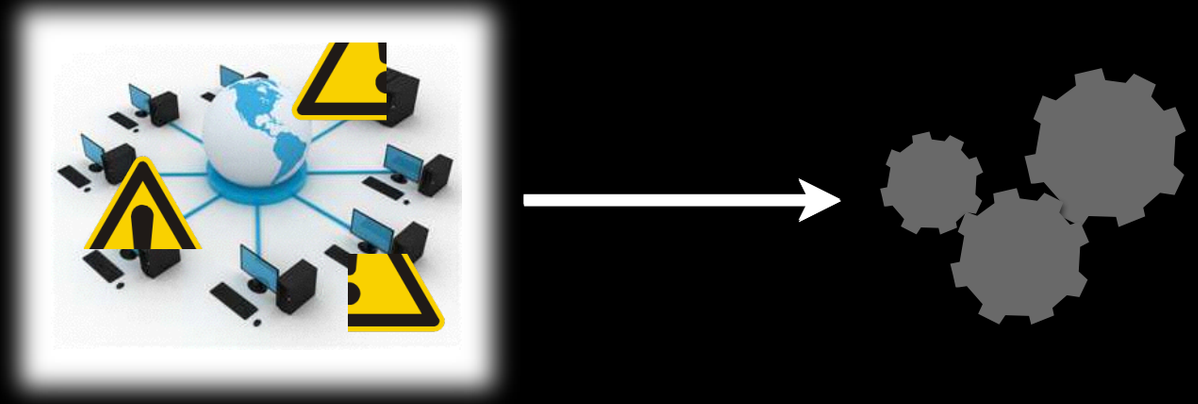
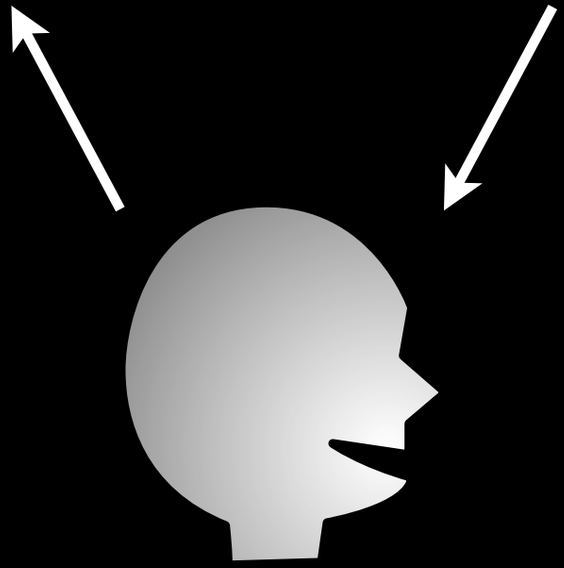
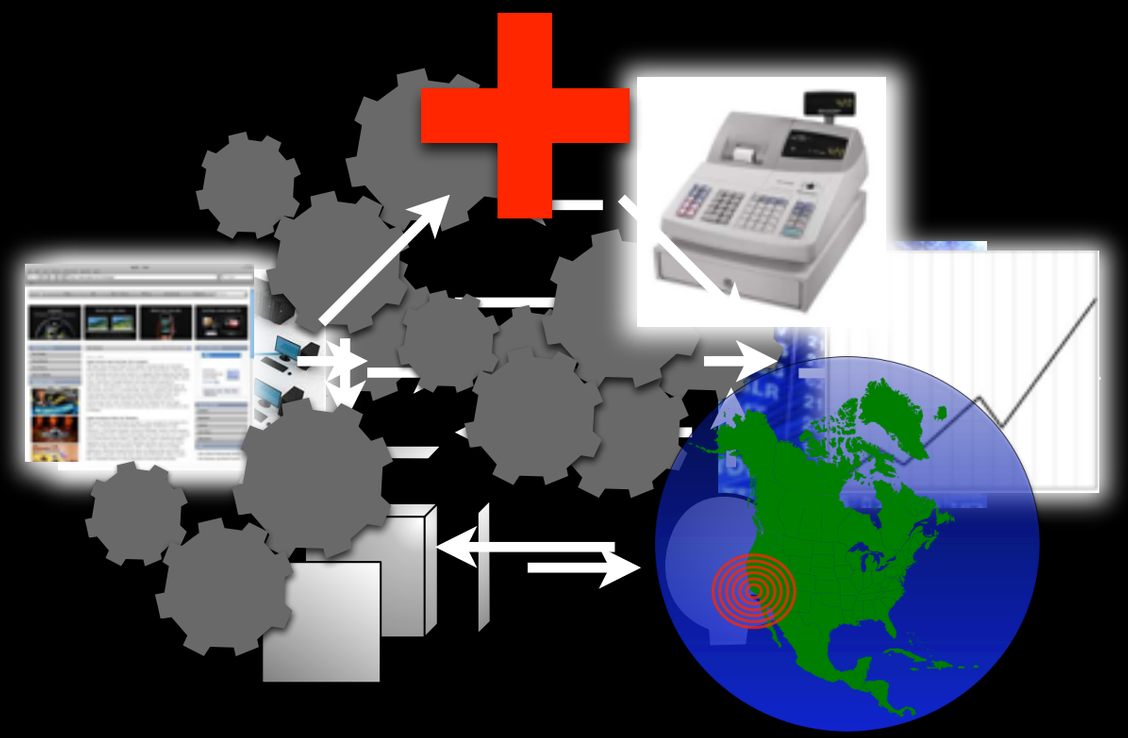
Exploiting Incrementality with DBToaster





