

Parallel and Distributed Computing

CS3006 (BDS-6A)

Lecture 22

Instructor: Dr. Syed Mohammad Irteza

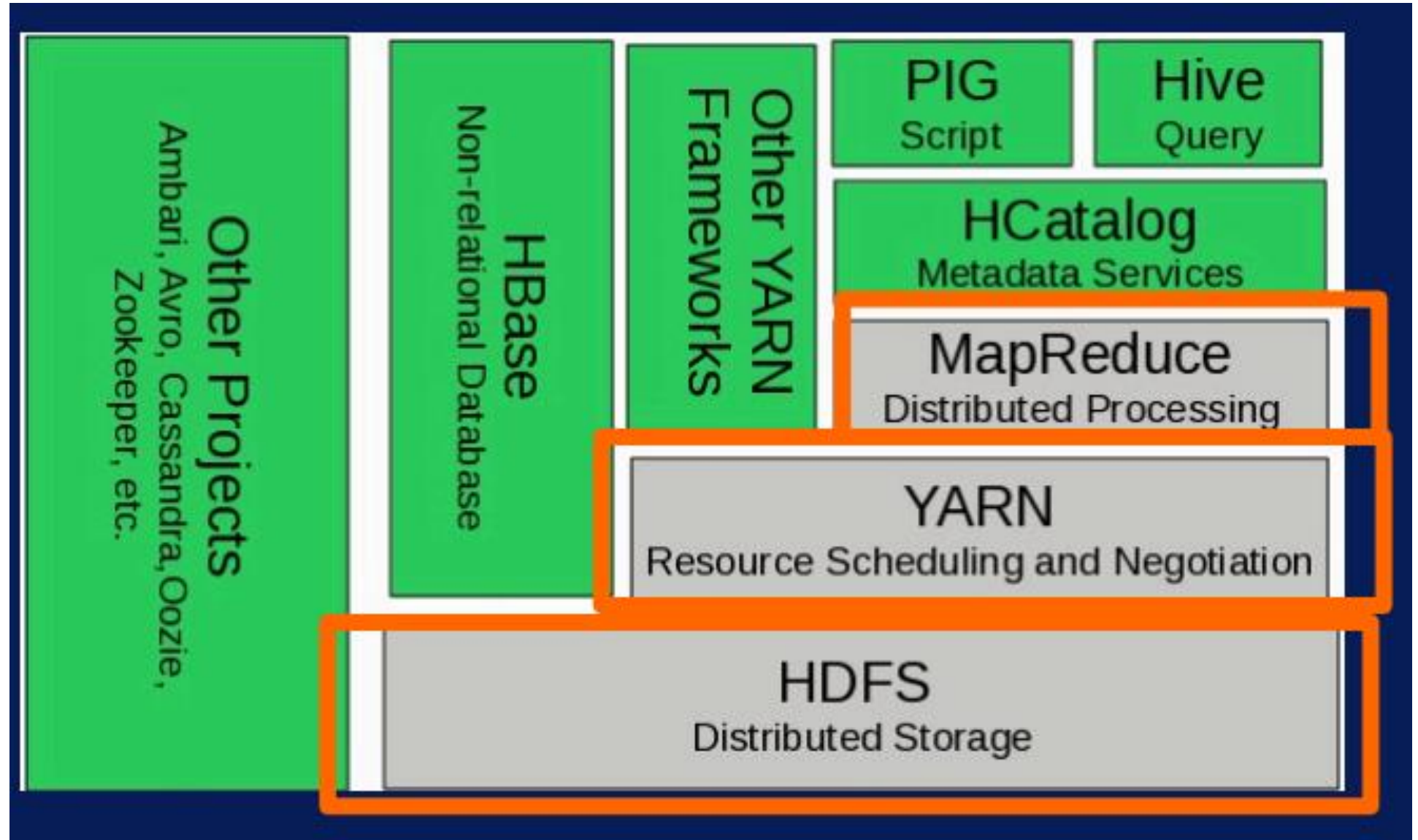
Assistant Professor, Department of Computer Science, FAST

20 April, 2023

Previous Lecture

- File Systems
- Distributed File Systems
 - Components:
 - Client module
 - Directory service
 - Flat file service
 - Examples
 - NFS (Architecture, client-server, RPCs, Virtual File System, Hierarchy, Mounting)
 - HDFS (Hadoop, YARN, MapReduce, etc.) – NameNode, DataNode
 - GFS
 - Andrew, Sprite

