Thinking in MapReduce

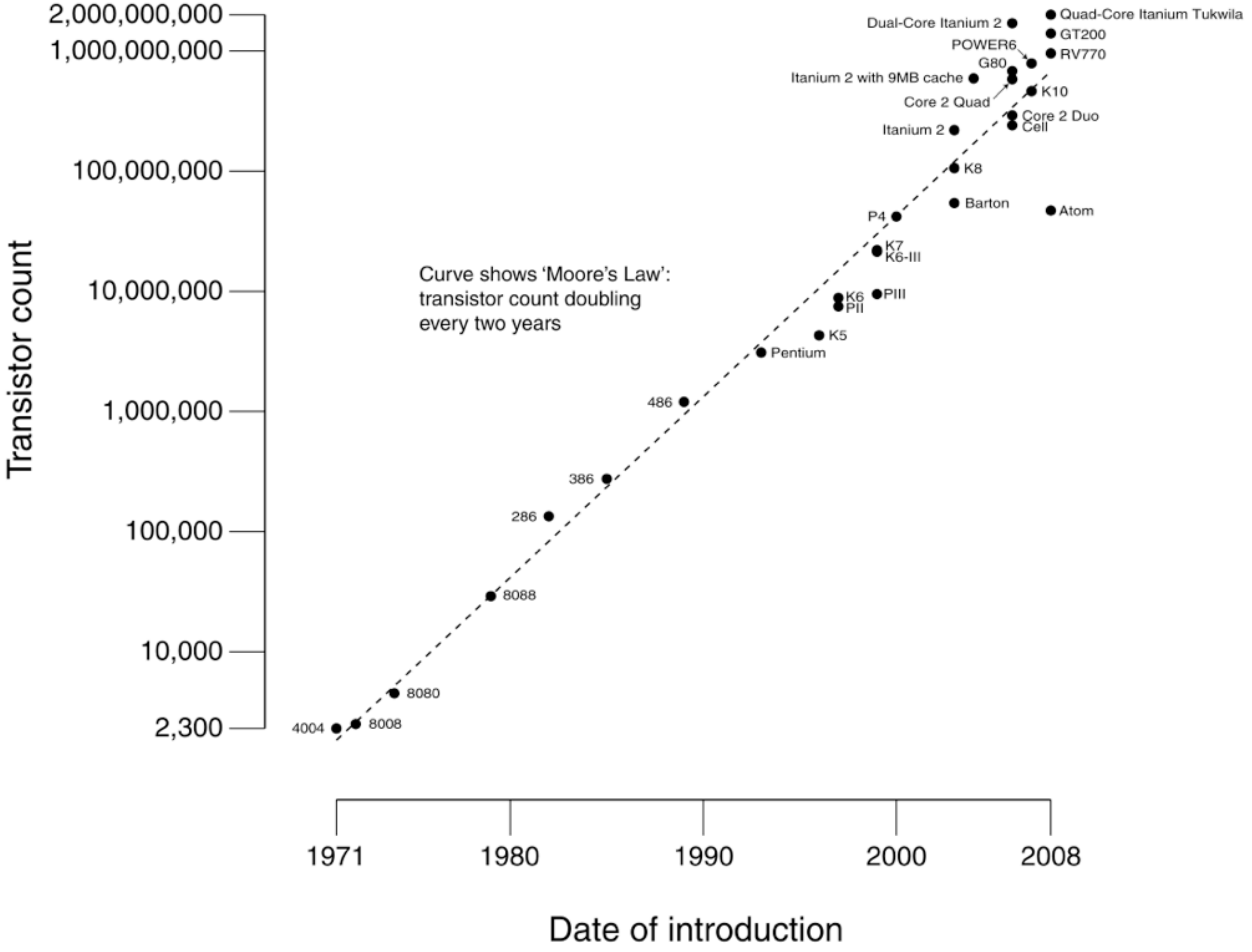
Ryan Brush

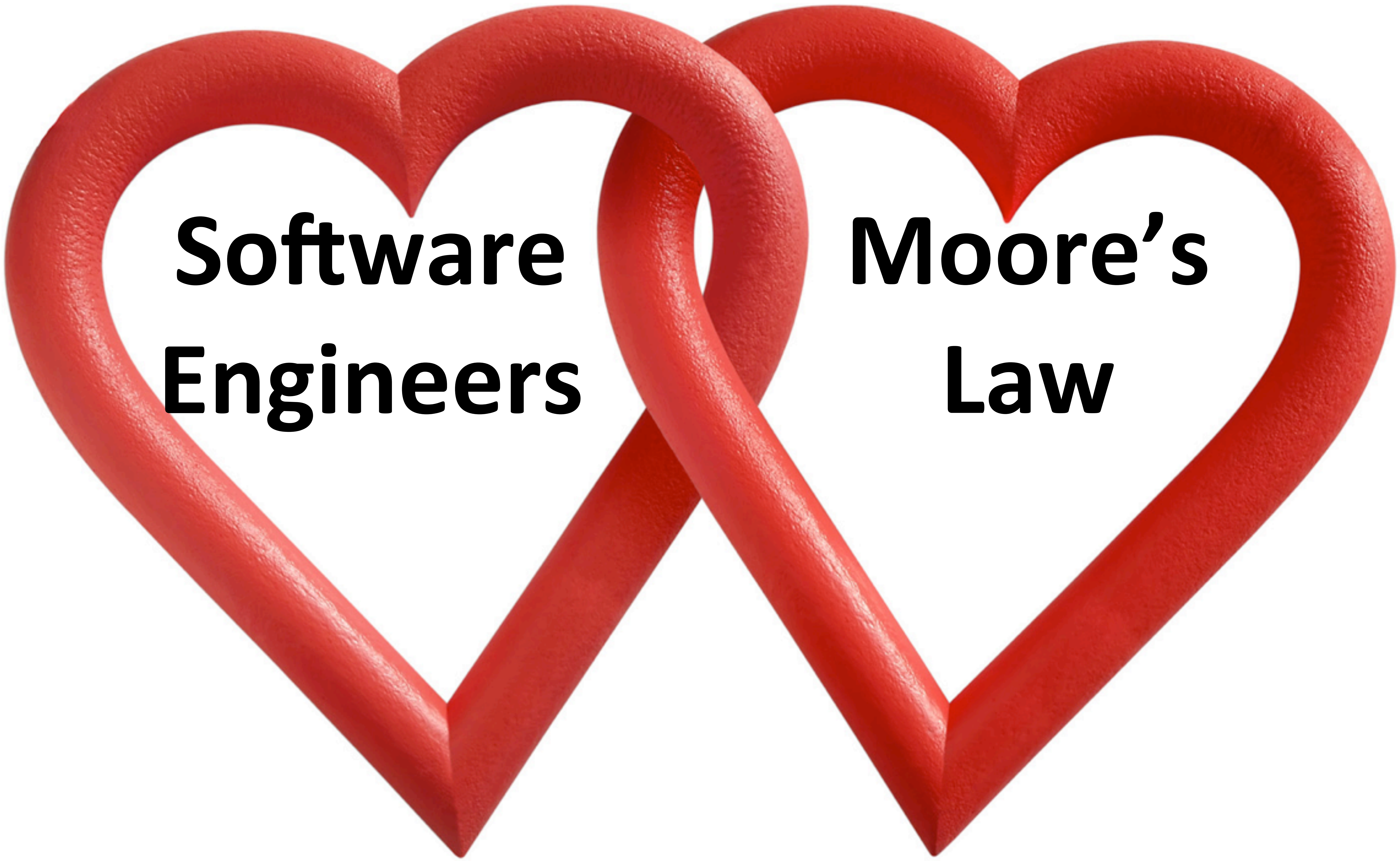
Cerner

We programmers have had
it pretty good

Hardware has scaled up faster than our problem sets

CPU Transistor Counts 1971-2008 & Moore's Law




Software
Engineers
Moore's
Law

But the party is ending
(or at least changing)

