Hadoop Internals for Oracle Developers and DBAs

Exploring the HDFS and MapReduce Data Flow

Tanel Põder
Enkitec

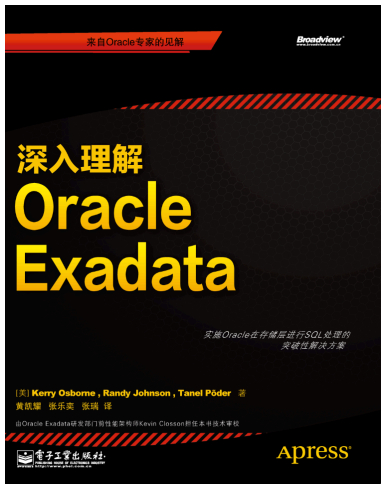
http://www.enkitec.com
http://blog.tanelpoder.com
@tanelpoder

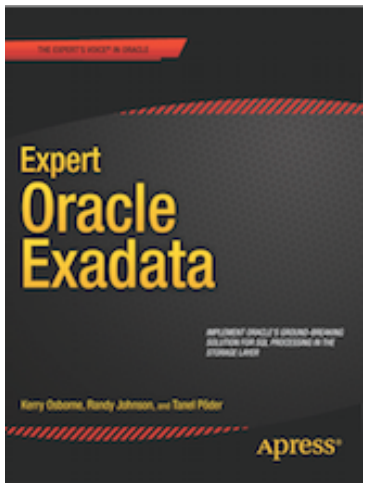
Intro: About me

- Tanel Põder
  - Former Oracle Database Performance geek
  - Present Exadata Performance geek
  - Aspiring Hadoop Perfomance geek

- This makes me feel old:
  - 20+ years of Linux
  - 15+ years of Oracle
  - 5+ years of Exadata
  - 1+ year of Hadoop

- My Hadoop interests
  - Performance stuff, DW offloading





Expert Oracle Exadata book
(with Kerry Osborne and Randy Johnson of Enkitec)

About Enkitec

- Enkitec
  - North America
  - UK, EMEA
- ~100 staff
  - In US, Europe
  - Consultants with Oracle experience of 15+ years on average
- What makes us so awesome ☺
  - 200+ Exadata implementations to date
  - Enkitec Exa-Lab ☺
    - We have 3 Exadatas (V2, X2-2, X3-2)
    - **Full-Rack Big Data Appliance**
    - Exalytics
    - ODA

ORACLE Platinum Partner

Cloudera Partner SI

Everything Exa

Planning/PoC
Implementation
Consolidation
Migration
Backup/Recovery
Patching
Troubleshooting
Performance
Capacity
Training

Oracle<->Hadoop

Enkitec's exalab: Big Data Appliance Hardware Features

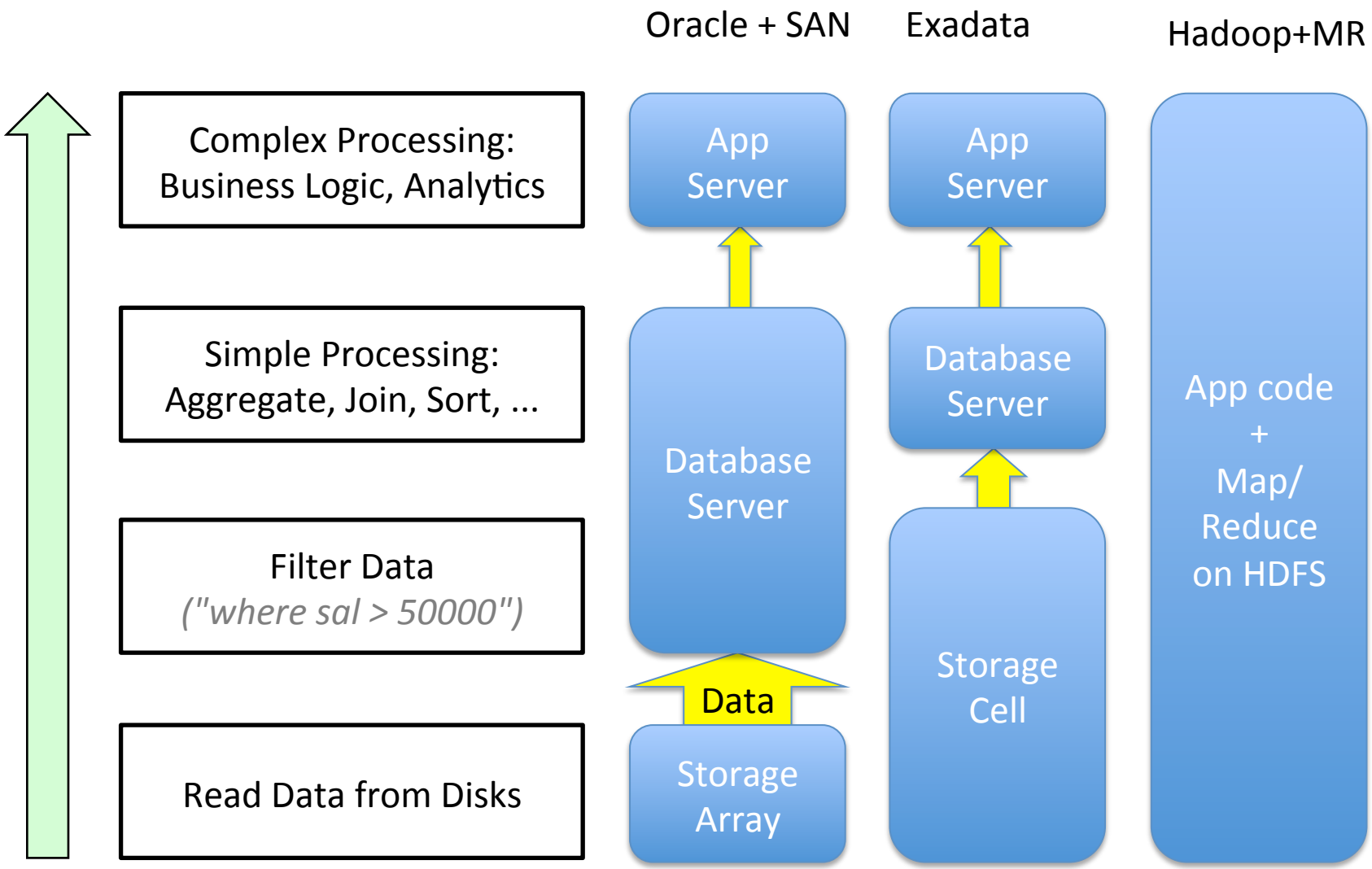
- A full rack of 18 servers
  - An engineered system
- 16 CPU cores per server
  - Total 288 cores in full rack
- 12 x 3 TB disks per server
  - 648 TB of disk space in rack
- Connected with InfiniBand
- **A Built in beer-holder!**

