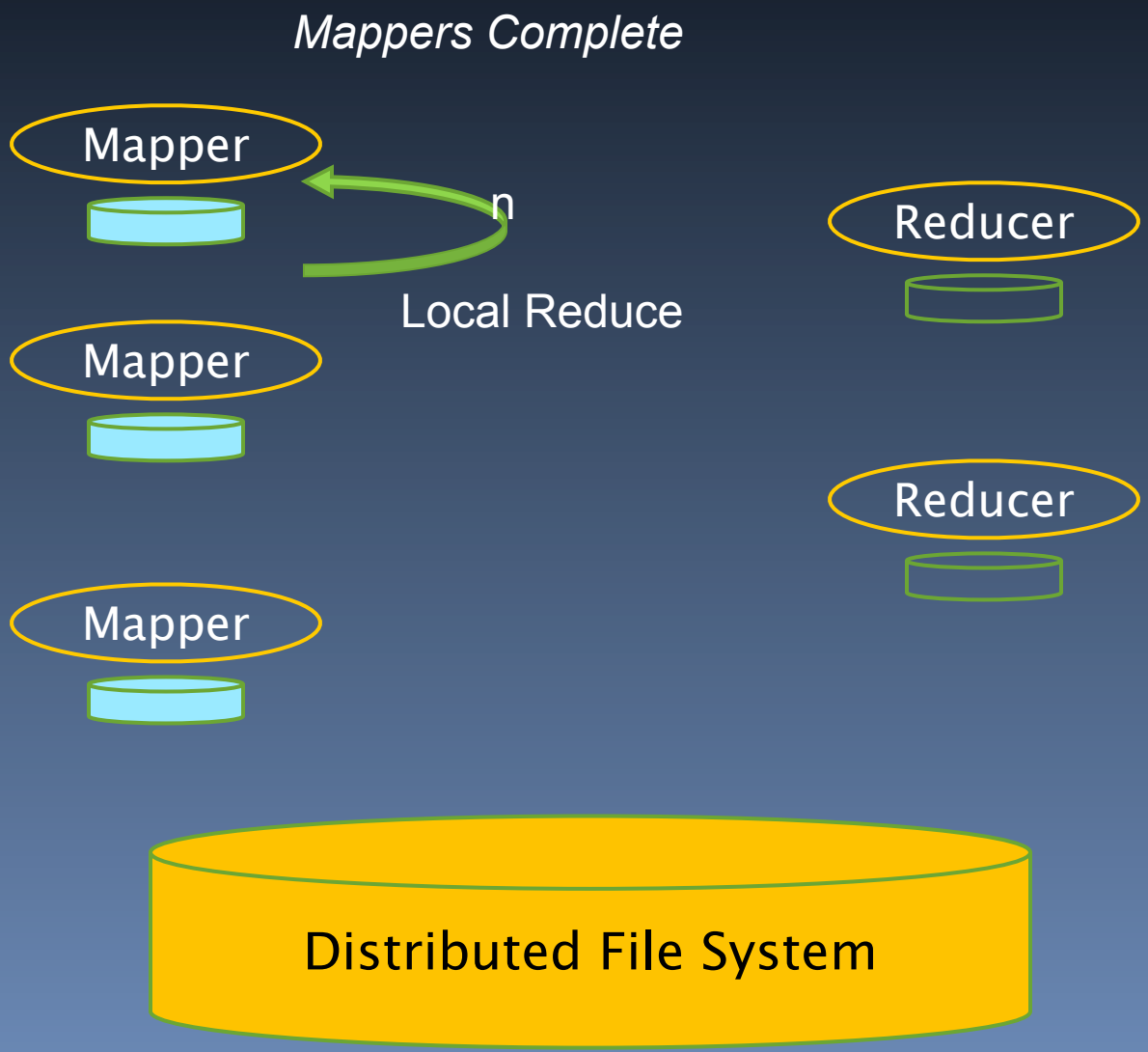
Relaxed Synchronization



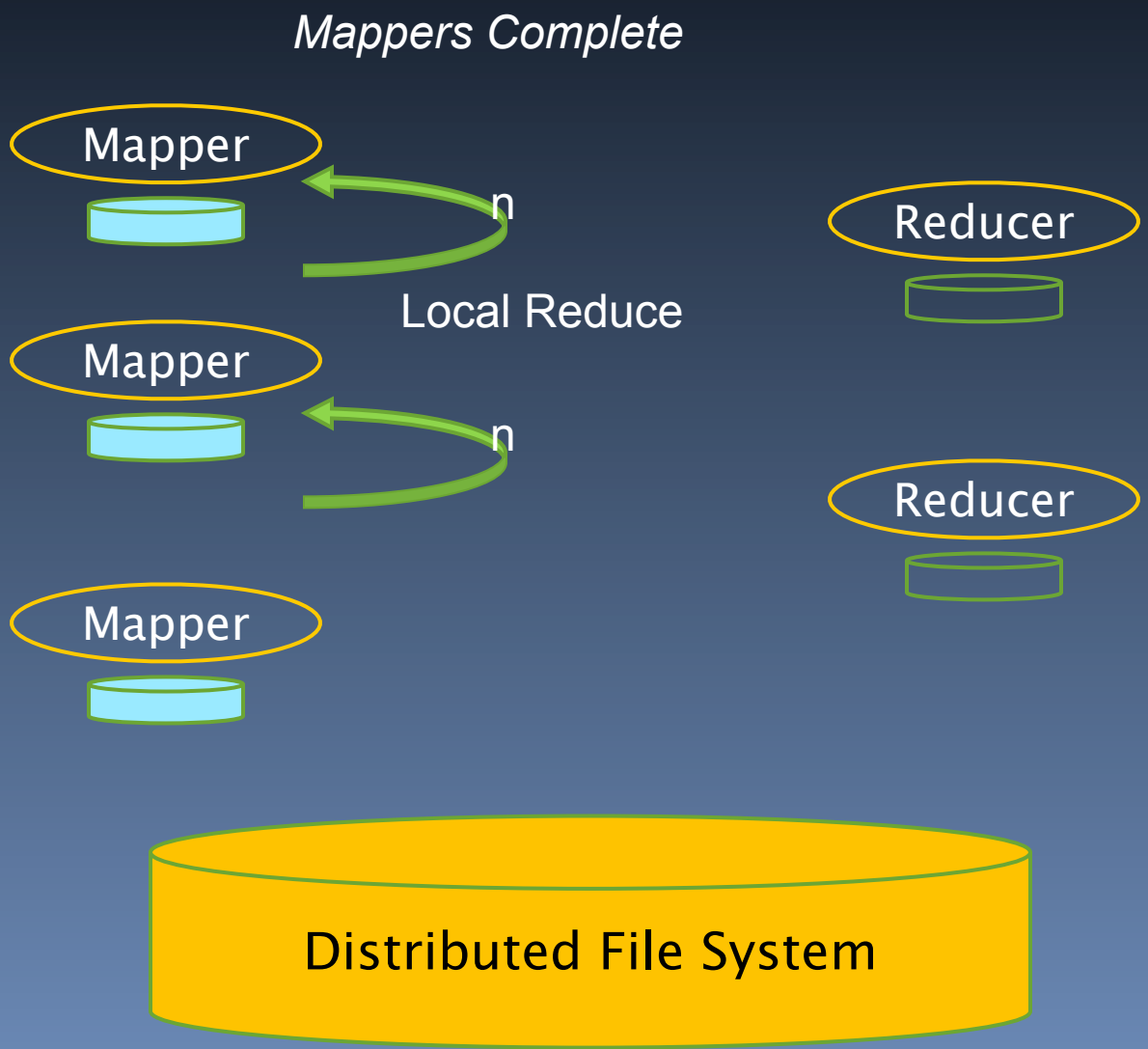
Relaxed Synchronization



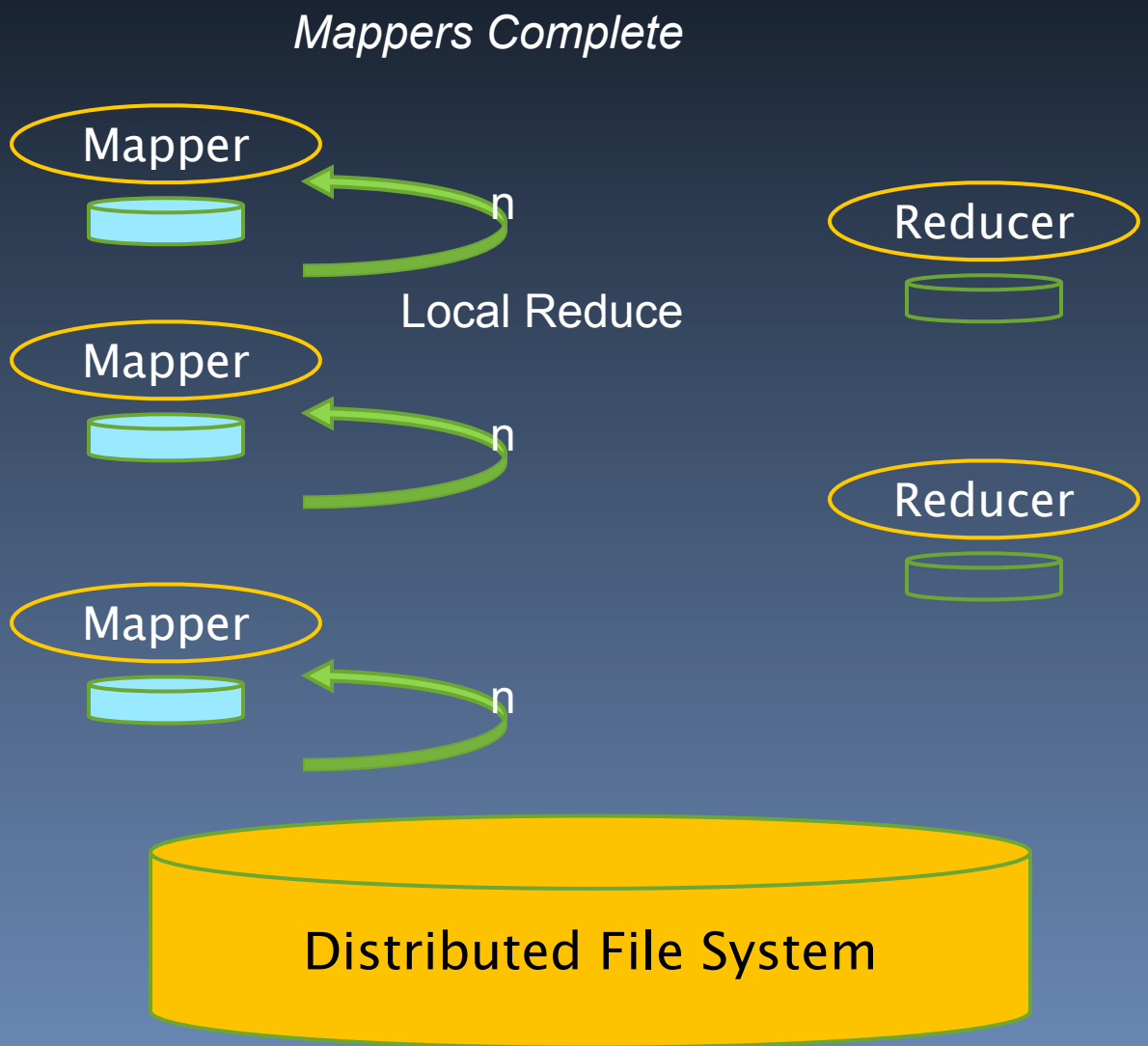
Relaxed Synchronization



Relaxed Synchronization

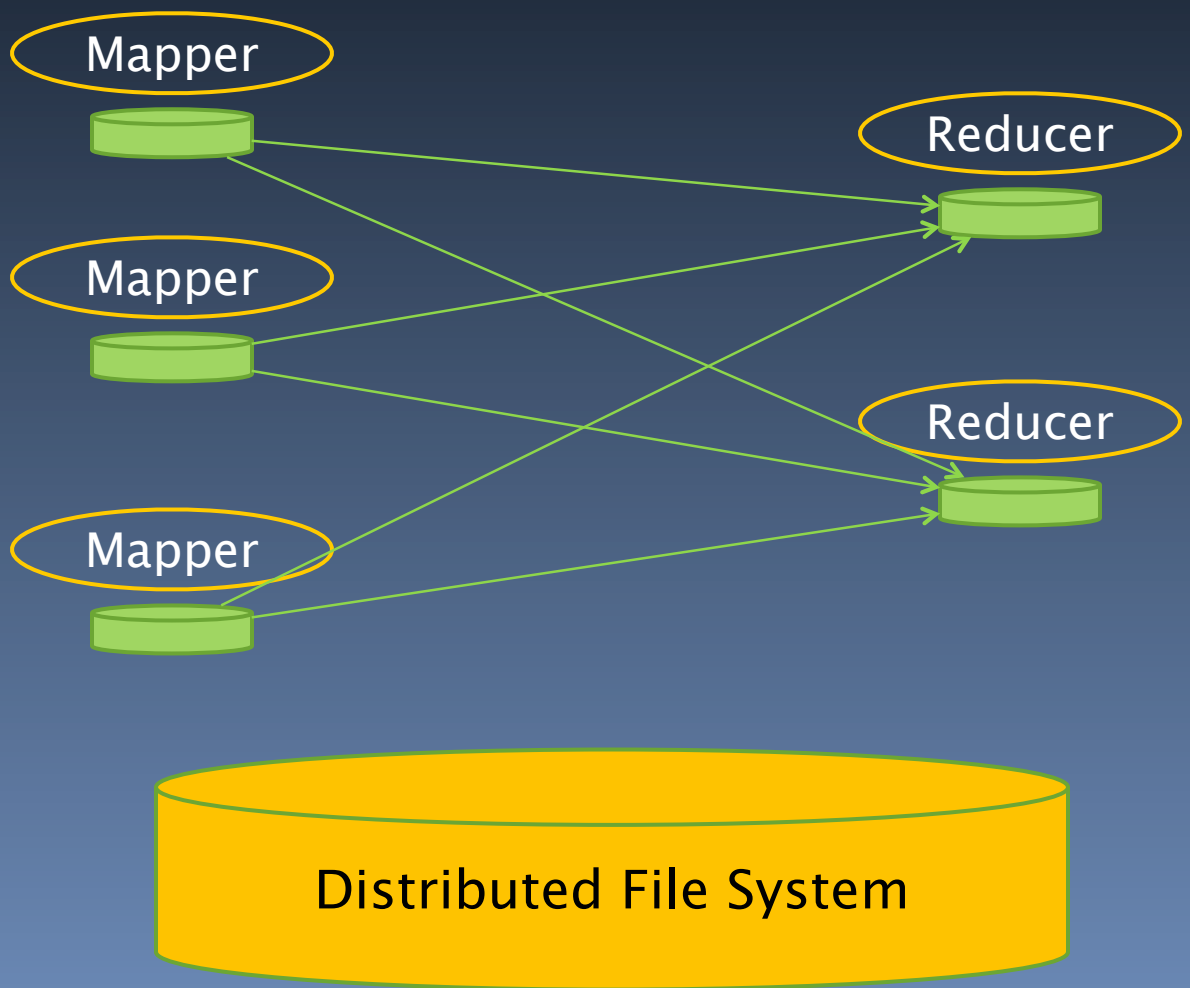


Relaxed Synchronization



Relaxed Synchronization

Perform global synchronization after local convergence or iteration bound



Relaxed Synchronization – Applicability

- Asynchronous algorithms
  - Order of operations doesn't matter
  - Monotonicity, commutativity
- Desirable input partitioning properties
  - Fewer dependencies across partitions
- Examples
  - Pagerank, eigenspectra,
  - clustering (K-Means), etc.

Pagerank

- Algorithmic asynchrony
- Input
  - Hubs-and-spokes model
  - Densely-connected components
  - Minimal dependencies across components

Pagerank – MapReduce

- Map input
  - Key: Source
  - Value: Outlinks
- Map output
  - Key: Destination Node ID
  - Value: Contribution of Src towards Dst's Pagerank

Pagerank – MapReduce

- Reduce input
  - Key: Node ID
  - Value: All inlinks' Pagerank contributions
- Reduce output
  - Key: Node ID
  - Value: Pagerank