Data is growing faster than
we can scale individual machines

So we have to spread our work across many machines

This is a big deal in health care

Fragmented Information

Spread across many systems

No one has the complete picture

We need to put the picture back together again

Better-informed decisions

Reduce systematic friction

Understand and improve the health of populations

Chart Search

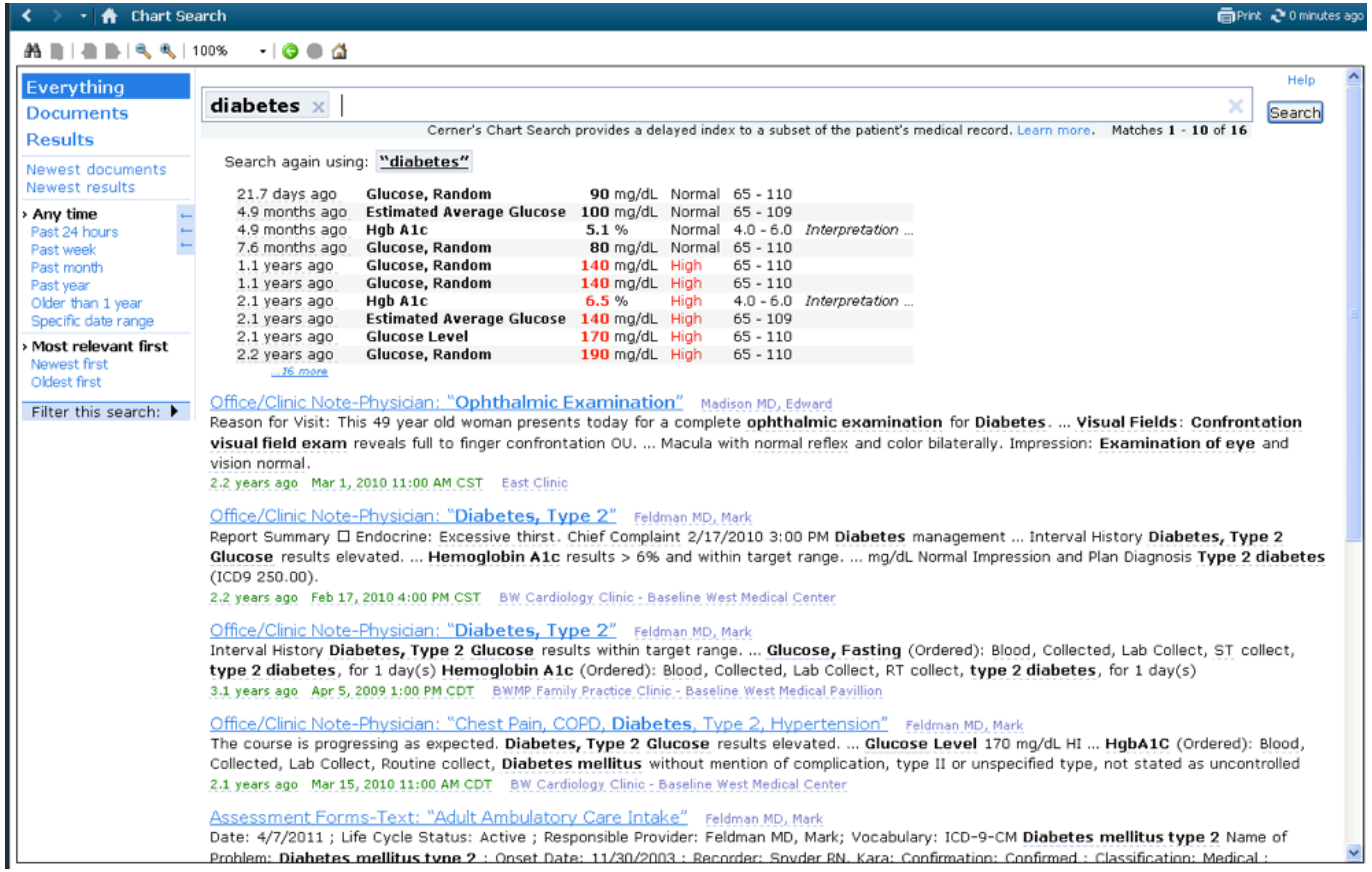
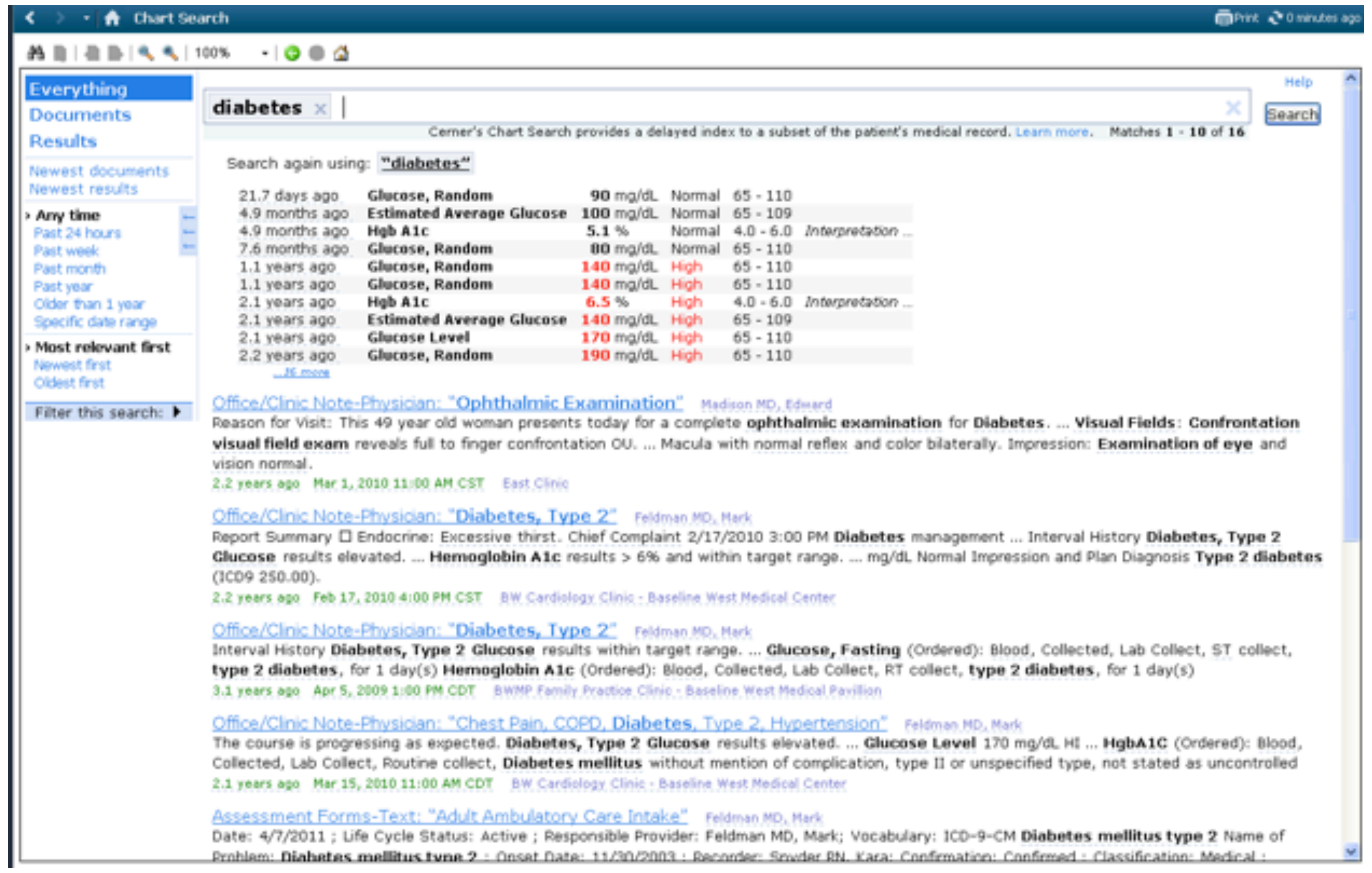
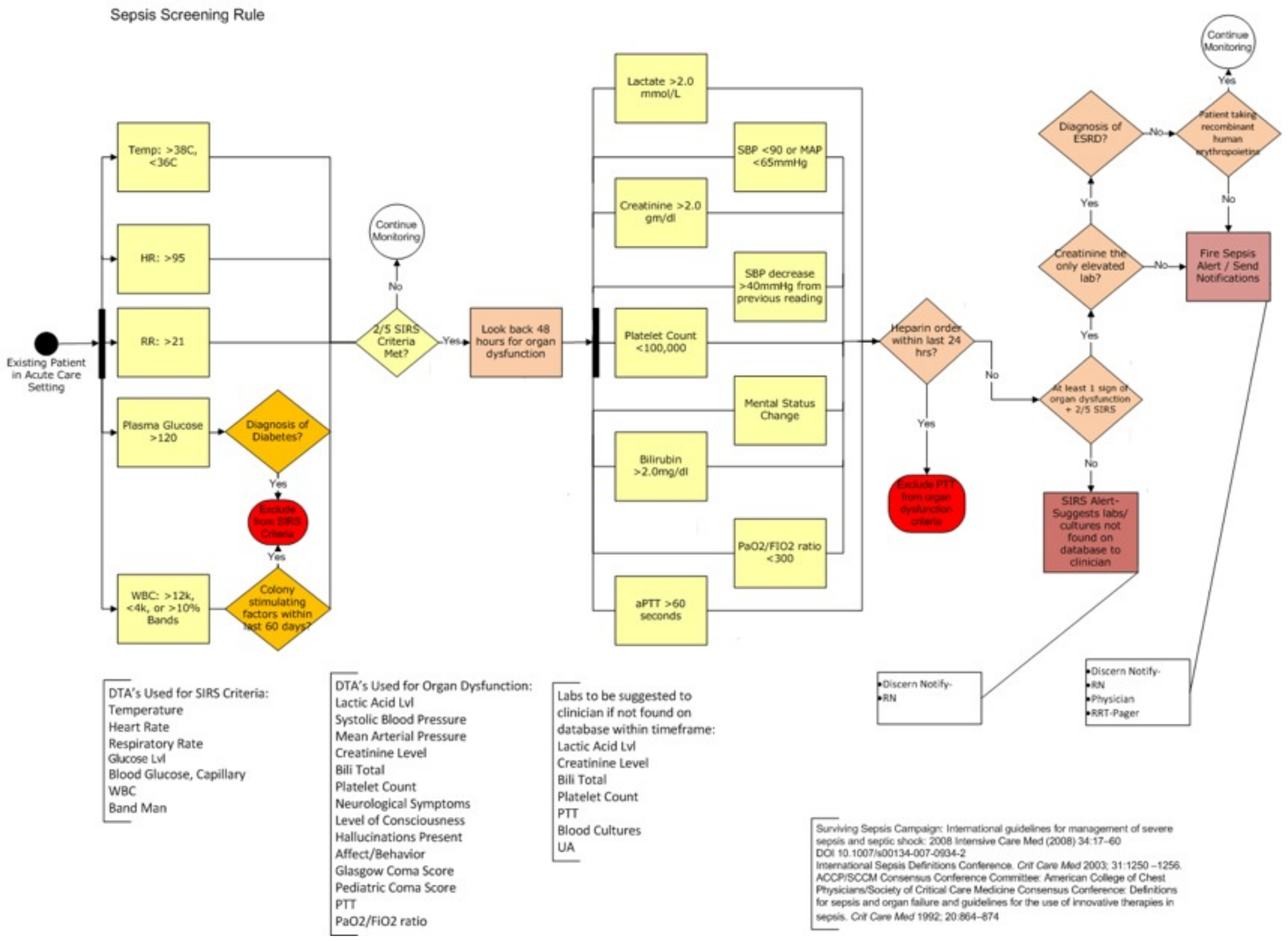


Chart Search

- Information extraction
- Semantic markup of documents
- Related concepts in search results