Why

also I am excited about

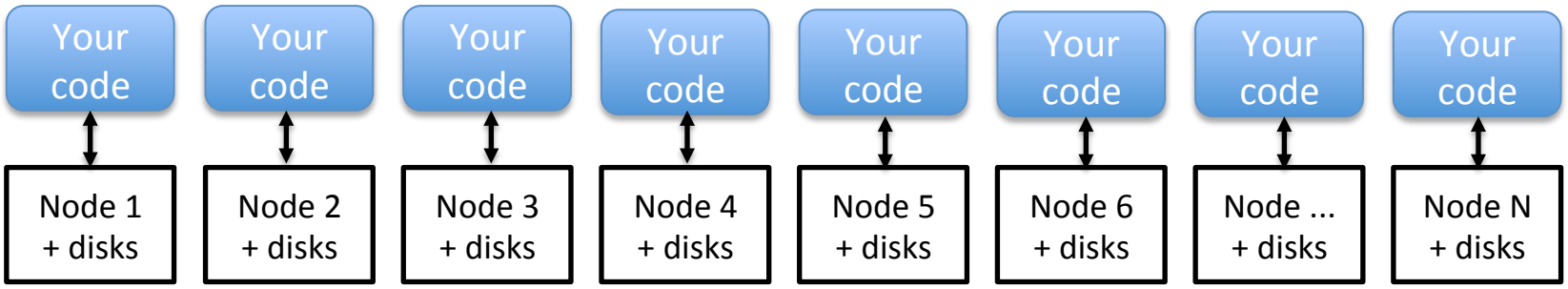
Hadoop?

Typical Data Processing



All processing can be close to data!

- One of the mantras of Hadoop:
  - *"Moving computation is cheaper than moving data"*
- MapReduce + HDFS hide the complexity of placing the computation of data on their local (or closest) cluster nodes
  - *No super-expensive interconnect network and complex scheduling & MPI coding needed*
  - ***No need for shared storage (expensive SAN + Storage Arrays!)***
- Now you can build a supercomputer (from commodity pizzaboxes) cheaply!



Oracle Database's internal "MapReduce"

- A *SELECT COUNT(*) FROM t* is not a real-life query
  - Gives very simple and scalable execution plan

- Similarly, a MapReduce task just counting words from a single file is not a real-life Hadoop job
  - But let's start from a simple example anyway

```
SQL> SELECT /*+ PARALLEL(4) */ COUNT(*) FROM t;

-------------------------------------------------------------------------------------------
| Id  | Operation              | Name     | E-Rows | Cost (%CPU)|    TQ  |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |          |        |   834 (100)|        |      |            |
|   1 |  SORT AGGREGATE        |          |      1 |            |        |      |            |
|   2 |   PX COORDINATOR       |          |        |            |        |      |            |
|   3 |    PX SEND QC (RANDOM) | :TQ10000 |      1 |            |  Q1,00 | P->S | QC (RAND)  |
|   4 |     SORT AGGREGATE     |          |      1 |            |  Q1,00 | PCWP |            |
|   5 |      PX BLOCK ITERATOR |          |    598K|   834   (1)|  Q1,00 | PCWC |            |
|*  6 |       TABLE ACCESS FULL| T        |    598K|   834   (1)|  Q1,00 | PCWP |            |
-------------------------------------------------------------------------------------------
```

