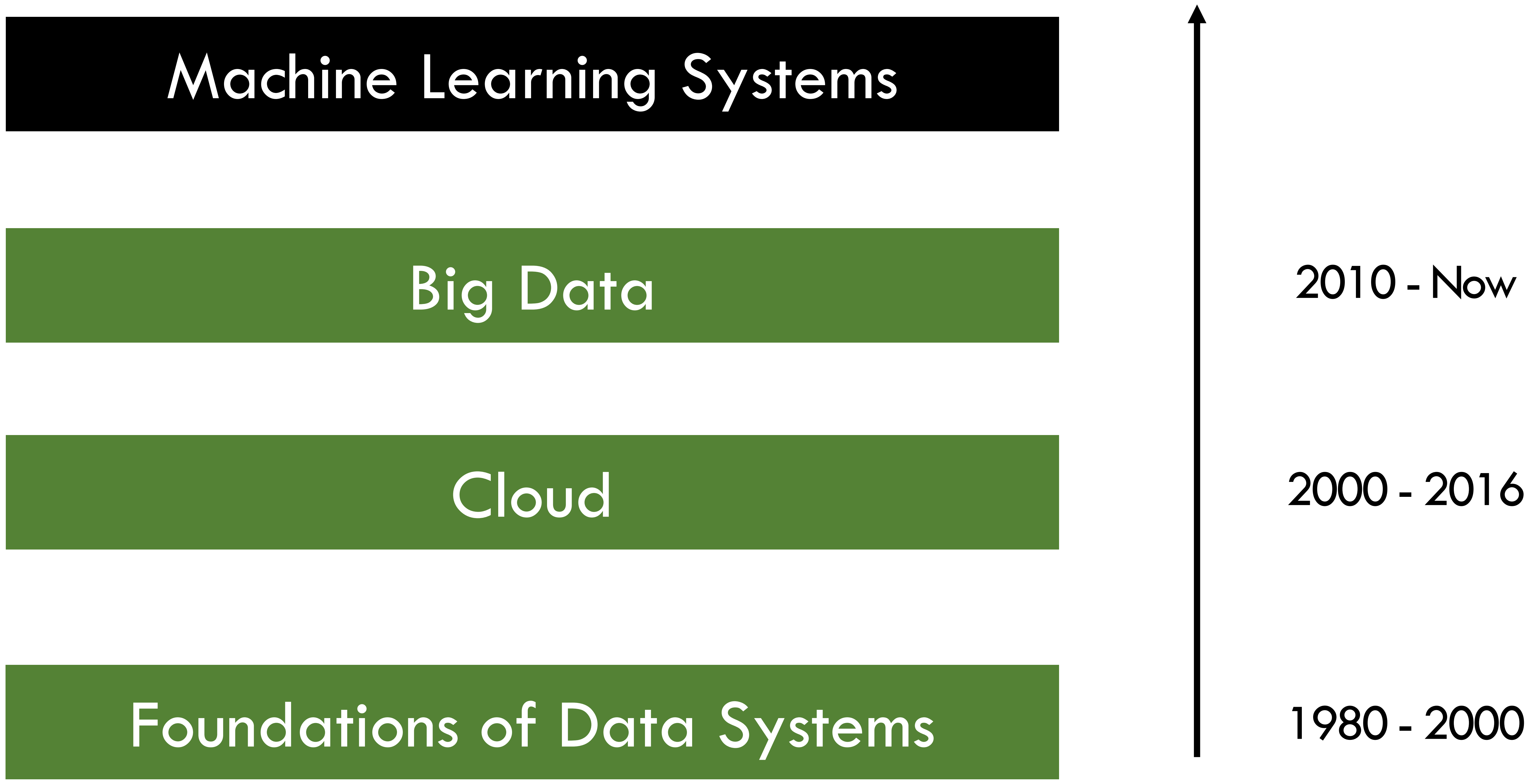
Where We Are


Machine Learning Systems
Big Data
Cloud
Foundations of Data Systems
2010 - Now
2000 - 2016
1980 - 2000

Today's topic: Stream Processing

- Computation vs. I/O: Arithmetic intensity
  - Loop fusion
- When MapReduce fails
- **Spark and RDD**
- Spark Ecosystem and Beyond
- Early ML systems: parameter server

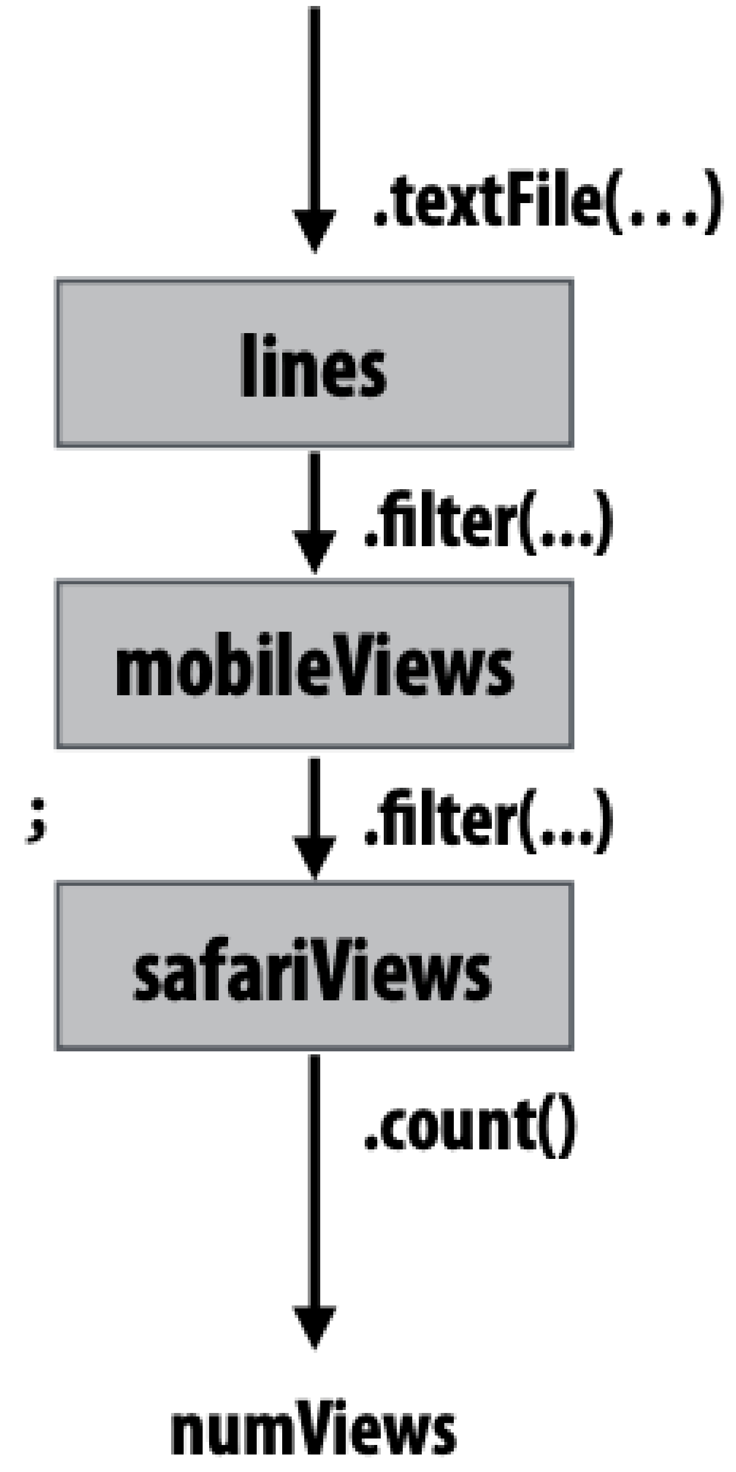
RDD: Spark's key programming abstraction:

- Read-only collection of records (immutable)
- RDDs can only be created by deterministic transformations on data in persistent storage or on existing RDDs

RDDs

```
// create RDD from file system data
var lines = spark.textFile("hdfs://15418log.txt");

// create RDD using filter() transformation on lines
var mobileViews = lines.filter((x: String) => isMobileClient(x));

// another filter() transformation
var safariViews = mobileViews.filter((x: String) => x.contains("Safari"));

// then count number of elements in RDD via count() action
var numViews = safariViews.count();
```

int



Predefined Set of Operators

Transformation

Action

RDD transformations and actions

Transformations: (data parallel operators taking an input RDD to a new RDD)

$$
\begin{array}{rcl}
map(f : \mathrm{T} \Rightarrow \mathrm{U}) & : & \mathrm{RDD[T]} \Rightarrow \mathrm{RDD[U]} \\
filter(f : \mathrm{T} \Rightarrow \mathrm{Bool}) & : & \mathrm{RDD[T]} \Rightarrow \mathrm{RDD[T]} \\
flatMap(f : \mathrm{T} \Rightarrow \mathrm{Seq[U]}) & : & \mathrm{RDD[T]} \Rightarrow \mathrm{RDD[U]} \\
sample(fraction : \mathrm{Float}) & : & \mathrm{RDD[T]} \Rightarrow \mathrm{RDD[T]} \ \text{(Deterministic sampling)} \\
groupByKey() & : & \mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, Seq[V])]} \\
reduceByKey(f : (\mathrm{V}, \mathrm{V}) \Rightarrow \mathrm{V}) & : & \mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]} \\
union() & : & (\mathrm{RDD[T]}, \mathrm{RDD[T]}) \Rightarrow \mathrm{RDD[T]} \\
join() & : & (\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \Rightarrow \mathrm{RDD[(K, (V, W))]} \\
cogroup() & : & (\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \Rightarrow \mathrm{RDD[(K, (Seq[V], Seq[W]))]} \\
crossProduct() & : & (\mathrm{RDD[T]}, \mathrm{RDD[U]}) \Rightarrow \mathrm{RDD[(T, U)]} \\
mapValues(f : \mathrm{V} \Rightarrow \mathrm{W}) & : & \mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, W)]} \ \text{(Preserves partitioning)} \\
sort(c : \mathrm{Comparator[K]}) & : & \mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]} \\
partitionBy(p : \mathrm{Partitioner[K]}) & : & \mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}
\end{array}
$$

Actions: (provide data back to the "host" application)

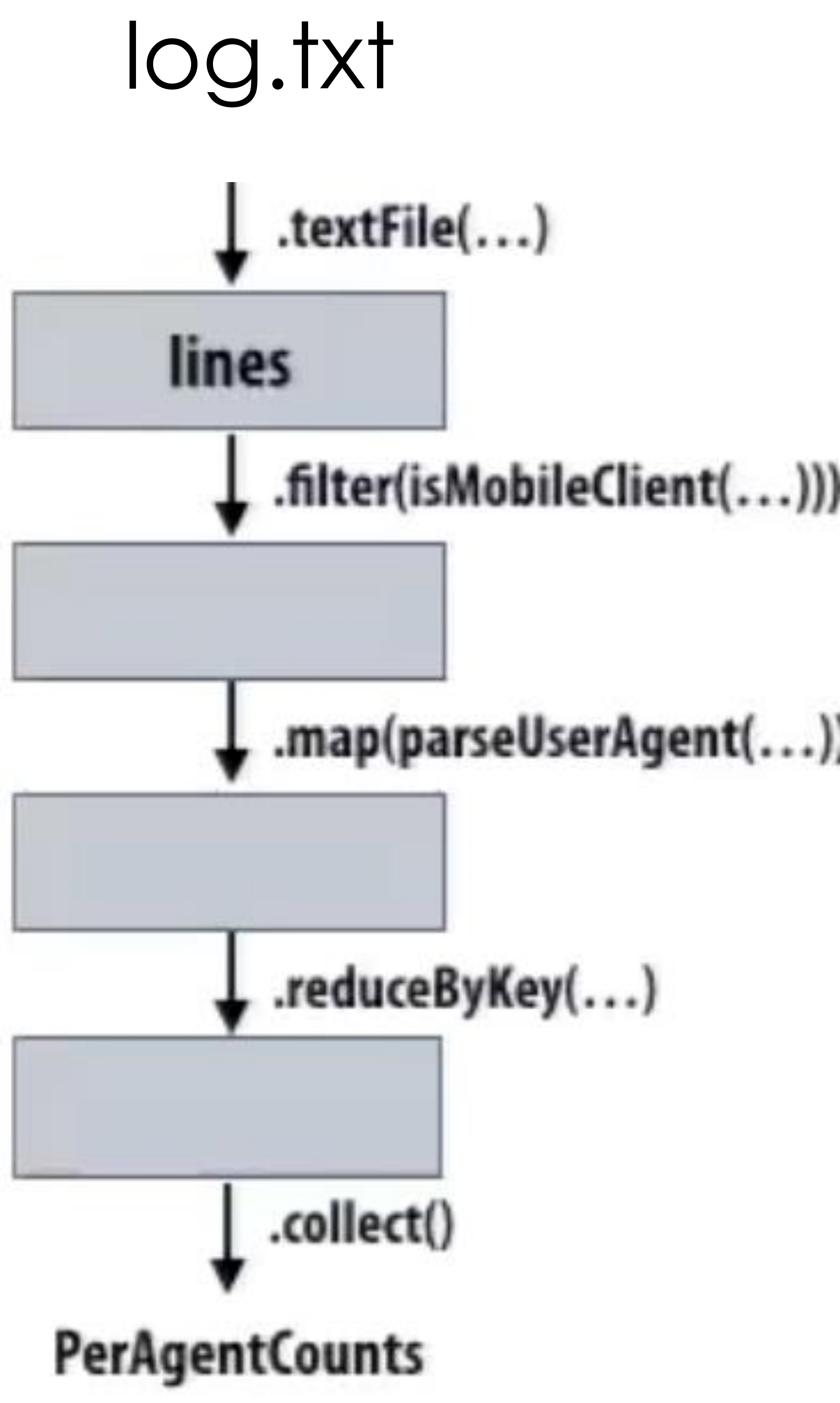
$$
\begin{array}{rcl}
count() & : & \mathrm{RDD[T]} \Rightarrow \mathrm{Long} \\
collect() & : & \mathrm{RDD[T]} \Rightarrow \mathrm{Seq[T]} \\
reduce(f : (\mathrm{T}, \mathrm{T}) \Rightarrow \mathrm{T}) & : & \mathrm{RDD[T]} \Rightarrow \mathrm{T} \\
lookup(k : \mathrm{K}) & : & \mathrm{RDD[(K, V)]} \Rightarrow \mathrm{Seq[V]} \ \text{(On hash/range partitioned RDDs)} \\
save(path : \mathrm{String}) & : & \text{Outputs RDD to a storage system, } e.g., \text{ HDFS}
\end{array}
$$