Hadoop Module



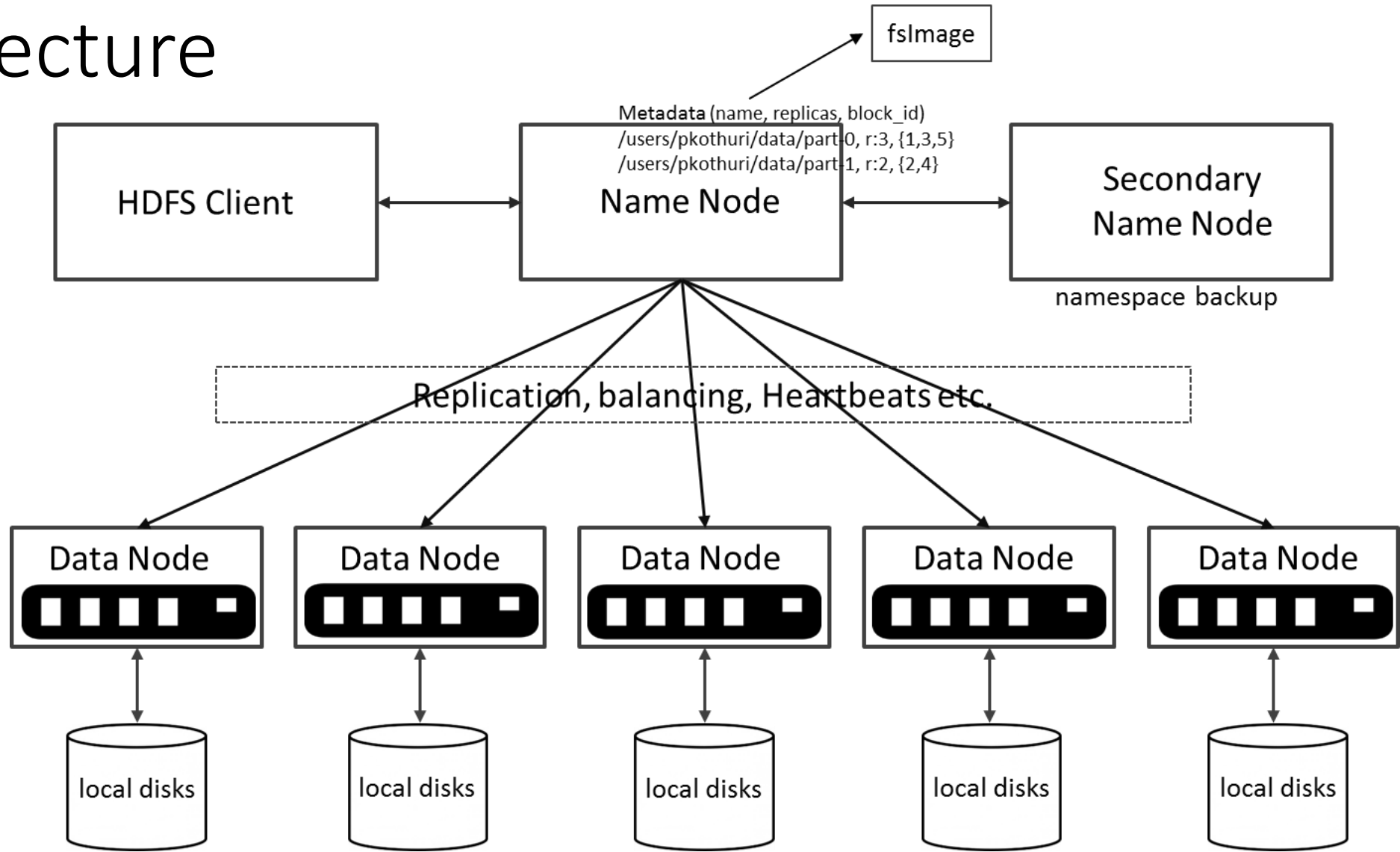
Hadoop Distributed File System

- **HDFS** is a distributed file system written in Java that is fault tolerant, scalable and extremely easy to expand.
- **HDFS** is the primary distributed storage for Hadoop applications.
- **HDFS** provides interfaces for applications to move themselves closer to data.

There are two types of machines in a **HDFS** cluster.

- NameNode is the heart of an **HDFS** filesystem, it maintains and manages the file system metadata. For example, what blocks make up a file, and on which DataNodes those blocks are stored.
- DataNode where **HDFS** stores the actual data, there are usually quite a few of these.