Simple "MapReduce" behavior in Oracle

- QC spawns 4 slaves and *maps* out work (PX granules) to them
  1. Slaves read only the PX granules "allocated" to them for reading by the PX BLOCK ITERATOR
  2. Slaves do the computation they can (count) and send results to QC
  3. QC *reduces* the 4 slaves results to one (sums the counts together)

```
-------------------------------------------------------------
| Id  | Operation                | Name     |TQ      |IN-OUT|
-------------------------------------------------------------
|   0 | SELECT STATEMENT         |          |        |      |
|   1 |  SORT AGGREGATE          |          |        |      |
|   2 |   PX COORDINATOR         |          |        |      |
|   3 |    PX SEND QC (RANDOM)   | :TQ10000 |  Q1,00 | P->S |
|   4 |     SORT AGGREGATE       |          |  Q1,00 | PCWP |
|   5 |      PX BLOCK ITERATOR   |          |  Q1,00 | PCWC |
|*  6 |       TABLE ACCESS FULL  | T        |  Q1,00 | PCWP |
-------------------------------------------------------------

   6 - access(:Z>=:Z AND :Z<=:Z)
```

Each slave essentially executes the same cursor, but the PX BLOCK ITERATOR (and the access predicate below) return different Data Block Address ranges to scan.

If an inter-instance PX SQL, the cursor has to be shipped to the library cache of all executing nodes

Measuring PX Data Flow – V$PQ_TQSTAT

- V$PQ_TQSTAT shows the dataflow stats between slave sets
  - Producer <-> Consumer
  - "Map" <-> "Reduce"

```
SQL> @tq
Show PX Table Queue statistics from last Parallel Execution in this
   session...

TQ_ID      TQ_FLOW
(DFO,SET)  DIRECTION    NUM_ROWS       BYTES       WAITS   TIMEOUTS PROCESS
---------- --------- ---------- ---------- ---------- ---------- -------
:TQ1,0000  Producer           1         32         13          0 P003
           Producer           1         32         14          0 P001
           Producer           1         32         14          0 P002
           Producer           1         32         11          0 P000

           Consumer           4        128         56         17 QC
```

http://blog.tanelpoder.com/files/scripts/tq.sql

More complex "MapReduce" behavior in Oracle

- The parallel query has more levels now – two table queues

```
SELECT /*+ PARALLEL(4) */ owner, COUNT(*) FROM t GROUP BY owner

Plan hash value: 129087698

-------------------------------------------------------------------------------
| Id  | Operation                | Name     |    TQ  |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |          |        |      |            |
|   1 |  PX COORDINATOR          |          |        |      |            |
|   2 |   PX SEND QC (RANDOM)    | :TQ10001 |  Q1,01 | P->S | QC (RAND)  |
|   3 |    HASH GROUP BY         |          |  Q1,01 | PCWP |            |
|   4 |     PX RECEIVE           |          |  Q1,01 | PCWP |            |
|   5 |      PX SEND HASH        | :TQ10000 |  Q1,00 | P->P | HASH       |
|   6 |       HASH GROUP BY      |          |  Q1,00 | PCWP |            |
|   7 |        PX BLOCK ITERATOR |          |  Q1,00 | PCWC |            |
|*  8 |         TABLE ACCESS FULL| T        |  Q1,00 | PCWP |            |
-------------------------------------------------------------------------------
```

Data Flow between Slave Sets (Stages) via memory or UDP

What is the Hadoop *physically*?

- Hadoop is an ecosystem of many (mainly) Java programs

- We will be talking about HDFS and MapReduce
  - HDFS = Hadoop's Distributed File System
  - MapReduce = A job placement, scheduling and data flow library
  - ... they are also just some Java programs

- *Lots* of JVMs!
  - Some run as daemons (various metadata servers)
  - Some get started & stopped for each MapReduce Job

  - And some higher level programs (like Hive, Pig) generate & run map-reduce jobs internally

HDFS processes

```
$ ps auxwww | grep hdfs
hdfs      2860  0.2  4.0 790500 159808 ?       Sl   19:28   0:30 /usr/java/
    jdk1.6.0_31/bin/java -Dproc_namenode -Xmx1000m -
    Dhdfs.audit.logger=INFO,RFAAUDIT -Dsecurity.audit.logger=INFO,RFAS -
    Djava.net.preferIPv4Stack=true -Dhadoop.log.dir=/var/log/hadoop-hdfs -
    Dhadoop.log.file=hadoop-cmf-hdfs1-NAMENODE-localhost.localdomain.log.out -
    Dhadoop.home.dir=/usr/lib/hadoop -Dhadoop.id.str=hdfs -
    Dhadoop.root.logger=INFO,RFA -Djava.library.path=/usr/lib/hadoop/lib/native -
    Dhadoop.policy.file=hadoop-policy.xml -Djava.net.preferIPv4Stack=true -
    Xmx181542808 -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:-
    CMSConcurrentMTEnabled -XX:CMSInitiatingOccupancyFraction=70
    org.apache.hadoop.hdfs.server.namenode.NameNode

hdfs      2918  0.2  3.5 772016 139824 ?       Sl   19:28   0:30 /usr/java/
    jdk1.6.0_31/bin/java -Dproc_datanode -Xmx1000m -
    Dhdfs.audit.logger=INFO,RFAAUDIT -Dsecurity.audit.logger=INFO,RFAS -
    Djava.net.preferIPv4Stack=true -Dhadoop.log.dir=/var/log/hadoop-hdfs -
    Dhadoop.log.file=hadoop-cmf-hdfs1-DATANODE-localhost.localdomain.log.out -
    Dhadoop.home.dir=/usr/lib/hadoop -Dhadoop.id.str=hdfs -
    Dhadoop.root.logger=INFO,RFA -Djava.library.path=/usr/lib/hadoop/lib/native -
    Dhadoop.policy.file=hadoop-policy.xml -Djava.net.preferIPv4Stack=true -server
    -Xmx181542808 -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:-
    CMSConcurrentMTEnabled -XX:CMSInitiatingOccupancyFraction=70
    org.apache.hadoop.hdfs.server.datanode.DataNode
```

