DBToaster

Higher-Order Delta Processing for Dynamic, Frequently Fresh Views

Yanif Ahmad — Johns Hopkins

Oliver Kennedy — University at Buffalo

Christoph Koch — EPFL

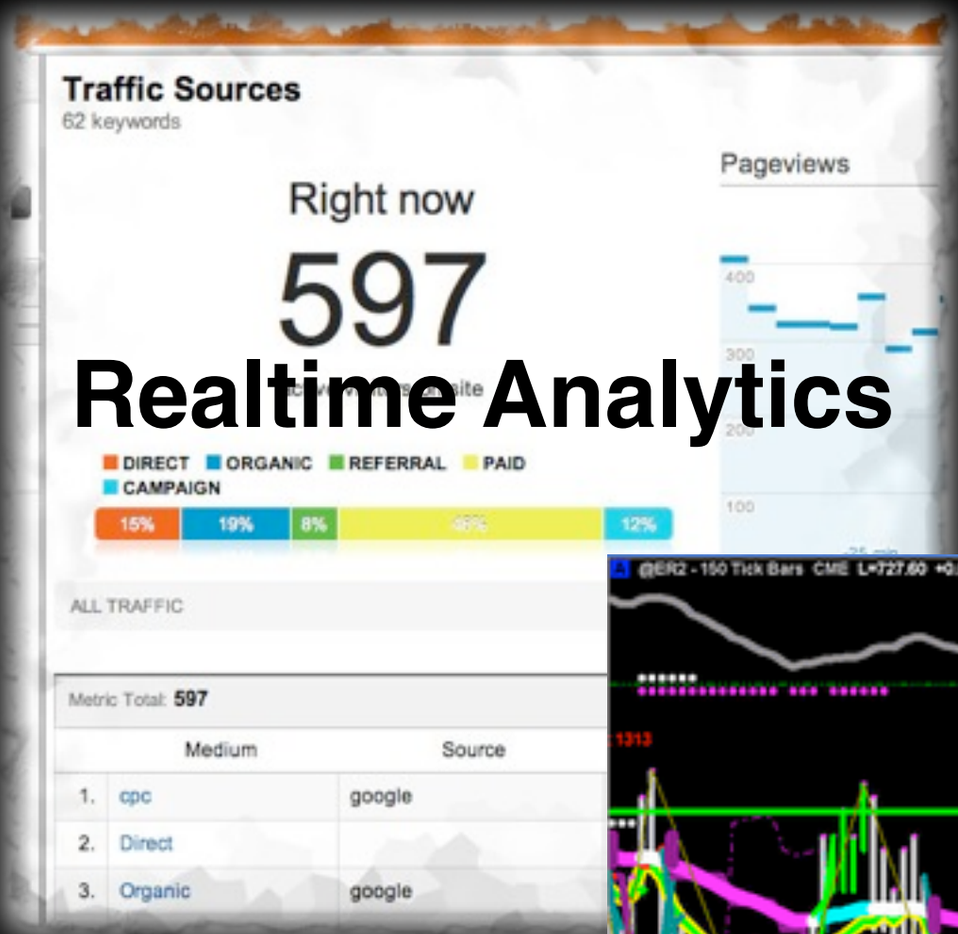
Milos Nikolic — EPFL

Realtime Monitoring Programs…

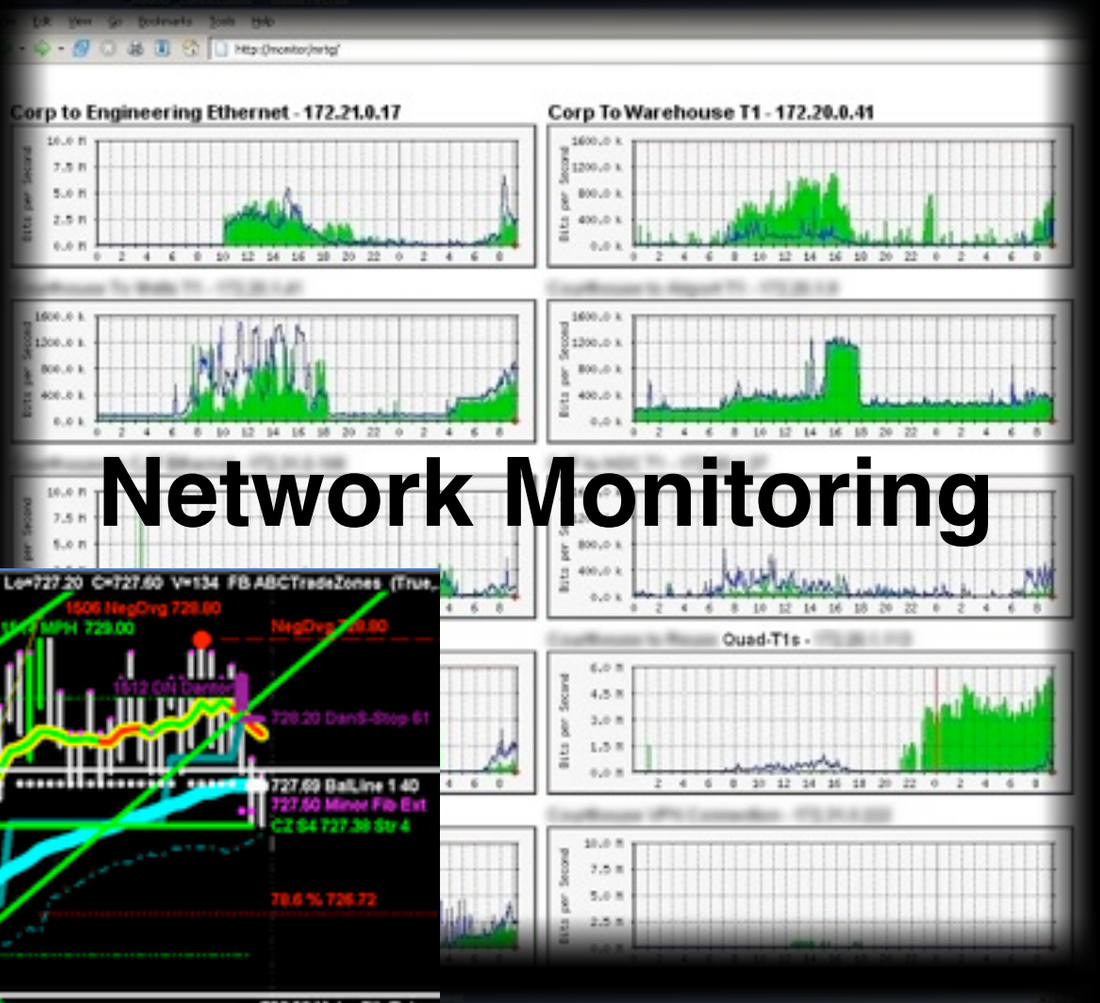
…Monitor The State of the World

…React to Conditions in that State

Realtime Monitoring Programs are Everywhere

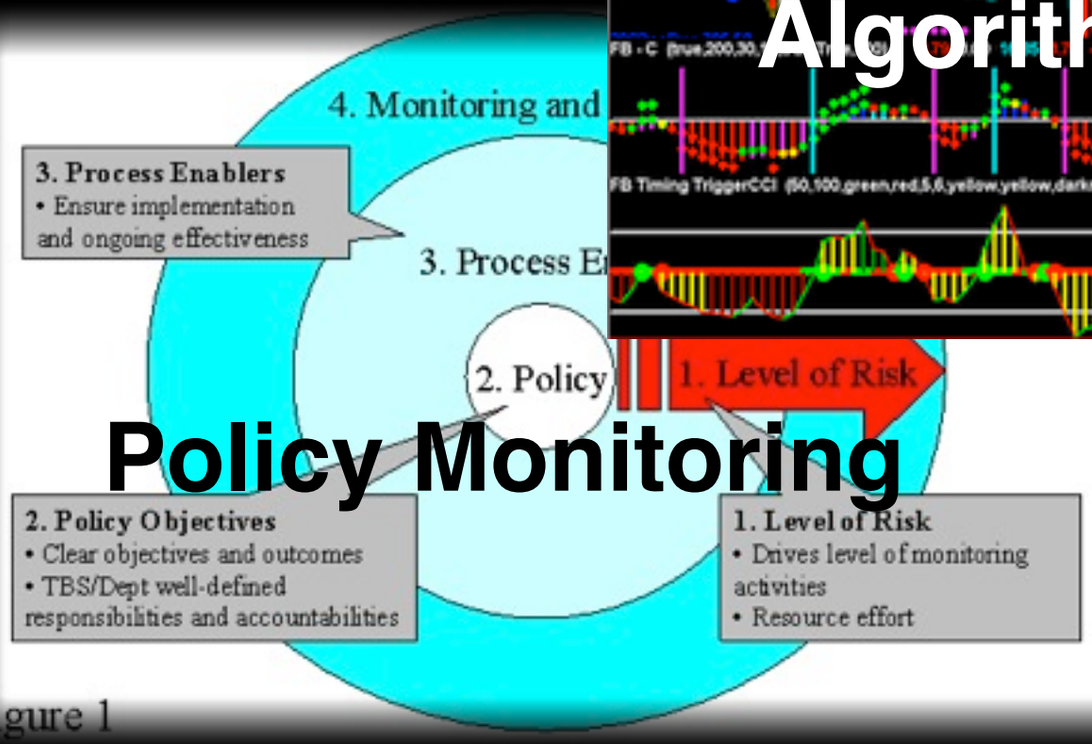


Realtime Analytics

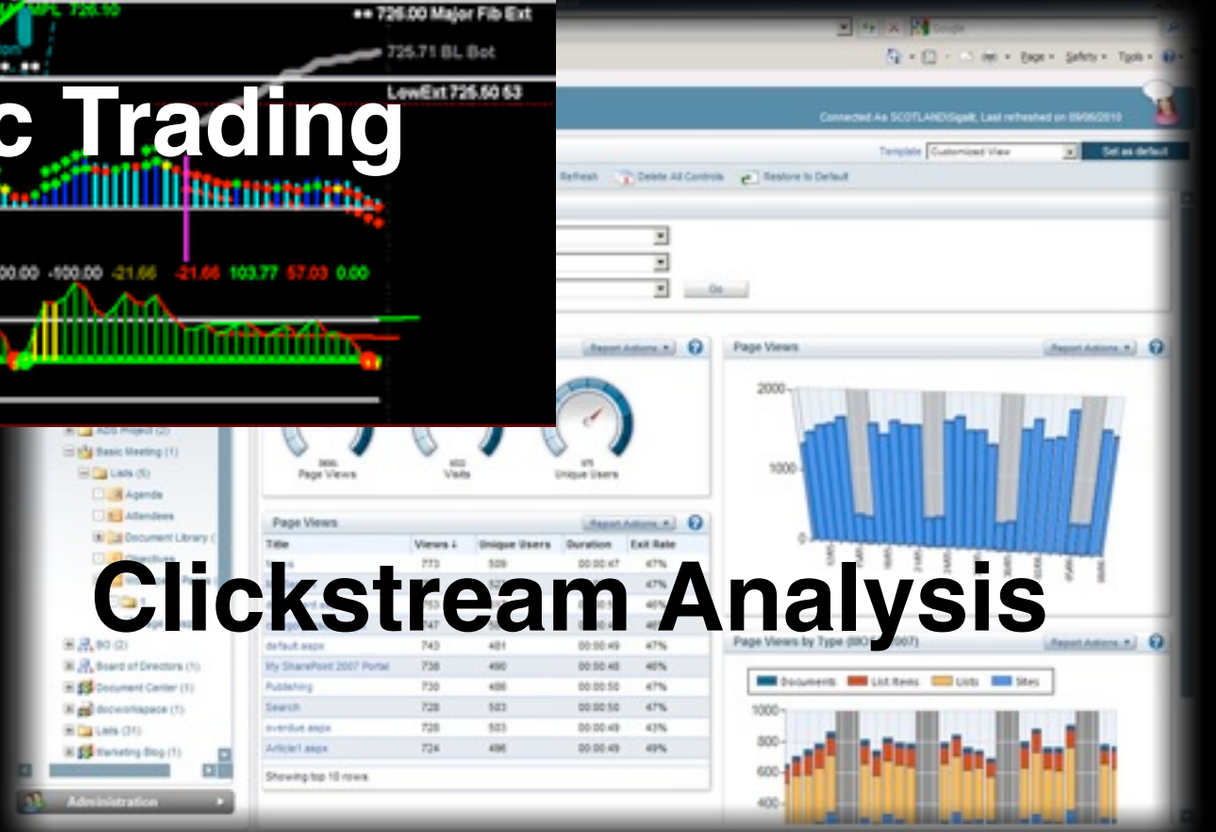


Network Monitoring

Algorithmic Trading

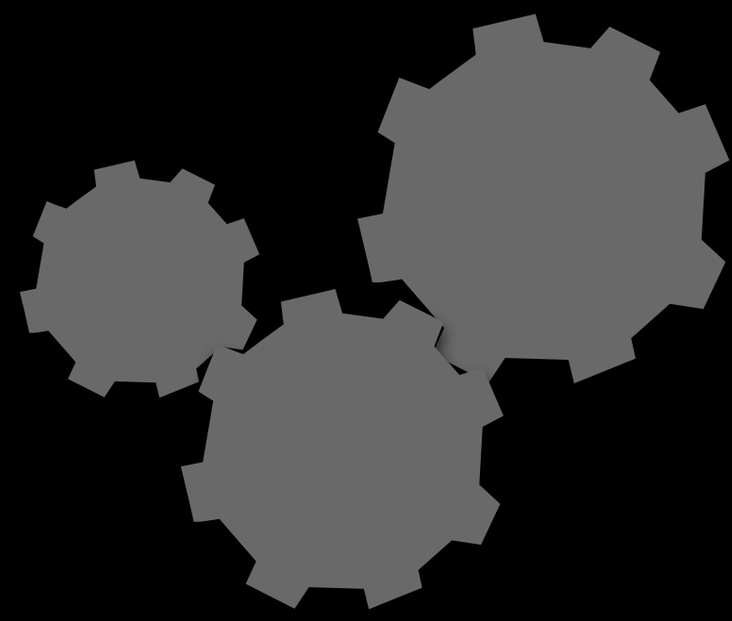


Policy Monitoring

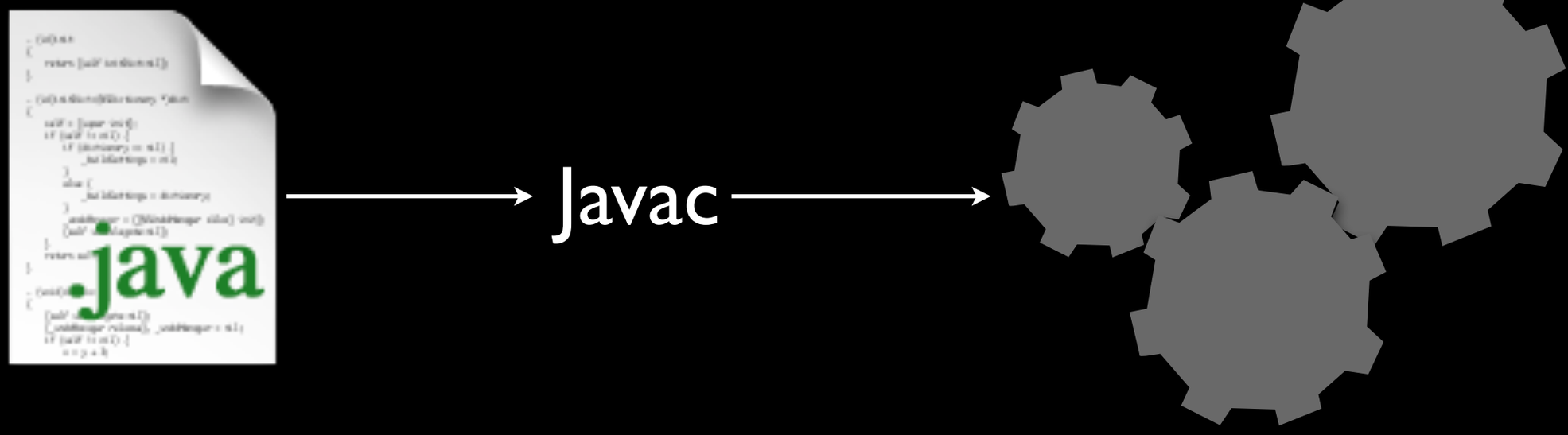


Clickstream Analysis

Monitoring Programs

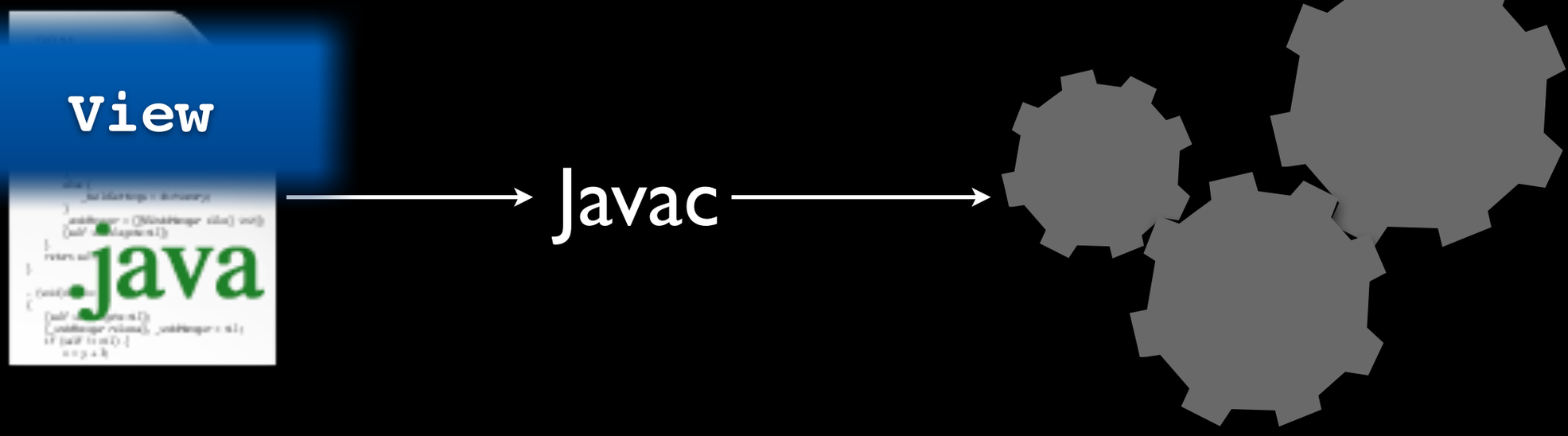


Monitoring Programs



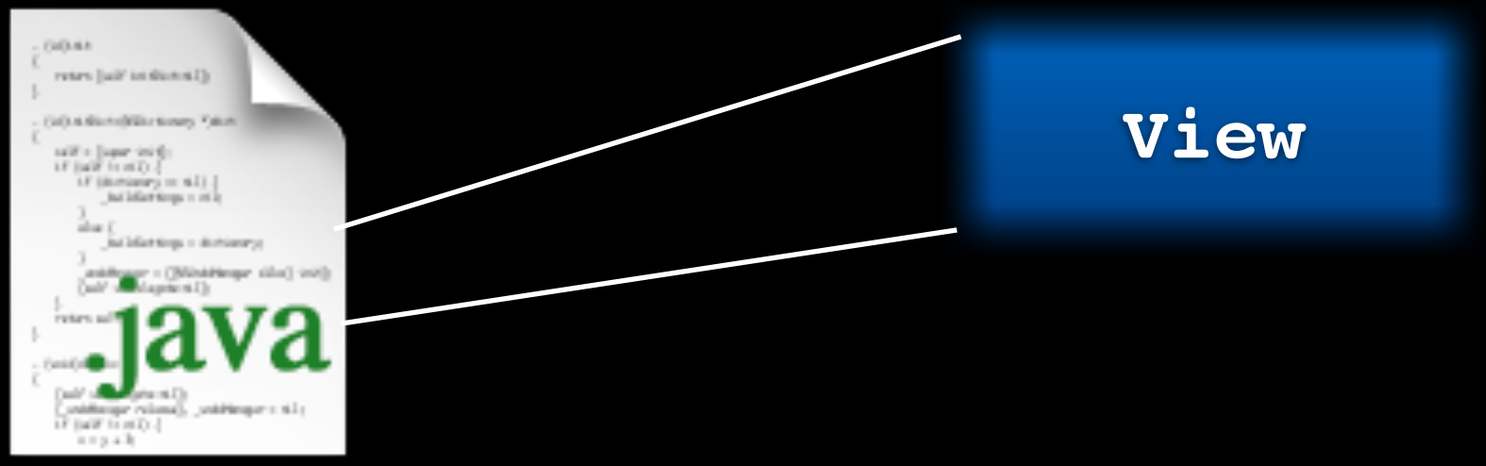
Problem: People write monitoring programs by hand

Monitoring Programs



Problem: People write monitoring programs by hand

Monitoring Programs



Monitoring Programs

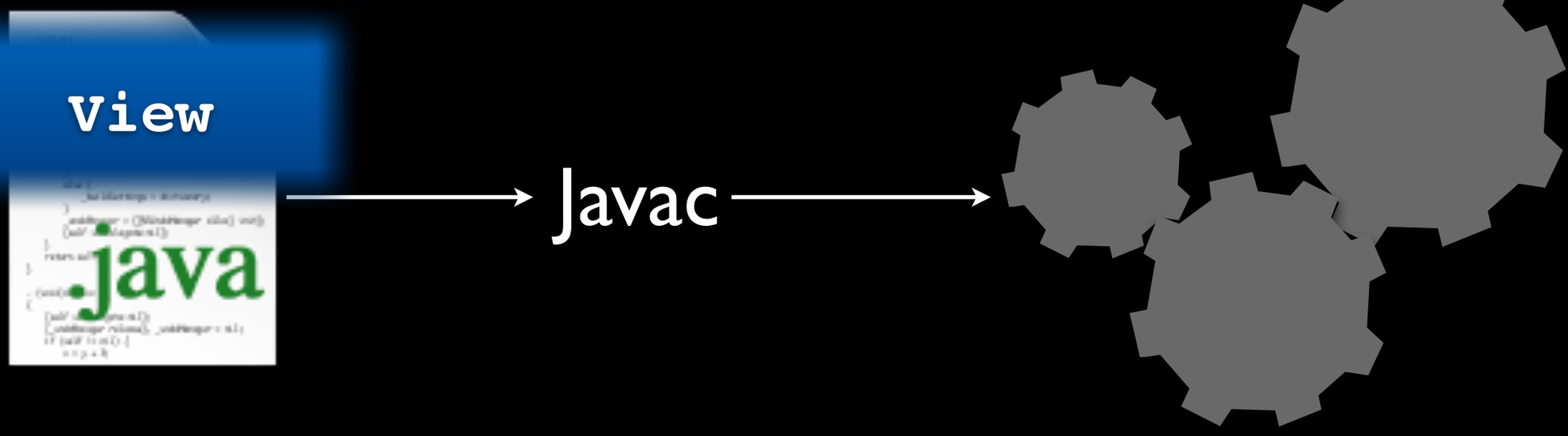
View

- An Aggregate Representation of the State of the World
- Maintained in Realtime as the State of the World Changes
- Needs to React to Changes In the World Quickly

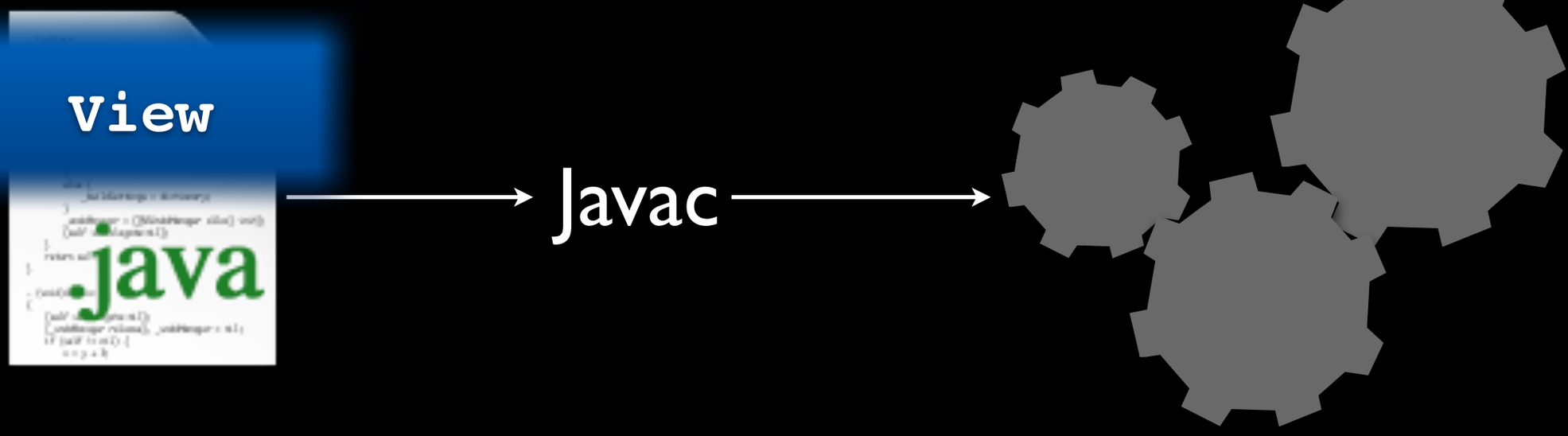
Not just Views

Frequently Fresh Views

Monitoring Programs

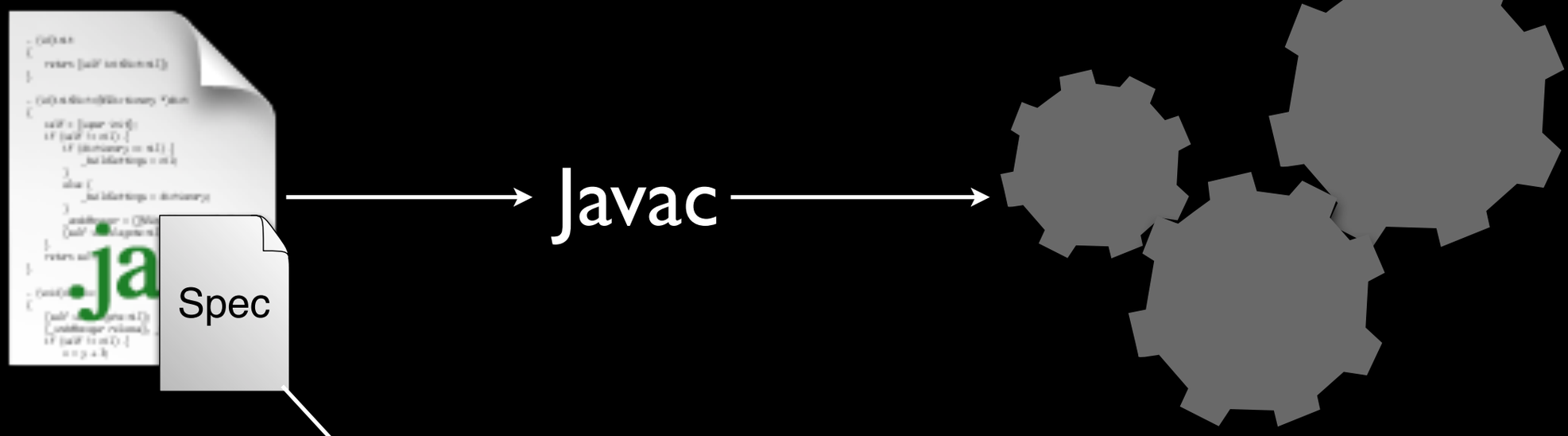


Monitoring Programs



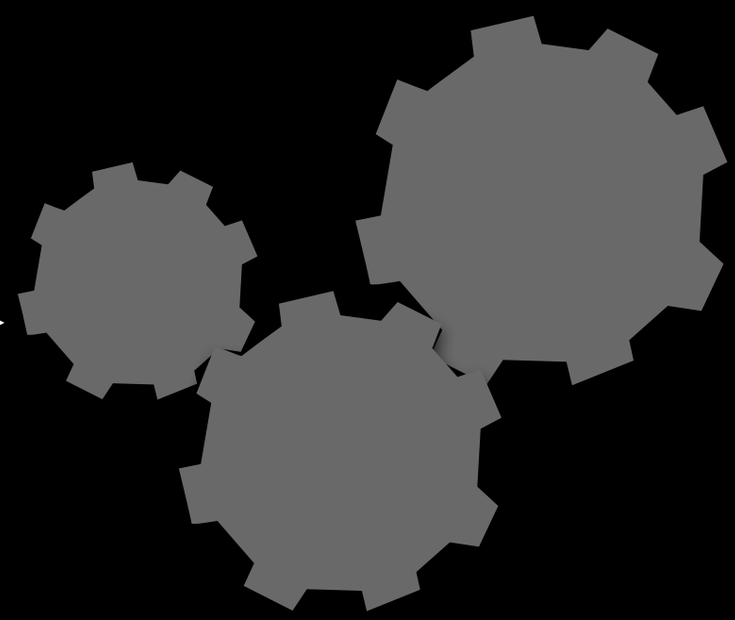
(The Current State of the Art)

Monitoring Programs



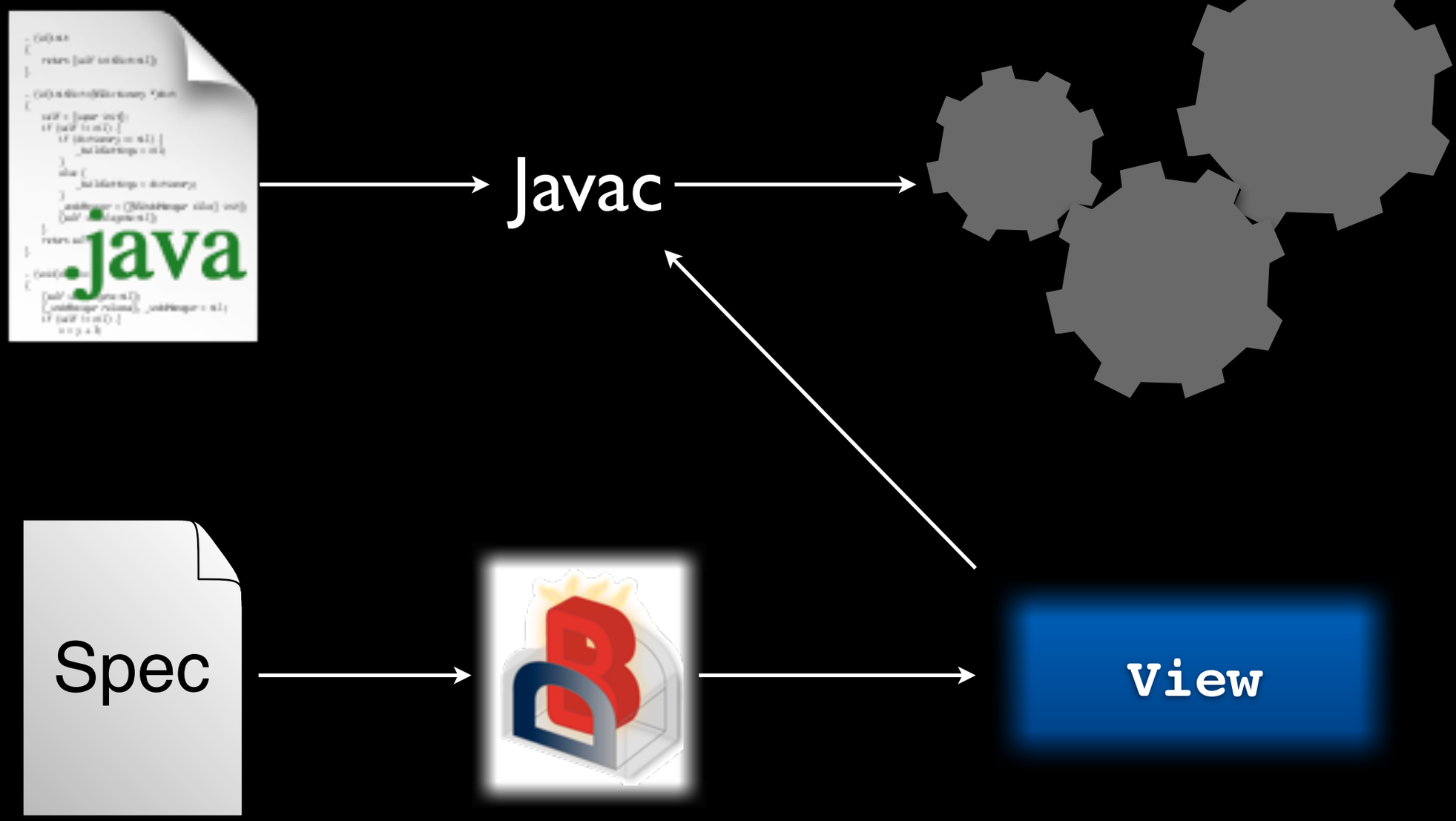
Monitoring Programs

Javac



Spec

Monitoring Programs



The DBToaster Compiler

The Viewlet Transform

The Viewlet Transform

Use Auxiliary Views to Speed Up View Maintenance

The Viewlet Transform

Use Auxiliary Views to Speed Up View Maintenance

The Delta of a Query Can Be Materialized!