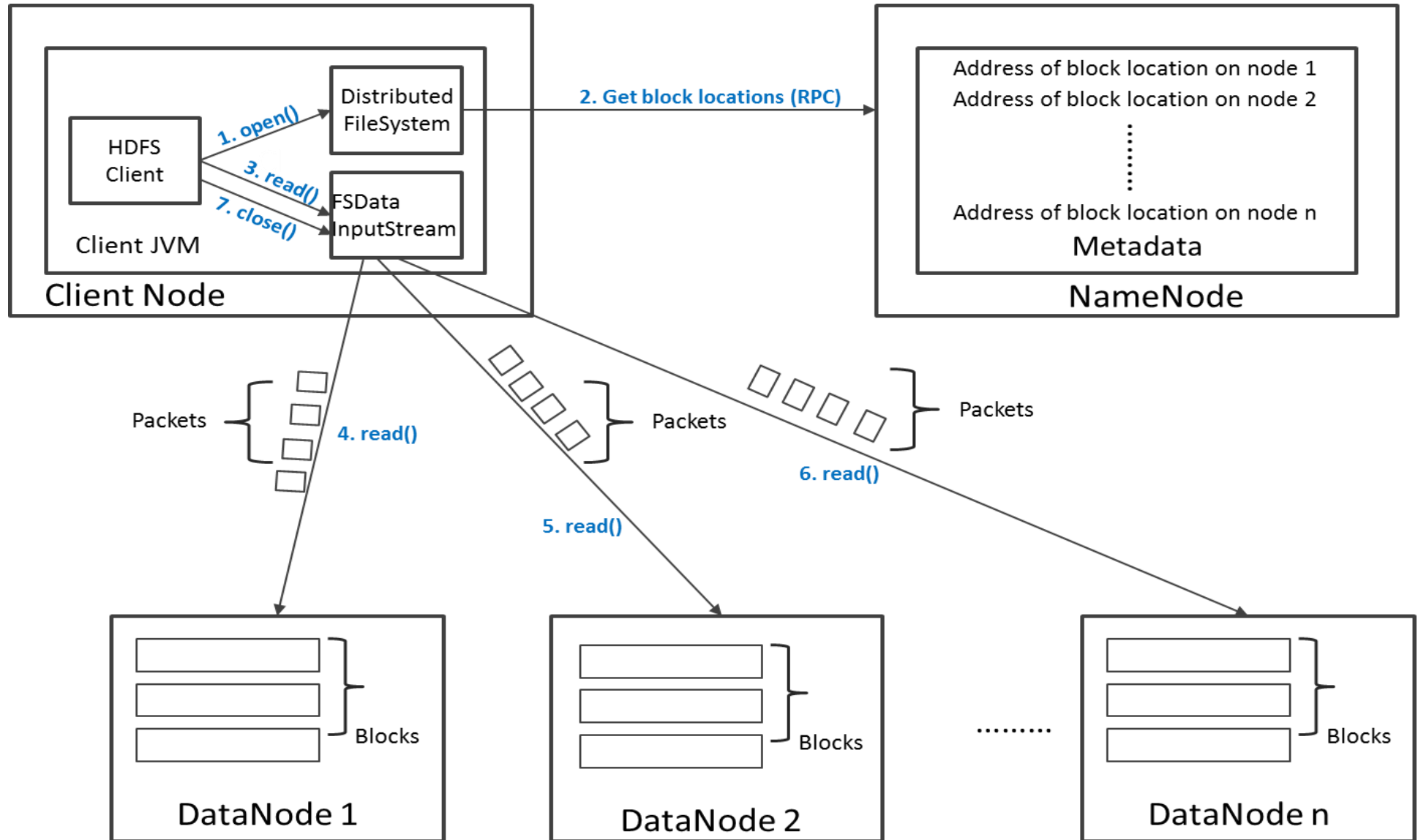
HDFS Architecture



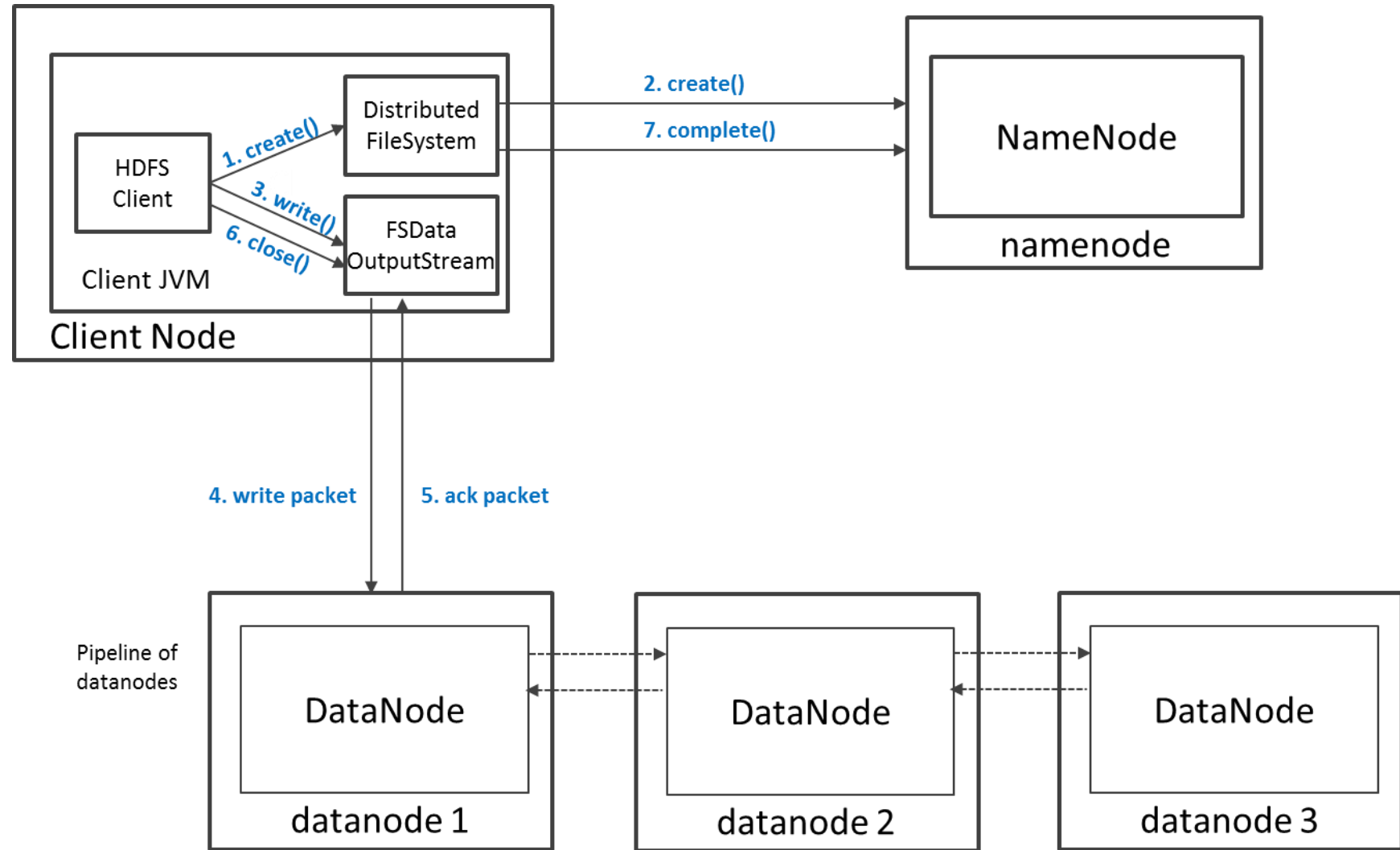
HDFS features

- 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, Map-Reduce)

Read Operation in HDFS



Write Operation in HDFS



HDFS Security

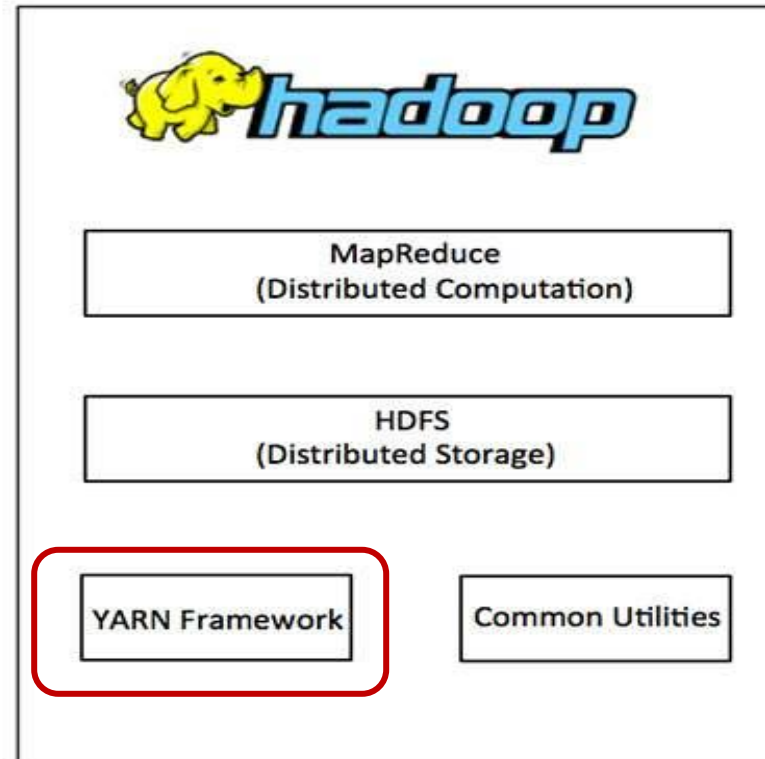
- Authentication to *Hadoop*
 - Simple – insecure way of using OS username to determine hadoop identity
 - Kerberos – authentication using kerberos ticket
 - Set by `hadoop.security.authentication=simple|kerberos`
- File and Directory permissions are same like in *POSIX*
 - read (r), write (w), and execute (x) permissions
 - also has an owner, group and mode
 - enabled by default (`dfs.permissions.enabled=true`)
- **ACLs** are used for implementation permissions that differ from natural hierarchy of users and groups
 - enabled by `dfs.namenode.acls.enabled=true`