HDFS Libraries & Config

```
$ ls -l /usr/lib/hadoop-hdfs/
total 6108
drwxr-xr-x. 2 root root    4096 Jul 15 03:33 bin
drwxr-xr-x. 2 root root    4096 Jul 15 03:33 cloudera
-rw-r--r--. 1 root root 4534749 May 27 20:00 hadoop-hdfs-2.0.0-cdh4.3.0.jar
-rw-r--r--. 1 root root 1692271 May 27 20:00 hadoop-hdfs-2.0.0-cdh4.3.0-tests.jar
lrwxrwxrwx. 1 root root      30 Jul 15 03:33 hadoop-hdfs.jar -> hadoop-hdfs-2.0.0-
                                                                cdh4.3.0.jar
```

```
# cat /etc/hadoop/conf/hdfs-site.xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<property>
    <name>dfs.namenode.http-address</name>
    <value>localhost.localdomain:50070</value>
  </property>
  <property>
    <name>dfs.replication</name>
    <value>3</value>
  </property>
  <property>
    <name>dfs.blocksize</name>
    <value>134217728</value>
  </property>
```

HDFS low-level structure

- It's just a bunch of regular files in an OS directory!
  - Residing on a regular *local* OS filesystem (like EXT3, EXT4, XFS, etc)

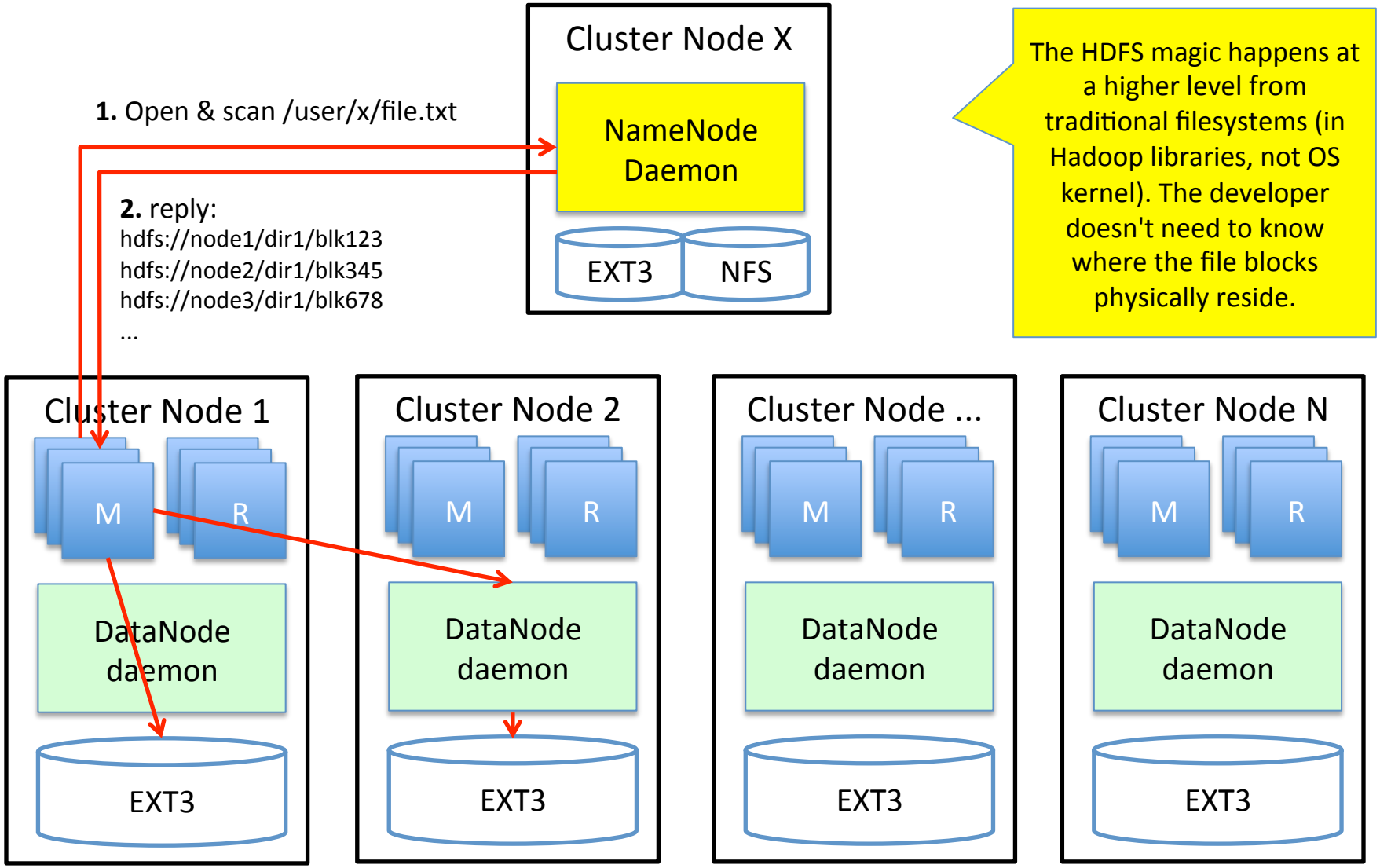
```
$ cd /dfs
$ ls -l
total 12
drwxr-xr-x. 3 hdfs hadoop 4096 Aug  5 19:29 dn
drwx------. 3 hdfs hadoop 4096 Aug  5 19:28 nn
drwx------. 3 hdfs hadoop 4096 Aug  5 19:28 snn
$
$ du -hs *
552M  dn
2.2M  nn
152K  snn

$ ls -lR dn
dn/current/BP-763425243-127.0.0.1-1373884718246/current/finalized/subdir45:
total 504672
-rw-r--r-- 1 hdfs hdfs       134 Aug  5 21:43 blk_1335263265743075945
-rw-r--r-- 1 hdfs hdfs        11 Aug  5 21:43 blk_1335263265743075945_1231.meta
-rw-r--r-- 1 hdfs hdfs        38 Aug  5 20:13 blk_-181798089218823916
-rw-r--r-- 1 hdfs hdfs 134217728 Aug  5 20:39 blk_2806839731572335391
-rw-r--r-- 1 hdfs hdfs   1048583 Aug  5 20:39 blk_2806839731572335391_1225.meta
```

