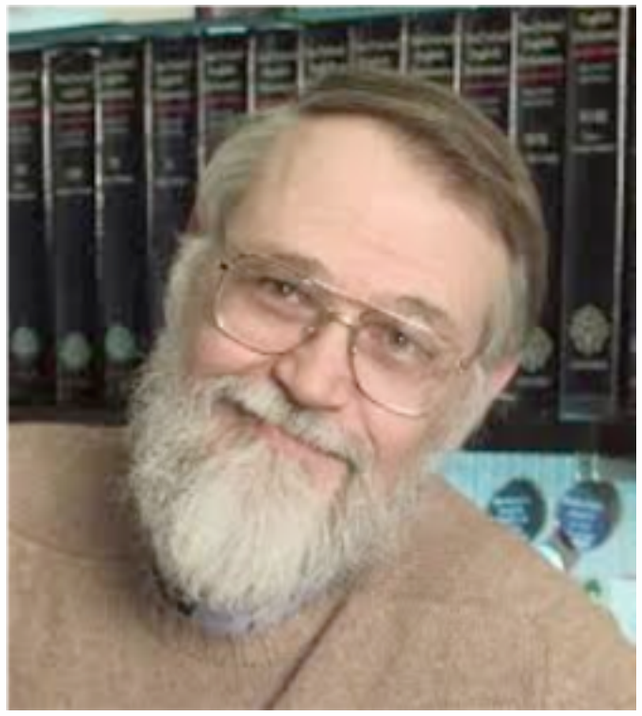
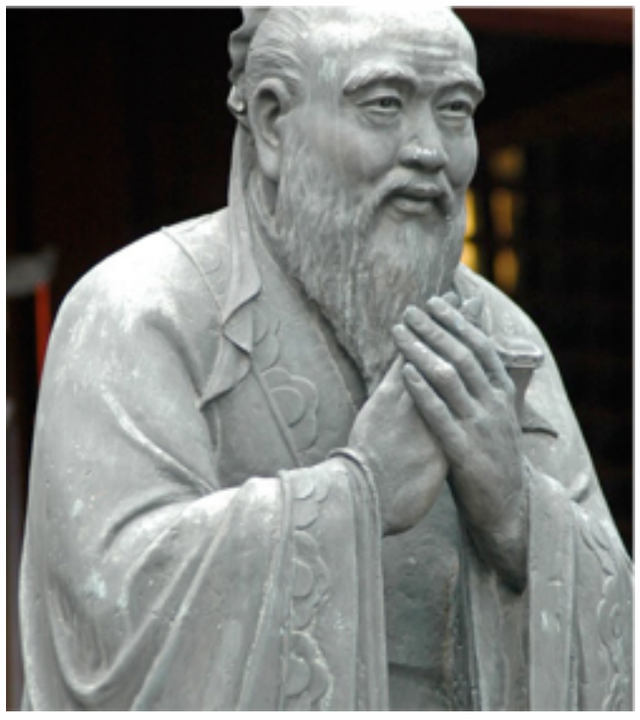
Scala in Hulu's Data Platform

