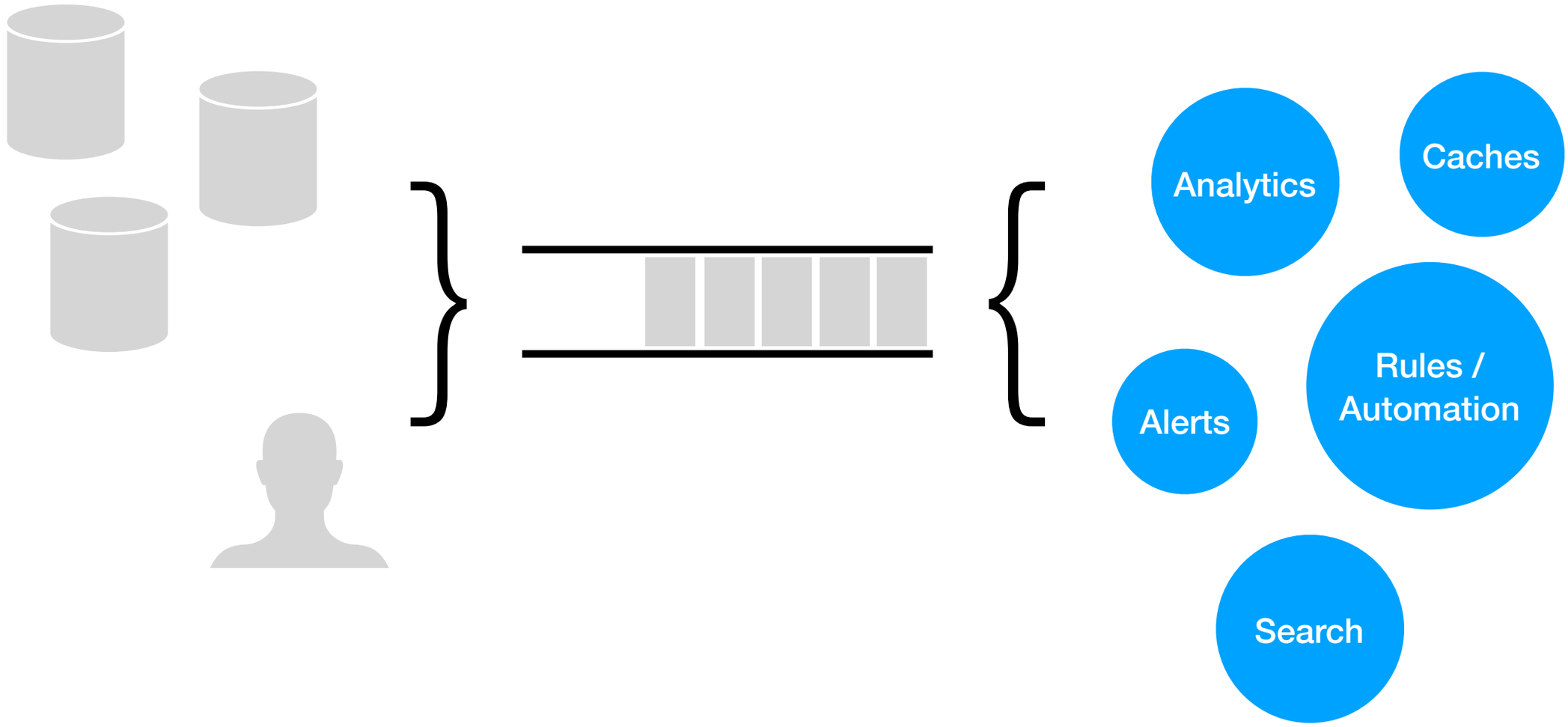
Declarative Differential Dataflows

Nikolas Göbel



Producers **Log** **Consumers**

Benefits

+ decoupled systems (coordinate w/ data, stateless consumers)

+ operational scalability (O(n) pipes instead of O(n^2))

+ no data silos

+ proven in practice (Kafka, AWS Kinesis)

Challenges

- redundant filter logic in consumers (=> “Streaming SQL”)

