A call for sanity in NoSQL

Nathan Marz
@nathanmarz

“Doofus programmer”

You broke the build!

Where are the tests?

The site is down!

DELETE FROM
users...


Oops, I forgot the WHERE clause!

=

=

No problem...

Mistakes are guaranteed

Create

Read

Update

Delete

Mutable database = **Guaranteed corruption**


Insane

Schemaless databases


Insane

Avoiding complexity


HBASE


Cassandra


riak


Voldemort


Neo4j


mongoDB

counter++


Insane

Denormalization

ID	Name	Location ID
1	Sally	3
2	George	1
3	Bob	3

Location ID	City	State	Population
1	New York	NY	8.2M
2	San Diego	CA	1.3M
3	Chicago	IL	2.7M

Normalized schema

Join is too expensive, so denormalize...

ID	Name	Location ID	City	State
1	Sally	3	Chicago	IL
2	George	1	New York	NY
3	Bob	3	Chicago	IL

Location ID	City	State	Population
1	New York	NY	8.2M
2	San Diego	CA	1.3M
3	Chicago	IL	2.7M

Denormalized schema

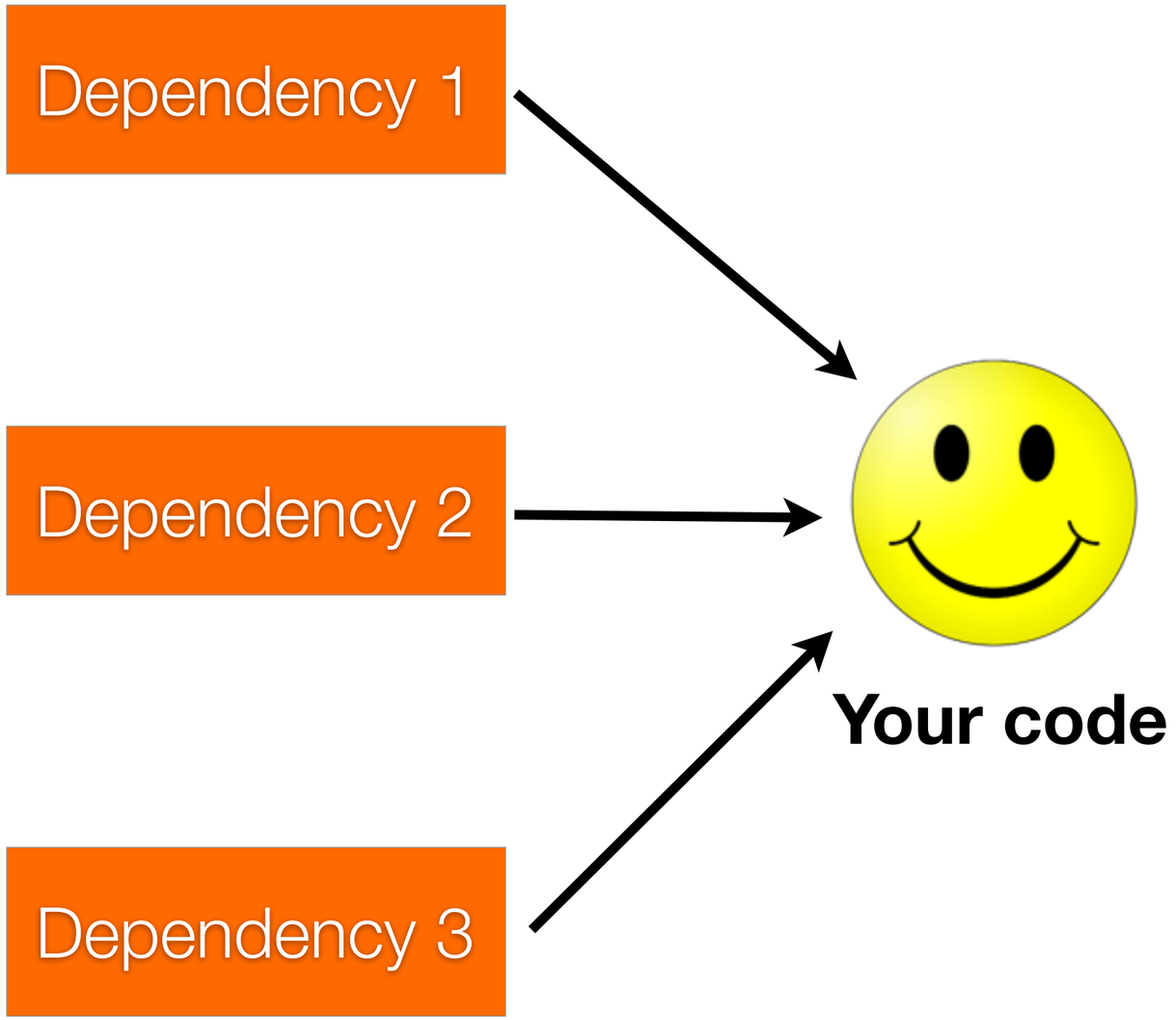

Insane

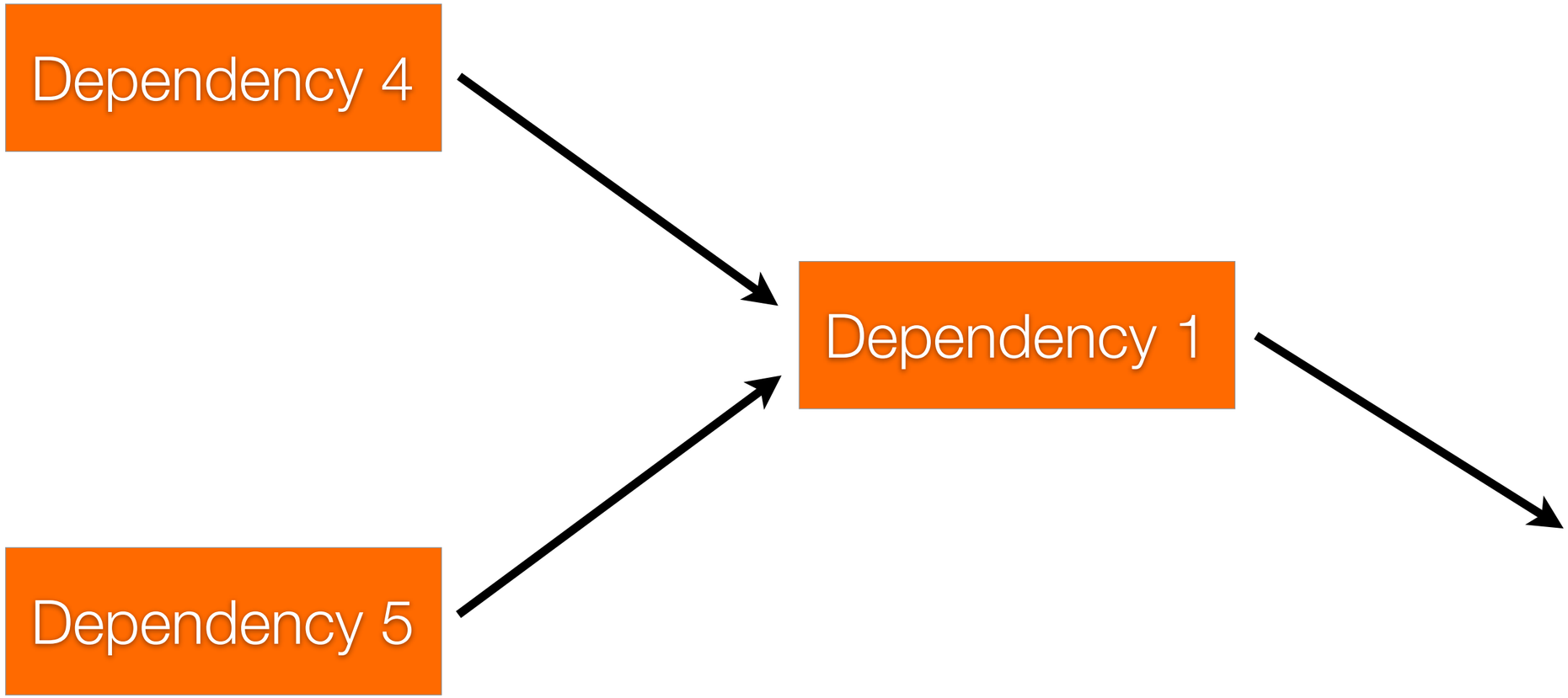
What is the source of the insanity?

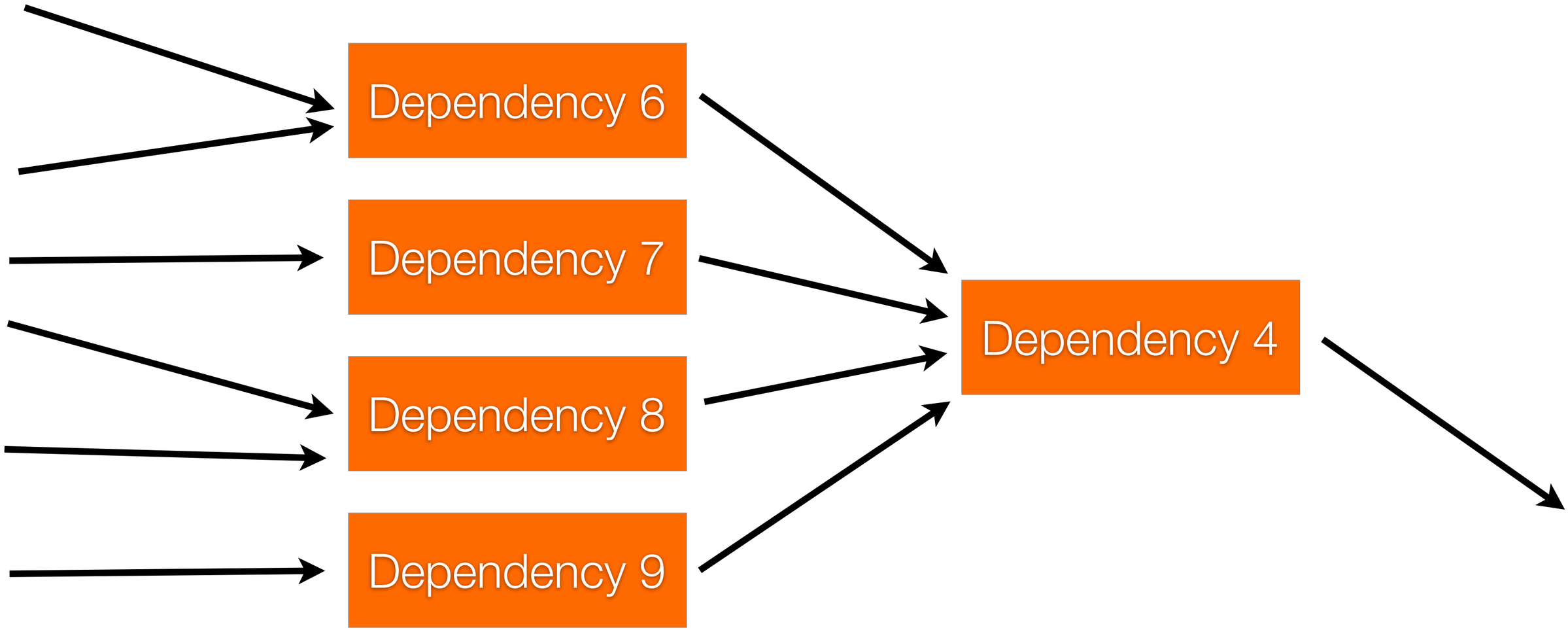
Code is deterministic.
I understand logic.
Therefore, I can write correct code.