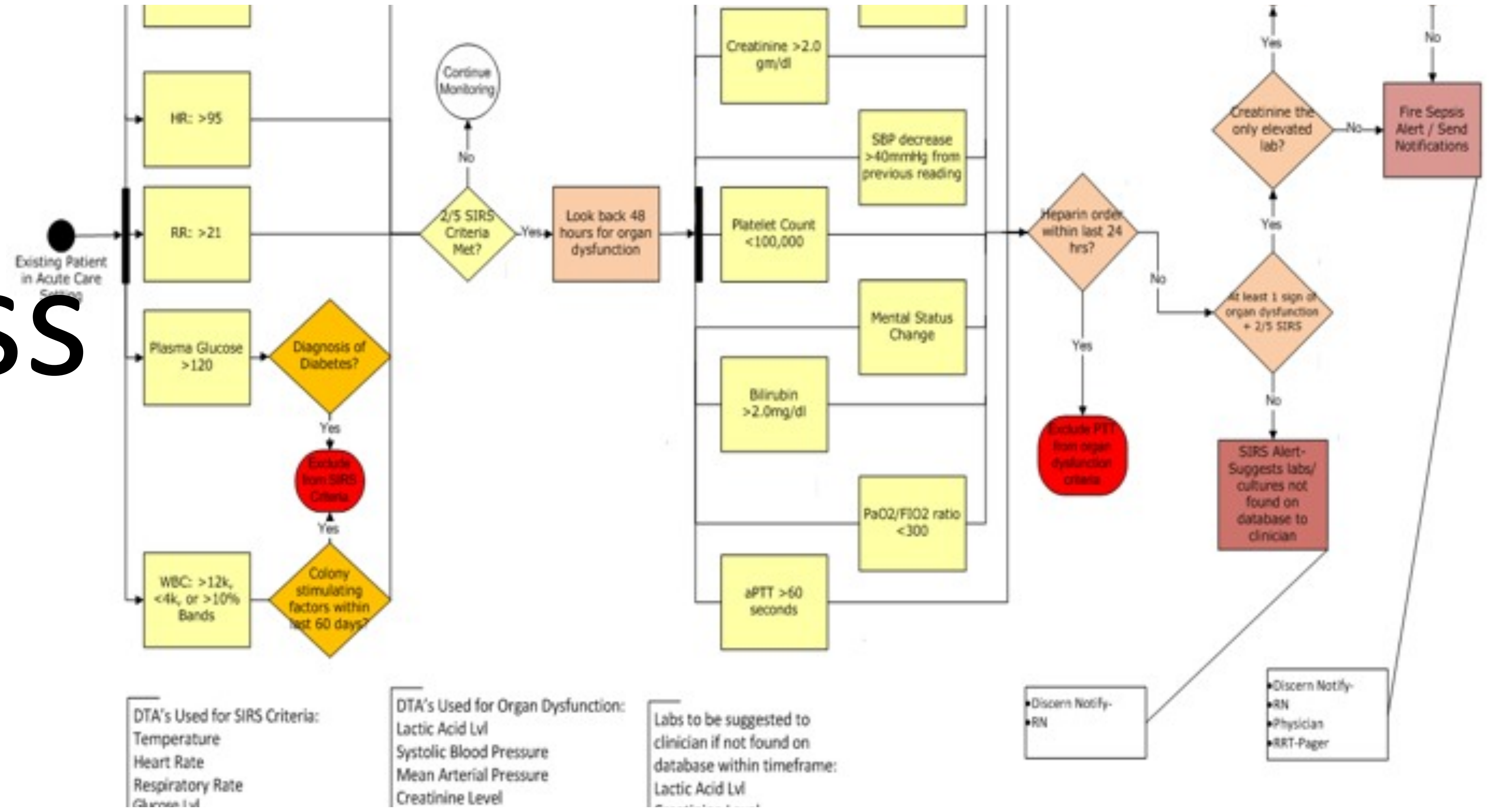


Medical Alerts

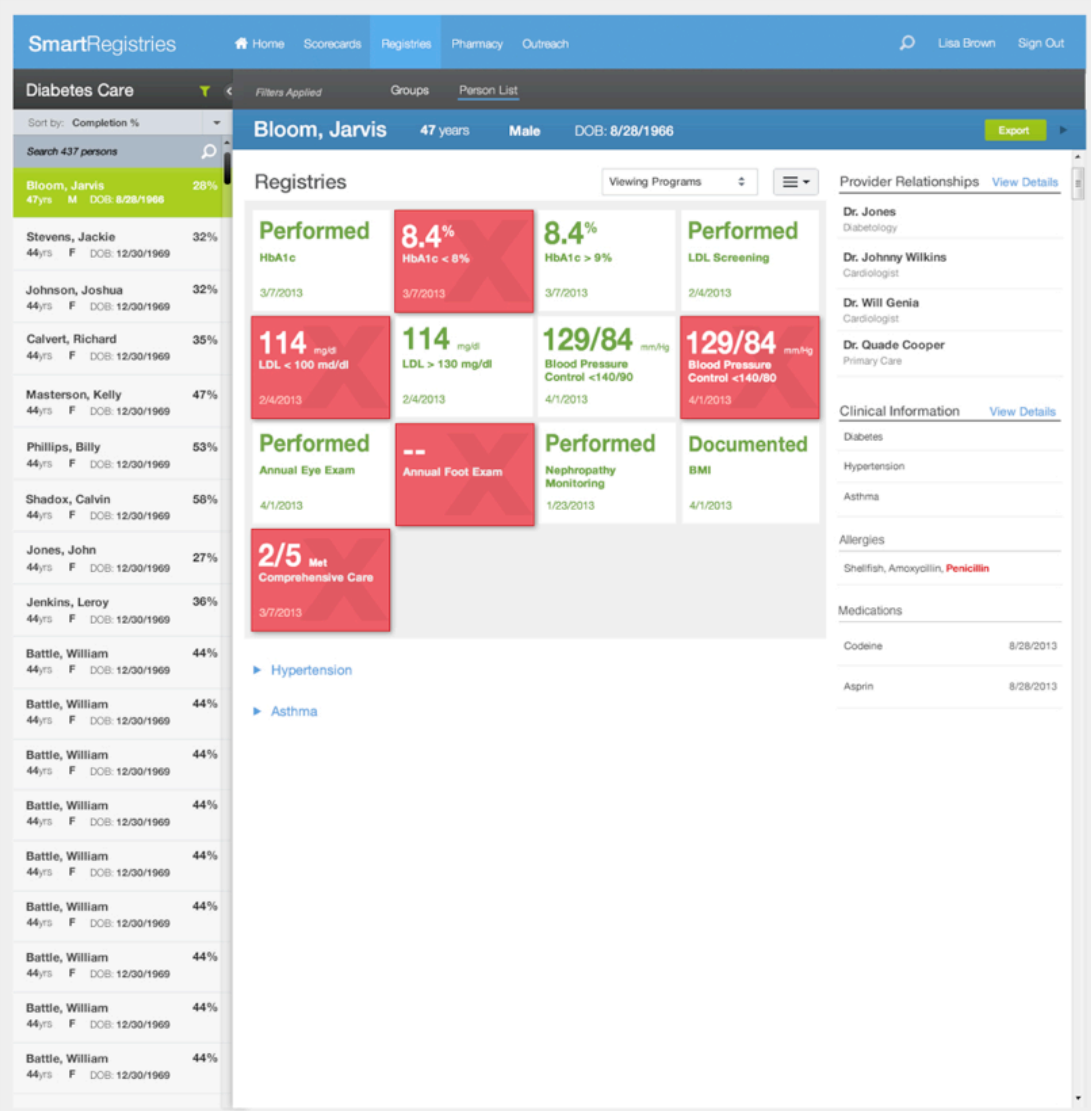


Medical Alerts

- Detect health risks in incoming data
- Notify clinicians to address those risks
- Quickly include new knowledge