Interfaces to HDFS

- Java API (`DistributedFileSystem`)
- C wrapper (`libhdfs`)
- HTTP protocol
- WebDAV protocol
- Shell Commands

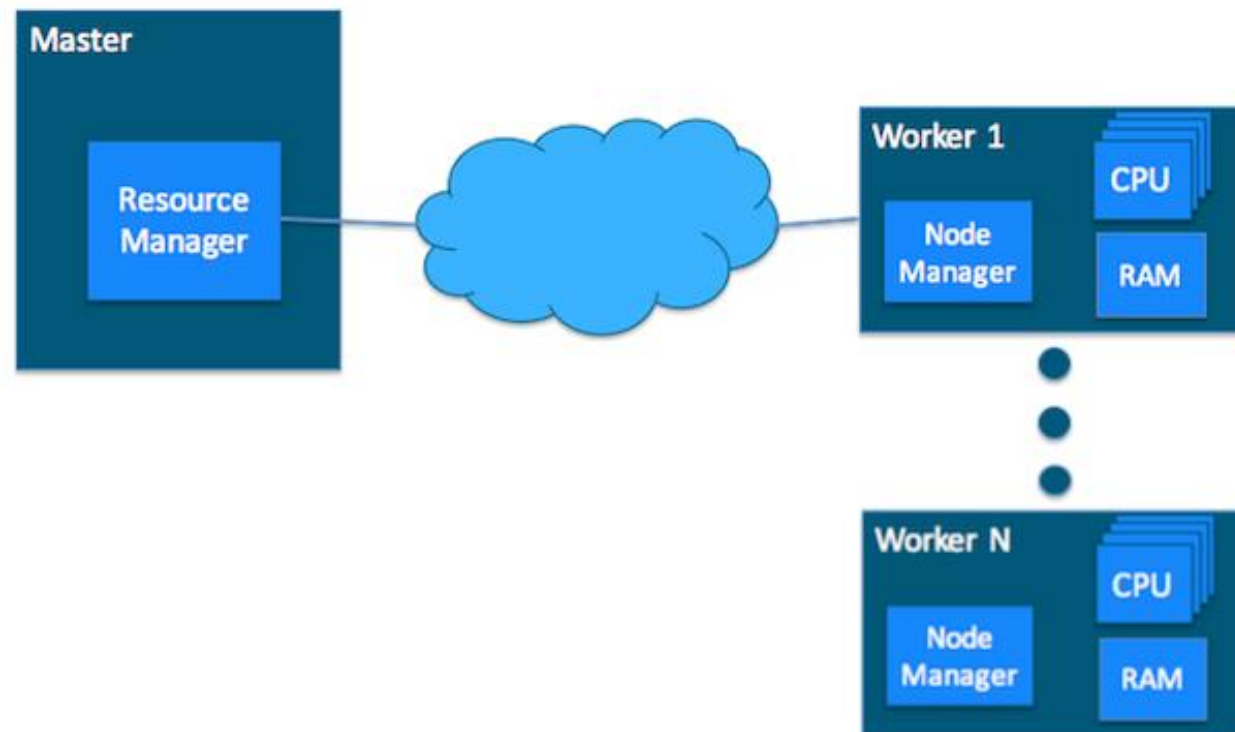
Yarn

- YARN is the prerequisite for Enterprise Hadoop
 - providing resource management and a central platform to deliver consistent operations, security, and data governance tools across Hadoop clusters.



YARN Cluster Basics

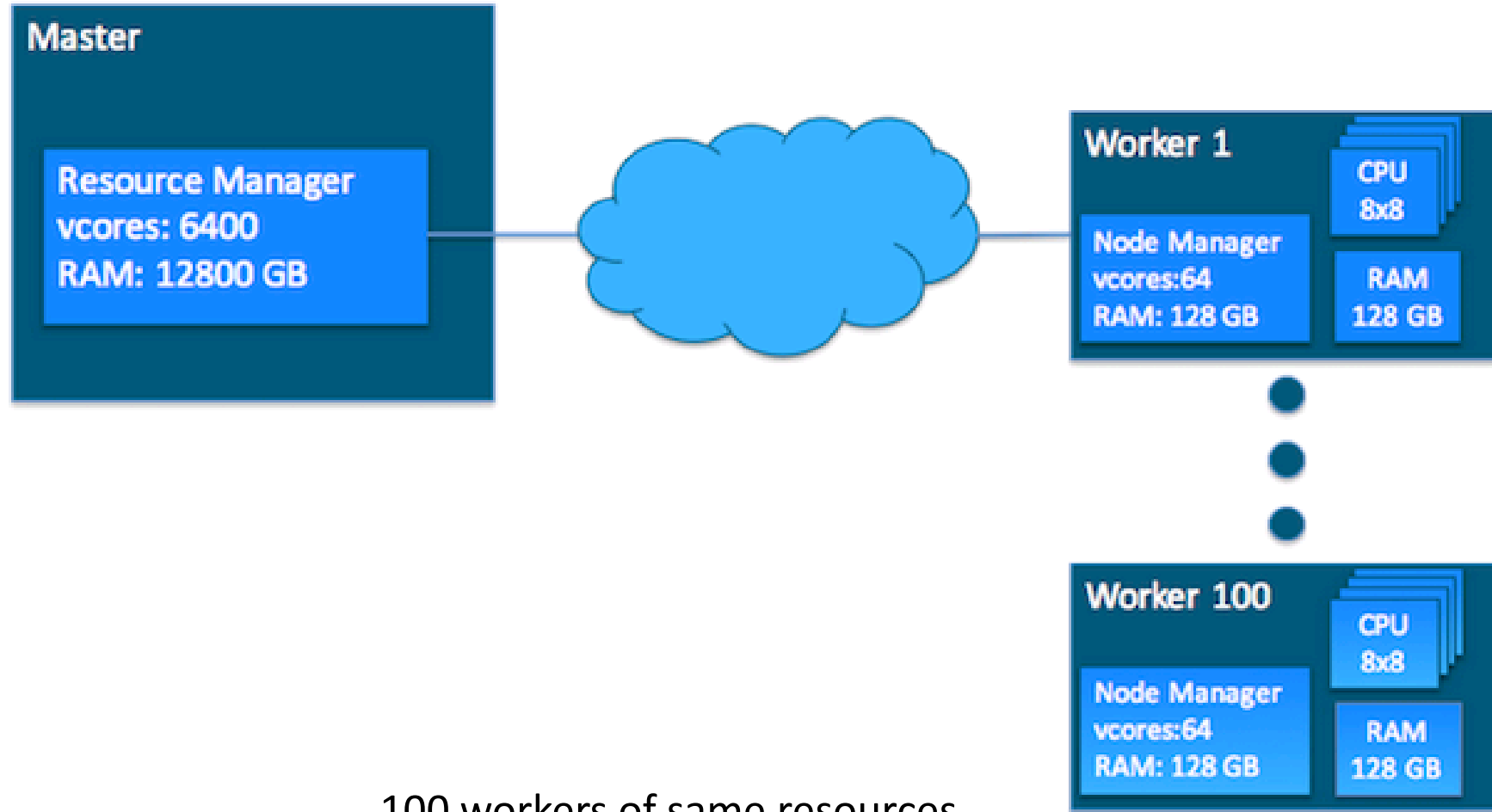
- In a *YARN cluster*, there are two types of hosts:
 - The *ResourceManager* is the master daemon that communicates with the client, tracks resources on the cluster, and orchestrates work by assigning tasks to *NodeManagers*.
 - A *NodeManager* is a worker daemon that launches and tracks processes spawned on worker hosts.



