Physics and Data Science: Reflections from a physicist in industry

Jim Pivarski

October 30, 2015

My trajectory

1999–2006: Graduate work on CLEO, a medium-sized collaboration, Ph.D. from Cornell.

2006–2011: Postgraduate work on CMS, a large collaboration, with Texas A&M University.

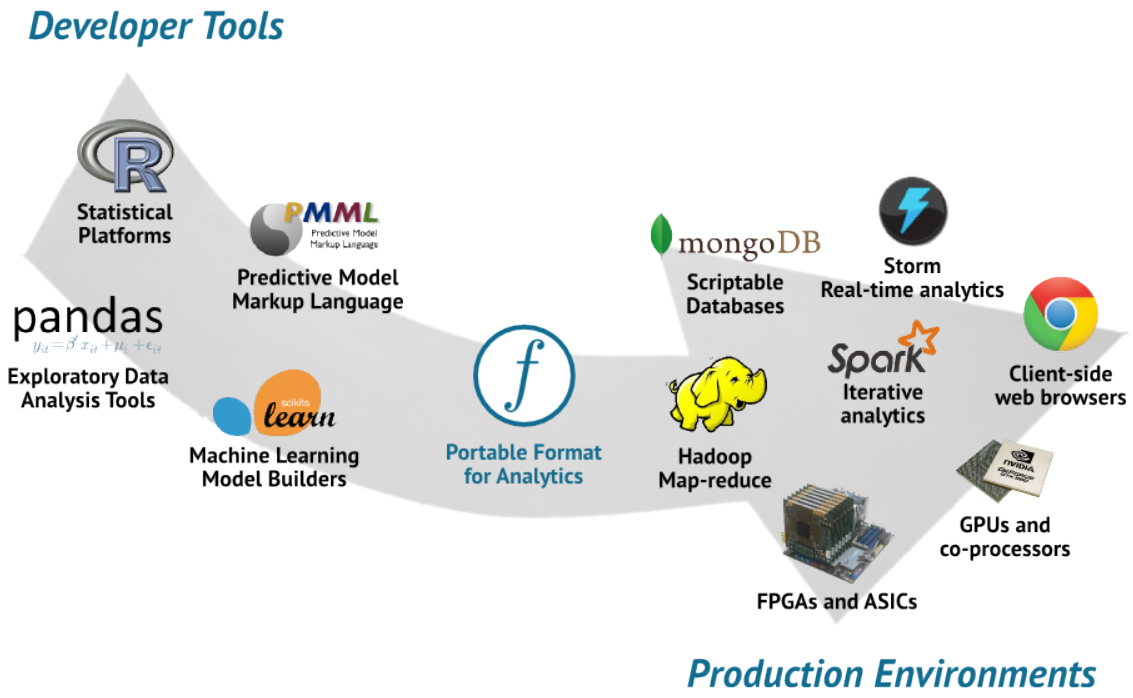
- Commissioning, mostly muon detector alignment.
- Search for leptonic jets in early LHC data.

2011–now: Data scientist for Open Data Group, a small consulting firm with clients in large companies.

- Analyzed data from hyperspectral satellite photos, automobile traffic, network traffic, web trends, Twitter sentiment, geolocation, Chicago crime, etc.
- Implemented data mining models with variations.
- Created the Portable Format for Analytics (PFA), a data mining model interchange format in JSON.

My trajectory

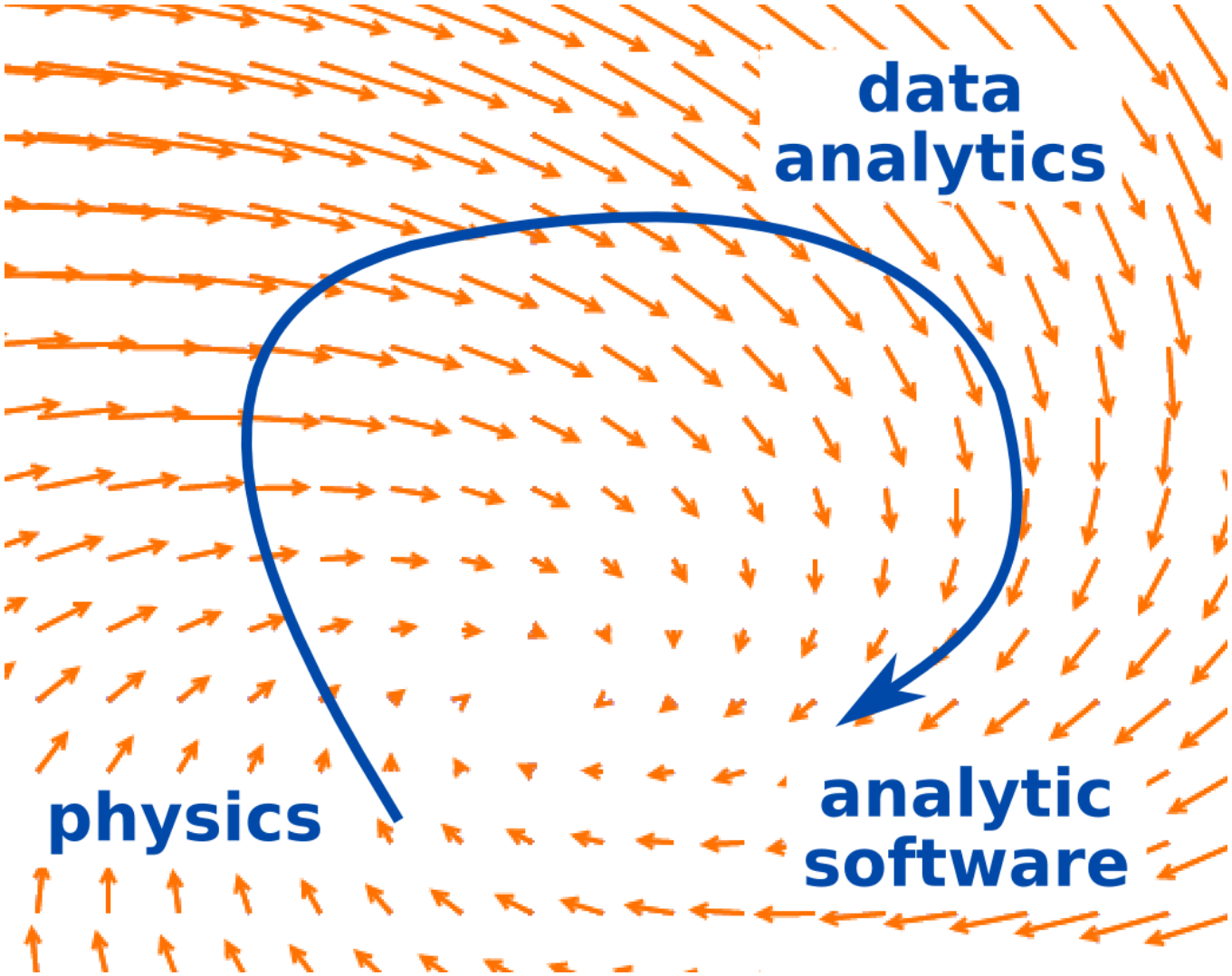
PFA has been my largest project: a successor to PMML (Predictive Model Markup Language), PFA simplifies the representation of statistical models and adds basic control constructs. Basically, a programming language in JSON with a large library of data mining primitives.



Recently adopted by the Data Mining Group (DMG), the organization that standardized and maintains PMML.

http://dmg.org

My trajectory



Outline

- What is data science?
- Statistical techniques that aren't often used in HEP
- Software tools: Hadoop, Spark, and all that
- Trends in analytic programming

What is data science?

What is data science?

Embarrassingly, most answers to this question come from Twitter...

- "A data scientist is a statistician who lives in San Francisco."
- "A data scientist is better at statistics than a software engineer and better at programming than a statistician."
- "A data scientist is a business analyst with a Mac."

What is data science?

Embarrassingly, most answers to this question come from Twitter...

- "A data scientist is a statistician who lives in San Francisco."
- "A data scientist is better at statistics than a software engineer and better at programming than a statistician."
- "A data scientist is a business analyst with a Mac."

Data science is the application of statistical techniques to business decisions with an emphasis on software development and large-scale deployment (a.k.a. "big data").

What is data science?