Prasan Samtani
prasan.samtani@hulu.com

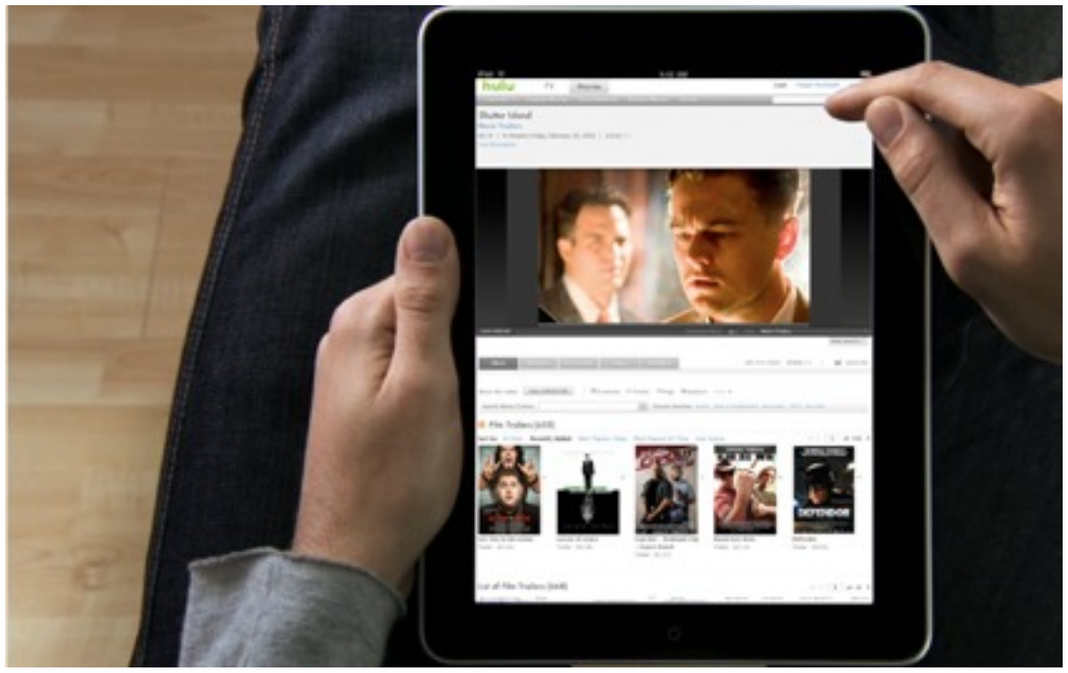
Disclaimer

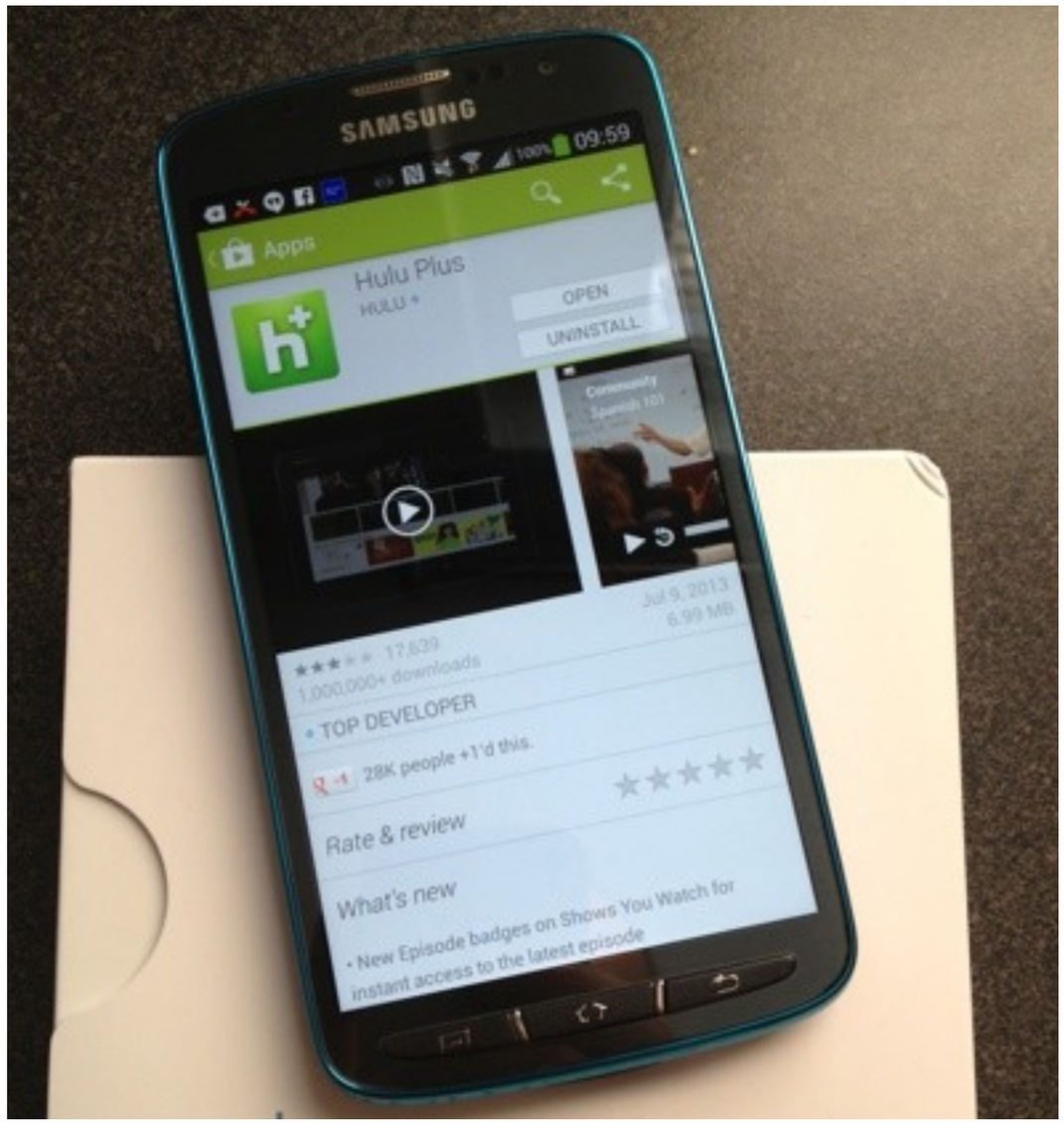




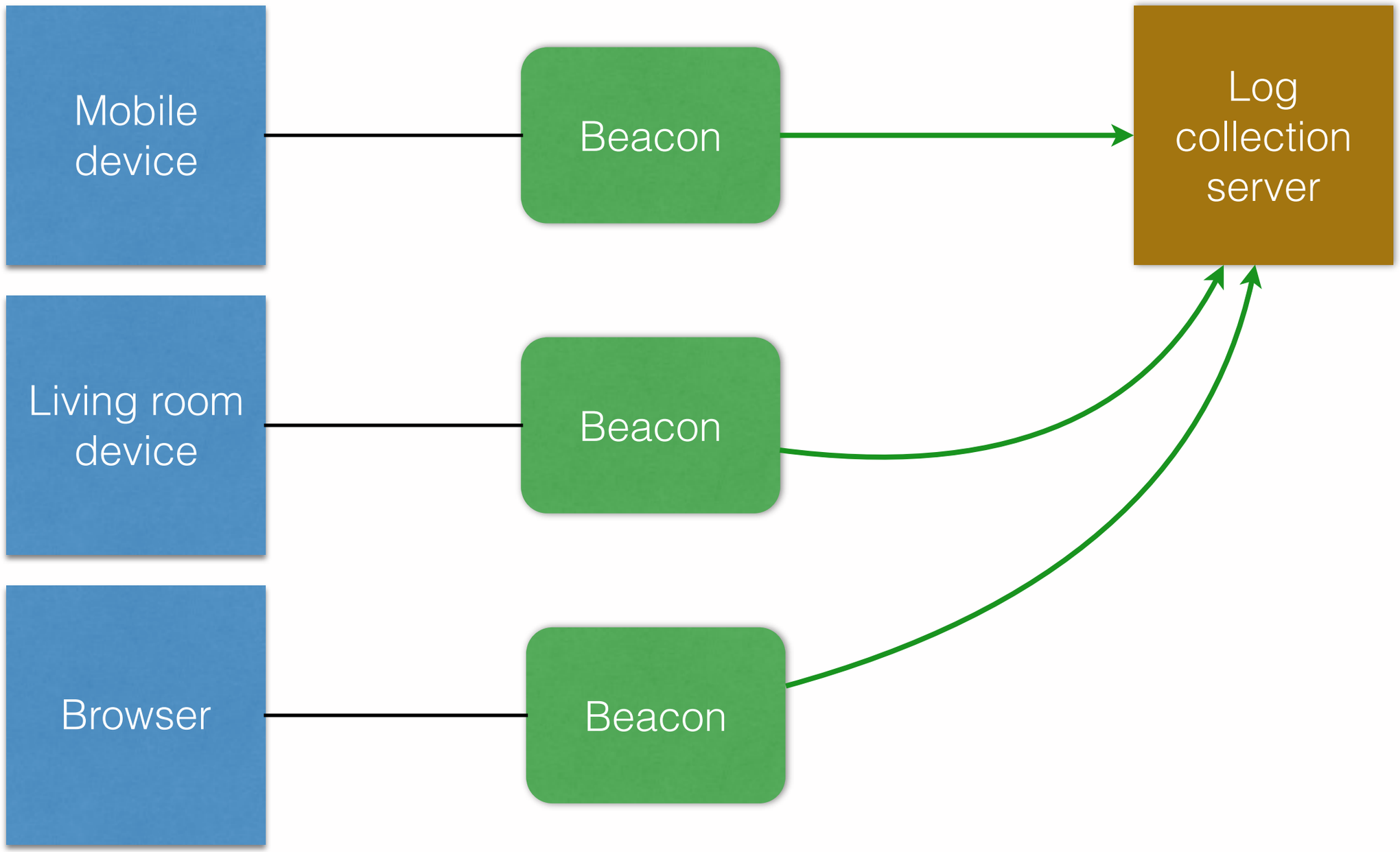
hulu™

- Streaming video service
- Next-day same-season TV
- ~4.5 million subscribers
- ~25 million unique visitors per month
- > 1 billion ads/month





