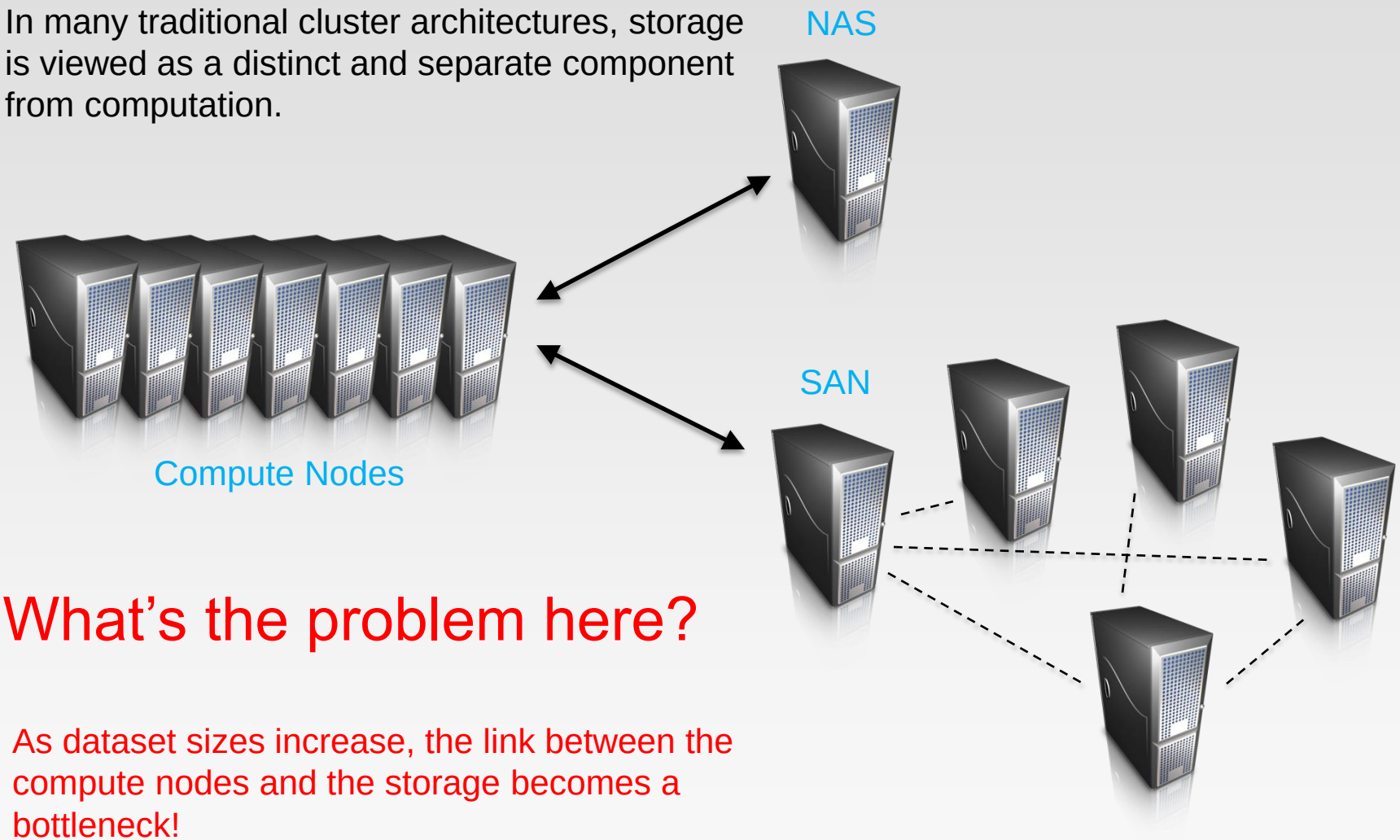
Move Data to Workers?

In many traditional cluster architectures, storage is viewed as a distinct and separate component from computation.



What's the problem here?

As dataset sizes increase, the link between the compute nodes and the storage becomes a bottleneck!

Latency and Throughput

- ***Latency*** is the time required to perform some action or to produce some result.
  - Measured in units of time -- hours, minutes, seconds, nanoseconds or clock periods.
  - I/O latency: the time that it takes to complete a single I/O.

- ***Throughput*** is the number of such actions executed or results produced per unit of time.
  - Measured in units of whatever is being produced (e.g., data) per unit of time.
  - Disk throughput: the maximum rate of sequential data transfer, measured by Mb/sec etc.

Distributed File System

- Don’t move data to workers… move workers to the data!
  - Store data on the local disks of nodes in the cluster
  - Start up the workers on the node that has the data local
- Why?
  - Not enough RAM to hold all the data in memory
  - Disk access is slow (low-latency), but disk throughput is reasonable (high throughput)
- A distributed file system is the answer
  - A distributed file system is a client/server-based application that allows clients to access and process data stored on the server as if it were on their own computer
  - GFS (Google File System) for Google’s MapReduce
  - HDFS (Hadoop Distributed File System) for Hadoop

Assumptions and Goals of HDFS

- Very large datasets
  - 10K nodes, 100 million files, 10PB
- Streaming data access
  - Designed more for batch processing rather than interactive use by users
  - The emphasis is on high throughput of data access rather than low latency of data access.
- Simple coherency model
  - Built around the idea that the most efficient data processing pattern is a write-once read-many-times pattern
  - A file once created, written, and closed need not be changed except for appends and truncates
- “Moving computation is cheaper than moving data”
  - Data locations exposed so that computations can move to where data resides

Assumptions and Goals of HDFS (Cont')

- Assumes Commodity Hardware
  - Files are replicated to handle hardware failure
  - Hardware failure is normal rather than exception. Detect failures and recover from them
- Portability across heterogeneous hardware and software platforms
  - designed to be easily portable from one platform to another

- HDFS is not suited for:
  - Low-latency data access (HBase is a better option)
  - Lots of small files (NameNodes hold metadata in memory)

Unique features of HDFS

- HDFS has a bunch of unique features that make it ideal for distributed systems:
  - Failure tolerant - data is duplicated across multiple DataNodes to protect against machine failures. The default is a replication factor of 3 (every block is stored on three machines).
  - Scalability - data transfers happen directly with the DataNodes so your read/write capacity scales fairly well with the number of DataNodes
  - Space - need more disk space? Just add more DataNodes and re-balance
  - Industry standard - Other distributed applications are built on top of HDFS (HBase, MapReduce)

- HDFS is designed to process large data sets with write-once-read-many semantics, it is not for low latency access