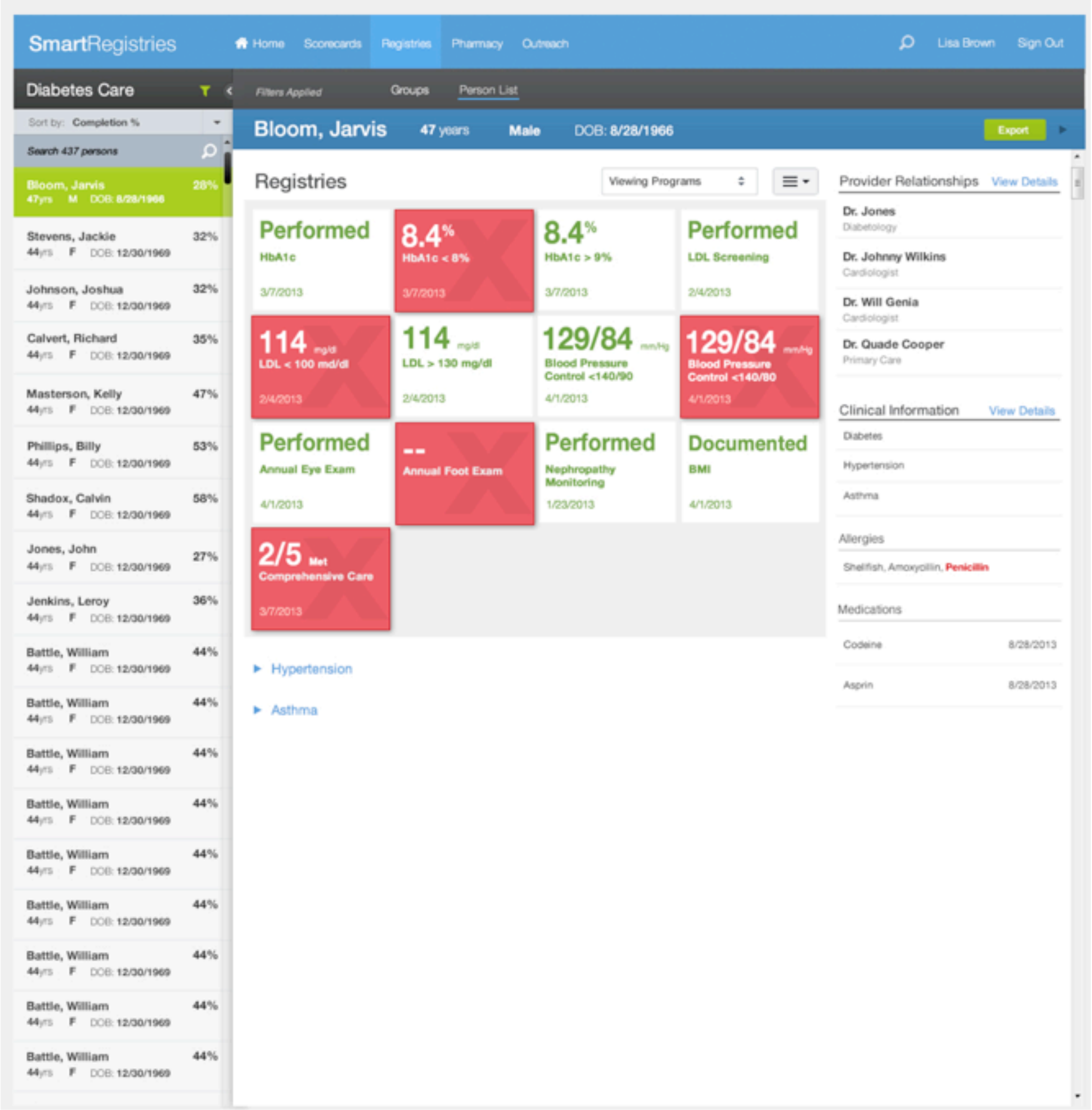


Population Health



Population Health

- –Securely bring together health data
- –Identify opportunities to improve care
- –Support application of improvements
- –Close the loop



The Unreasonable Effectiveness of Data

Simple models with lots of data almost always outperform complex models with less data

Peter Norvig, http://www.youtube.com/watch?v=yvDCzhbjYWs

So how can we tackle such large data sets?

Can we adapt what has worked historically?

After all,

Relational Databases are Awesome

Atomic, transactional updates

Guaranteed consistency

Relational Databases are Awesome

Declarative queries

Easy to reason about

Long track record of success

Relational Databases are Awesome

…so use them!

Relational Databases are Awesome

…so use them!

But…

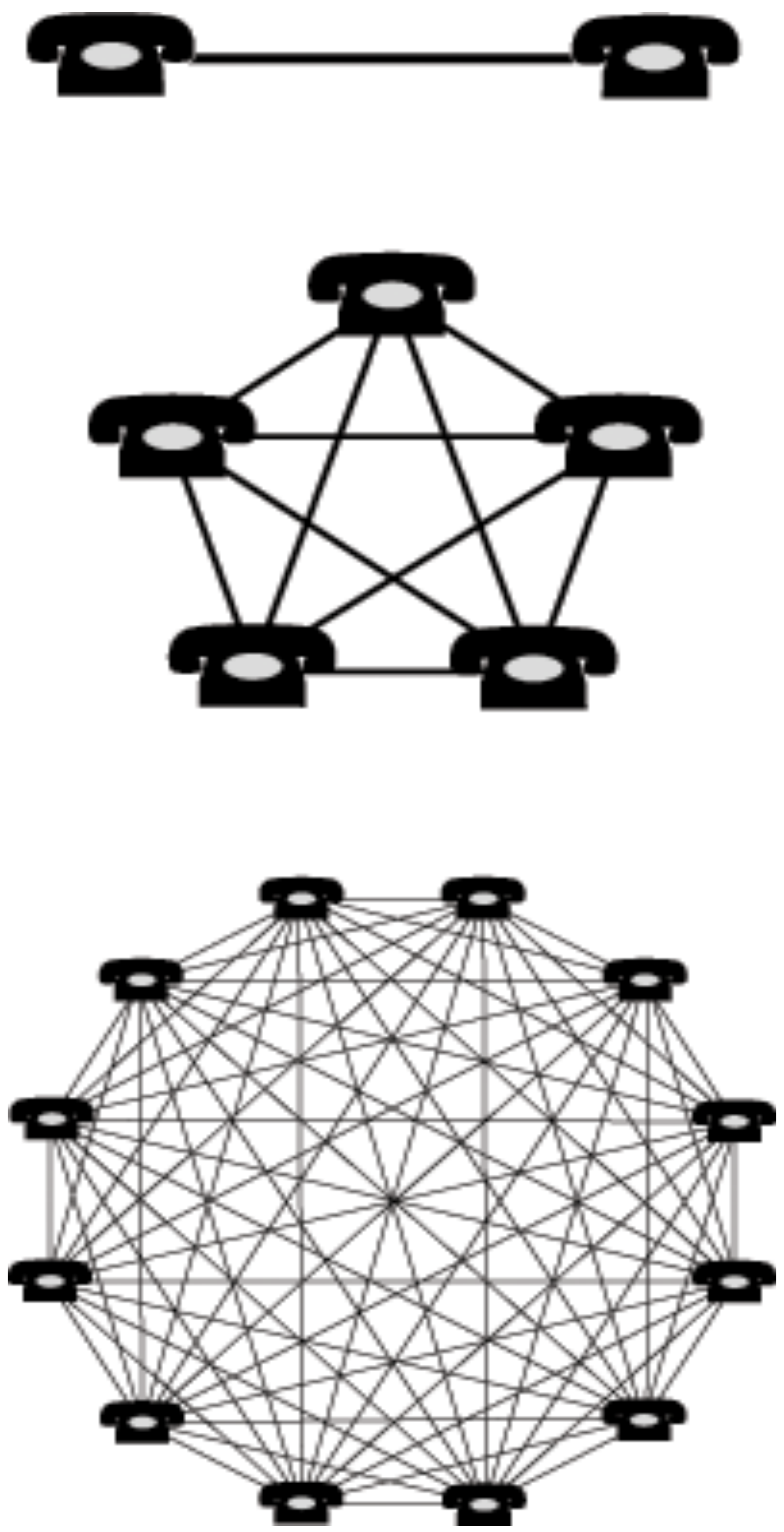
Those advantages have a cost

Global, atomic, consistent state means global coordination

Coordination does not scale linearly

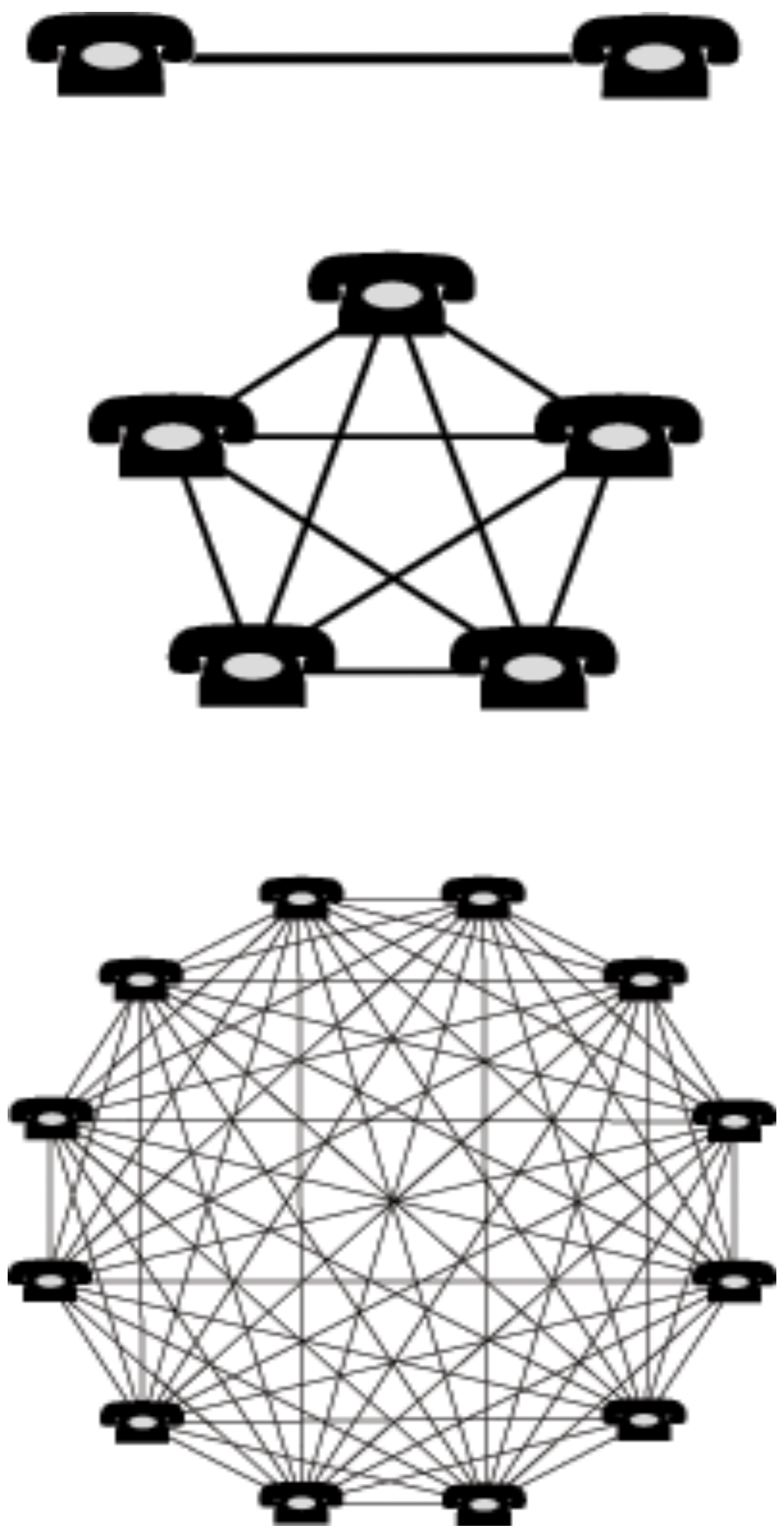
The costs of coordination

Remember the network effect?