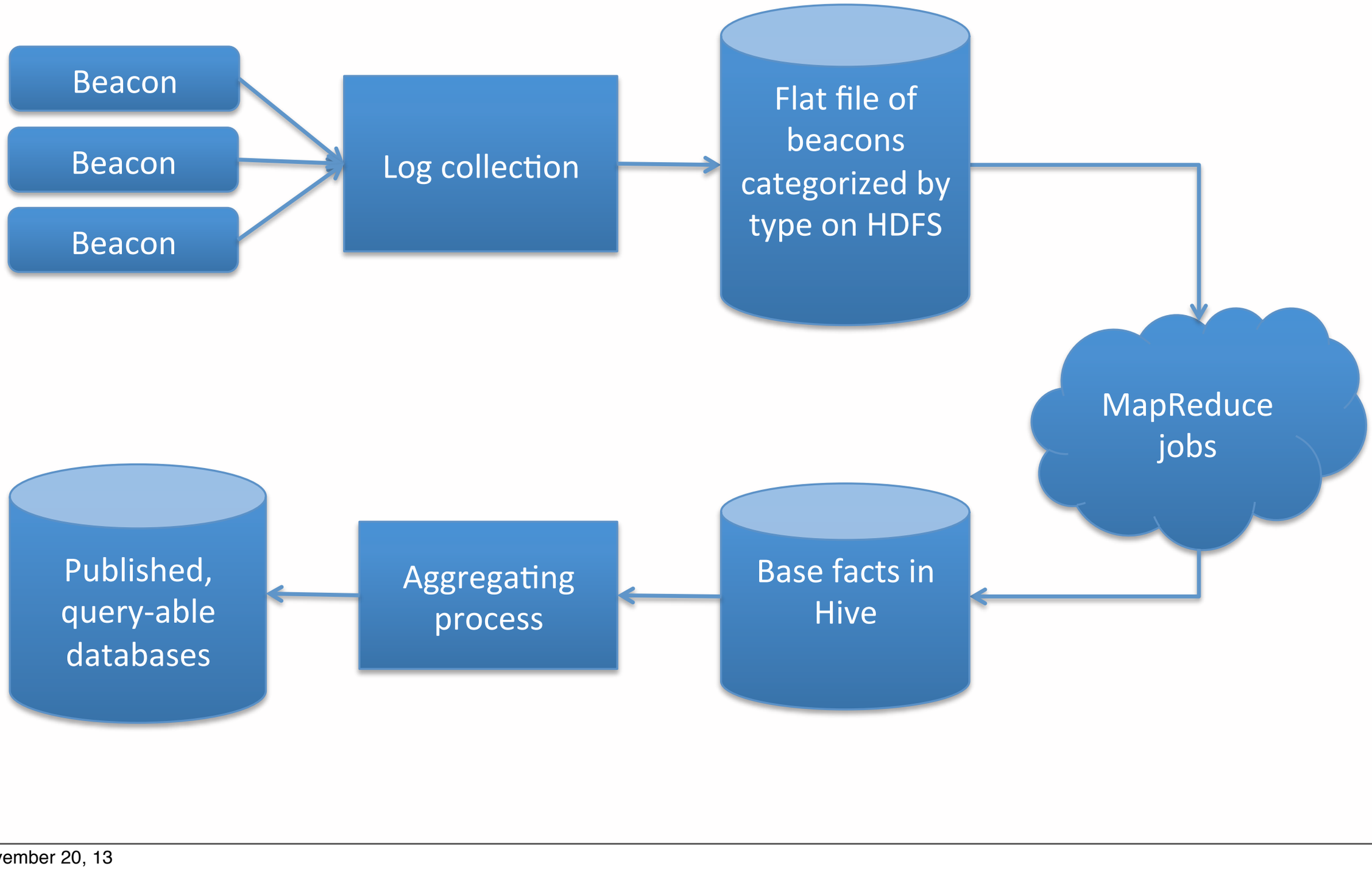
Beacons


Mobile device
Beacon
Log collection server
Living room device
Beacon
Browser
Beacon

What's in a beacon

80 2013-04-01 00:00:00
/v3/playback/start?
bitrate=650
&cdn=Akamai
&channel=Anime
&clichéent=Explorer
&computerguid=EA8FA1000232B8F6986C3E0BE55E9333
&contentid=5003673
…