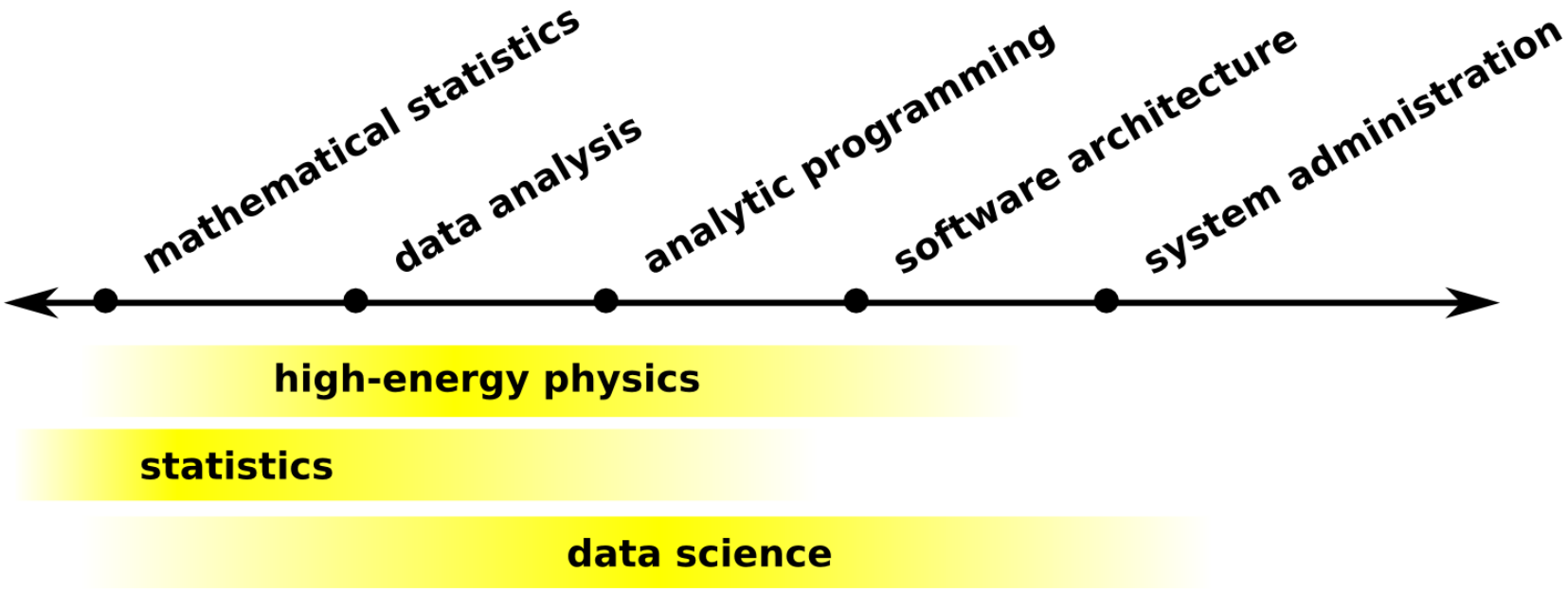
Embarrassingly, most answers to this question come from Twitter...

- "A data scientist is a statistician who lives in San Francisco."
- "A data scientist is better at statistics than a software engineer and better at programming than a statistician."
- "A data scientist is a business analyst with a Mac."

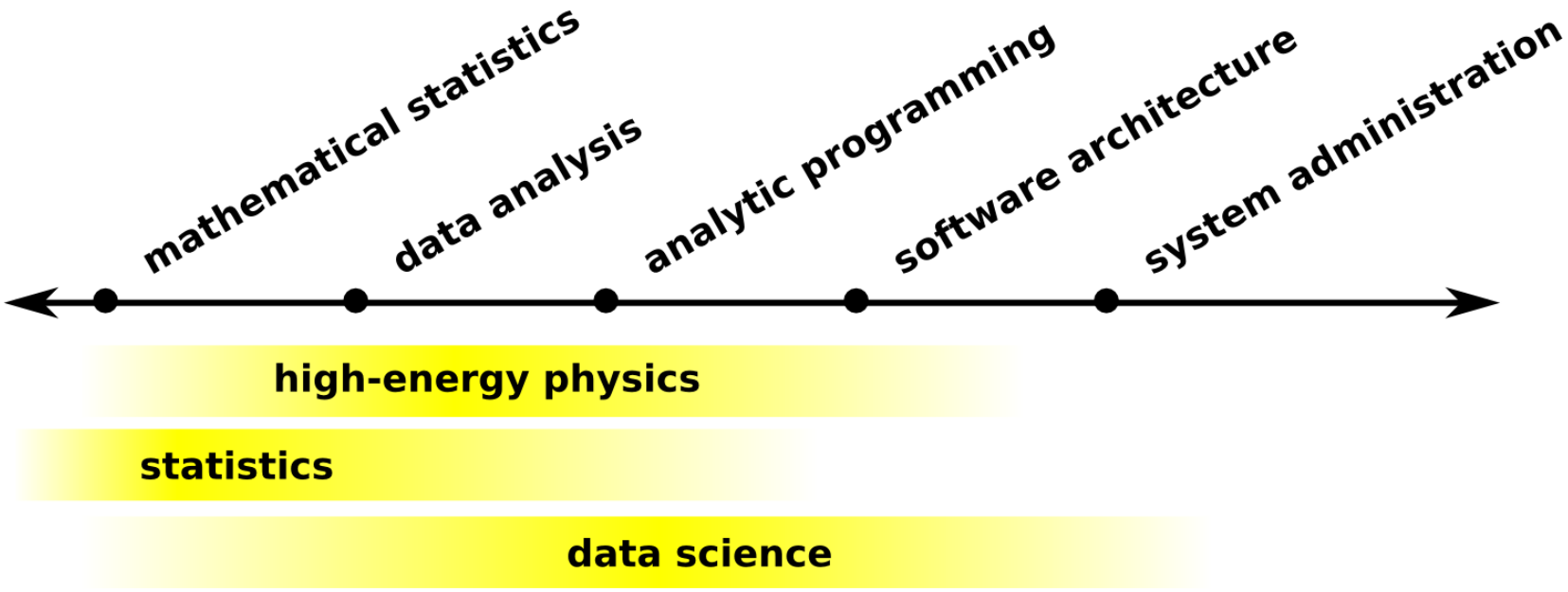
Data science is the application of statistical techniques to business decisions with an emphasis on software development and large-scale deployment (a.k.a. "big data").

In other words, what high-energy physicists do, but applied to business decisions.

What is data science?



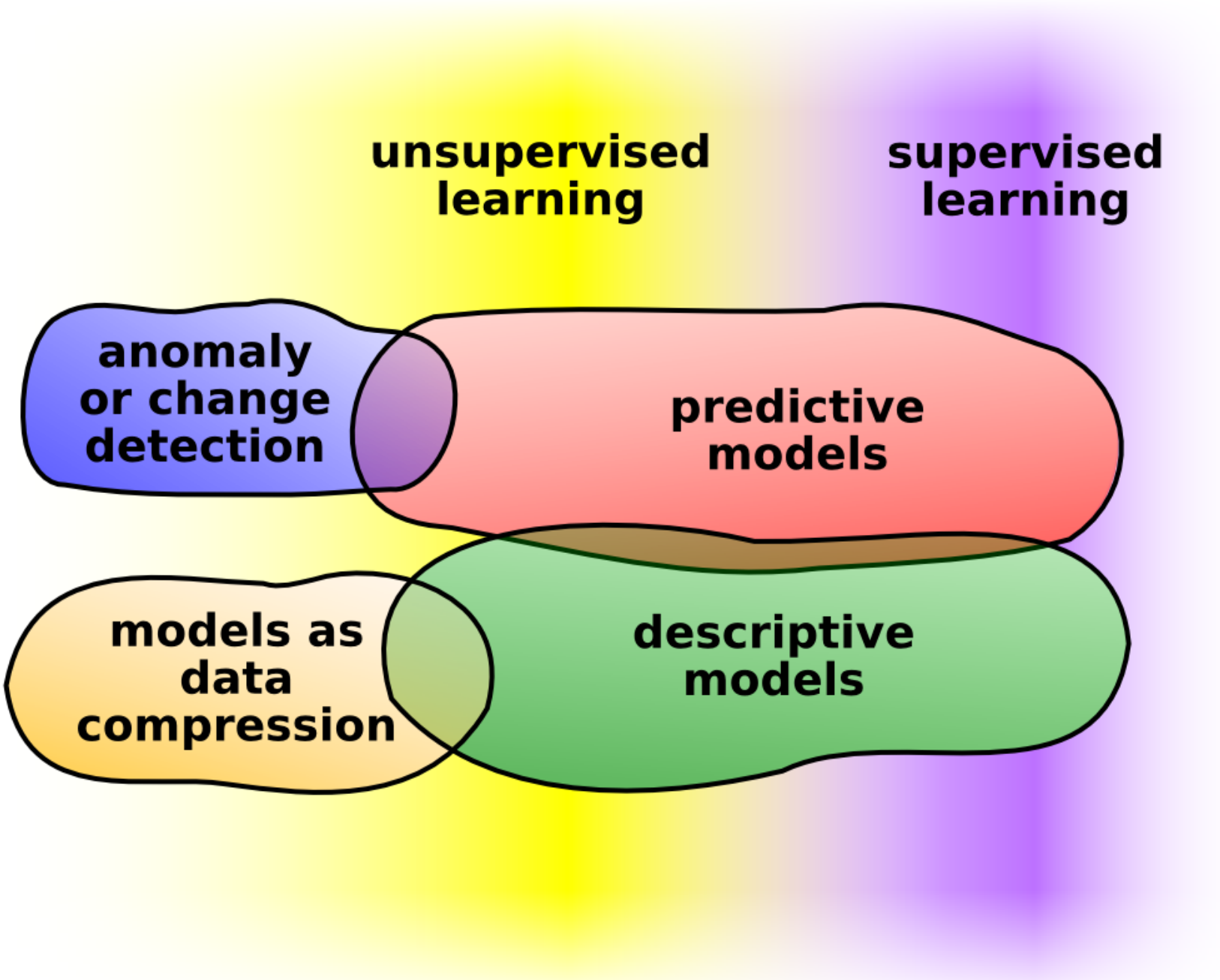
What is data science?