- change detection (*diffing*) handled by consumers

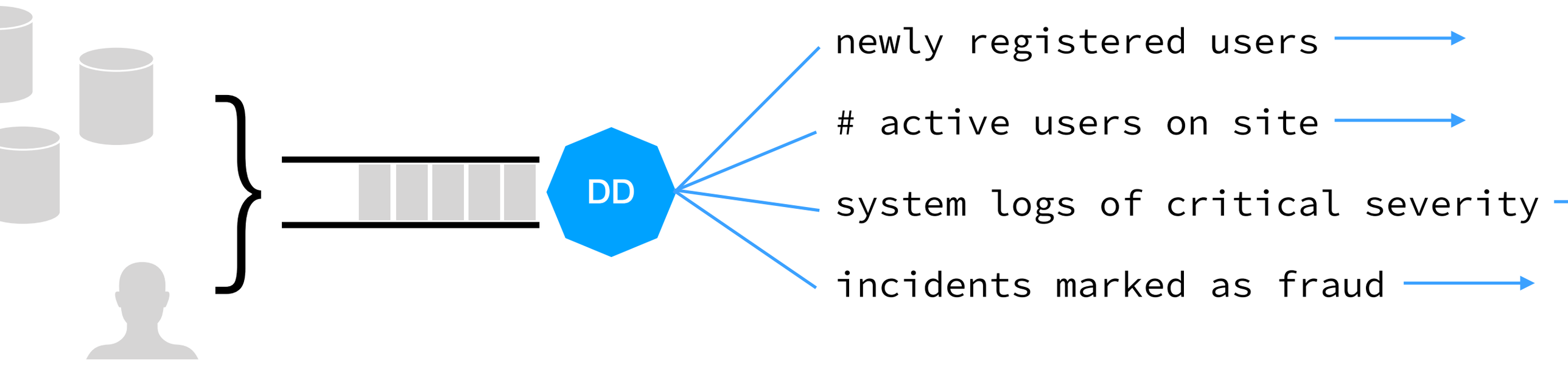
- have to keep up w/ producers (=> “Consumer Groups”)

- weak processing primitives (no iteration)

Rephrased problem: Efficiently routing data change to many, heterogenous clients.

Approach

- consumers declare interest in data via Datalog queries
- utilise Differential Dataflow to run queries over the log
- outputs already diffs, ideal for maintaining derived views



Differential Dataflow

```
/// BFS

let nodes = roots.map(|x| (x, 0));

nodes.iterate(|inner| {

  let edges = edges.enter(&inner.scope());
  let nodes = nodes.enter(&inner.scope());

  inner.join_map(&edges, |_k,l,d| (*d, l+1))
       .concat(&nodes)
       .group(|_, s, t| t.push((*s[0].0, 1)))
})
```

Challenge I:
Synthesise data flows w/o compilation?

```
/// BFS

let nodes = roots.map(|x| (x, 0));

nodes.iterate(|inner| {

    let edges = edges.enter(&inner.scope());
    let nodes = nodes.enter(&inner.scope());

    inner.join_map(&edges, |_k,l,d| (*d, l+1))
         .concat(&nodes)
         .group(|_, s, t| t.push((*s[0].0, 1)))
})
```

Differential collections are statically typed and too flexible for synthesis.

Solution:
Unified data model

```
;; unified data model

[123 :name “Alice”]
[123 :friend 456]
[456 :age 28]
[456 :authenticated? true]
 …
```

```
;; transactions

[[-1 456 :age 28]
 [+1 456 :age 29]
 … ]
```

Solution:
Unified data model

```
;; unified data model

[123 :name “Alice”]
[123 :friend 456]
[456 :age 28]
[456 :authenticated? true]
...
```