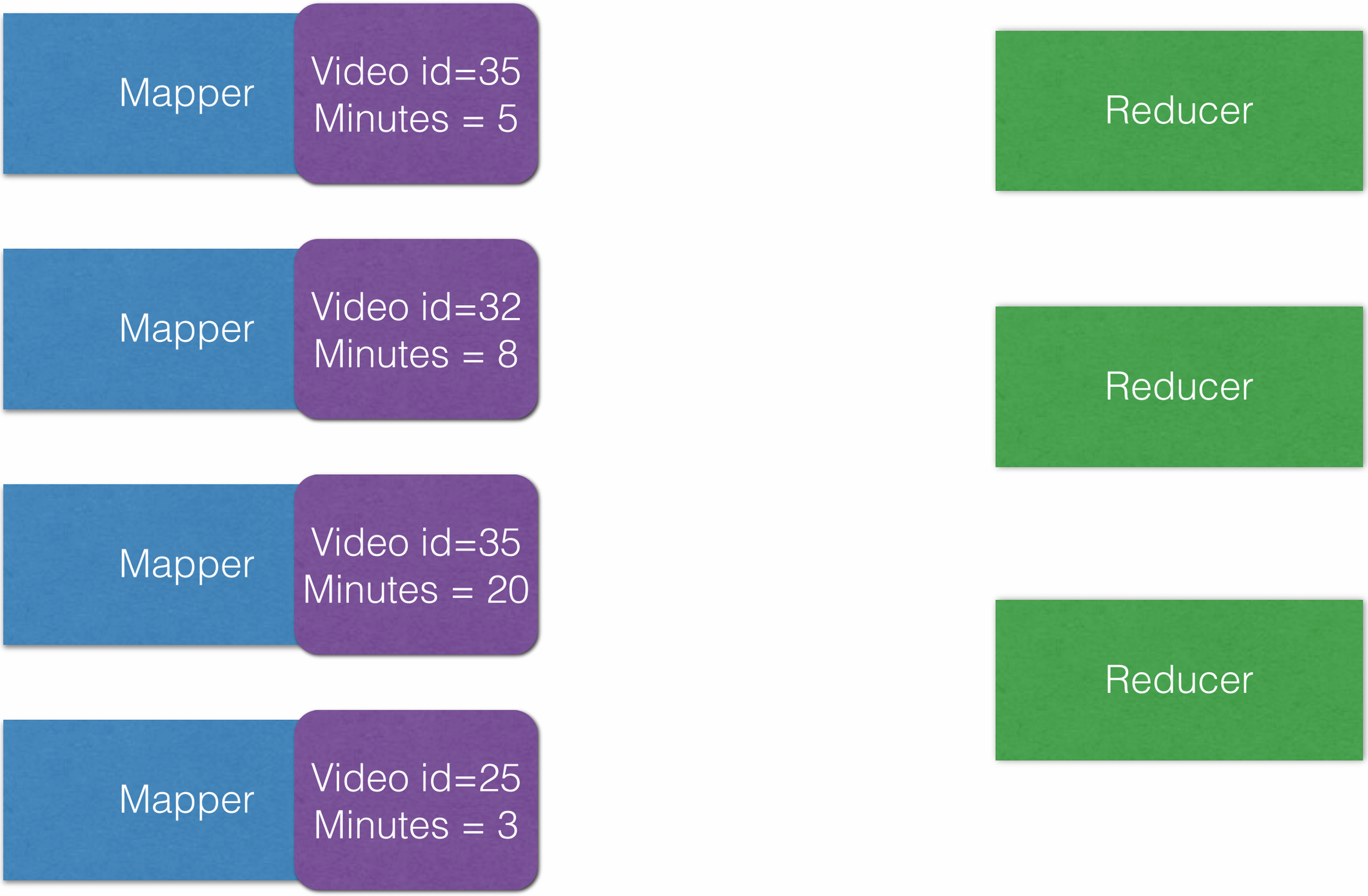
Data pipeline



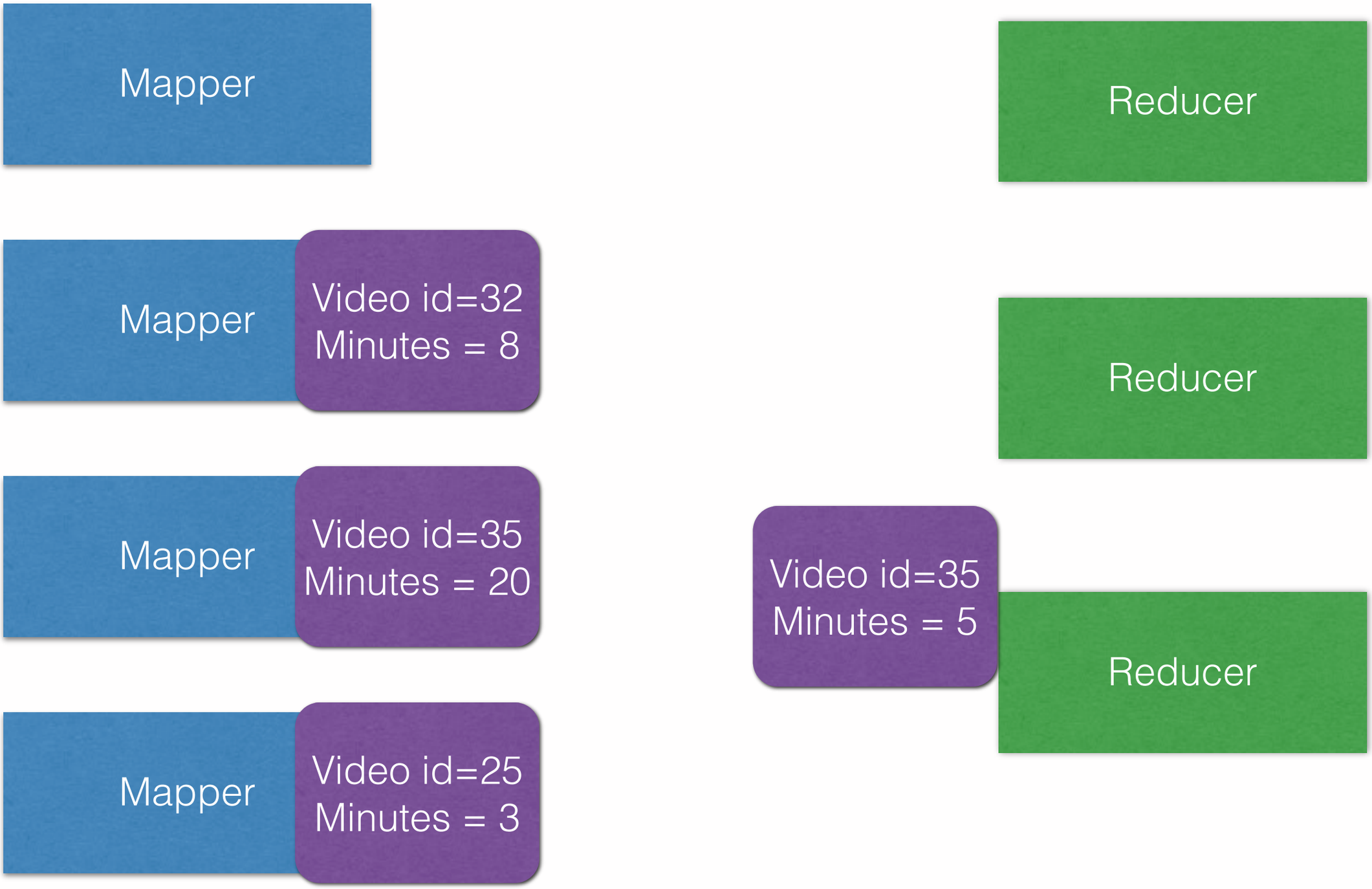
Base-facts

computerguid	***00-755925-925755-925***
userid	5238518
video_id	***289696***
content_partner_id	398
distribution_partner_id	***602***
distro_platform_id	14
is_on_hulu	***0***
onhulu_offhulu_plus_id	2
package_id	***1000***
…	
hourid	***383149***
watched	76426