Monitoring Programs

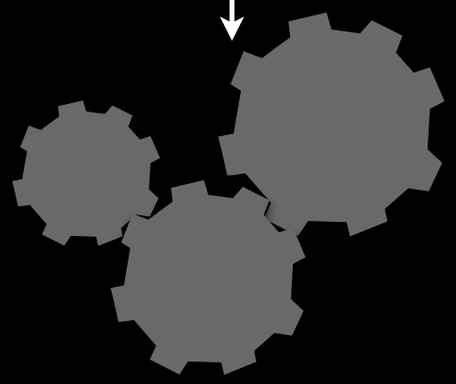


Network Monitoring

Server Status

Task Allocations

Task Properties



Servers Per Task > Task QOS?

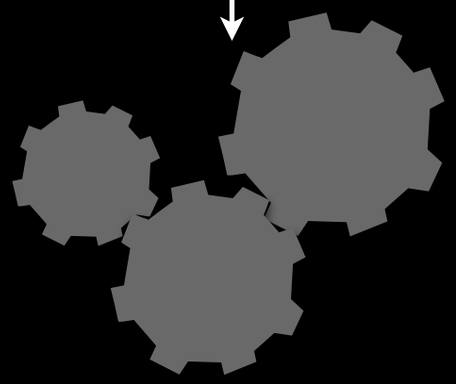