Value

Entity

Attribute

```
;; transactions

[[-1 456 :age 28]
 [+1 456 :age 29]
 ...]
```

Retraction

Assertion

Challenge II:
Differential-compatible Query Language

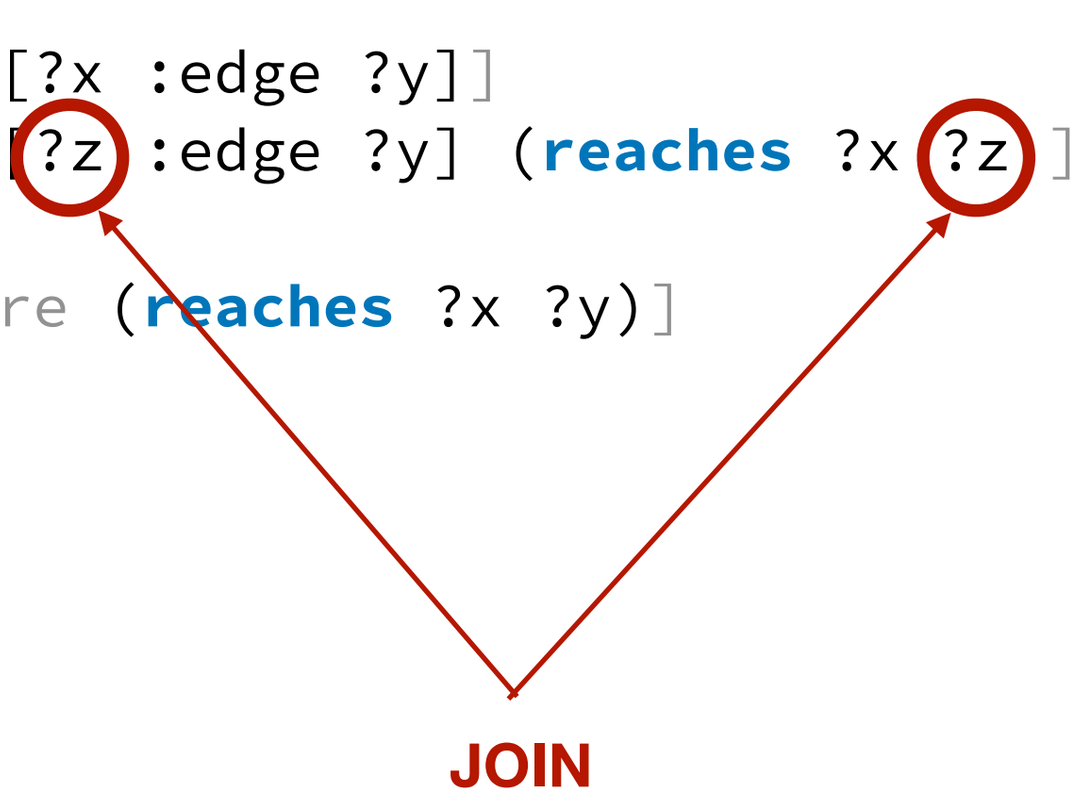
```
/// BFS

let nodes = roots.map(|x| (x, 0));

nodes.iterate(|inner| {

  let edges = edges.enter(&inner.scope());
  let nodes = nodes.enter(&inner.scope());

  inner.join_map(&edges, |_k,l,d| (*d, l+1))
       .concat(&nodes)
       .group(|_, s, t| t.push((*s[0].0, 1)))
})
```

Datalog

```
;; Reachability

[(reaches ?x ?y) [?x :edge ?y]]

[(reaches ?x ?y) [?z :edge ?y] (reaches ?x ?z)]

[:find ?x ?y :where (reaches ?x ?y)]
```

Datalog

```
;; Reachability

[(reaches ?x ?y) [?x :edge ?y]]
[(reaches ?x ?y) [?z :edge ?y] (reaches ?x ?z)]

[:find ?x ?y :where (reaches ?x ?y)]
```



Datalog

UNION

```
;; Reachability

[(reaches ?x ?y) [?x :edge ?y]]
[(reaches ?x ?y) [?z :edge ?y] (reaches ?x ?z)]

[:find ?x ?y :where (reaches ?x ?y)]
```

Datalog

```
;; Reachability

[(reaches ?x ?y) [?x :edge ?y]]
[(reaches ?x ?y) [?z :edge ?y] (reaches ?x ?z)]

[:find ?x ?y :where (reaches ?x ?y)]
```

ITERATE

Challenge III:
Constant bindings

```
;; all nodes reachable starting from node 100

[(reaches ?x ?y) [?x :edge ?y]]

[(reaches ?x ?y) [?z :edge ?y] (reaches ?x ?z)]

[:find ?y :where (reaches 100 ?y)]
```

Solution: Input transform

```
;; all nodes reachable starting from node 100

[:find ?y :where (reaches 100 ?y)]
```

```
[:find ?y
 :in ?in0
 :where (reaches ?in0 ?y)]
```

```
// ?in0
scope.new_collection_from([[100]])
```

Challenge IV: Negation

```
;; unbound negation

[:find ?e
 :where
 (not [?e :name "Alice"])]
```

```
;; tautologies

[:find ?e
 :where
 (or [?e :name “Alice”]
     (not [?e :name “Alice”]))]
```

```
;; contradictions

[:find ?e
 :where
 (and [?e :name “Alice”]
      (not [?e :name “Alice”]))]
```

Solution?
Negation ~ Set Difference

```
;; unbound negation

[:find ?e
 :where
 (not [?e :name "Alice"])]
```

?

```
;; tautologies

[:find ?e
 :where
 (or [?e :name “Alice”]
     (not [?e :name “Alice”]))]
```

```
;; contradictions

[:find ?e
 :where
 (and [?e :name “Alice”]
      (not [?e :name “Alice”]))]
```