Yarn Resource Monitoring (i)

- YARN currently defines two resources:
 - *v-cores*
 - *Memory*
- Each *NodeManager* tracks
 - its own local resources and
 - communicates its resource configuration to the ResourceManager
- The *ResourceManager* keeps
 - a running total of the cluster's available resources.

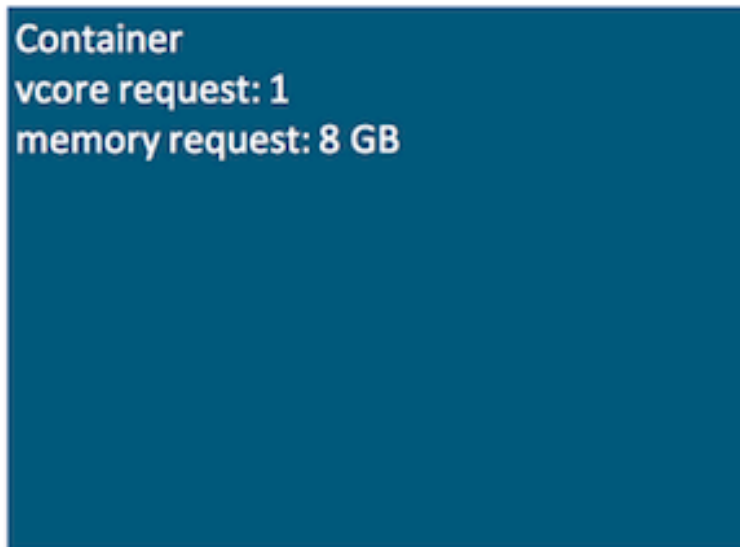
Yarn Resource Monitoring (ii)



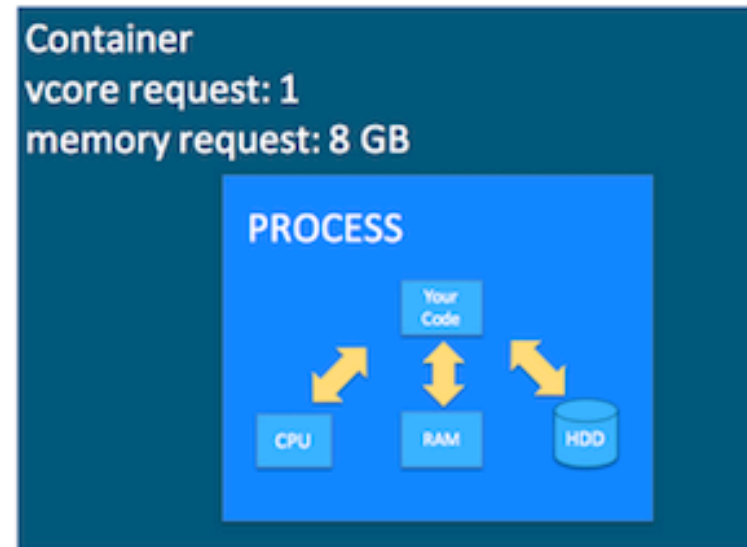
100 workers of same resources

Yarn Container

- Containers
 - a request to hold resources on the YARN cluster.
 - a container hold request consists of vcore and memory