Repeating the map-reduce example

```
// 1. create RDD from file system data
// 2. create RDD with only lines from mobile clients
// 3. create RDD with elements of type (String,Int) from line string
// 4. group elements by key
// 5. call provided reduction function on all keys to count views
var perAgentCounts = spark.textFile("hdfs://log.txt")
                          .filter(x => isMobileClient(x))
                          .map(x => (parseUserAgent(x),1));
                          .reduceByKey((x,y) => x+y)
                          .collect();
```

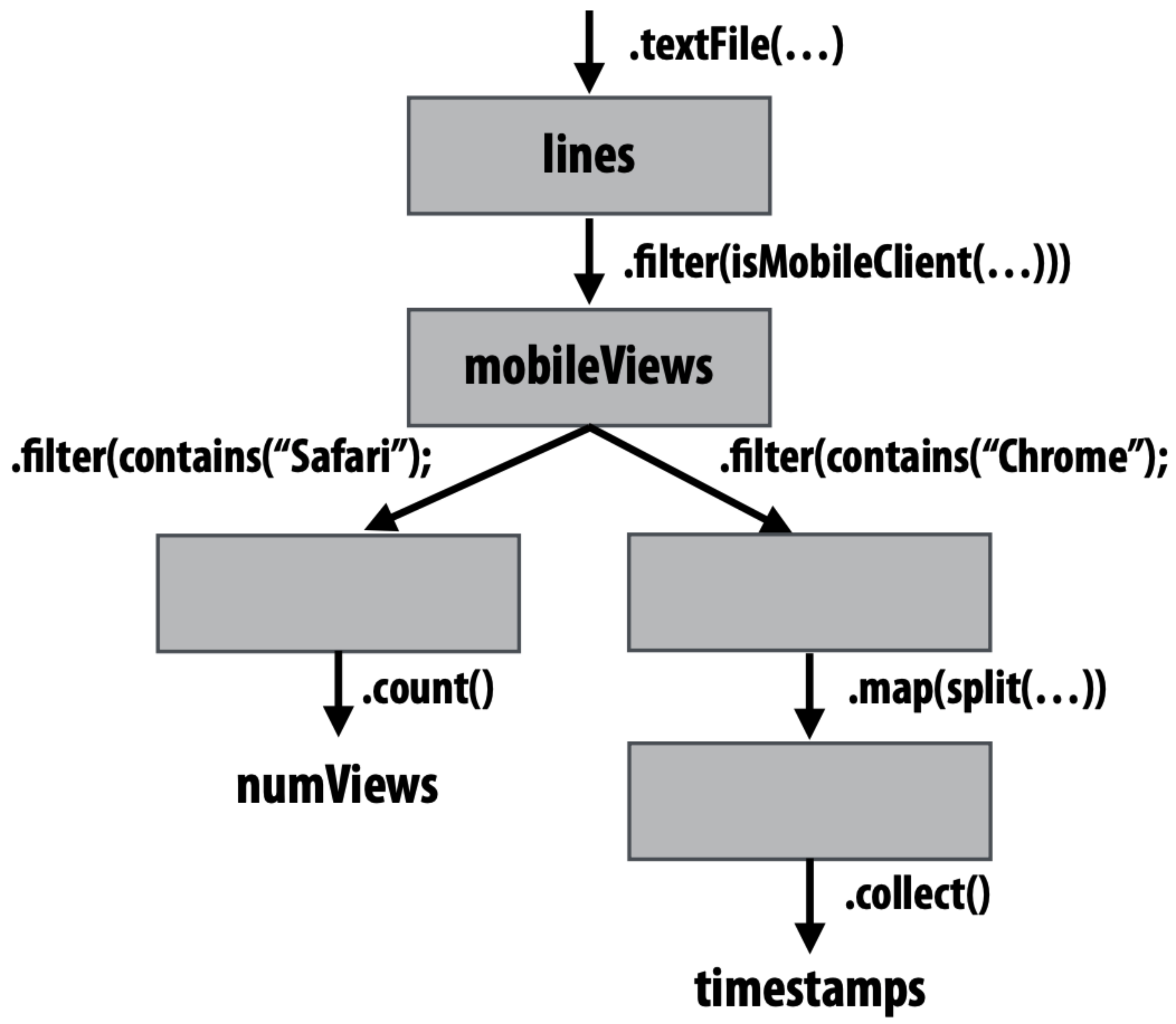
Array [String,int]

"Lineage": Sequence of RDD operations needed to compute output



Another Spark program

```
// create RDD from file system data
var lines = spark.textFile("hdfs://log.txt");

// create RDD using filter() transformation on lines
var mobileViews = lines.filter((x: String) => isMobileClient(x));

// instruct Spark runtime to try to keep mobileViews in memory
mobileViews.persist();

// create a new RDD by filtering mobileViews
// then count number of elements in new RDD via count() action
var numViews = mobileViews.filter(_.contains("Safari")).count();

// 1. create new RDD by filtering only Chrome views
// 2. for each element, split string and take timestamp of // page view
// 3. convert RDD to a scalar sequence (collect() action)
var timestamps = mobileViews.filter(_.contains("Chrome"))
                            .map(_.split(" ")(0))
                            .collect();
```



Discussion

- How do you like this programming model?
- v.s. map reduce
  - Flexibility and Expressiveness?
  - Simplicity?
  - Scalability?
  - Fault tolerance?

How do we implement RDDs?

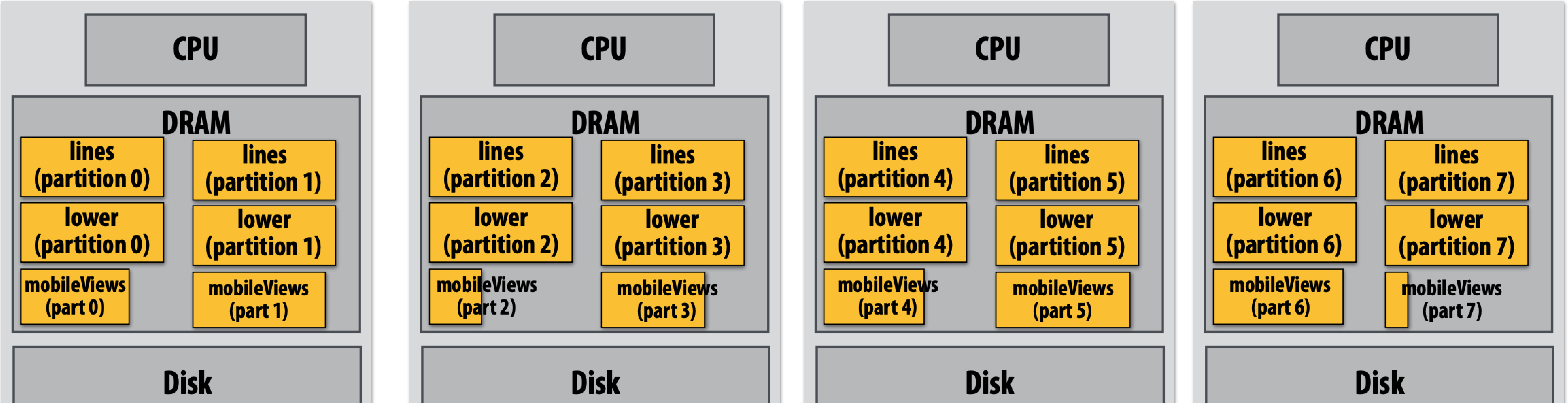
- In particular, how should they be stored?
  - var lines = spark.textFile(“hdfs://log.txt”);
  - var lower = lines.map(_.toLower());
  - var mobileViews = lower.filter(x => isMobileClient(x));
  - var howMany = mobileViews.count();

Question: should we think of RDD’s like arrays?

How do we implement RDDs?

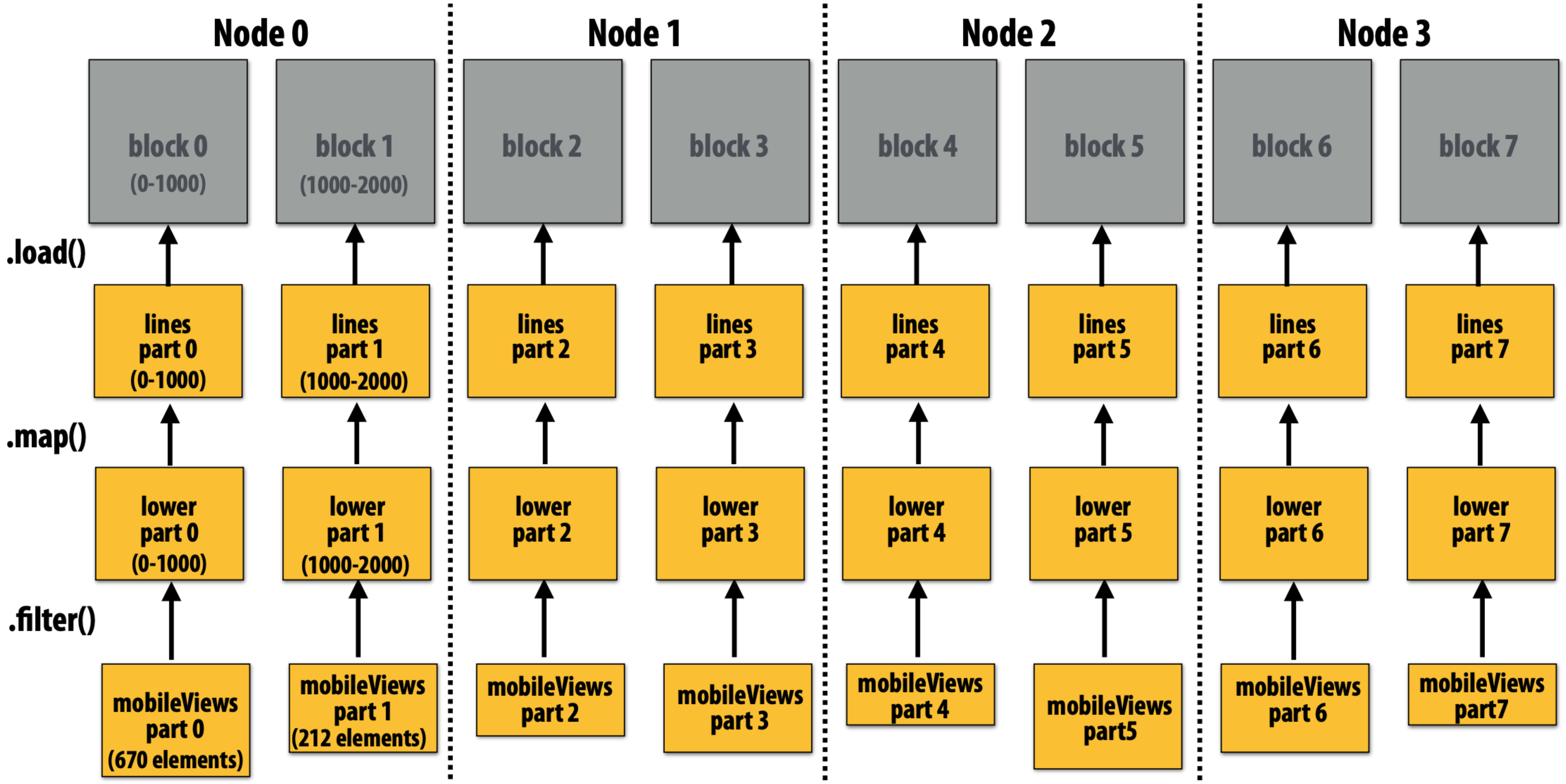
- In particular, how should they be stored?
  - var lines = spark.textFile(“hdfs://log.txt”);
  - var lower = lines.map(_.toLower());
  - var mobileViews = lower.filter(x => isMobileClient(x));
  - var howMany = mobileViews.count();

Question: Array -> In-memory representation would be huge! (larger than original flie on disk)



RDD partitioning and dependencies

```
var lines = spark.textFile(“hdfs://log.txt”);
var lower = lines.map(_.toLower());
var mobileViews = lower.filter(x => isMobileClient(x));
var howMany = mobileViews.count();
```



Black lines show dependencies between RDD partitions.

Implementing sequence of RDD ops efficiently

```
var lines = spark.textFile("hdfs://log.txt");
var lower = lines.map(_.toLower());
var mobileViews = lower.filter(x => isMobileClient(x));
var howMany = mobileViews.count();
```

- Recall "loop fusion" from start of lecture
- The following code stores only a line of the log file in memory, and only reads input data from disk once ("streaming" solution)

```
int count = 0;
while (inputFile.eof()) {
   string line = inputFile.readLine();
   string lower = line.toLower;
   if (isMobileClient(lower))
      count++;
}
```

A simple interface for RDDs

```
var lines = spark.textFile("hdfs://log.txt");
var lower = lines.map(_.toLower());
var mobileViews = lower.filter(x => isMobileClient(x));
var howMany = mobileViews.count();
```

```
// create RDD by mapping fun onto input (parent) RDD
RDD::map(RDD parent, func) {
   return new RDDFromMap(parent, func);
}

// create RDD from text file on disk
RDD::textFile(string filename) {
   return new RDDFromTextFile(open(filename));
}

// count action (forces evaluation of RDD)
RDD::count() {
   int count = 0;
   while (hasMoreElements()) {
      var el = next();
      count++;
   }
}
```