Programming fallacy

Your code is wrong


Dependency 1
Dependency 2
Dependency 3
Your code

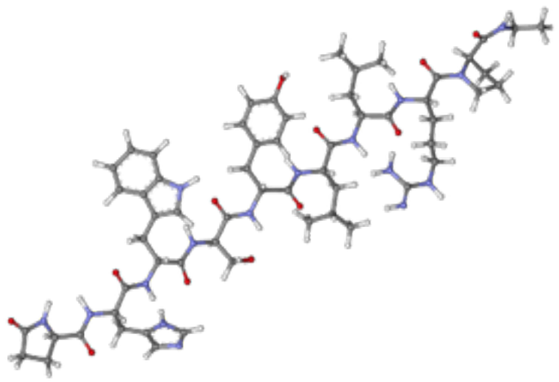

Dependency 4
Dependency 5
Dependency 1


Dependency 6
Dependency 7
Dependency 8
Dependency 9
Dependency 4

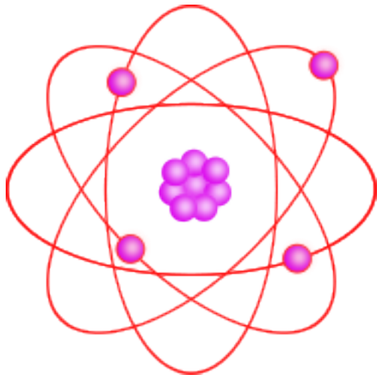
Hardware

Dependency 3,000,000

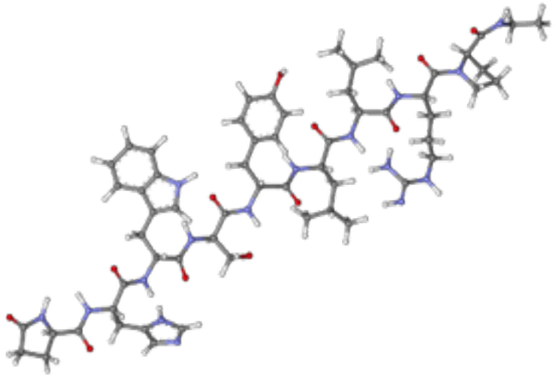
Electronics

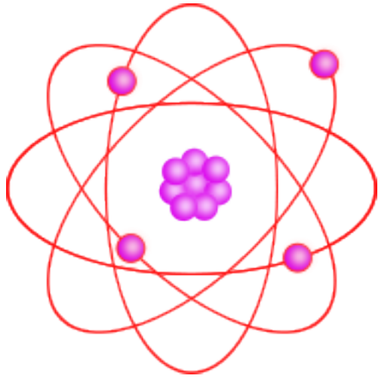


Chemistry



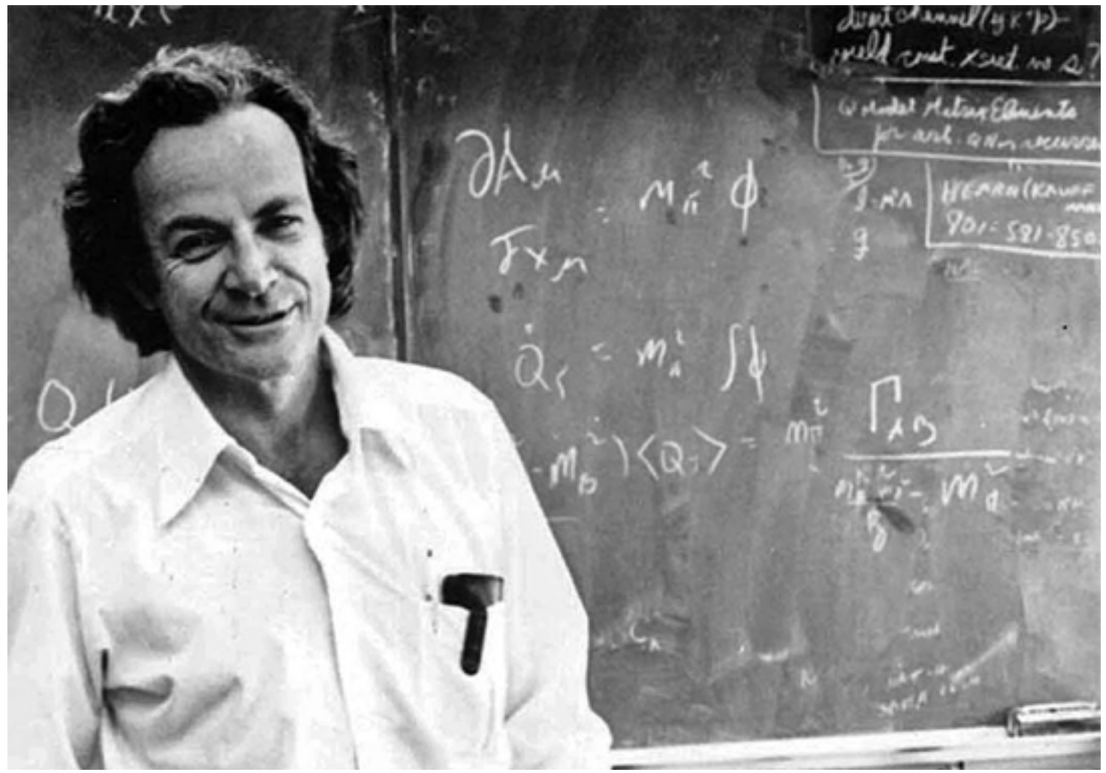
Atomic physics





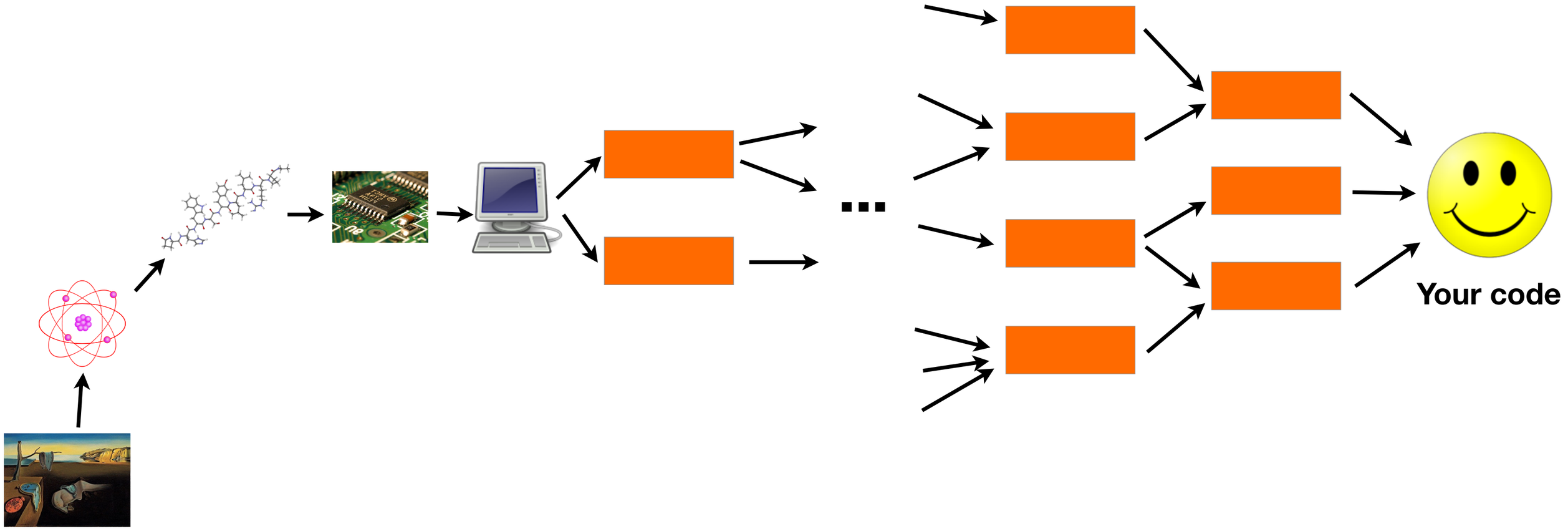
Quantum mechanics

I think I can safely say that nobody understands quantum mechanics.



Richard Feynman

Your code is wrong

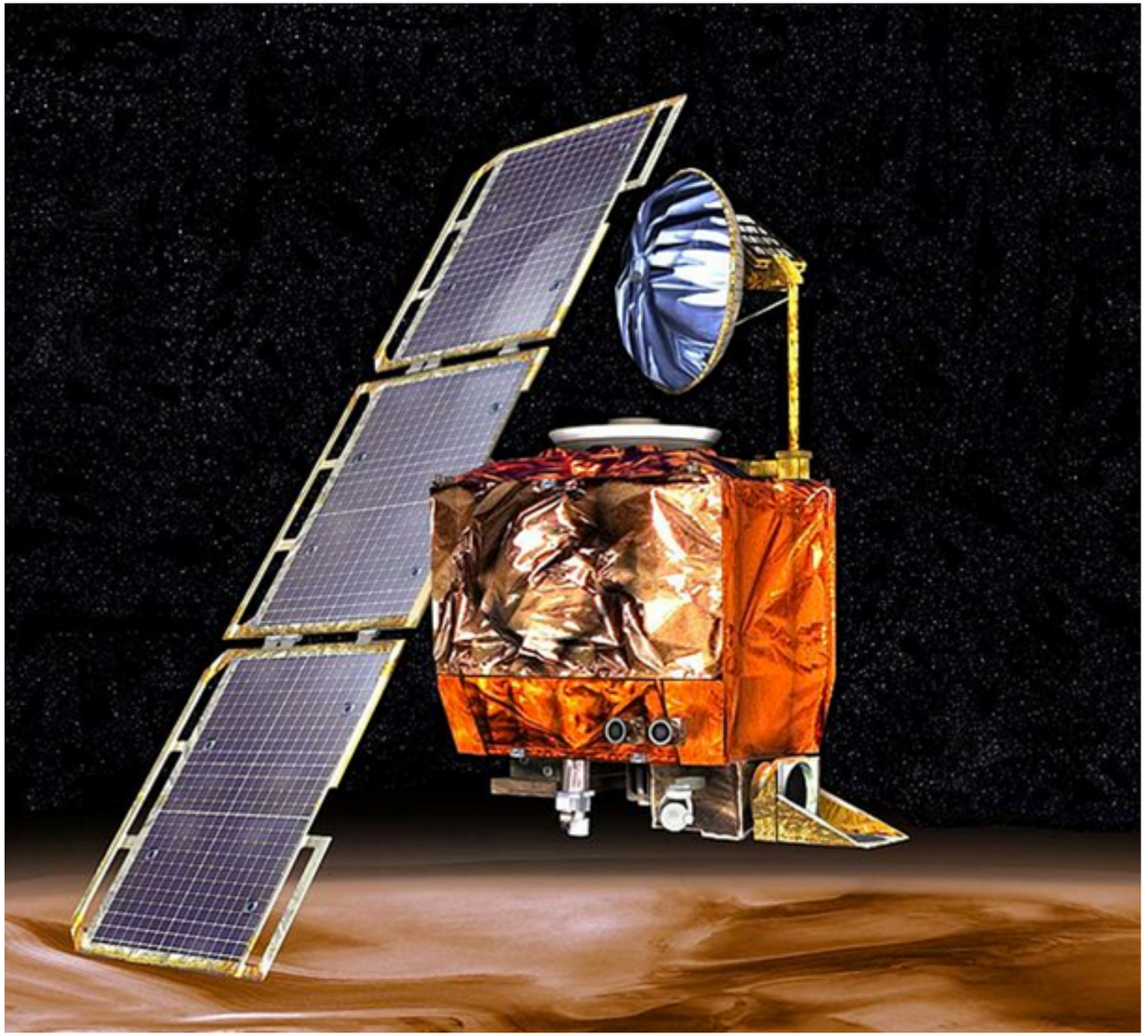

Your code

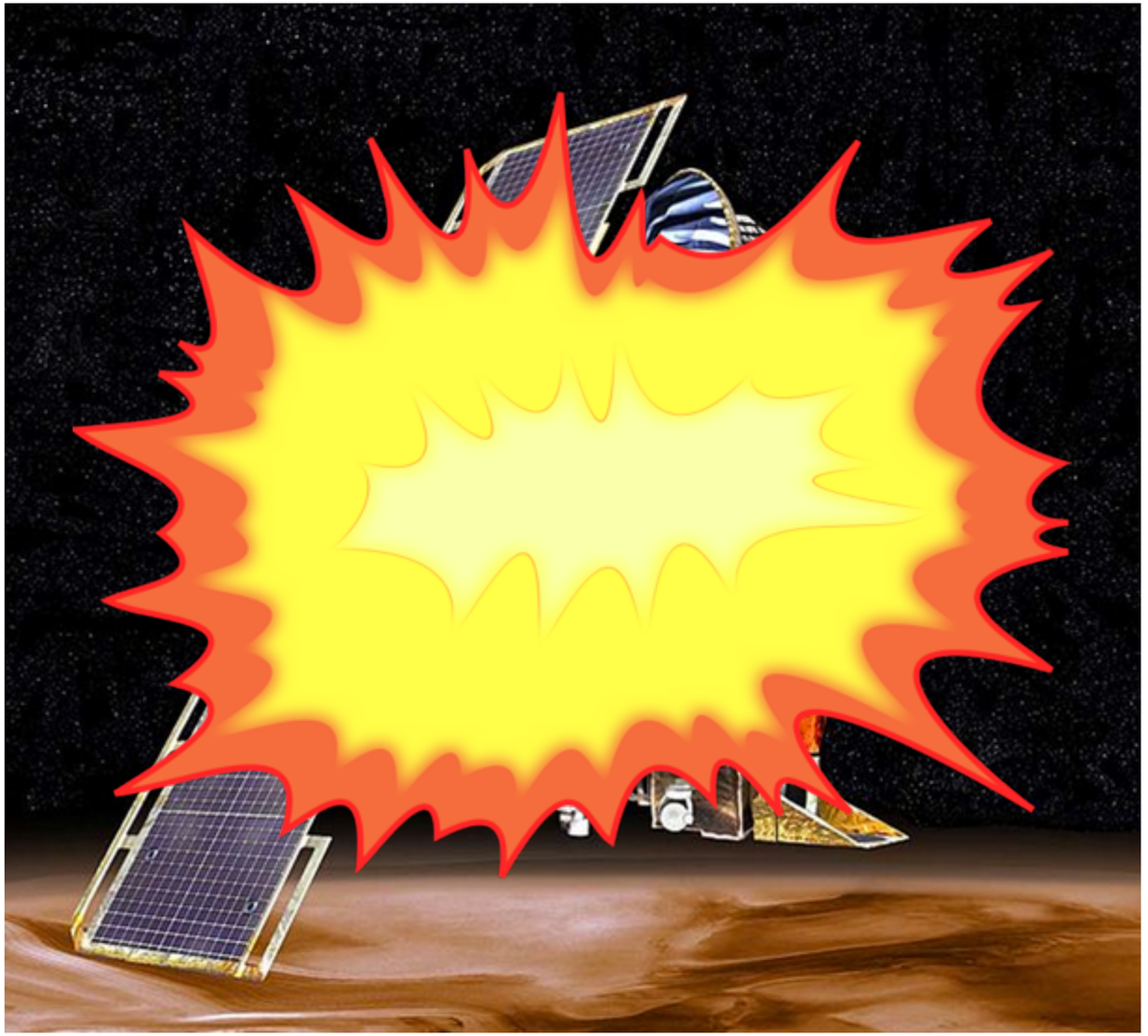
Infinite regress

All the software you've used has had bugs in it

Including the software

you've written





Insanity

- Mutability
- Schemaless databases
- Eventual consistency / read-repair
- Denormalization