The Viewlet Transform

```
SELECT SUM(R.A * S.C)
FROM   R, S
WHERE  R.B = S.B
```

A Simple 2-Way Join Aggregate

The Viewlet Transform

```
q[] := SELECT SUM(R.A * S.C)
       FROM   R, S
       WHERE  R.B = S.B
```

A Simple 2-Way Join Aggregate

The Viewlet Transform

```
ON +R(∂A, ∂B):
 q[] +=  SELECT SUM(∂A * S.C)
         FROM   S
         WHERE  ∂B = S.B
```

Materialize and Incrementally Maintain The Query

The Viewlet Transform

```
ON +R(∂A, ∂B):
 q[] +=  SELECT SUM(∂A * S.C)
         FROM   S
         WHERE  ∂B = S.B
```

The Delta of the Original Query

Materialize and Incrementally Maintain The Query

The Viewlet Transform

```
ON +R(∂A, ∂B):
  q[] += ∂A * ( SELECT SUM(S.C)
                FROM   S
                WHERE  ∂B = S.B )
```

Optimize

The Viewlet Transform

```
ON +R(∂A, ∂B):
  q[] += ∂A * ( SELECT SUM(S.C)
                FROM   S
                WHERE  ∂B = S.B )
```

Optimize

The Viewlet Transform

```
ON +R(∂A, ∂B):
  q[] += ∂A * ( SELECT S.B, SUM(S.C)
                FROM   S
                GROUP BY S.B )[∂B]
```

Optimize

The Viewlet Transform

```
ON +R(∂A, ∂B):
  q[] += ∂A * ( SELECT S.B, SUM(S.C)
                FROM   S
                GROUP BY S.B )[∂B]
```

Just an Ordinary Query

Optimize

The Viewlet Transform

```
ON +R(∂A, ∂B):
  q[] += ∂A * mR[∂B]
```

```
mR[B] := SELECT S.B, SUM(S.C)
         FROM   S
         GROUP BY S.B
```

Extract and Materialize The Delta View

The Viewlet Transform

```
ON +R(∂A, ∂B):
  q[] += ∂A * mR[∂B]
```

A Hash Map (indexed by S.B)

```
mR[B] := SELECT S.B, SUM(S.C)
         FROM   S
         GROUP BY S.B
```

Extract and Materialize The Delta View

The Viewlet Transform

```
ON +R(∂A, ∂B):
  q[] += ∂A * mR[∂B]
```

```
ON +S(∂B, ∂C):
  mR[B] += SELECT ∂B,SUM(∂C)
```

Incrementally Maintain The Delta View

The Viewlet Transform

```
ON +R(∂A, ∂B):
  q[] += ∂A * mR[∂B]
```

```
ON +S(∂B, ∂C):
  mR[∂B] += ∂C
```

Optimize

The Viewlet Transform

```
ON +R(∂A, ∂B):
  q[] += ∂A * mR[∂B]
  mS[∂B] += ∂A
```

```
ON +S(∂B, ∂C):
  mR[∂B] += ∂C
  q[] += ∂C * mS[∂B]
```

Repeat for the Other Deltas of the Query

The Viewlet Transform

The Viewlet Transform

- Take the Deltas
  - Optimize and Materialize Them
    - Take the Deltas
      - Optimize and Materialize Them
        - ...

The Viewlet Transform

- Take the Deltas
  - Optimize and Materialize Them
    - Take the Deltas
      - Optimize and Materialize Them
        - ...

Performance

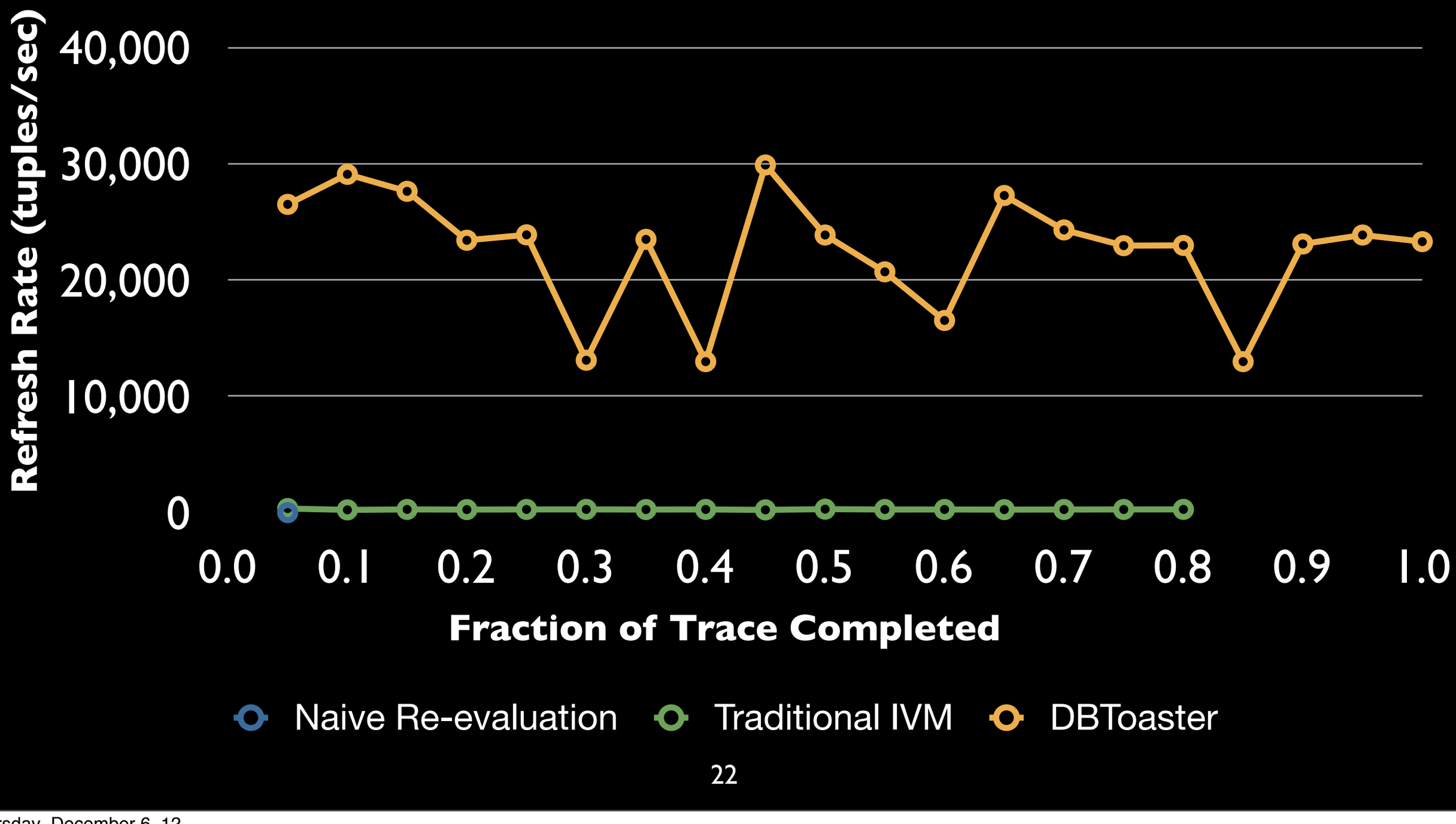
- TPC-H Workload
  - Simulated Realtime Data Warehouse
  - Update Stream Derived from TPC-H Gen

- Financial Benchmark
  - 24 hr Trace for an Actively Traded Stock.

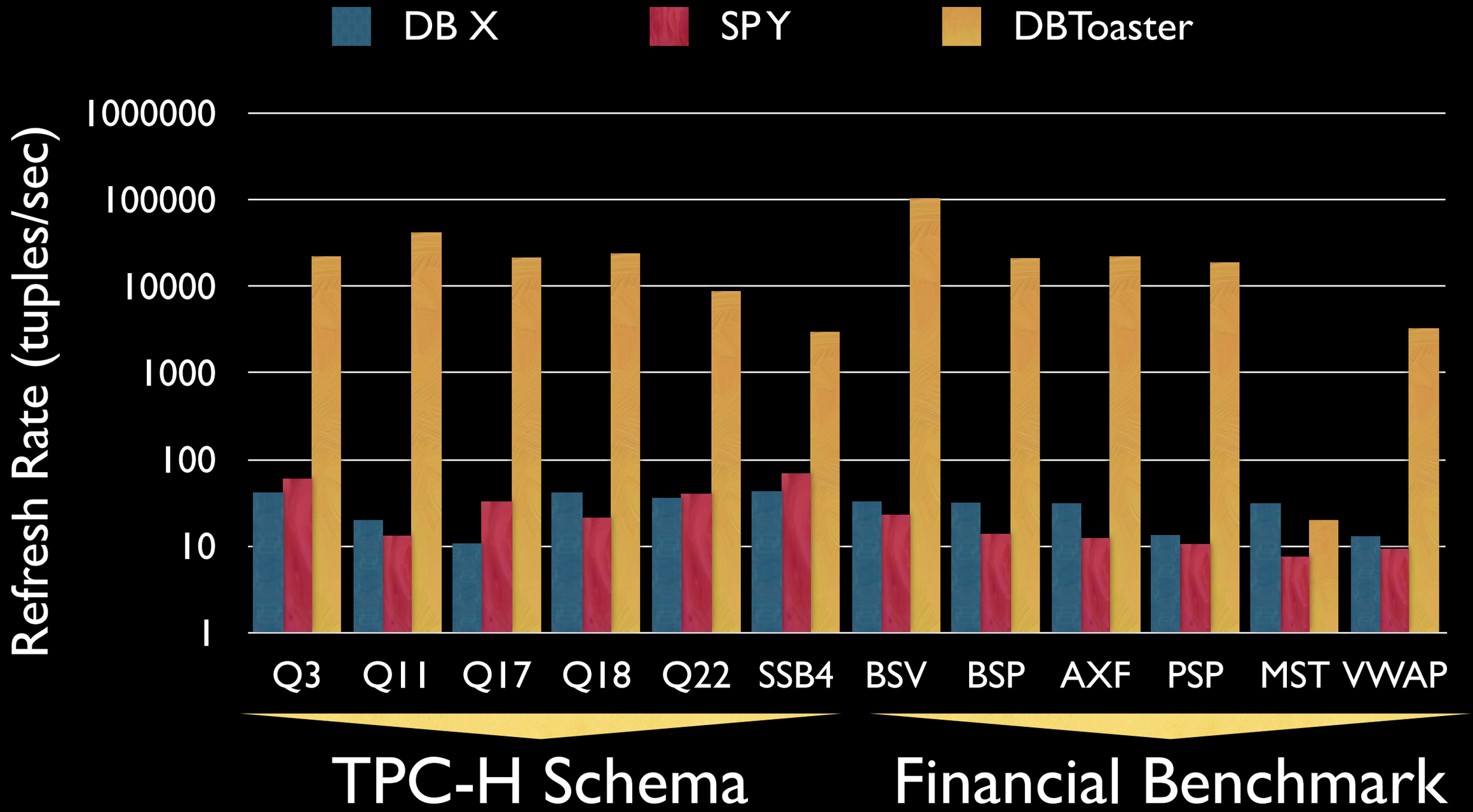
TPCH: Q3

Refresh Rate

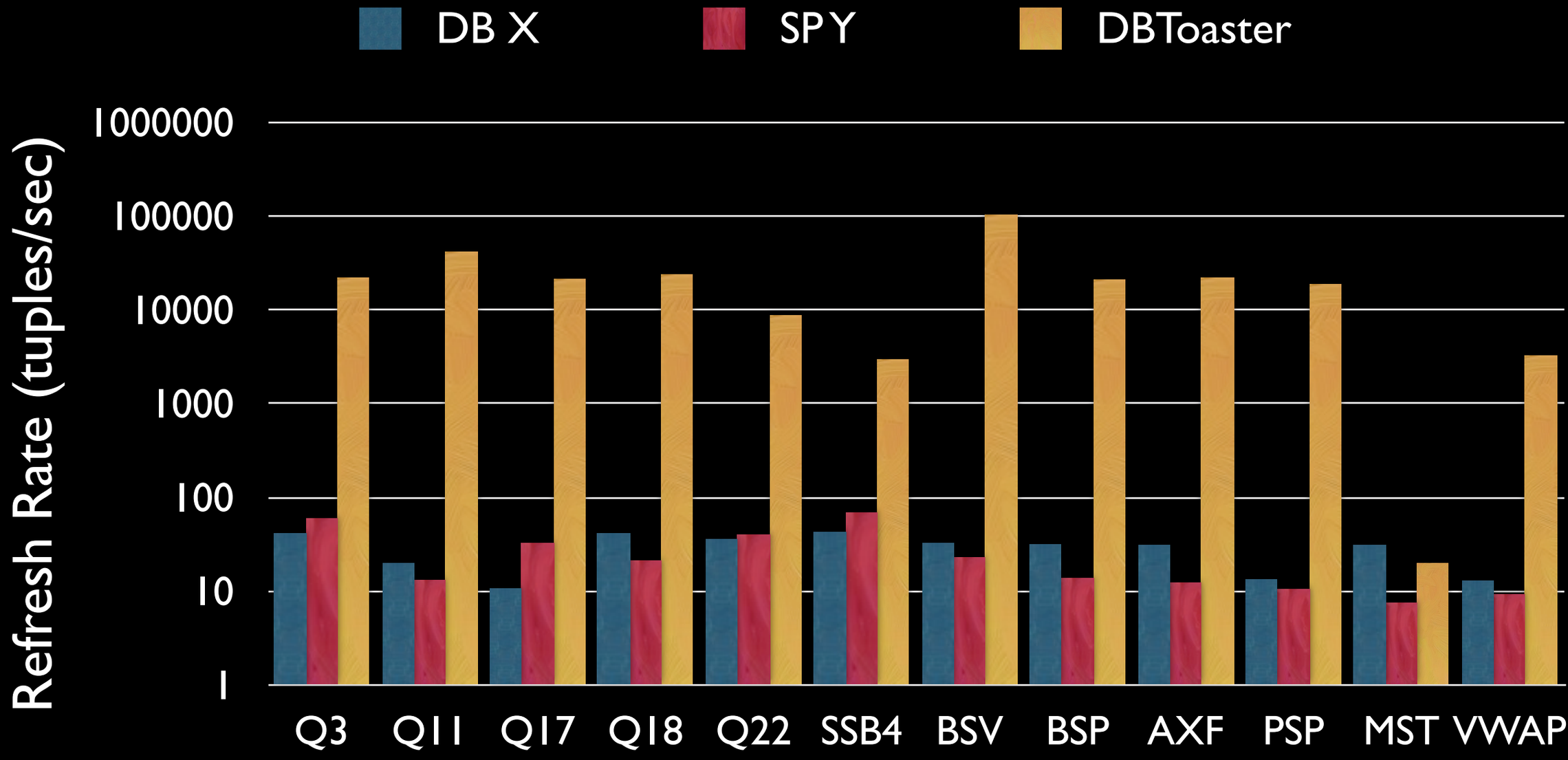
(3-Way Join)



DBToaster vs Commercial Engines



DBToaster vs Commercial Engines



DBToaster is consistently 3 OOM better!

Limitations of Commercial Systems

- OLTP IVM is not designed for aggregating Low-Latency/Single-Tuple Updates.
- OLTP IVM doesn't support our full query workload.
- Stream Processors are not designed for rapidly changing long-lived data.

Limitations of Commercial Systems

- OLTP IVM is not designed for aggregating Low-Latency/Single-Tuple Updates.
- OLTP IVM doesn't support our full query workload.
- Stream Processors are not designed for rapidly changing long-lived data.

DBToaster opens entirely new application domains!

Conclusions

- The Viewlet Transform generates auxiliary views that make incremental maintenance fast.
- Materializing only part of an auxilliary view can sometimes be faster.
- DBToaster is commonly 3 OoM faster than Commercial Systems.

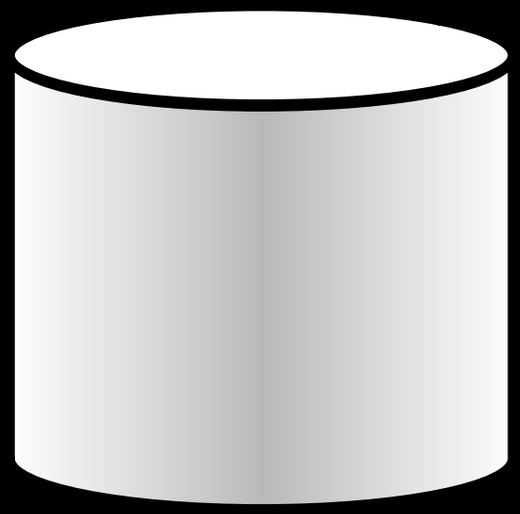
Download Now: http://www.dbtoaster.org

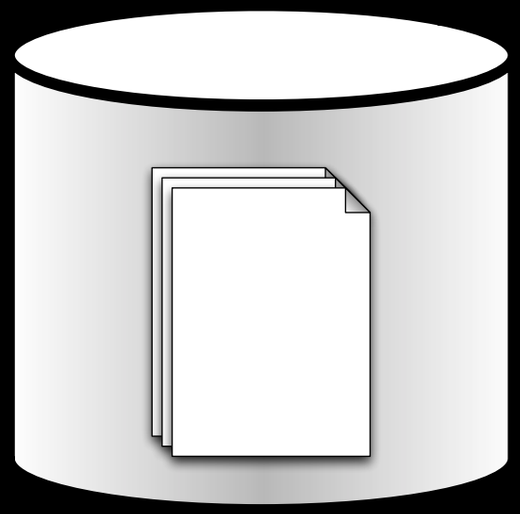
Cumulus: Hybrid Consistency Aggregation Queries

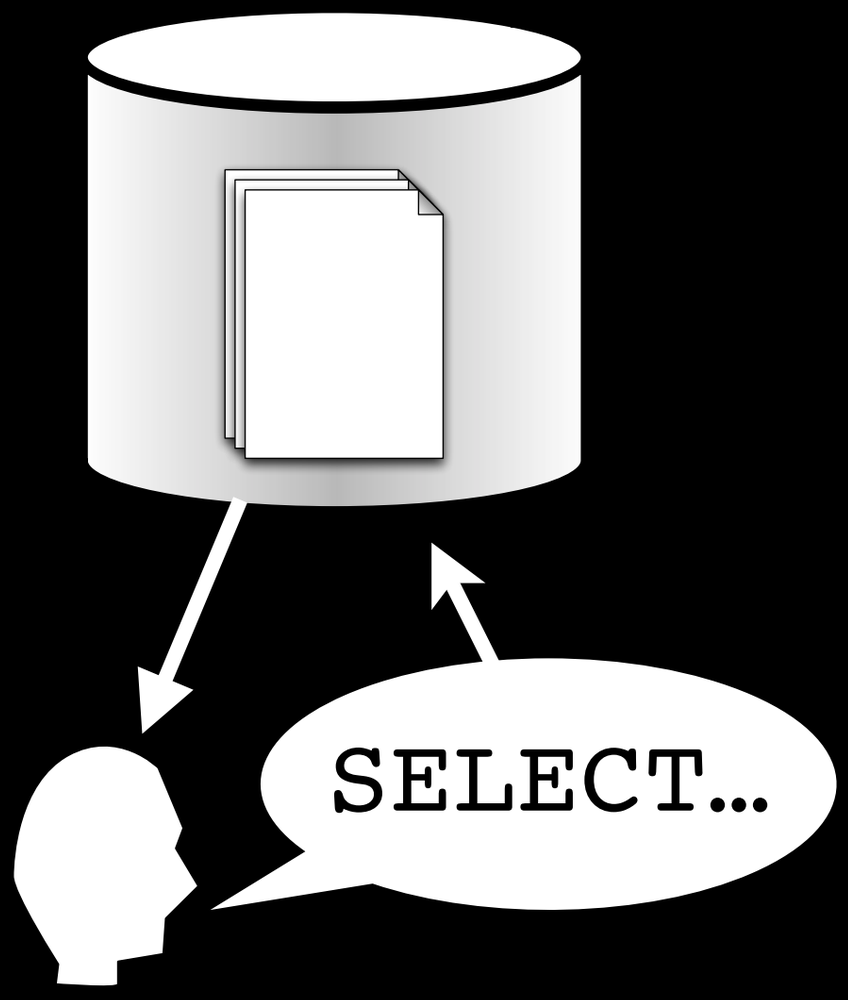
(in the cloud)

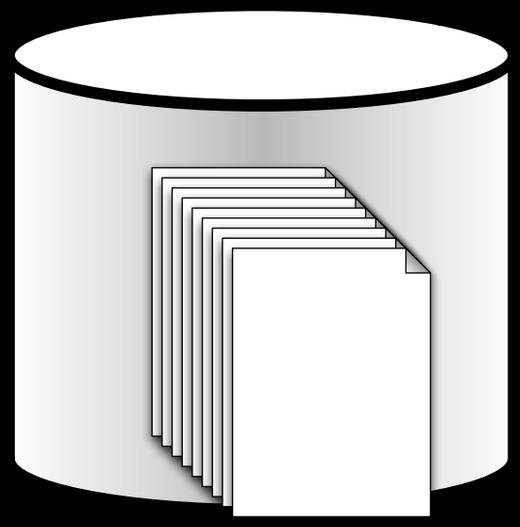
Oliver Kennedy

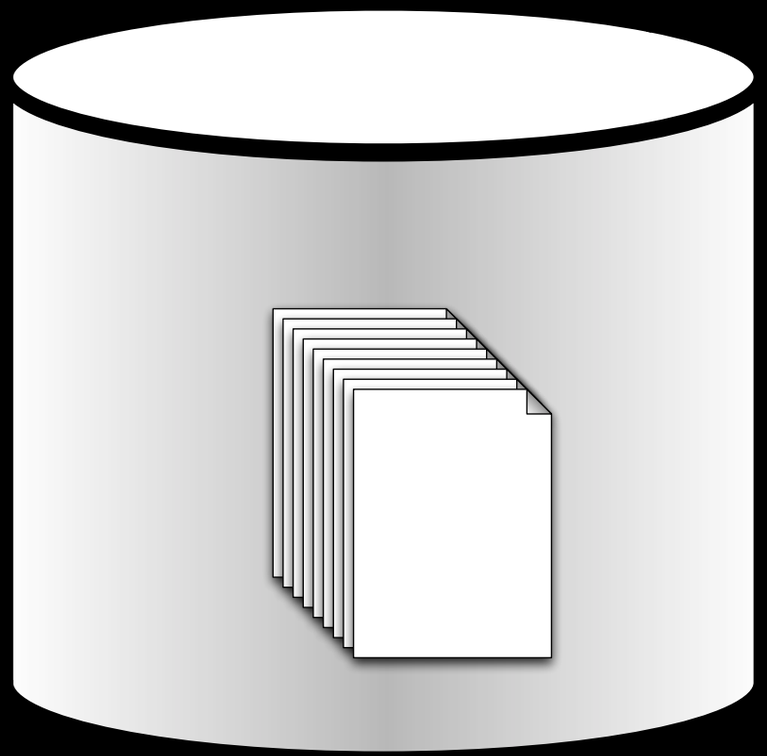
(joint work with JHU's Yanif Ahmad and Yotam Barnoy)