Container as a hold



The task running as a process inside a container

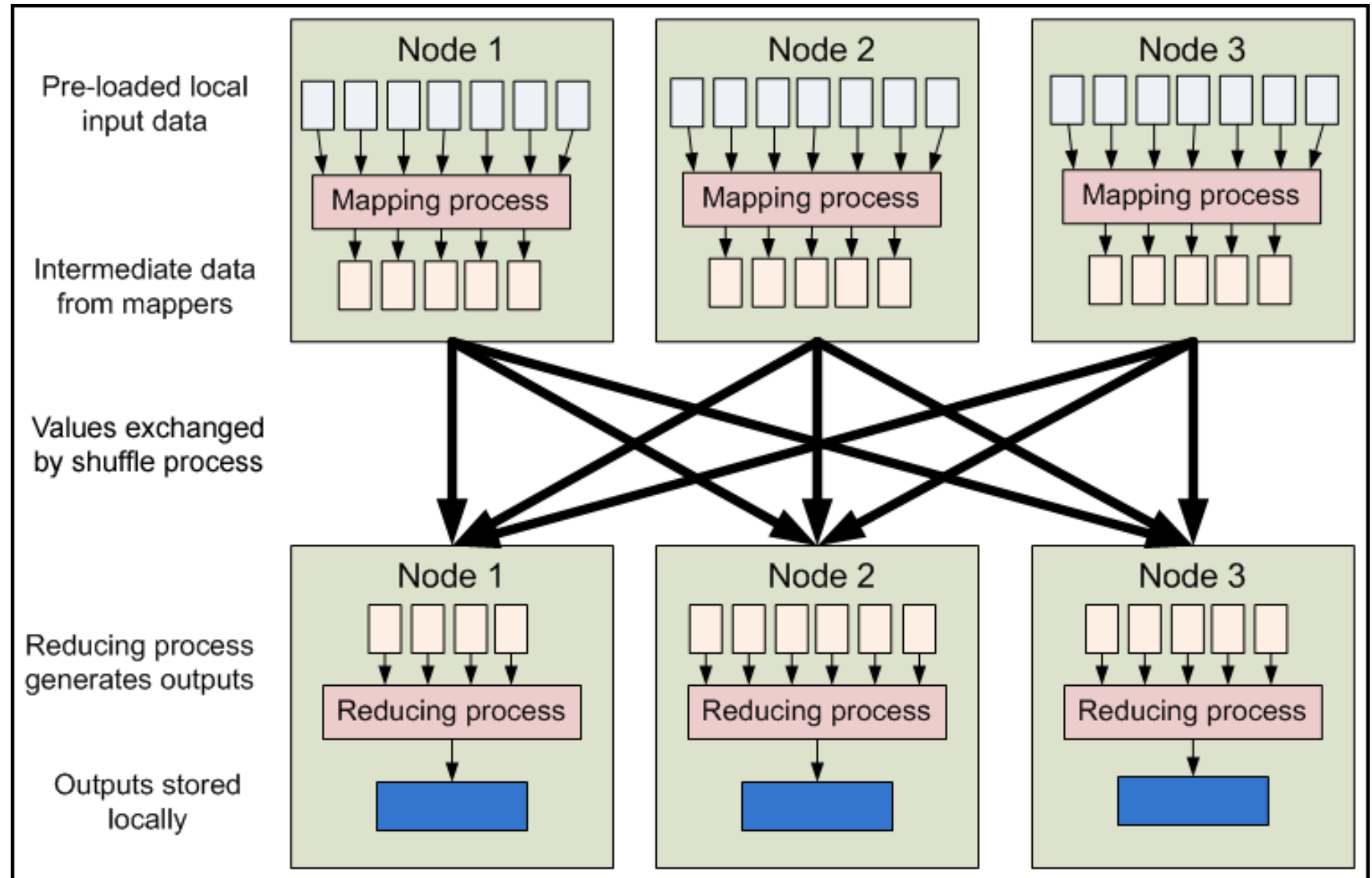
Yarn Application and ApplicationMaster

- Yarn application
 - It is a YARN client program that is made up of one or more tasks.
 - Example: MapReduce Application
- ApplicationMaster
 - It helps coordinate tasks on the YARN cluster for each running application.
 - It is the first process run after the application starts.

MapReduce - What?

- MapReduce is a programming model for efficient distributed computing
- It works like a Unix pipeline
 - `cat input | grep | sort | uniq -c | cat > output`
 - **Input** | **Map** | Shuffle & Sort | **Reduce** | **Output**
- Efficiency from
 - Streaming through data, reducing seeks
 - Pipelining
- A good fit for a lot of applications
 - Log processing
 - Web index building

MapReduce - Dataflow



MapReduce - Features

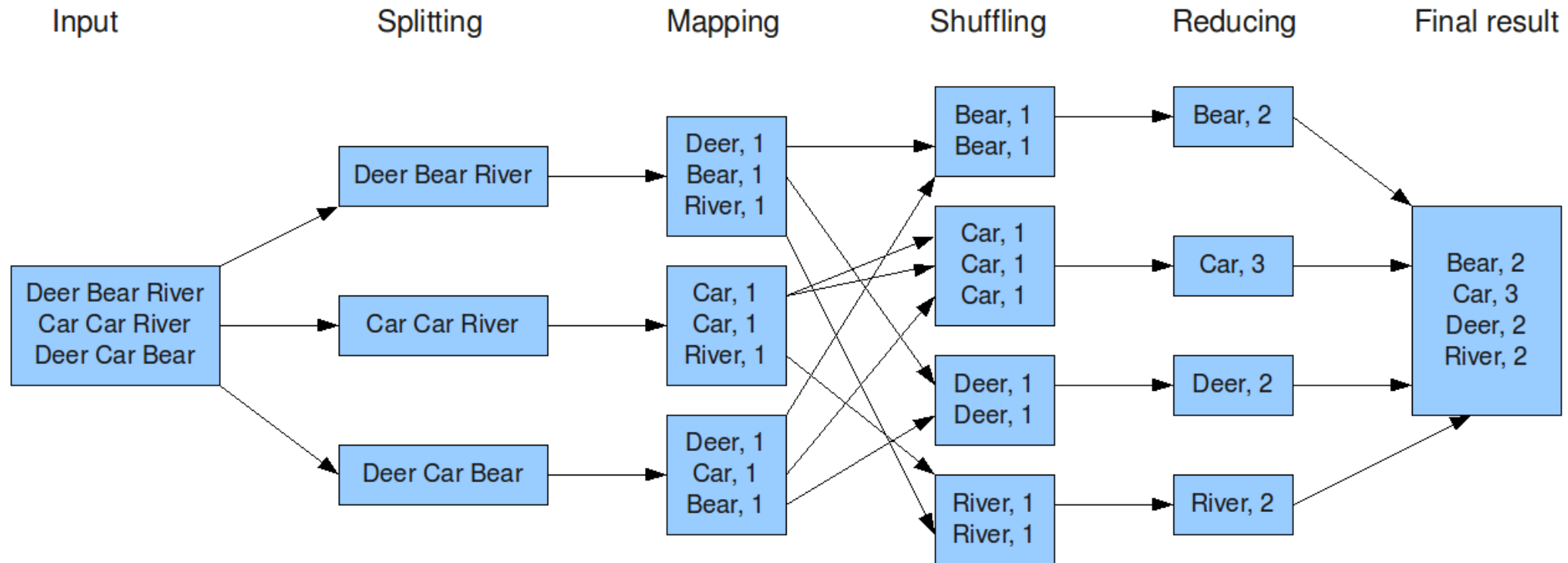
- Fine grained Map and Reduce tasks
 - Improved load balancing
 - Faster recovery from failed tasks
- Automatic re-execution on failure
 - In a large cluster, some nodes are always slow or flaky
 - Framework re-executes failed tasks
- Locality optimizations
 - With large data, bandwidth to data is a problem
 - Map-Reduce + HDFS is a very effective solution
 - Map-Reduce queries HDFS for locations of input data
 - Map tasks are scheduled close to the inputs when possible

Word Count Example

- Mapper
 - Input: value: lines of text of input
 - Output: key: word, value: 1
- Reducer
 - Input: key: word, value: set of counts
 - Output: key: word, value: sum
- Launching program
 - Defines this job
 - Submits job to cluster

Word Count Dataflow

The overall MapReduce word count process



Hadoop Related Subprojects

- Pig
 - High-level language for data analysis
- HBase
 - Table storage for semi-structured data
- Zookeeper
 - Coordinating distributed applications
- Hive
 - SQL-like Query language and Metastore
- Mahout
 - Machine learning

What is MapReduce?