The costs of coordination

$$channels = \frac{n(n-1)}{2}$$



2 nodes = 1 channel

5 nodes = 10 channels

12 nodes = 66 channels

25 nodes = 300 channels

The result is we don’t scale linearly as we add nodes

Independence $\Longrightarrow$ Parallelizable

Independence → **Parallelizable**

Parallelizable → **Scalable**

“Shared Nothing” architectures are the most scalable…

…but most real-world problems require us to share *something*…

…so our designs usually have a *parallel* part and a *serial* part

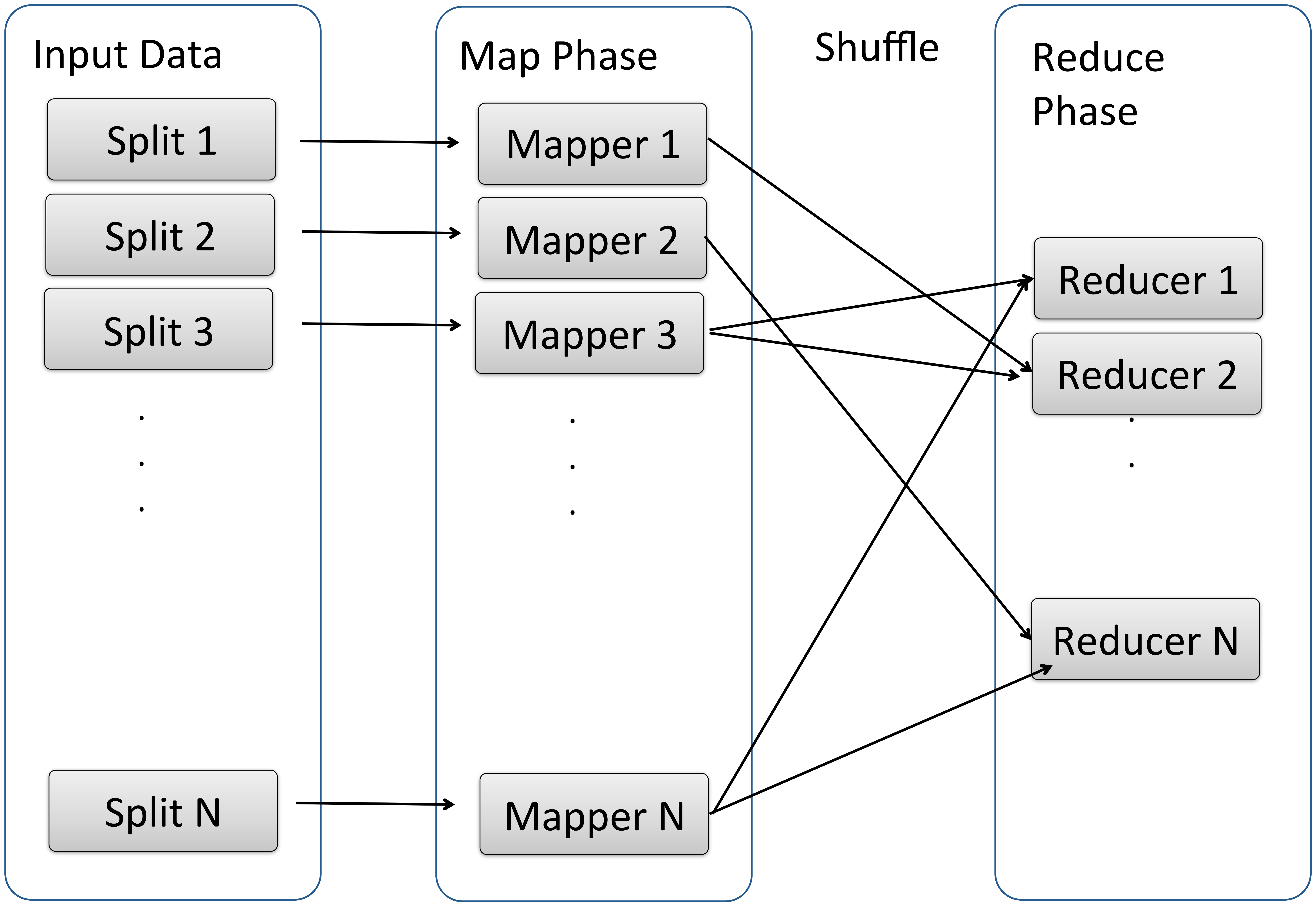
The key is to make sure the vast majority of our work in the cloud is ***independent*** and ***parallelizable***.

Amdahl’s Law

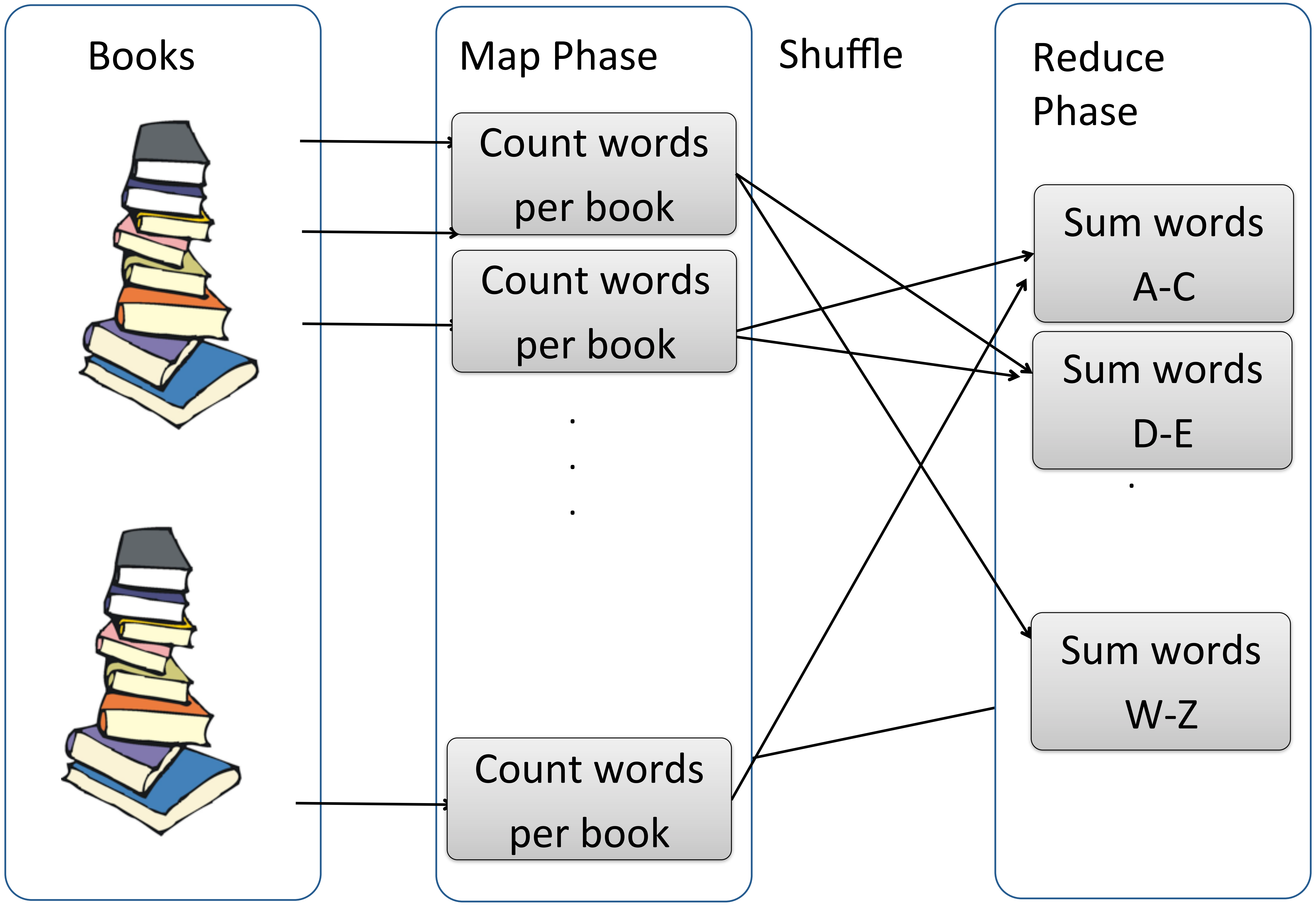
$$S(N) = \frac{1}{(1-P)+\frac{P}{N}}$$

S : speed improvement
P : ratio of the problem that
can be parallelized
N: number of processors