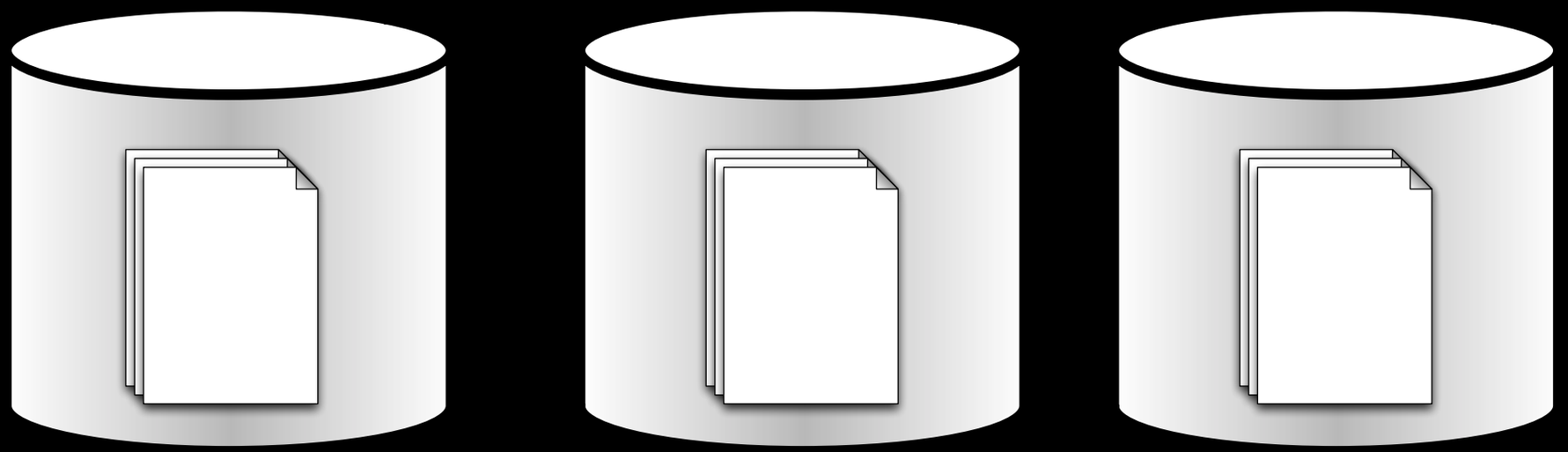


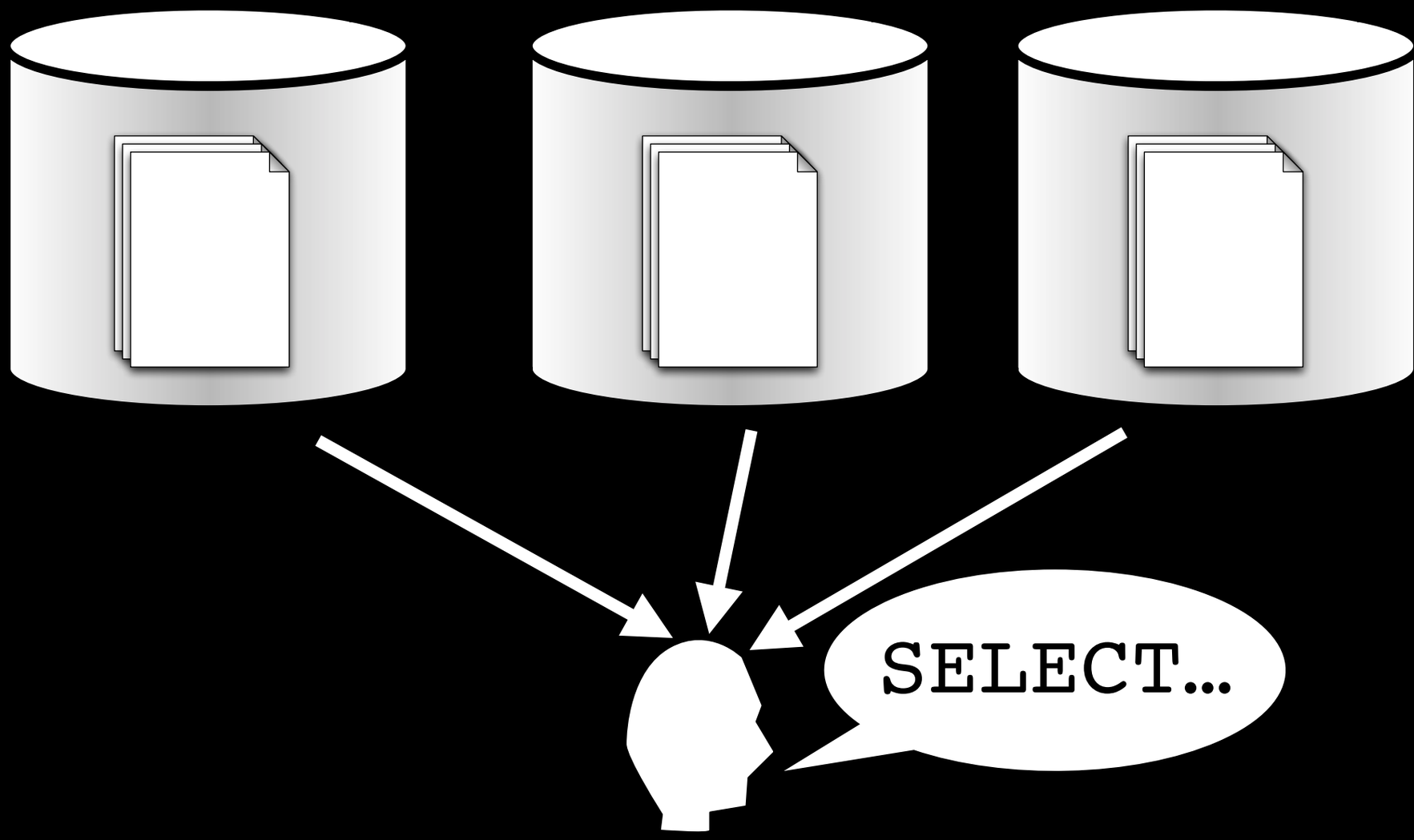


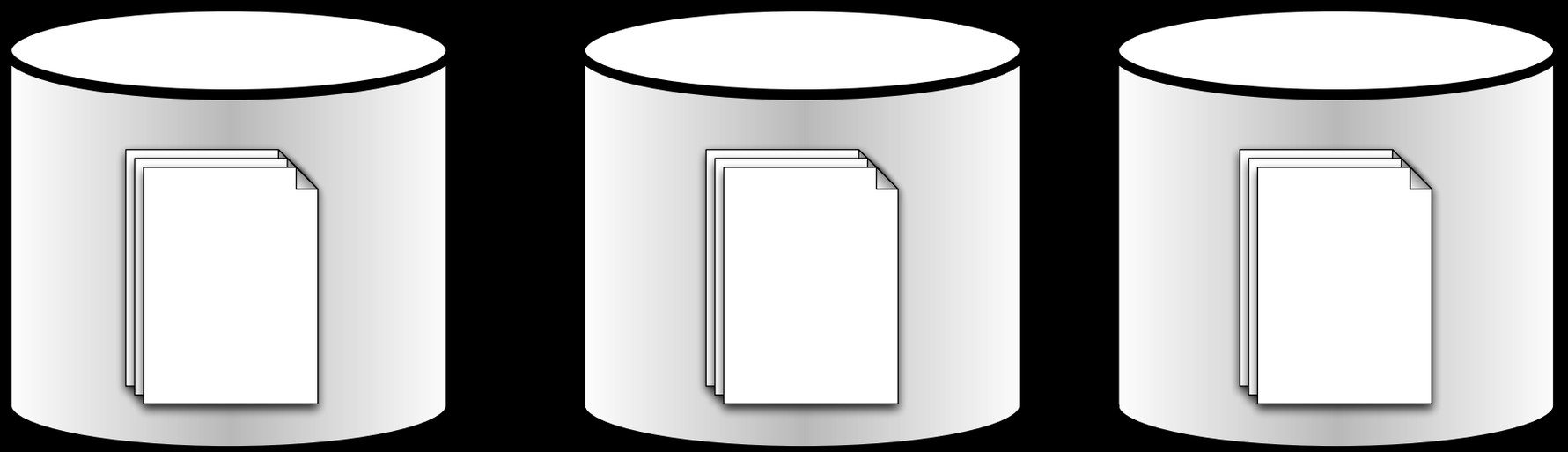

SELECT…

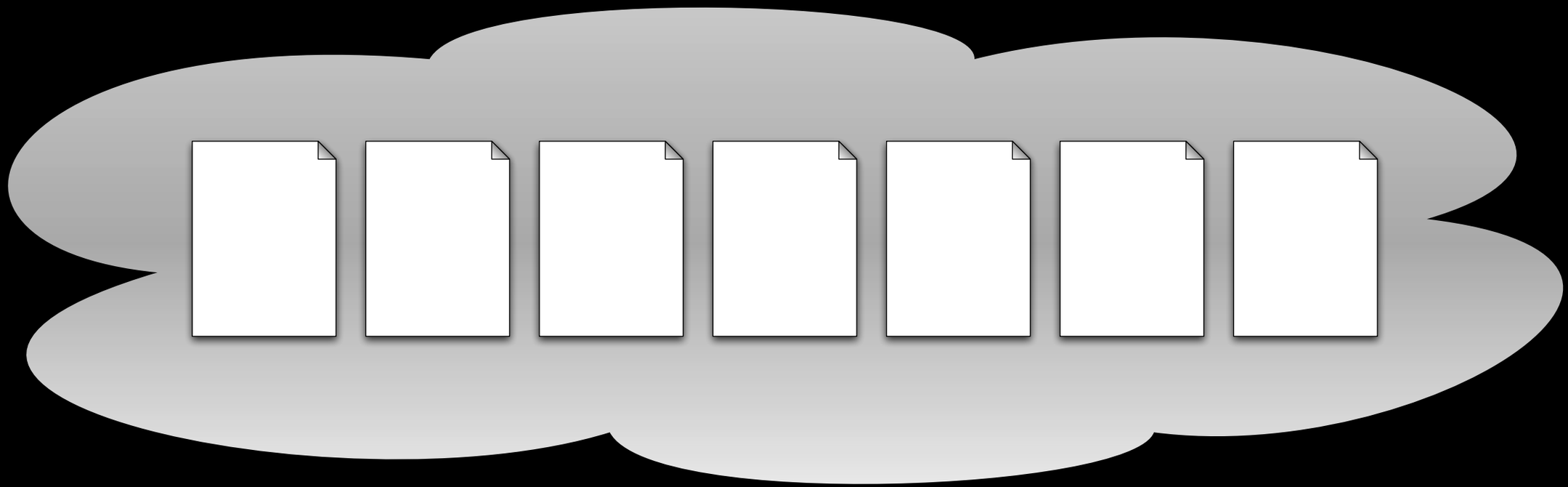


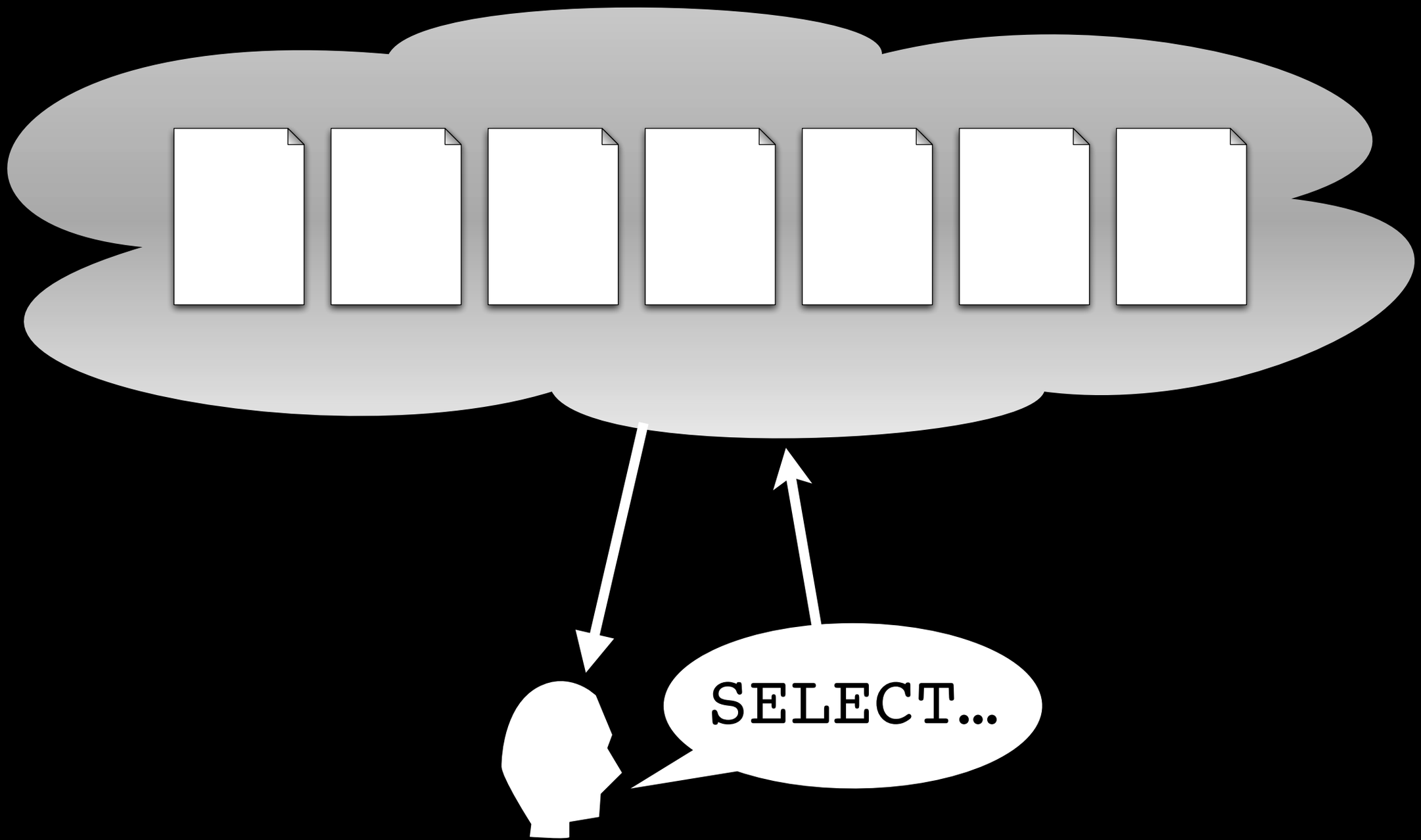


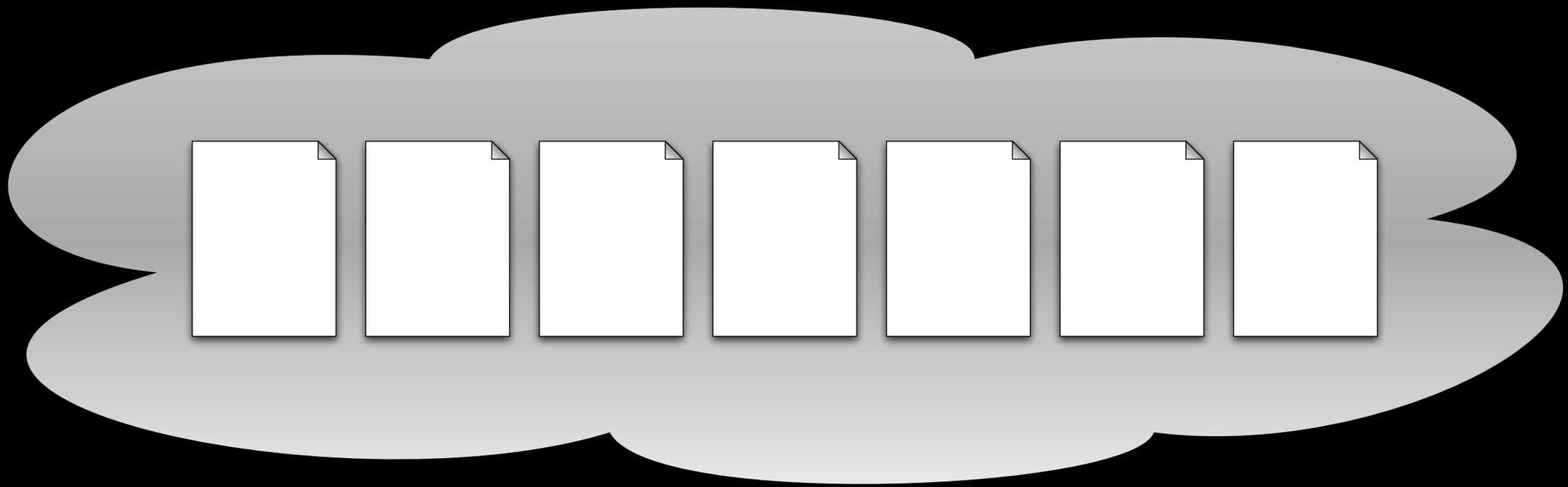


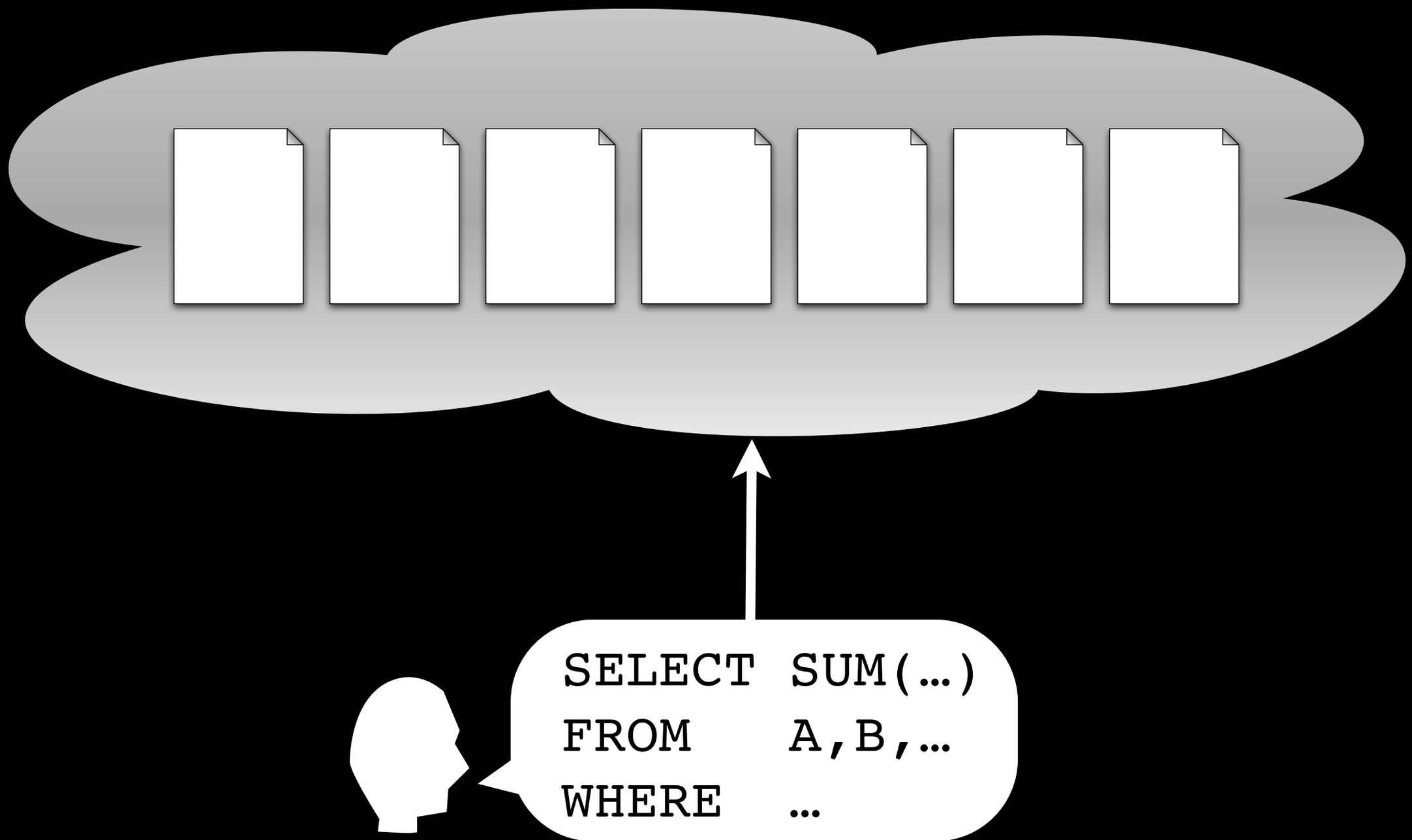

SELECT...

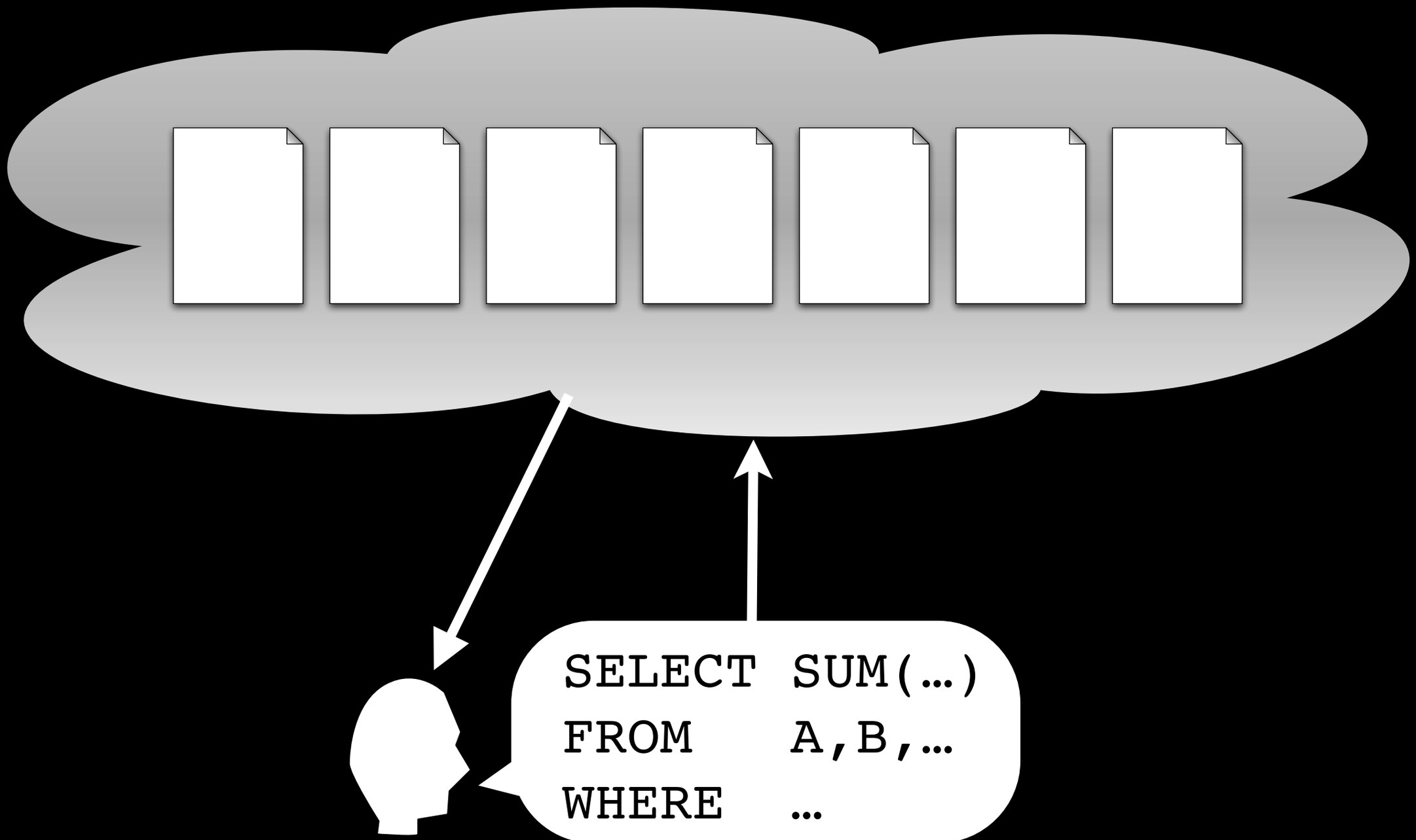


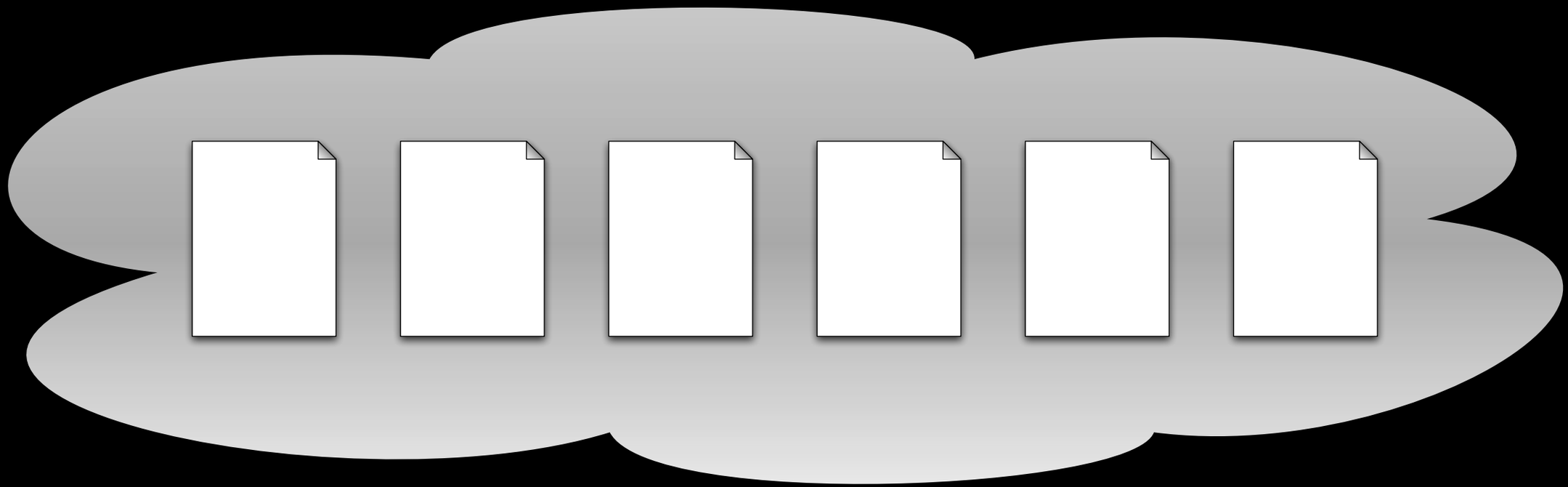


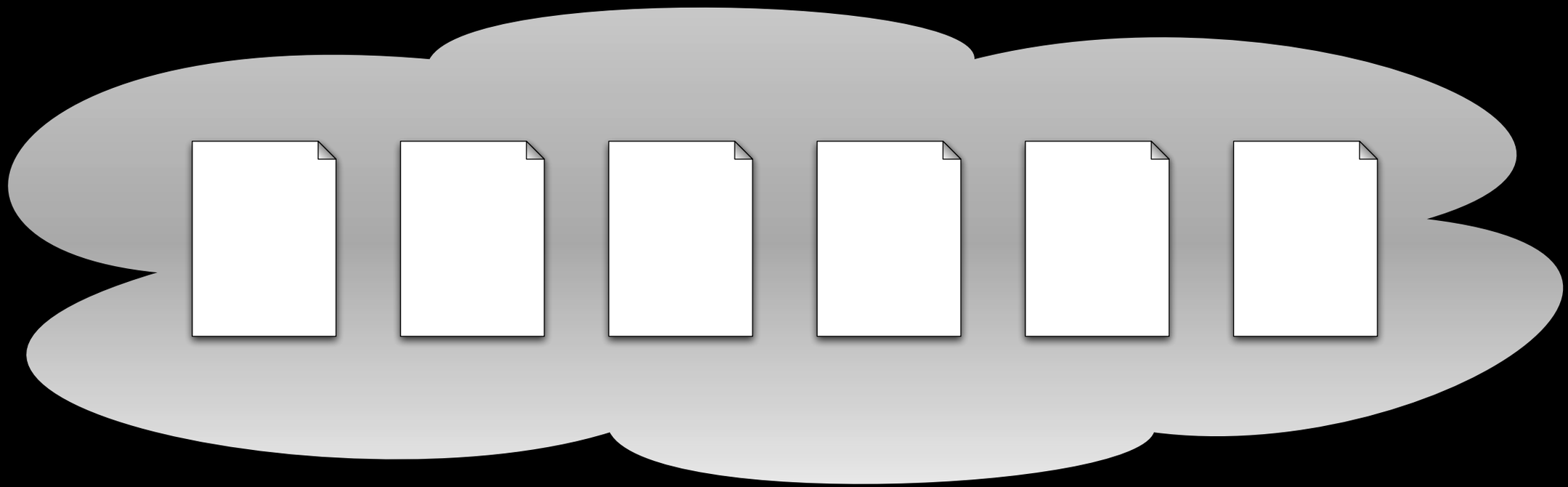

SELECT...



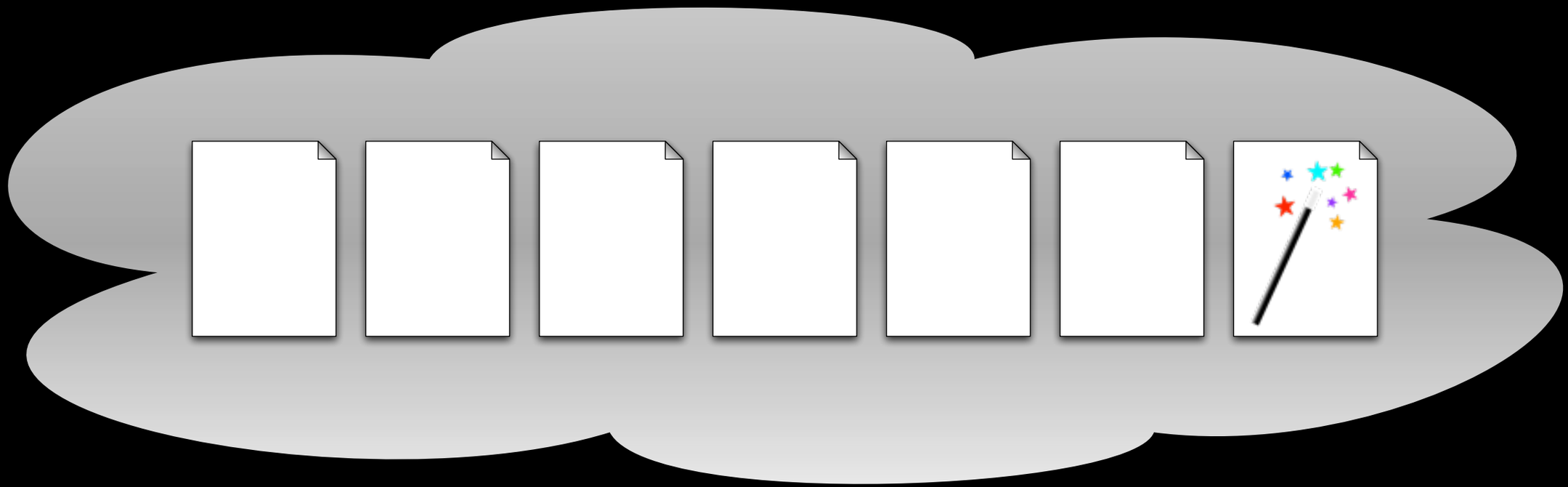

SELECT SUM(…)
FROM A,B,…
WHERE …

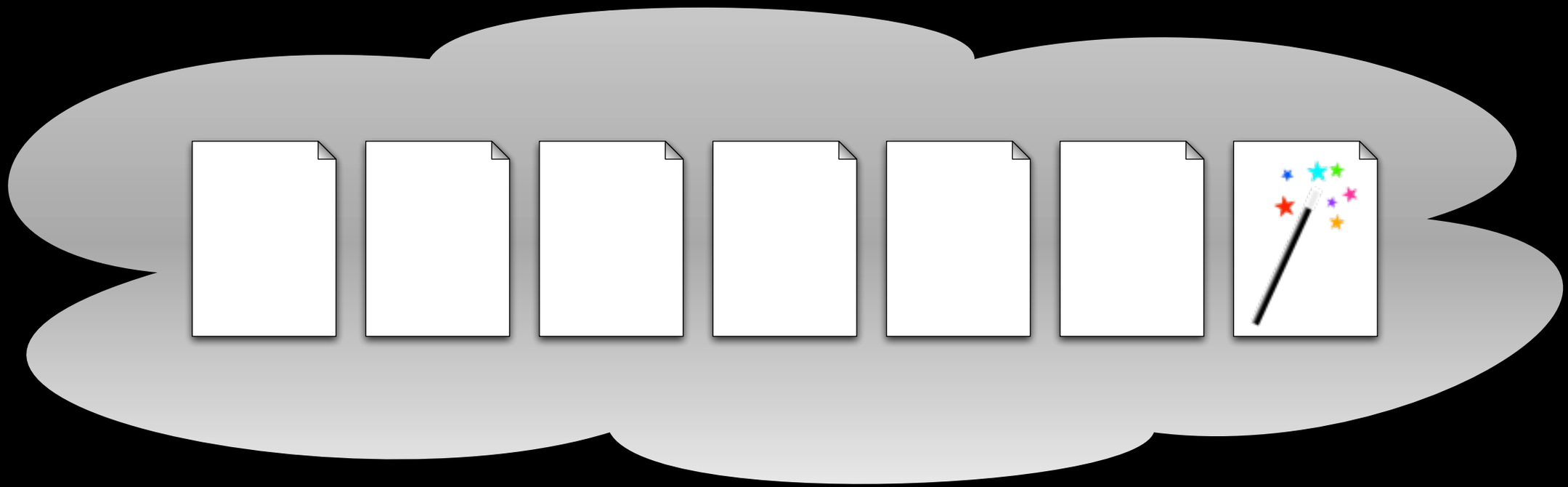

SELECT SUM(…)
FROM A,B,…
WHERE …

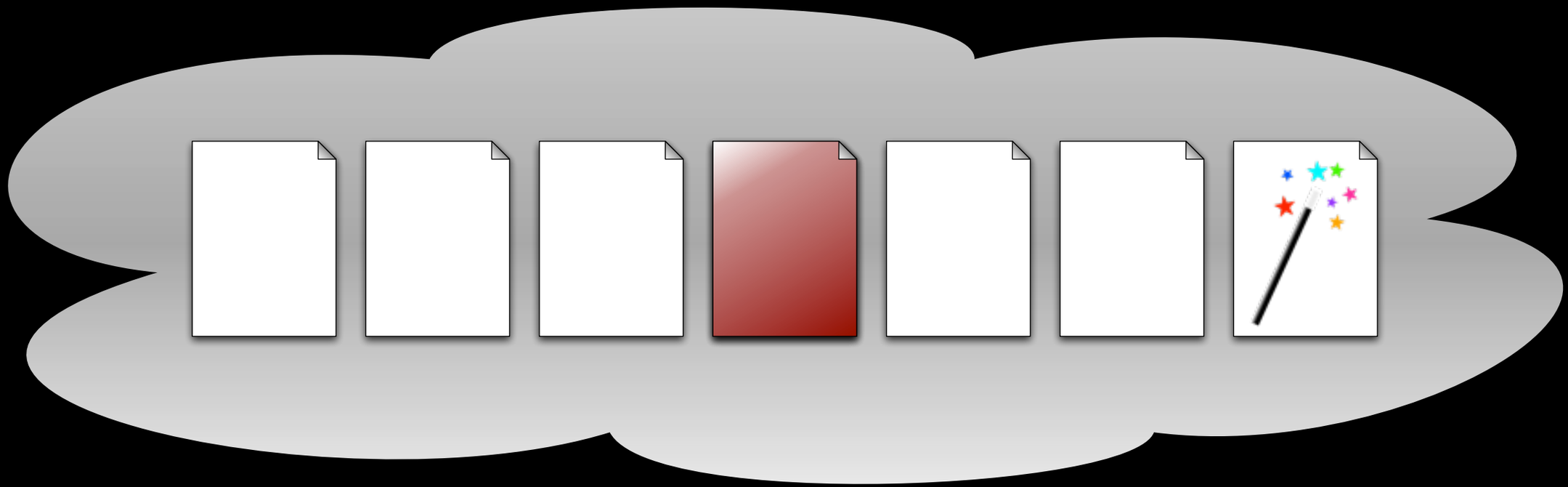




```
SELECT SUM(…)
FROM   A,B,…
WHERE  …
```







“Magic Maps”

Map Maintenance Messages

System Design

Hybrid Consistency

$Map_1[V_1, V_2, \ldots]$

$$Map_1[V_1,V_2,\ldots] \mathrel{+}= f(Map_2,Map_3,\ldots,V_1,V_2,\ldots)$$

Insert Delete

```
ON Event(V1,V2,…) DO {
  Map1[V1,V2,…] += f(Map2,Map3,…,V1,V2,…)
}
```

```
ON Event(V1,V2,…) DO {
  Map1[V1,V2,…] += f(Map2,Map3,…,V1,V2,…)
  Map2[V1,V2,…] += f(Map1,Map3,…,V1,V2,…)
}
```

```
SELECT SUM(R.A * T.D)
FROM   R, S, T
WHERE  R.B = S.B
  AND  S.C = T.C
```

```
ON Insert_R(VA,VB) DO {

    SELECT SUM(R.A * T.D)
    FROM   R, S, T
    WHERE  R.B = S.B
      AND  S.C = T.C

}
```

```
ON Insert_R(VA,VB) DO {

 Result[]  +=

    SELECT SUM(R.A * T.D)
    FROM   R, S, T
    WHERE  R.B = S.B
      AND  S.C = T.C

}
```

```
ON Insert_R(VA,VB) DO {

 Result[]  +=

    SELECT VA * SUM(T.D)
    FROM         S, T
    WHERE   VB    = S.B
      AND   S.C = T.C

}
```

```
ON Insert_R(VA,VB) DO {

 Result[]  +=

            VA * Map2[VB]

}
```