dn = DataNode (actual data)
nn = NameNode (file metadata)
snn = Secondary NameNode

HDFS names the OS-level files and splits large ones to "blocks" (dfs.blocksize) across multiple nodes

.meta files hold checksums for the file

HDFS high-level picture



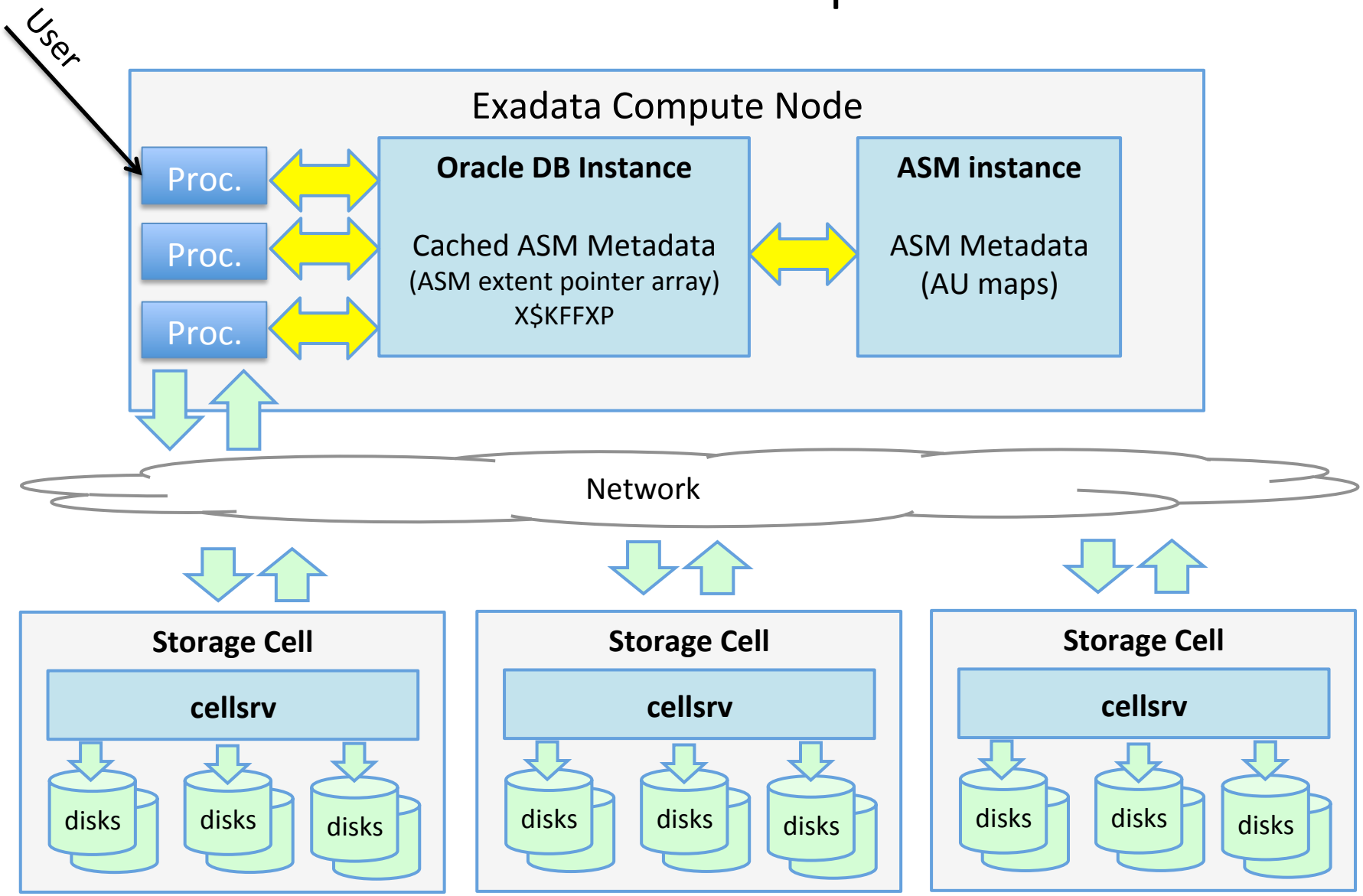
HDFS Design and Tradeoffs

- HDFS is designed for "write once, read many" use cases