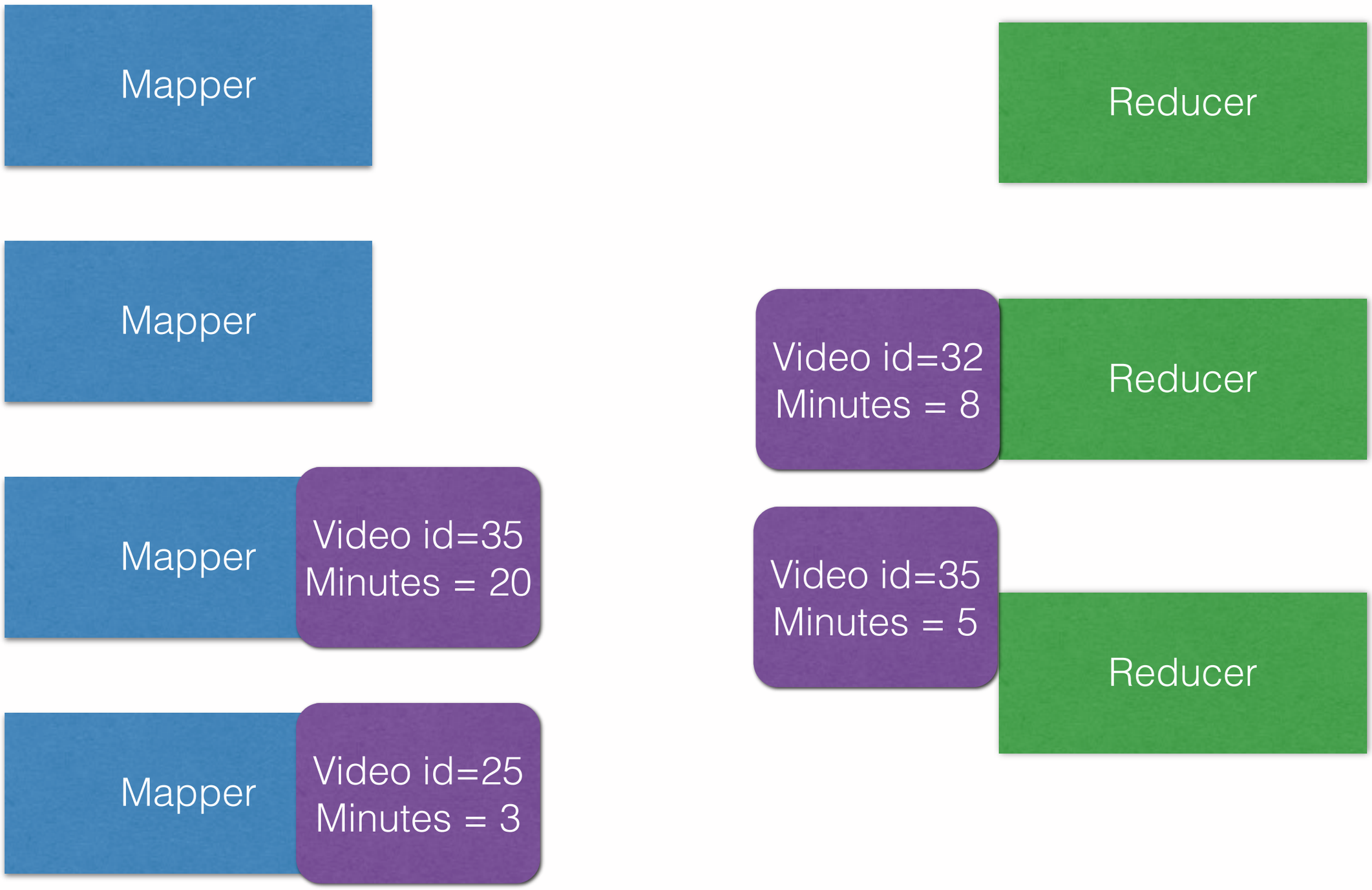
Hadoop & MapReduce



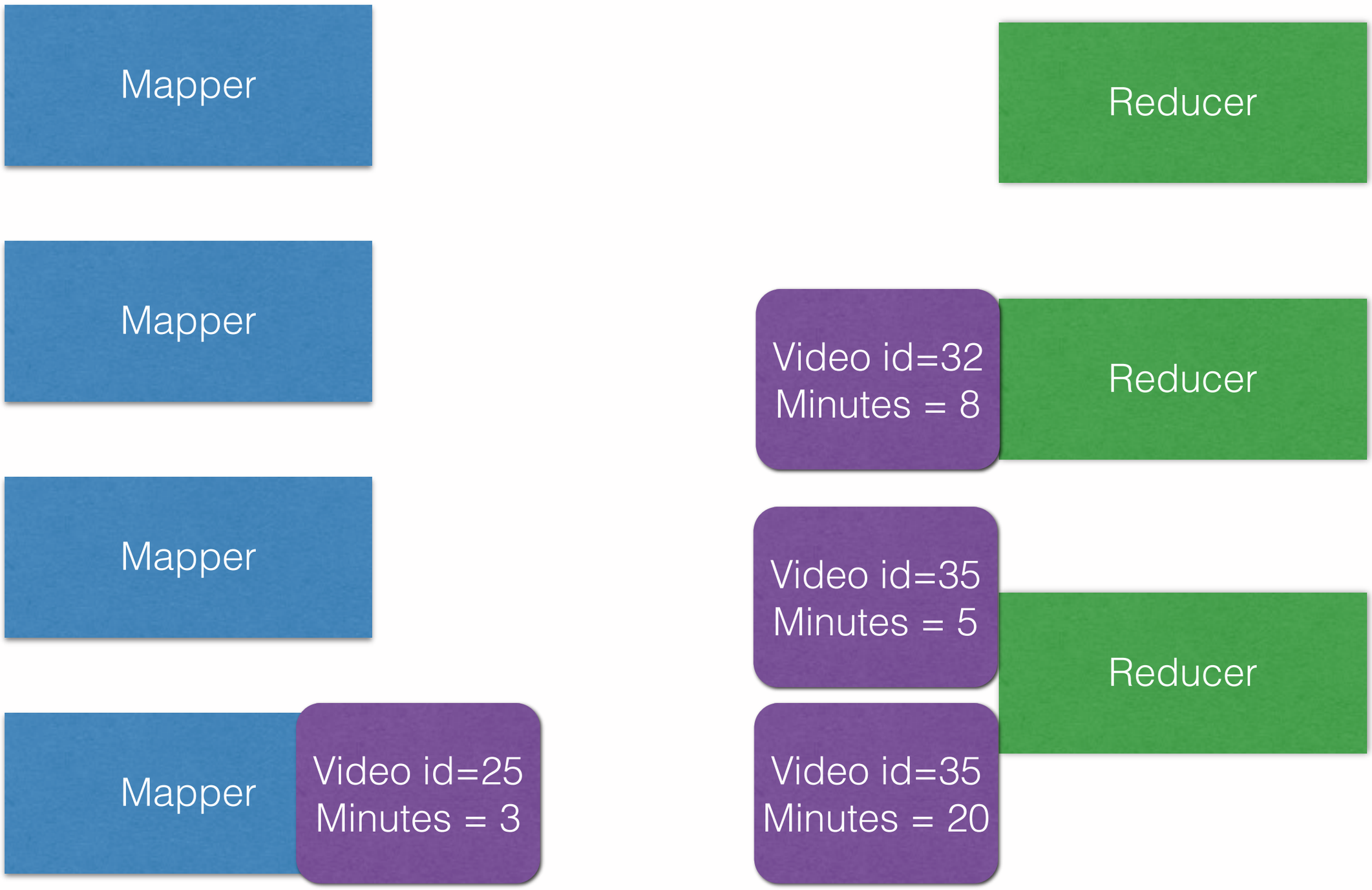
Hadoop & MapReduce



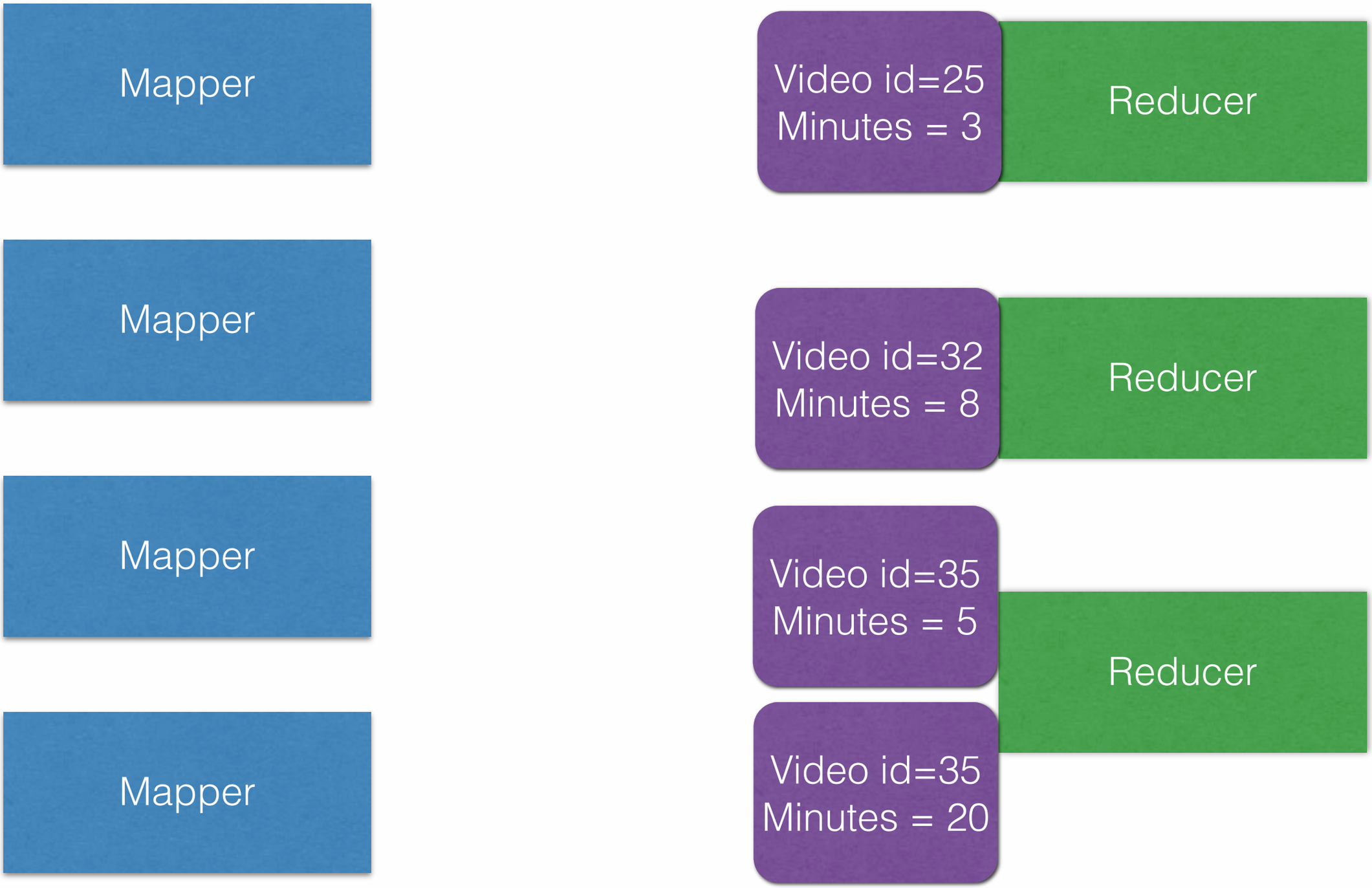
Hadoop & MapReduce



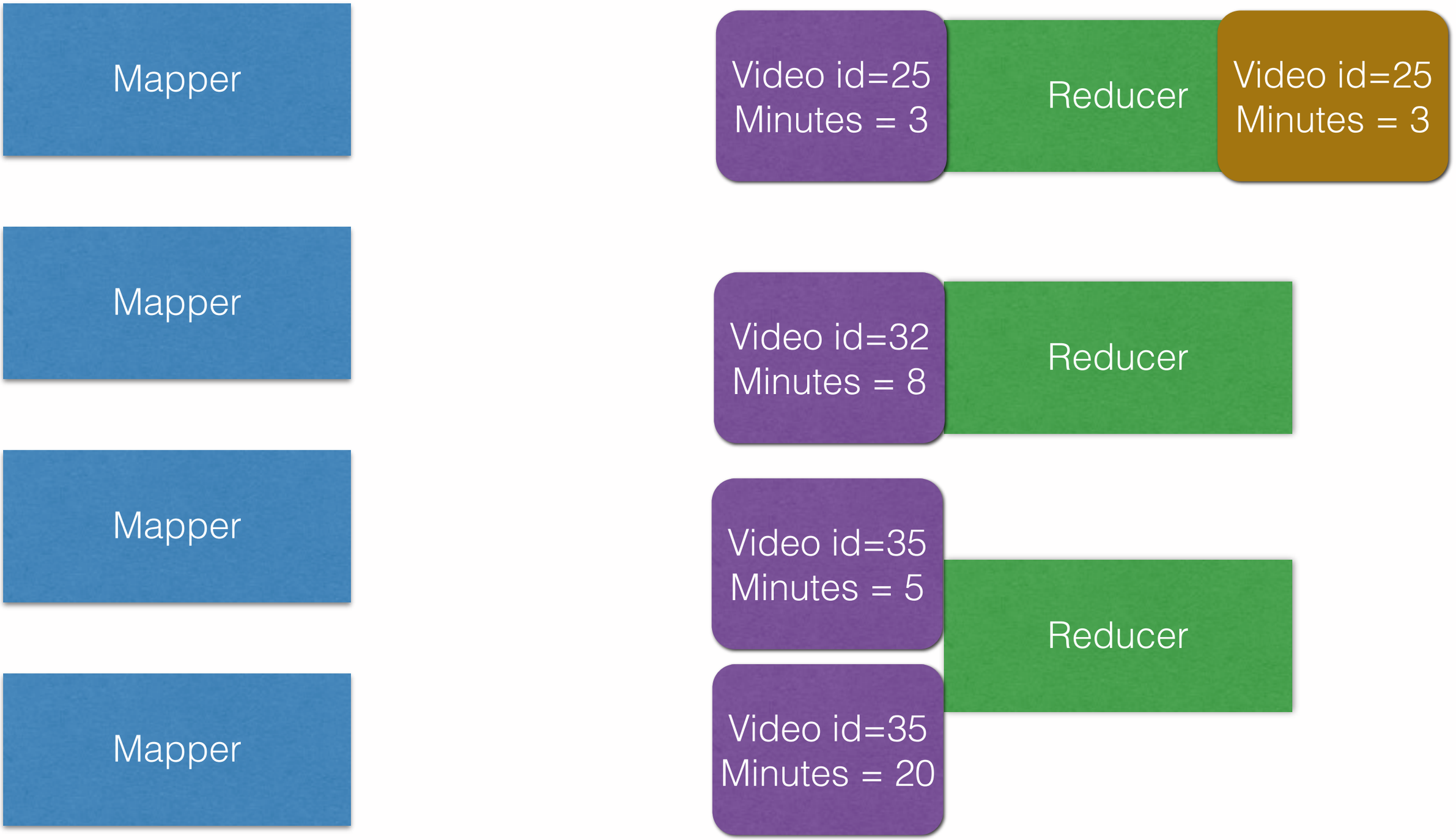
Hadoop & MapReduce



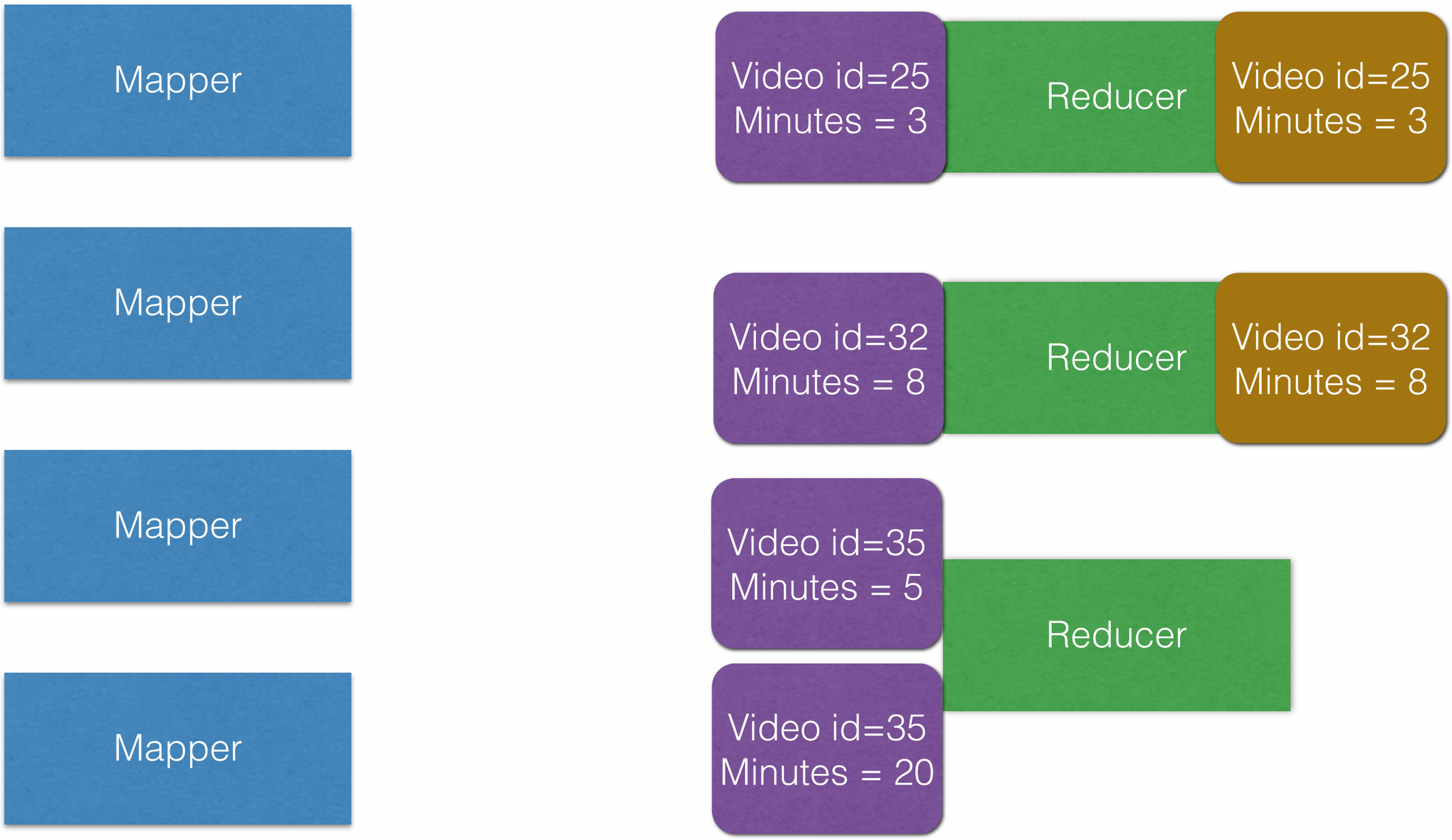