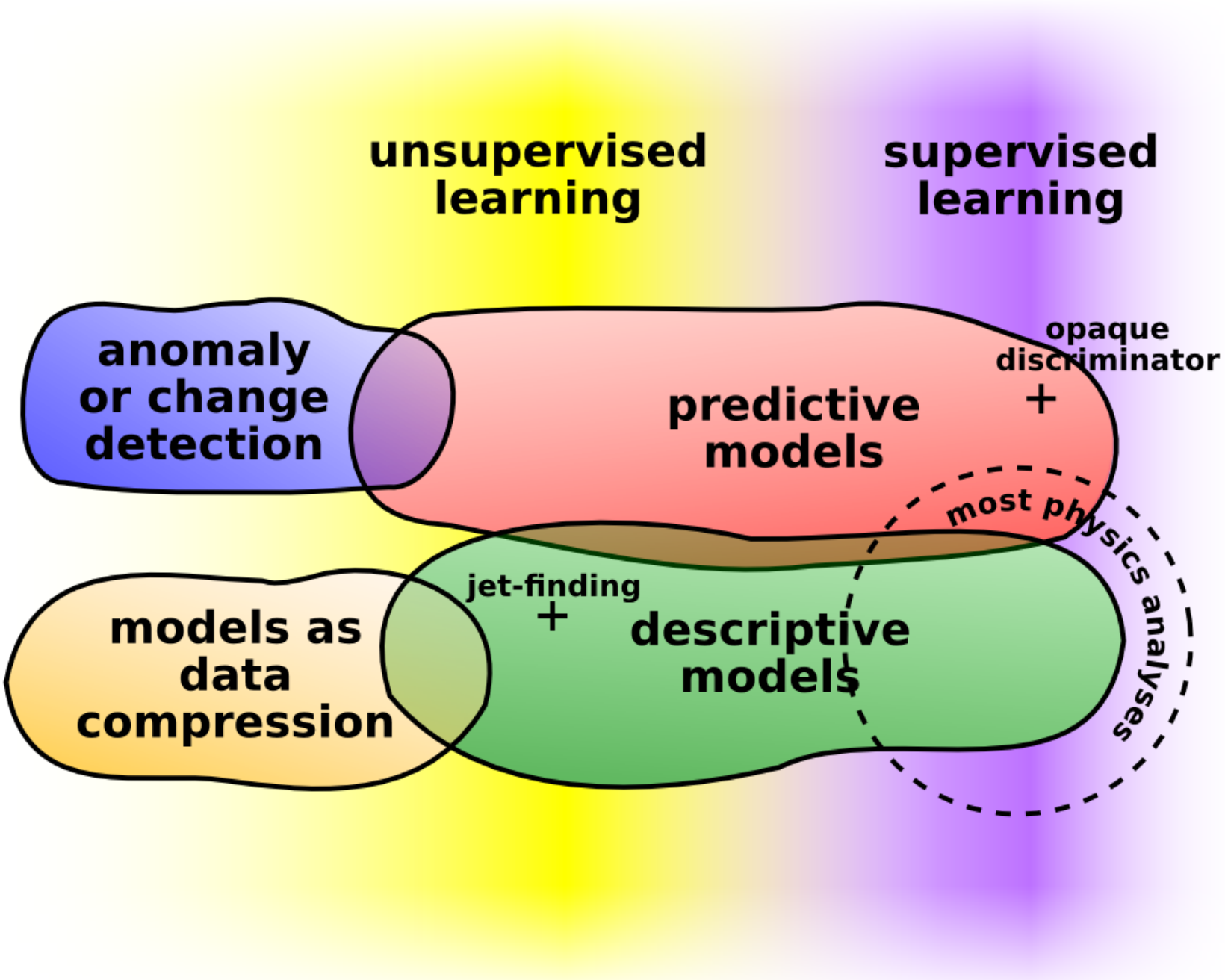
Because of the overlap, many data scientists come from physics, but knowledge can flow the other way, too.

Statistical techniques

Fundamentals: what is a model?



Fundamentals: what is a model?



Fundamentals: what is a model?

Descriptive: Model is the final product, revealing an underlying relationship in the data: e.g. the Standard Model.

Predictive: Model is a means to performing predictions and is judged on the quality of those predictions: e.g. stock predictions.

Fundamentals: what is a model?

Descriptive: Model is the final product, revealing an underlying relationship in the data: e.g. the Standard Model.

Predictive: Model is a means to performing predictions and is judged on the quality of those predictions: e.g. stock predictions.

Supervised: Training procedure accentuates differences between two labeled datasets: e.g. signal and background Monte Carlo.

Unsupervised: Training procedure finds patterns on its own: e.g. groups of nearby particles are a jet.

Fundamentals: what is a model?

Descriptive: Model is the final product, revealing an underlying relationship in the data: e.g. the Standard Model.

Predictive: Model is a means to performing predictions and is judged on the quality of those predictions: e.g. stock predictions.

Supervised: Training procedure accentuates differences between two labeled datasets: e.g. signal and background Monte Carlo.

Unsupervised: Training procedure finds patterns on its own: e.g. groups of nearby particles are a jet.

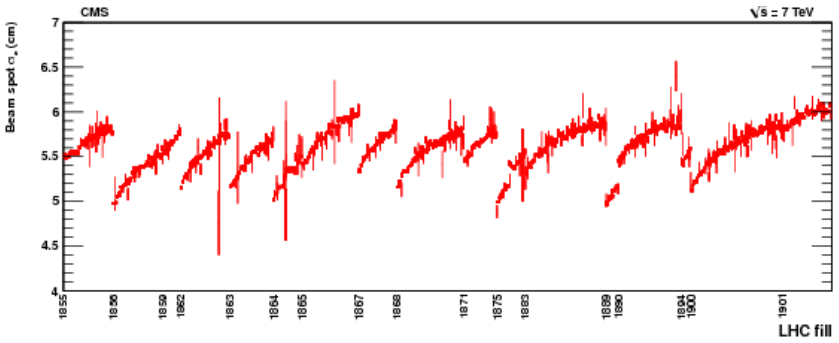
Anomaly detection: Model describes the normal situation to identify unusual cases or changes: e.g. data quality monitor.

Data compression: Model summarizes data for the sake of describing it with fewer bits: e.g. image compression.

Potential applications in physics

Some quantities, such as the LHC beamspot, are stable for long periods of time but might jump at discrete times.

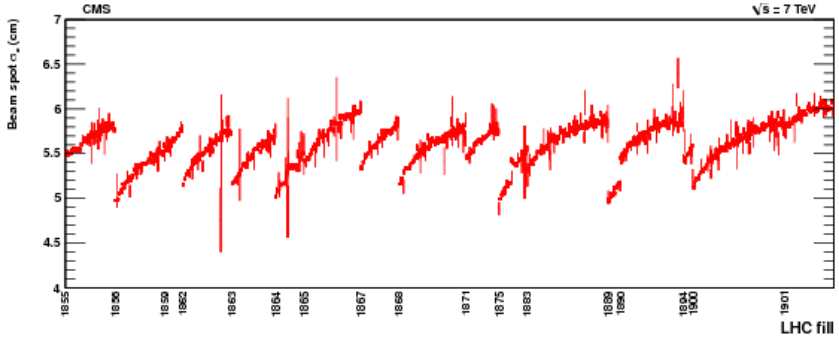
Want to identify jumps and combine statistics in the stable periods.



Potential applications in physics

Some quantities, such as the LHC beamspot, are stable for long periods of time but might jump at discrete times.

Want to identify jumps and combine statistics in the stable periods.



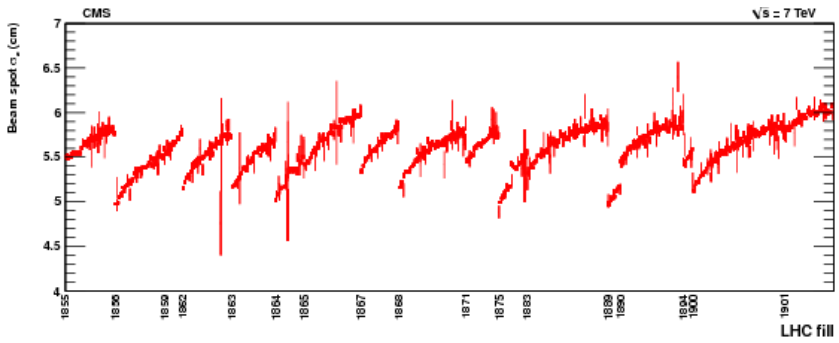
- Generalized Likelihood Ratio (GLR) is a change detection technique that finds the time when one distribution jumps to another using a maximum likelihood scan.

$$\underbrace{X_1, X_2, \ldots, X_{\lambda-1}}_{f_0 \in \mathcal{F}_0} \nearrow \overbrace{X_\lambda, X_{\lambda+1}, \ldots}^{f_1 \in \mathcal{F}_1}$$

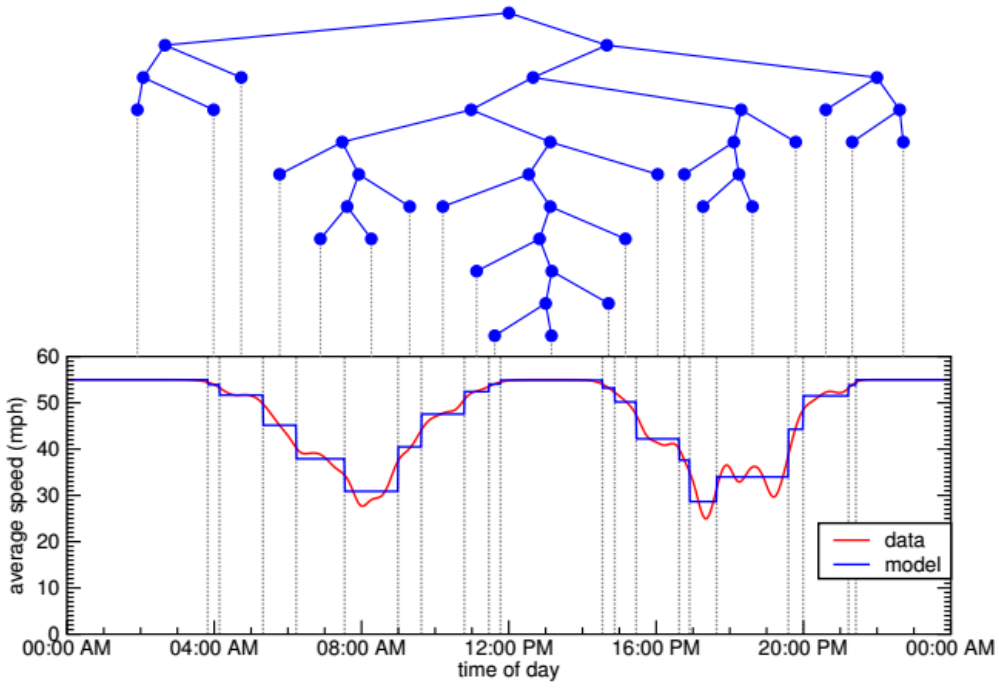
Potential applications in physics

Some quantities, such as the LHC beamspot, are stable for long periods of time but might jump at discrete times.