Move Task to New Servers

Computational Advertising

Site Information

Available Ads

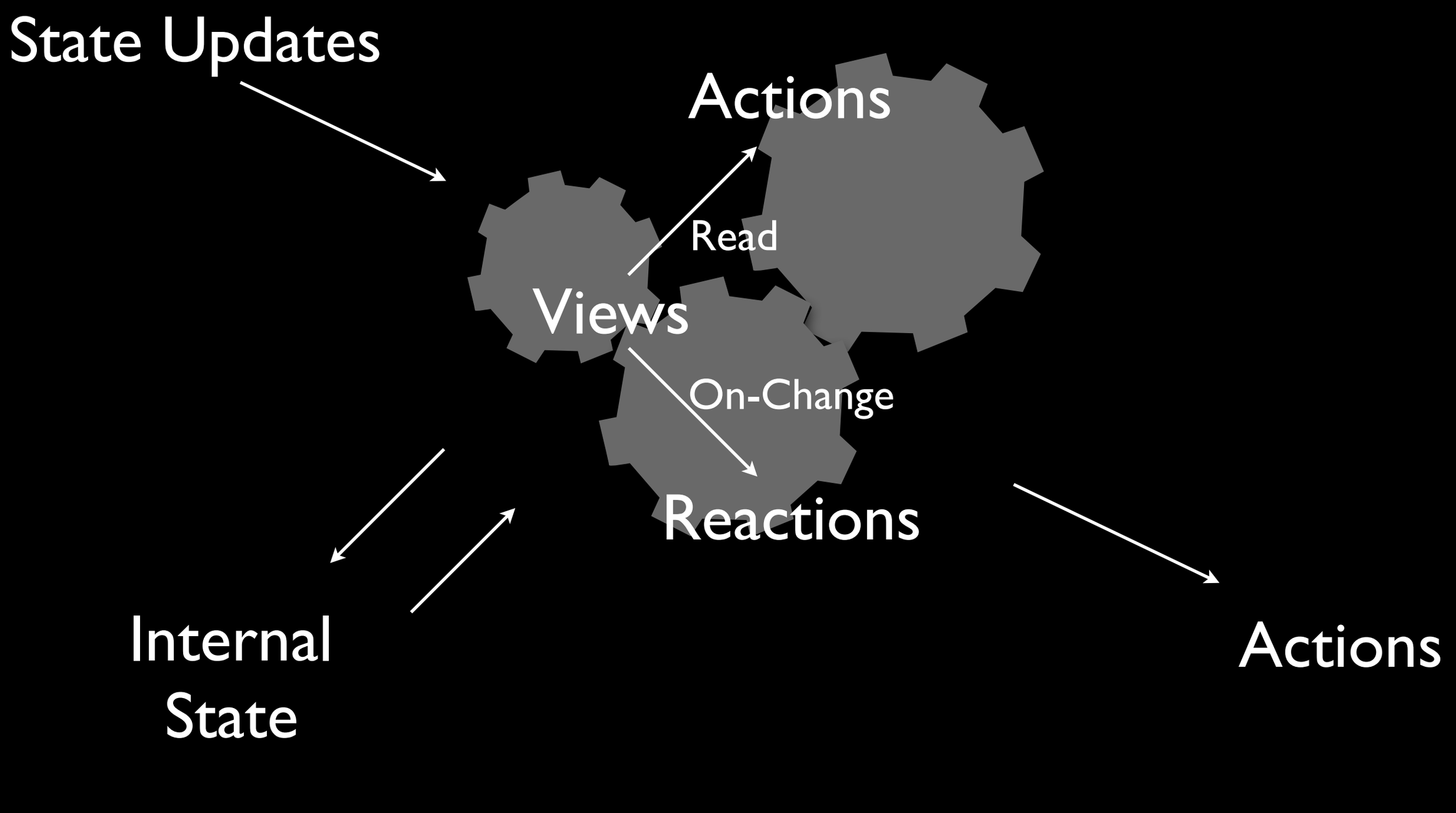
User Clicks



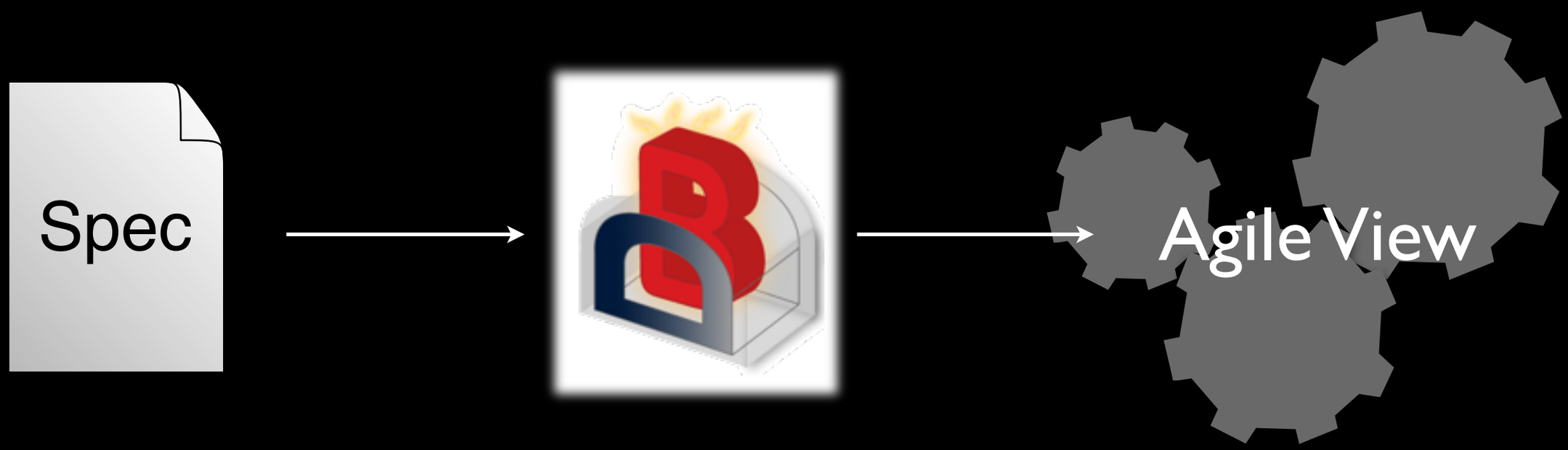
Good Ad Offers

Which Ad To Show?