Person	Location
Sally	New York
Bob	Chicago

Mutability

Person	Location
Sally	London
Bob	Chicago

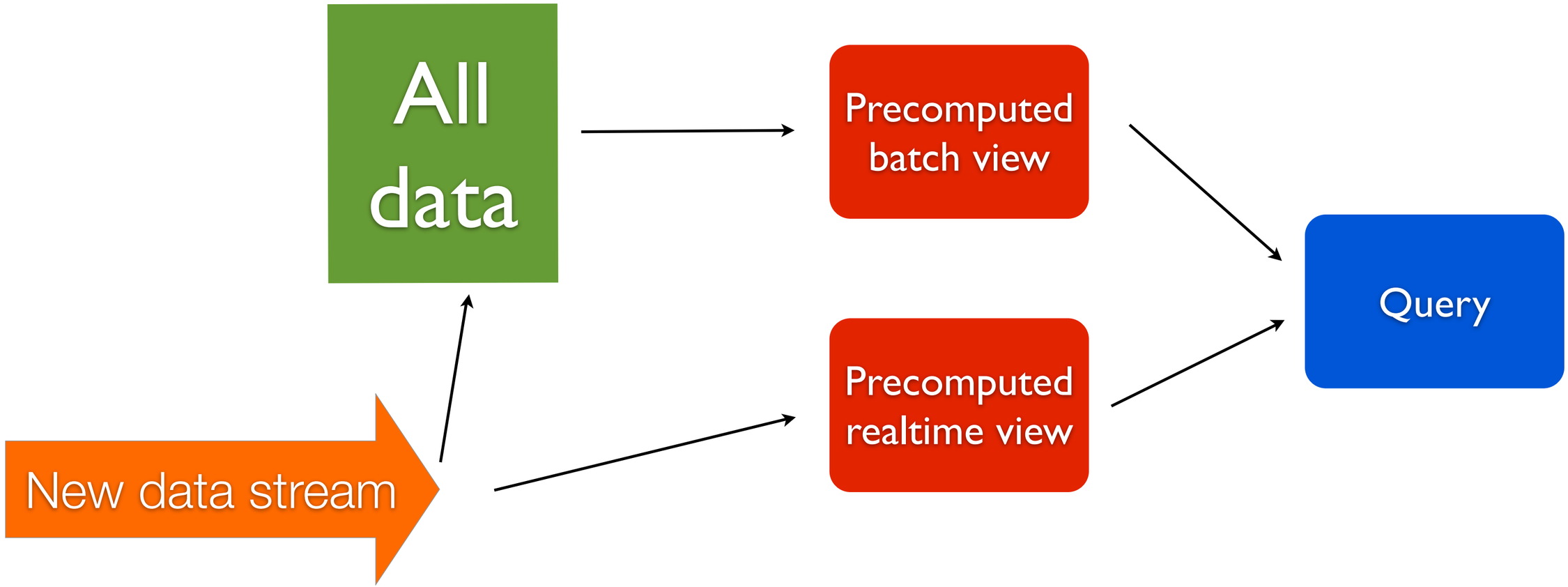
Mutability

Person	Location	Time
Sally	New York	1318358351
Bob	Chicago	1327928370

Immutability

Person	Location	Time
Sally	New York	1318358351
Bob	Chicago	1327928370
Sally	London	1338273801

Immutability



Lambda Architecture

Relational database


NewSQL

All data problems?

What does a data system do?

Retrieve data that you previously stored?

Put

Get

Counterexamples

Store location information on people

How many people live in a particular location?

Where does Sally live?

What are the most populous locations?

Counterexamples

Store pageview information

How many pageviews on September 2nd?

How many unique visitors over time?

Counterexamples

Store transaction history for bank account

How much money does George have?

How much money do people spend on housing?

What does a data system do?

Query = Function(All data)

Example query

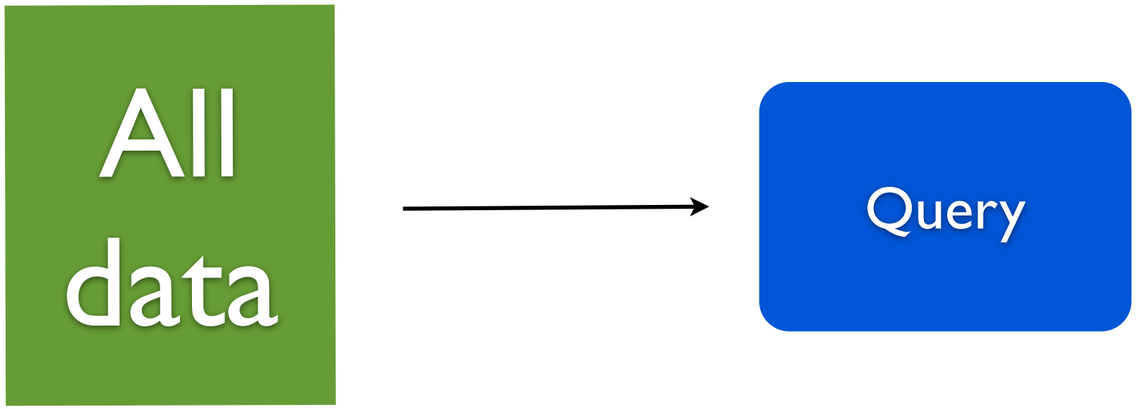
Total number of pageviews to a URL over a range of time

Example query

```
function pageviewsOverTime(allData, url, start, end) {
    count = 0
    for(data: allData) {
        if(data.url == url &&
            data.timestamp >= start &&
            data.timestamp <= end) {
             count++
        }
    }
    return count
}
```

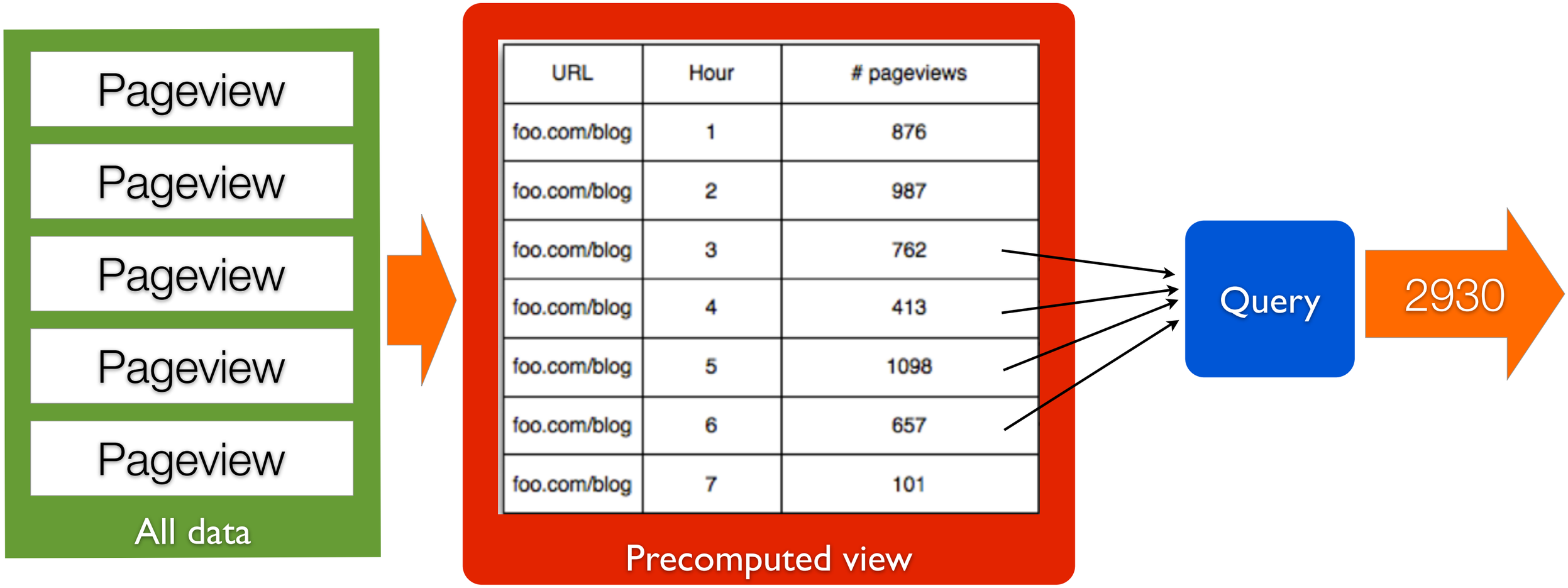
Implementation

On-the-fly computation



Precomputation

Example query

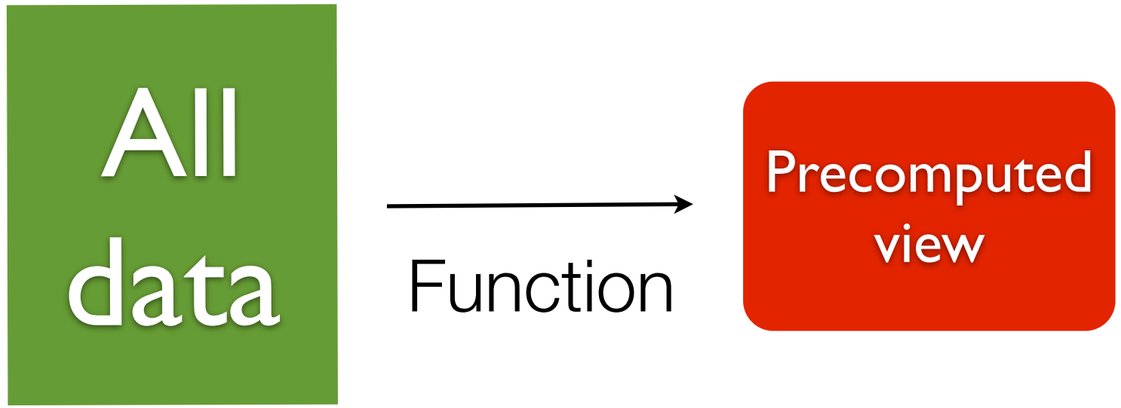


Precomputation

(Immutable)

Precomputation

Computing views



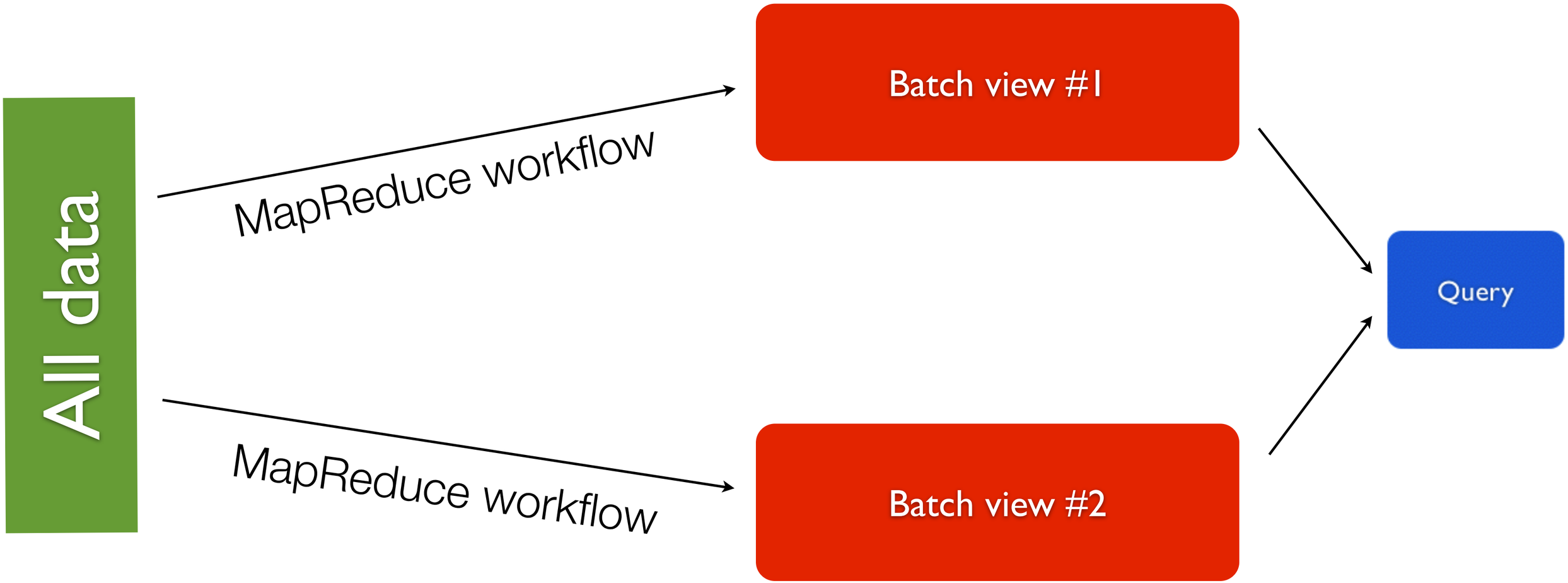
Function that takes in all data as input