- **MapReduce** is a library that runs an operation in parallel for you if you specify the input, map step and reduce step.

- Original Paper:

MapReduce: Simplified Data Processing on Large Clusters, Jeffrey Dean & Sanjay Ghemawat, OSDI '04

<http://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf>

Cited over 22,000 times!

MapReduce

- Goal: a *library* to make running *programs across multiple machines* easy
- Many *challenges* in *writing programs spanning many machines*, including:
 - Machines failing
 - Communicating over the network
 - Coordinating tasks
 - Etc.
- MapReduce handles these challenges! The programmer just needs to express their problem as a *MapReduce program* (*map* + *reduce* steps) and they can easily run it with the **MapReduce library**.
- Example of how the *right abstraction* can *revolutionize* computing
- An *open-source implementation* immediately appeared: *Hadoop* (not optimized well)

Remote Procedure Call (RPC)

- Request/response protocols are useful and widely used but rather clunky to use
 - e.g. need to define the set of requests, including how they are represented in network messages
- A nice abstraction is remote procedure call
 - Programmer simply invokes a procedure
 - But it executes on remote machine (the server)
 - By using RPC, programmers of distributed applications avoid the details of the interface with the network.
 - RPC subsystem handles message formats, sending and receiving, handling timeouts, etc

Remote Procedure Call (RPC)

- **Goal:** making the programming of distributed systems look similar, if not identical, to conventional programming – that is, achieving a high level of distribution transparency
- The role of underlying RPC system is to
 - hide important aspects of distribution, including the encoding and decoding of parameters and results
 - the passing of messages and the preserving of the required semantics for the procedure call
 - Programmers also do not need to know the programming language or underlying platform used to implement the service (an important step towards managing heterogeneity in distributed systems).

RPC call semantics

- **Maybe semantics:** With maybe semantics, the remote procedure call may be executed once or not at all. Maybe semantics arises when no fault-tolerance measures are applied
- **At-least-once semantics:** the invoker receives either a result, in which case the invoker knows that the procedure was executed at least once, or an exception informing it that no result was received
- **At-most-once semantics:** With at-most-once semantics, the caller receives either a result, in which case the caller knows that the procedure was executed exactly once, or an exception informing it that no result was received, in which case the procedure will have been executed either once or not at all.

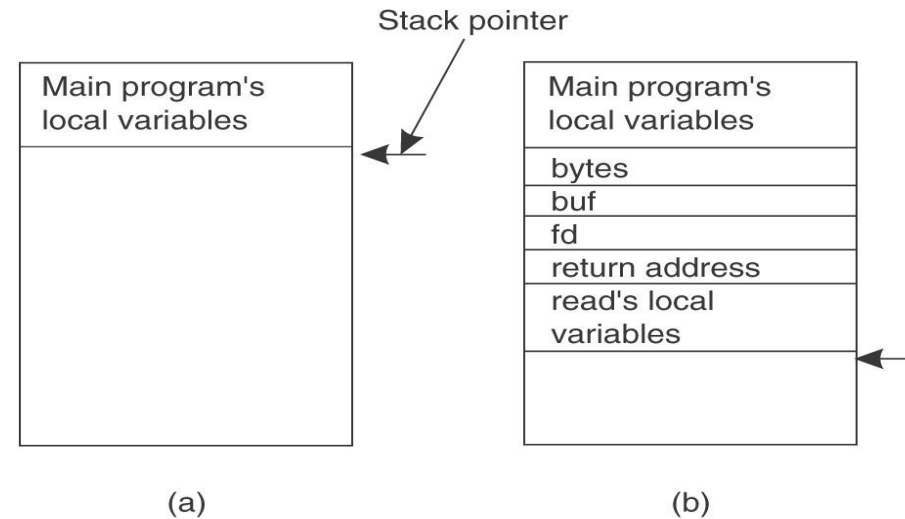
RPC implementation

When a process on *machine A* *calls* a procedure on machine *B*, *the calling process on A is suspended*, and execution of the called procedure takes place on *B*.

Information can be transported from the caller to the callee in the parameters and can come back in the procedure result.

No message passing at all is visible to the programmer. This method is known as Remote Procedure Call, or often just RPC.