Solution??
Negation ~ Anti-Join

```
;; unbound negation

[:find ?e
 :where
 (not [?e :name "Alice"])]
```

```
;; tautologies

[:find ?e
 :where
 (or [?e :name “Alice”]
     (not [?e :name “Alice”]))]
```

```
;; contradictions

[:find ?e
 :where
 (and [?e :name “Alice”]
      (not [?e :name “Alice”]))]
```

Solution???
Anti-Join against all tuples, push not-clauses down.

```
;; unbound negation

[:find ?e
 :where
 (not [?e :name "Alice"])]
```

```
;; tautologies

[:find ?e
 :where
 (or [?e :name “Alice”]
     (not [?e :name “Alice”]))]
```

```
;; contradictions

[:find ?e
 :where
 (and [?e :name “Alice”]
      (not [?e :name “Alice”]))]
```

Set Difference in Differential?

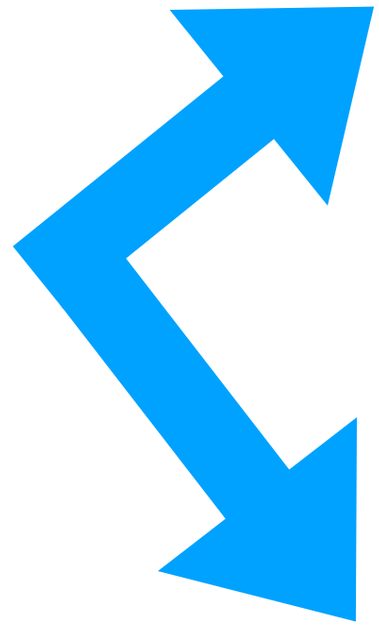
```
;; given unbound query…

[:find ?e
 :where
 (not [?e :name "Alice"])]
```

```
;; …and some data

[[+1 100 :name “Alice"]]
```

Set Difference in Differential?

```
;; given unbound query…

[:find ?e
 :where
 (not [?e :name "Alice"])]
```

∅



```
;; …and some data

[[+1 100 :name “Alice"]]
```

```
[[[{“Eid": 100}], -1]]
```

Challenge V: Union-compatibility

```
;; union w/ extra symbols

[:find ?x ?y
 :where
 (or [?x :edge ?y]
     (and [?x :edge ?z]
          [?z :edge ?y]))]
```

```
;; negation (if ~ set difference)

[:find ?e
 :where
 (or [?e :name “Alice”]
     (not
      [?p :purchase/user ?e]
      [?p :purchase/year 2018]))]
```

Solution:
Explicit unification scope
(add a projection)

```
;; union w/ extra symbols

[:find ?x ?y
 :where
 (or-join [?x ?y]
   [?x :edge ?y]
    (and [?x :edge ?z]
         [?z :edge ?y]))]
```

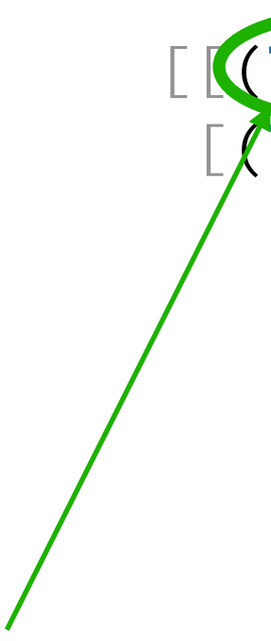
```
;; negation (if ~ set difference)

[:find ?e
 :where
 (or [?e :name “Alice”]
     (not-join [?e]
       [?p :purchase/user ?e]
       [?p :purchase/year 2018]))]
```

Non-Challenge: Recursion

```
[:find ?x ?y :where (label ?x ?y)]

[[(label ?x ?y) [?x :label ?y]]
 [(label ?x ?y) [?z :edge ?y] (label ?x ?z)]]
```