Want to identify jumps and combine statistics in the stable periods.



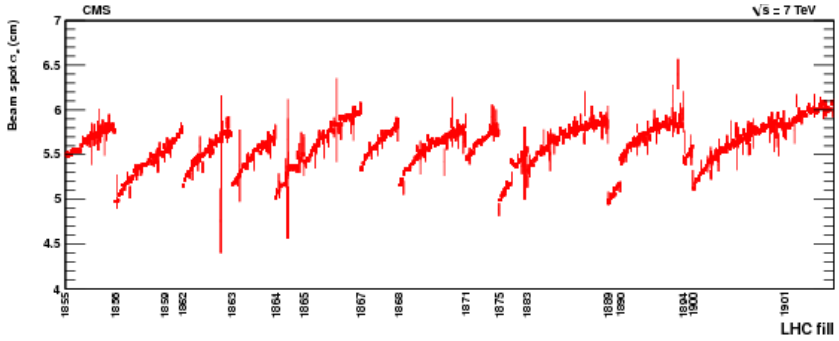
- Regression trees can use time as a regressor to partition the sequence into periods of minimal variance (or minimal linear-fit χ^2, etc.).



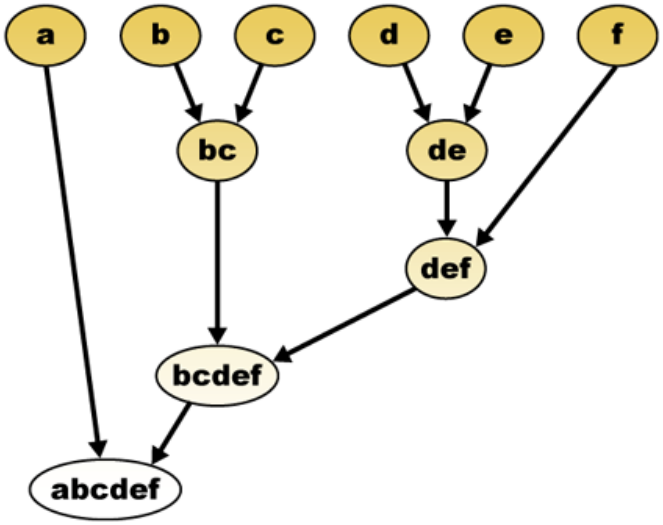
Potential applications in physics

Some quantities, such as the LHC beamspot, are stable for long periods of time but might jump at discrete times.

Want to identify jumps and combine statistics in the stable periods.



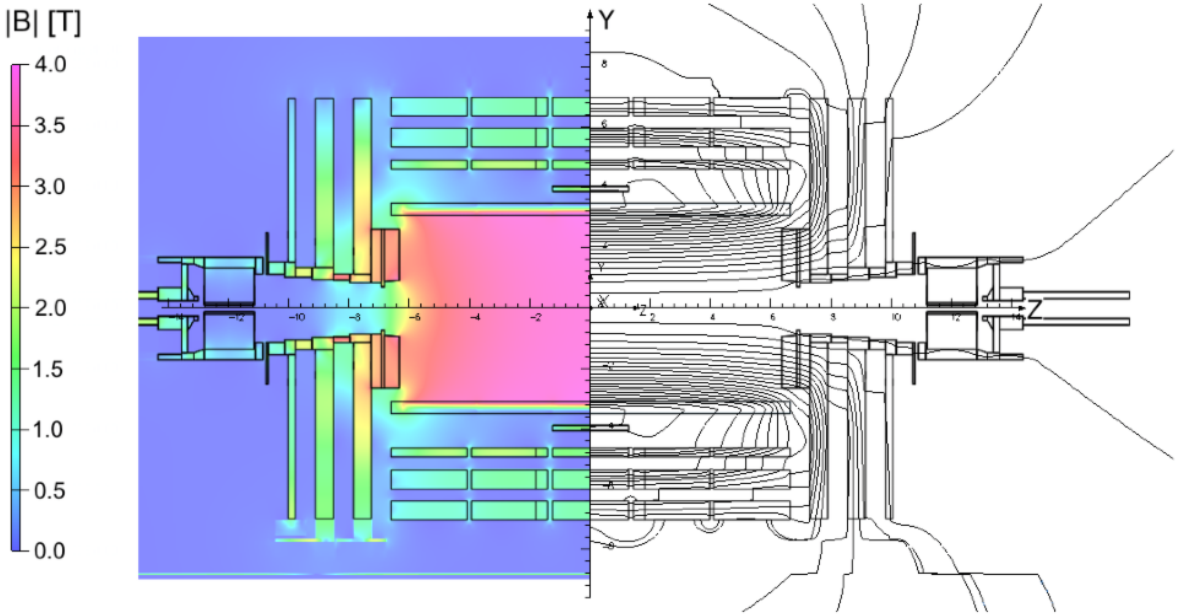
- Hierarchical clustering, especially single-linkage, groups chains of gradually changing parameters and splits at the biggest jumps.

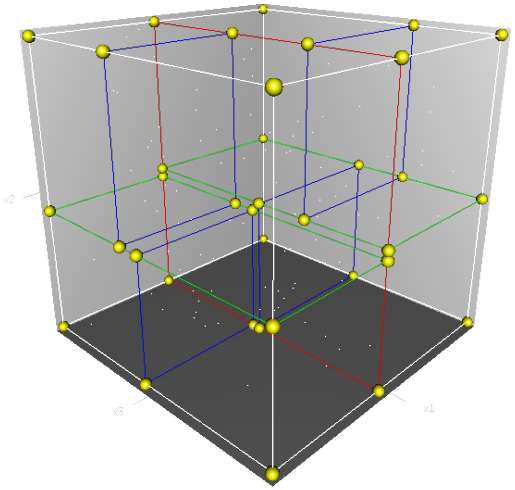


Potential applications in physics

Some quantities, such as the CMS magnetic field, have been simulated in high detail, but a simplified model must be used in tracking because of memory constraints.

One could again use a regression tree, this time with r, ϕ, z as regressors, rather than time, to split the space by importance.





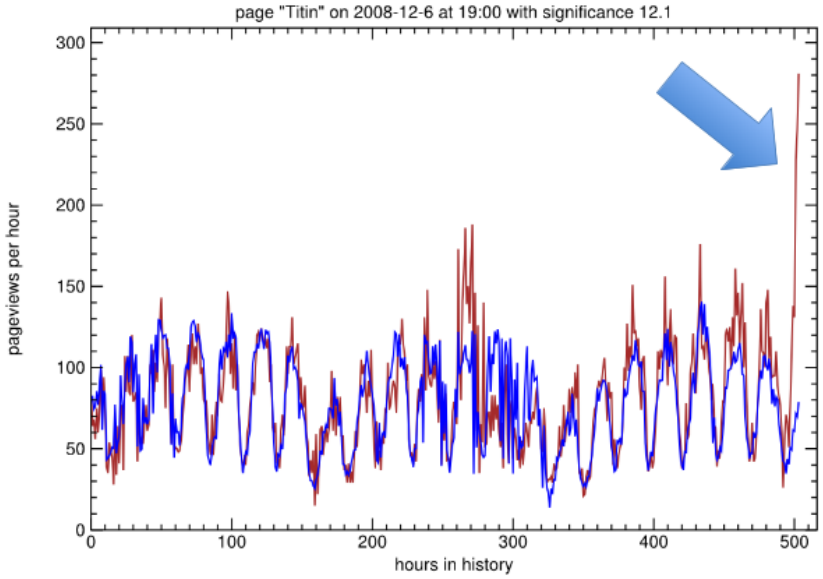
This is an example of using a model as compression (not even "data").

Potential applications in physics

Physics is not the only field in which data quality must be monitored by human operators.

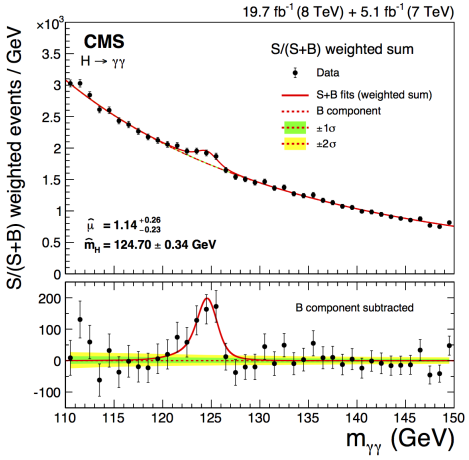
Projects with large numbers of distributions to watch use anomaly detection models as the first pass.

- CUSUM (cumulative sum) models catch slowly drifting trends, in which the accumulation of recent past is more significant than any one outlier.
- Holt-Winters is an extension of Exponentially Weighted Moving Average (EWMA) that can absorb complex behaviors, such as cycles, into the "normal" distribution.