MapReduce Primer



MapReduce Example: Word Count

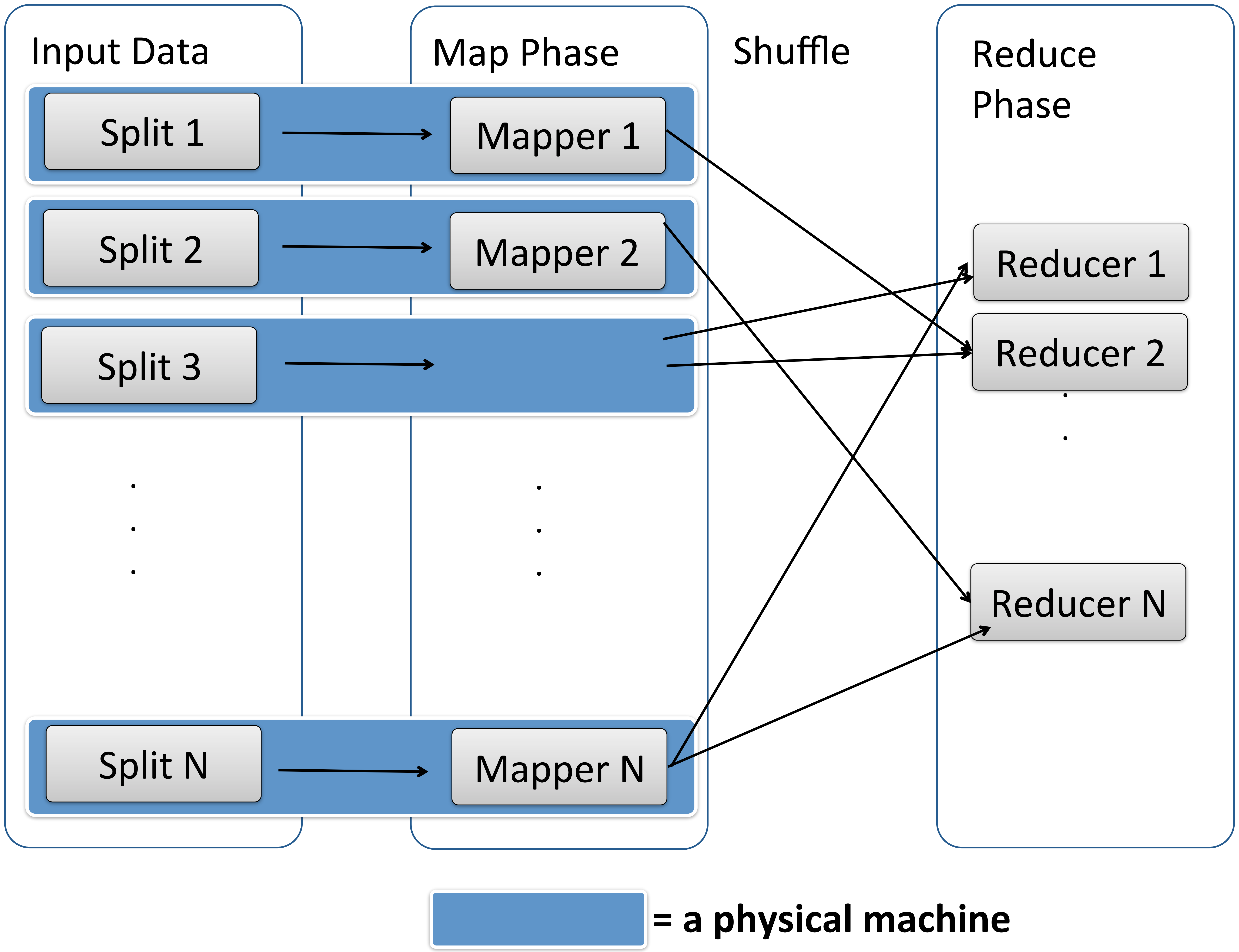


The network is a shared resource

Too much data to move to computation

So move *computation* to *data*

MapReduce Data Locality



Data locality only guaranteed in the Map phase

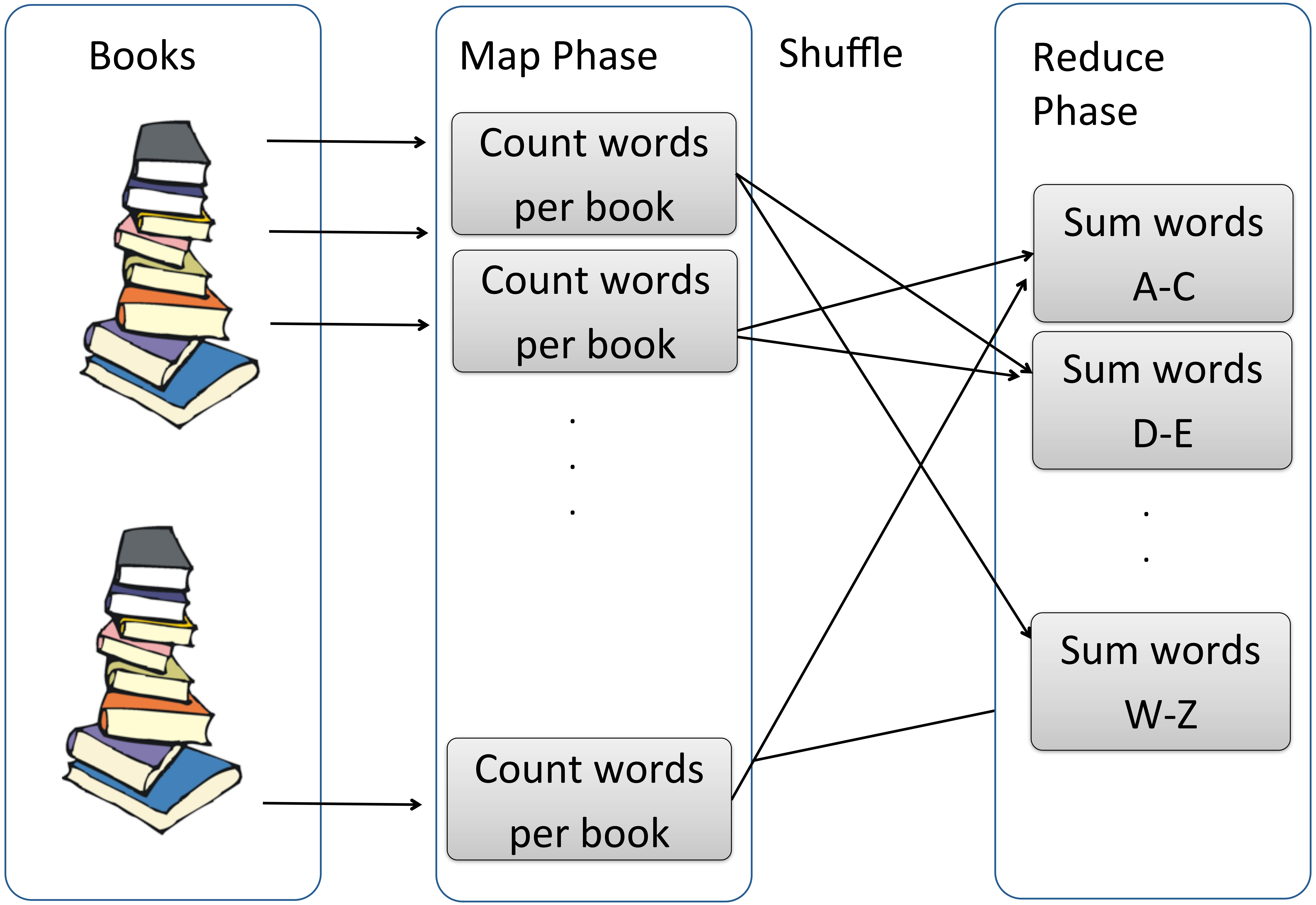
So do as much work as possible there

Some jobs have no reducer at all!