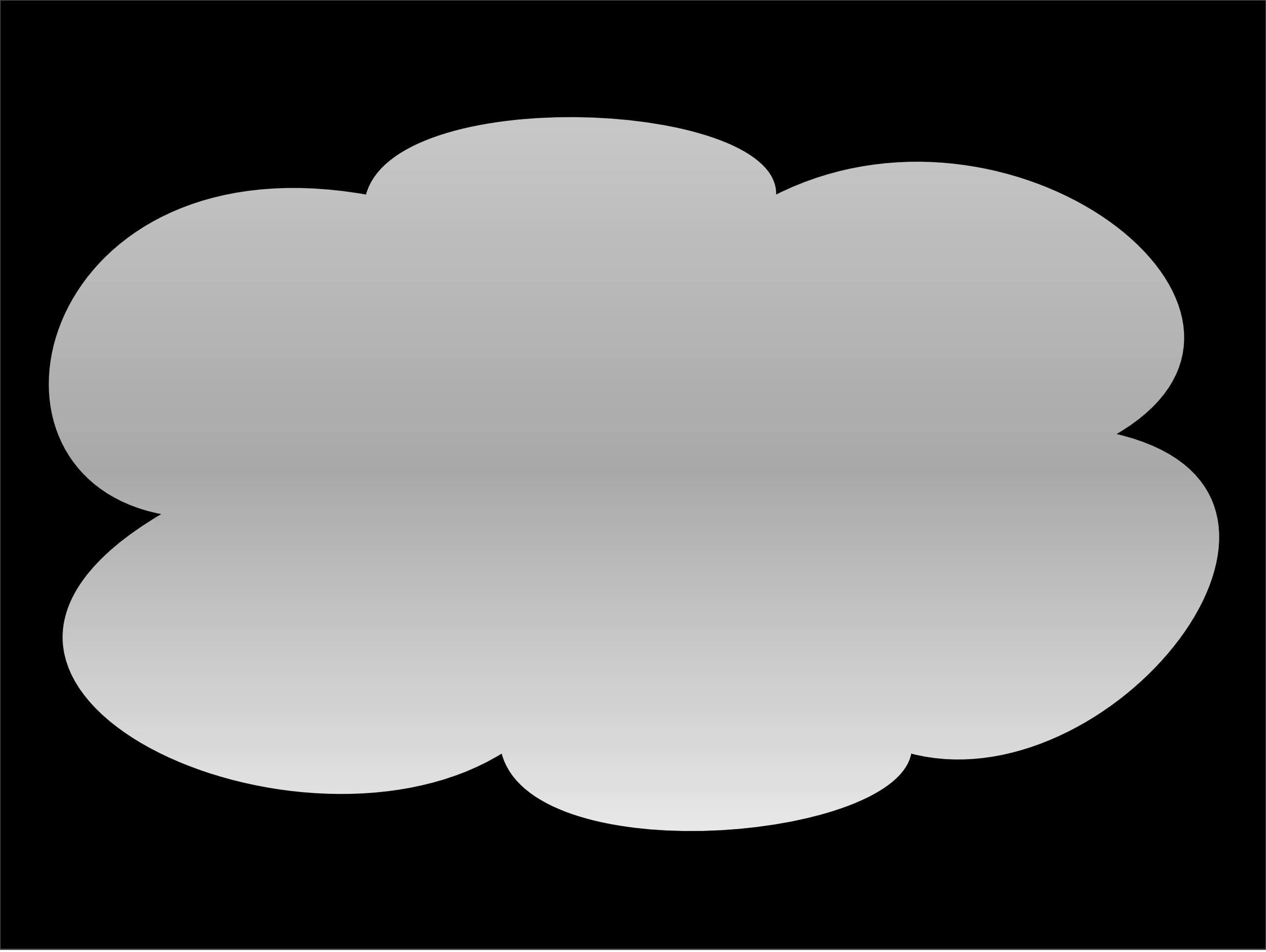
```
ON Insert_R(A,B) DO {
  Result[]  += A * Map2[B]
  Map1[C]   += A * Map3[B,C]
  Map5[B]   += A }

ON Insert_T(C,D) DO {
  Result[] += Map1[C] * D
  Map2[B]  += D * Map3[B,C]
  Map4[C]  += D }

ON Insert_S(B,C) DO {
  Result[]  += Map5[B] * Map4[C]
  Map1[C]   += Map5[B]
  Map2[B]   += Map4[C]
  Map3[B,C] += 1 }
```

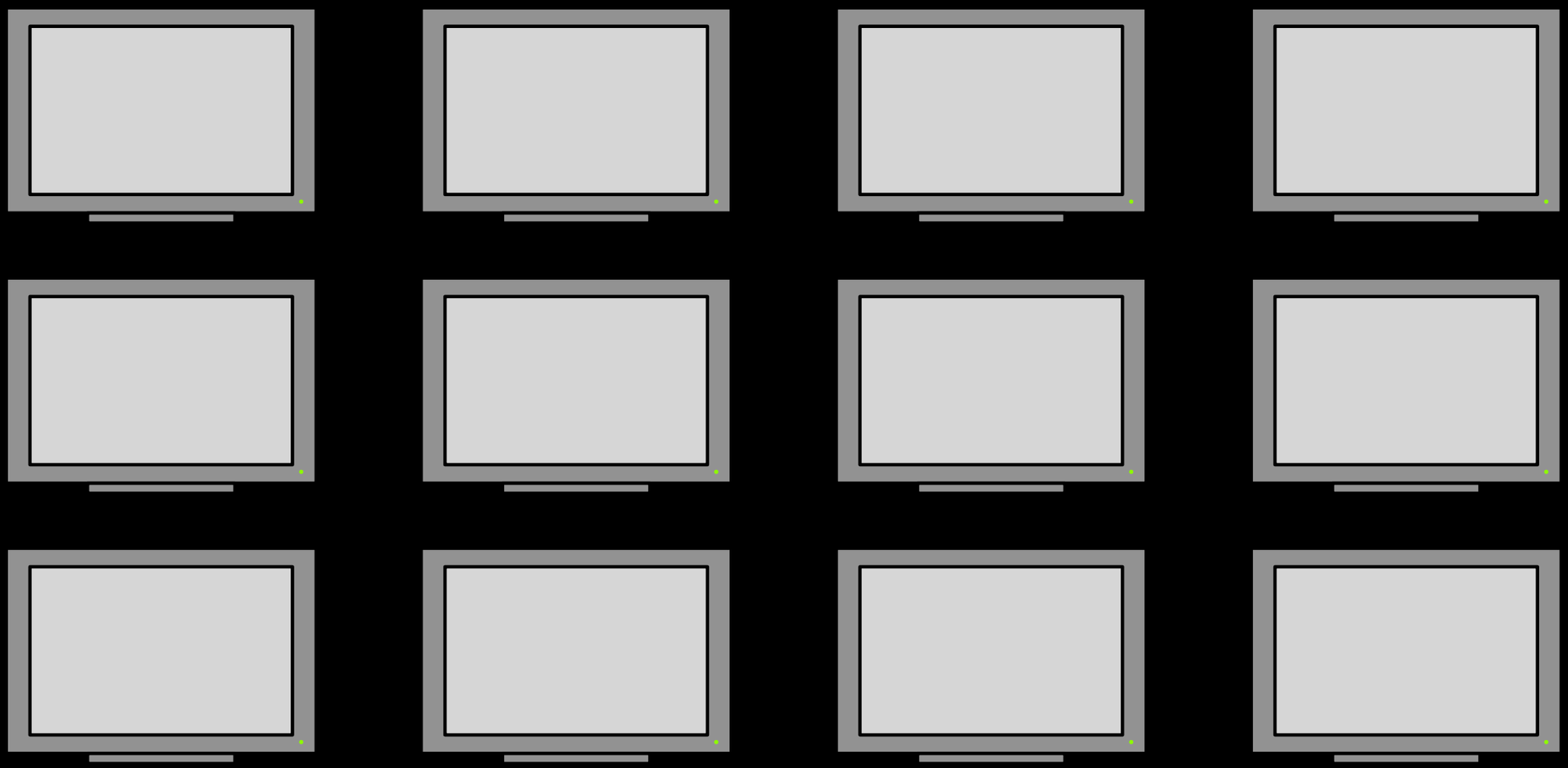
“Magic Maps”

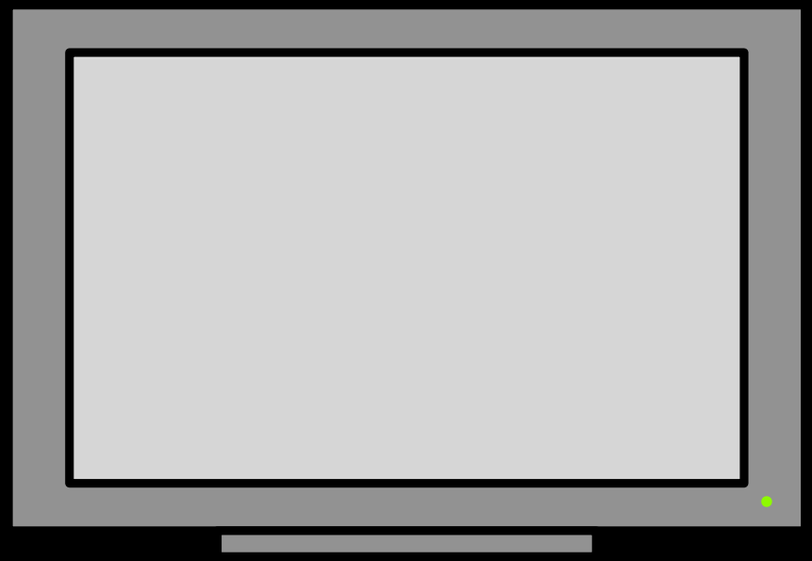
Map Maintenance Messages

System Design

Hybrid Consistency



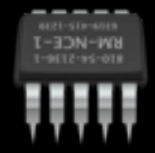


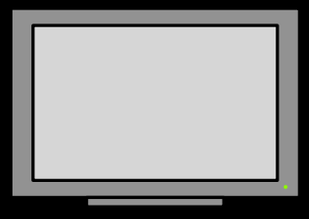


```
Map1
Map2
Map3
…
```

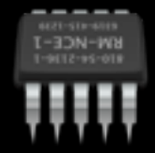
$$Map_1[\,] \mathrel{+}= Map_2[\,] * Map_3[\,]$$

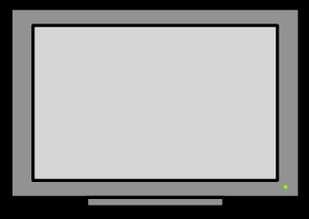
$Map_1[] \mathrel{+}= Map_2[] * Map_3[]$



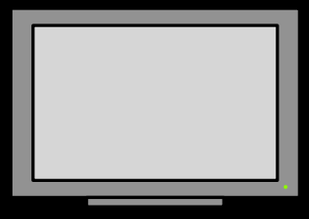


$Map_1[] \mathrel{+}= Map_2[] * Map_3[]$



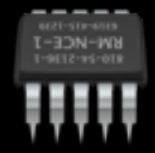


$Map_1[] \mathrel{+}= Map_2[] * Map_3[]$



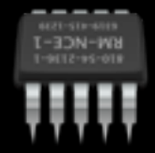
Map_1	
Map_2	
Map_3	

$Map_1[\,] \mathrel{+}= Map_2[\,] * Map_3[\,]$



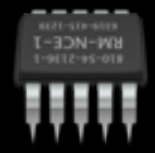
Map_1	$Map_2 * Map_3$
Map_2	
Map_3	

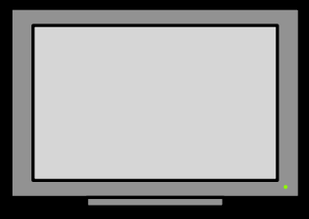
$Map_1[\,] \mathrel{+}= Map_2[\,] * Map_3[\,]$



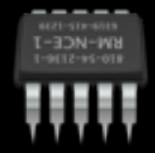
Map_1	$Map_2 * Map_3$
Map_2	
Map_3	

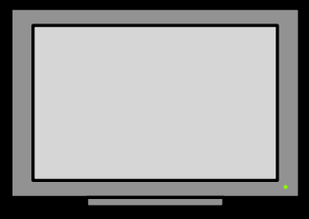
$Map_1[] \mathrel{+}= Map_2[] * Map_3[]$



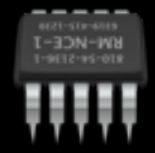


$Map_1[] \mathrel{+}= Map_2[] * Map_3[]$



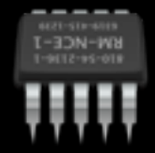


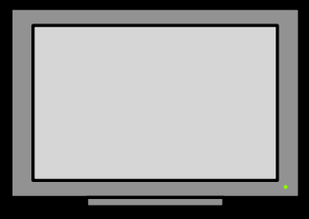
$Map_1[] \mathrel{+}= Map_2[] * Map_3[]$



Map_2, Map_3	
Map_1	
Map_1	

$Map_1[] \mathrel{+}= Map_2[] * Map_3[]$



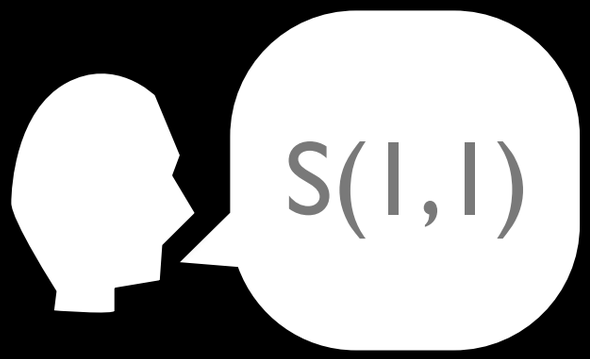


Map_2,Map_3	$Map_2 * Map_3$
Map_1	
Map_1	

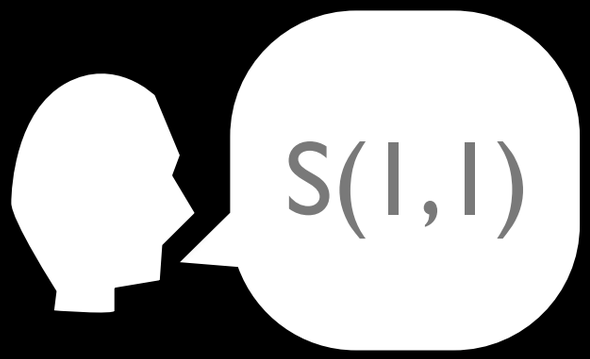
$Map_1[] \mathrel{+}= Map_2[] * Map_3[]$

Map_2, Map_3	$Map_2 * Map_3$
Map_1	
Map_1	

```
ON Insert_R(A,B) DO {
  Result[]  += A * Map2[B]
  Map1[C]   += A * Map3[B,C]
  Map5[B]   += A }

ON Insert_T(C,D) DO {
  Result[] += Map1[C] * D
  Map2[B]  += D * Map3[B,C]
  Map4[C]  += D }

ON Insert_S(B,C) DO {
  Result[]  += Map5[B] * Map4[C]
  Map1[C]   += Map5[B]
  Map2[B]   += Map4[C]
  Map3[B,C] += 1 }
```

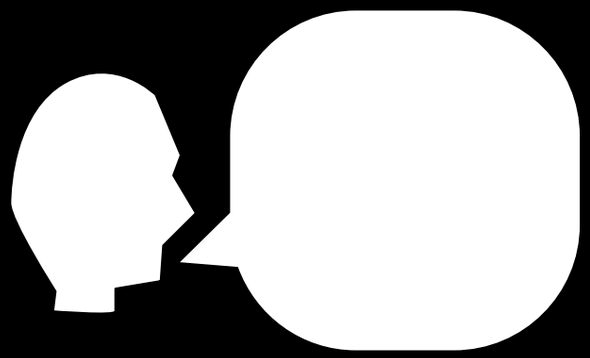



```
ON Insert_R(A,B) DO {
  Result[]  += A * Map2[B]
  Map1[C]   += A * Map3[B,C]
  Map5[B]   += A }

ON Insert_T(C,D) DO {
  Result[] += Map1[C] * D
  Map2[B]  += D * Map3[B,C]
  Map4[C]  += D }

ON Insert_S(B,C) DO {
  Result[]  += Map5[B] * Map4[C]
  Map1[C]   += Map5[B]
  Map2[B]   += Map4[C]
  Map3[B,C] += 1 }
```



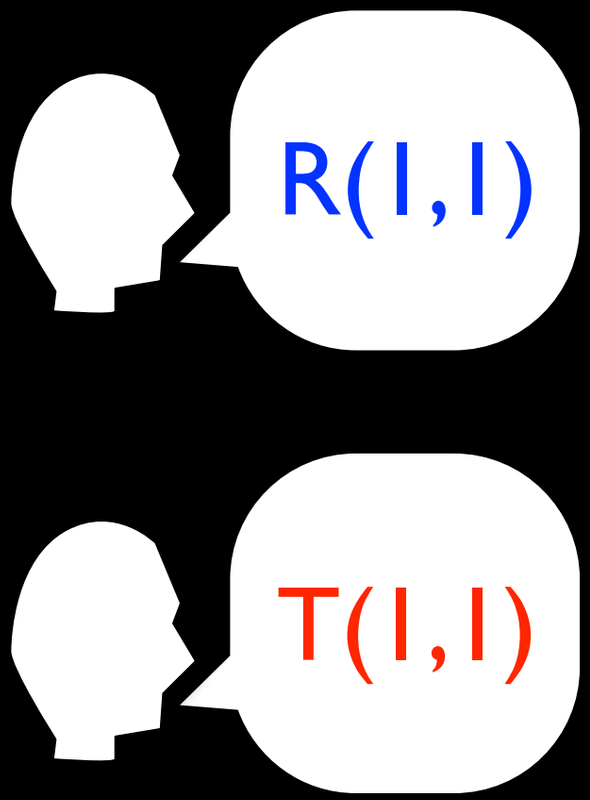

```
ON Insert_R(A,B) DO {
  Result[]  += A * Map2[B]
  Map1[C]   += A * Map3[B,C]
  Map5[B]   += A }

ON Insert_T(C,D) DO {
  Result[] += Map1[C] * D
  Map2[B]  += D * Map3[B,C]
  Map4[C]  += D }

ON Insert_S(B,C) DO {
  Result[]  += Map5[B] * Map4[C]
  Map1[C]   += Map5[B]
  Map2[B]   += Map4[C]
  Map3[B,C] += 1 }
```

Map1	
Map2	
Map3	
Map4	
Map5	
Result	



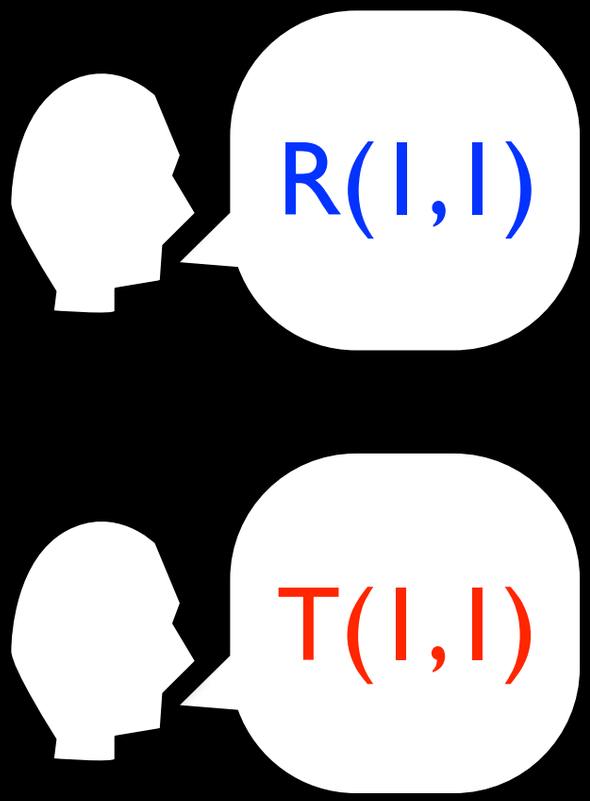
```
ON Insert_R(A,B) DO {
  Result[]  += A * Map2[B]
  Map1[C]   += A * Map3[B,C]
  Map5[B]   += A }

ON Insert_T(C,D) DO {
  Result[] += Map1[C] * D
  Map2[B]  += D * Map3[B,C]
  Map4[C]  += D }

ON Insert_S(B,C) DO {
  Result[]  += Map5[B] * Map4[C]
  Map1[C]   += Map5[B]
  Map2[B]   += Map4[C]
  Map3[B,C] += 1 }
```

Map1	
Map2	
Map3	<1,1>→1
Map4	
Map5	
Result	



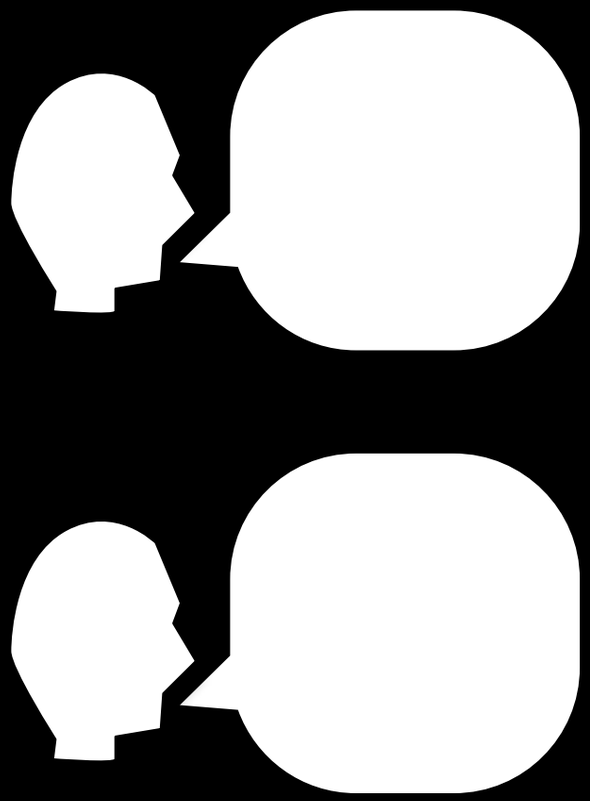

```
ON Insert_R(A,B) DO {
  Result[]  += A * Map2[B]
  Map1[C]   += A * Map3[B,C]
  Map5[B]   += A }

ON Insert_T(C,D) DO {
  Result[] += Map1[C] * D
  Map2[B]  += D * Map3[B,C]
  Map4[C]  += D }

ON Insert_S(B,C) DO {
  Result[]  += Map5[B] * Map4[C]
  Map1[C]   += Map5[B]
  Map2[B]   += Map4[C]
  Map3[B,C] += 1 }
```

Map1	
Map2	
Map3	<1,1>→1
Map4	
Map5	
Result	




```
ON Insert_R(A,B) DO {
  Result[]  += A * Map2[B]
  Map1[C]   += A * Map3[B,C]
  Map5[B]   += A }

ON Insert_T(C,D) DO {
  Result[] += Map1[C] * D
  Map2[B]  += D * Map3[B,C]
  Map4[C]  += D }

ON Insert_S(B,C) DO {
  Result[]  += Map5[B] * Map4[C]
  Map1[C]   += Map5[B]
  Map2[B]   += Map4[C]
  Map3[B,C] += 1 }
```

Map1	
Map2	
Map3	<1,1>→1
Map4	
Map5	
Result	

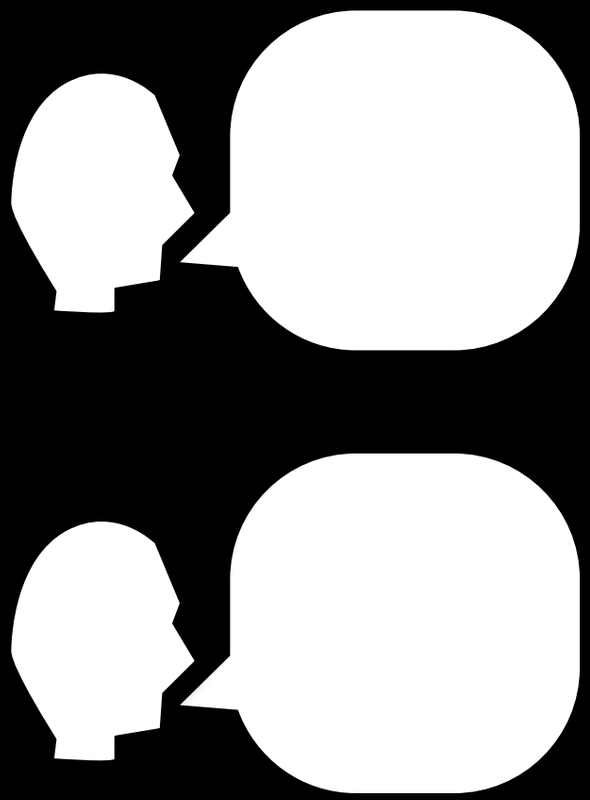


```
ON Insert_R(A,B) DO {
  Result[]  += A * Map2[B]
  Map1[C]   += A * Map3[B,C]
  Map5[B]   += A }

ON Insert_T(C,D) DO {
  Result[] += Map1[C] * D
  Map2[B]  += D * Map3[B,C]
  Map4[C]  += D }

ON Insert_S(B,C) DO {
  Result[]  += Map5[B] * Map4[C]
  Map1[C]   += Map5[B]
  Map2[B]   += Map4[C]
  Map3[B,C] += 1 }
```

Map1	
Map2	
Map3	<1,1>→1
Map4	
Map5	
Result	

Map1
Map3
Map4

Map2
Map3
Map5

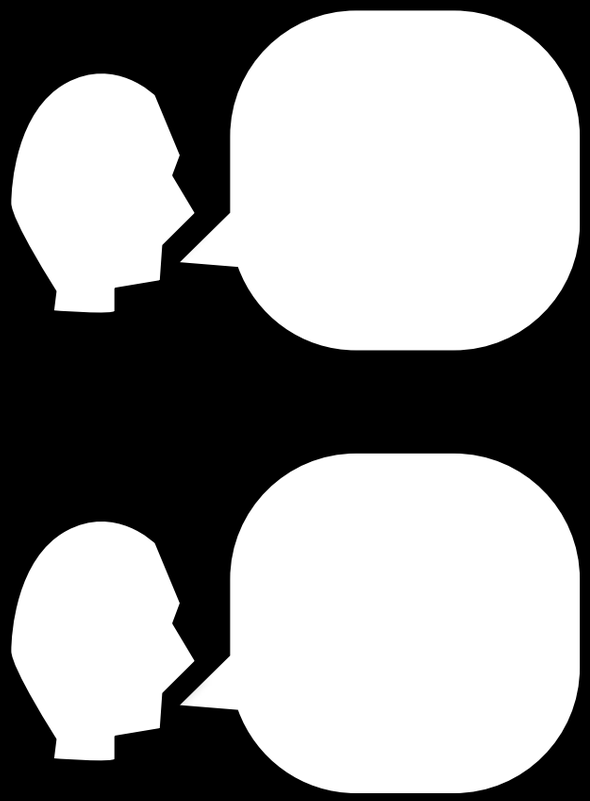


```
ON Insert_R(A,B) DO {
  Result[]  += A * Map2[B]
  Map1[C]   += A * Map3[B,C]
  Map5[B]   += A }

ON Insert_T(C,D) DO {
  Result[] += Map1[C] * D
  Map2[B]  += D * Map3[B,C]
  Map4[C]  += D }

ON Insert_S(B,C) DO {
  Result[]  += Map5[B] * Map4[C]
  Map1[C]   += Map5[B]
  Map2[B]   += Map4[C]
  Map3[B,C] += 1 }
```

Map1	
Map2	
Map3	<1,1>→1
Map4	<1>→1
Map5	<1>→1
Result	

Map1
Map3
Map4

Map2
Map3
Map5

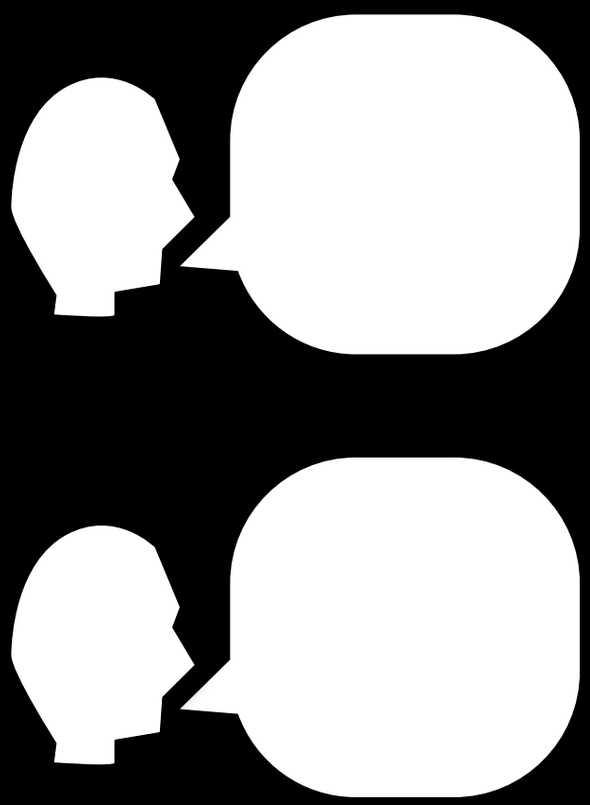


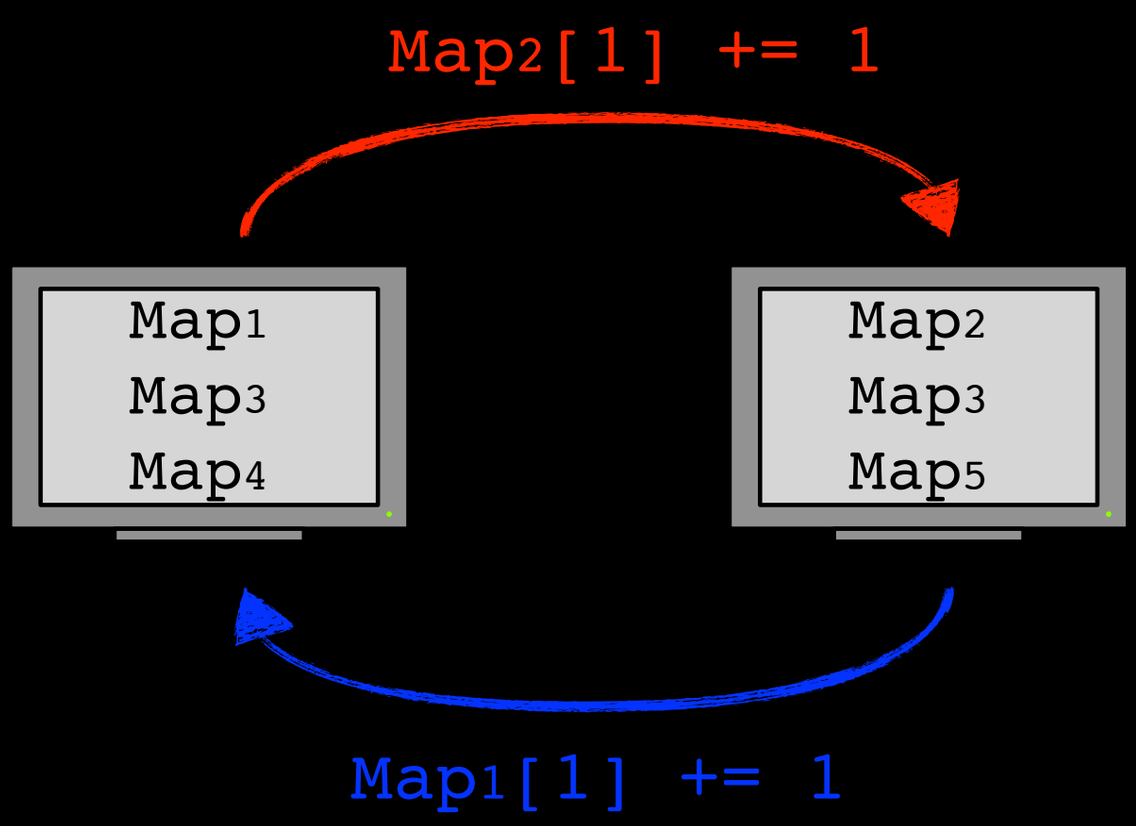
Map1
Map3
Map4

Map2
Map3
Map5

```
ON Insert_R(A,B) DO {
  Result[]  += A * Map2[B]
  Map1[C]   += A * Map3[B,C]
  Map5[B]   += A }

ON Insert_T(C,D) DO {
  Result[] += Map1[C] * D
  Map2[B]  += D * Map3[B,C]
  Map4[C]  += D }

ON Insert_S(B,C) DO {
  Result[]  += Map5[B] * Map4[C]
  Map1[C]   += Map5[B]
  Map2[B]   += Map4[C]
  Map3[B,C] += 1 }
```

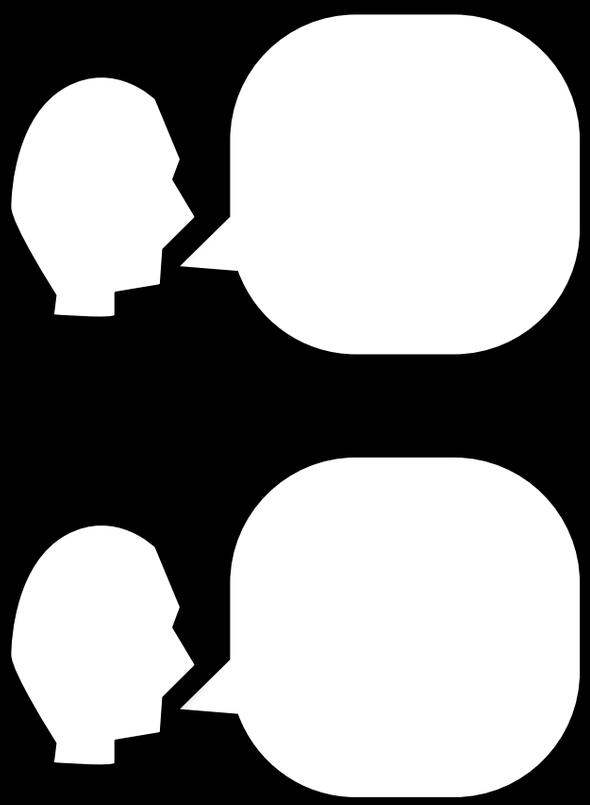
Map1	<1>→1
Map2	<1>→1
Map3	<1,1>→1
Map4	<1>→1
Map5	<1>→1
Result	