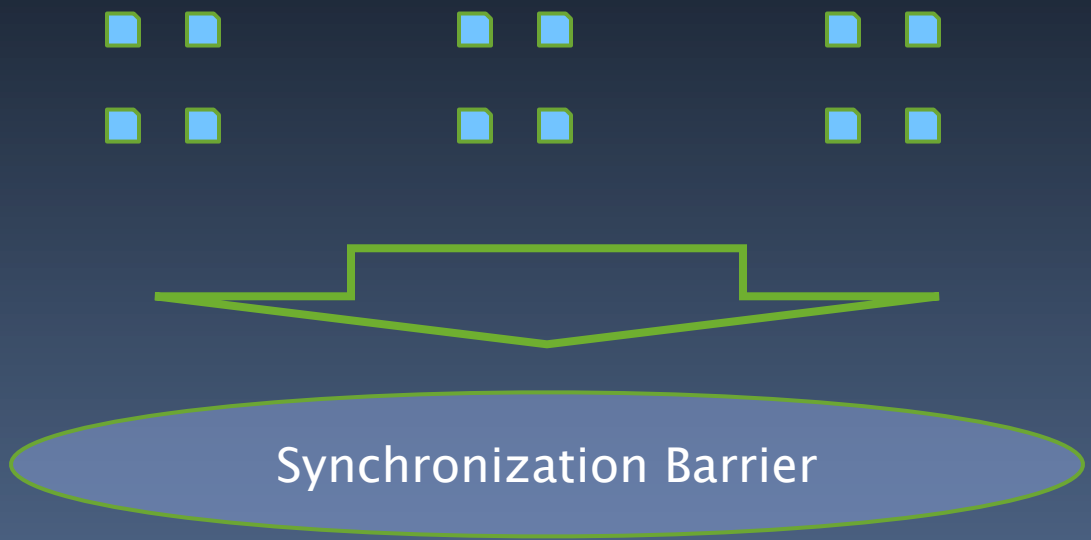
Pagerank - General

Pagerank – General

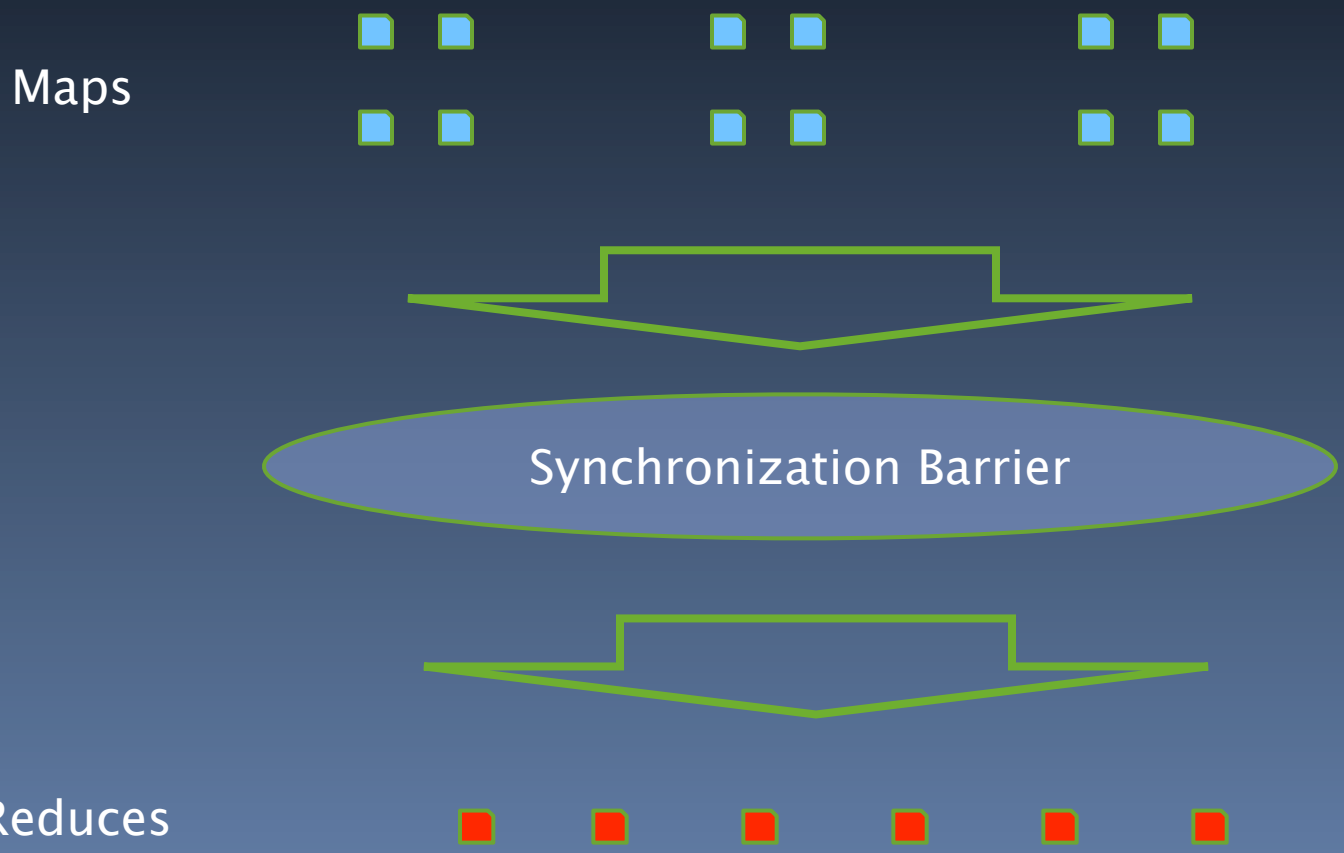
Maps

Pagerank – General