- Large sequential streaming reads (scans) of files
  - Not optimal for random seeks and small IOs
- "Inserts" are appended to the end of the file
- No updates
- HDFS performs the replication ("mirroring") and is designed anticipating frequent DataNode failures

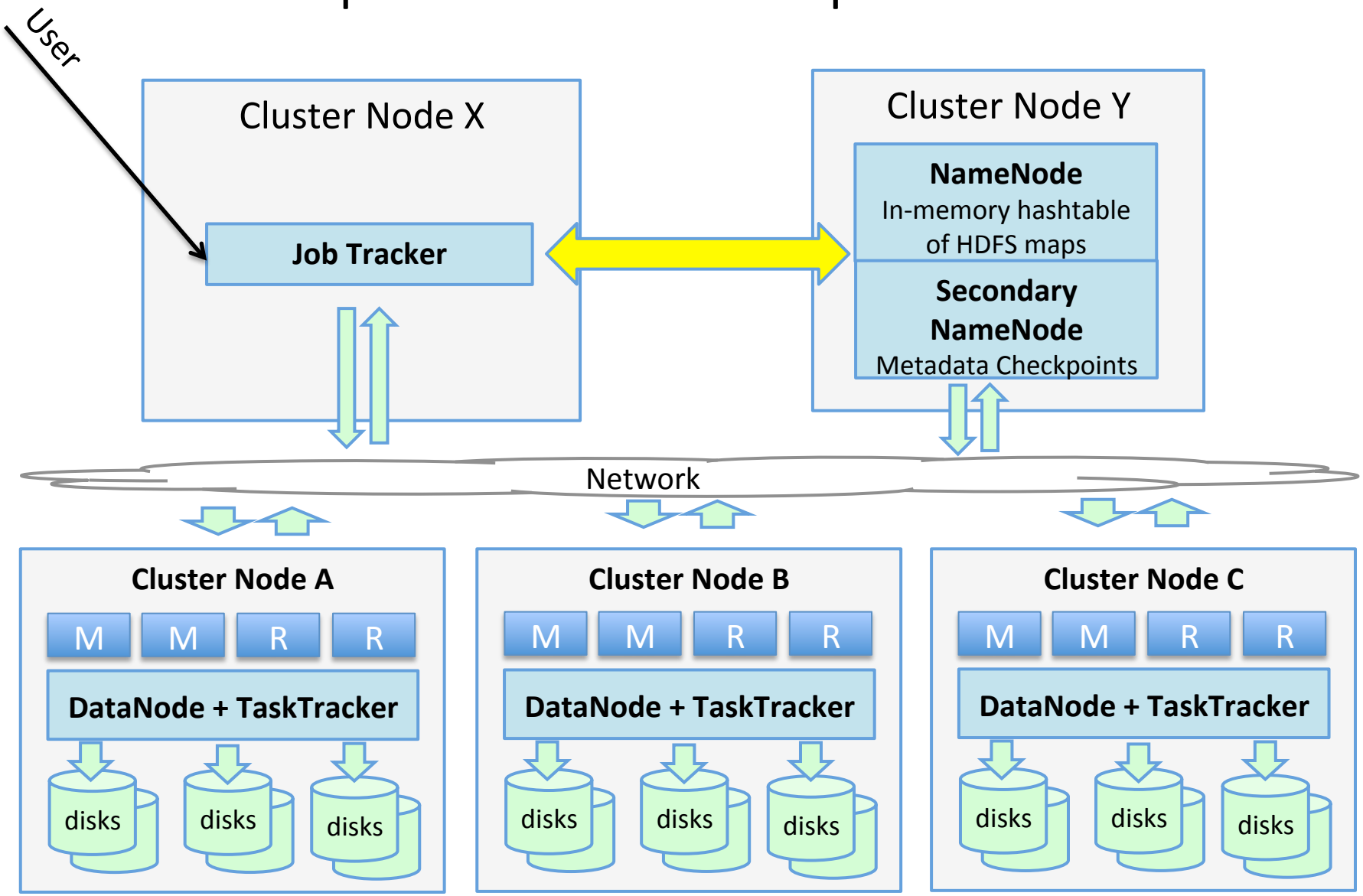
HBASE supports this thanks to its indexed file format + cache