Hadoop & MapReduce



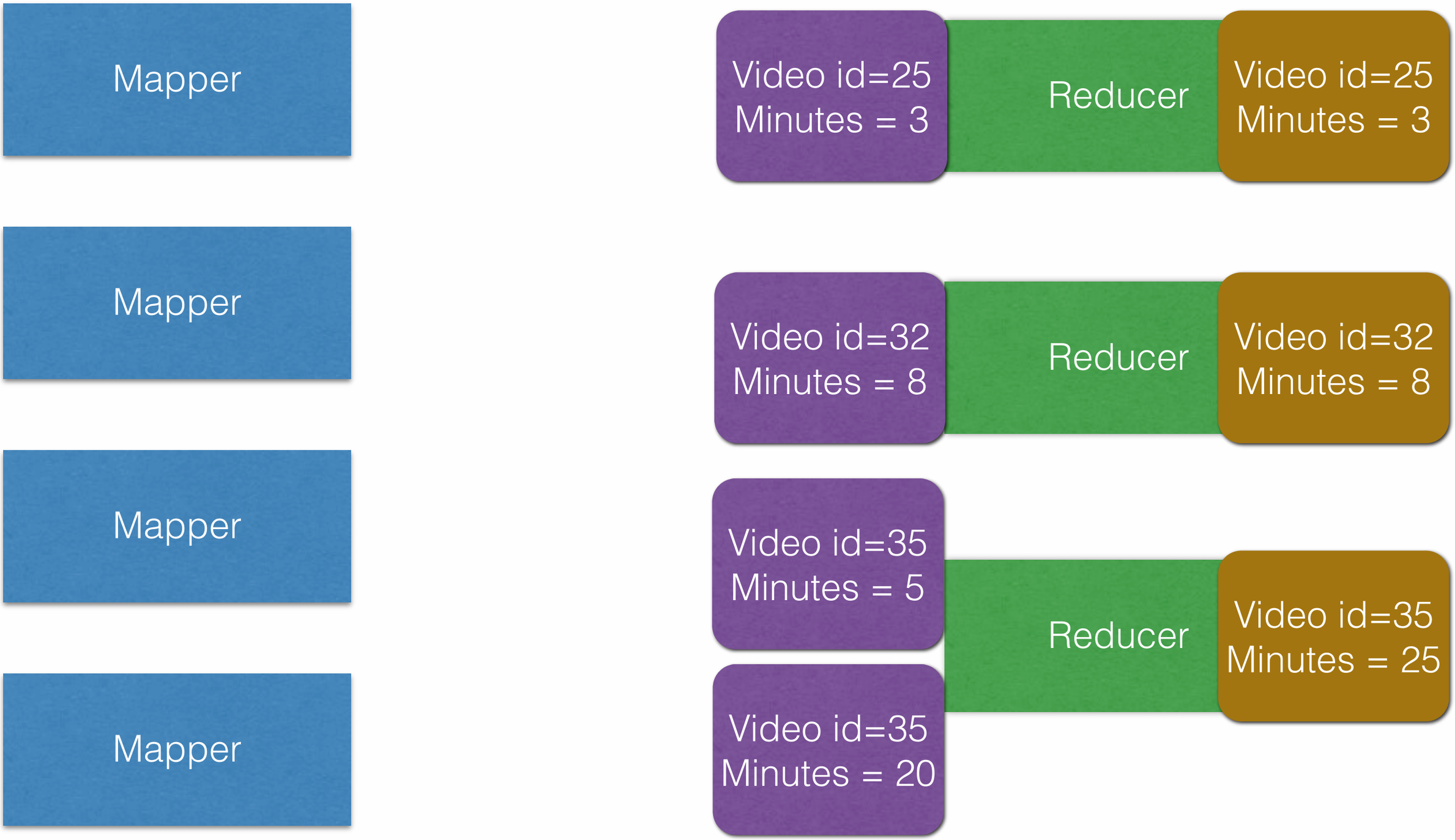
Hadoop & MapReduce



Hadoop & MapReduce

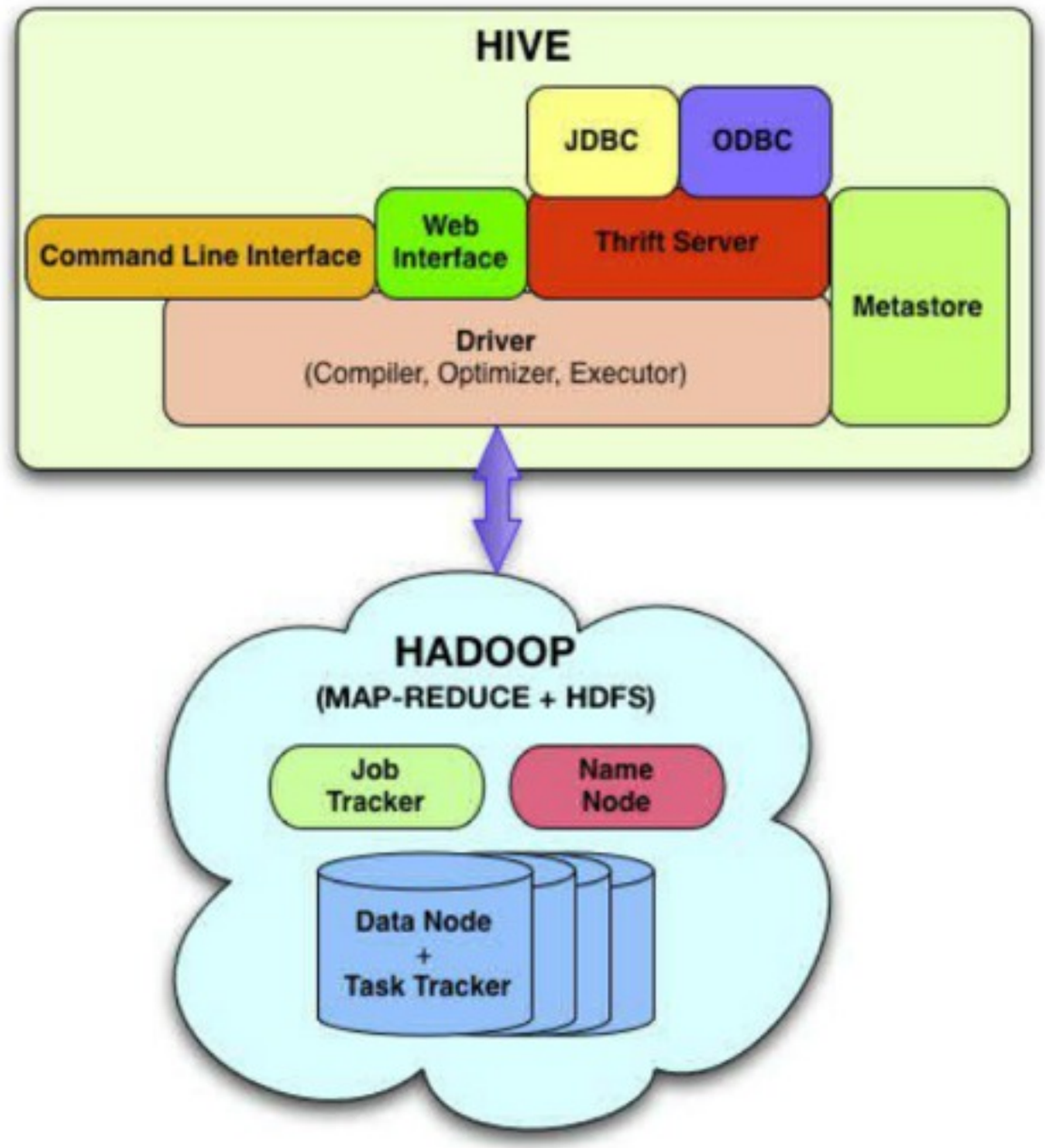


Hadoop & MapReduce



Aggregations

- Base-facts live in relational tables in HIVE
- Aggregations are now a matter of performing the right HiveQL queries once data is ready



Dispatching jobs to the cluster

- The job scheduler has a single function: To dispatch jobs to the cluster when they are ready, according to some notion of priority
- Written in Scala
- Lots of database interaction

Why Scala?