Batch processing

MapReduce

MapReduce is a framework for computing arbitrary functions on arbitrary data

MapReduce precomputation




All data
Precomputed view
Query

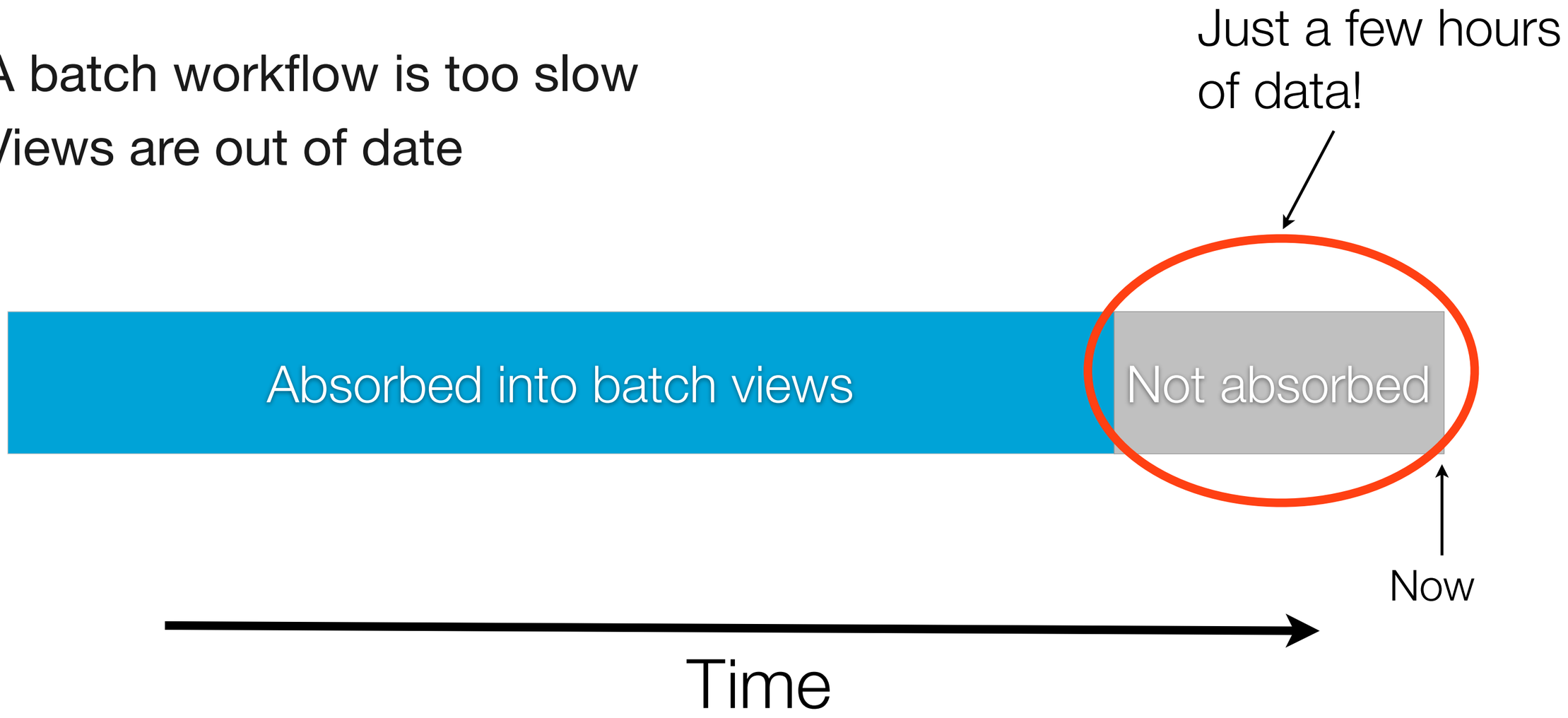
Normalized

Denormalized

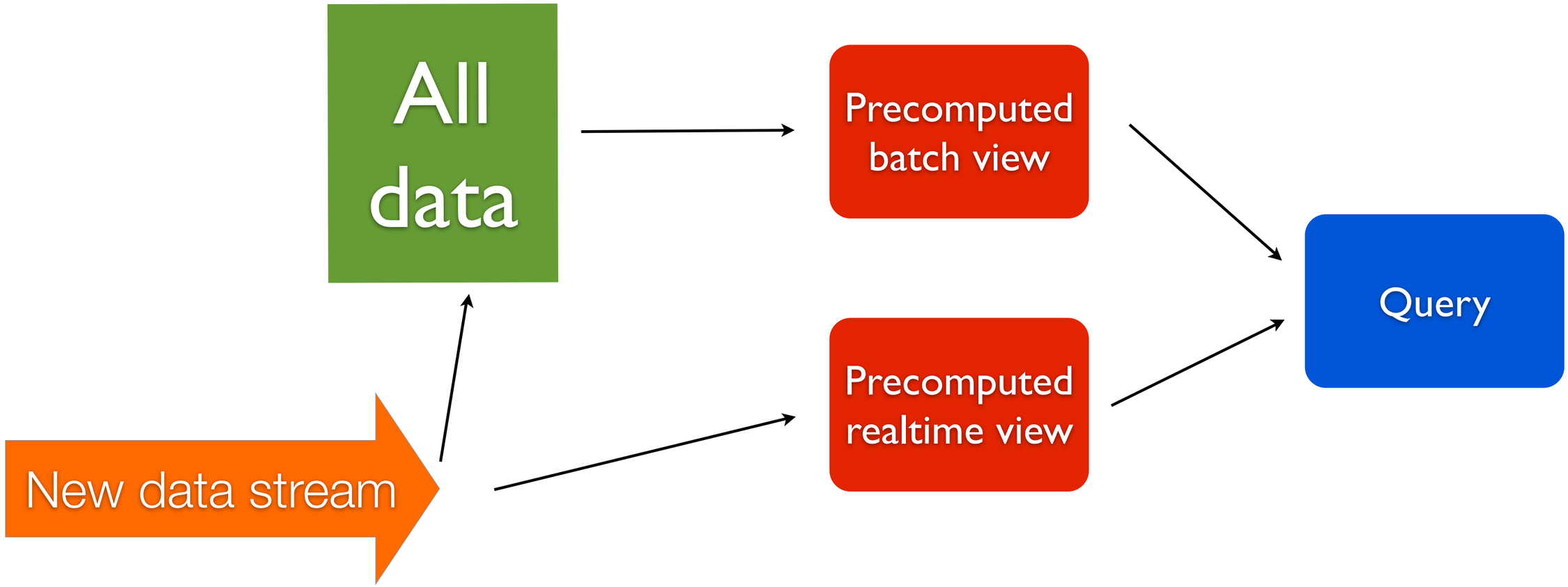
Human fault-tolerant

Not quite...

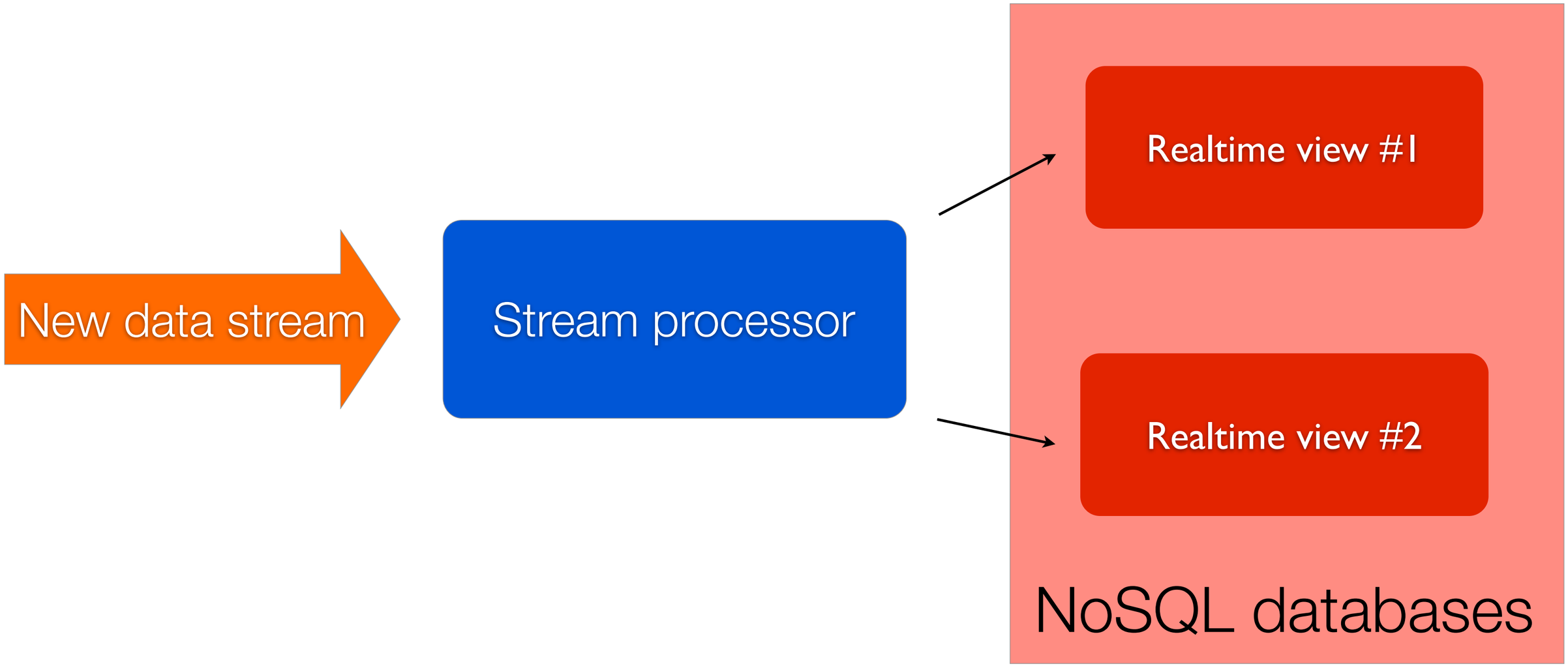
- A batch workflow is too slow
- Views are out of date