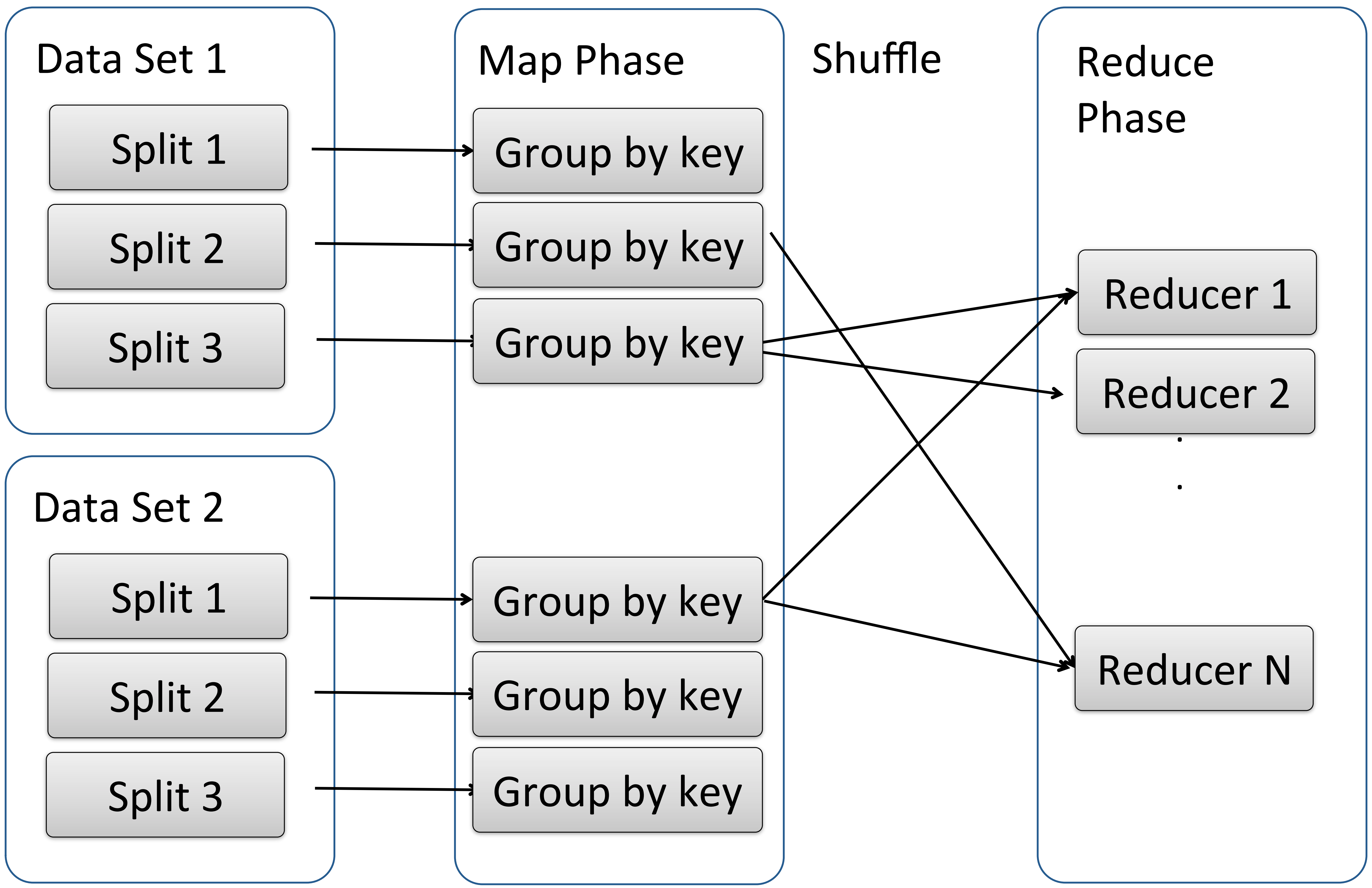
MapReduce is a building block

So let’s build higher-level functions

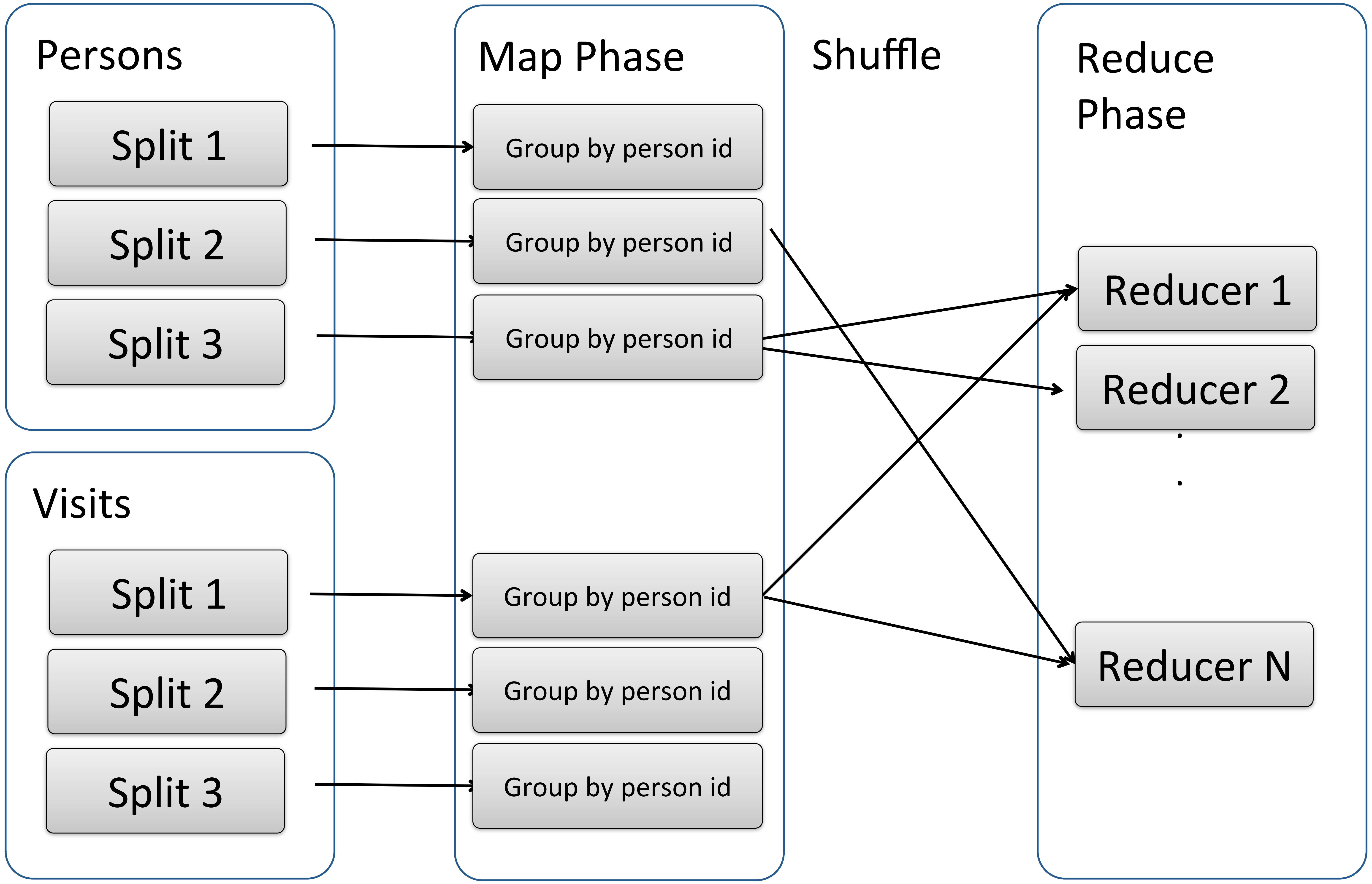
Grouping and Aggregating



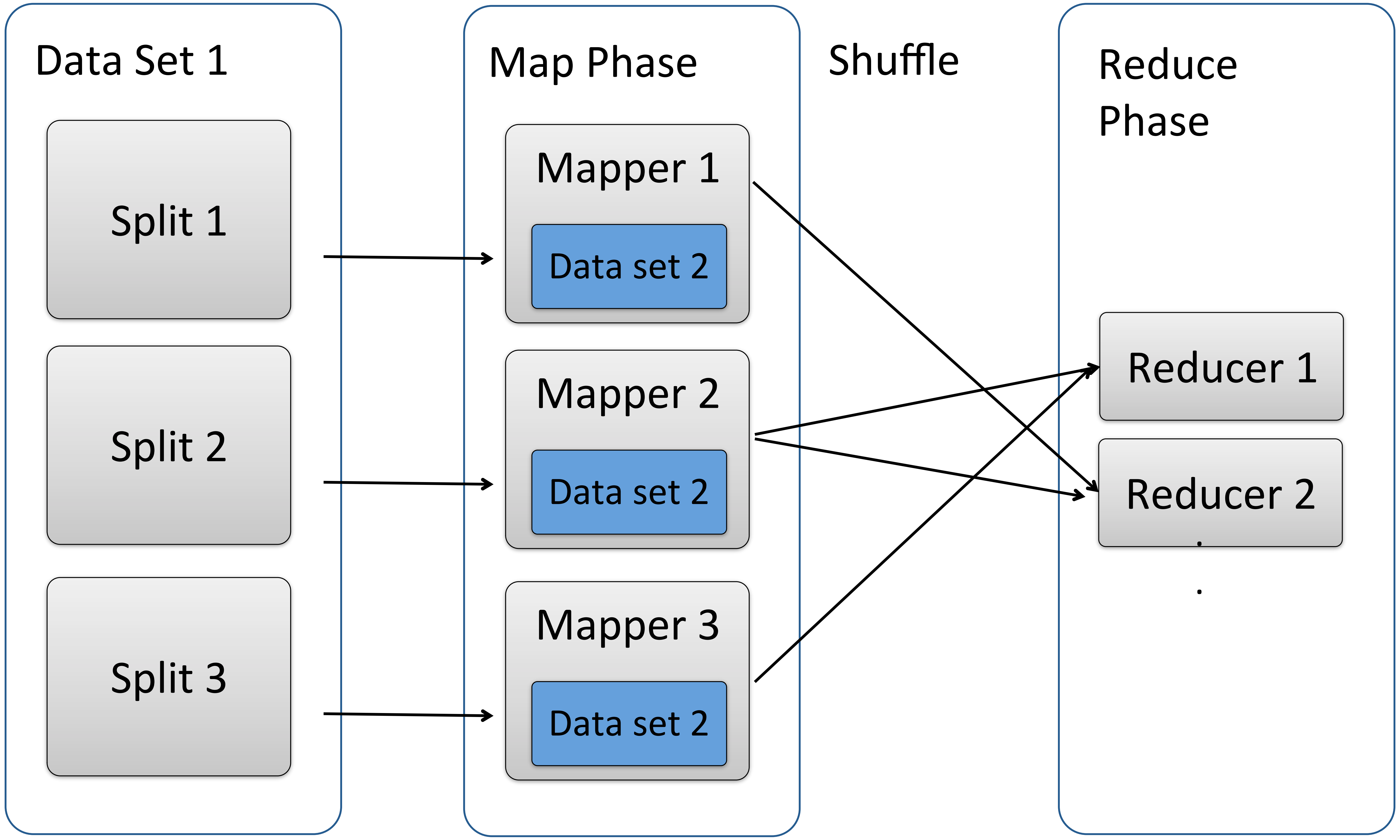
Joins


Data Set 1
Split 1
Split 2
Split 3
Data Set 2
Split 1
Split 2
Split 3
Map Phase
Group by key
Group by key
Group by key
Group by key
Group by key
Group by key
Shuffle
Reduce Phase
Reducer 1
Reducer 2
Reducer N

Joins


Persons
Split 1
Split 2
Split 3
Visits
Split 1
Split 2
Split 3
Map Phase
Group by person id
Group by person id
Group by person id
Group by person id
Group by person id
Group by person id
Shuffle
Reduce Phase
Reducer 1
Reducer 2
Reducer N