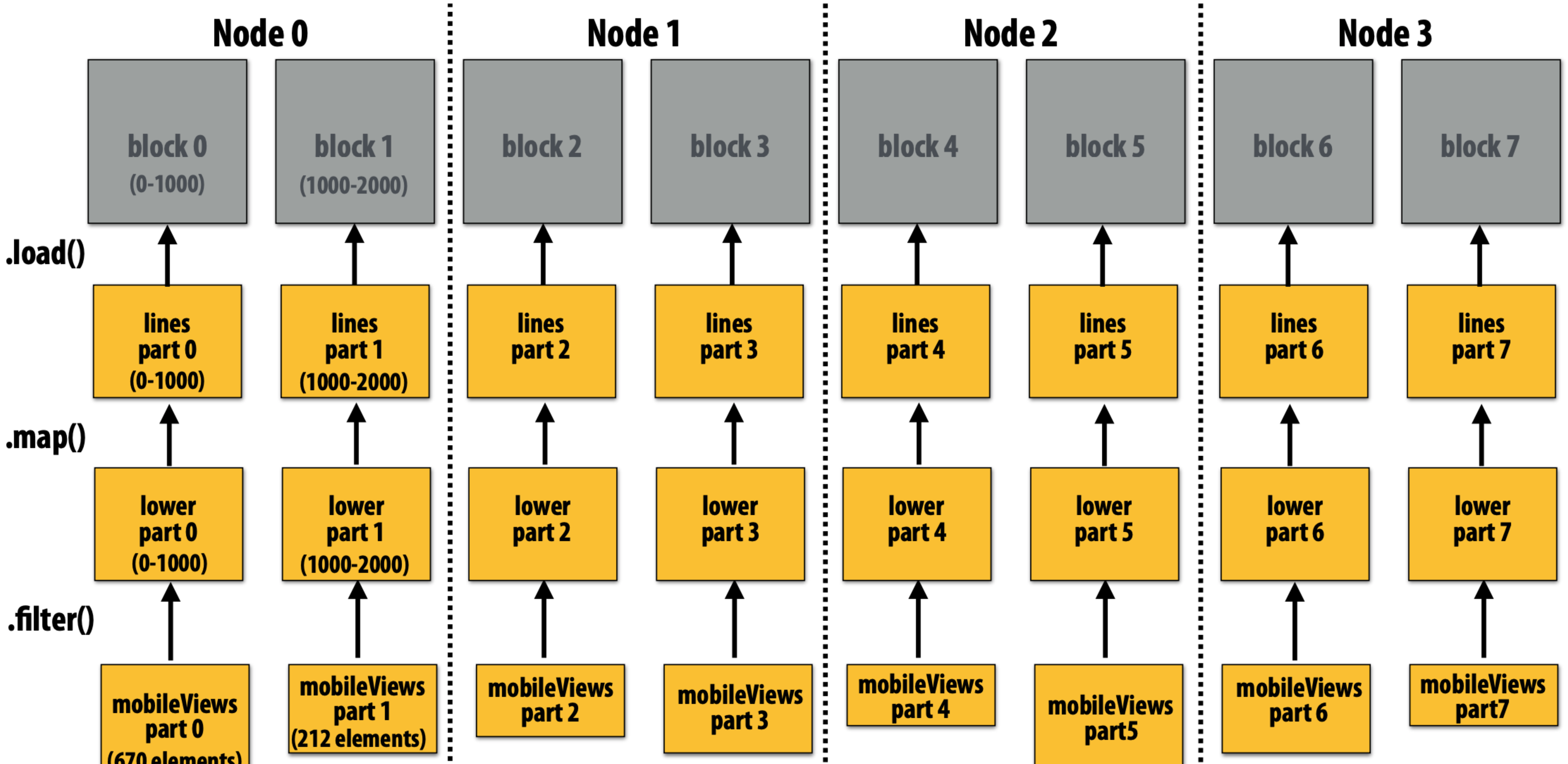
```
RDD::hasMoreElements() {
   parent.hasMoreElements();
}

// overloaded since no parent exists
RDDFromTextFile::hasMoreElements() {
   return !inputFile.eof();
}

RDDFromTextFile::next() {
   return inputFile.readLine();
}

RDDFromMap::next() {
   var el = parent.next();
   return el.toLower();
}

RDDFromFilter::next() {
  while (parent.hasMoreElements()) {
    var el = parent.next();
    if (isMobileClient(el))
      return el;
}
```

Narrow dependencies

```
var lines = spark.textFile("hdfs://log.txt");
var lower = lines.map(_.toLower());
var mobileViews = lower.filter(x => isMobileClient(x));
var howMany = mobileViews.count();
```

"Narrow dependencies" = each partition of parent RDD referenced by at most one child RDD partition

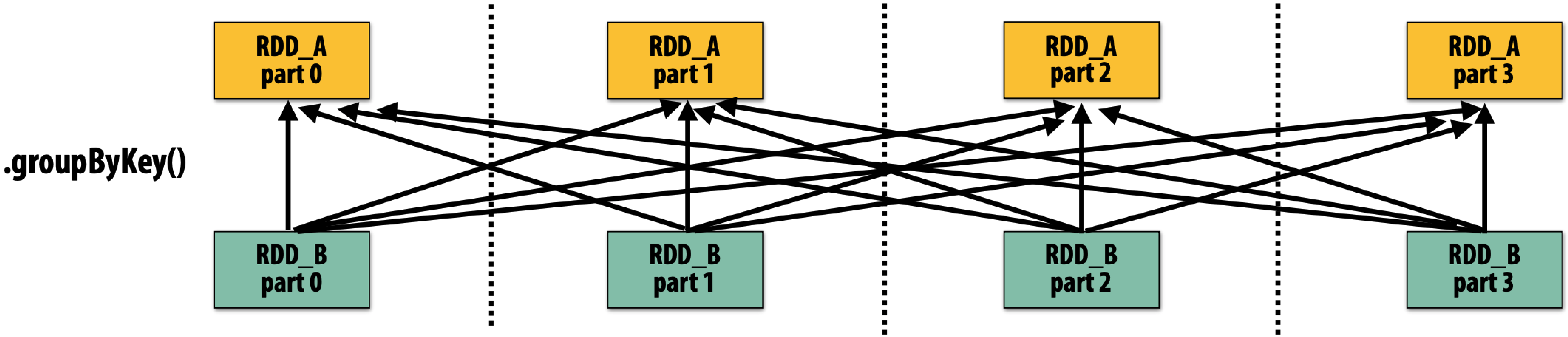
- Allows for fusing of operations
  (here: can apply map and then filter all at once on input element)
- In this example: no communication between nodes of cluster
  (communication of one int at end to perform count() reduction)



Wide dependencies

groupByKey: RDD[(K,V)] → RDD[(K,Seq[V])]

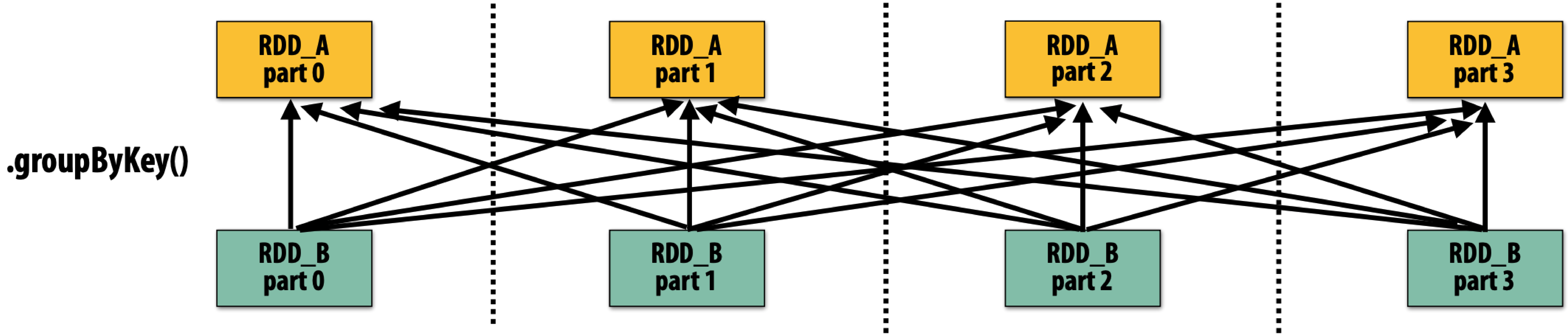
"Make a new RDD where each element is a sequence containing all values from the parent RDD with the same key."



Wide dependencies = each partition of parent RDD referenced by multiple child RDD partitions

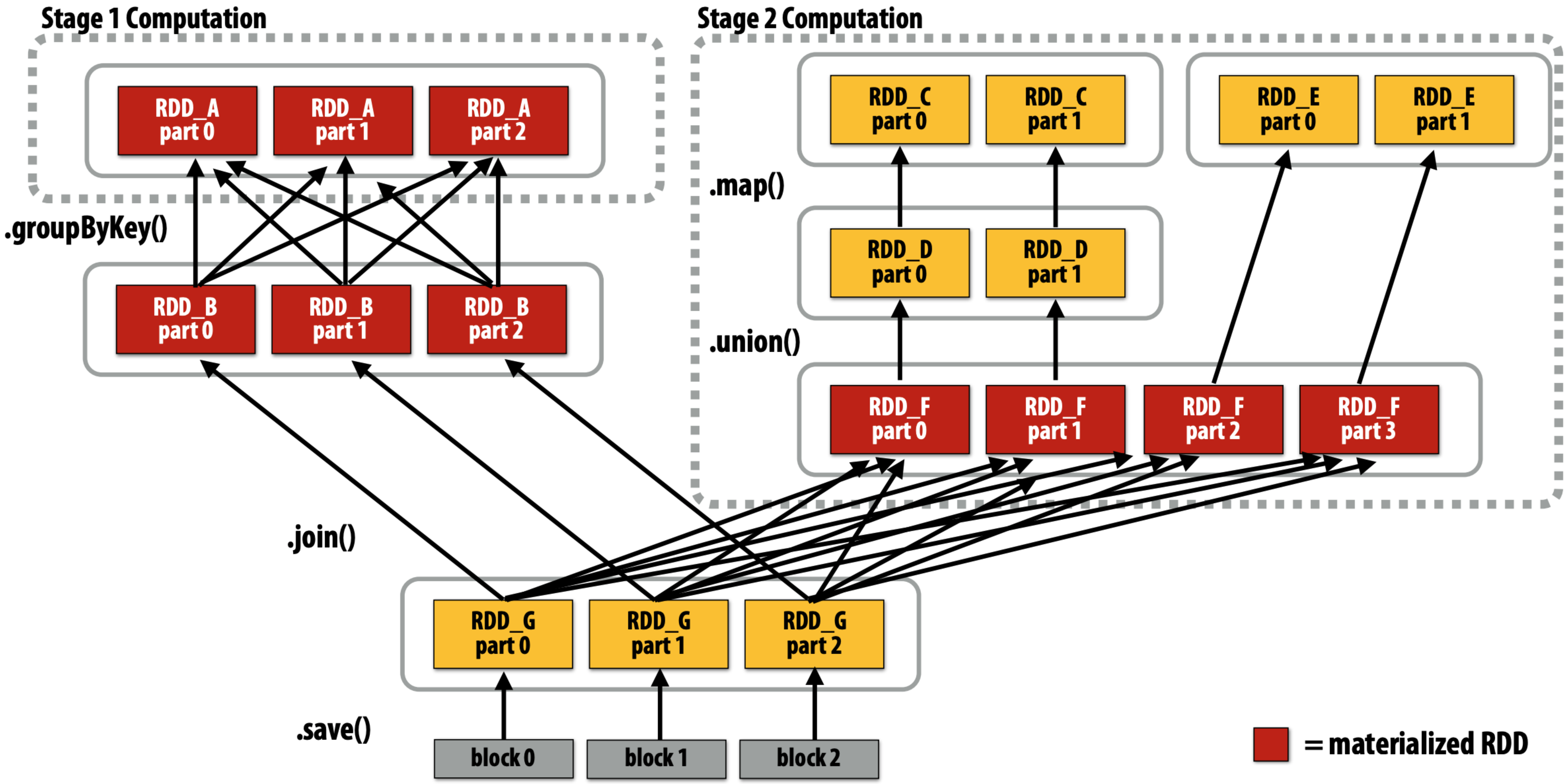
Wide dependencies

Wide dependencies = each partition of parent RDD referenced by multiple child RDD partitions



Challenges:
- Must compute all of RDD_A before computing RDD_B
  - Example: groupByKey() may induce all-to-all communication as shown above
- May trigger significant recompilation of ancestor lineage upon node failure (will address resilience in a few slides)

Scheduling Spark computations

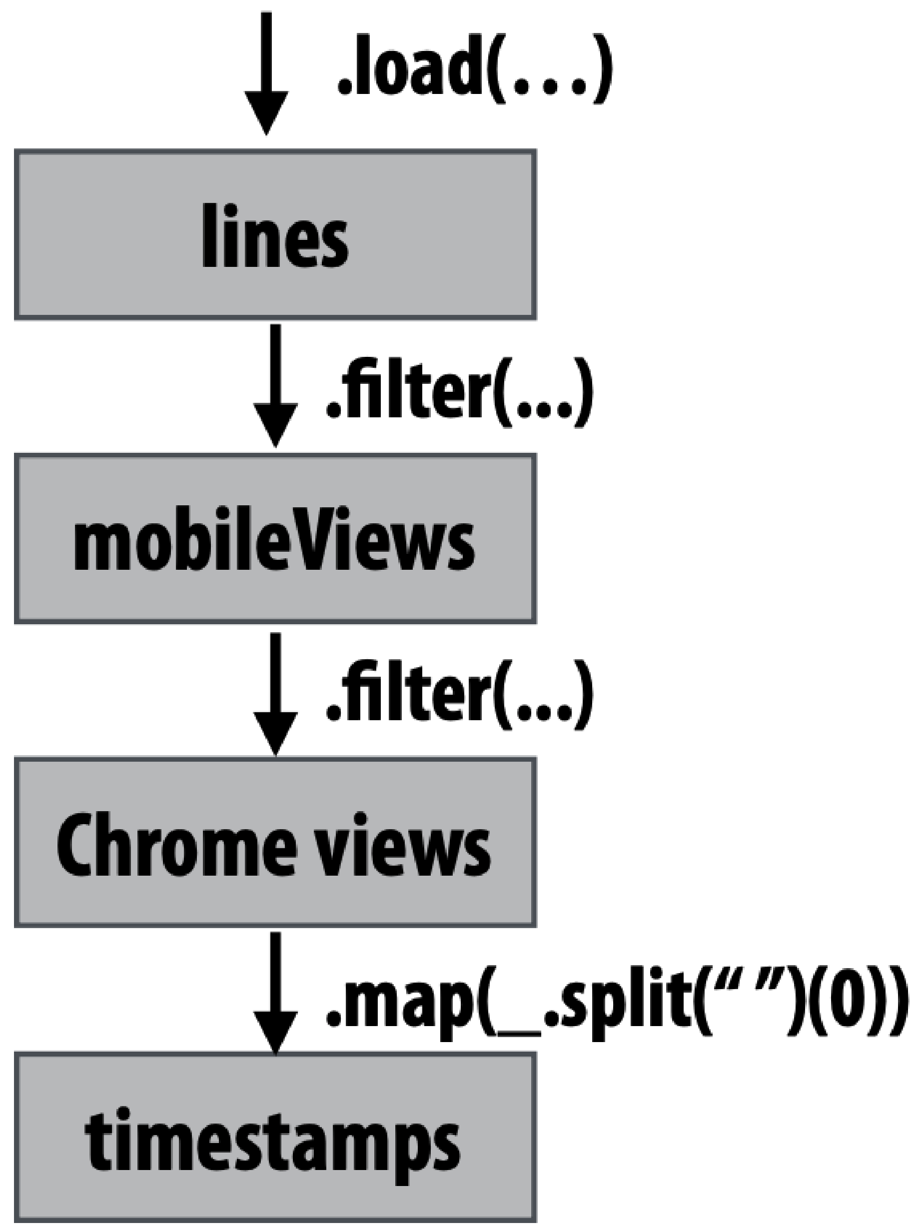


- Actions (e.g., save()) trigger evaluation of Spark lineage graph.
  - Stage 1 Computation: do nothing since input already materialized in memory
  - Stage 2 Computation: evaluate map in fused manner, only actually materialize RDD F
  - Stage 3 Computation: execute join (could stream the operation to disk, do not need to materialize)