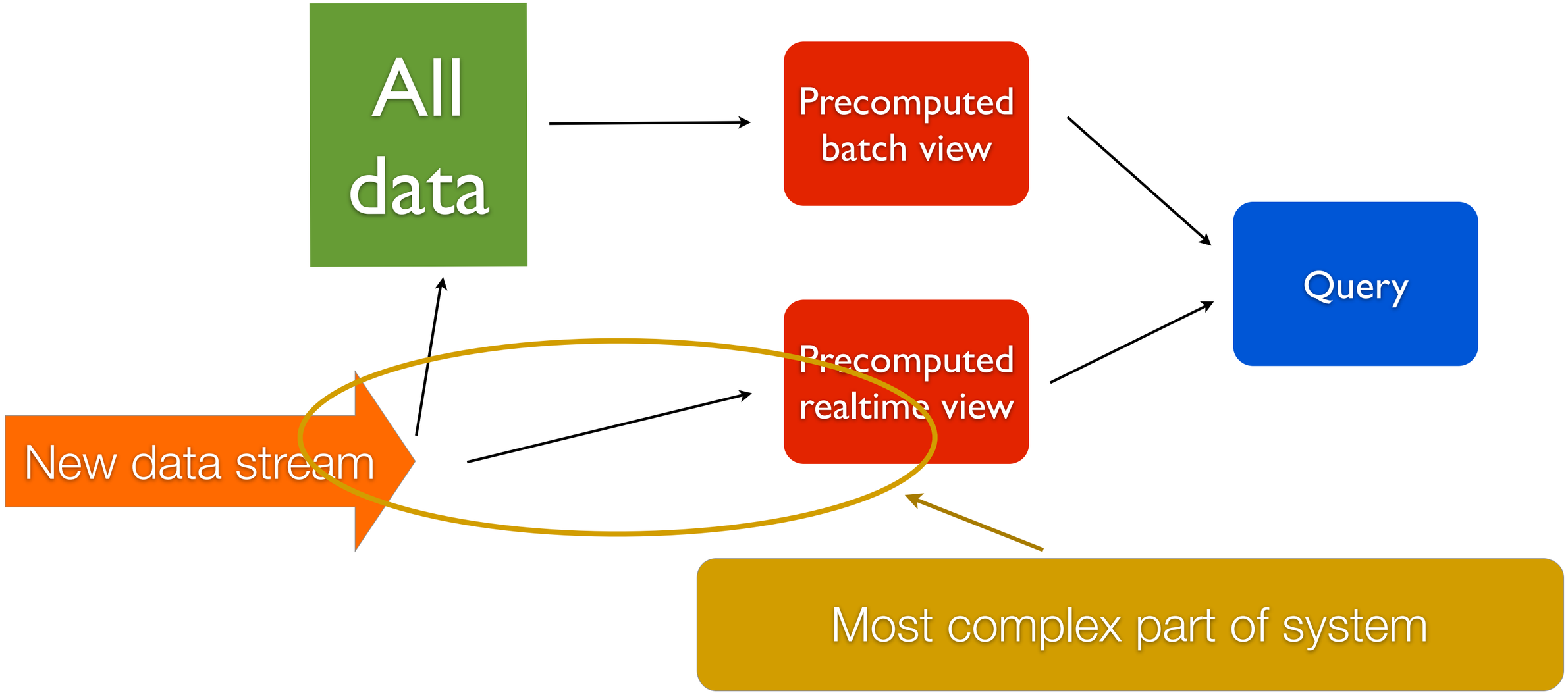
Precomputation



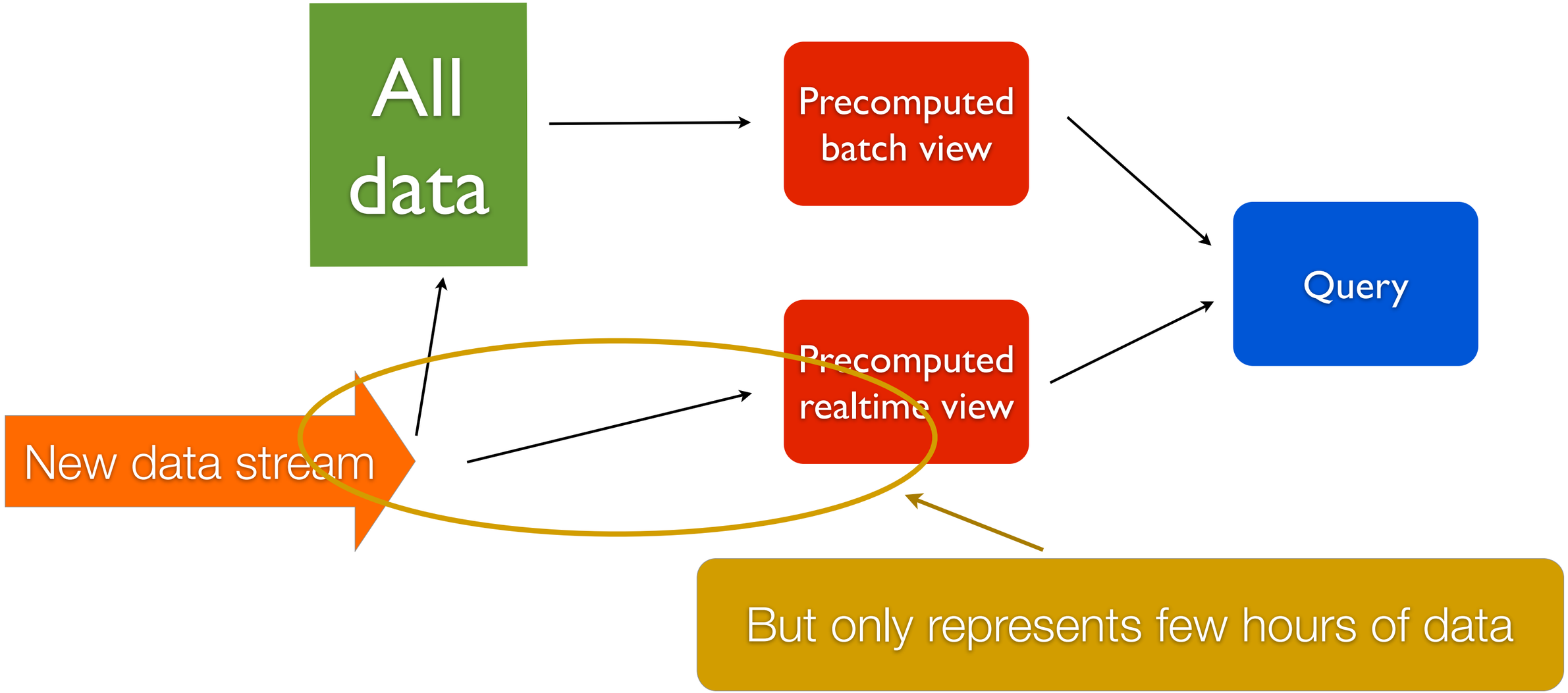
Compute parallel realtime views


New data stream
Stream processor
Realtime view #1
Realtime view #2
NoSQL databases