Monitoring Programs

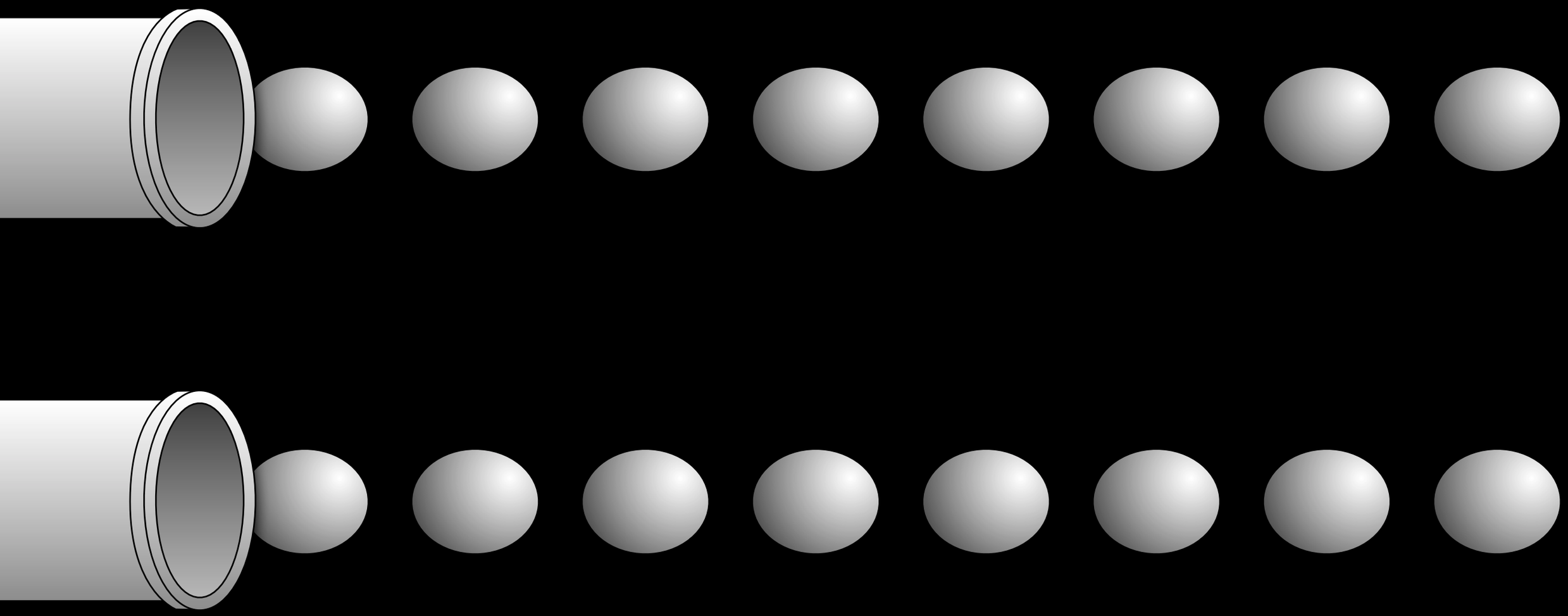


Monitoring Programs