HBASE supports random lookups and updates

Exadata IO flow & address space translation



Hadoop IO flow & address space translation



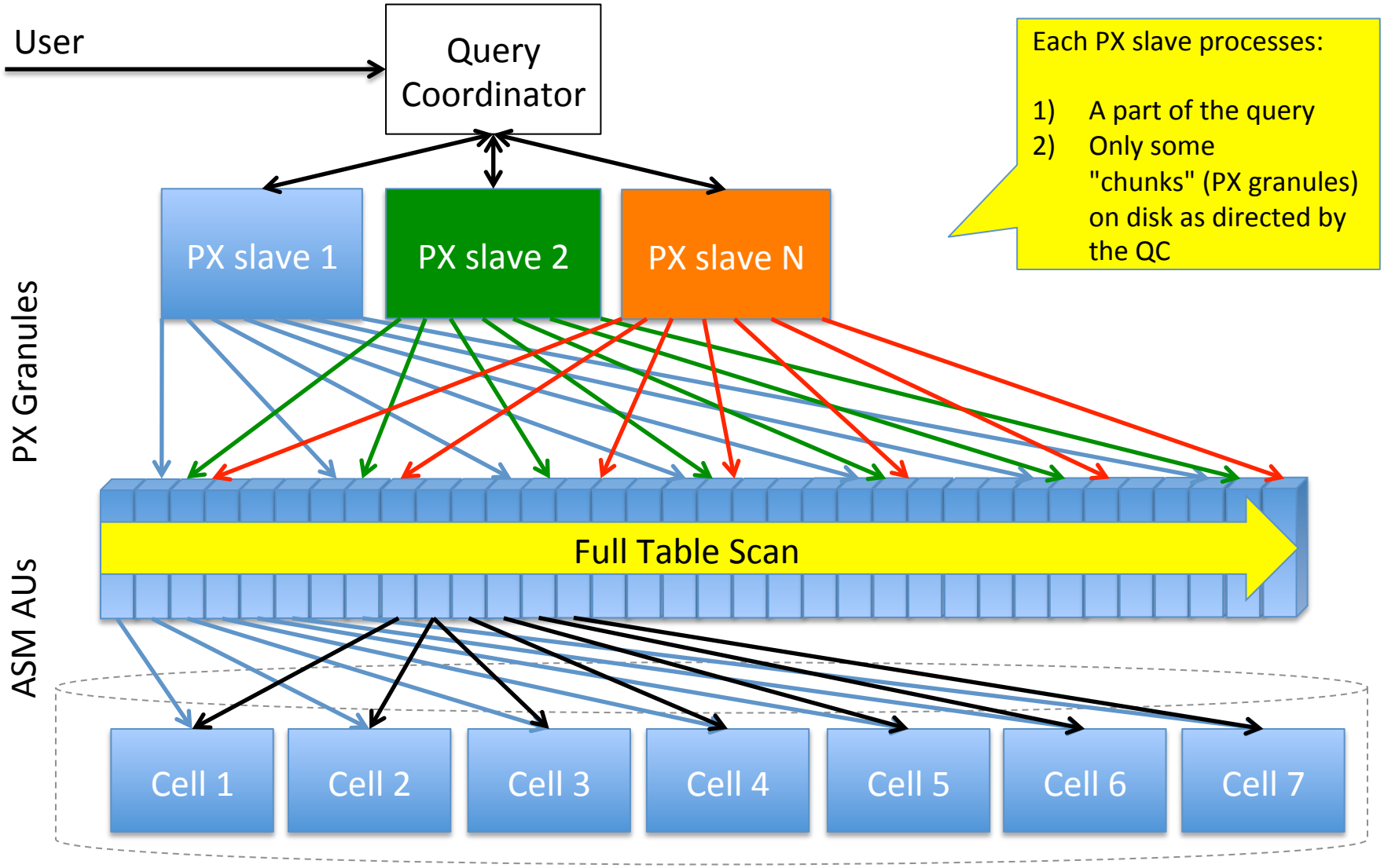
Hadoop Jobs vs Tasks

- If Hadoop ***Jobs*** were Oracle SQL statements executed in parallel
  - The JobTracker would be like Oracle's **Query Coordinator**
- … then ***Tasks*** would be like the Parallel Execution Slaves*
  - Max amount of concurrently running Tasks configured by task slots
  - An input-split (range of data) passed to a Task for mapping is like the PX-granule in Oracle