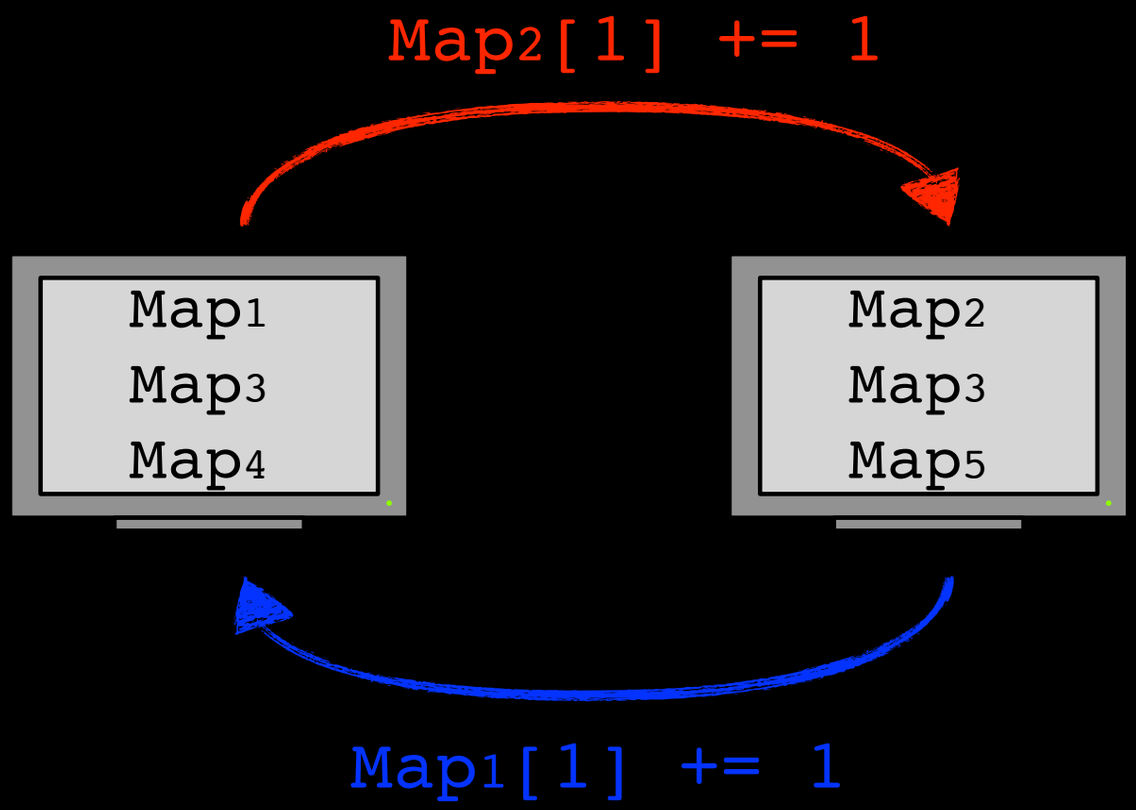

```
ON Insert_R(A,B) DO {
  Result[]  += A * Map2[B]
  Map1[C]   += A * Map3[B,C]
  Map5[B]   += A }

ON Insert_T(C,D) DO {
  Result[] += Map1[C] * D
  Map2[B]  += D * Map3[B,C]
  Map4[C]  += D }

ON Insert_S(B,C) DO {
  Result[]  += Map5[B] * Map4[C]
  Map1[C]   += Map5[B]
  Map2[B]   += Map4[C]
  Map3[B,C] += 1 }
```

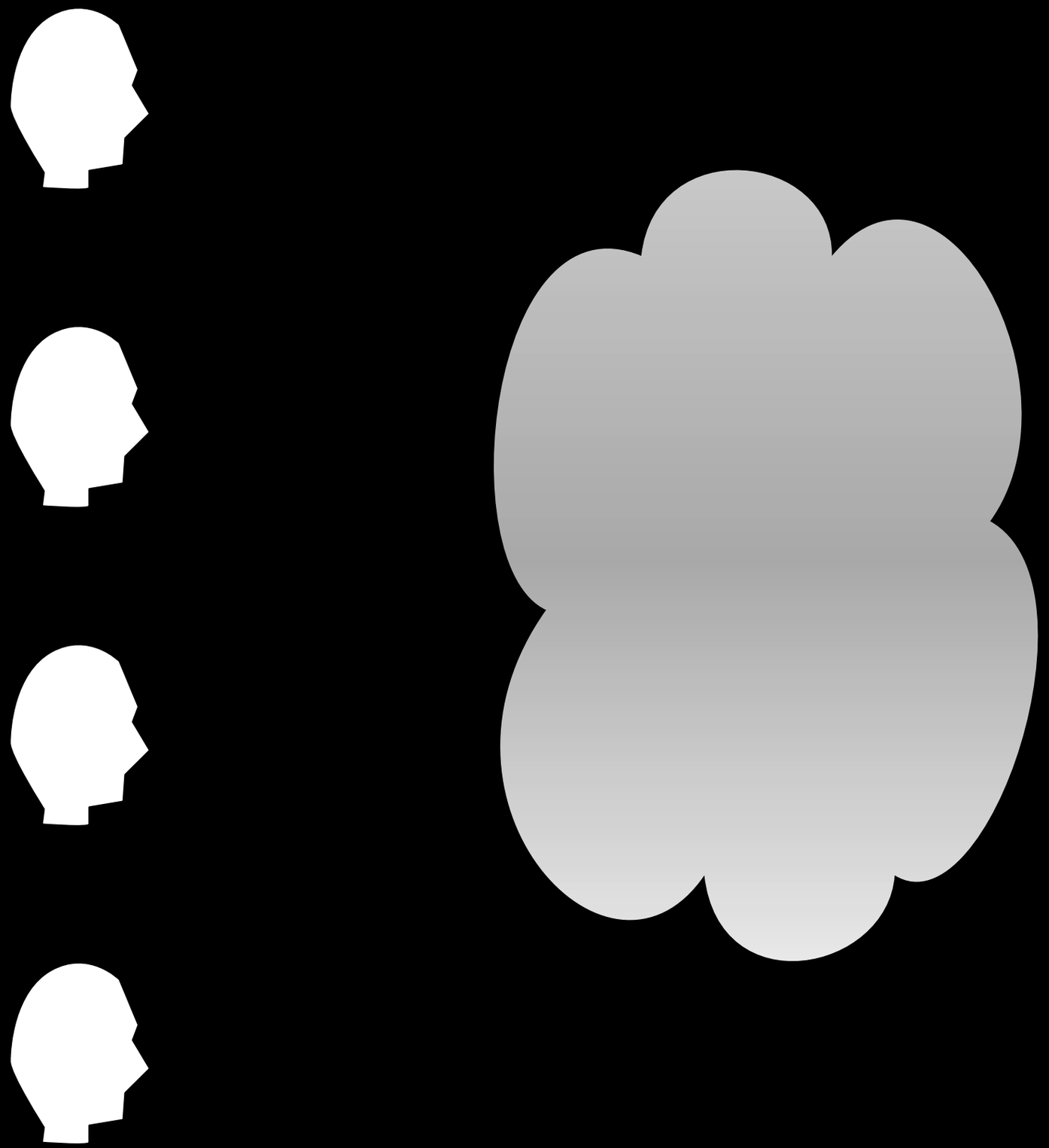
Map1	<1>→1
Map2	<1>→1
Map3	<1,1>→1
Map4	<1>→1
Map5	<1>→1
Result	

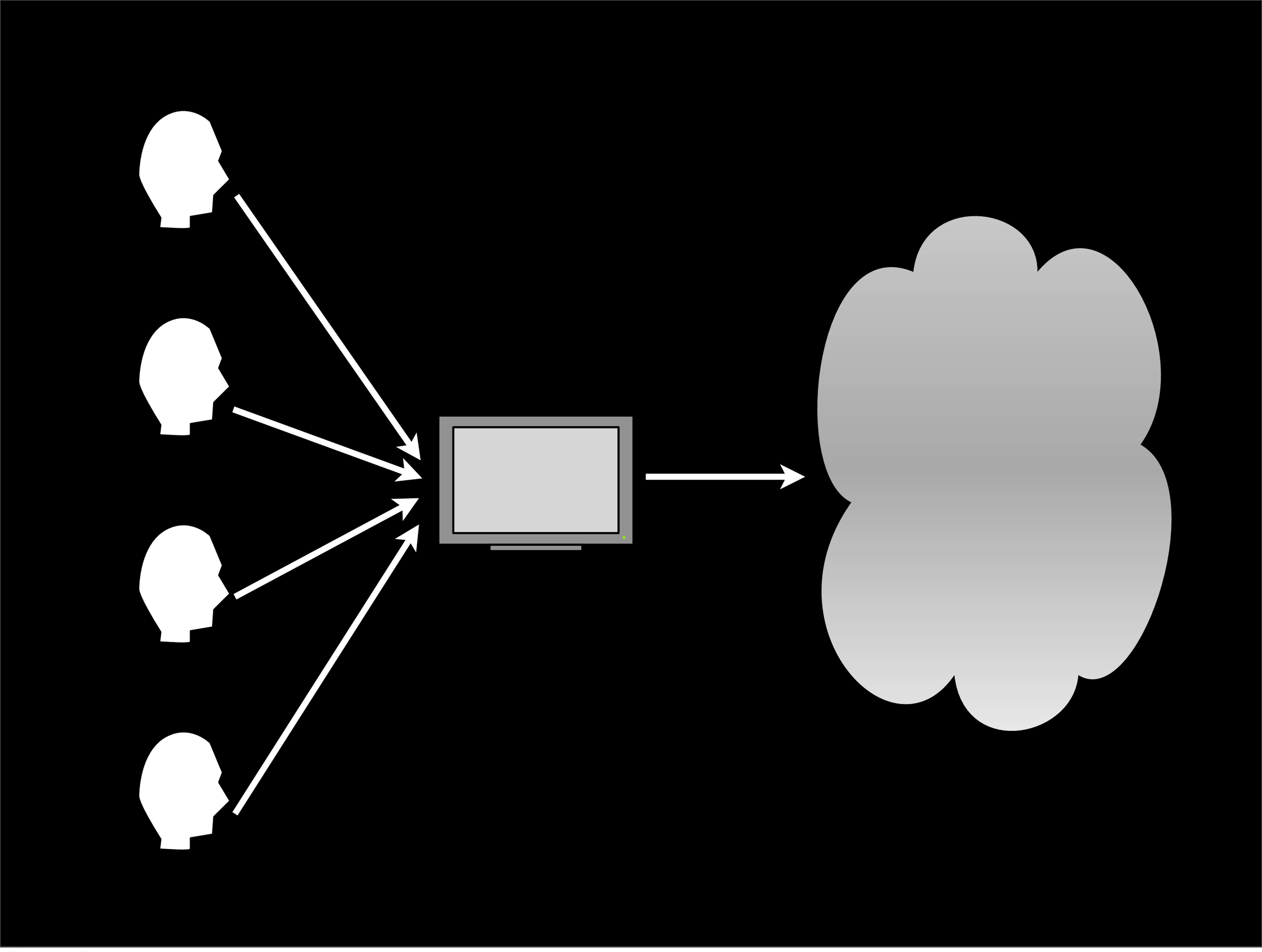


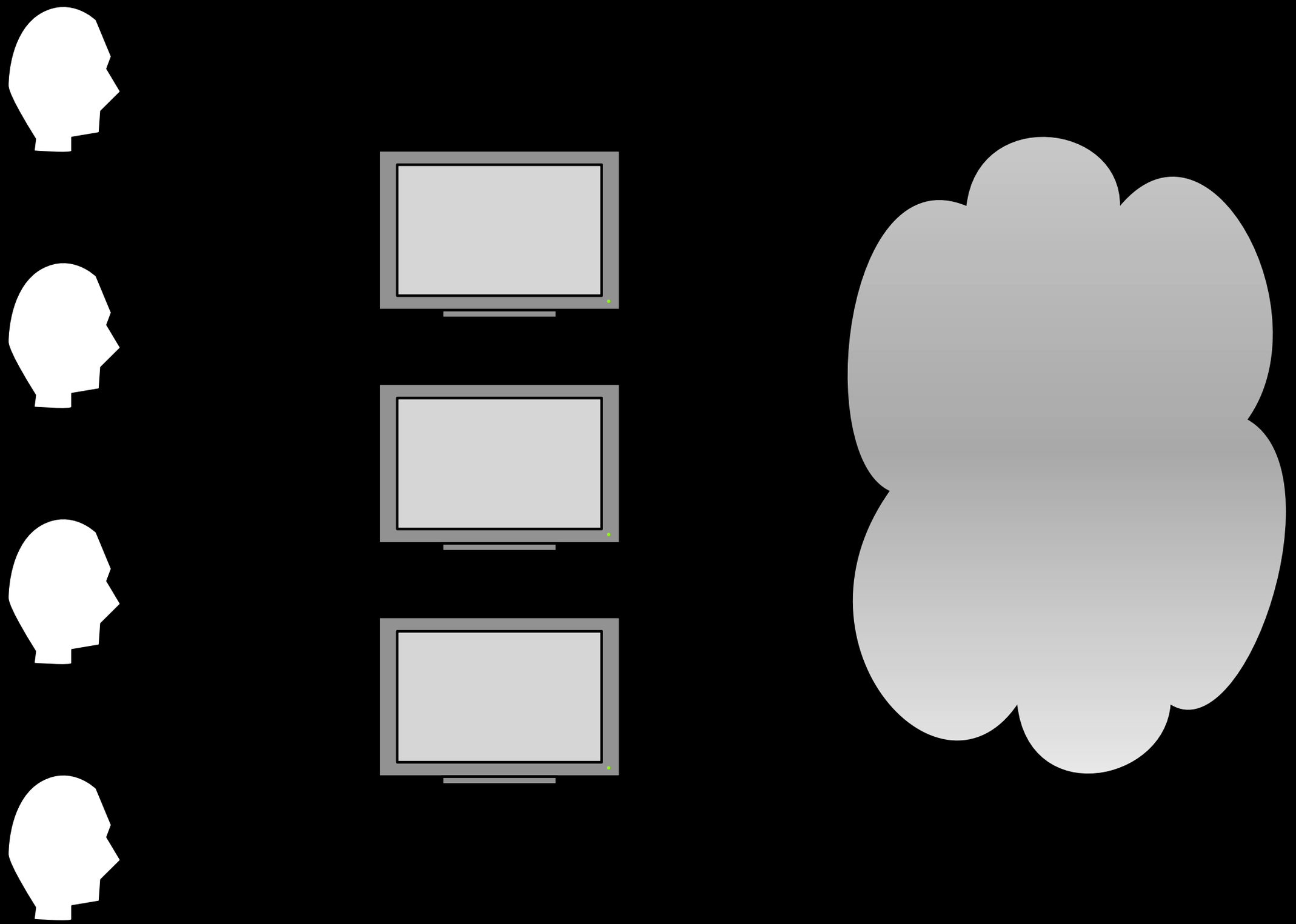
```
ON Insert_R(A,B) DO {
  Result[]  += A * Map2[B]
  Map1[C]   += A * Map3[B,C]
  Map5[B]   += A }

ON Insert_T(C,D) DO {
  Result[] += Map1[C] * D
  Map2[B]  += D * Map3[B,C]
  Map4[C]  += D }

ON Insert_S(B,C) DO {
  Result[]  += Map5[B] * Map4[C]
  Map1[C]   += Map5[B]
  Map2[B]   += Map4[C]
  Map3[B,C] += 1 }
```

Map1	<1>→1
Map2	<1>→1
Map3	<1,1>→1
Map4	<1>→1
Map5	<1>→1
Result	??







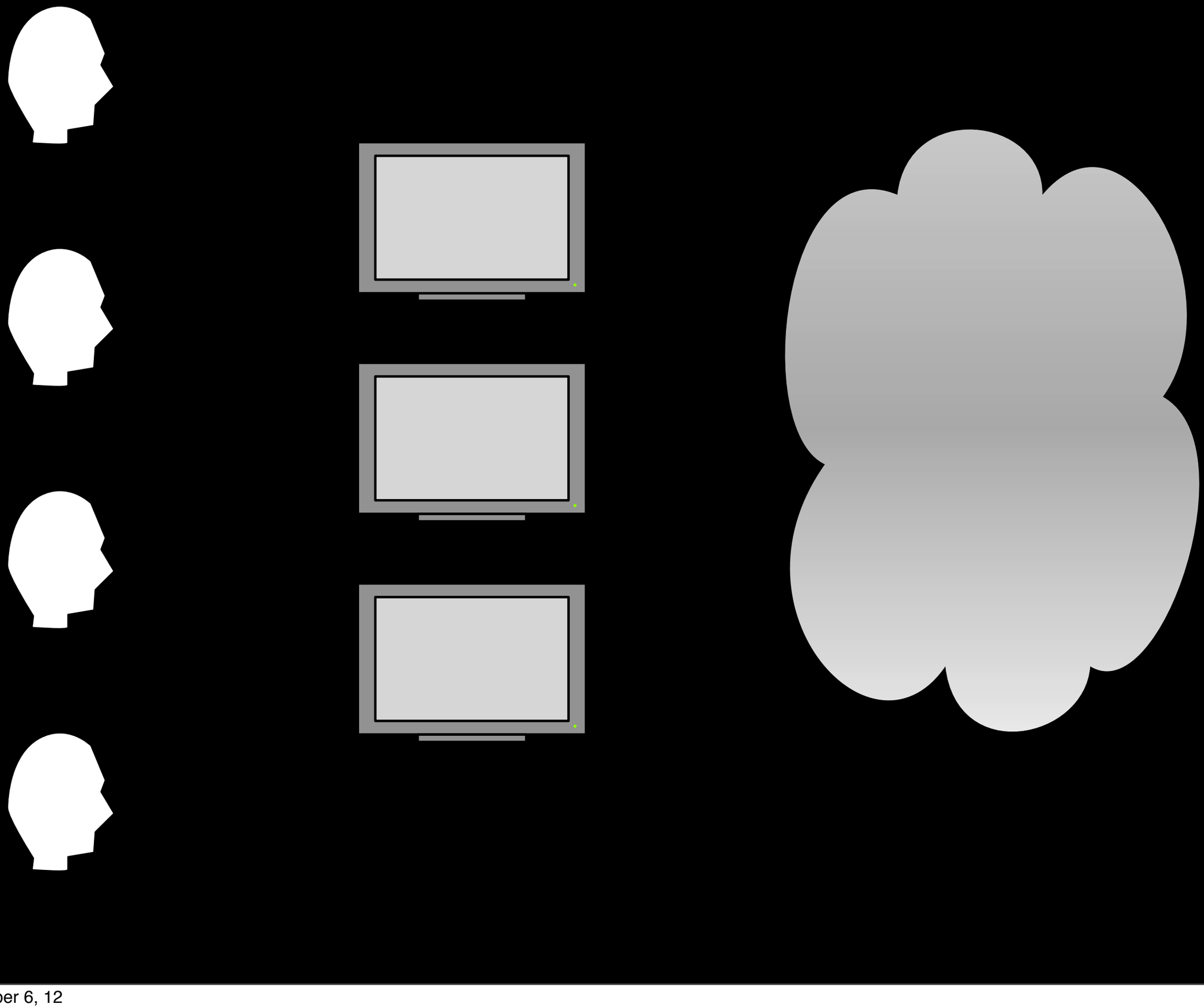


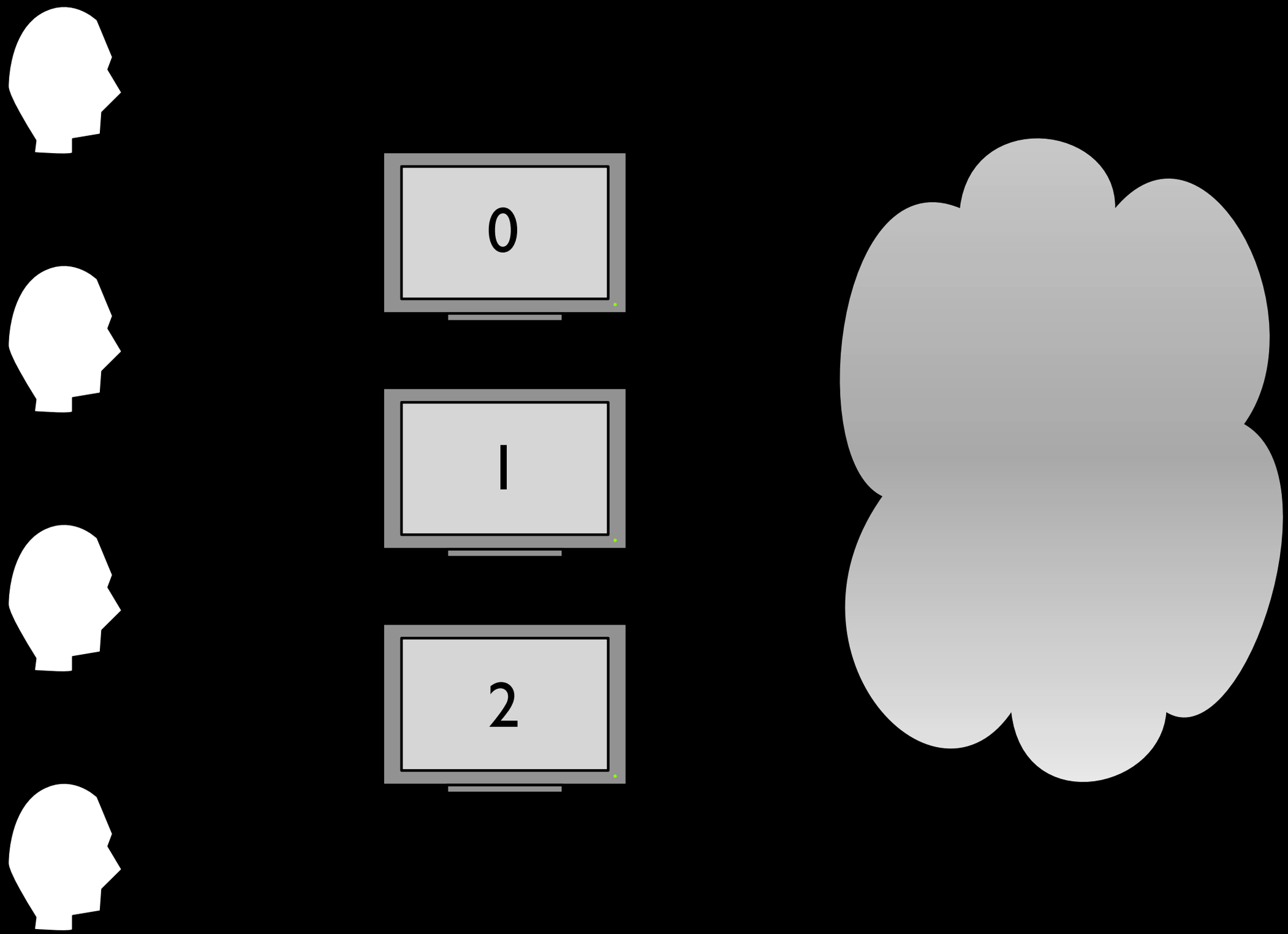
“Magic Maps”