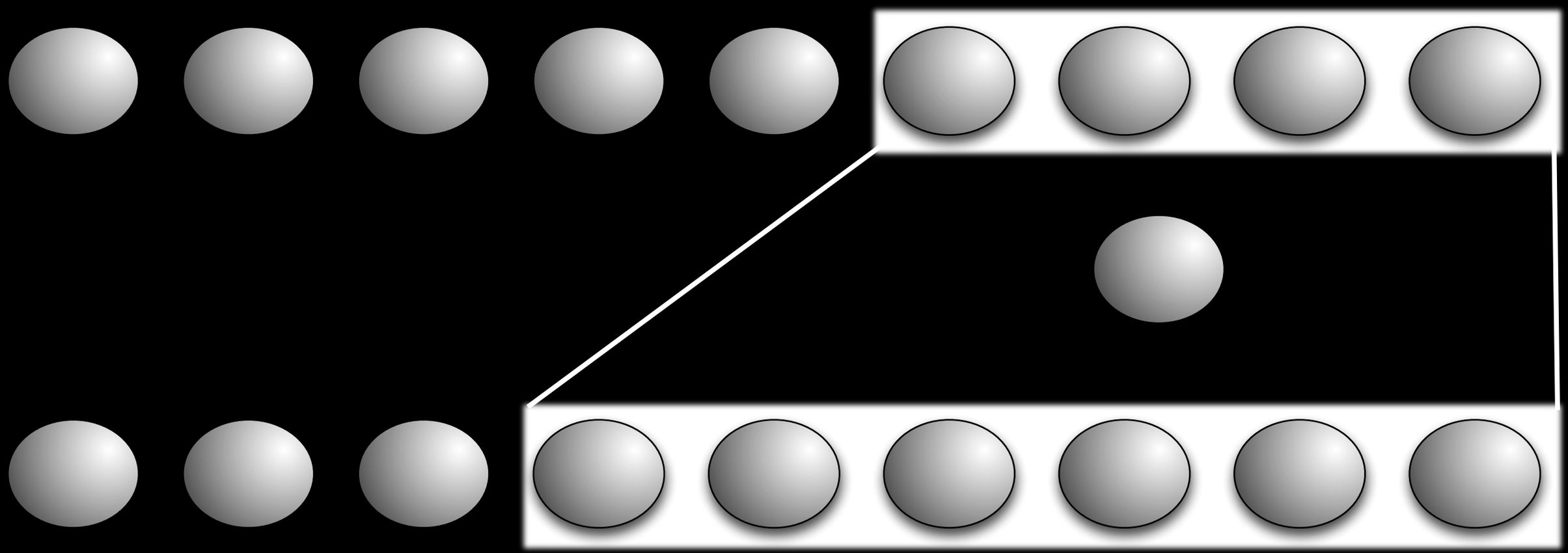


- **Existing Tools**
- DBToaster
- Cumulus