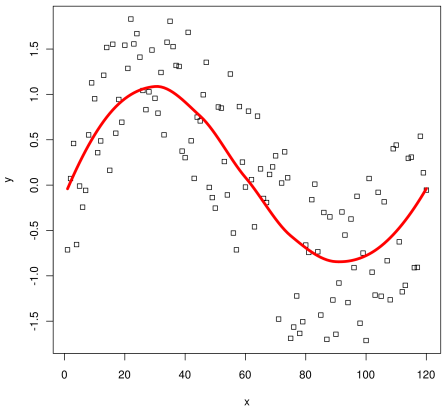
Potential applications in physics

Non-linear regression is probably HEP's favorite model, but in some cases, one does not want to assume an ansatz (e.g. background combinatorics).



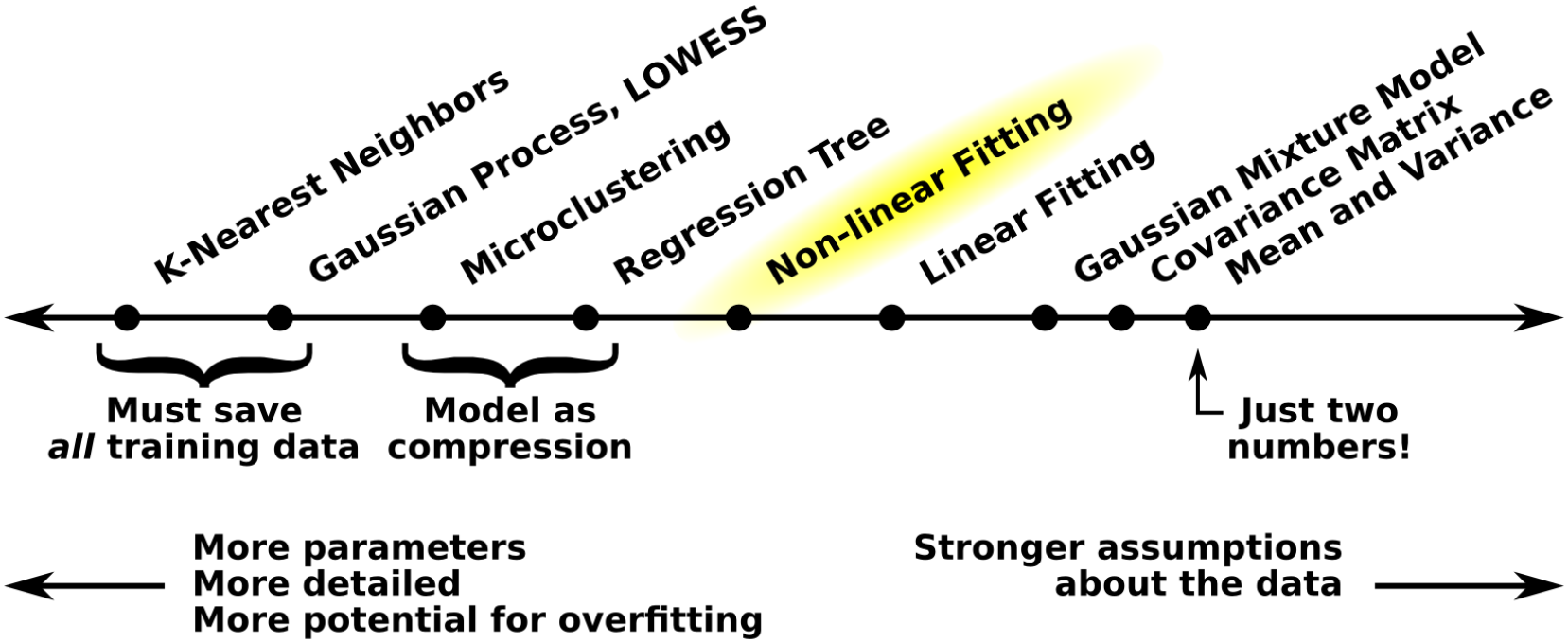
Non-parametric fitting techniques:

- K-Nearest Neighbors (KNN): make a curve by averaging the k nearest points from the training dataset. Alternatively, average all points within a ball of specified radius.
- Locally Weighted Scatterplot Smoothing (LOWESS): make a curve by linearly fitting the training dataset $\{x_i\}$ at each x_p with weight $w_i = e^{-(x_i - x_p)^2/2/\sigma}$.



- Gaussian Process: make a curve by assuming nearby points to be more correlated than distant points.

Spectrum of regression



Software tools

Big Data

For decades, HEP had the biggest datasets anywhere.

In the past 10–15 years, internet-related datasets have reached TB and PB scales: advertisement click-throughs, shopping correlations, web search engines, cybersecurity, Facebook pictures, YouTube videos, etc.

Big Data

For decades, HEP had the biggest datasets anywhere.

In the past 10–15 years, internet-related datasets have reached TB and PB scales: advertisement click-throughs, shopping correlations, web search engines, cybersecurity, Facebook pictures, YouTube videos, etc.

But analyses of these datasets are also *tightly coupled.* For example,

- A customer buys diapers online; what other products should you recommend?
- Analysis of diaper purchases across all customers reveals a correlation with milk; recommend milk.

The dataset, originally indexed by customer, must be re-indexed by product (or even by sets of products) to make a recommendation.

Big Data

For decades, HEP had the biggest datasets anywhere.

In the past 10–15 years, internet-related datasets have reached TB and PB scales: advertisement click-throughs, shopping correlations, web search engines, cybersecurity, Facebook pictures, YouTube videos, etc.

But analyses of these datasets are also *tightly coupled.* For example,

- A customer buys diapers online; what other products should you recommend?
- Analysis of diaper purchases across all customers reveals a correlation with milk; recommend milk.

The dataset, originally indexed by customer, must be re-indexed by product (or even by sets of products) to make a recommendation.

By contrast, most HEP analyses can process events independently.

Big Data

Big Data (my definition): a dataset that is too large to be processed on one computer; must resort to a distributed calculation.

Distributed calculations are qualitatively different from in-memory calculations.

They get increasingly difficult as the problems get more tightly coupled, since data must be shuffled among processors.

	diapers	milk	beer	
customer 1	☑	☑	☐	computer 1
customer 2	☑	☐	☑	
customer 3	☑	☐	☐	
customer 4	☐	☑	☑	computer 2
customer 5	☐	☑	☐	
customer 6	☑	☑	☐	
customer 7	☑	☐	☐	computer 3
customer 8	☐	☑	☑	
customer 9	☐	☑	☑	

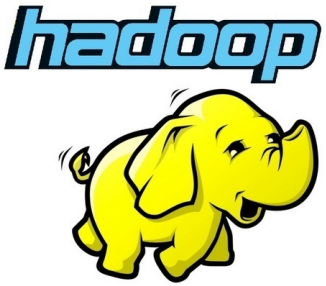
Google had an re-indexing problem: a set of webpages containing words had to be re-indexed as a set of words pointing to webpages, so that you can search for pages by keyword.

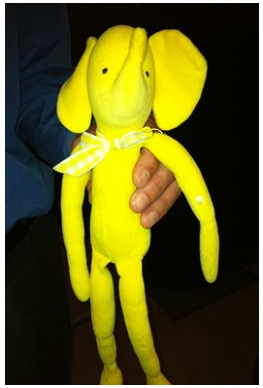
Big Data

Google had an re-indexing problem: a set of webpages containing words had to be re-indexed as a set of words pointing to webpages, so that you can search for pages by keyword.

Their solution, called "map-reduce," was published as a white paper in 2004.

It was immediately reimplemented as an open source product, Apache Hadoop.





Hadoop is now almost synonymous with Big Data, and it has spawned an ecosystem of tools that interoperate with it, much like ROOT in HEP.

Map-reduce

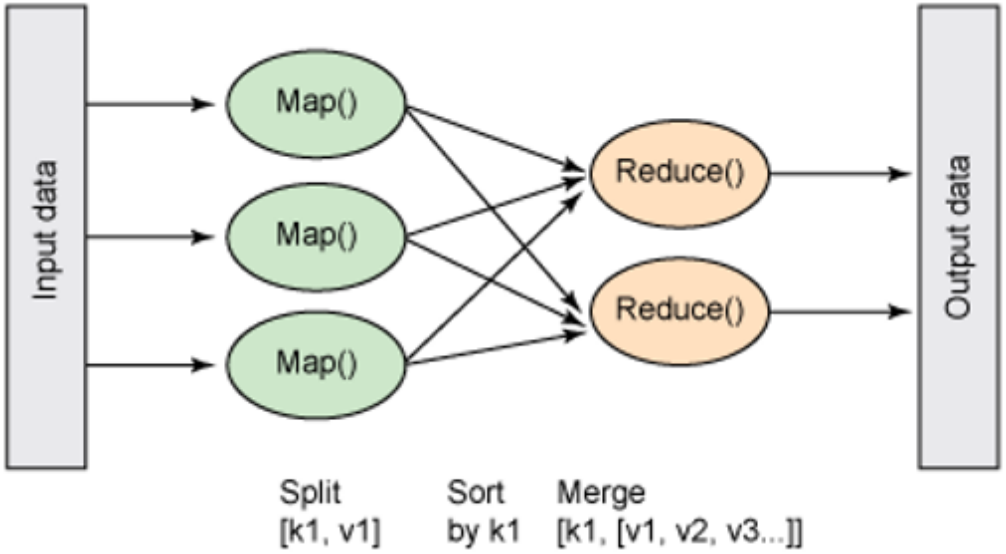
Hadoop executes two sets of independent, identical processors:

- Mappers, which transform each input to a ⟨key, value⟩ pair.
- Reducers, each operates on all values that have a given key.

```
def mapper(webpage):
  for word in webpage.split():
    yield (word, webpage)
```

```
def reducer(word, webpages):
  searchIndex[word] = {}
  for webpage in webpages:
    searchIndex[word].add(webpage)
```

The system groups data by key in an optimized way (independent partial sorts followed by merge, minimizing network bandwidth).