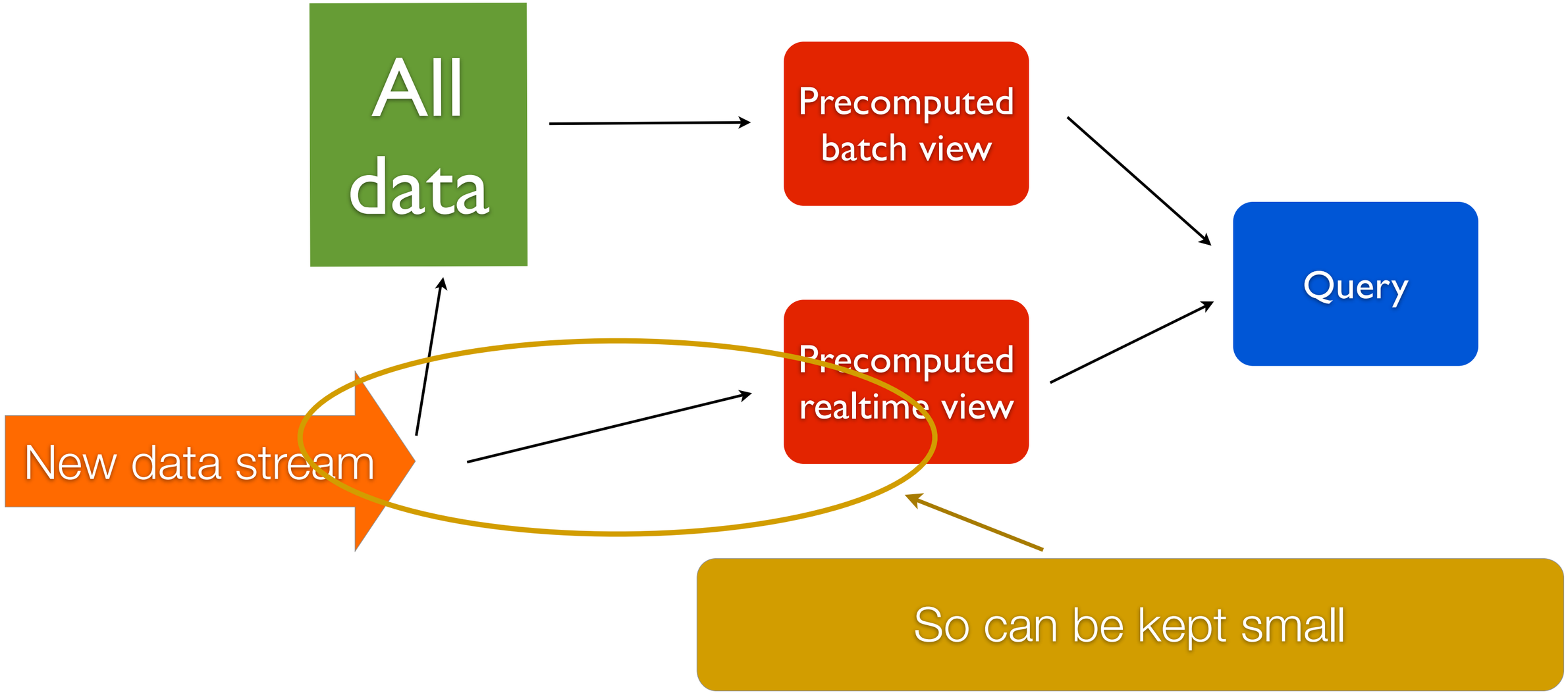
Precomputation



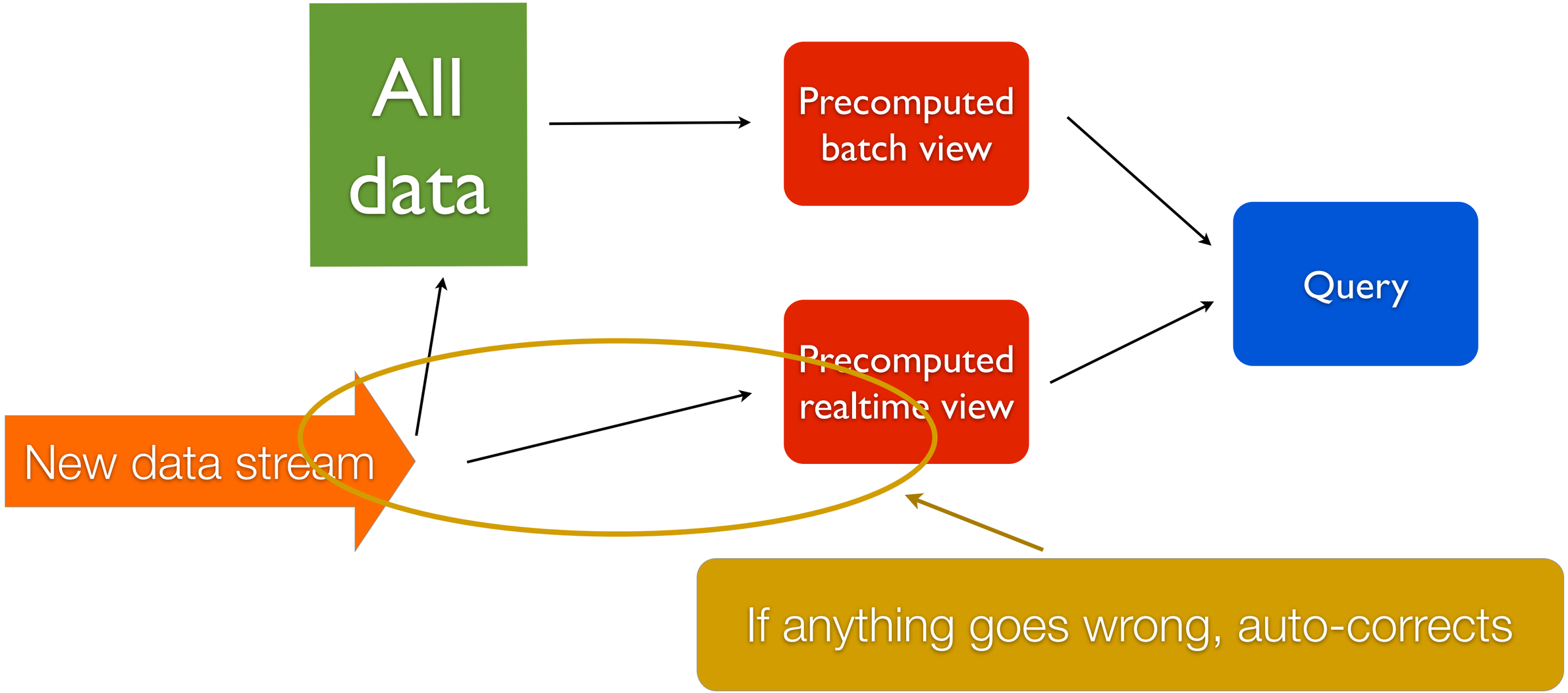
Precomputation



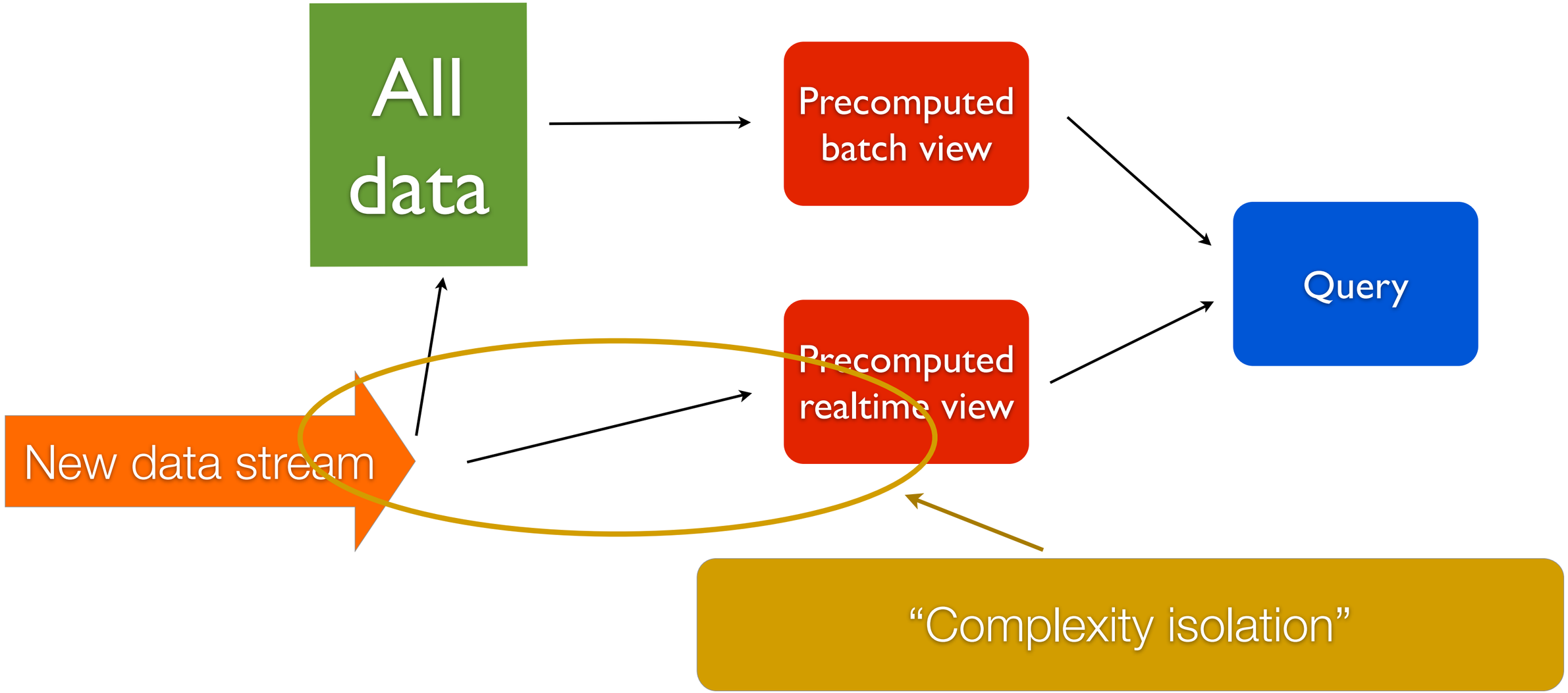
Precomputation



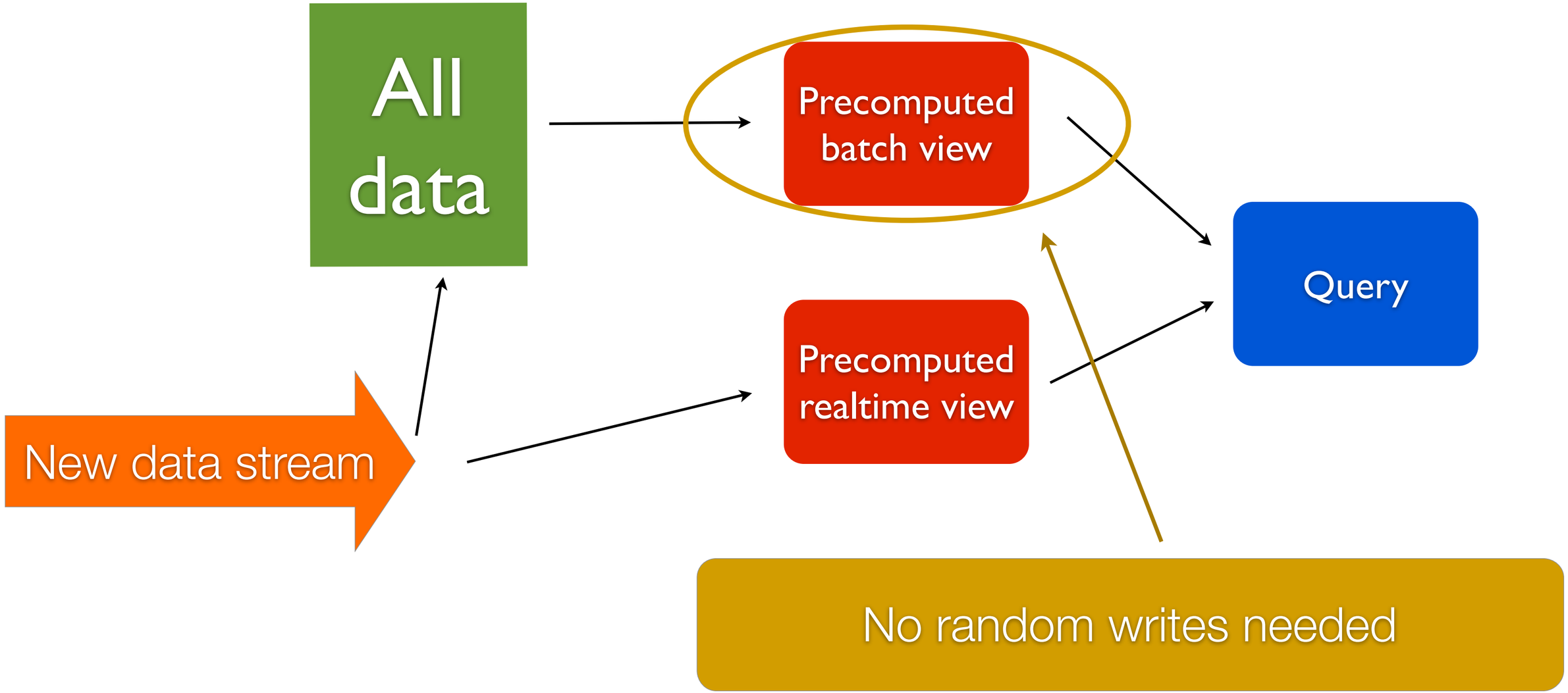
Precomputation

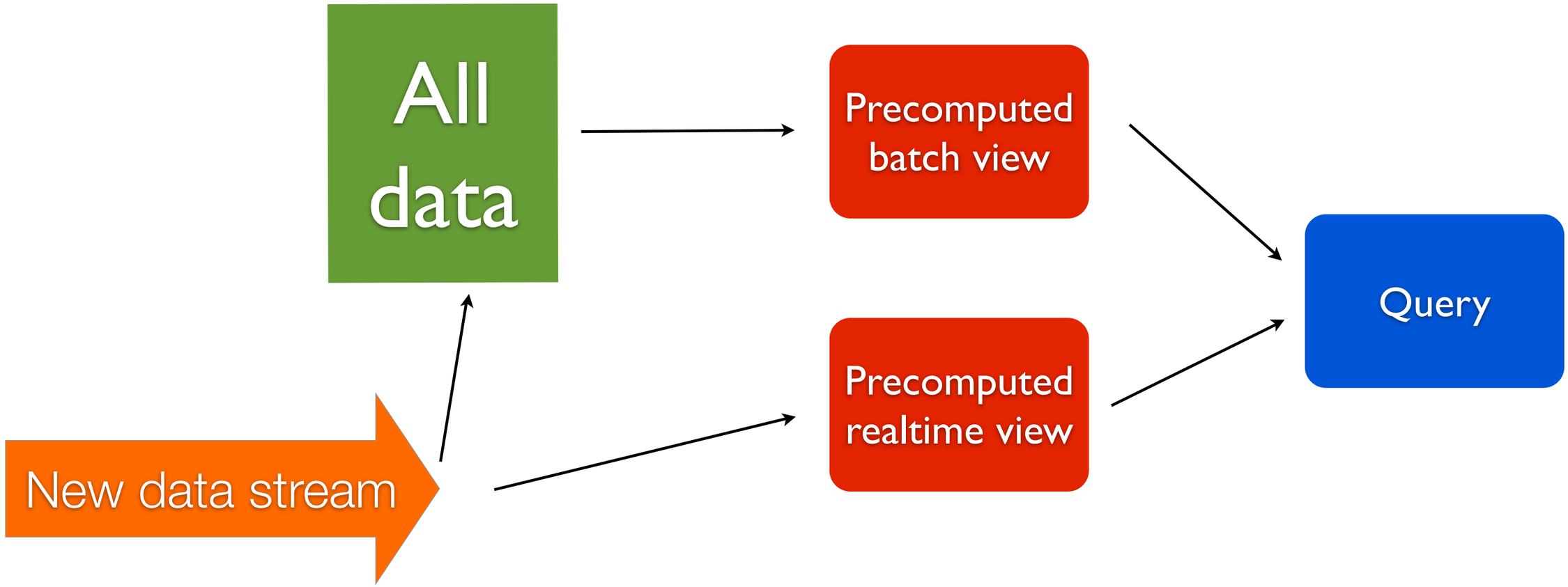


Precomputation



Precomputation





"Lambda Architecture"

Applying the Lambda Architecture

Unique visitors over a range of hours

Hours

Pageviews

1	2	3	4
A B C A	B	C D E	C D E

Unique visitors over a range of hours

Hours

1	2	3	4
Pageviews	A B C A	B	C D E	C D E

Equivs

A = B
B = D

What should view look like?

Attempt #1

URL	Hour range	Count
foo.com/blog	1-1	3
	1-2	6
	1-3	6
	2-2	4
	2-3	5
	3-3	1
bar.com/foo	1-1	3
	1-2	6
	1-3	6
	2-2	4
	2-3	5
	3-3	1

Attempt #2

URL	Hour	Count
foo.com/blog	1	3
	2	6
	3	6
	4	8
	5	5
	6	1
bar.com/foo	1	1
	2	11
	3	23
	4	8
	5	9
	6	0

Attempt #3

URL	Hour	Visitors
foo.com/blog	1	A, B, C
	2	A
	3	D, E
	4	A
	5	B, C, D
	6	F, G
bar.com/foo	1	A
	2	B, C
	3	A, C
	4	D
	5	A, E
	6	A

Attempt #4

URL	Hour	HyperLogLog set
foo.com/blog	1	<HLL>
	2	<HLL>
	3	<HLL>
	4	<HLL>
	5	<HLL>
	6	<HLL>
bar.com/foo	1	<HLL>
	2	<HLL>
	3	<HLL>
	4	<HLL>
	5	<HLL>
	6	<HLL>

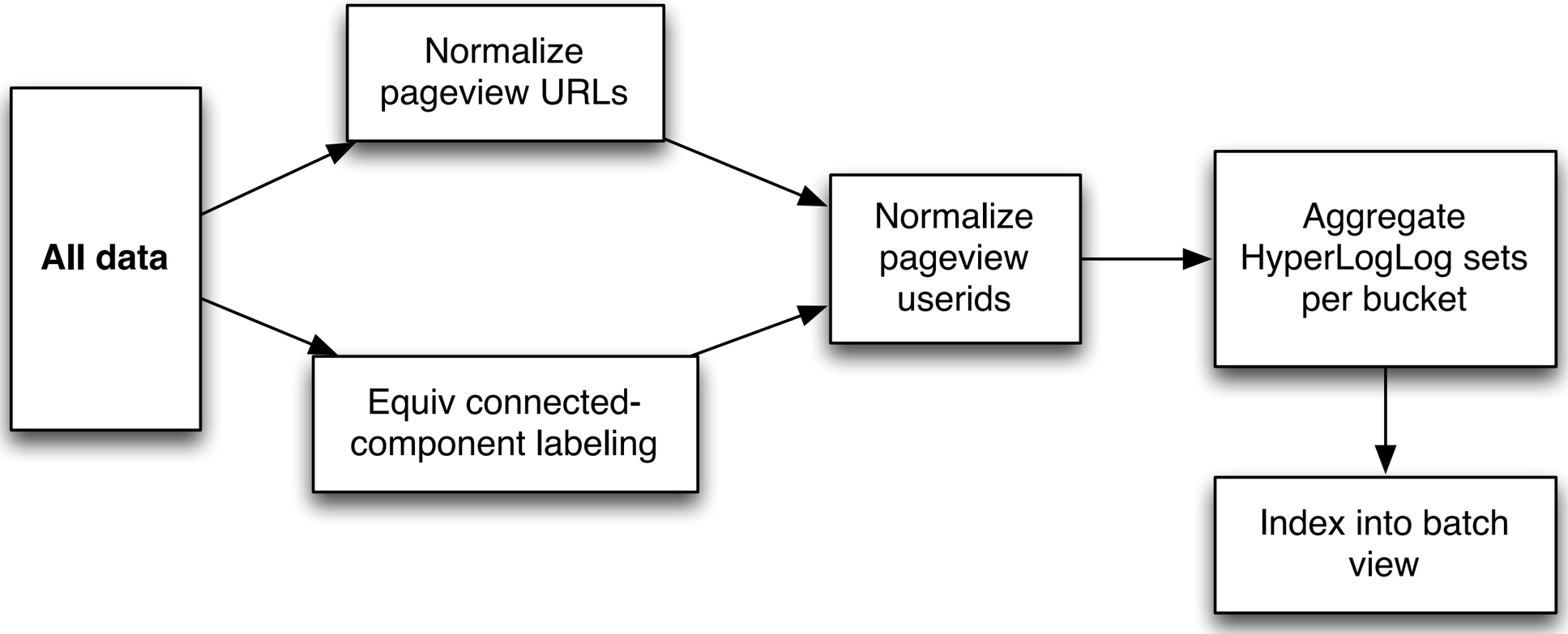
Final attempt

URL	Hour	HyperLogLog set
foo.com/blog	1	<HLL>
	2	<HLL>
	3	<HLL>
	4	<HLL>
	5	<HLL>
	6	<HLL>
bar.com/foo	1	<HLL>
	2	<HLL>
	3	<HLL>
	4	<HLL>
	5	<HLL>
	6	<HLL>