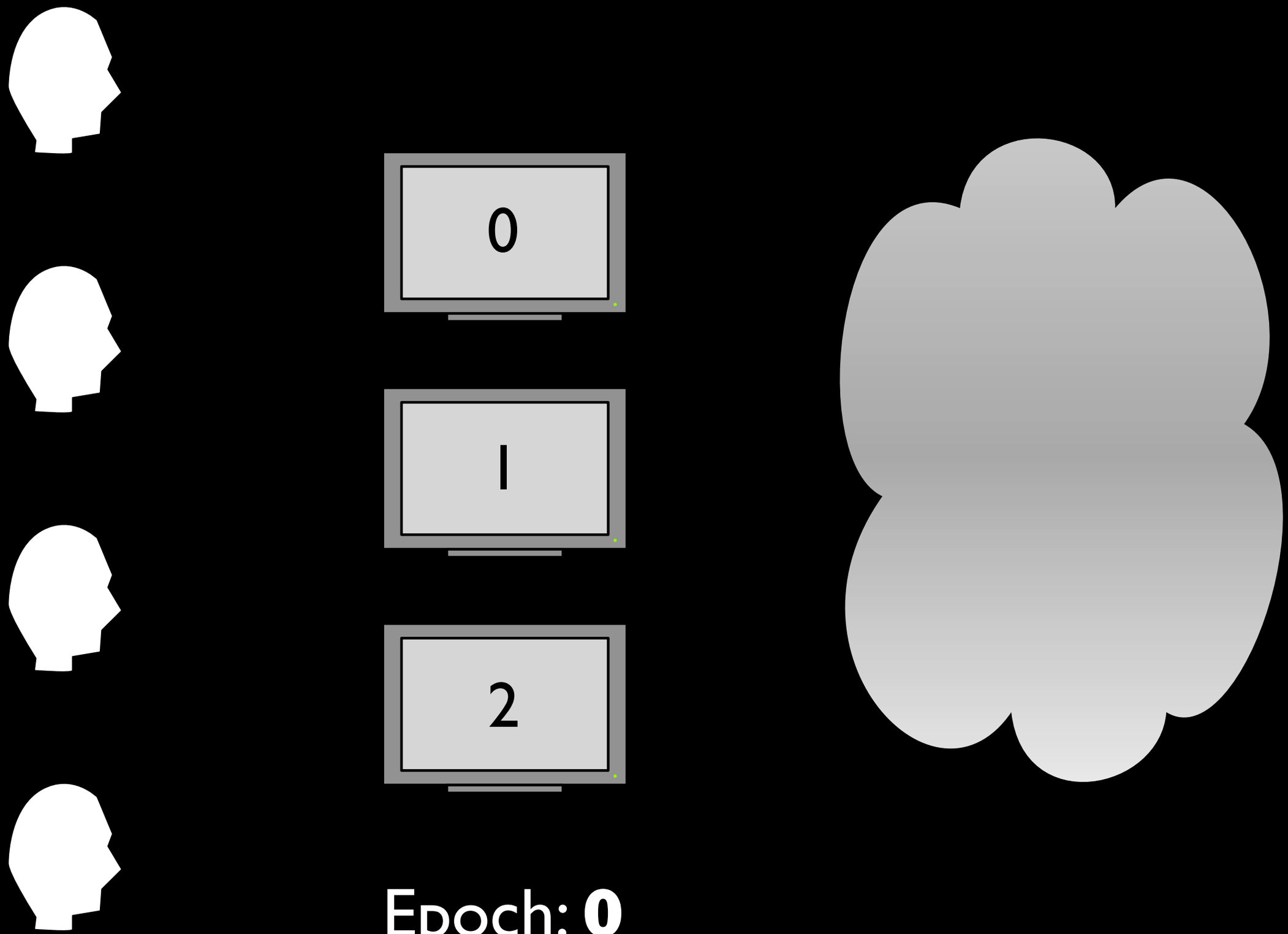
Map Maintenance Messages

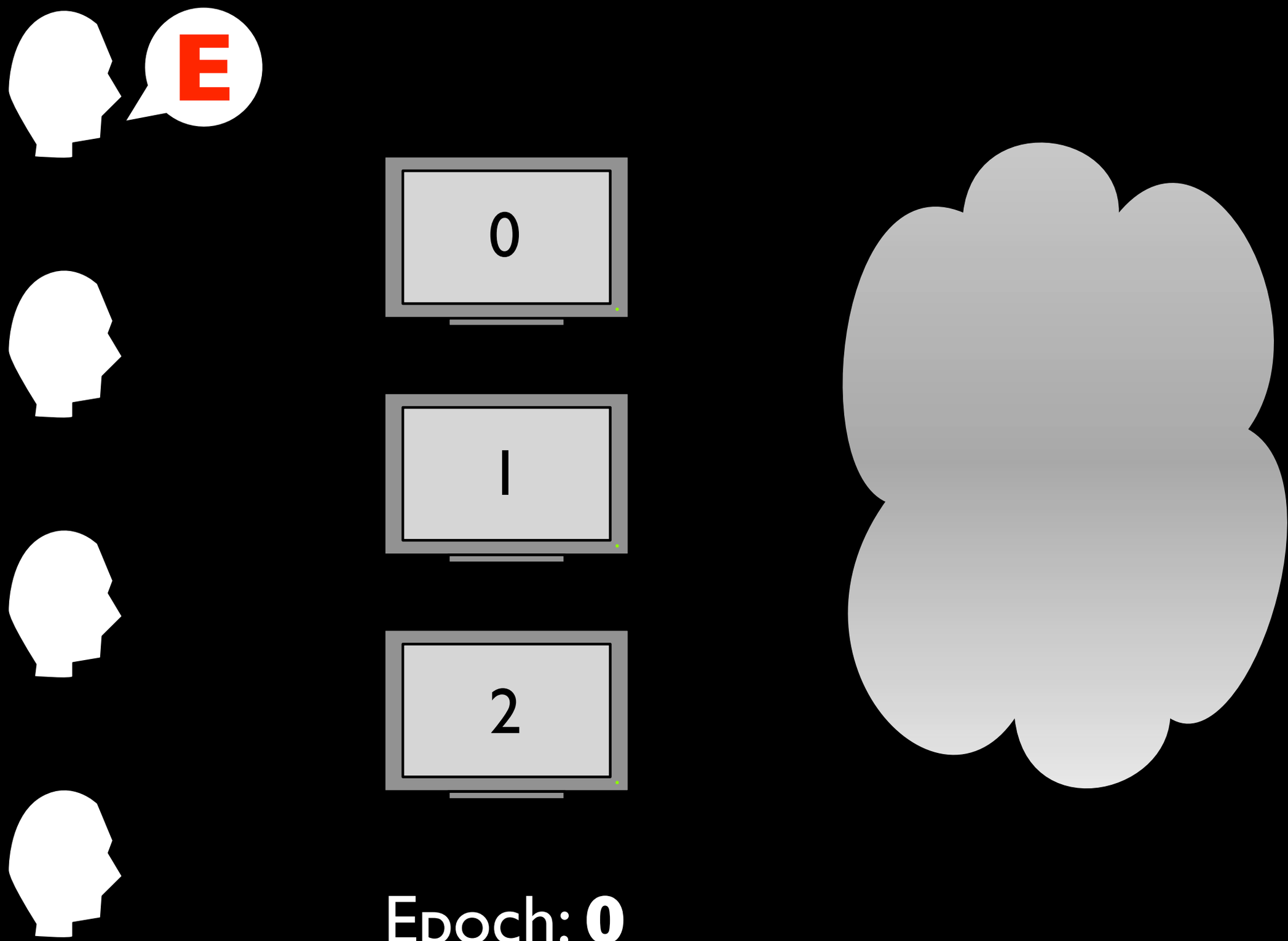
System Design

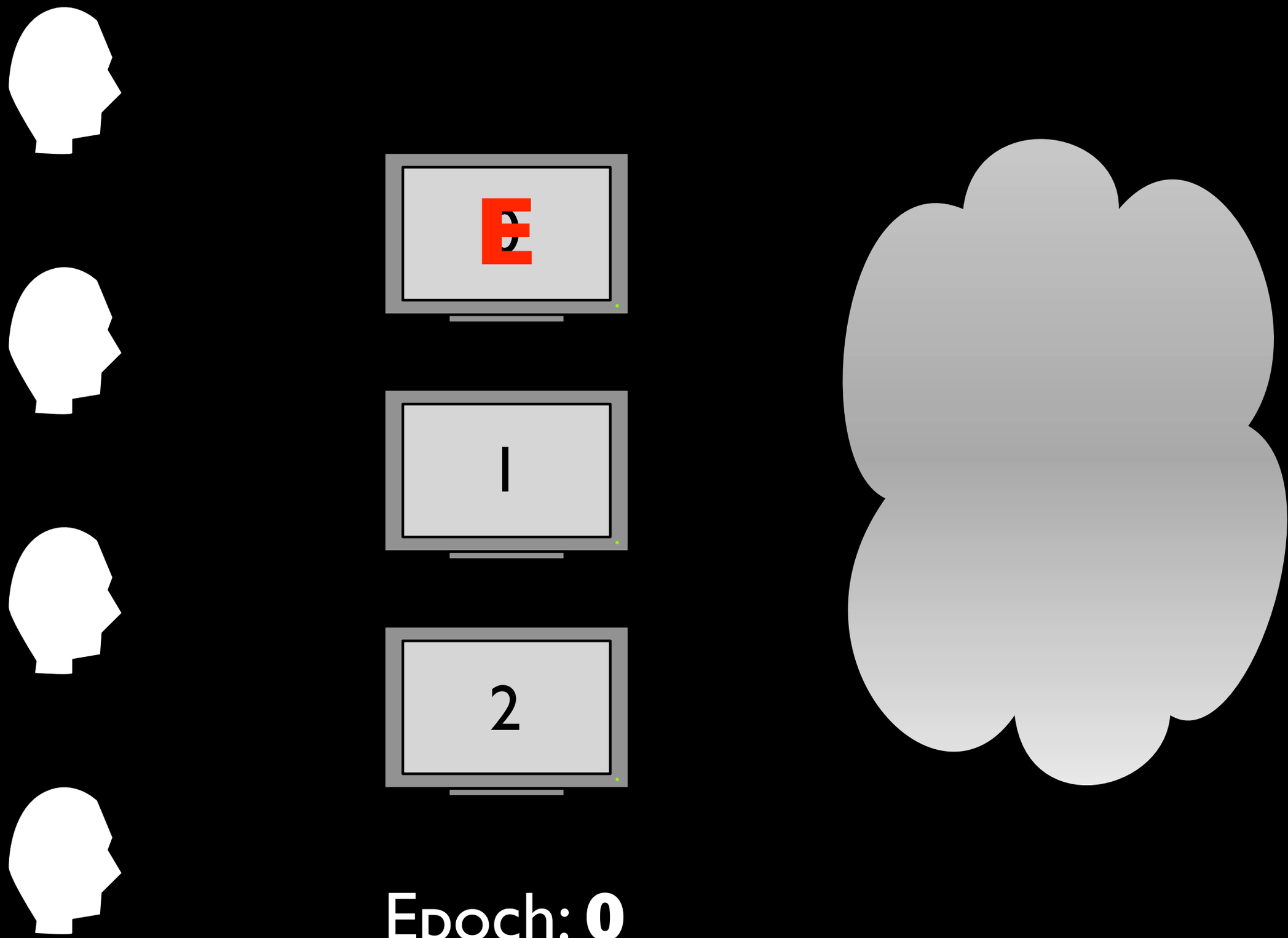
Hybrid Consistency

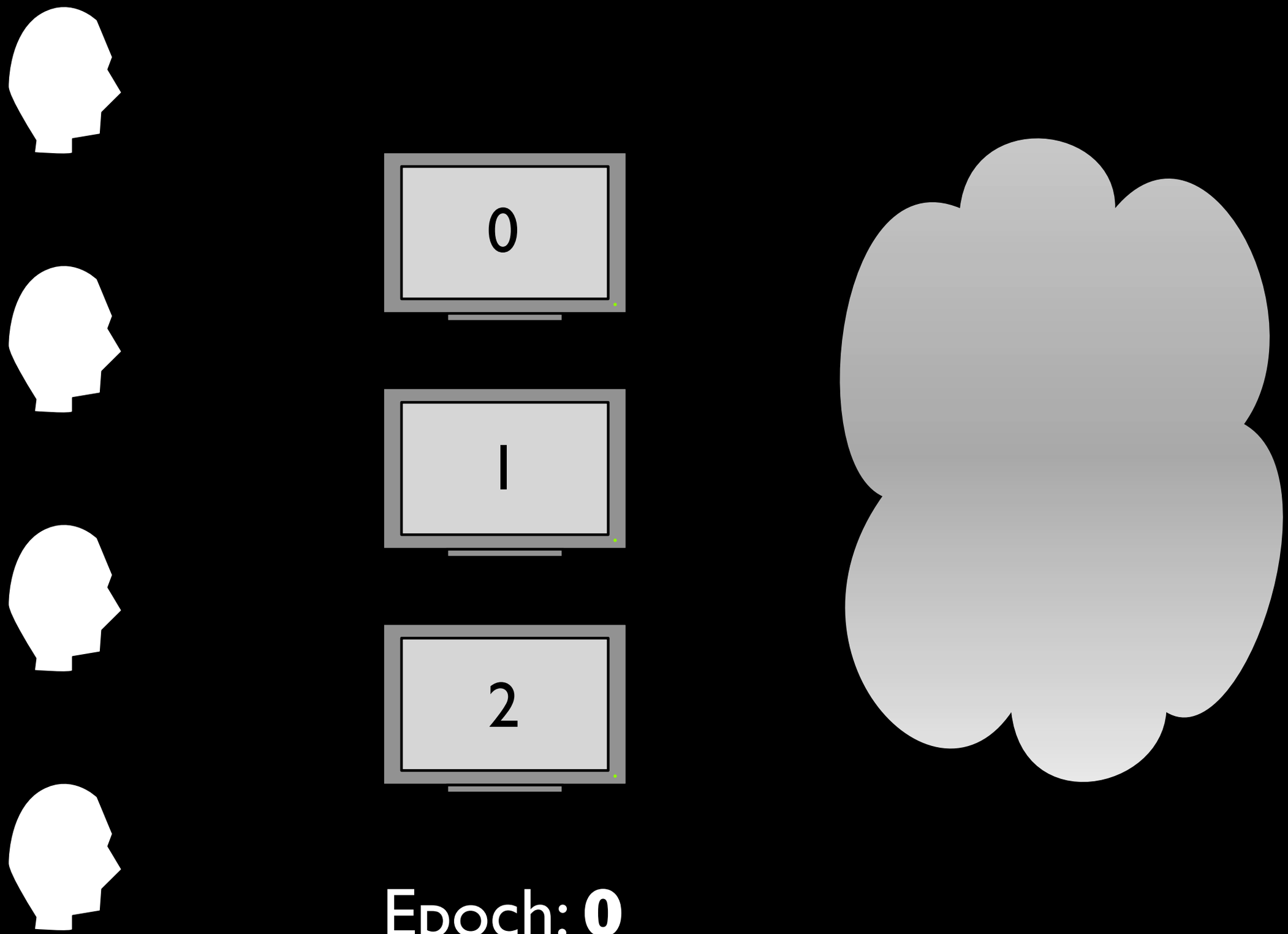


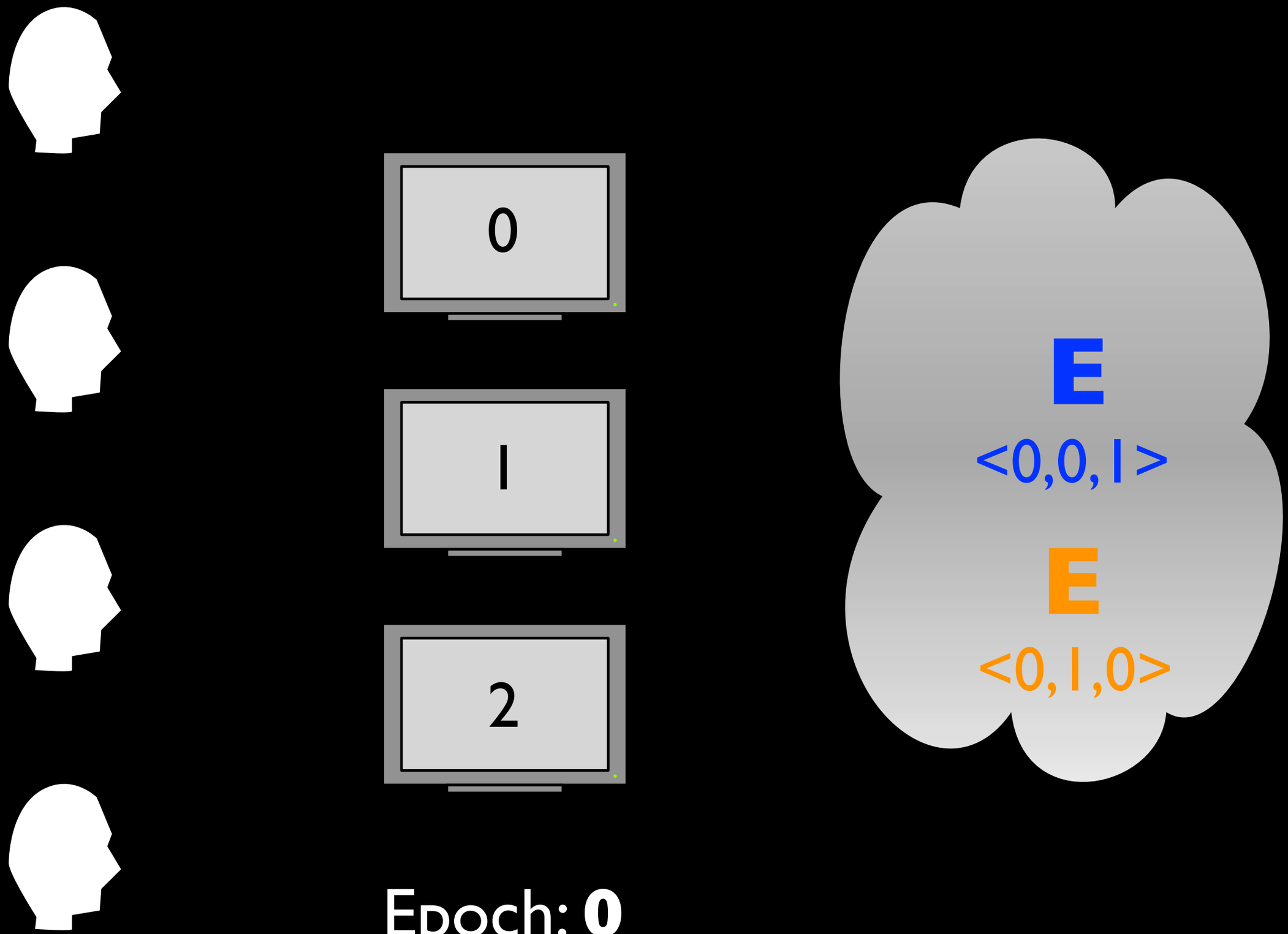

0
1
2


0
1
2
Epoch: 0


E
0
1
2
Epoch: 0

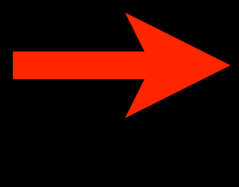

E
1
2
Epoch: 0


0
1
2
Epoch: 0


0
1
2
E
<0,0,1>
E
<0,1,0>
Epoch: 0

E

<0,1,3>



Map₂[1,3] += 2

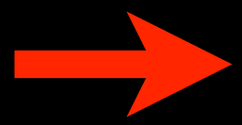
<0,2,4>

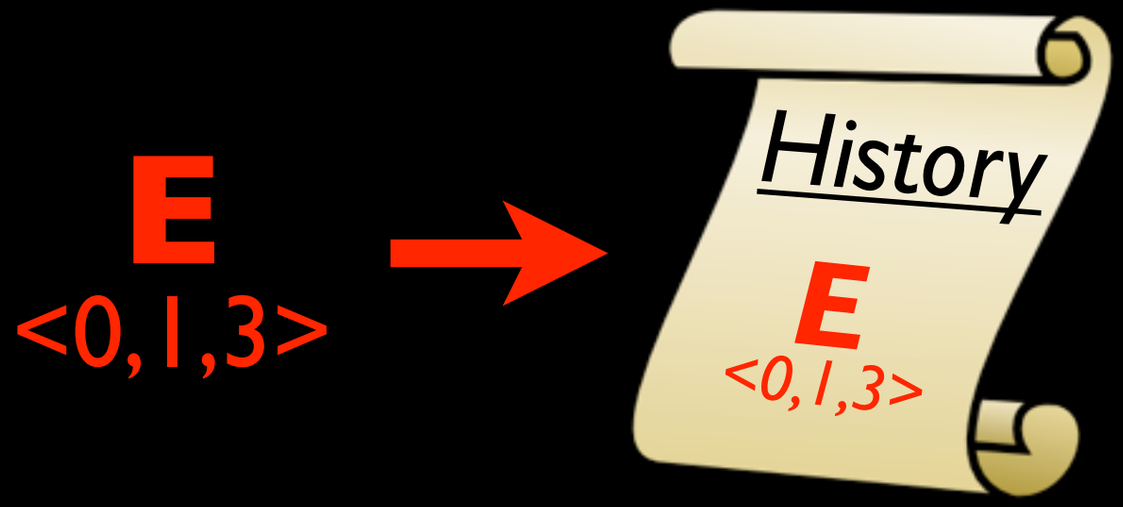
E

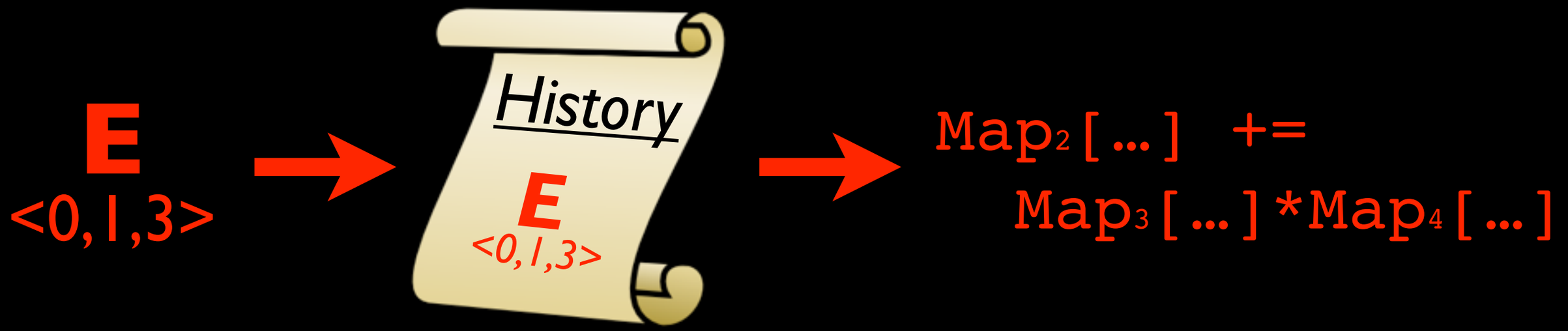
<0,1,3>

E

<0,1,3>




E
<0,1,3>
History
E
<0,1,3>


E
<0,1,3>
History
E
<0,1,3>
Map2[…] +=
Map3[…]*Map4[…]

E
<0,1,3>

$Map_2[\ldots]$ += $Map_3[\ldots] * Map_4[\ldots]$

$\Sigma\delta$

$Map_3[\ldots]$
$Map_4[\ldots]$

E
<0,1,3>

$Map_2[\ldots]$ += $Map_3[\ldots] * Map_4[\ldots]$

$\Sigma\delta$

$Map_3[\ldots]$
$Map_4[\ldots]$

$Map_2[\ldots]$ += …
<0,1,3>

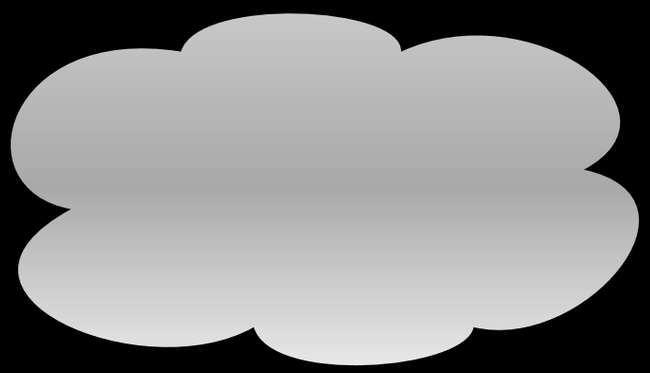
$Map_3[\ldots] * Map_4[\ldots]$

E
<0,1,3>

$Map_2[\ldots]$ += $Map_3[\ldots]$*$Map_4[\ldots]$

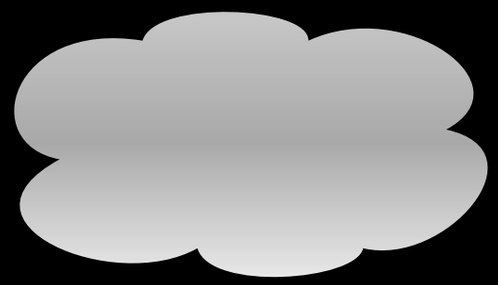
$\Sigma\delta$

$Map_3[\ldots]$
$Map_4[\ldots]$



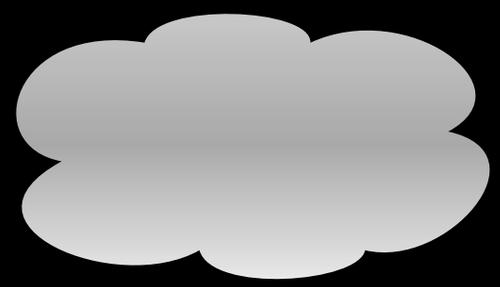
$Map_2[\ldots]$ += …
<0,1,3>

$Map_3[\ldots]$*$Map_4[\ldots]$



→ **$Map_2[1,3]$ += 2**

<0,2,4>

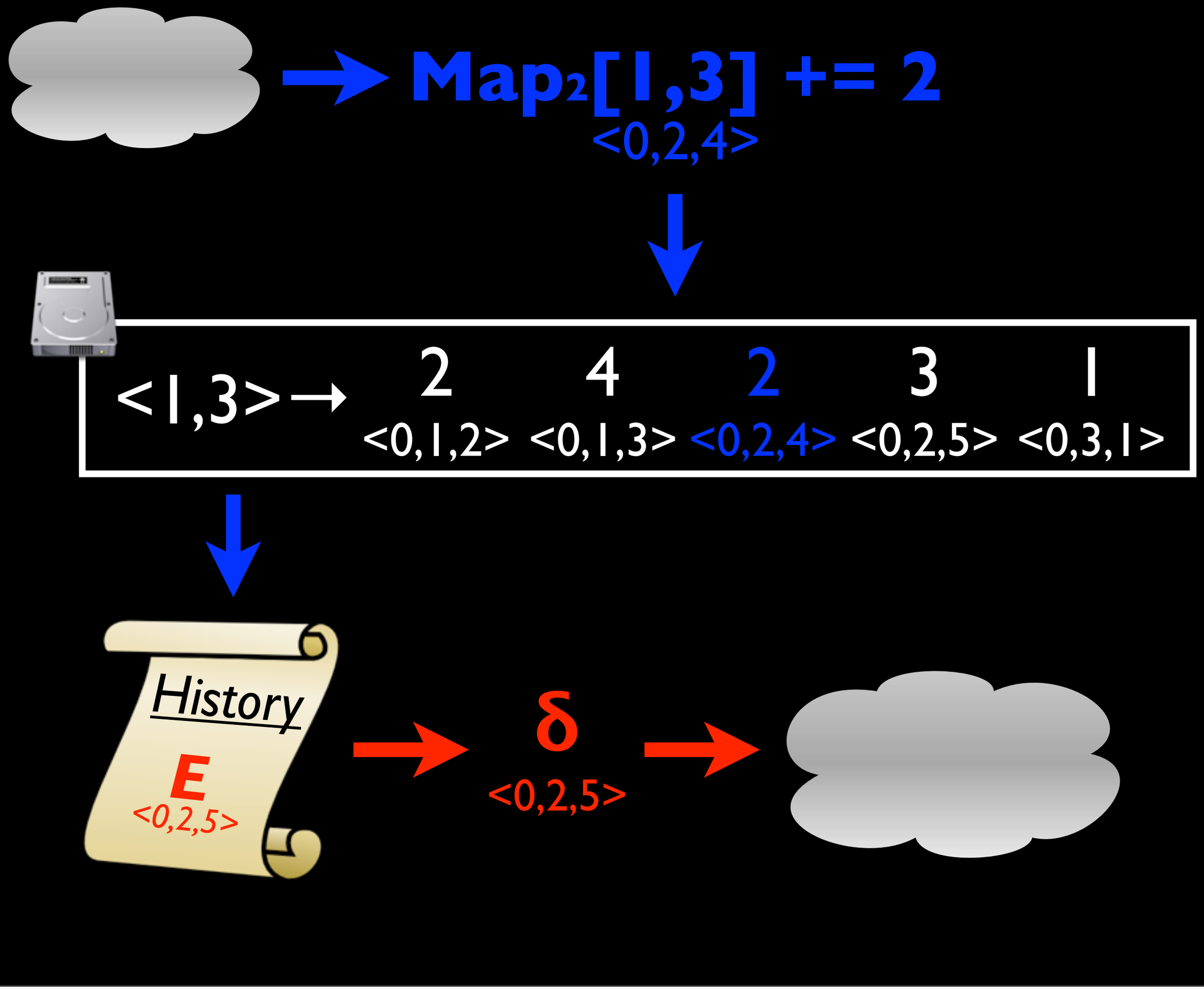


→ **Map₂[1,3] += 2**

<0,2,4>

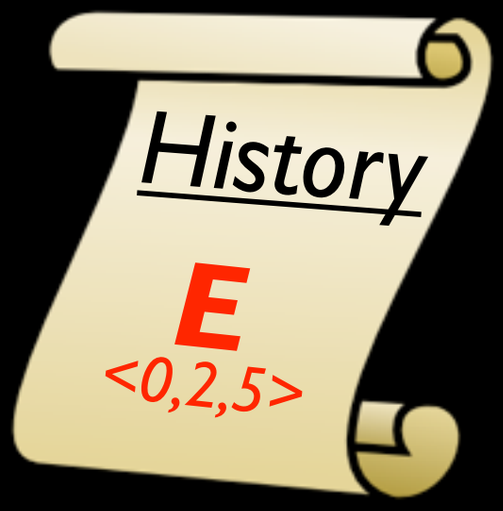
<1,3>→ 2 4 2 3 1

<0,1,2> <0,1,3> <0,2,4> <0,2,5> <0,3,1>

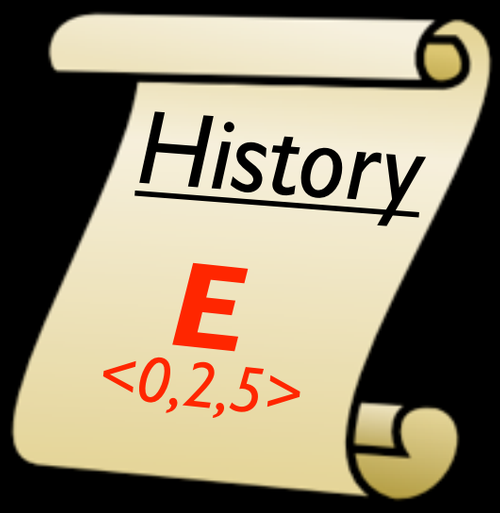

Map2[1,3] += 2
<0,2,4>
<1,3>→
2
<0,1,2>
4
<0,1,3>
2
<0,2,4>
3
<0,2,5>
1
<0,3,1>
History
E
<0,2,5>
δ
<0,2,5>

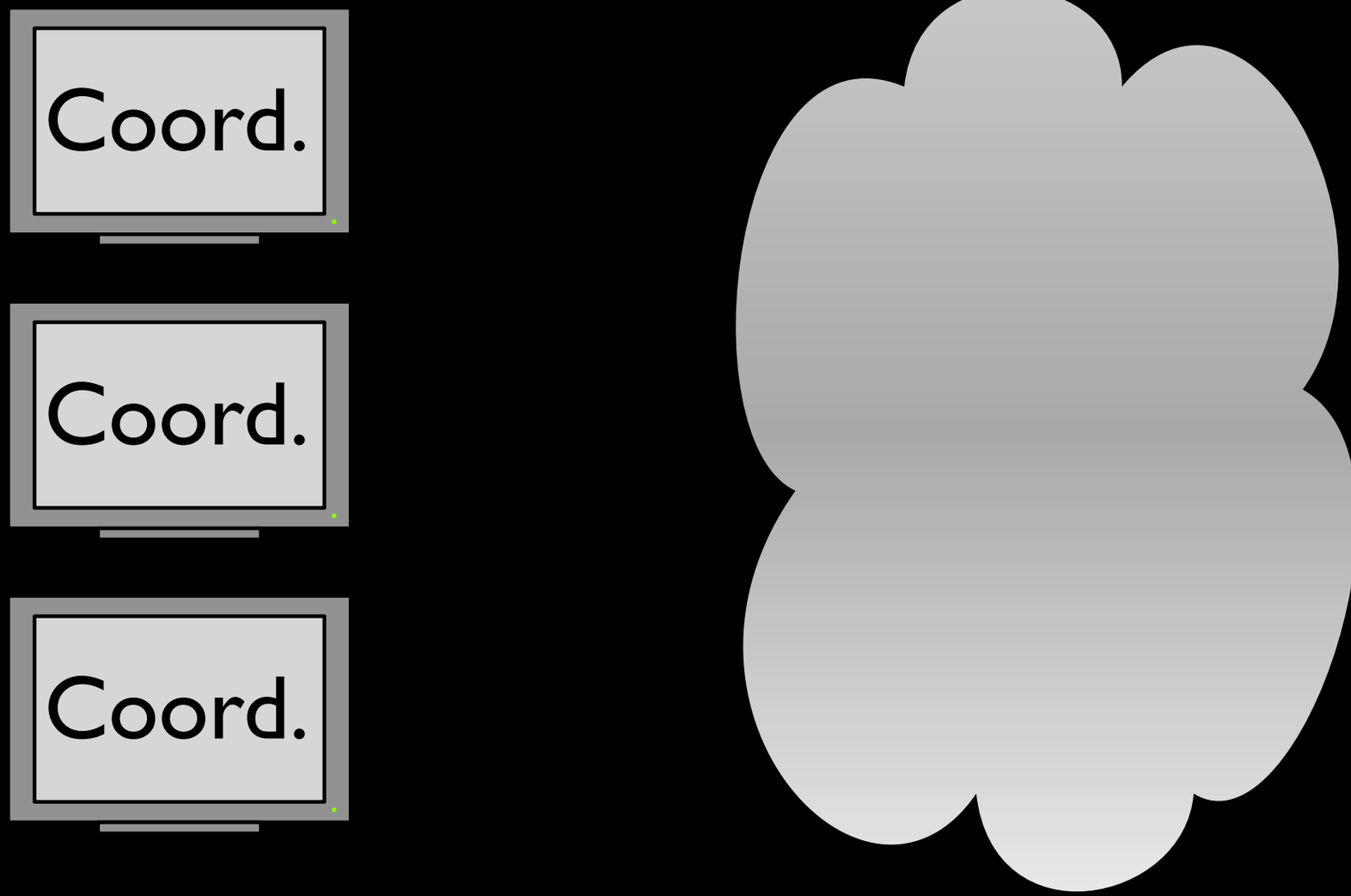
<1,3>→ 2 4 2 3 1

<0,1,2> <0,1,3> <0,2,4> <0,2,5> <0,3,1>




<1,3>→
2
<0,1,2>
4
<0,1,3>
2
<0,2,4>
3
<0,2,5>
1
<0,3,1>


History
E
<0,2,5>


Coord.
Coord.
Coord.

Coord.

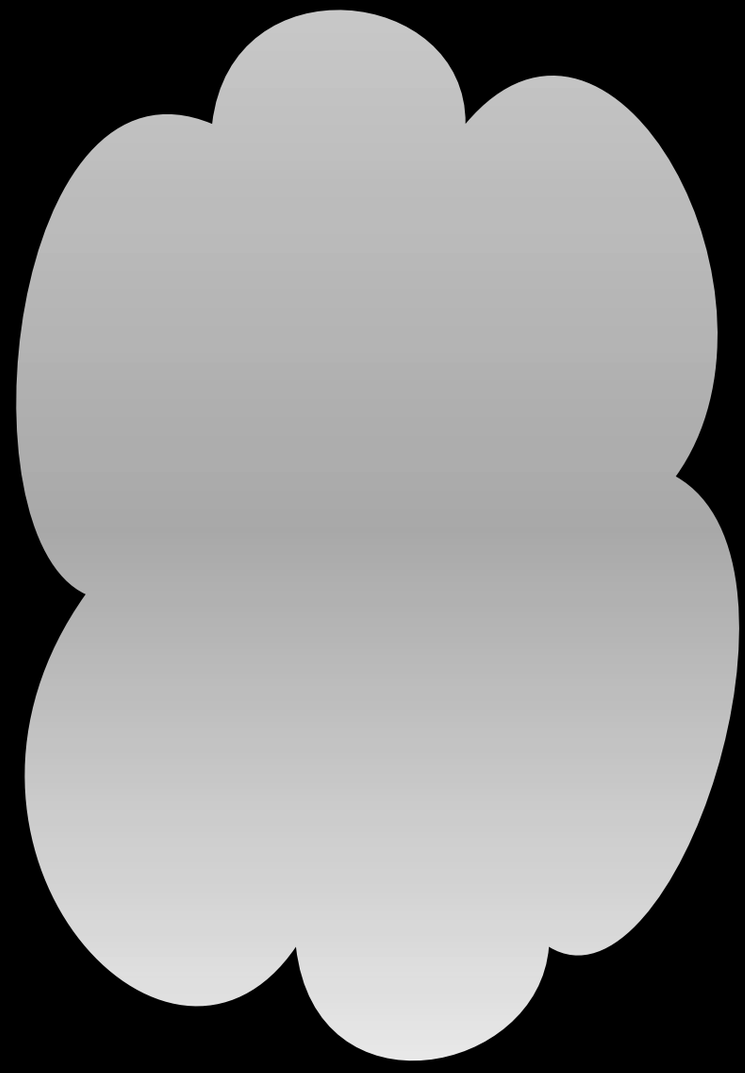
!

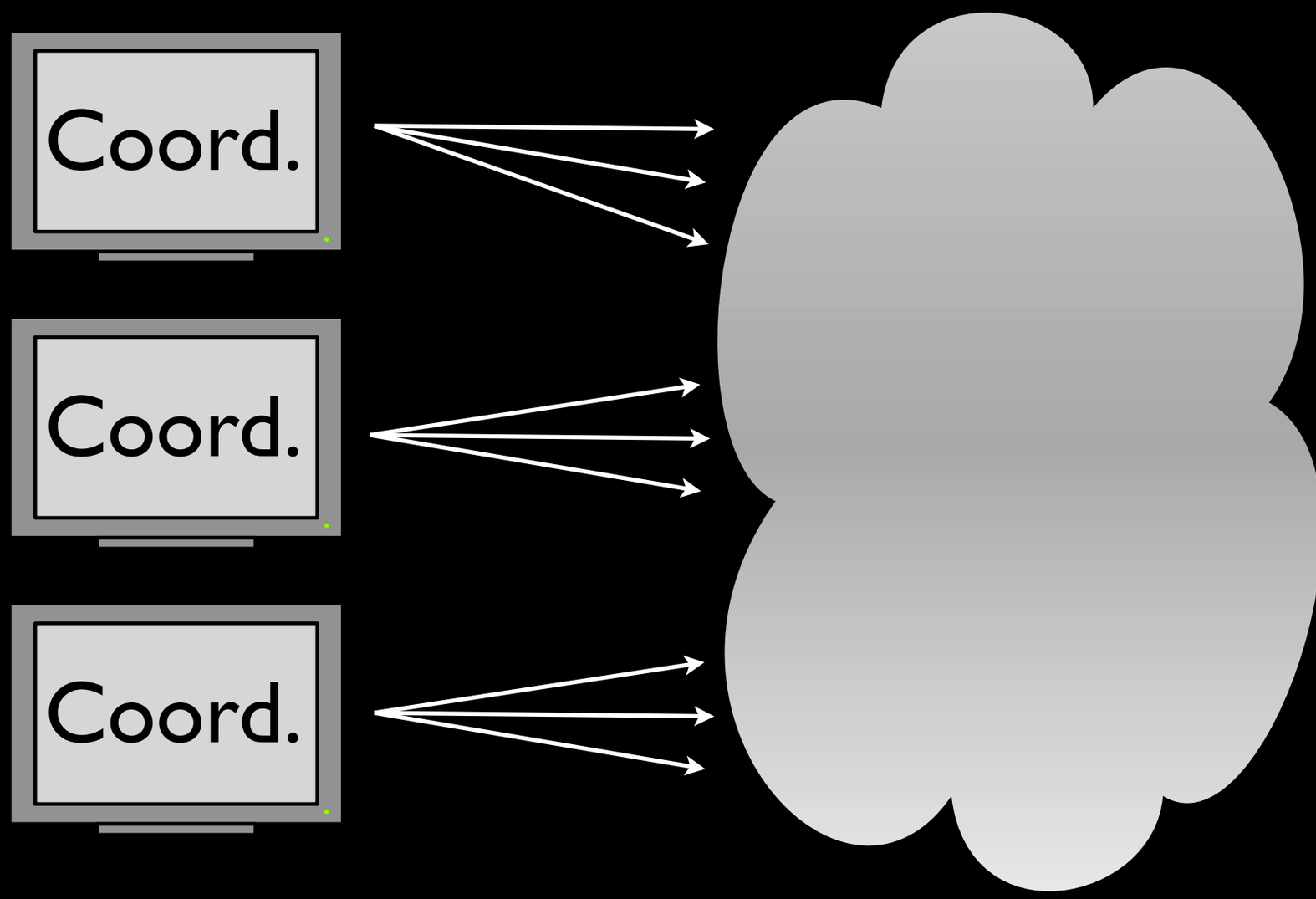
Coord.