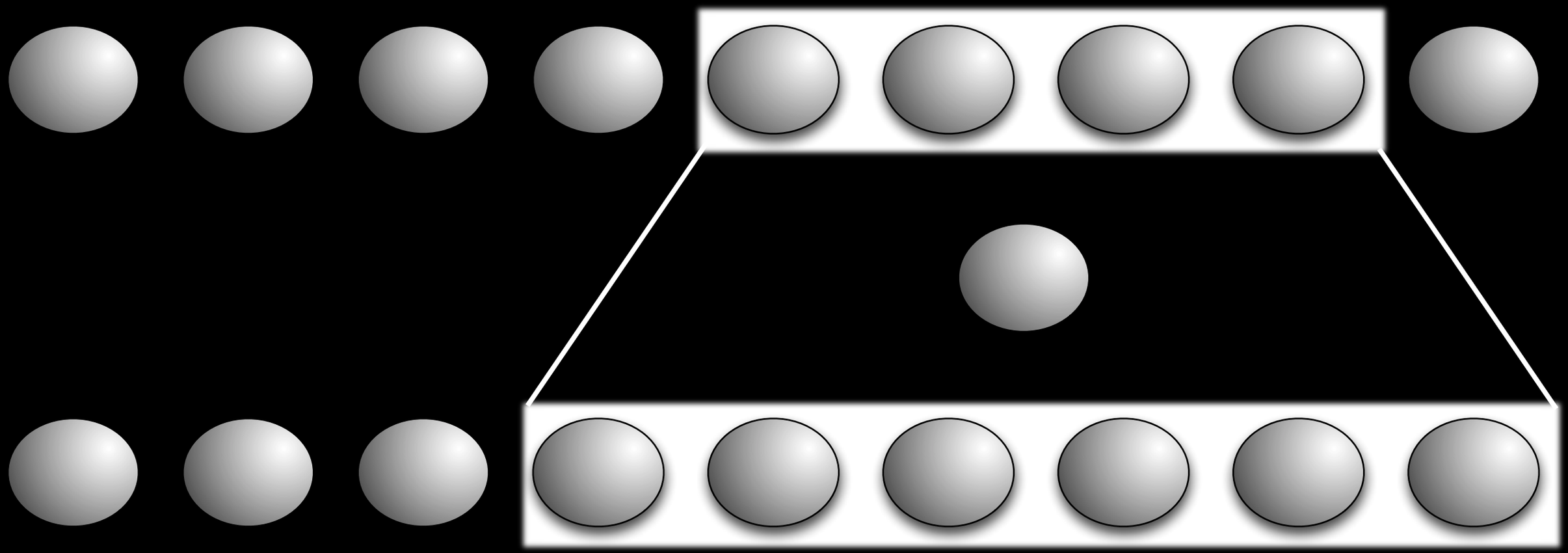
Stream Processors