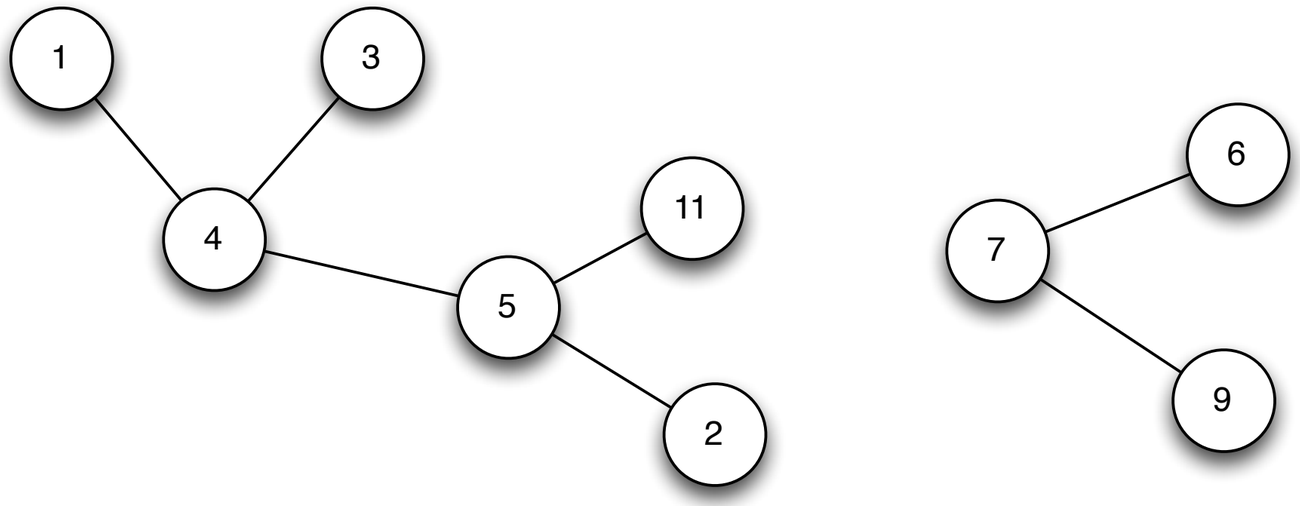
URL	Week	HyperLogLog set
foo.com/blog	1	<HLL>
	2	<HLL>
	3	<HLL>
	4	<HLL>
	5	<HLL>
	6	<HLL>
bar.com/foo	1	<HLL>
	2	<HLL>
	3	<HLL>
	4	<HLL>
	5	<HLL>
	6	<HLL>

URL	Month	HyperLogLog set
foo.com/blog	1	<HLL>
	2	<HLL>
	3	<HLL>
	4	<HLL>
	5	<HLL>
	6	<HLL>
bar.com/foo	1	<HLL>
	2	<HLL>
	3	<HLL>
	4	<HLL>
	5	<HLL>
	6	<HLL>