!

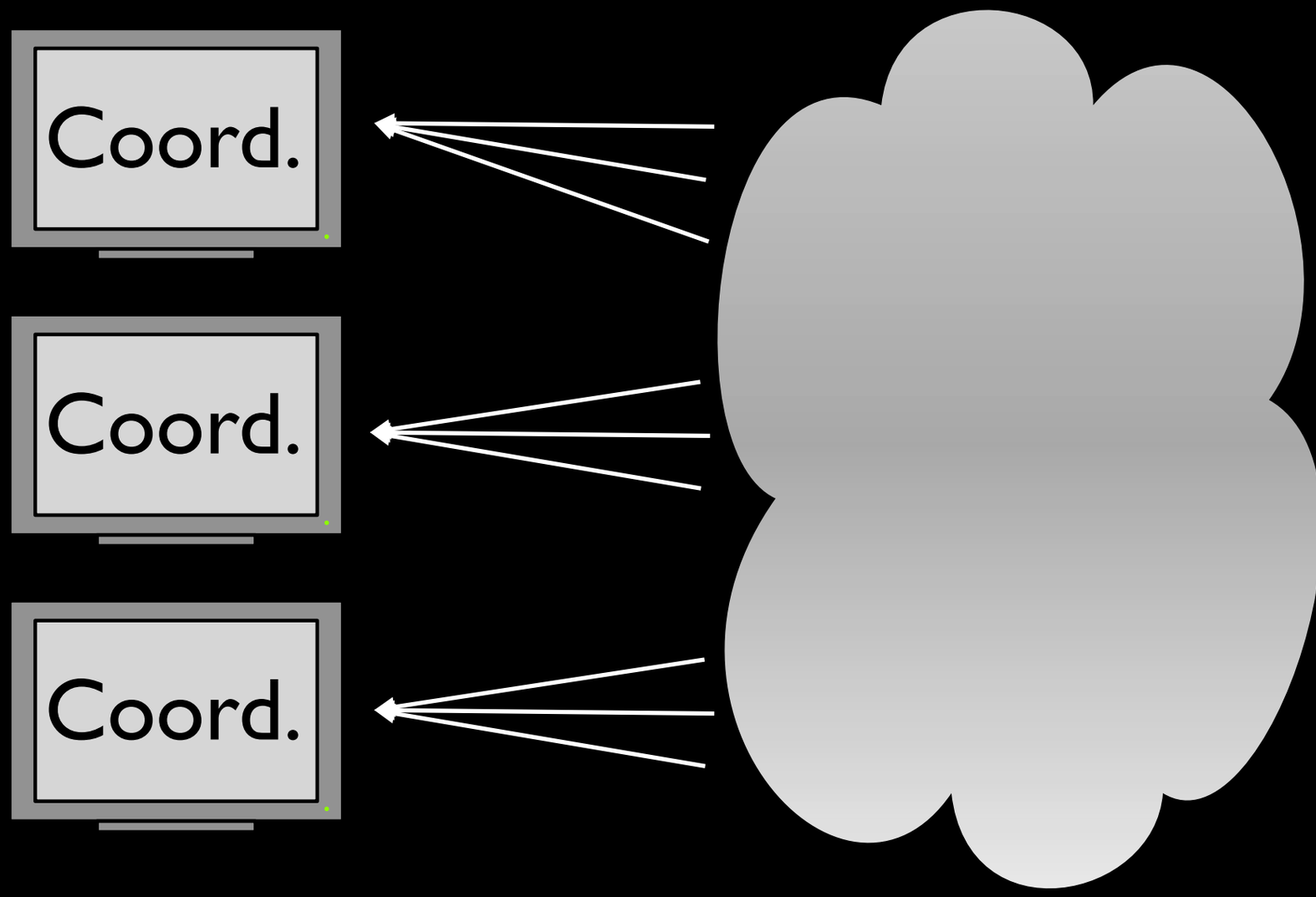
Coord.

!

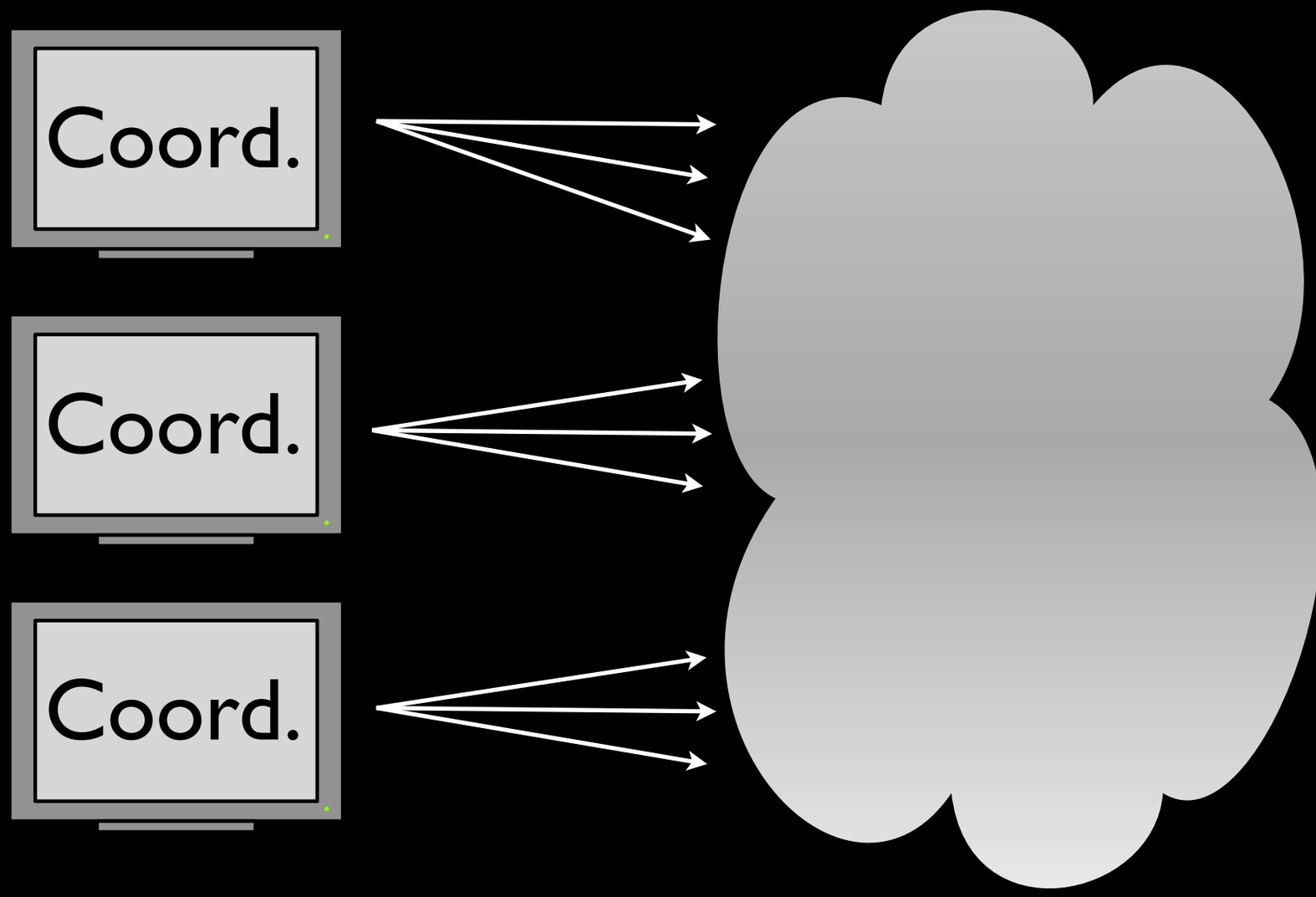


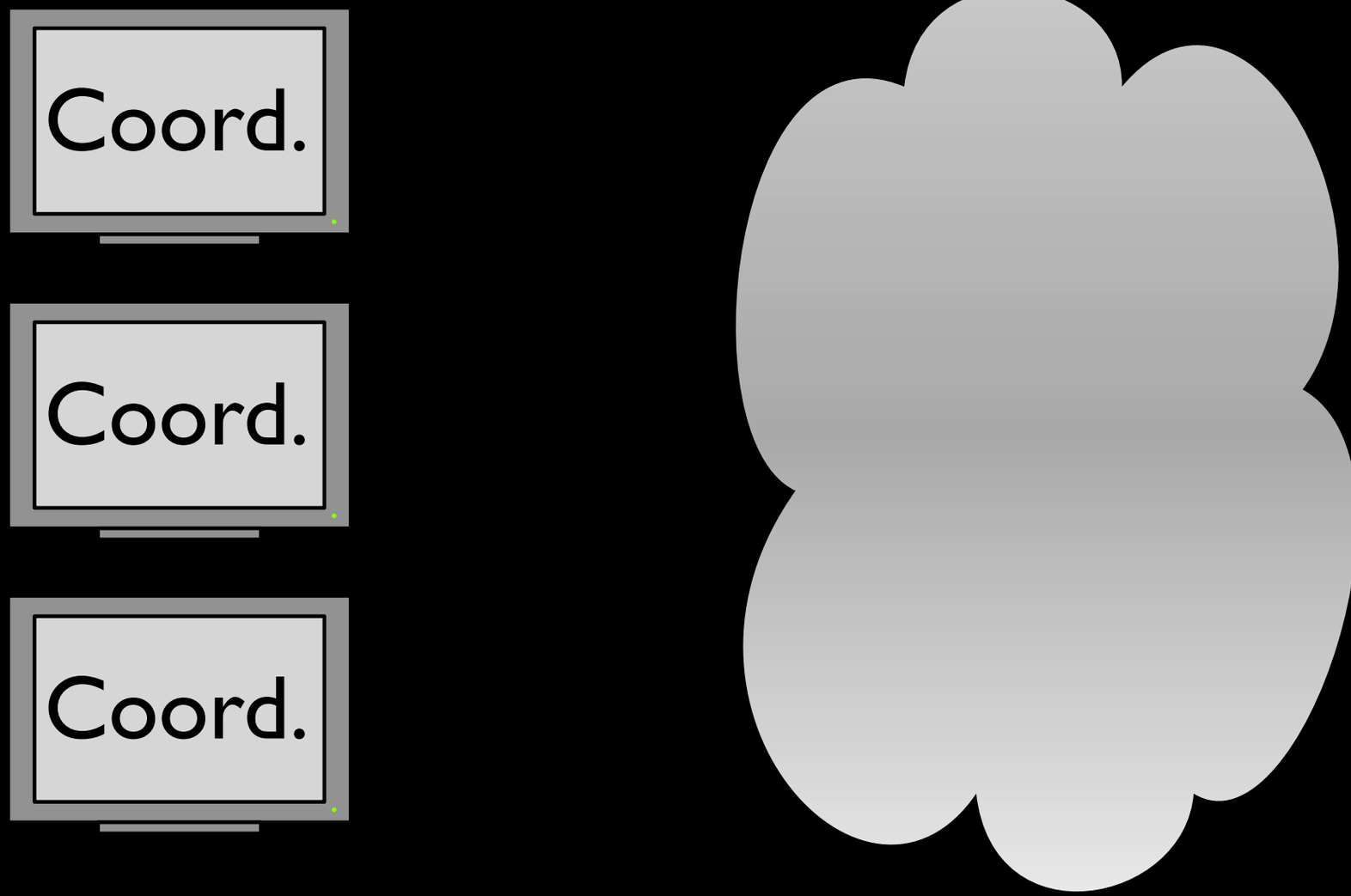


Epoch Commit

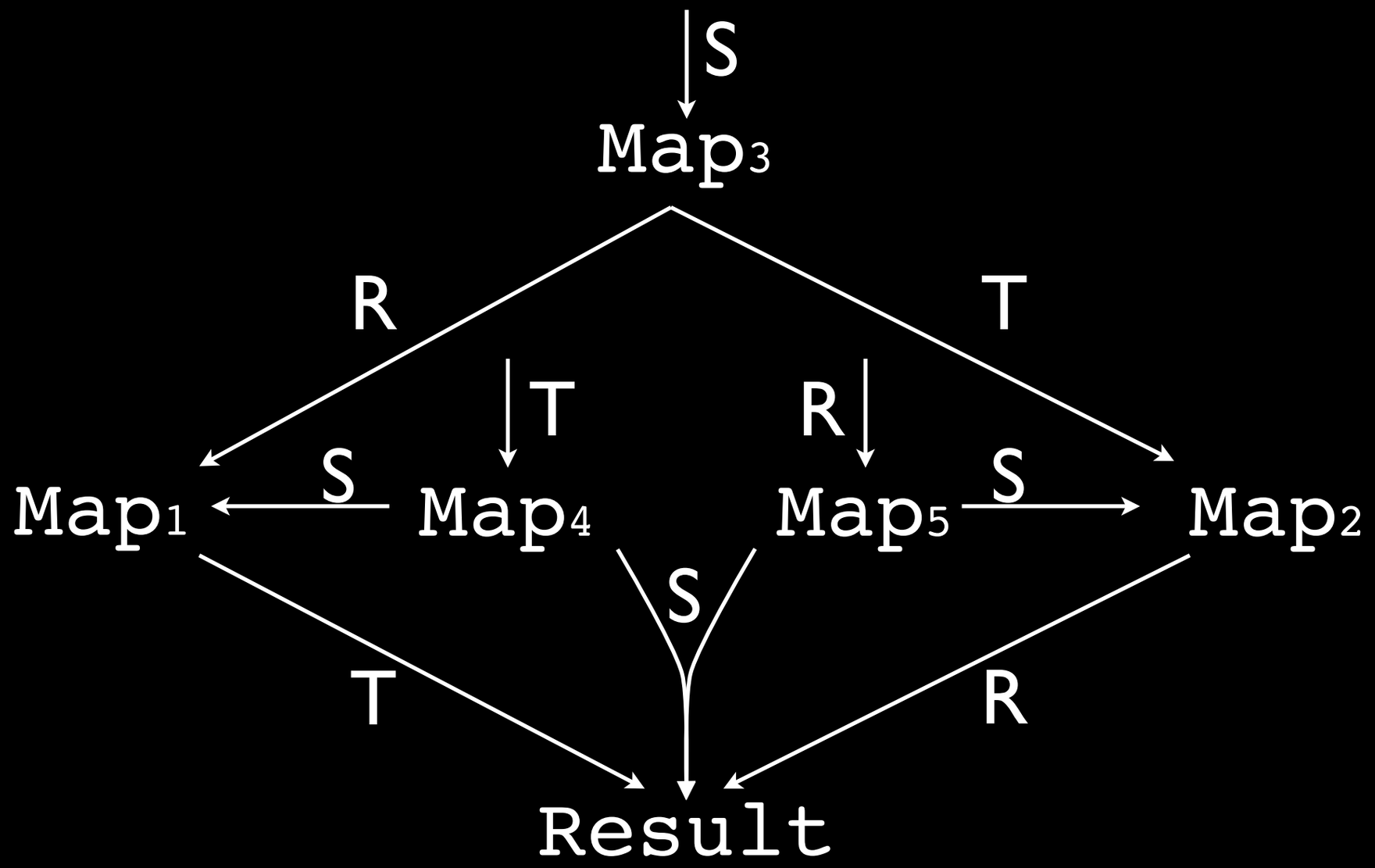


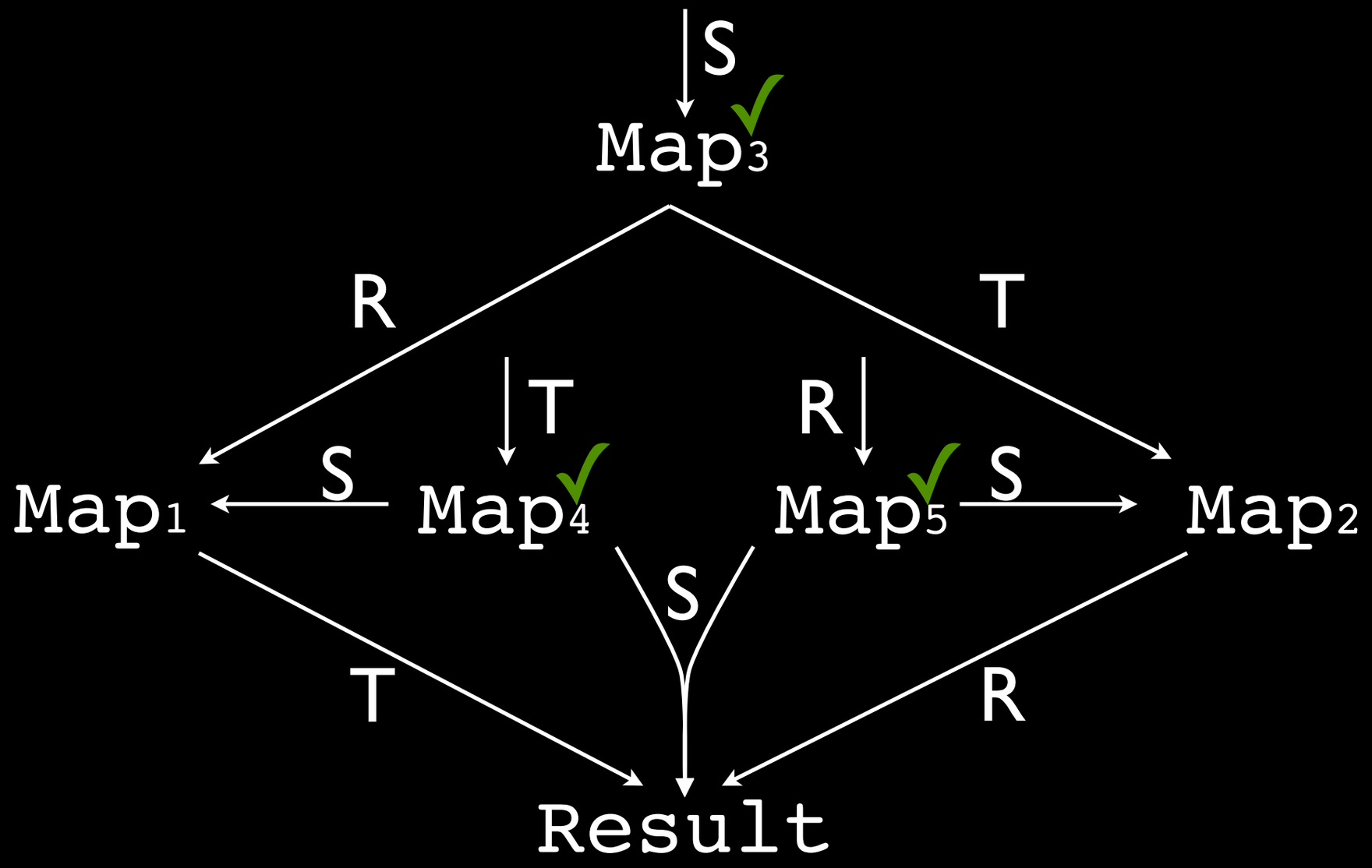
Commit OK

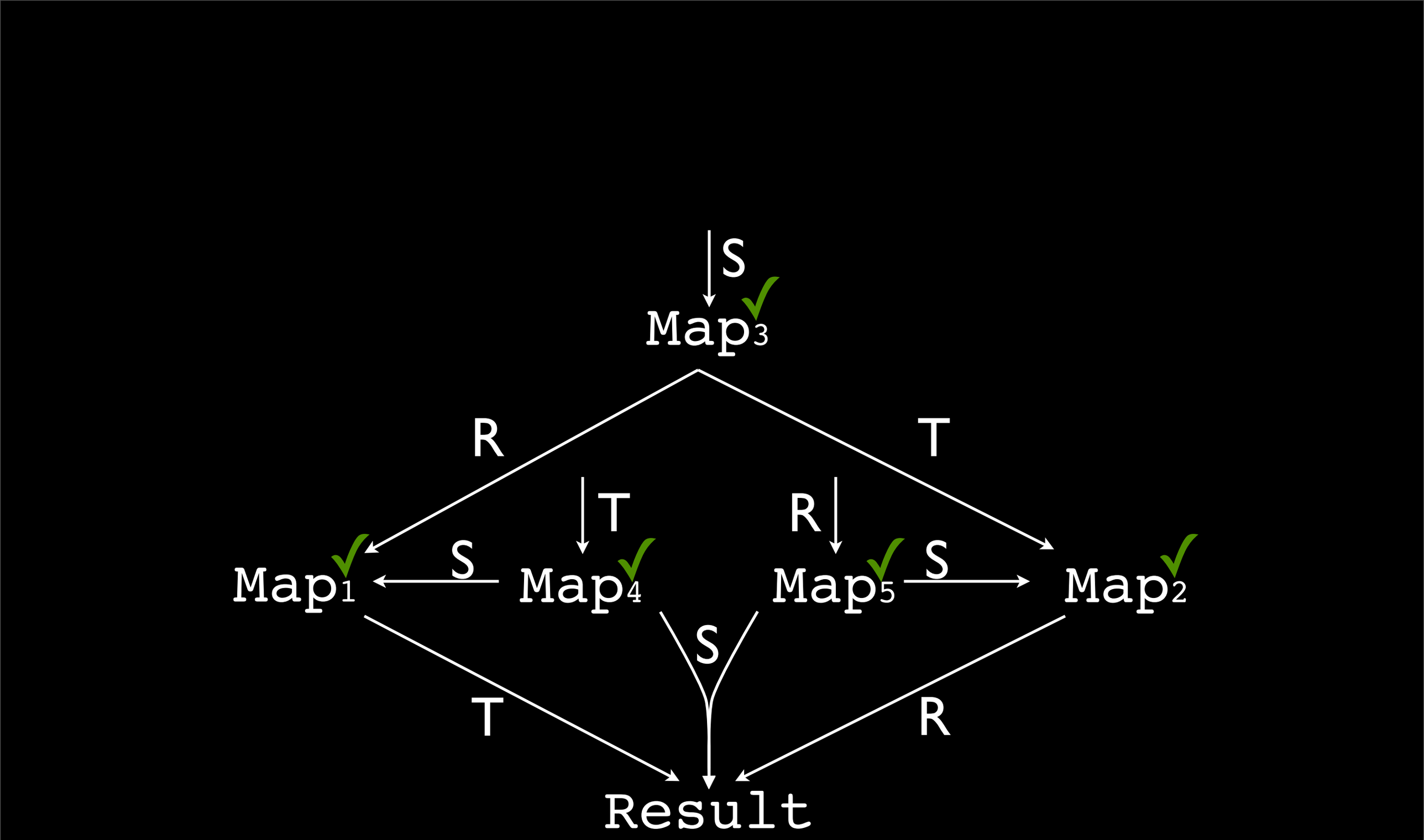

Coord.
Coord.
Coord.
Epoch End

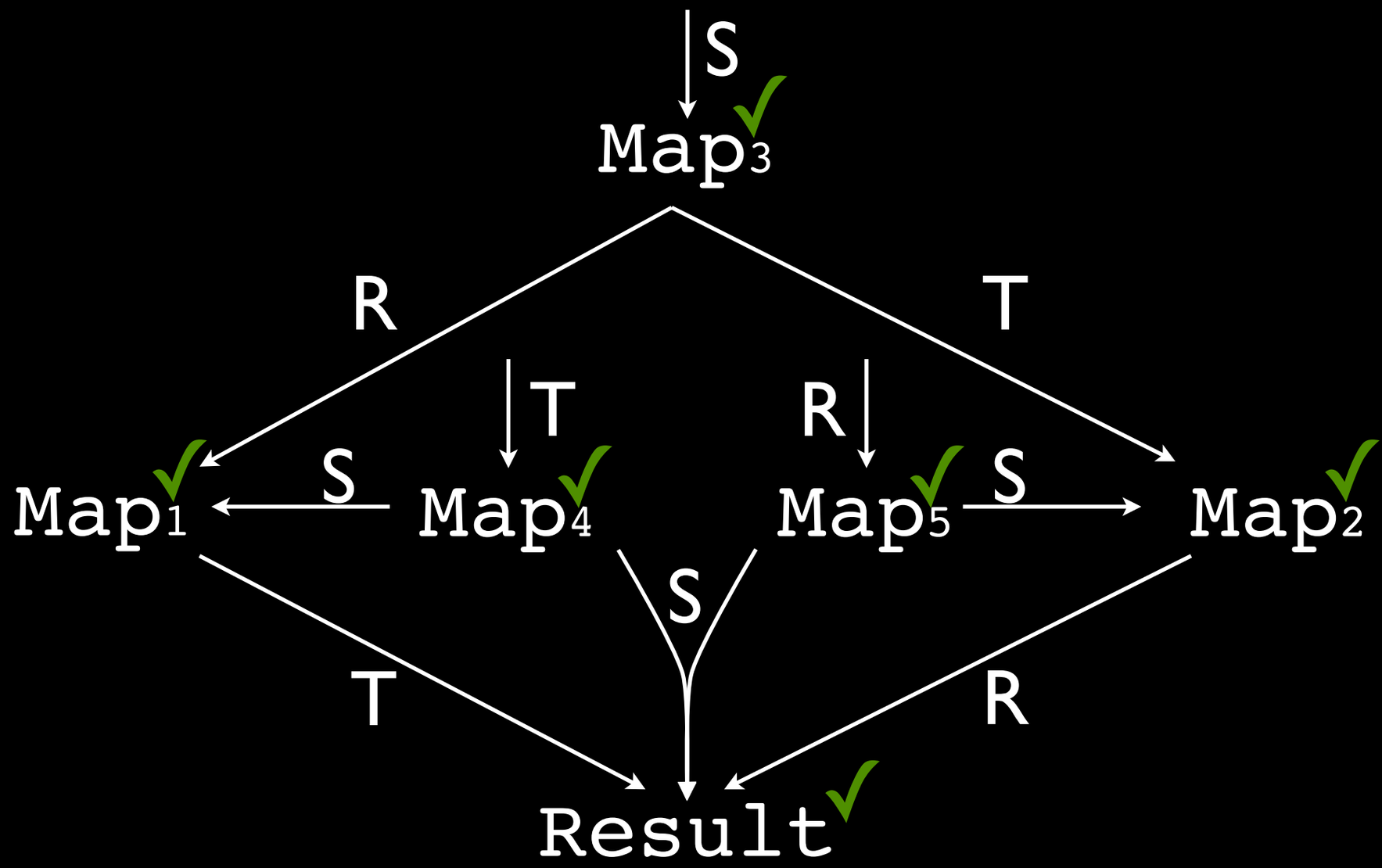

Coord.
Coord.
Coord.

```
ON Insert_R(A,B) DO {
  Result[]  += A * Map2[B]
  Map1[C]   += A * Map3[B,C]
  Map5[B]   += A }

ON Insert_T(C,D) DO {
  Result[] += Map1[C] * D
  Map2[B]  += D * Map3[B,C]
  Map4[C]  += D }

ON Insert_S(B,C) DO {
  Result[]  += Map5[B] * Map4[C]
  Map1[C]   += Map5[B]
  Map2[B]   += Map4[C]
  Map3[B,C] += 1 }
```


S
Map3
R
T
T
R
S
S
Map1
Map4
Map5
Map2
S
T
R
Result


S
Map3
R
T
T
R
S
Map1
Map4
Map5
S
Map2
S
T
R
Result


S
Map3
R
T
T
R
S
Map1
Map4
Map5
S
Map2
S
T
R
Result


S
Map3
R
T
T
R
S
Map1
Map4
Map5
S
Map2
S
T
R
Result

<1,3> → 2 4 2 3 1

<0,1,2> <0,1,3> <0,2,4> <0,2,5> <0,3,1>

<1,3>→ 12

<Epoch 0>

Cumulus

- Additive deltas streamline distribution of query processing.
- Key/Value stores and query processing engines compliment each other.
- Cumulus' hybrid consistency engine supports both low-latency and high-precision applications.