Map-Side Joins



Filtering

Map or reduce functions can simply discard data we’re not interested in

And Others

Distinct

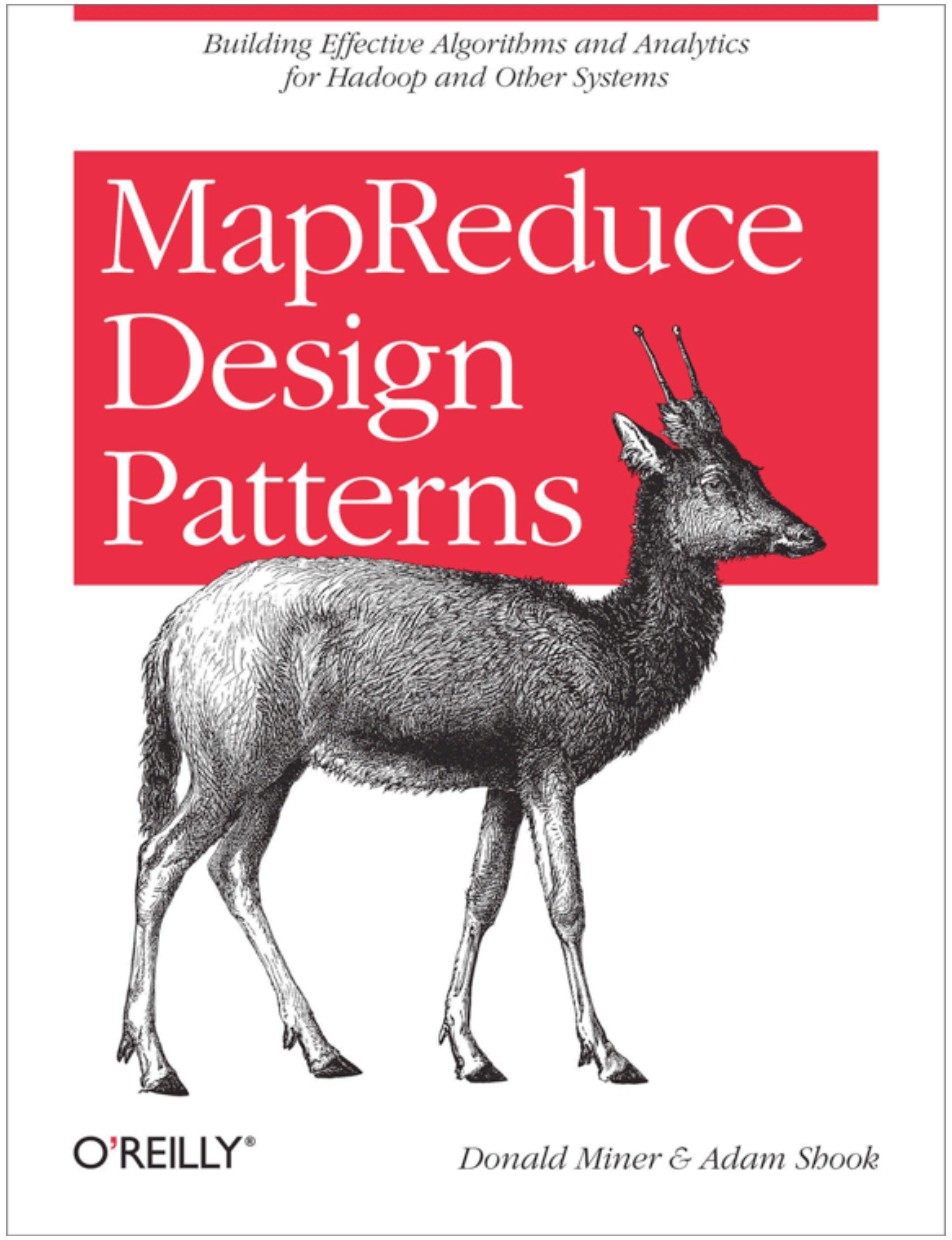
Sort

Binning

Top N

...

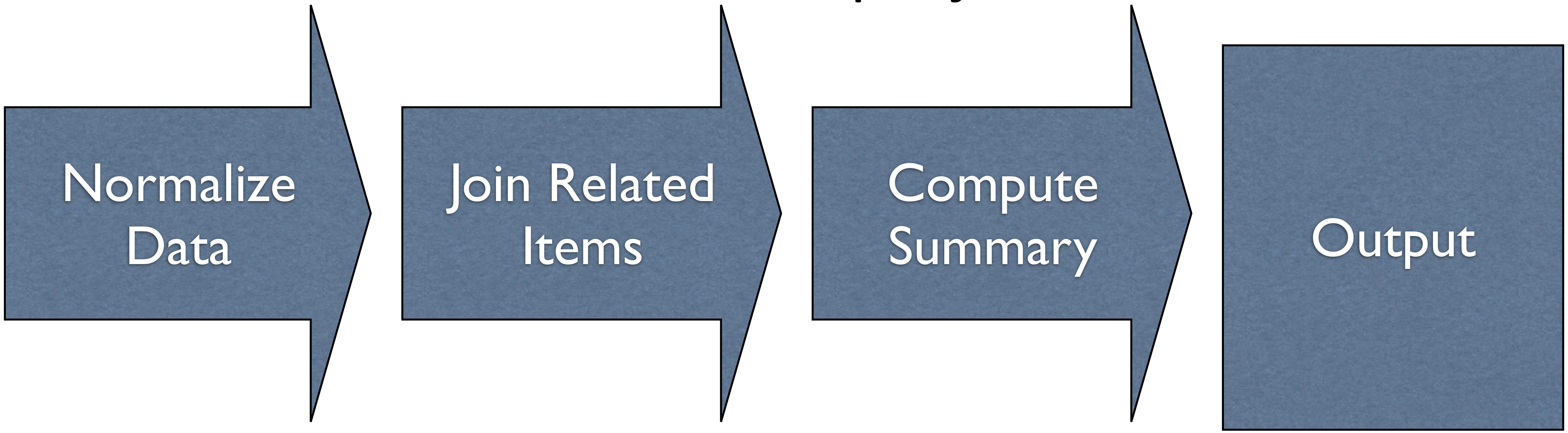
More sophisticated patterns composable



Chain Jobs Together

Large-scale joins must have a reduce phase

Multiple joins or group-by operations mean multiple jobs



Codified in High-Level Libraries

Hive, Pig, Cascading, and Crunch provide simple means to use these patterns

The era of writing MapReduce by hand is over

Apache Crunch

How do we use these tools?

Start with the question you want to ask, then transform the data to answer it.

output = transform (input)

Functional Programming over
Place-Oriented Programming

Work with data holistically

Re-running functions simpler to reason about than updating state

Hadoop makes this possible at scale

Don't be afraid to re-process the world

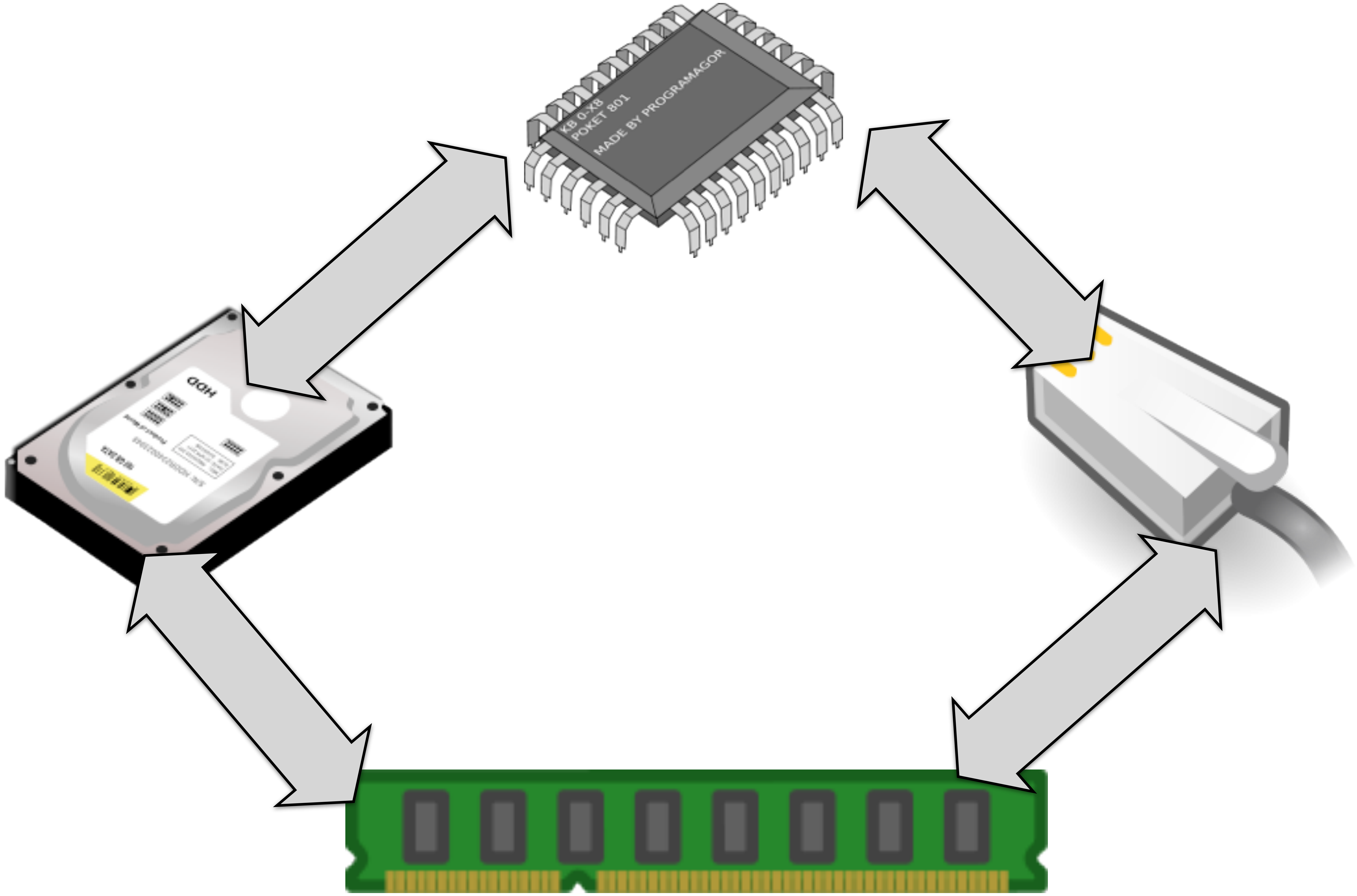
Something's wrong, we're above 95% usage!

-Traditional System Administrator

Something's wrong, we're below 95% usage!

-Hadoop System Administrator

Maximize Resource Usage



From Databases to Dataspaces

(Also referred to as Data Lakes)

Franklin, Halevy, Maier, http://homes.cs.washington.edu/~alon/files/dataspacesDec05.pdf

Bring all of your data together...

..structured or unstructured...

...transform it with unlimited computation...

...at any time for any new need.

Franklin, Halevy, Maier, http://homes.cs.washington.edu/~alon/files/dataspacesDec05.pdf

And offer a variety of interactive access patterns.

SQL, Search, Domain-Specific Apps

Hadoop is becoming an adaptive, multi-purpose platform.

The gap between asking novel questions
and our ability to answer them
is closing.

Questions?