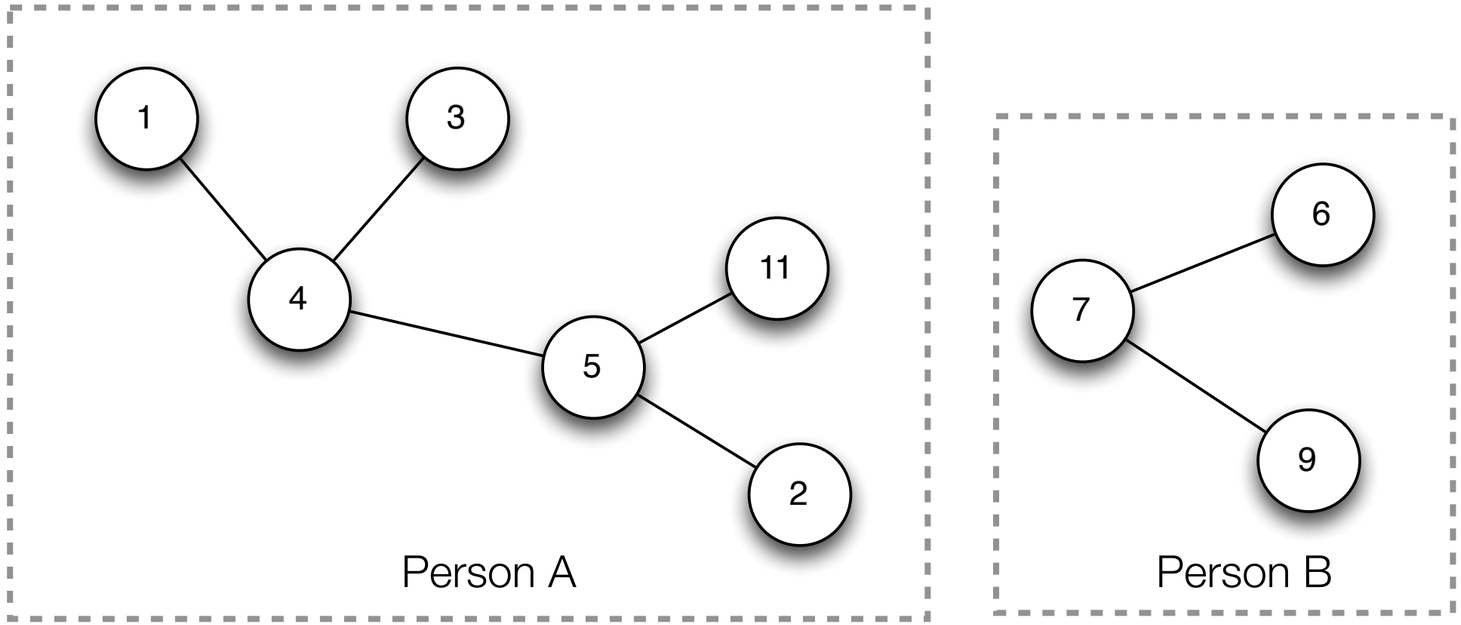
Batch workflow



Equiv graph



Equiv graph



Equiv graph

1 -> A

2 -> A

3 -> A

4 -> A

5 -> A

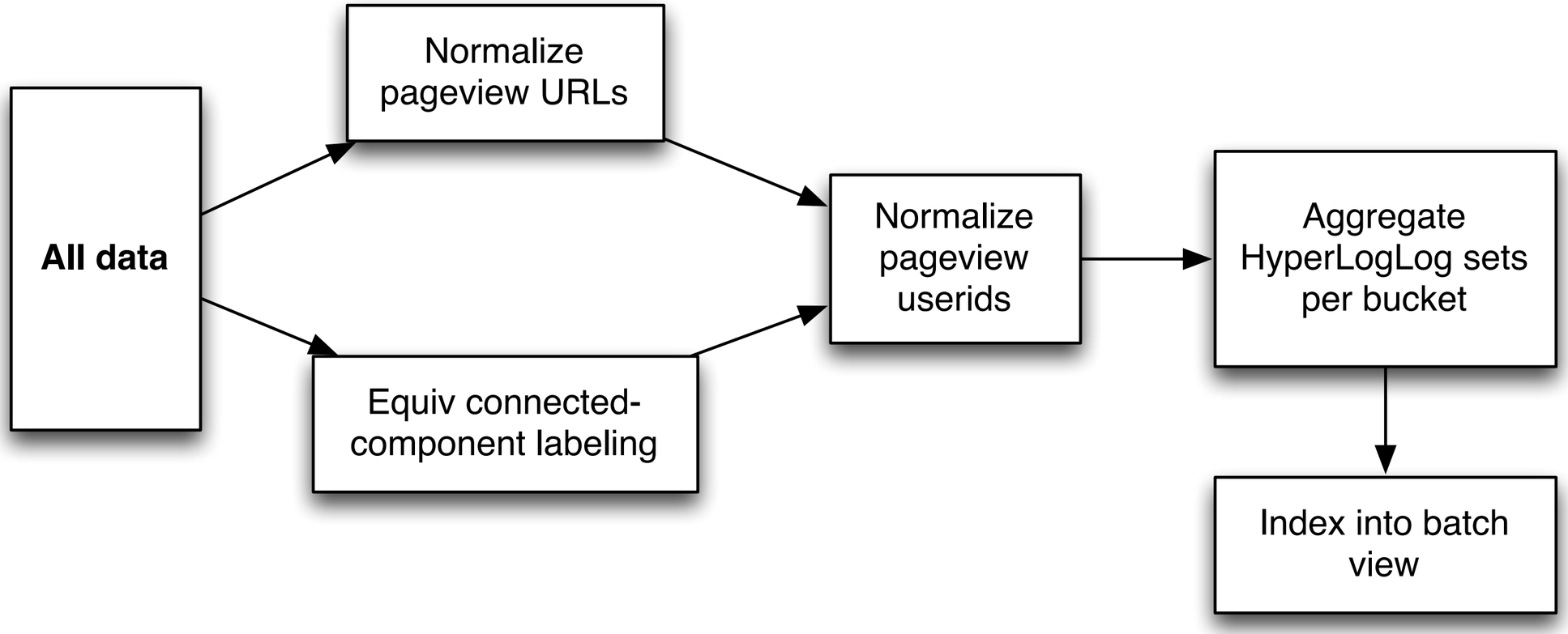
11 -> A

6 -> B

7 -> B

9 -> B

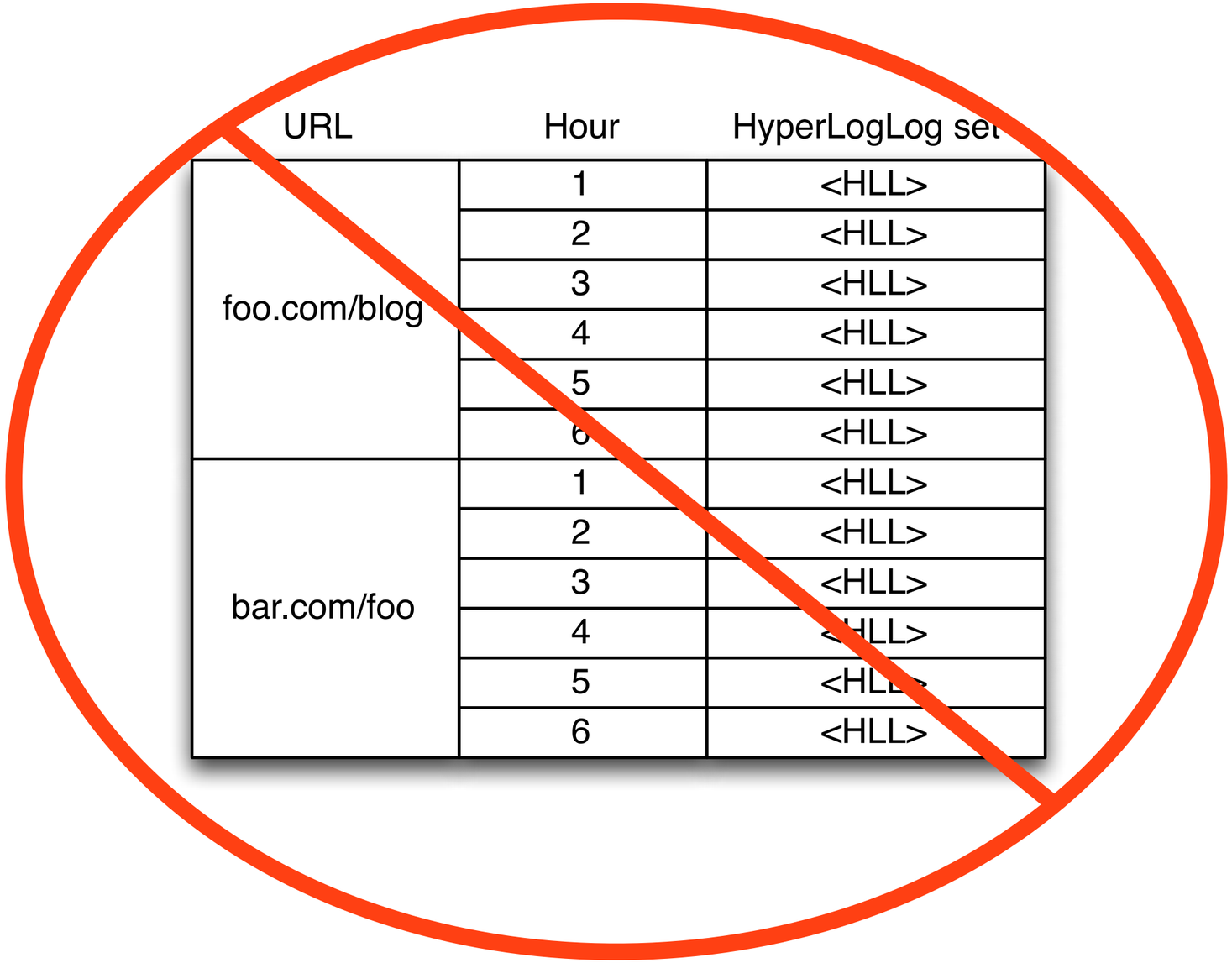
Batch workflow



Realtime workflow

The equiv problem

The equiv problem

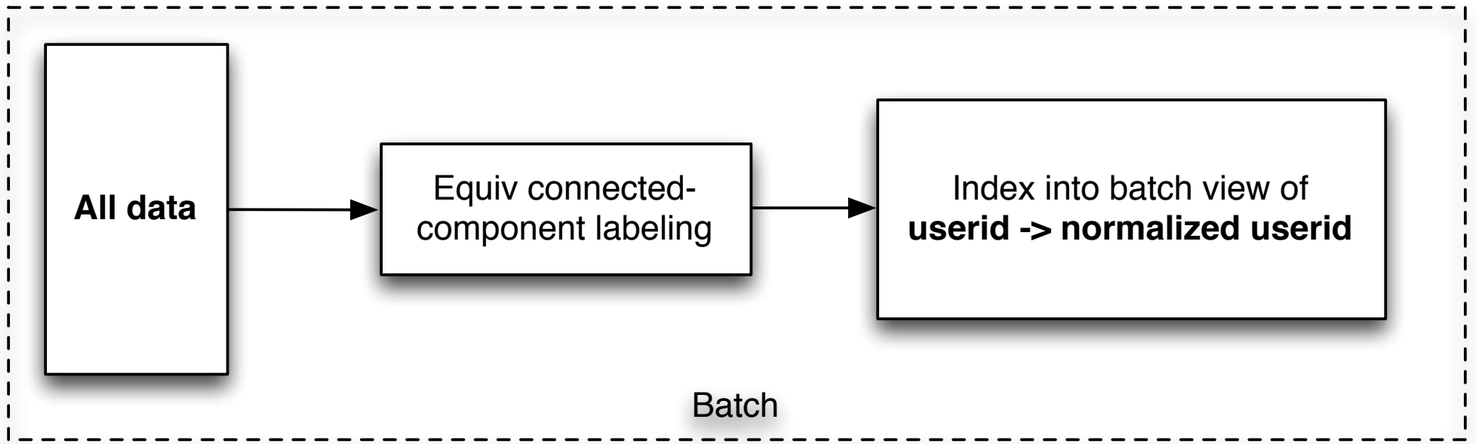


URL	Hour	HyperLogLog set
foo.com/blog	1	<HLL>
	2	<HLL>
	3	<HLL>
	4	<HLL>
	5	<HLL>
	6	<HLL>
bar.com/foo	1	<HLL>
	2	<HLL>
	3	<HLL>
	4	<HLL>
	5	<HLL>
	6	<HLL>

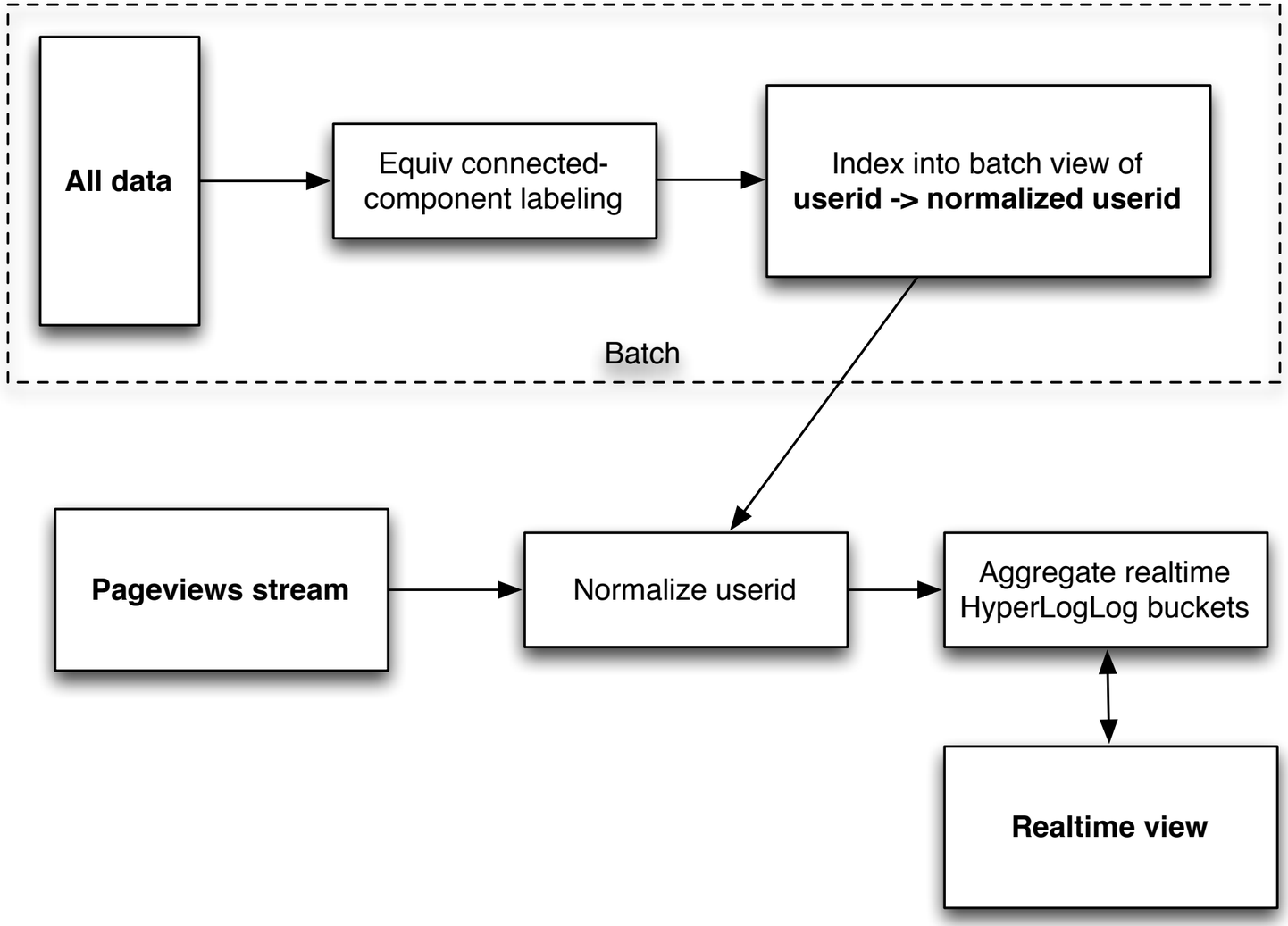
The equiv problem

URL	Hour	Visitors
foo.com/blog	1	A, B, C
	2	A
	3	D, E
	4	A
	5	B, C, D
	6	F, G
bar.com/foo	1	A
	2	B, C
	3	A, C
	4	D
	5	A, E
	6	A

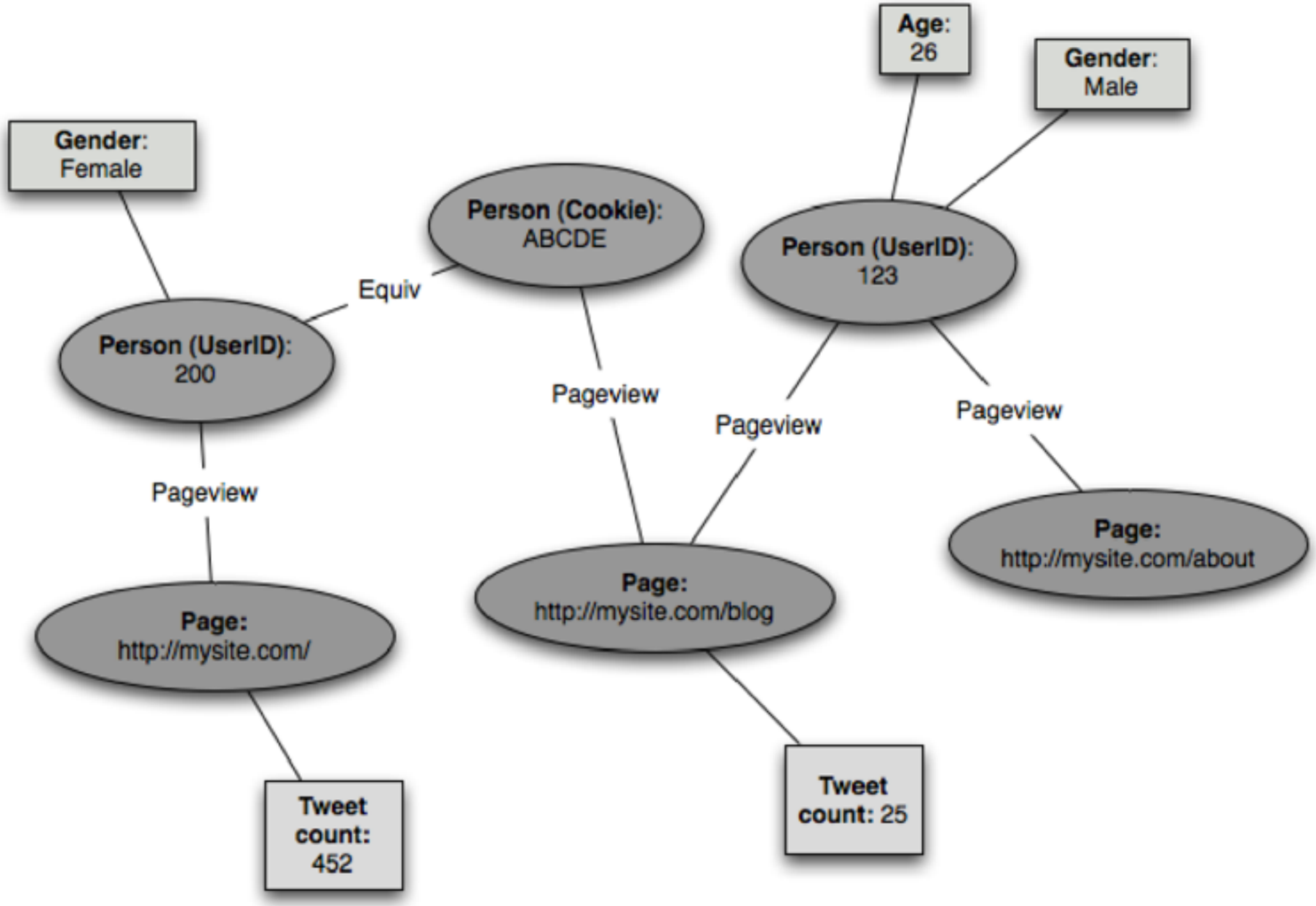
Solving the equiv problem



Realtime workflow



Schema


Age:
26
Gender:
Male
Gender:
Female
Person (Cookie):
ABCDE
Person (UserID):
123
Equiv
Person (UserID):
200
Pageview
Pageview
Pageview
Pageview
Page:
http://mysite.com/about
Page:
http://mysite.com/blog
Page:
http://mysite.com/
Tweet
count: 25
Tweet
count:
452

Schema

```
union PersonID {
  1: string cookie;
  2: i64 user_id;
}

union PageID {
  1: string url;
}
```

Schema

```
enum GenderType {
  MALE = 1,
  FEMALE = 2
}

union PersonPropertyValue {
  1: string full_name;
  2: GenderType gender;
}

struct PersonProperty {
  1: required PersonID id;
  2: required PersonPropertyValue property;
}
```

Schema

```
struct EquivEdge {
  1: required PersonID id1;
  2: required PersonID id2;
}

struct PageViewEdge {
  1: required PersonID person;
  2: required PageID page;
}
```

Schema

```
union DataUnit {
  1: PersonProperty person_property;
  2: EquivEdge equiv;
  3: PageViewEdge page_view;
}
```

Schema

```
struct Data {
  1: required i32 timestamp_secs;
  2: required DataUnit dataunit;
}
```

Conclusions from example

- Avoids every single insane complexity I talked about

- Powerful to be able to extract more out of a piece of data the longer you have it

- Eventual accuracy is a super useful technique

- Schemas can be useful and non-painful

- A Lambda Architecture is fundamentally easy to extend with new views/queries

Thank you