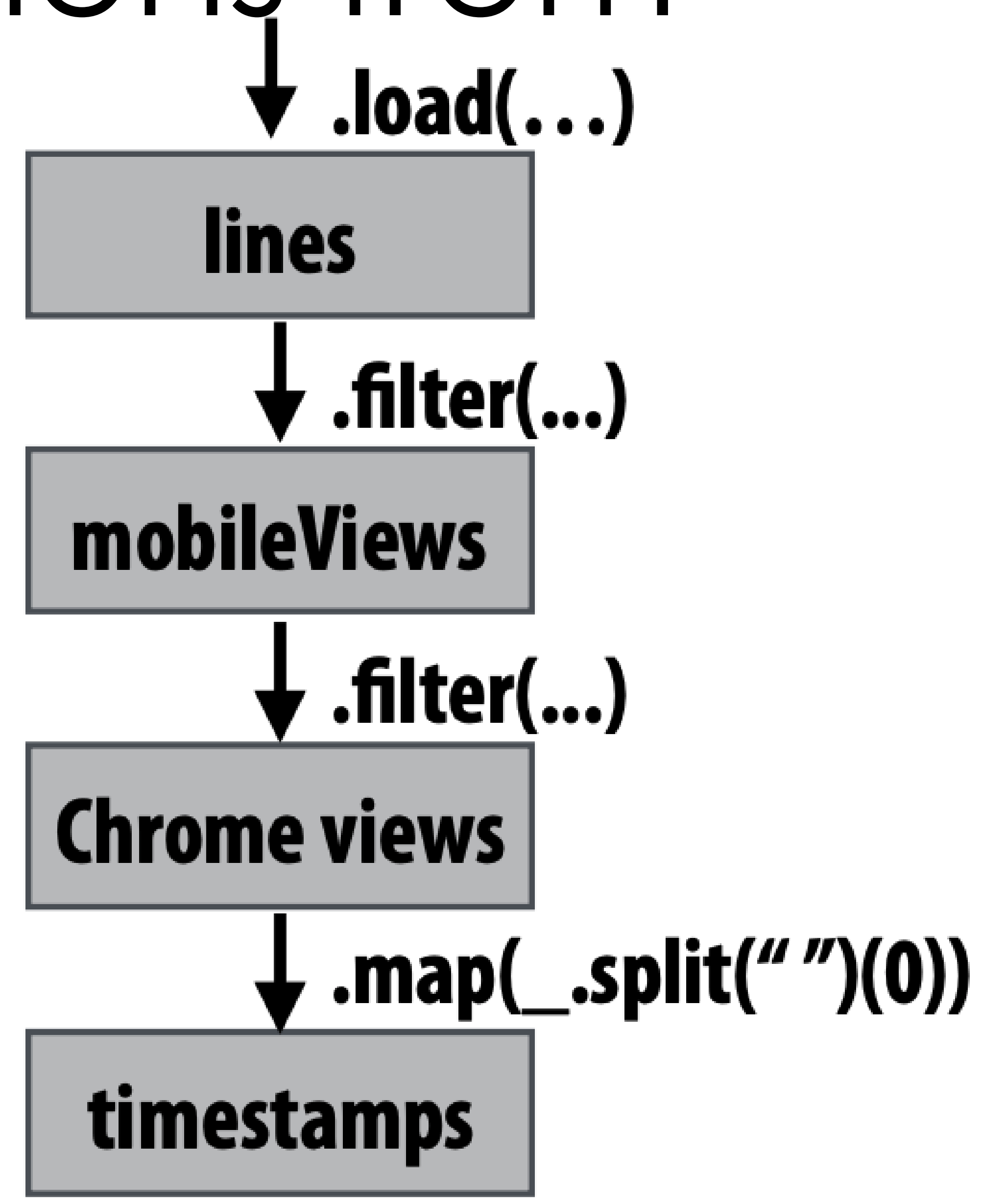
Implementing resilience via lineage

- RDD transformations are bulk, deterministic, and functional
  - Implication: runtime can always reconstruct contents of RDD from its lineage (the sequence of transformations used to create it)
  - Lineage is a log of transformations
  - Efficient: since log records bulk data-parallel operations, overhead of logging is low (compared to logging fine-grained operations, like in a database)

```
// create RDD from file system data
var lines = spark.textFile("hdfs://15418log.txt");
// create RDD using filter() transformation on lines
var mobileViews = lines.filter((x: String) => isMobileClient(x));
// 1. create new RDD by filtering only Chrome views
// 2. for each element, split string and take timestamp of // page view (first element)
// 3. convert RDD To a scalar sequence (collect() action)
var timestamps = mobileView.filter(_.contains("Chrome")) .map(_.split(" ")(0));
```

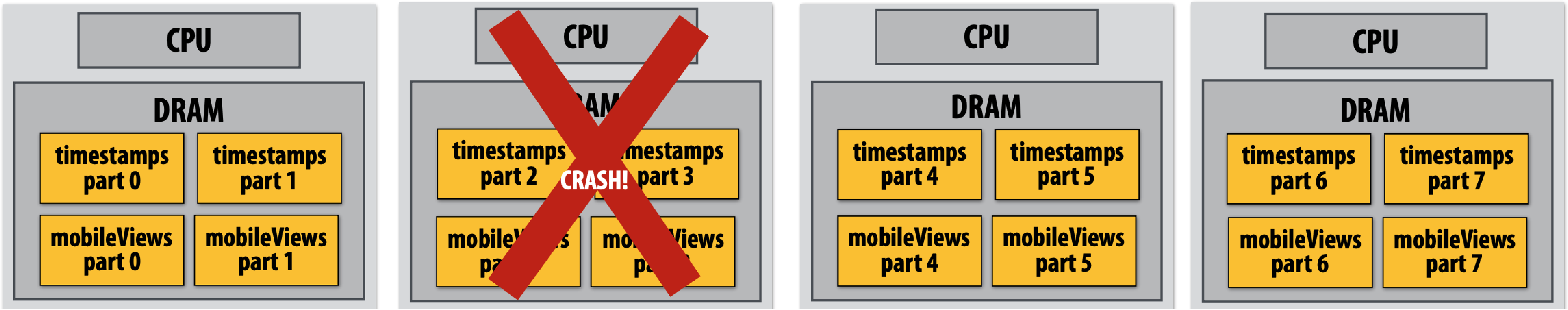


Upon node failure: recompute lost RDD partitions from lineage

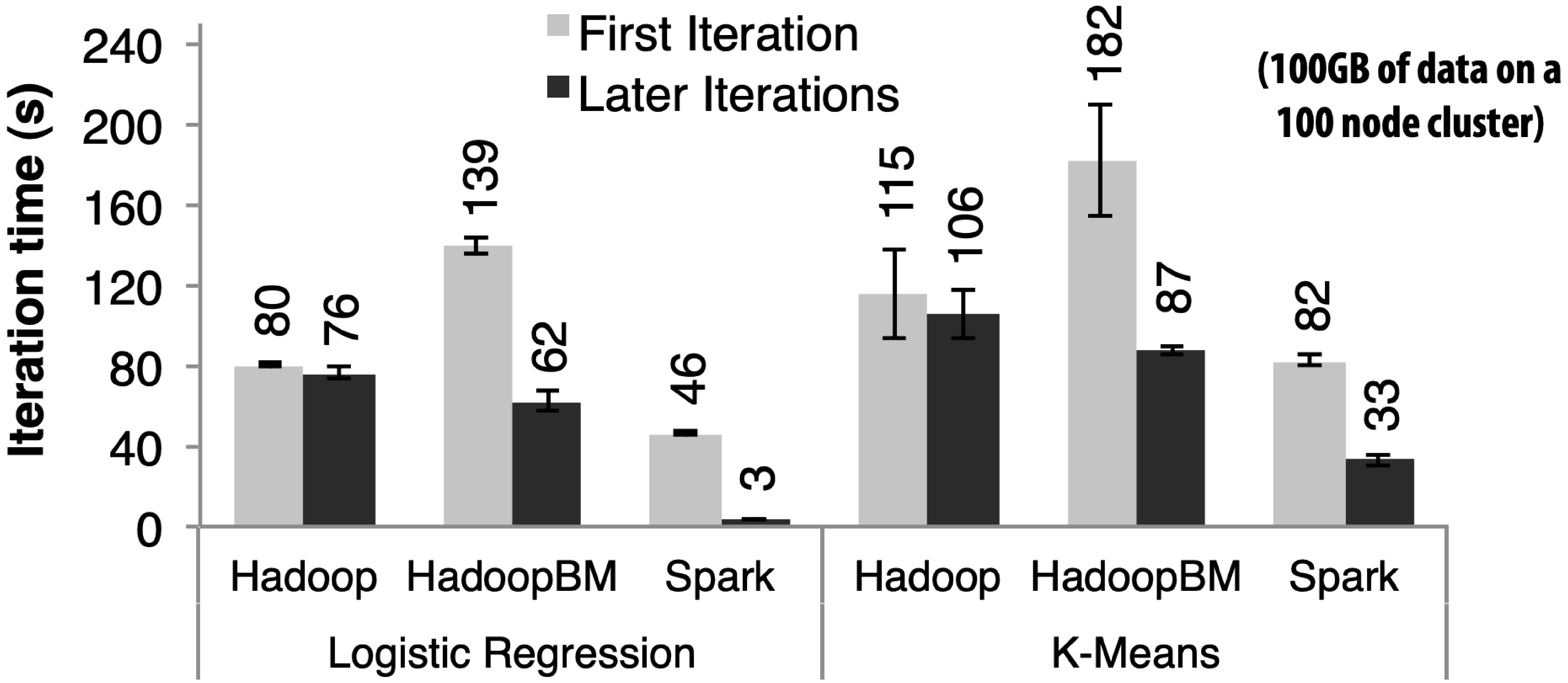


Must reload required subset of data from disk and recompute entire sequence of operations given by lineage to regenerate partitions 2 and 3 of RDD timestamps.

Note: (not shown): file system data is replicated so assume blocks 2 and 3 remain accessible to all nodes



Spark Performance



Spark Improves MapReduce Over

- Easy for programmers because you express your computation by chaining atomic operators
- Much fewer I/O -> very improved AI

Spark Cons?

- Debuggability
- Bulky
    - Map-reduce is not bulky as it works well if you only have one worker. That’s why now every PL has a “map” function

Caution: “scale out” is not the entire story

- Distributed systems designed for cloud execution address many difficult challenges, and have been instrumental in the explosion of “big-data” computing and large-scale analytics
  - Scale-out parallelism to many machines
  - Resiliency in the face of failures
  - Complexity of managing clusters of machines
- But scale out is not the whole story:

20 Iterations of Page Rank

scalable system	cores	twitter	uk-2007-05
GraphChi [10]	2	3160s	6972s
Stratosphere [6]	16	2250s	-
X-Stream [17]	16	1488s	-
Spark [8]	128	857s	1759s
Giraph [8]	128	596s	1235s
GraphLab [8]	128	249s	833s
GraphX [8]	128	419s	462s
Single thread (SSD)	1	300s	651s
Single thread (RAM)	1	275s	-

name	twitter_rv [11]	uk-2007-05 [4]
nodes	41,652,230	105,896,555
edges	1,468,365,182	3,738,733,648
size	5.76GB	14.72GB

Further optimization of the baseline brought time down to 110s

Vertex order (SSD)	1	300s	651s
Vertex order (RAM)	1	275s	-
Hilbert order (SSD)	1	242s	256s
Hilbert order (RAM)	1	110s	-

[“Scalability! At what COST?” McSherry et al. HotOS 20

Modern Spark ecosystem

Compelling feature: enables integration/composition of multiple domain-specific frameworks (since all collections implemented under the hood with RDDs and scheduled using Spark scheduler)

Spark SQL

```
sqlCtx = new HiveContext(sc)
results = sqlCtx.sql(
  "SELECT * FROM people")
names = results.map(lambda p: p.name)
```

Interleave computation and database query
Can apply transformations to RDDs produced by SQL queries

Spark MLlib

Machine learning library build on top of Spark abstractions.

```
points = spark.textFile("hdfs://...")
              .map(parsePoint)

model = KMeans.train(points, k=10)
```

Spark GraphX

GraphLab-like library built on top of Spark abstractions.

```
graph = Graph(vertices, edges)
messages = spark.textFile("hdfs://...")
graph2 = graph.joinVertices(messages) {
  (id, vertex, msg) => ...
}
```

Story time: Spark and Databricks

- Initially just an open-source project by a few students
- The community grows because of advantages over Hadoop and Map-reduce
- Students were about to graduate and could not commit time to those projects, what's next?
- "We asked Hortonworks if they wanted to take over Spark…They were not willing… We started Databricks."
- Hortonworks -> later merged with Cloudera at 2019