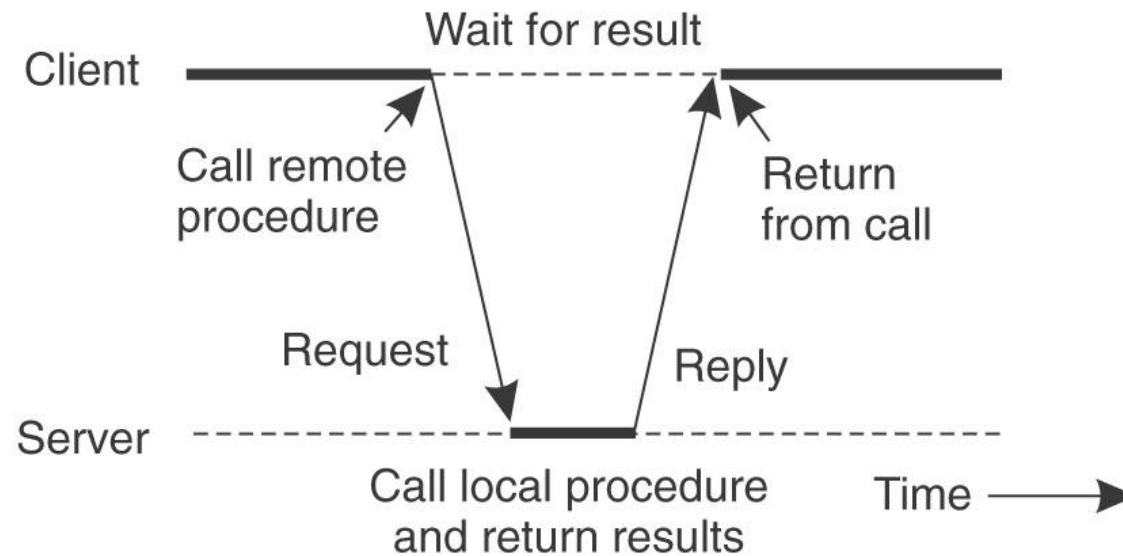
Ordinary procedure/function call



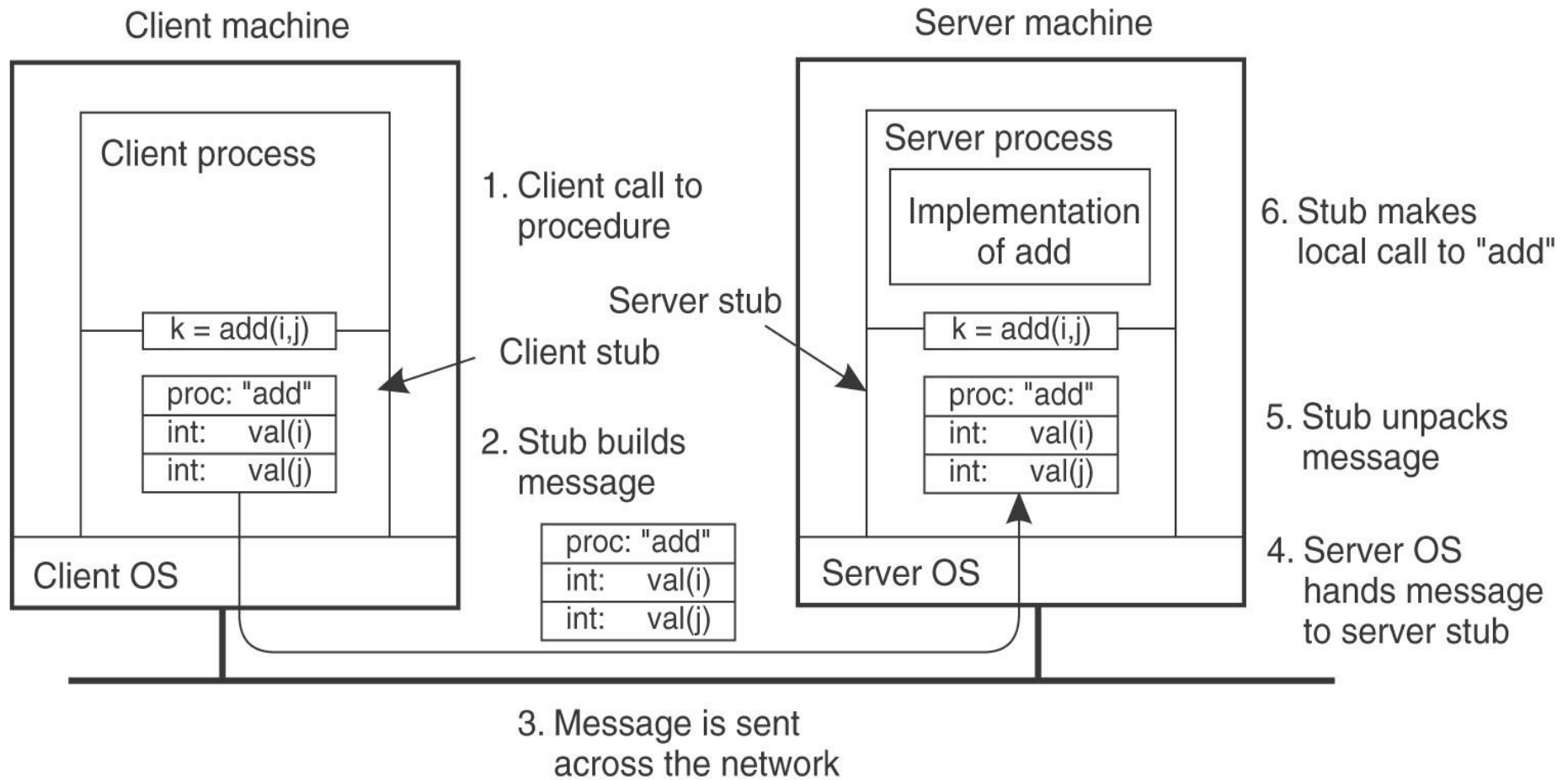
- `count = read(fd, buf, bytes)`
- After the read procedure has finished running, it puts the return value in a register, removes the return address, and transfers control back to the caller. The Caller then removes the parameters from the stack, returning the stack to the original state it had before the call.

Remote Procedure Call

- We would like to do the same if called procedure or function is on a remote server



RPC Stubs



Marshalling Arguments

- *Marshalling* is the packing of function parameters into a message packet
 - the RPC stubs call type-specific functions to *marshal* or *unmarshal* the parameters of an RPC
 - Client stub marshals the arguments into a message
 - Server stub unmarshals the arguments and uses them to invoke the service function
 - on return:
 - the server stub marshals return values
 - the client stub unmarshals return values, and returns to the client program

Issue #1 — representation of data

- Big endian vs. little endian

Solution

- Each stub converts machine representation to/from network representation
- Clients and servers must *not* try to cast data!

0	3	0	2	0	1	5	0
L	7	L	6	I	5	J	4

(a)

0	5	1	0	2	0	3	0
J	4	I	5	L	6	L	7

(b)

0	0	1	0	2	0	3	5
L	4	L	5	I	6	J	7

(c)

Issue #2 — Pointers and References

- Pointers are **only valid within one address space**
- Cannot be interpreted by another process

Even on same machine!

Solution:

1. Copy the array into the message and send it to the server.
2. The server stub can then call the server with a pointer to this array, even though this pointer has a different numerical value than the second parameter of read has.
3. Changes the server makes using the pointer (e.g., storing data into it) directly affect the message buffer inside the server stub.
4. When the server finishes, the original message can be sent back to the client stub, which then copies it back to the client.

References

1. Slides of Dr. Haroon Mahmood

Helpful Links (on gRPC):

1. <https://www.youtube.com/watch?v=gnchfOojMk4> (ByteByteGo)
2. <https://www.youtube.com/watch?v=XRXTsQwyZSU> (Stephane Maarek)
3. <https://www.youtube.com/watch?v=hVrwuMnCtok> (IBM)