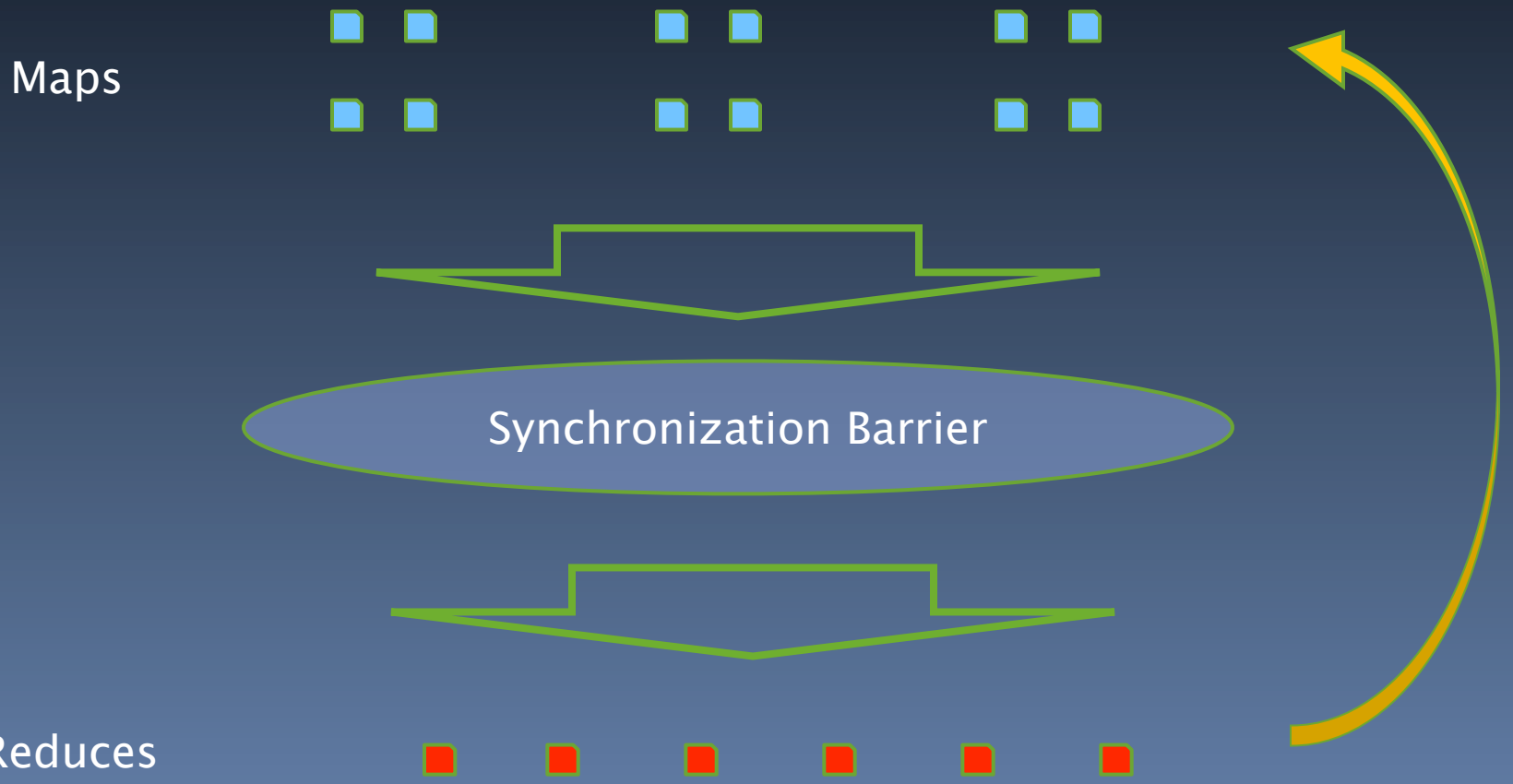
Maps



Pagerank – General



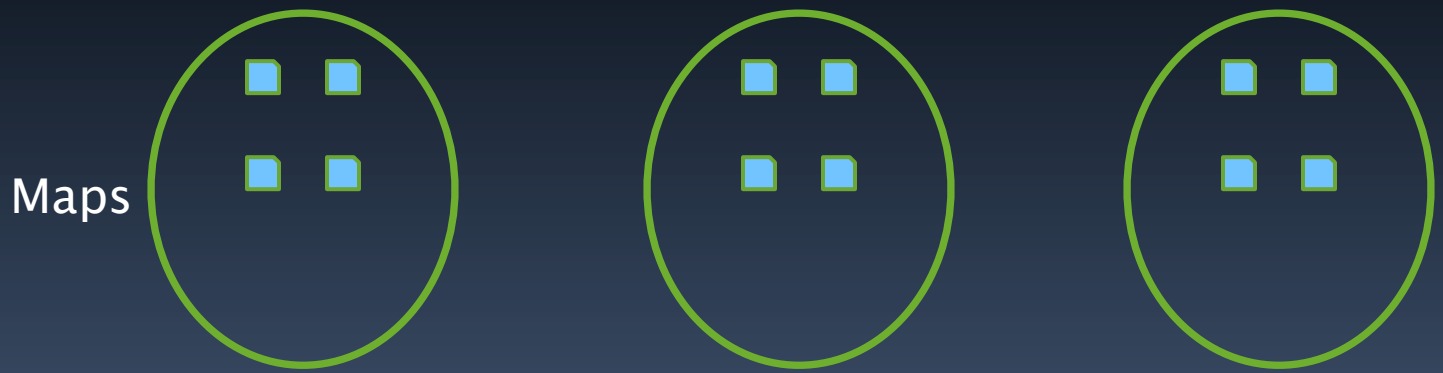
Pagerank – General



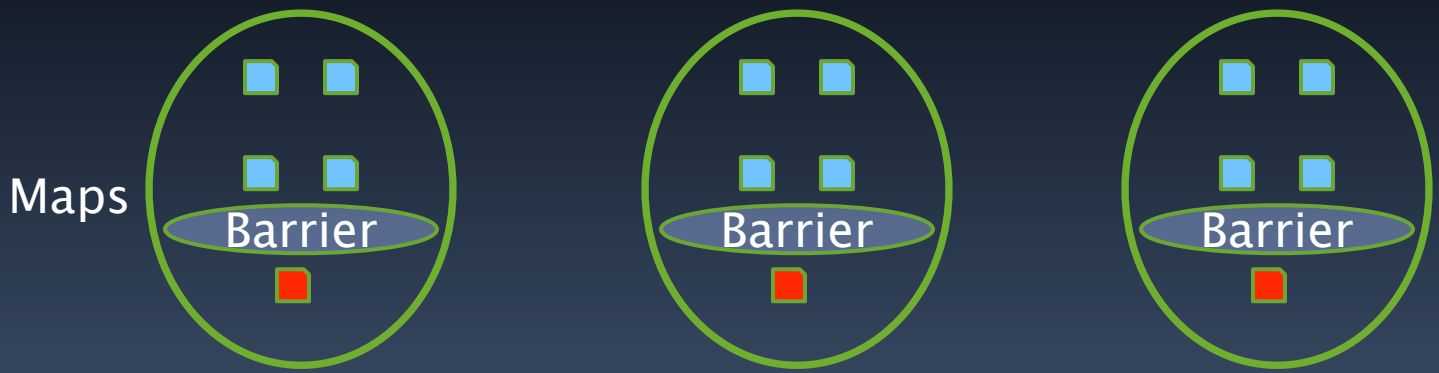