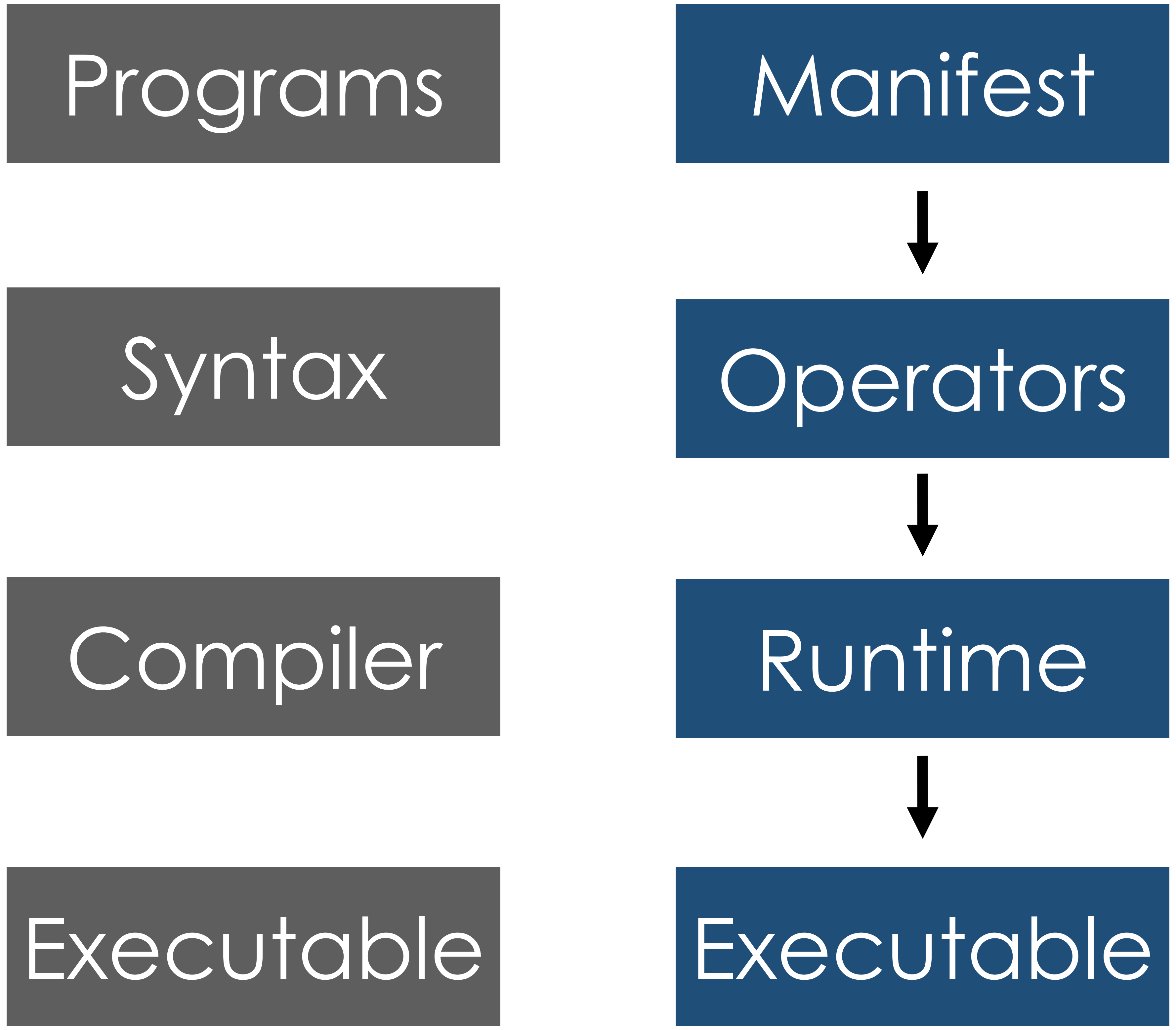
Spark and Databricks

- Cloudera: data platform company, founded by Hadoop authors
  - Used to be a unicorn / high-profile / high-tech company
  - Was beat hard by Databricks / Snowflake
  - Went to public 2017, stock price keeps declining…, merged with Hortonworks in 2018, went to private in 2021 after being acquired by investment companies.
- Databricks: 7 cofounders, Initial CEO is Prof. Ion Stoica.
  - They tried to sell Spark but were unsuccessful
  - Switched to Ali Ghodsi: Iranian-Swedish, visitor to UC Berkeley, no US-born nor US-educated

Spark and Databricks

- Databricks struggled for quite a few years
  - Raised up to Series I (Seed, A, B, C, D, E, F, G, H, I)
  - Almost failed during 2018 – 2020
  - Data warehousing and OLAP gradually become a business, why?
    - Competitors all failed
    - Customer Education
    - Data indeed bigger and bigger
  - Intended to go public in 2022, but hit covid
  - Valued at 43B today (is there any bubble? No one knows)
  - Create 3 billionaires
  - Competitions with Snowflake are intense

After Spark:
All Modern Data/ML Systems follow a similar architecture



A fixed set of operators

A trusted runtime with a small set of pre-loaded implementations

After Spark: Many new systems

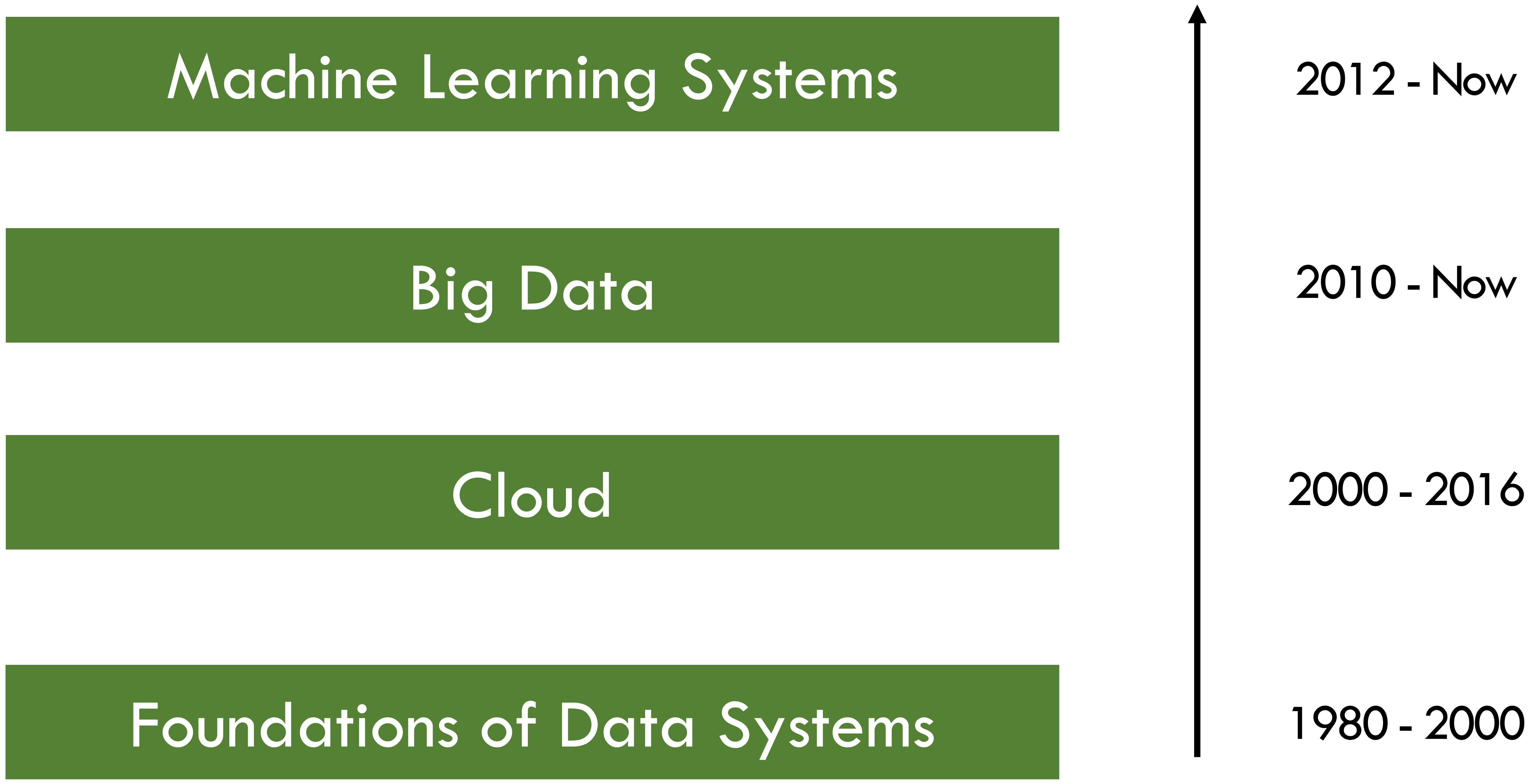
APACHE Spark™

GraphLab

Naiad

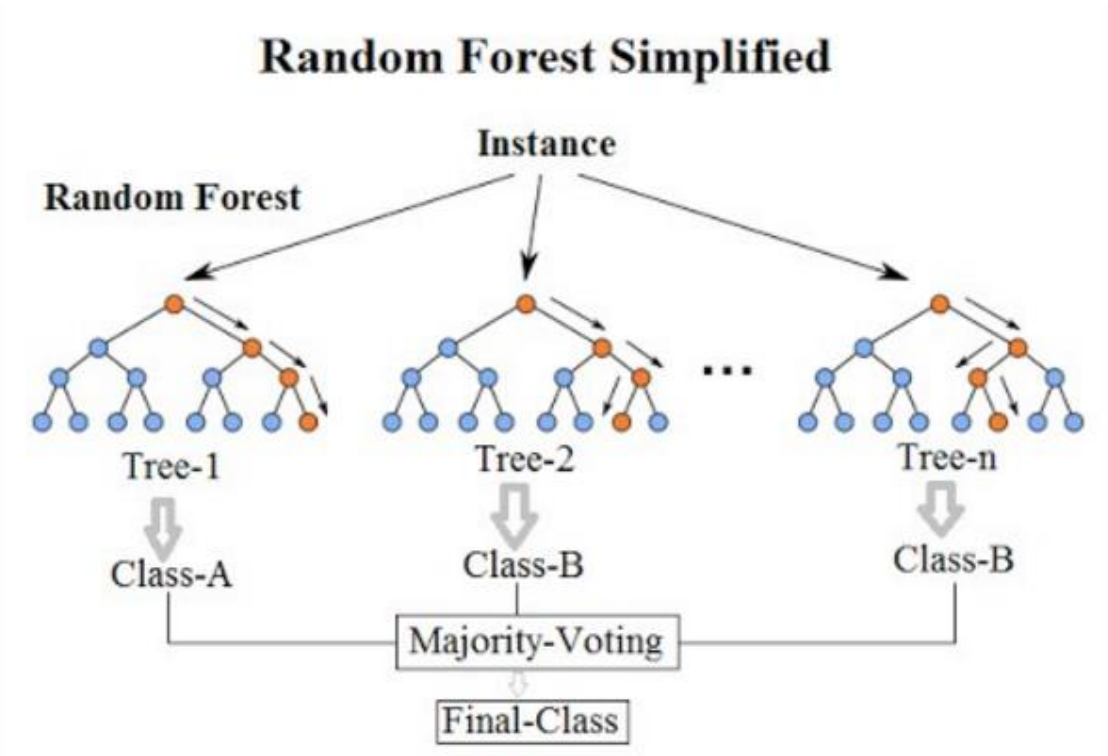
TensorFlow

Where We Are


Machine Learning Systems
2012 - Now
Big Data
2010 - Now
Cloud
2000 - 2016
Foundations of Data Systems
1980 - 2000

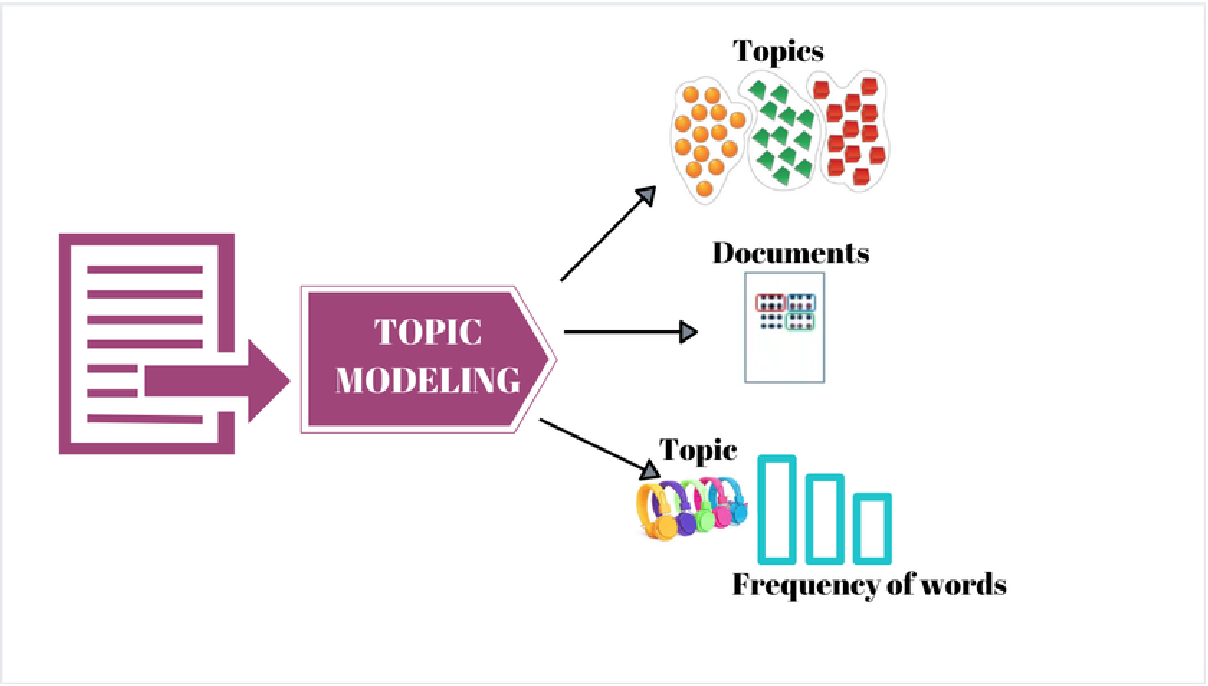
ML Era starts (roughly 2012, when Spark starts to take off)

- ML was still a mess in 2012

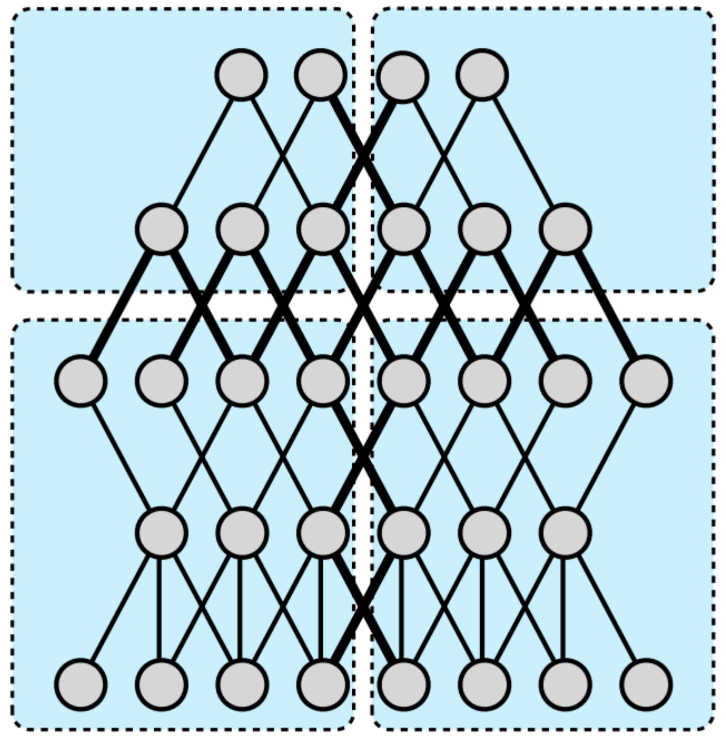


XGBOOST

Spark mllib



LDA



Torch (lua) / Theano / distbelief

Diversity -> Good News or Bad New?