Laasie (and BarQL)

(with Luke Ziarek, Sumit Agarwal, and Daniel Bellinger)

Collaborative Web Applications

Logos are the property of their respective owners

Collaborative Web Applications

Dropbox

Google Docs

Twitter

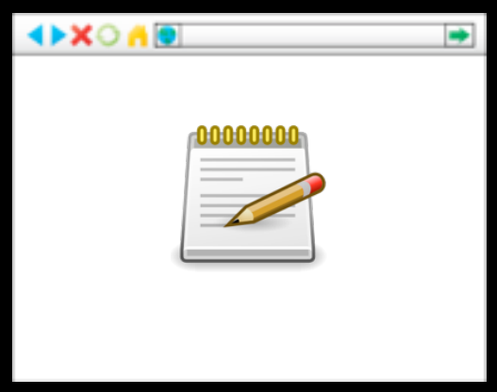
Office 365

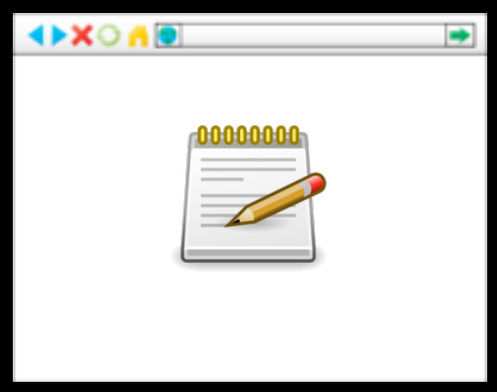
Google Wave

Facebook

Logos are the property of their respective owners

Collaborative Web Applications





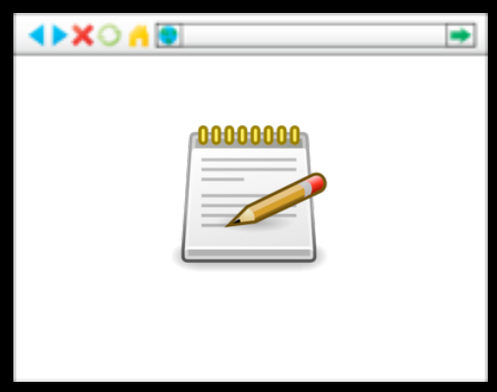
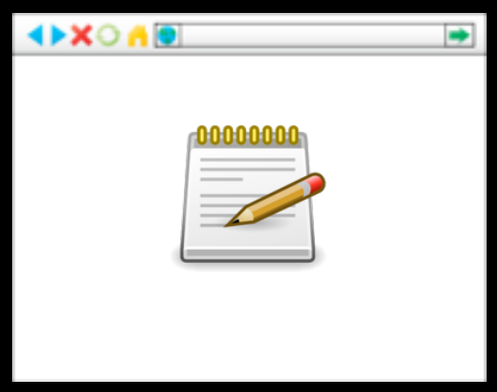
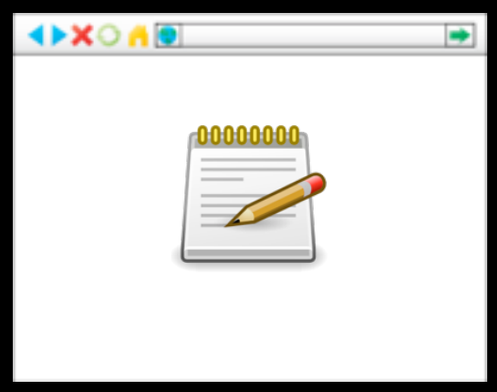
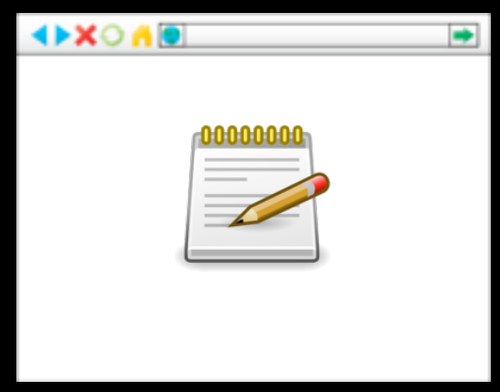


Image Source: openclipart.org, OpenIconLibrary

Collaborative Web Applications



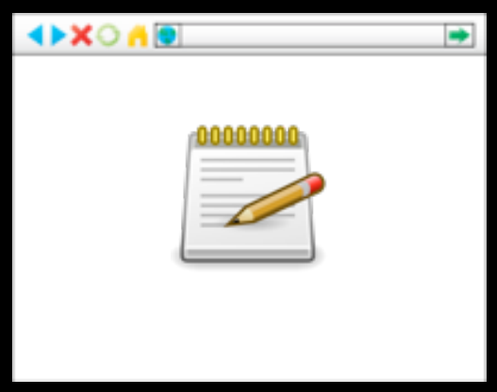


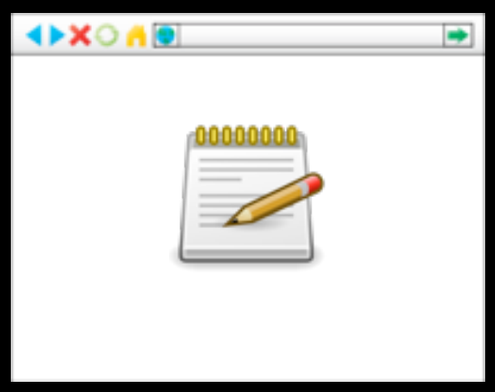


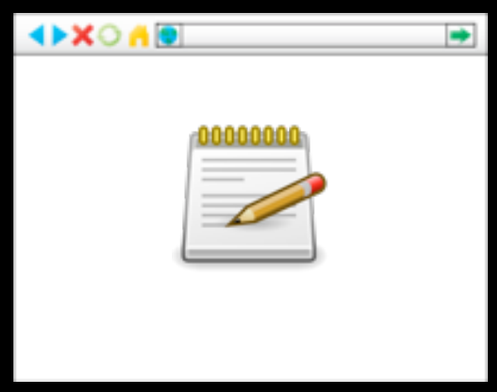
The Application Lives in the Browser

Image Source: openclipart.org, OpenIconLibrary

Collaborative Web Applications



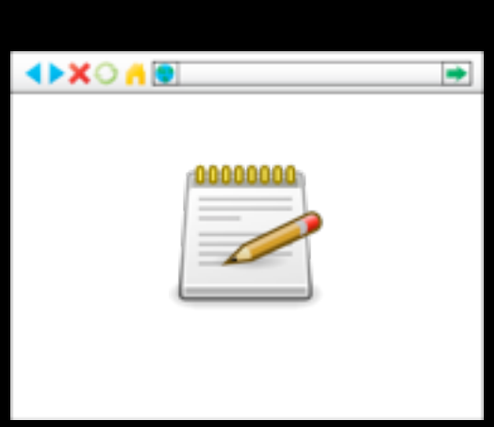




The Application Lives in the Browser
The Server Just Relays and Persists Application State

Image Source: openclipart.org, OpenIconLibrary

Collaborative Applications



Request Updates for `Q(state)`

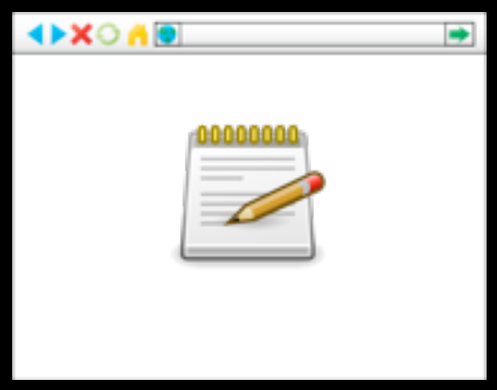
{Update 1, Update 2, …}

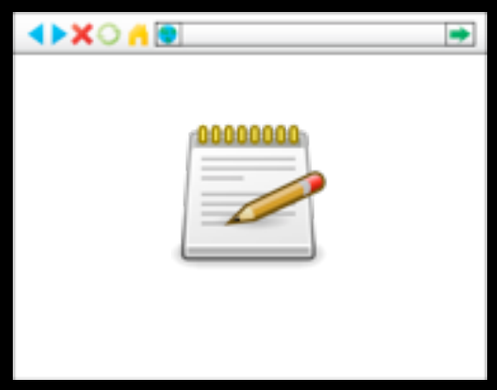
Post Update: `state := Q(state)`

Success!

Image Source: openclipart.org, OpenIconLibrary

State as a Log





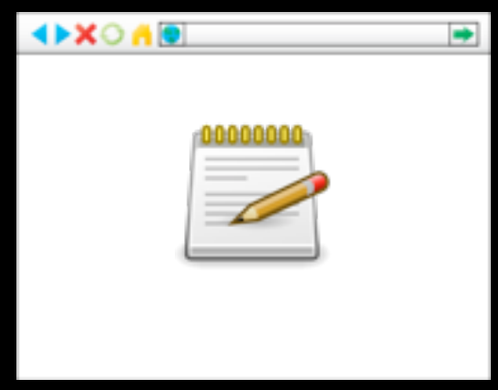
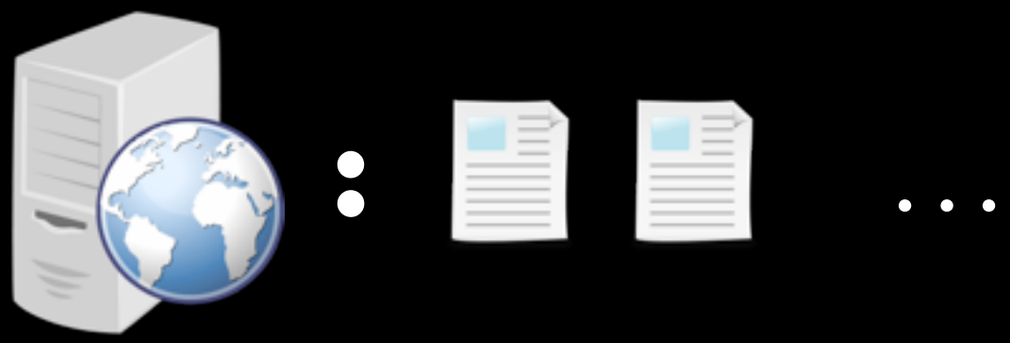


Image Source: openclipart.org, OpenIconLibrary

State as a Log

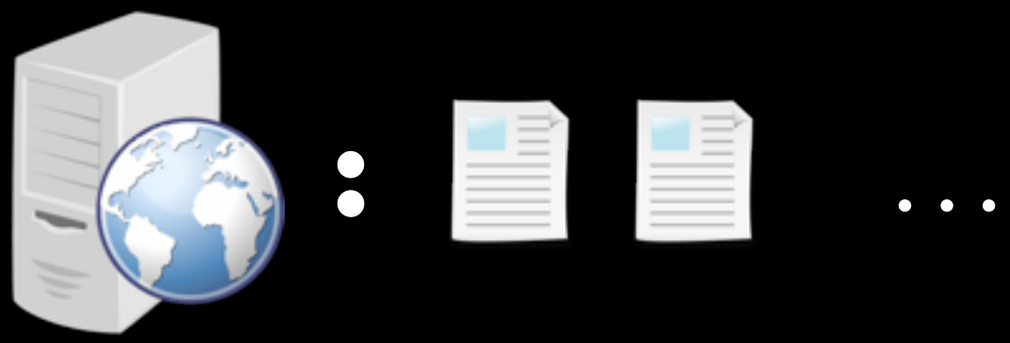


Clients Log State Updates

The Server Relays Log Entries

Image Source: openclipart.org, OpenIconLibrary

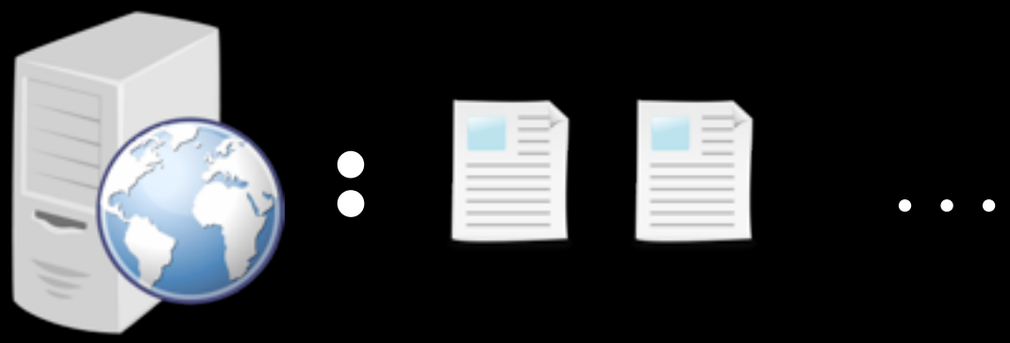
State as a Log



Clients Can Leave Unexpectedly

Image Source: openclipart.org, OpenIconLibrary

State as a Log

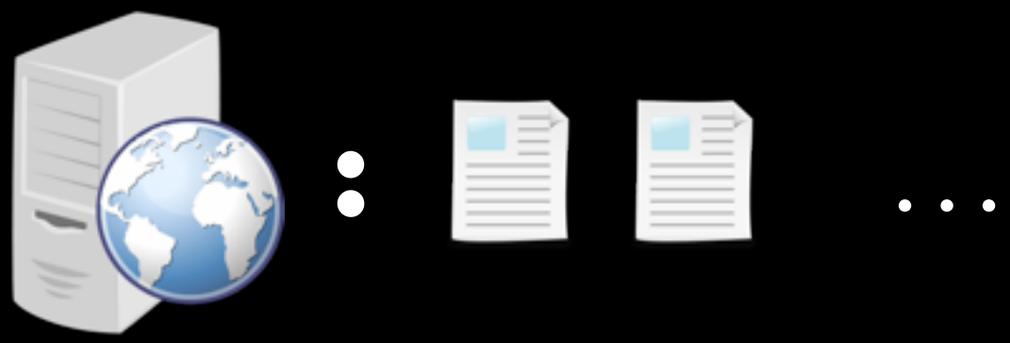


Clients Can Leave Unexpectedly

When Clients Arrive, the Server Can Restore Their State

Image Source: openclipart.org, OpenIconLibrary

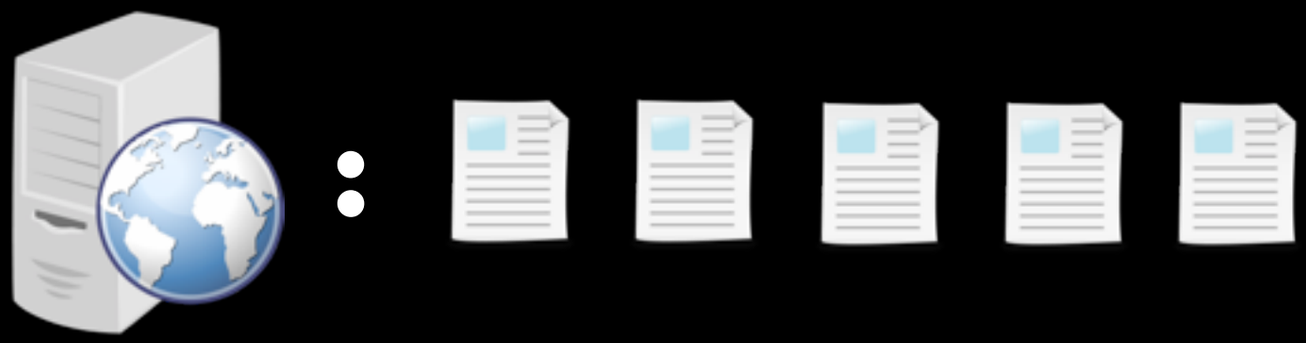
State as a Log



But Clients Can Leave For Other Reasons

Image Source: openclipart.org, OpenIconLibrary

State as a Log



But Clients Can Leave For Other Reasons

...And Only Need The Latest Changes When It Returns

Image Source: openclipart.org, OpenIconLibrary

Log as a Service

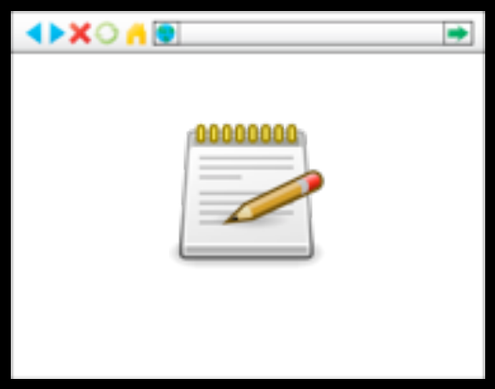


Image Source: openclipart.org, OpenIconLibrary

Log as a Service

I need all docs

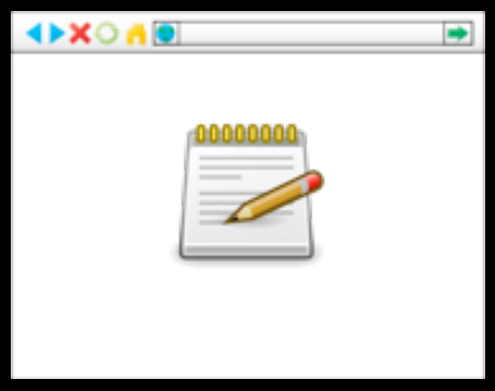
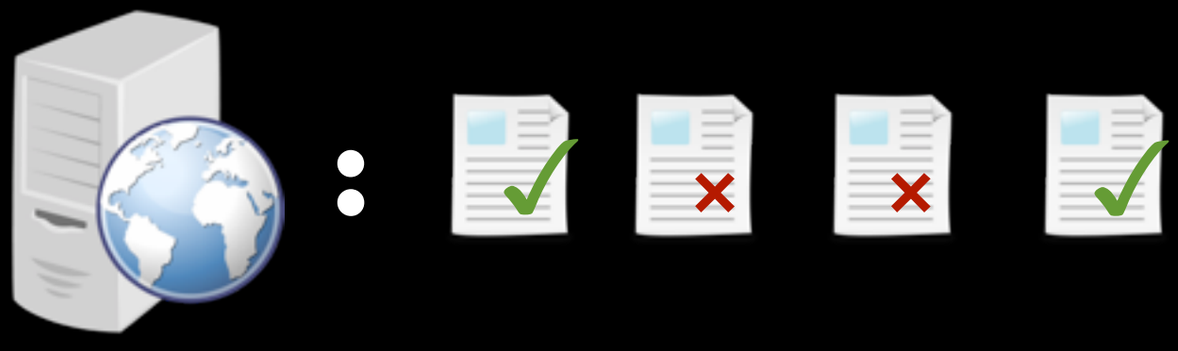
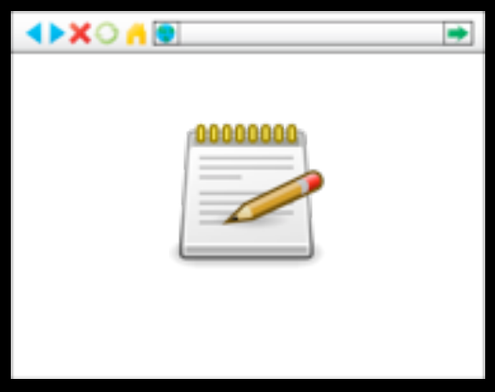


Image Source: openclipart.org, OpenIconLibrary

Log as a Service



I need all docs



... about VLDB

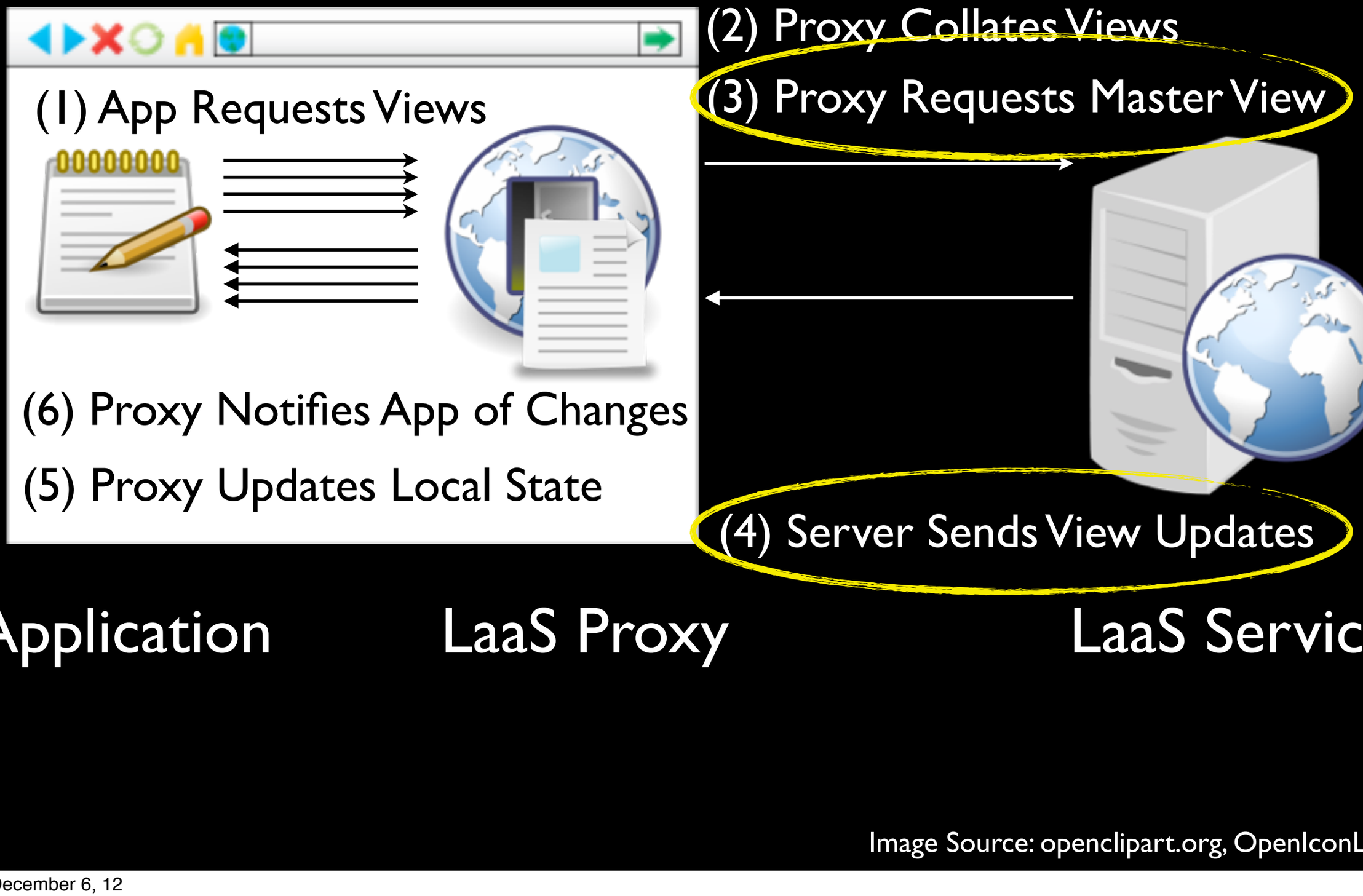
... by Bob

... modified in the last 2 days

Client state is a View!

Image Source: openclipart.org, OpenIconLibrary

Laasie



Application LaaS Proxy LaaS Service

Image Source: openclipart.org, OpenIconLibrary

Laasie

Making the Log Scale: Indexing

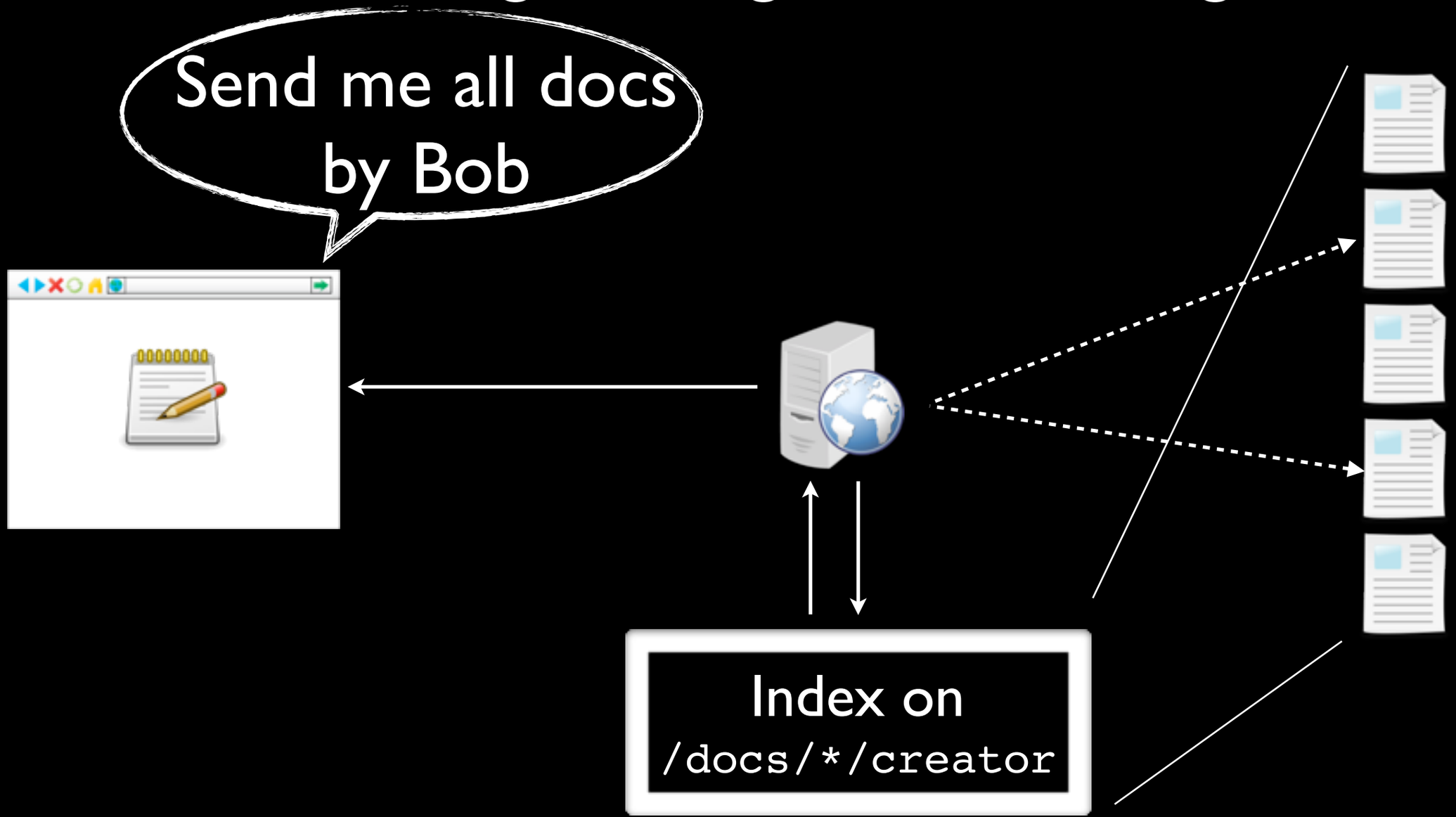


Image Source: openclipart.org, OpenIconLibrary

Laasie

Making the Log Scale: Materialization

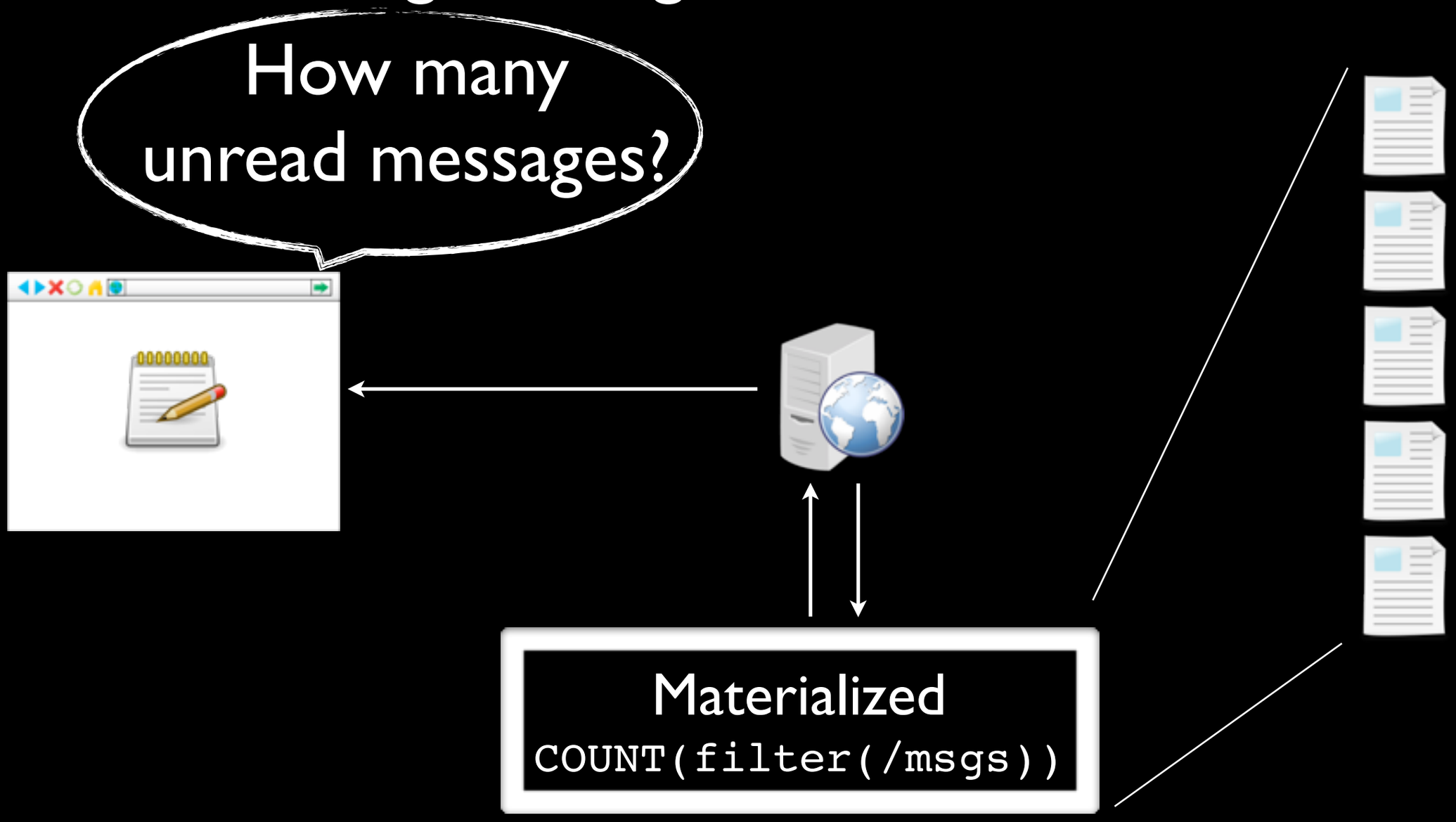


Image Source: openclipart.org, OpenIconLibrary

Laasie

Making the Log Scale: Log Rewrites

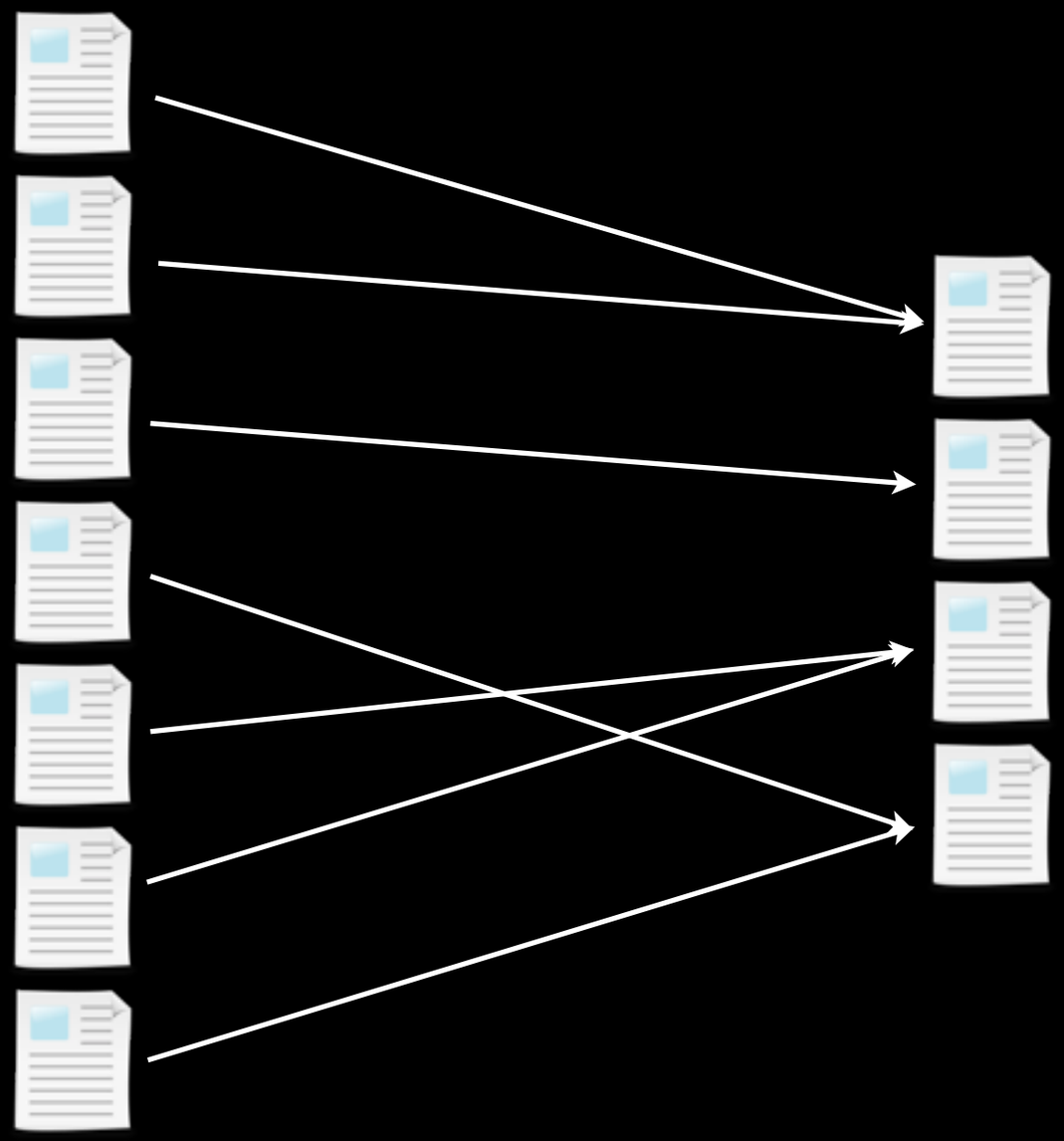


Image Source: openclipart.org

Active Research

- Cumulus -- Distributed Monitoring
- Laasie -- Collaborative Applications
- Mìmisbrunnr -- Managing Uncertain Knowledge
- GraphDBs -- Graphs as First Class Objects

http://okennedy.xthemage.net/?page=research