Pagerank - Relaxed Synch.

Maps

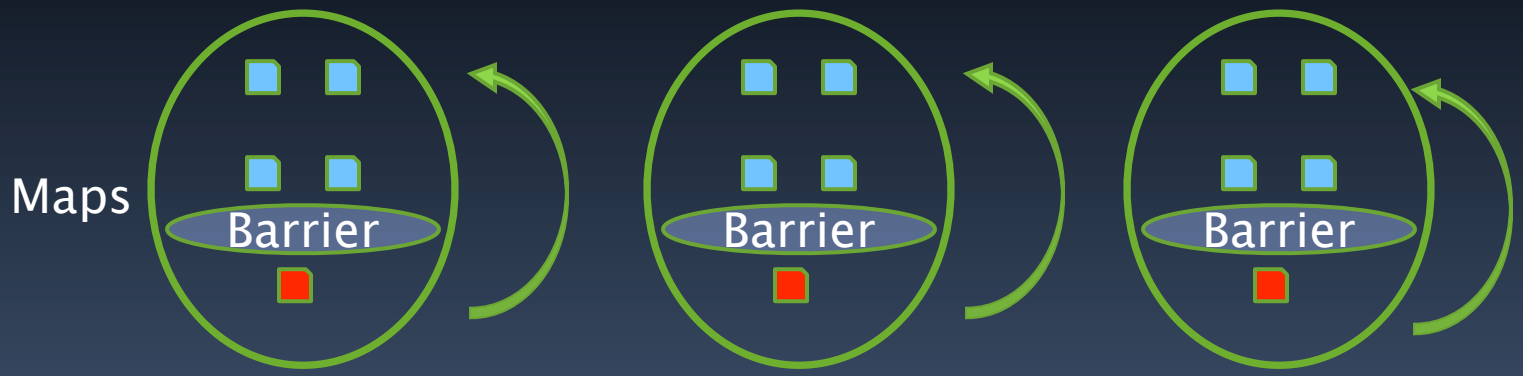
Pagerank – Relaxed Synch.



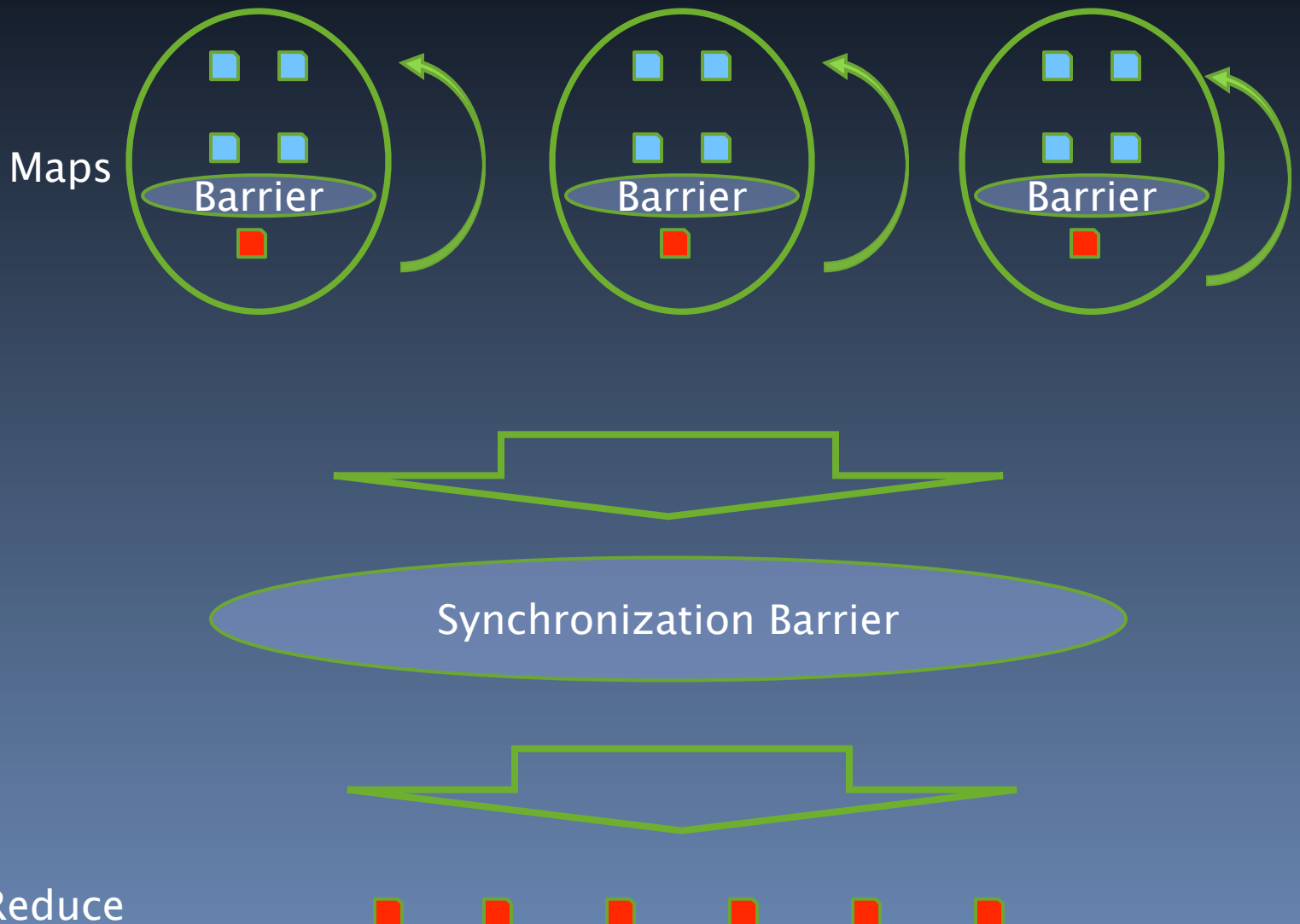
Pagerank – Relaxed Synch.