- ML is so diverse
  - Cons:
    - There is no unified model / computation
    - Hard to build a programming model / interface that cover a diverse range of applications
    - No idea where the system bottlenect is
  - Pros:
    - A lot of opportunities: Gold mining era

Gradient descent is what people find very common first

Gradient / backward computation

$$\theta^{(t)} = \theta^{(t-1)} + \boxed{\varepsilon \cdot \nabla_{\mathcal{L}}(\theta^{(t-1)}, D^{(t)})}$$

objective data

- The first unification:
  - Most ML algorithms are **iterative-convergent**
  - Gradient descent is the master equation behind

How to Distribute this Equation?

Gradient / backward computation

$$\theta^{(t)} = \theta^{(t-1)} + \boxed{\varepsilon \cdot \nabla_{\mathcal{L}}(\theta^{(t-1)}, D^{(t)})}$$

objective

data

$$\theta^{(t+1)} = \theta^{(t)} + \varepsilon \sum_{p=1}^{P} \nabla_{\mathcal{L}}(\theta^{(t)}, D_p^{(t)})$$

How to perform this sum?

Problems if expressing this in Spark

- ML is too diverse; hard to express their computation in coarse-grained data transformations.

$map(f : \mathrm{T} \Rightarrow \mathrm{U})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[U]}$
$filter(f : \mathrm{T} \Rightarrow \mathrm{Bool})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[T]}$
$flatMap(f : \mathrm{T} \Rightarrow \mathrm{Seq[U]})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[U]}$
$sample(fraction : \mathrm{Float})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[T]}$ (Deterministic sampling)
$groupByKey()$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, Seq[V])]}$
$reduceByKey(f : (\mathrm{V}, \mathrm{V}) \Rightarrow \mathrm{V})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$
$union()$	:	$(\mathrm{RDD[T]}, \mathrm{RDD[T]}) \Rightarrow \mathrm{RDD[T]}$
$join()$	:	$(\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \Rightarrow \mathrm{RDD[(K, (V, W))]}$
$cogroup()$	:	$(\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \Rightarrow \mathrm{RDD[(K, (Seq[V], Seq[W]))]}$
$crossProduct()$	:	$(\mathrm{RDD[T]}, \mathrm{RDD[U]}) \Rightarrow \mathrm{RDD[(T, U)]}$
$mapValues(f : \mathrm{V} \Rightarrow \mathrm{W})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, W)]}$ (Preserves partitioning)
$sort(c : \mathrm{Comparator[K]})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$
$partitionBy(p : \mathrm{Partitioner[K]})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$

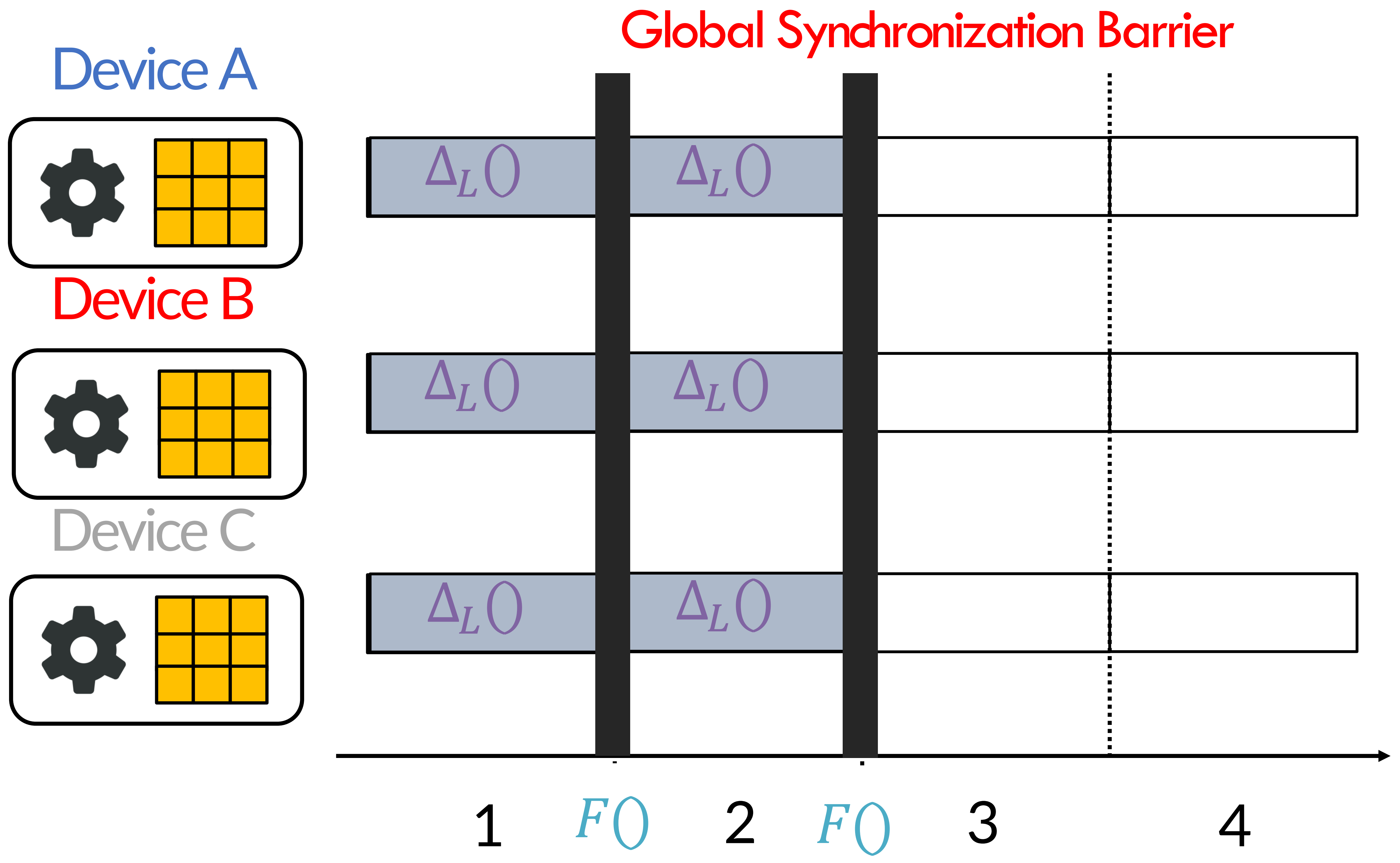
Problems if expressing this in Spark

$$\theta^{(t+1)} = \theta^{(t)} + \varepsilon \sum_{p=1}^{P} \nabla_{\mathcal{L}}(\theta^{(t)}, D_p^{(t)})$$

- Very heavy communication per iteration
- Compute : communication = 1:10 in the era of 2012

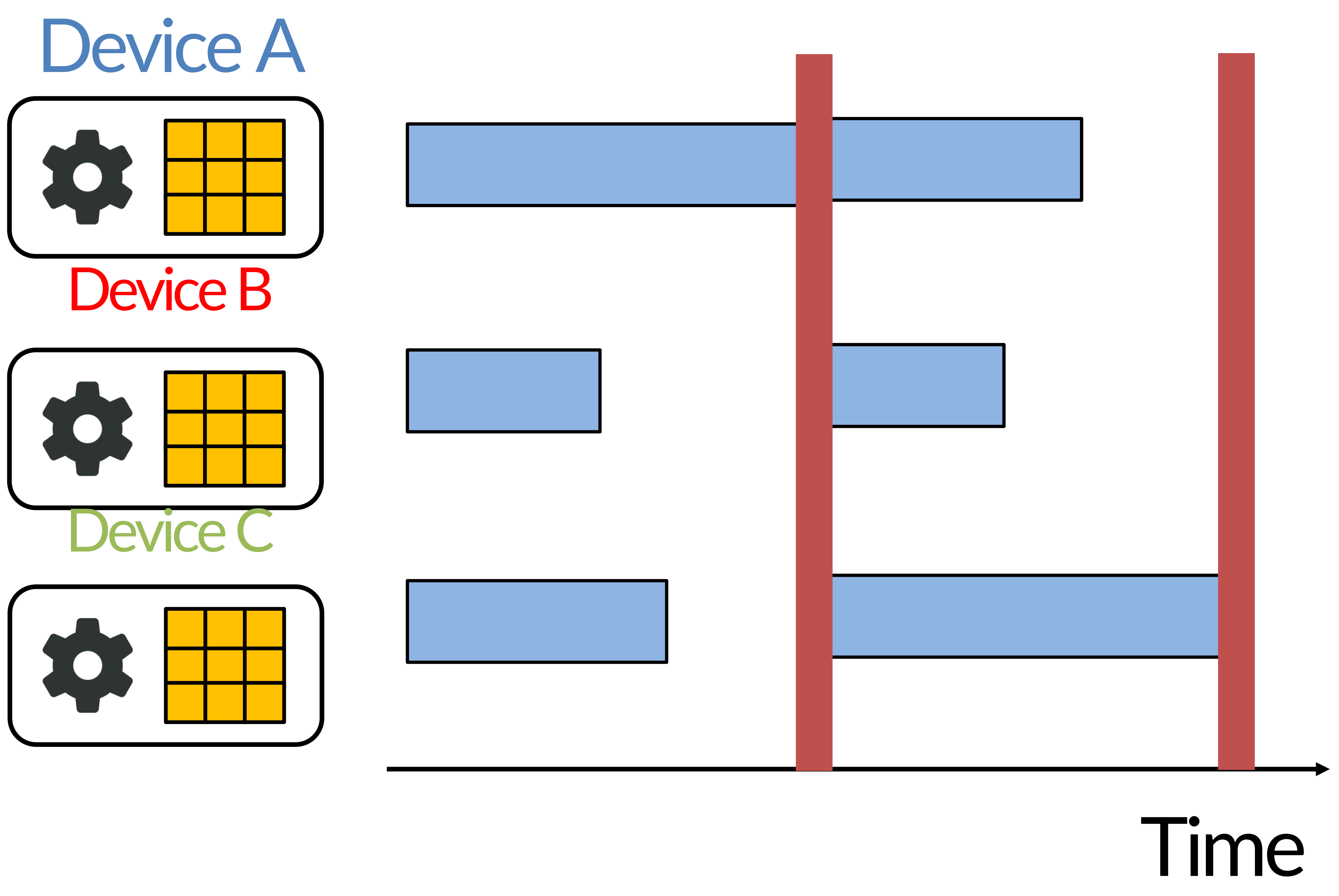
Consistency

$$\theta^{(t+1)} = \theta^{(t)} + \varepsilon \sum_{p=1}^{P} \nabla_{\mathcal{L}}(\theta^{(t)}, D_p^{(t)})$$



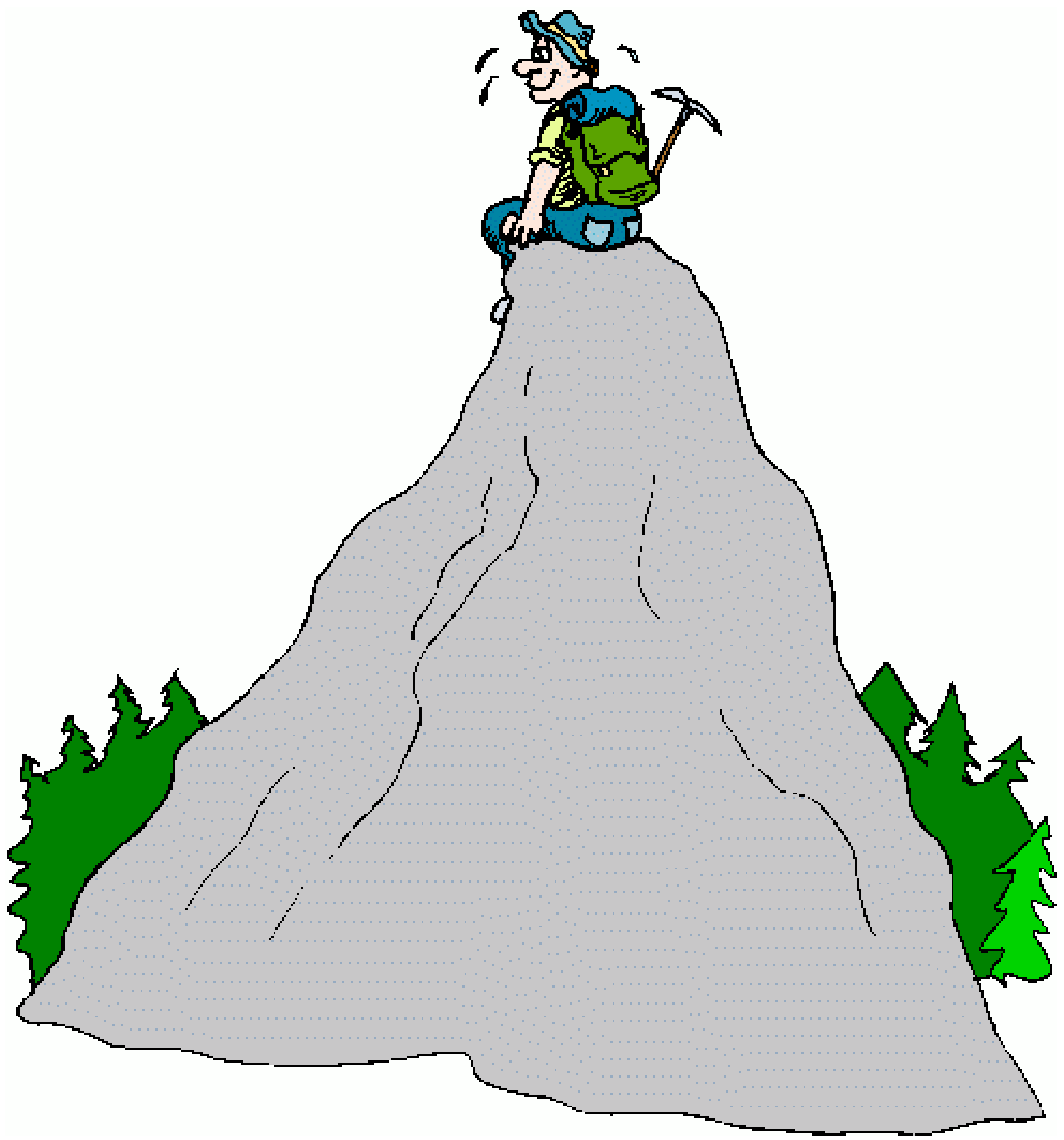
BSP's Weakness: Stragglers

- **BSP suffers from stragglers**
  - Slow devices (stragglers) force all devices to wait
  - More devices → higher chance of having a straggler
- **Stragglers are usually transient, e.g.**
  - Temporary compute/network load in multi-user environment
  - Fluctuating environmental conditions (temperature, vibrations)
- BSP's throughput is greatly decreased in large clusters/clouds, where stragglers are unavoidable