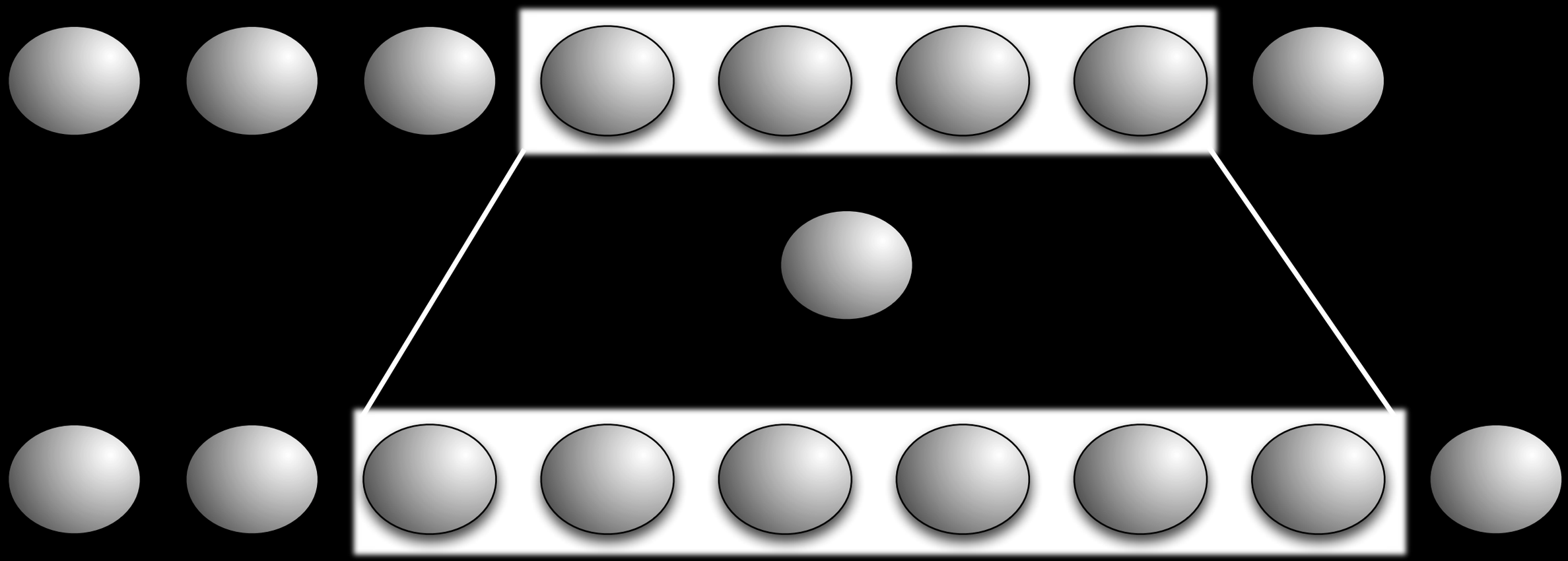
Stream Processors



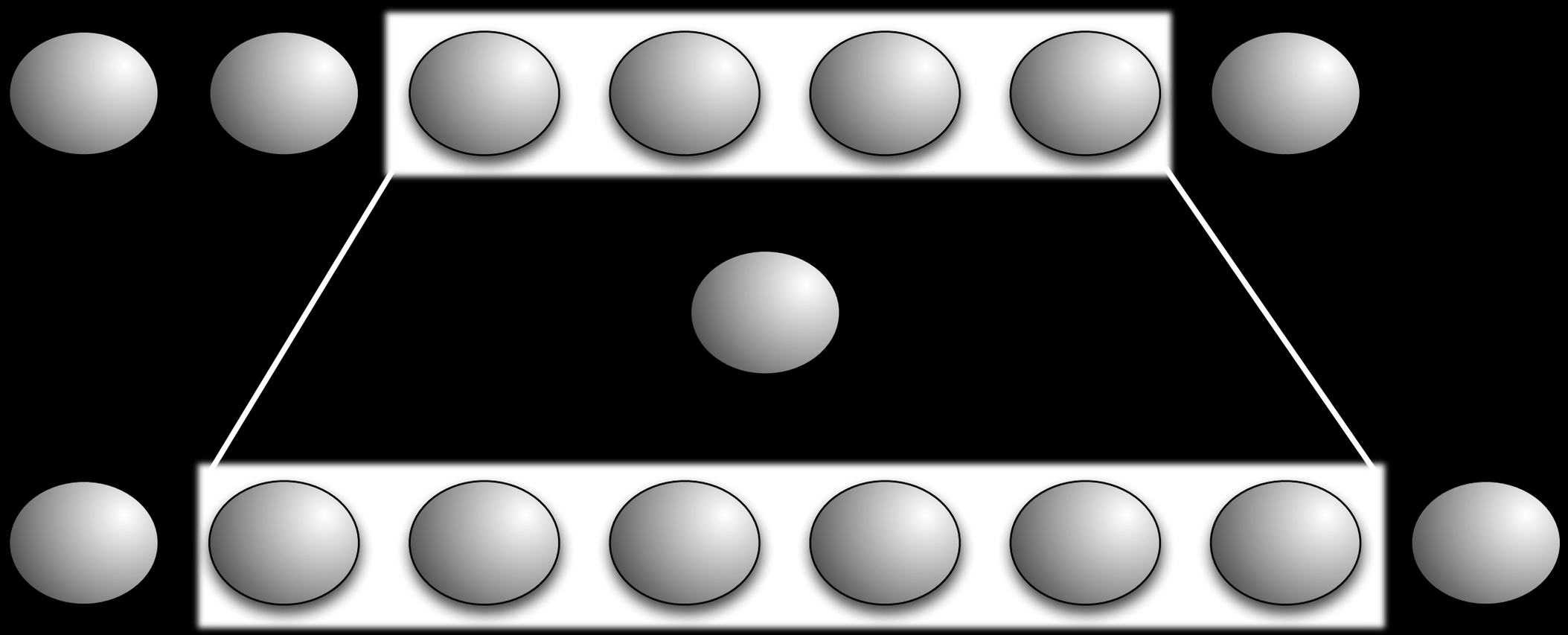