Potential application in physics

Most physics analyses apply the same function to all events independently. However, alignment and calibration are more tightly coupled:

- Alignment residuals must be re-indexed from tracks to subdetectors.
- Calibration responses must be re-indexed from π^0s to subdetectors.

```
def mapper(event):
    for track in event:
        for hit in track:
            key = hit.subdetector()
            residual = track.projection(hit) - hit.pos()
            yield (key, residual)
```

```
def reducer(subdetector, residuals):
    numer = 0
    denom = 0
    for residual in residuals:
        numer = numer + residual
        denom = denom + 1
    move(subdetector, numer/denom)  # shift by residual mean
```

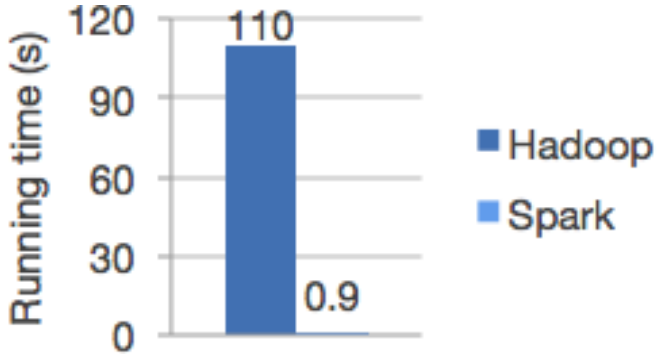
Map-reduce can be applied to *one iteration* of alignment or calibration.

Iterative map-reduce

Hadoop was not designed for iterative map-reduce.

Apache Spark is a framework that runs within a Hadoop cluster to speed up iteration.

It provides more control over the workflow topology (not just `map` and `reduce`) and allows the user to cache some datasets in RAM memory, so that they don't have to be re-loaded from disk.



Disk access is the biggest bottleneck for calculations of this type.

Alignment example in Spark


```
@case class Ratio(numer, denom):
    def add(self, other):
        return Ratio(self.numer + other.numer,
                     self.denom + other.denom)
    def value(self):
        return self.numer / self.denom

def makePair(track, hit):
    key = hit.subdetector()
    residual = track.projection(hit) - hit.pos()
    return (key, Ratio(residual, 1))
```

```
tracks.cache()   # tells Spark to keep tracks in memory

for iteration in range(100):
    keyValuePairs =
        tracks.flatMap(lambda track: track.hits.map(
                           lambda hit: makePair(track, hit)))

    corrections =
        keyValuePairs.reduceByKey(lambda r1, r2: r1.add(r2))
                     .mapValues(lambda r: r.value())

    corrections.foreach(move)
```

Streaming calculations

Another common problem in industry: how to perform distributed calculations in real-time?

- Example: update shopping correlations while customers are shopping, so that recommendations are based on purchases made in the last few minutes.

Hadoop and Spark are batch processors: they work on one big dataset.

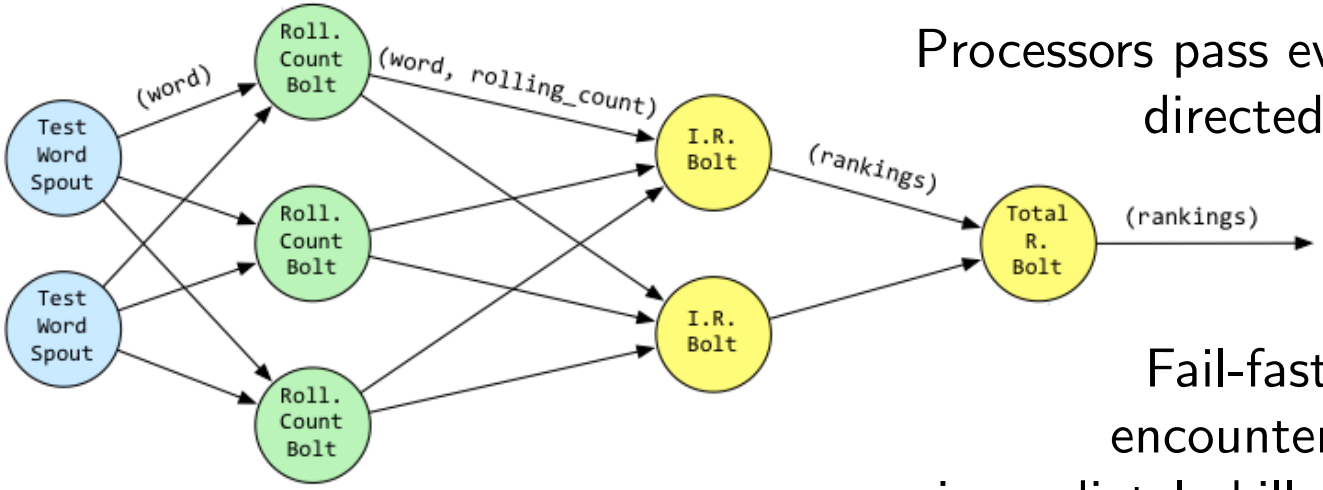
Streaming calculations

Another common problem in industry: how to perform distributed calculations in real-time?

- Example: update shopping correlations while customers are shopping, so that recommendations are based on purchases made in the last few minutes.

Hadoop and Spark are batch processors: they work on one big dataset.

Storm (and others, including "Spark-Streaming") are streaming processors: data flow through continuously.



Processors pass events through a directed, acyclic graph.

Fail-fast: if a processor encounters an error, it is immediately killed and replaced.

Streaming calculations

Streaming processors can die and drop events, or the wrong version of code might be deployed online, so they are less trustworthy than the Nth revision of a batch calculation.

However, batch calculations are not up-to-date. Want best of both...

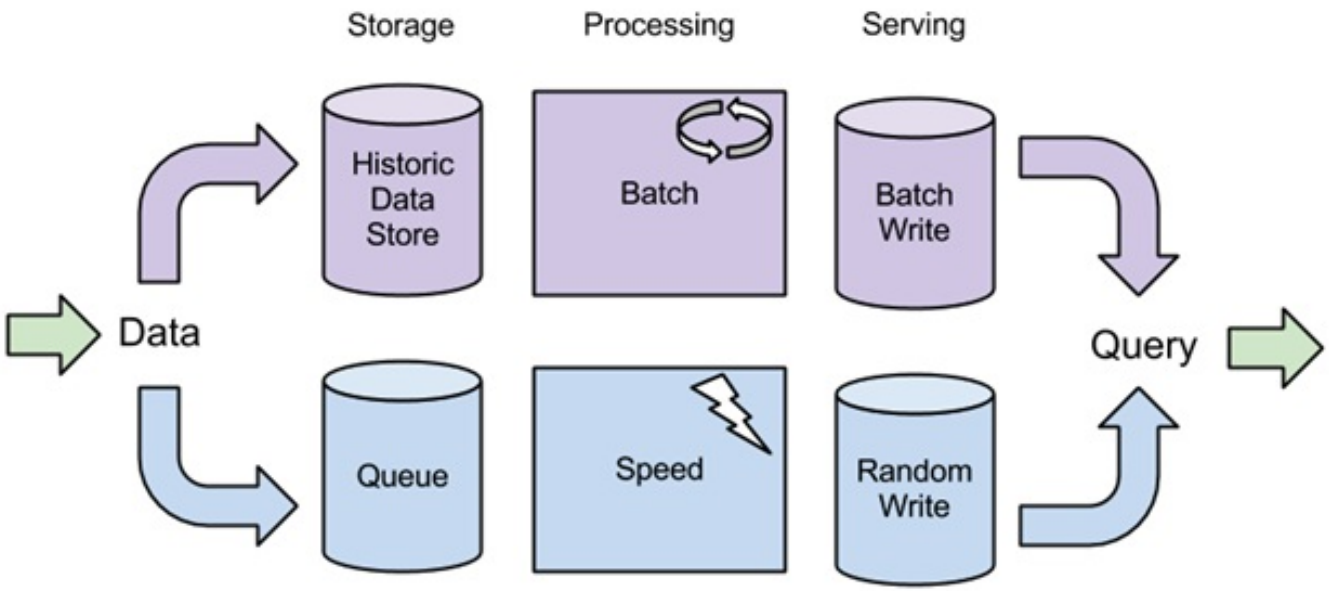
Streaming calculations

Streaming processors can die and drop events, or the wrong version of code might be deployed online, so they are less trustworthy than the Nth revision of a batch calculation.

However, batch calculations are not up-to-date. Want best of both...

Lambda architecture

Solution: run both. Batch-process everything up to the last hour, stream-process starts fresh each hour, and queries combine results.



Programming trends

Language choice

Everything mentioned so far was written in Java (or Scala or Clojure) with hooks for Python and R. Little support for C/C++. Why is that?

Language choice

Everything mentioned so far was written in Java (or Scala or Clojure) with hooks for Python and R. Little support for C/C++. Why is that?

C/C++

Hardest to debug. Mixes analysis with low-level concerns.

JVM (Java et al)

Human-readable internals, stack traces, runtime types.

Python and R

Interactive prompt! Everything can be inspected at runtime.

Language choice

Everything mentioned so far was written in Java (or Scala or Clojure) with hooks for Python and R. Little support for C/C++. Why is that?