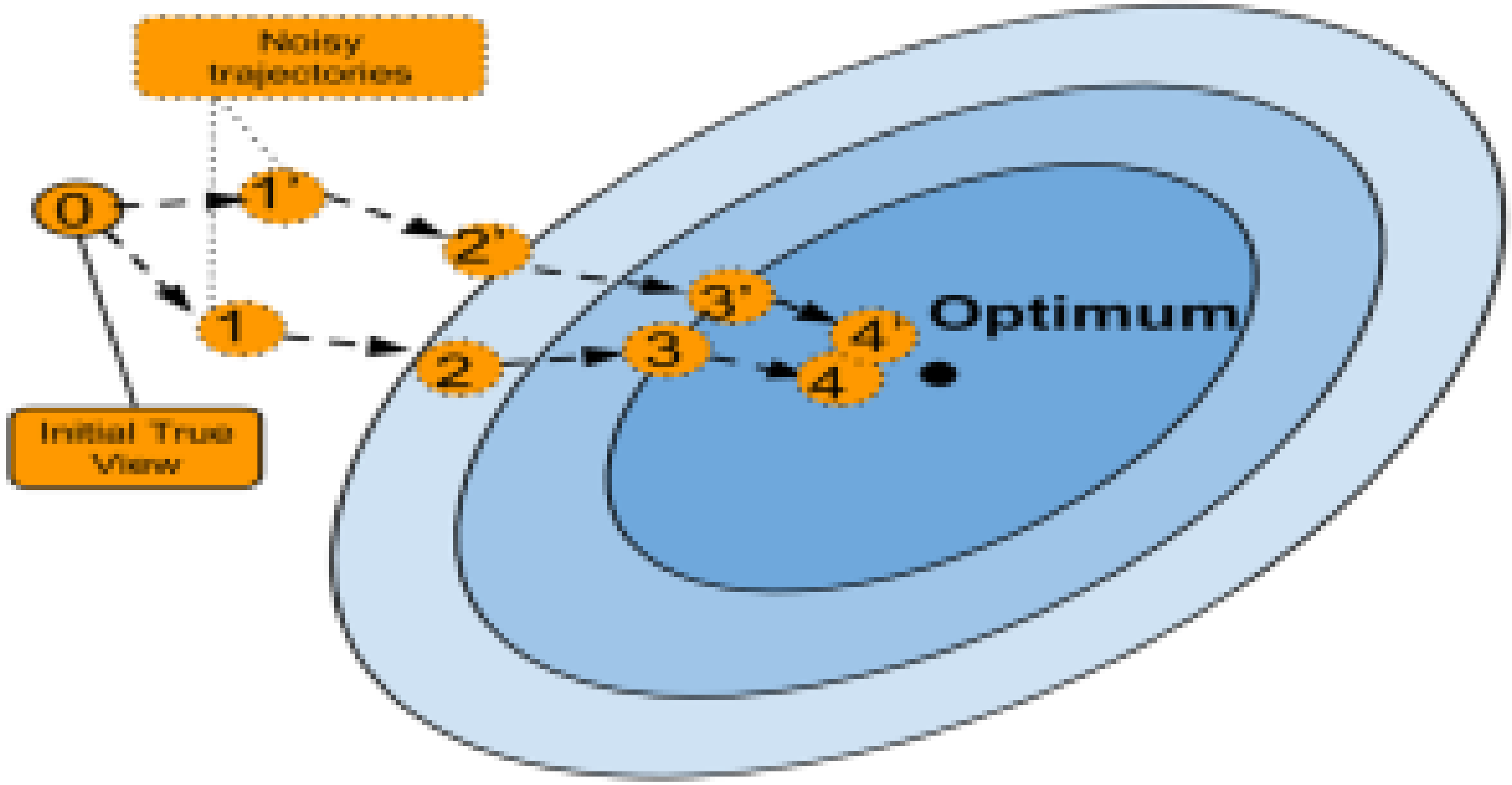


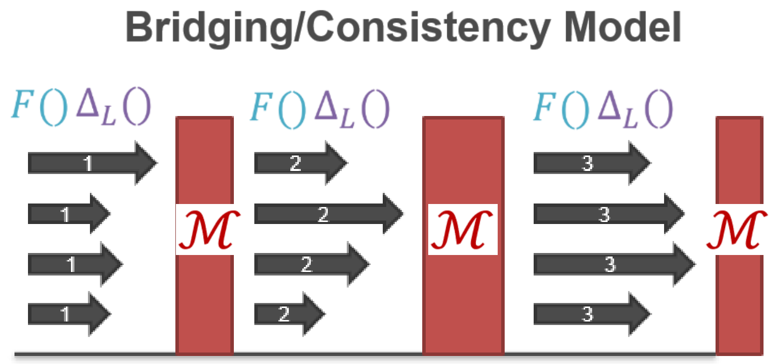
An interesting property of Gradient Descent (ascent)

$$\theta^{(t+1)} = \theta^{(t)} + \varepsilon \sum_{p=1}^{P} \nabla_{\mathcal{L}}(\theta^{(t)}, D_p^{(t)})$$



Machine Learning is Error-tolerant (under certain conditions)

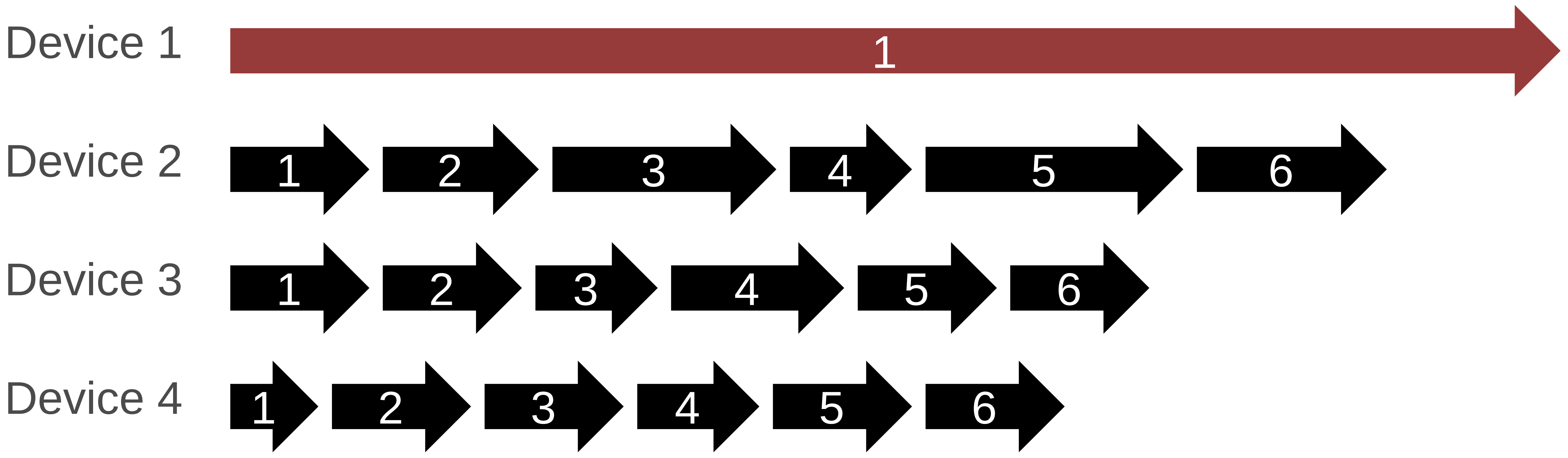




Background: Asynchronous Communication (No Consistency)

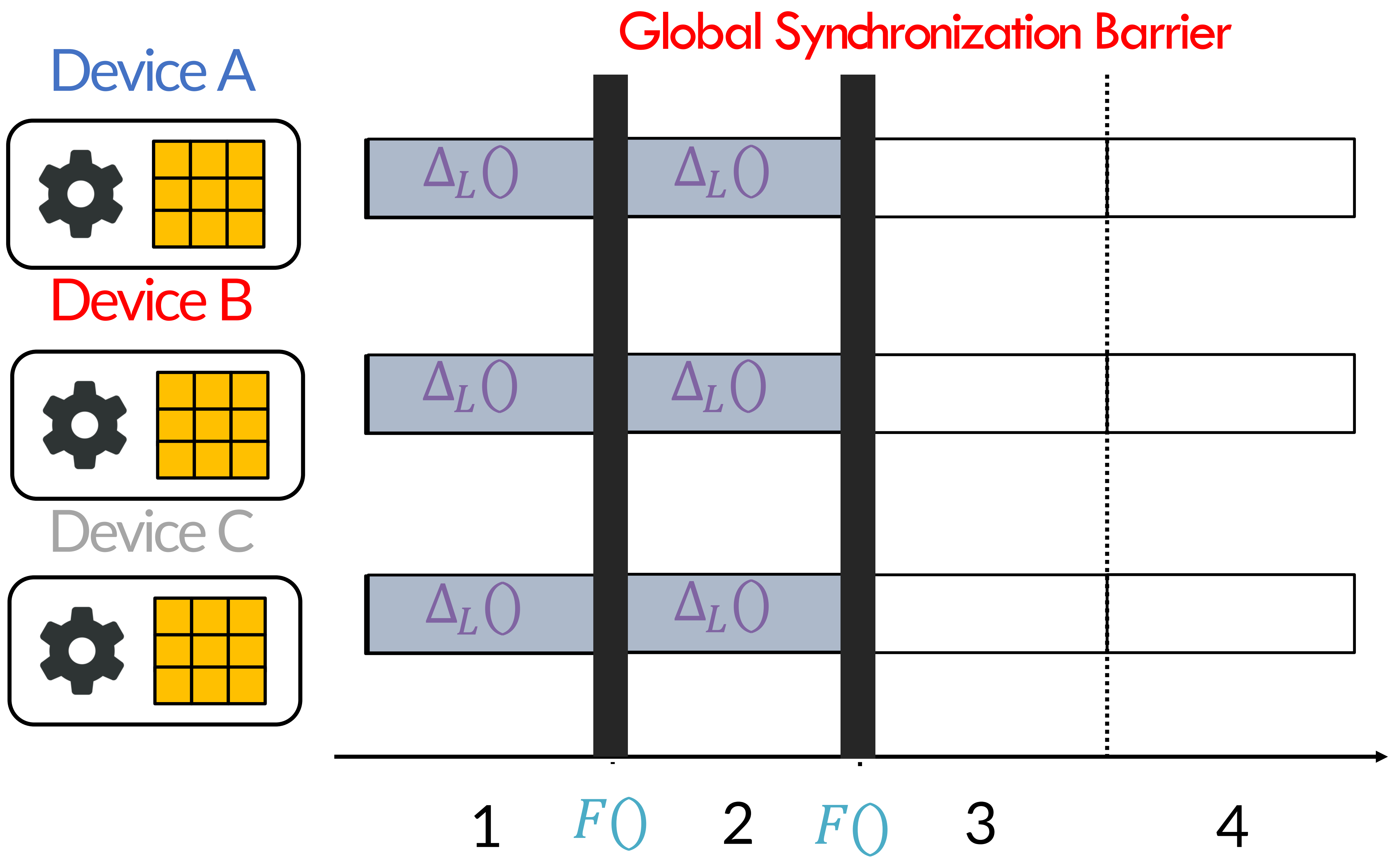
- **Asynchronous (Async):** removes all communication barriers
  - Maximizes computing time
  - Transient stragglers will cause messages to be extremely stale
    - Ex: Device 2 is at $t = 6$, but Device 1 has only sent message for $t = 1$

- **Some Async software:** <u>messages can be applied while computing</u> $F()$, $\Delta_L()$
  - Unpredictable behavior, can hurt statistical efficiency!