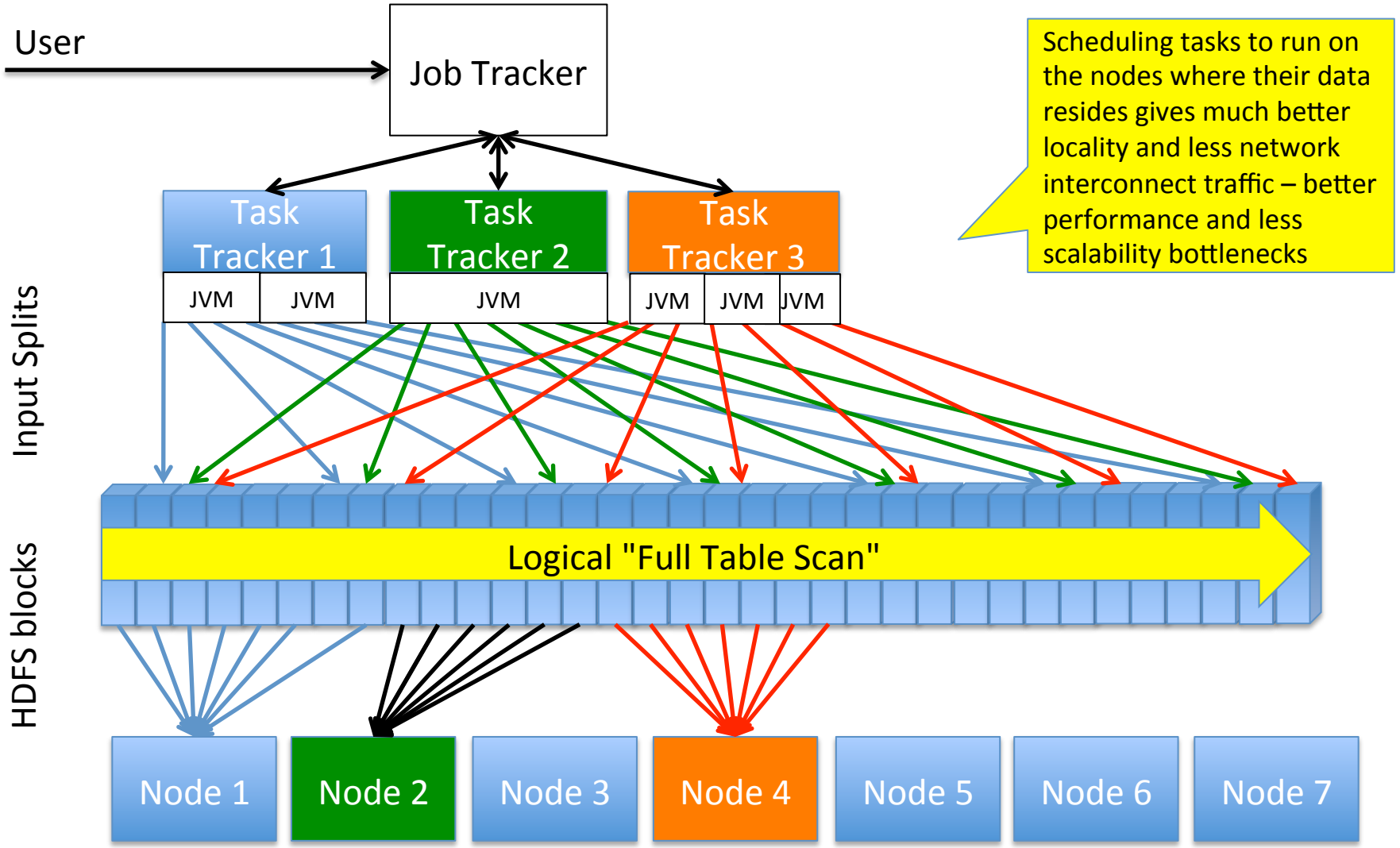
Hadoop Job Placement

- A Hadoop Job gets split into many Tasks for distributed parallel processing:
  - The JobTracker first tries to schedule the Tasks on the nodes where the corresponding data resides
  - If there are no available Task Slots on the node, then another node (in the same rack is picked)

Logical view: Oracle Exadata Parallel Full Table Scan



Logical view: Hadoop "Parallel Full Table Scan"



What is Hadoop MapReduce physically, anyway?

- MapReduce is just a Java library which can talk to HDFS, NameNodes, Job Trackers etc
  - So your (Java) app needs to use the MapReduce library
  - Somewhat like using JDBC to run SQL
  - ... you would use MapReduce to scan (HDFS) files
  - Note that MapReduce does not *have* to work with HDFS, many other data sources are supported (like FTP, HTTP, Amazon S3 etc)

- You can call the MapReduce library directly in Java (and other JVM-based languages)
  - The MapReduce library logic and application logic would reside in the same JVM – less IPC and communication overhead
  - Indirect APIs for C, Python, etc also available

A typical MapReduce app run

1. Your MapReduce app with required libraries packed into JAR
2. `$ hadoop jar app.jar MainClass arg1 arg2`
3. The app.jar gets copied to HDFS, replicated
4. The Task Trackers copy the JAR to local tmp directory
5. The Task Tracker launches a new JVM with the JAR and Class name to execute
6. Your code in the JAR will take over and call out HDFS, input, output libraries as needed
7. After X seconds (minutes, hours) you will have an output file in HDFS

It is possible to cache the JARs for future use using DistributedCache and even reuse JVMs for next task runs

Thanks!!!

- Questions?
  - Ask now :)
- Or Contact
  - tanel@tanelpoder.com
  - http://blog.tanelpoder.com
  - @tanelpoder
- http://www.enkitec.com
  (we rock! ;-)