C/C++	JVM (Java et al)	Python and R
Hardest to debug. Mixes analysis with low-level concerns.	Human-readable internals, stack traces, runtime types.	Interactive prompt! Everything can be inspected at runtime.
Fastest, raw machine access, static bytecode, manual memory management.	Medium speed, dynamic optimizer. Garbage collector is fast but pauses.	Slowest, though performance-critical code is external.

Language choice

Everything mentioned so far was written in Java (or Scala or Clojure) with hooks for Python and R. Little support for C/C++. Why is that?

C/C++	JVM (Java et al)	Python and R
Hardest to debug. Mixes analysis with low-level concerns.	Human-readable internals, stack traces, runtime types.	Interactive prompt! Everything can be inspected at runtime.
Fastest, raw machine access, static bytecode, manual memory management.	Medium speed, dynamic optimizer. Garbage collector is fast but pauses.	Slowest, though performance-critical code is external.
Used by physicists (and cybersecurity).	Used for large-scale business analytics; suited to networking.	Used by statisticians and data scientists for laptop-analyses.

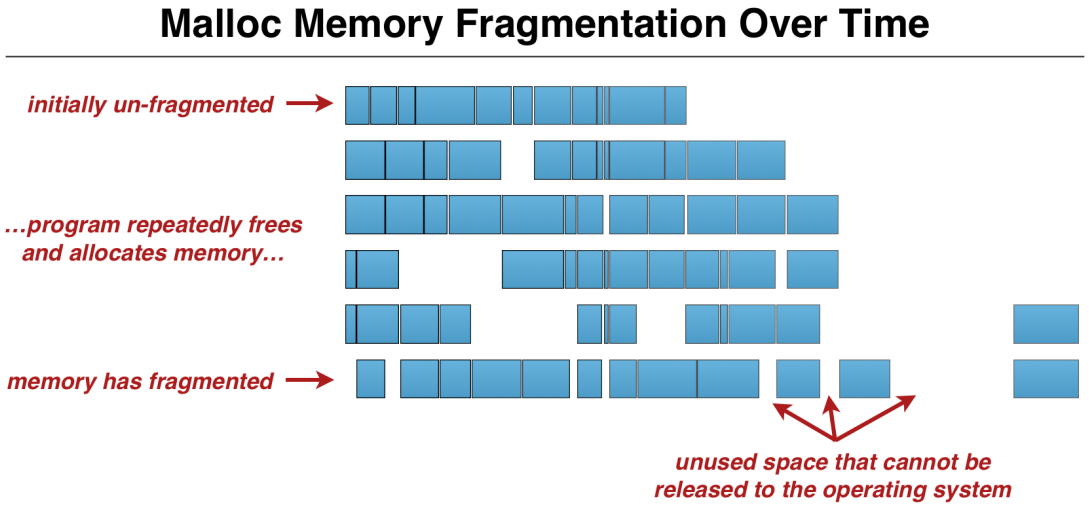
Language choice

Everything mentioned so far was written in Java (or Scala or Clojure) with hooks for Python and R. Little support for C/C++. Why is that?

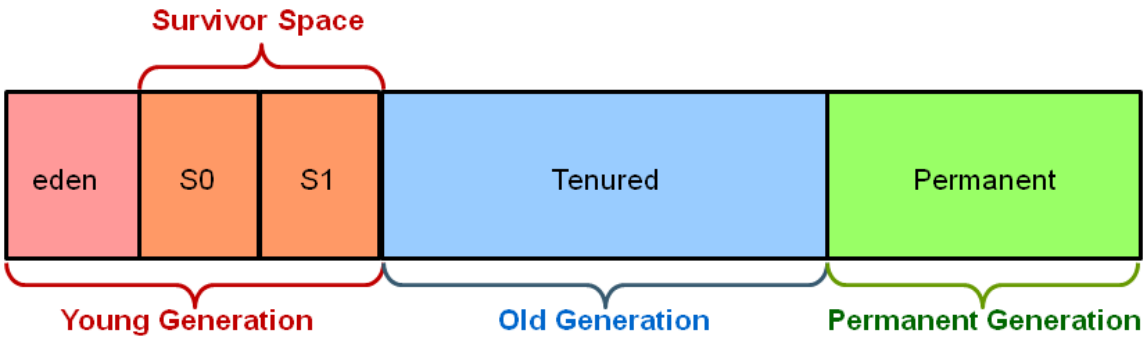
C/C++	JVM (Java et al)	Python and R
Hardest to debug. Mixes analysis with low-level concerns.	Human-readable internals, stack traces, runtime types.	Interactive prompt! Everything can be inspected at runtime.
Fastest, raw machine access, static bytecode, manual memory management.	Medium speed, dynamic optimizer. Garbage collector is fast but pauses.	Slowest, though performance-critical code is external.
Used by physicists (and cybersecurity).	Used for large-scale business analytics; suited to networking.	Used by statisticians and data scientists for laptop-analyses.

Garbage collectors are actually fast

Static memory management fragments the heap, so the "new" (or "malloc") command has to *search* for an available block.



Managed memory fills like a stack, so there is no search, and the whole stack is cleared when surviving objects are moved to longer-term storage.



Garbage collectors are actually fast

New/delete (malloc/free) memory management:

new	delete	random pauses
slow	fast	don't happen

Garbage collectors:

new	delete	random pauses
fast	n/a	happen

Garbage collectors are actually fast

New/delete (malloc/free) memory management:

new	delete	random pauses
slow	fast	don't happen

Garbage collectors:

new	delete	random pauses
fast	n/a	happen

- Good for most distributed applications because network data is intermittent; pauses fill the cracks.
- Worked against a high-throughput application I wrote: input stream was continuous, so garbage collector pauses caused data to overflow the input queue. *Would have been better slower and uniform!*

Immutable data

Another trend is the restriction to immutable data: variables that don't vary and data structures that can only be replaced, not changed.

Mutable variable:

```
result = 0
for i in range(10):
  result = result + i
```

Immutable variable:

```
def add(i):
  if i < 10:
    return i + add(i+1)
  else:
    return 0
result = add(0)
```

Mutable data structure:

```
result = []
for i in range(10):
  result.append(i)
```

Immutable data structure:

```
def appended(i):
  if i < 10:
    return [i] + appended(i+1)
  else:
    return []
result = appended(0)
```

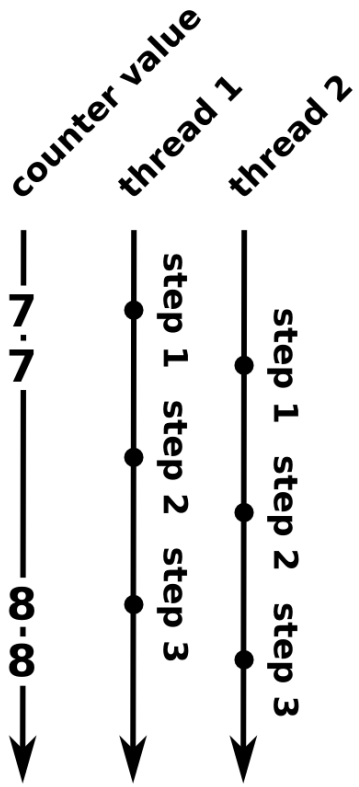
Immutable data

Why impose this limitation? Because calculations are distributed.

- It is extremely difficult to maintain consistent values for mutable variables across computers in a network (CAP theorem).
- Multiple threads in the same computer acting on a shared variable can easily corrupt it.

Example: concurrent access to a mutable counter.

```
def updateCounter():
    # step 1
    currentValue = getCounterValue()
    # step 2
    newValue = currentValue + 1
    # step 3
    setCounterValue(newValue)
```



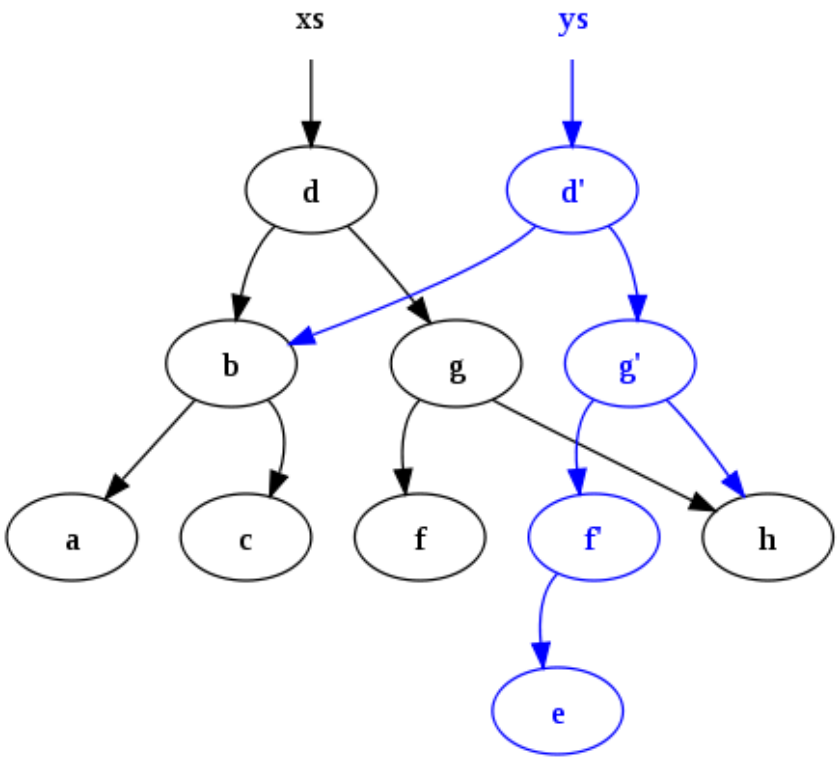
Immutable data

Curse of immutable data: large data structures must be replaced by a new version for each change, suggesting an expensive copy.