Non-Challenge: Recursion

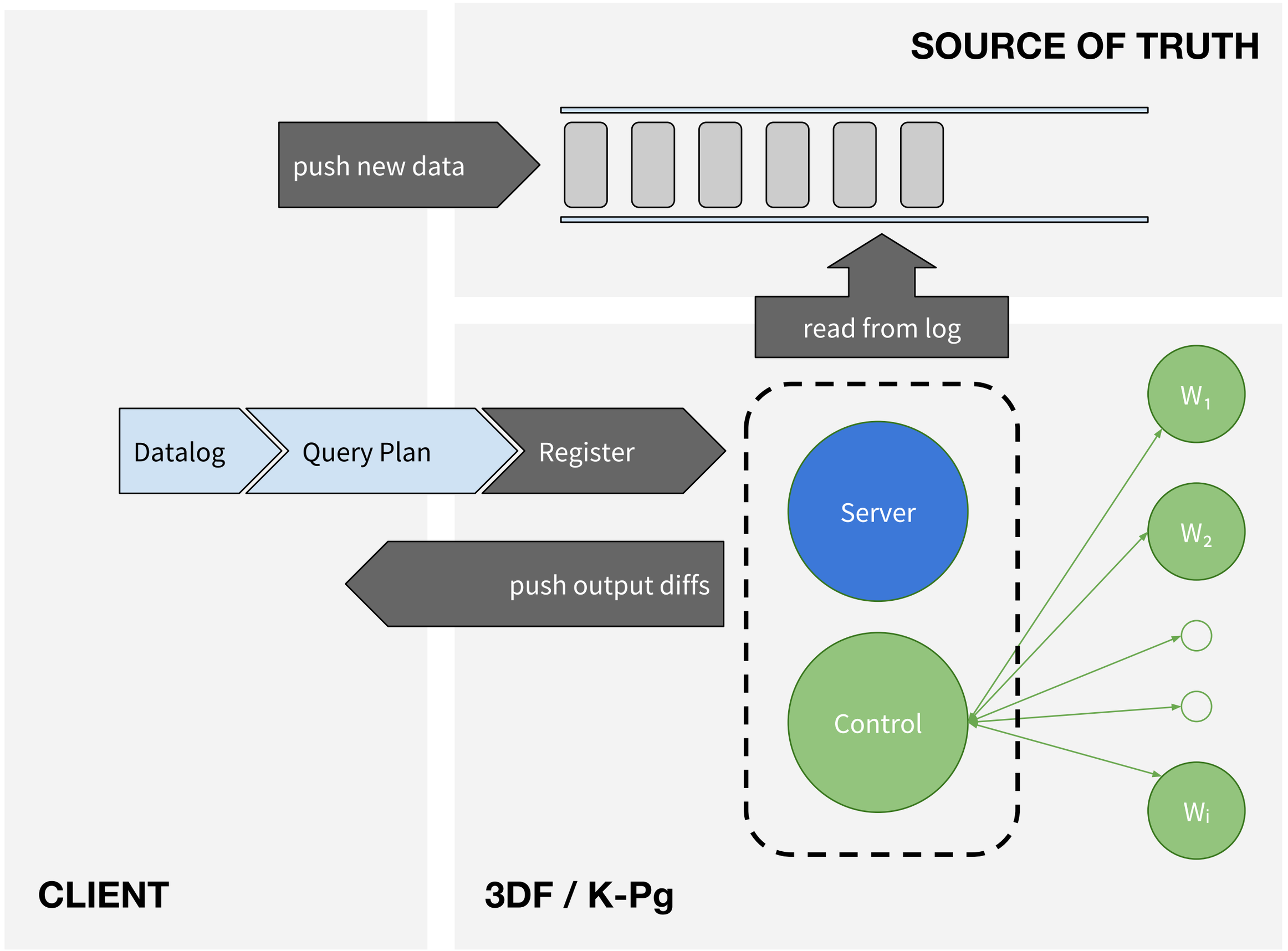
```
[:find ?x ?y :where (label ?x ?y)]

[[(label ?x ?y) [?x :label ?y]]
 [(label ?x ?y) [?z :edge ?y] (label ?x ?z)]]
```



everything else stays the same!

```
struct NamedRelation {
    variable: Variable<Vec<Value> …>
    tuples: Collection<Vec<Value> …>
}
```

3DF in Context


SOURCE OF TRUTH
push new data
read from log
Datalog
Query Plan
Register
push output diffs
Server
Control
W_1
W_2
W_i
CLIENT
3DF / K-Pg

Evaluation: Label Propagation

```
[:find ?x ?y :where (label ?x ?y)]

[[(label ?x ?y) [?x :label ?y]]
 [(label ?x ?y) [?z :edge ?y] (label ?x ?z)]]
```

System	Cores	HTTPD (169M)	POSTGRES (639M)
SociaLite	4	4 hrs	OOM
Graspan	4	678s	8628s
Hand-rolled	1	10.9s	37s
our system	1	151s	440s

Conclusions

with querying and diffing handled by Differential…

+ simple, declarative consumers

+ consumer performance must match throughput of relevant novelty, not throughput of overall log

+ Differential's powerful processing primitives available for further processing