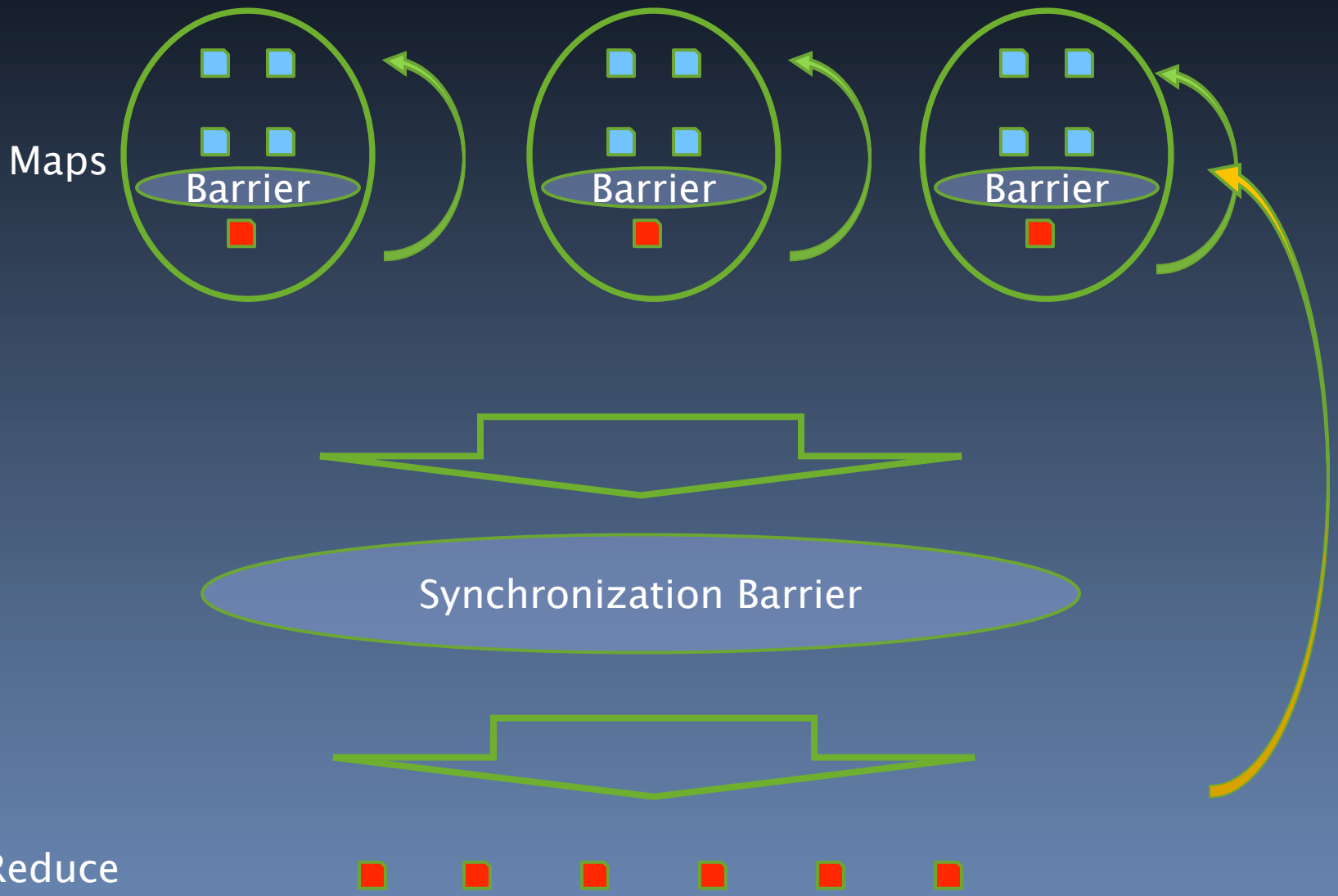
Pagerank – Relaxed Synch.



Pagerank – Relaxed Synch.



Pagerank – Relaxed Synch.



Evaluation

- Pagerank implementation
  - General – synchronization after every iteration
  - Eager
    - Each Map has local “MapReduce”, using a local hashtable for local Pageranks
    - Currently, each local map is implemented sequentially