Immutable data

Curse of immutable data: large data structures must be replaced by a new version for each change, suggesting an expensive copy.

Blessing of immutable data: if immutable (and acyclic), multiple versions can share parts without *fully* copying.



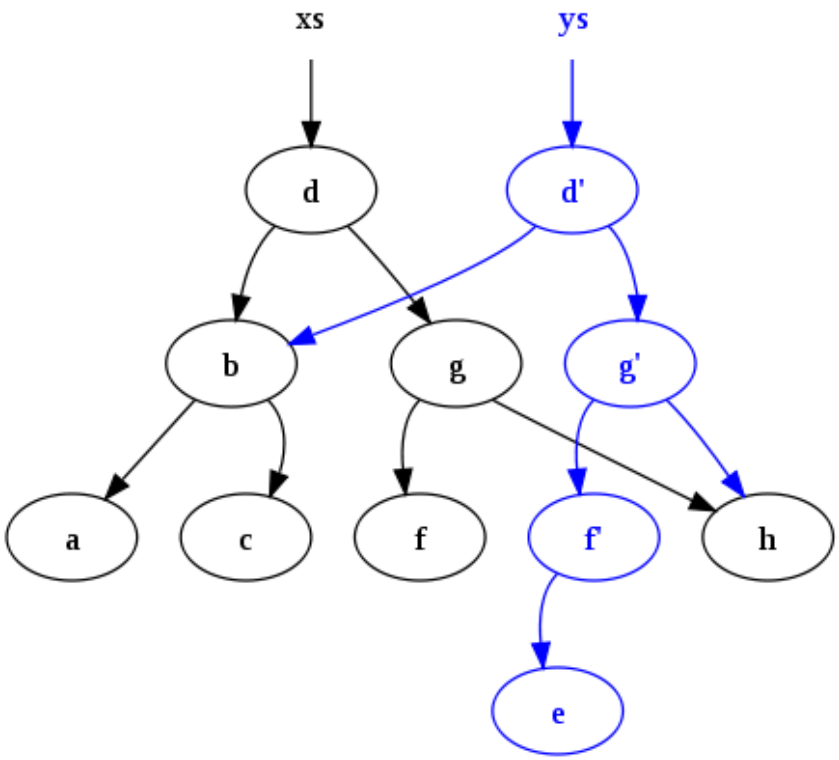
`ys = xs with e added to f`

Only the path from `f` to root needs to be duplicated.

Immutable data

Curse of immutable data: large data structures must be replaced by a new version for each change, suggesting an expensive copy.

Blessing of immutable data: if immutable (and acyclic), multiple versions can share parts without *fully* copying.



`ys = xs with e added to f`

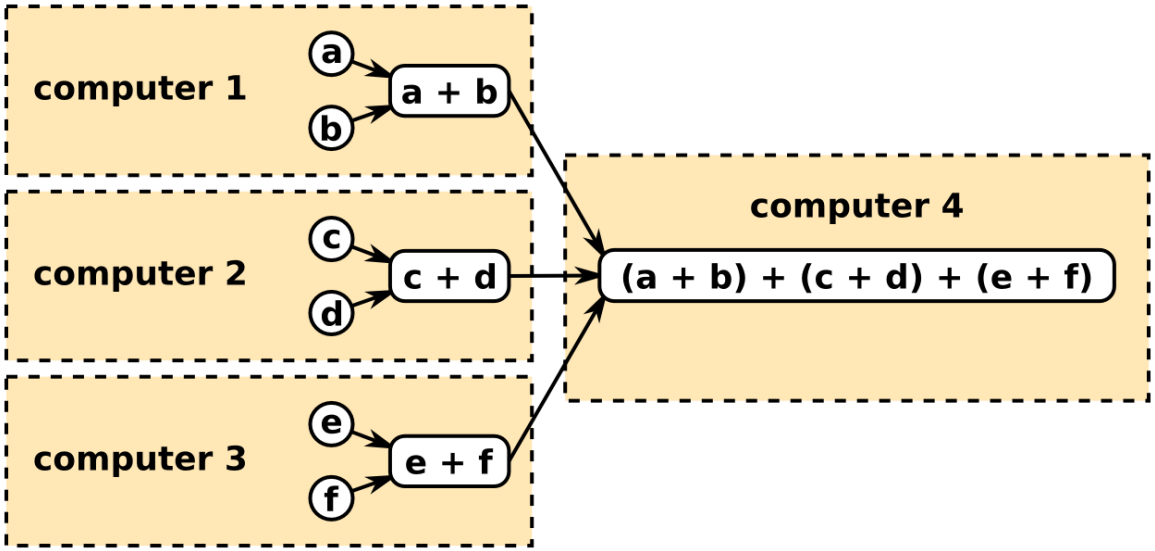
Only the path from `f` to root needs to be duplicated.

	update	"copy"
mutable	$\mathcal{O}(1)$	$\mathcal{O}(n)$
immutable	$\mathcal{O}(\log(n))$	$\mathcal{O}(1)$

Monoids for combining results

A monoid is a group without inverses:

- an identity: 0 for which $a + 0 = a$
- an associative operator: $a + (b + c) = (a + b) + c$

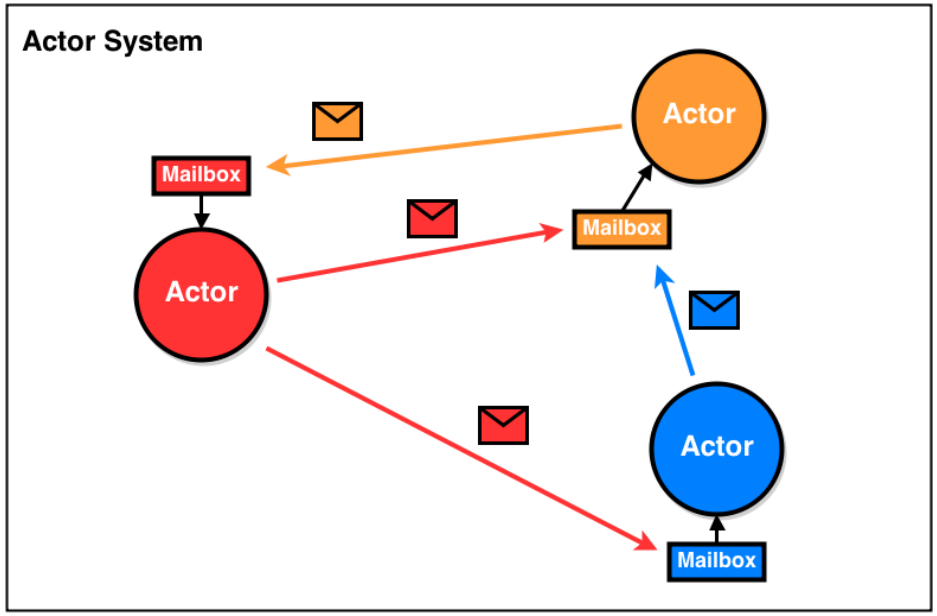



```
@case class Ratio(numer, denom):
    def add(self, other):
        return Ratio(self.numer + other.numer,
                     self.denom + other.denom)
    def value(self):
        return self.numer / self.denom
```

Actors for encapsulating work

A distributed calculation can be made out of small, single-threaded mini-programs called Actors.

- Actors have a mailbox that accumulates messages in a queue.
- Their only behavior is to react to these messages, one at a time.
- They can send messages to other actors.
- They can freely use mutable variables to process a message, and may or may not be allowed to maintain persistent, mutable state. (Would not be fail-fast.)
- The Akka framework is a popular example.



Last topic: static typing

C/C++ and Java declare and check data types during compilation to generate more efficient bytecode.

Python and R only check types immediately before they are used.

But a compile-time (static) type check is also a great way to prevent human error, especially in large or long-running calculations.

Last topic: static typing

C/C++ and Java declare and check data types during compilation to generate more efficient bytecode.

Python and R only check types immediately before they are used.

But a compile-time (static) type check is also a great way to prevent human error, especially in large or long-running calculations.

- My most common bug involved a Python `None` (representing a missing value) entering arithmetic expressions.
- Requiring explicitly named classes is safer than generic tuples or argument lists with many parameters of the same type.
- Wrappers around numbers prevents unit mismatches, as does CMSSW's distinction between vectors and affine points.
- A type check for matrix dimensions would be a great tool.

Last topic: static typing

Example: preventing all "null" errors at runtime.

- C/C++ raises a segmentation fault when reading `NULL` pointers.
- Java raises an exception that provides a line number in the code.
- Python's null is `None`, which raises an exception if you try to use it in arithmetic.
- Some languages convert it to zero and move on!

Scala has a wrapper type for objects that could be null: `Option[T]`.

```
var x: Option[Double] = Some(3.14)
x = None

var y: Double = x match {
  case Some(y) => y + 3      // y is a new variable
  case None => -999          // no new variables in this scope
}
```

The type system forces the programmer to handle the null case; the above code will never raise an exception at runtime.

Conclusions

- Data scientists cover the same range of activities as experimental high-energy physicists, but work with a broader class of models for more diverse purposes.
- Some of these statistical techniques could be useful in physics.
- The software tools could be useful as well:
  - Spark for iterative map-reduce could formalize and accelerate alignment and calibration.
  - Streaming processors or a lambda architecture could aid data monitoring.
- Data scientists use more high-level programming languages and techniques, which reduce debugging time.
  - Doesn't have to be the JVM. New languages (Rust, Julia, Numba just-in-time compiler for Python) produce native bytecode with high-level abstractions.