Background: Strict Consistency

- **Baseline:** Bulk Synchronous Parallel (BSP)
  - MapReduce, Spark, many DistML Systems
- Devices compute updates $\Delta_L()$ between global barriers (iteration boundaries)
  - Messages $\mathcal{M}$ exchanged only during barriers
- **Advantage: Execution is serializable**
  - Same guarantees as sequential algo!
  - Provided that aggregation $F()$ is agnostic to order of messages $\mathcal{M}$ (e.g. in SGD)

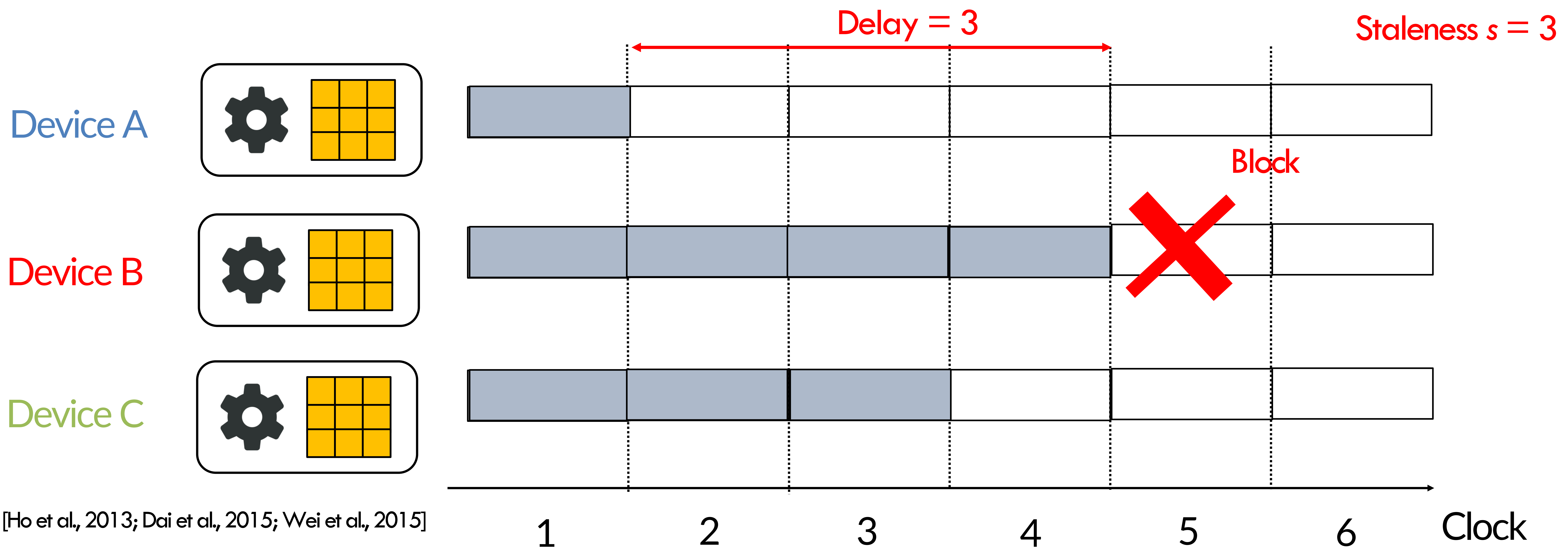


Background: Bounded Consistency

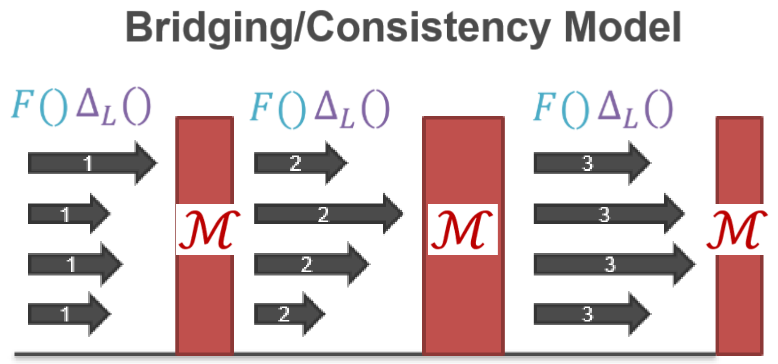
Bounded consistency models: Middle ground between BSP and fully-asynchronous (no-barrier)

e.g. **Stale Synchronous Parallel (SSP):** Devices allowed to iterate at different speeds

- Fastest & slowest device must not drift $> s$ iterations apart (in this example, $s = 3$)
    - s is the maximum staleness



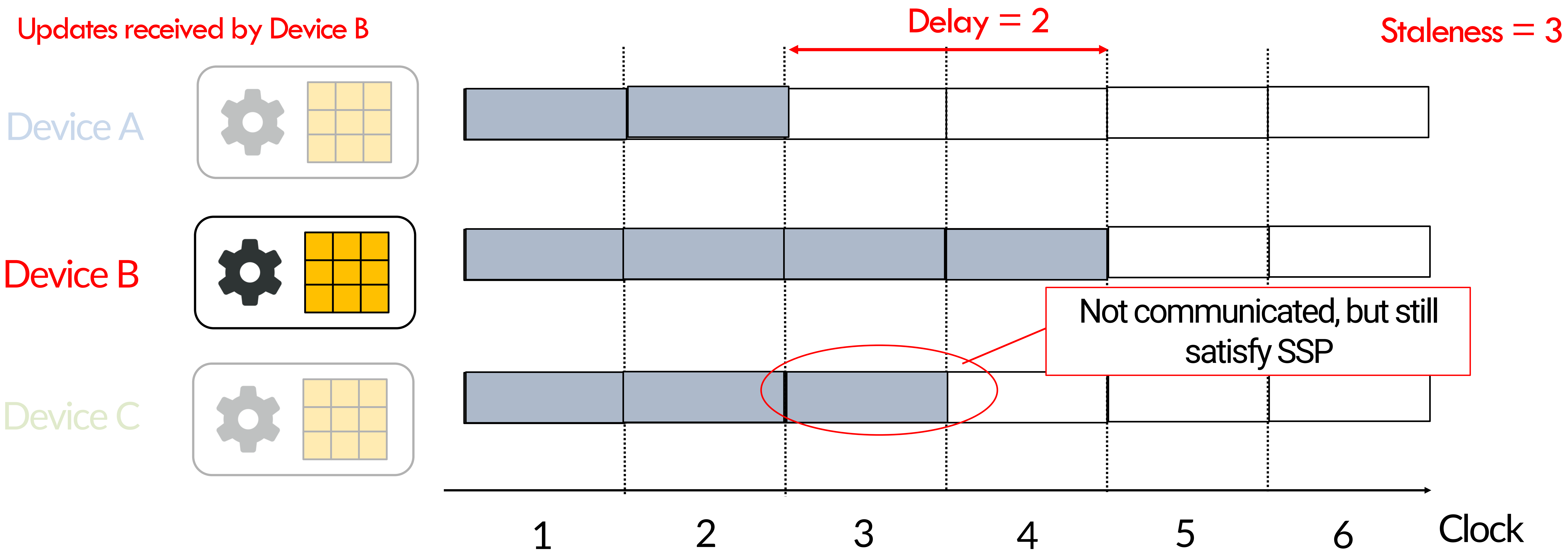
[Ho et al., 2013; Dai et al., 2015; Wei et al., 2015]



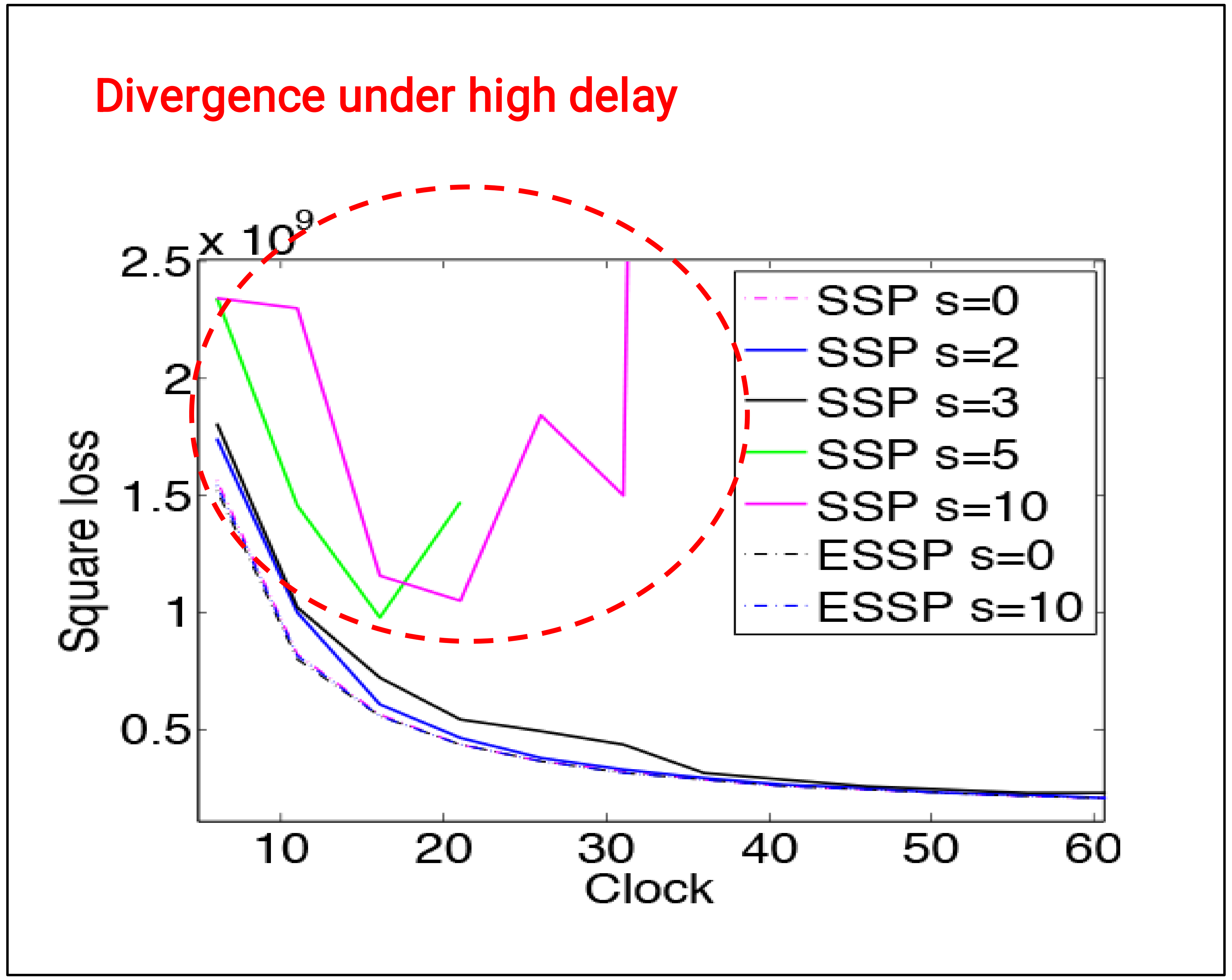
SSP: "Lazy" Communication

SSP [Ho et al., 2013]: devices avoid communicating unless necessary

- i.e. when staleness condition is about to be violated
- Favors throughput at the expense of statistical efficiency



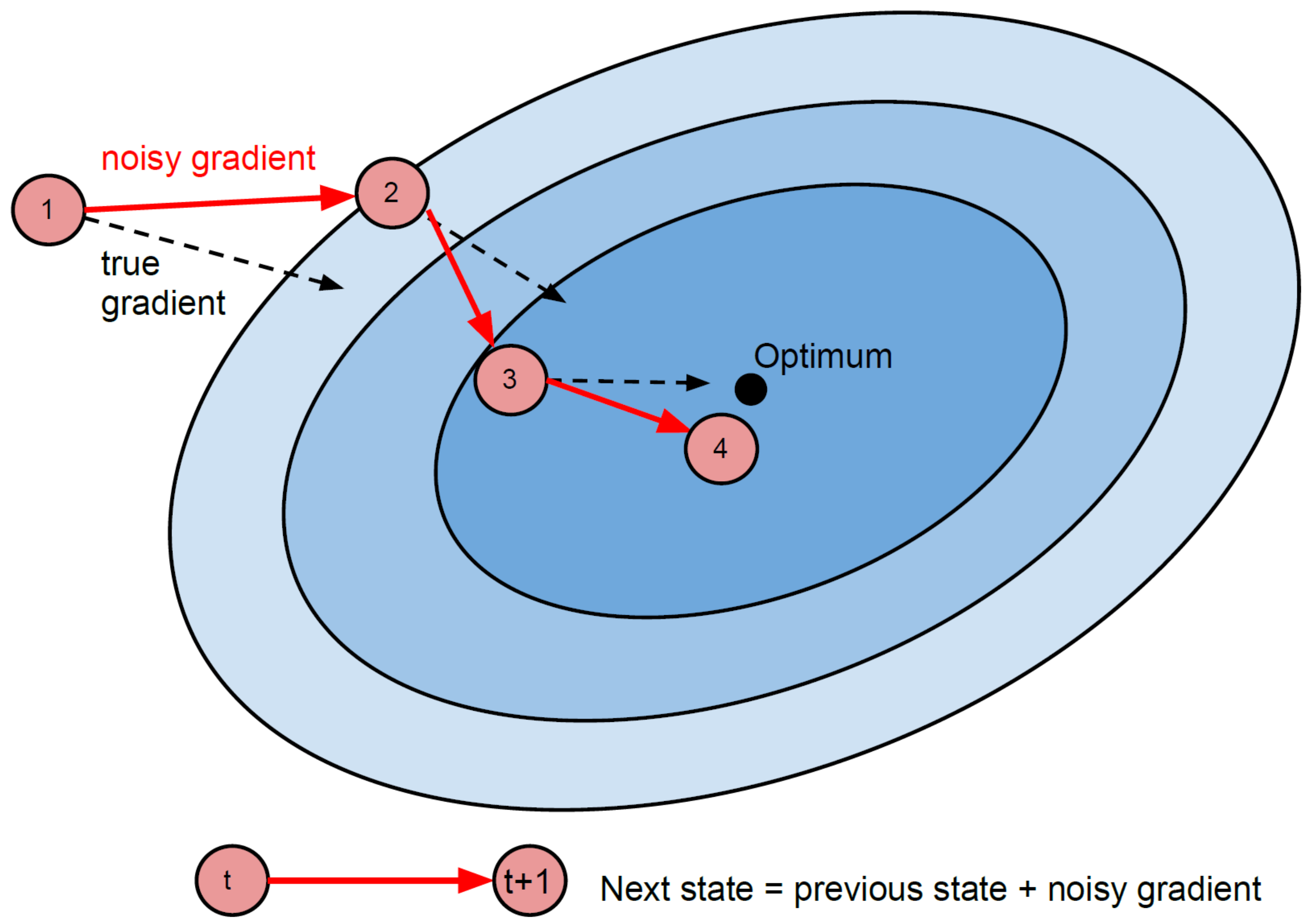
Impacts of Consistency/Staleness: Unbounded Staleness



Theory: (E)SSP Expectation Bound

- **Goal:** minimize convex $f(\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T} f_t(\mathbf{x})$

  (Example: Stochastic Gradient)
  - L-Lipschitz, problem diameter bounded by F^2
  - Staleness s, using P parallel devices
  - Use step size $\eta_t = \frac{\sigma}{\sqrt{t}}$ with $\sigma = \frac{F}{L\sqrt{2(s+1)P}}$
- **(E)SSP converges according to**
  - Where T is the number of iterations



$$R[\mathbf{X}] := \overbrace{\left[\frac{1}{T}\sum_{t=1}^{T} f_t(\tilde{\mathbf{x}}_t)\right]}^{\text{Difference between SSP estimate and true optimum}} - f(\mathbf{x}^*) \leq 4FL\sqrt{\frac{2(s+1)P}{T}}$$

Parameter Server