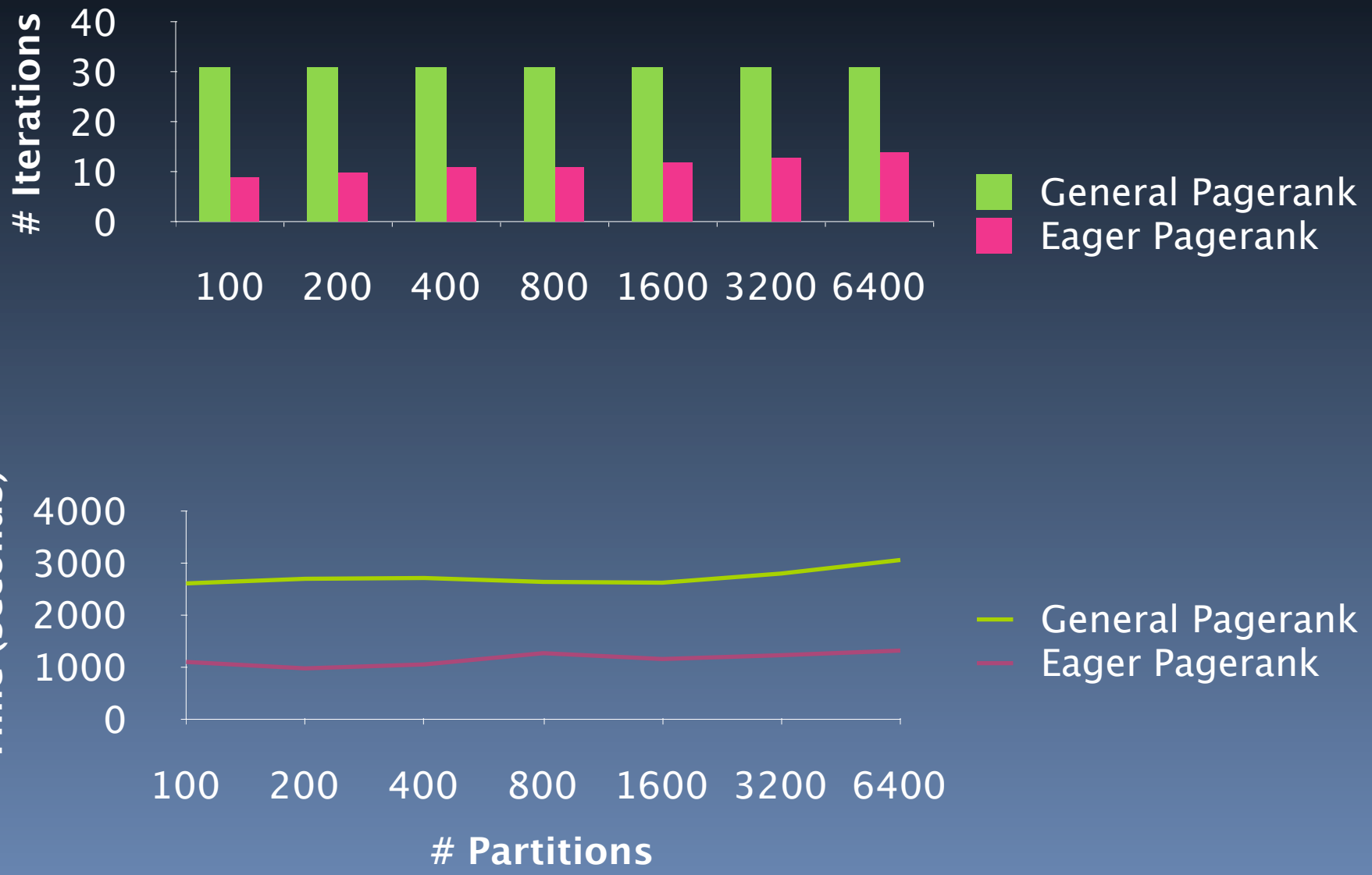
Test bed: CluE cluster

Machines	460
Physical resources	Intel® Xeon™ CPU 2.80 GHz 4 GB RAM, 2 x 400 GB hard drives
Virtual machines	1 per host
Software	Hadoop 0.17.2, Java 1.6
Heap space	1 GB per slave

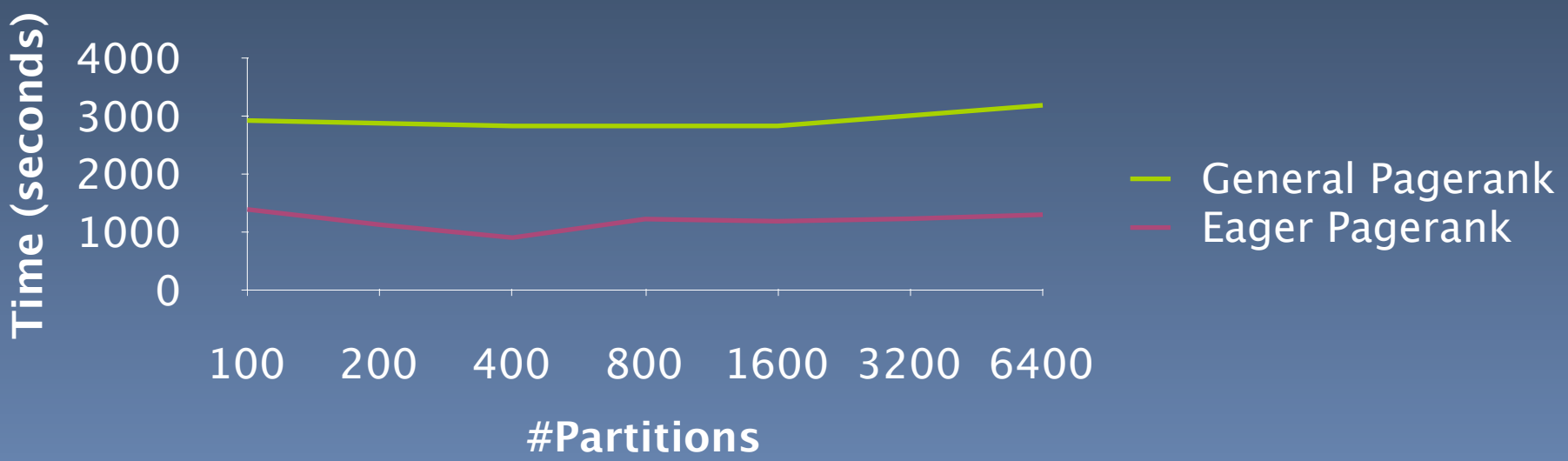
Input Graphs

- Synthetic power-law graph
  - Generate using preferential attachment
  - 100,000 nodes, 3 million edges
  - Damping factor: 0.85
- Stanford web graph
  - From Stanford WebBase project - 2002
  - 280,000 nodes, 3 million edges
  - Damping factor: 0.75
- Partition graphs using Metis
  - small cost (< 10 secs)