- Good concurrency support
- More importantly, easy to construct new ideas and think in terms of higher level abstractions
- “Programs are for people to read, and only incidentally for computers to execute” - Harold Abelson

```
object Job extends Table[(Int, String, Option[Int], Option[Int], Int, Int)]("job") {
  def part1 = allowPickup ~ maxDate ~ regionId ~ ignoreTrailingData ~ jobId ~ jobType ~ fileBatch ~ ignoreLeadingData ~ minDateRan
  def part2 = harpyOutputBasePath ~ enabled ~ dependsOn <>(JobData2, JobData2.unapply _)
  def all = (part1, part2)
  def * = jobId ~ name ~ dependsOn ~ jobType ~ runPriority ~ isKPI
  def allowPickup = column[Int]("allowpickup")
  def maxDate = column[Option[Date]]("maxdate")
  def regionId = column[Int]("regionid")
  def ignoreTrailingData = column[Int]("ignoretrailingdata")
  def jobId = column[Int]("jobid")
  def jobType = column[Option[Int]]("jobtype")
  def fileBatch = column[Int]("filebatch")
  def ignoreLeadingData = column[Int]("ignoreleadingdata")
  def minDateRangeStart = column[Long]("mindaterangestart")
  def inputFileType = column[String]("inputfiletype")
  def customJar = column[Option[String]]("custom_jar")
  def priority = column[Option[Int]]("priority")
  def runPriority = column[Int]("runpriority")
  def isKPI = column[Int]("isKPI")
  def hoursBeforeDay = column[Option[Int]]("hoursbeforeday")
  def errorThreshold = column[Option[Int]]("error_threshold")
  def hoursAfterDay = column[Option[Int]]("hoursafterday")
  def minDate = column[Option[Date]]("mindate")
  def hardStopAction = column[Int]("hardstop_action")
  def name = column[String]("name")
  def harpyOutputBasePath = column[Option[String]]("harpy_output_basepath")
  def enabled = column[Int]("enabled")
  def dependsOn = column[Option[Int]]("dependson")
}
case class JobData1(allowPickup: Int, maxDate: Option[Date], regionId: Int, ignoreTrailingData: Int, jobId: Int, jobType: Option[I
case class JobData2(harpyOutputBasePath: Option[String], enabled: Int, dependsOn: Option[Int])
case class JobData(data1: JobData1, data2: JobData2) {
  def allowPickup = data1.allowPickup
  def maxDate = data1.maxDate
  def regionId = data1.regionId
  def ignoreTrailingData = data1.ignoreTrailingData
  def jobId = data1.jobId
  def jobType = data1.jobType
  def fileBatch = data1.fileBatch
  def ignoreLeadingData = data1.ignoreLeadingData
  def minDateRangeStart = data1.minDateRangeStart
  def inputFileType = data1.inputFileType
```

Introducing slickint

```
package org.zenbowman.slickdemo
import scala.slick.driver.MySQLDriver.simple._
import java.sql.Date
table=Job, primaryKey=jobId, dbname=job, *=jobId ~ name ~ dependsOn ~ jobType ~ runPriority ~ isKPI
jobId               : Int           : jobid
name                : String        : name
enabled             : Int           : enabled
minDateRangeStart   : Long          : mindaterangestart
fileBatch           : Int           : filebatch
dependsOn           : Int?          : dependson
allowPickup         : Int           : allowpickup
jobType             : Int?          : jobtype
minDate             : Date?         : mindate
maxDate             : Date?         : maxdate
hoursBeforeDay      : Int?          : hoursbeforeday
hoursAfterDay       : Int?          : hoursafterday
hardStopAction      : Int           : hardstop_action
inputFileType       : String        : inputfiletype
ignoreTrailingData  : Int           : ignoretrailingdata
ignoreLeadingData   : Int           : ignoreleadingdata
priority            : Int?          : priority
errorThreshold      : Int?          : error_threshold
runPriority         : Int           : runpriority
regionId            : Int           : regionid
harpyOutputBasePath : String?       : harpy_output_basepath
customJar           : String?       : custom_jar
isKPI               : Int           : isKPI
```

Generating the Slick code

- To generate the Scala code:
  - python slickint.py [input-filename].slickint > [output-filename].scala
- Outstanding issues (wouldn't mind some help)
  - Insertion into tables with > 22 columns
  - Automatic schema detection from the database (at least for MySQL/Postgres)
- github.com/zenbowman/slickint

Handling external dependencies

- Jobs can be dependent on whether an external process has run
- We want a generic method for determining whether a job has run based on the status of one or more databases

Gate-keeper: simple case

```
result = "SELECT 1 FROM hbaseingest.ingeststatus
          WHERE lastfullupdate > '$currentHour'
          AND hbasetable='dma'"
     with "jdbc:mysql://****:3306/hbaseingest?
zeroDateTimeBehavior=convertToNull"
     user "****" password "****";
```

Gate-keeper: complex case

```
doomsdayrun = "SELECT min(ended_at_utc) FROM doomsday.run WHERE
started_at_utc >= '$nextHour'" with
    with “jdbc:mysql://***:3306/doomsday?
useGmtMillisForDatetimes=true&useJDBCCompliantTimezoneShift=tru
e&useTimezone=false”
    user “****” password "******";


result = "SELECT CASE WHEN COUNT(*) = 0 THEN 1 ELSE 0 END FROM
hbaseingest.ingeststatus WHERE lastdiffupdate <
'$doomsdayrun';"
    with “jdbc:mysql://****:3306/hbaseingest?
zeroDateTimeBehavior=convertToNull&useLegacyDatetimeCode=false&
useGmtMillisForDatetimes=true&useJDBCCompliantTimezoneShift=tru
e&useTimezone=false”
    user “****” password "****";
```

What kind of your system does your language encourage?

Parser combinators

```
object GateKeeperConcepts {

  case class GateKeeperVariable(name: String)

  trait GateKeeperEvaluator {
    def getSql: String
     def getResult(rs: ResultSet): Option[String]
  }

  case class SimpleSqlEvaluator(sqlString: String) extends GateKeeperEvaluator {
    def getSql: String = sqlString
    def getResult(rs: ResultSet) = { ... }
  }

  case class ConnectionParameters(connectionString: String, userName: String, password:
String)

  case class GateKeepingExpression(leftHandSide: GateKeeperVariable, rightHandSide:
GateKeeperEvaluator, connection: ConnectionParameters)

  case class GateKeepingSpec(expressions: Seq[GateKeepingExpression])

}
```

The parser

```
object GateKeeperDSL extends StandardTokenParsers {

  import GateKeeperConcepts._

  lexical.reserved +=(
    "with", "user", "password"
    )

  lazy val gateKeepingSpec: Parser[GateKeepingSpec] =
    rep(gateKeepingExpression) ^^ GateKeepingSpec

  lazy val gateKeepingExpression: Parser[GateKeepingExpression] =
    ident ~ "=" ~ gateKeeperEvaluator ~ "with" ~ stringLit ~ "user" ~ stringLit ~ "password" ~
stringLit ~ ";" ^^ {
      case name ~ _ ~ rhs ~ _ ~ conn ~ _ ~ username ~ _ ~ pwd ~ _ =>
GateKeepingExpression(GateKeeperVariable(name), rhs, ConnectionParameters(conn, username,
pwd))
    }

  lazy val gateKeeperEvaluator: Parser[GateKeeperEvaluator] =
    simpleSqlEvaluator

  lazy val simpleSqlEvaluator: Parser[SimpleSqlEvaluator] =
    stringLit ^^ {
      case sqlString => SimpleSqlEvaluator(sqlString)
    }
}
```

If you find yourself writing too much code to do a mundane job, or if you have trouble expressing the process comfortably, maybe you are using the wrong language

- Rob Pike and Brian Kernighan, The Practice of Programming

Syntactic sugar causes cancer of the semicolon

- Alan Perlis