Performance: Synthetic power law graph

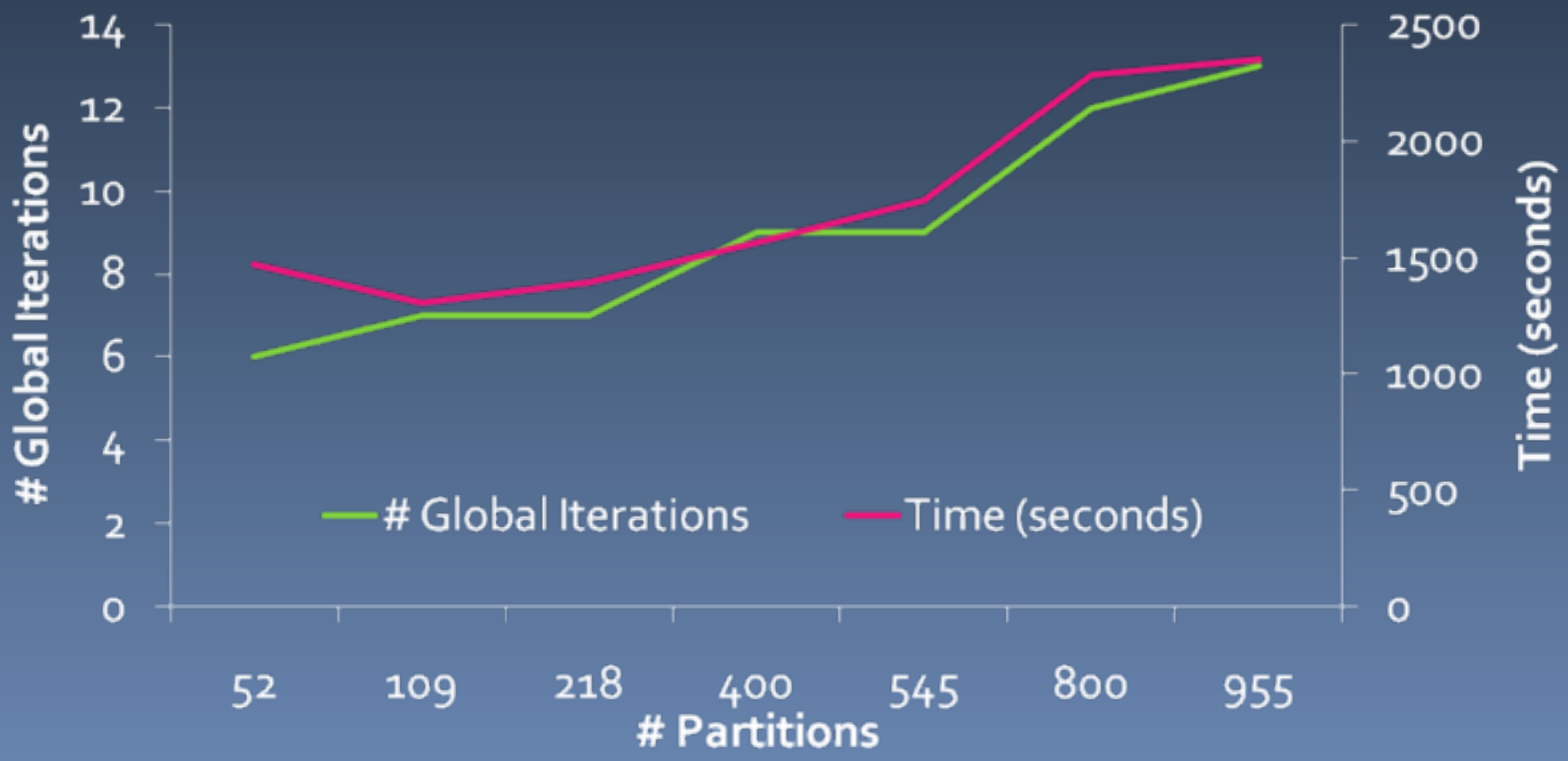


Performance: Stanford Web Graph

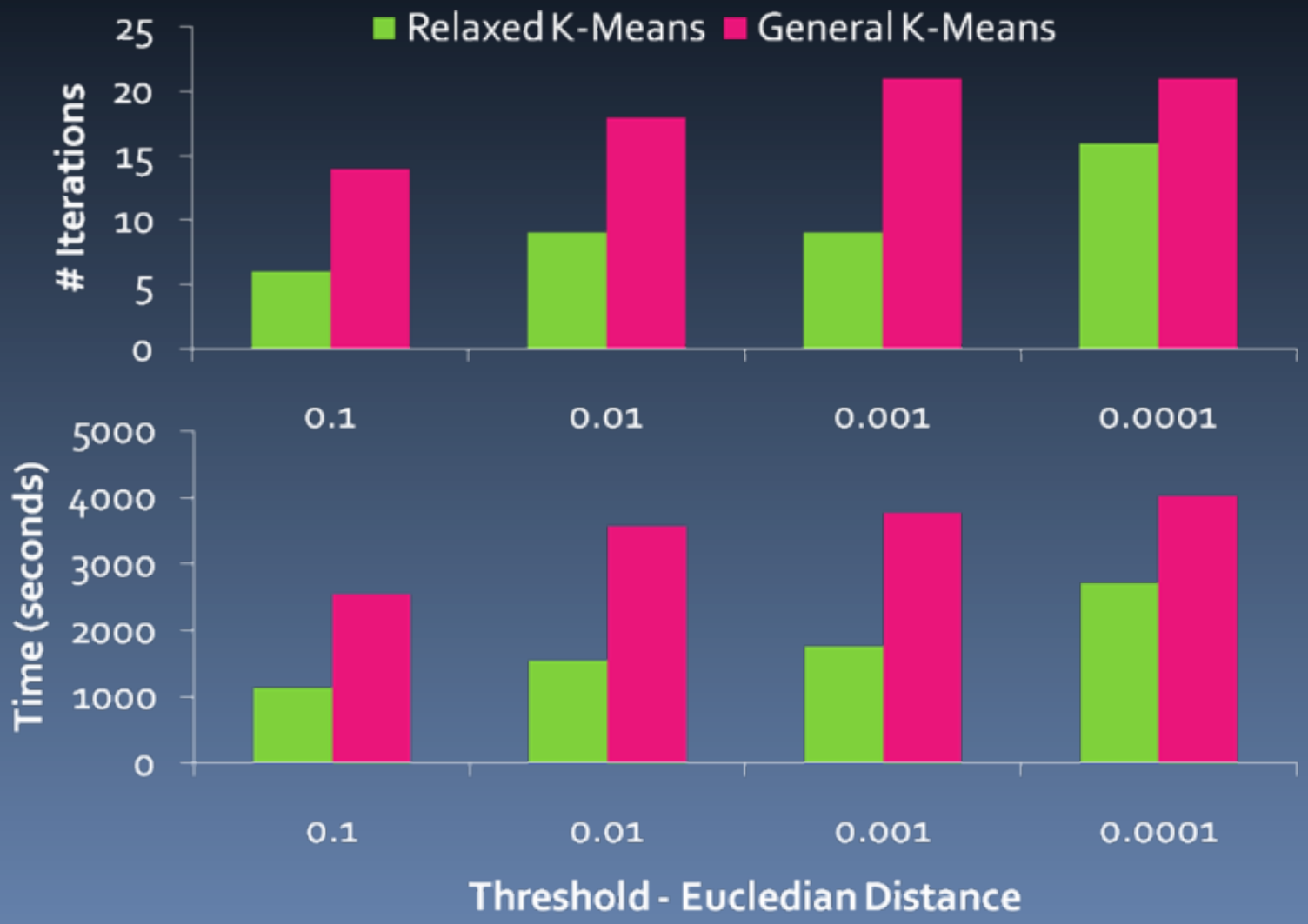


K-Means

- On US Census data (200K points, 68 dimensions)
  - Relaxed approach uses pairwise clustering
- General K-Means: 18 iterations
- Threshold: 0.01 for convergence



K-Means – threshold



Future Work

- Implement hierarchical relaxed synchronization
- Extracting operations from semantics
  - Pregel
- Validation on larger data sets
- Build a system with asynchronous data transfer – caches etc.
- Understand implications on fault-tolerance

Summary

- Can enhance scalability and performance MapReduce for
  - an interesting class of unstructured graph algorithms
- Key ideas
  - Relaxed synchronization
  - Algorithmic asynchrony
- Beneficial tradeoff between increased serial operation counts and reduced global synchronization

Questions?

Other details

- Additional overhead
  - Metis – graph partitioning
  - < 10 seconds
- Local iterations
  - Per-partition ~ 40 per one global iteration