Stream Processors



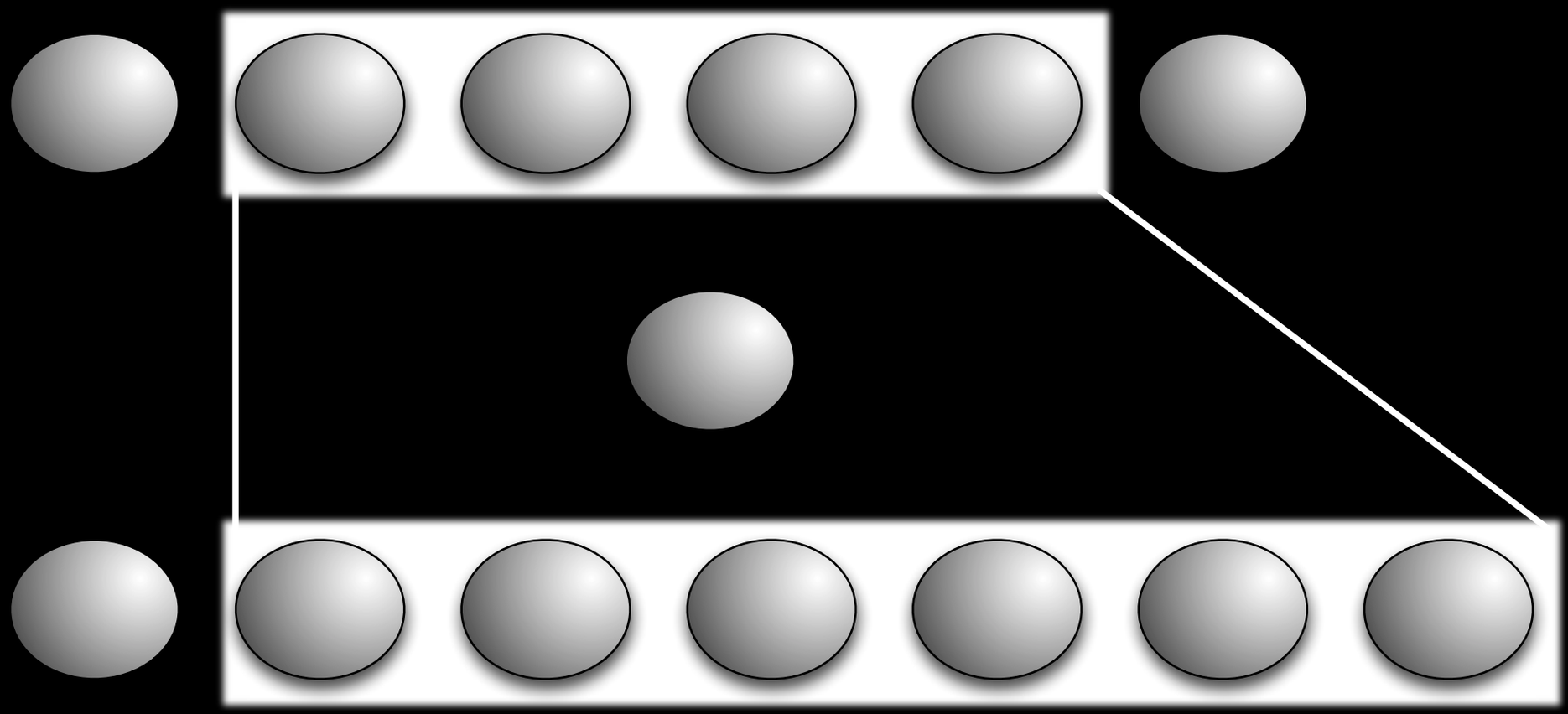
Stream Processors



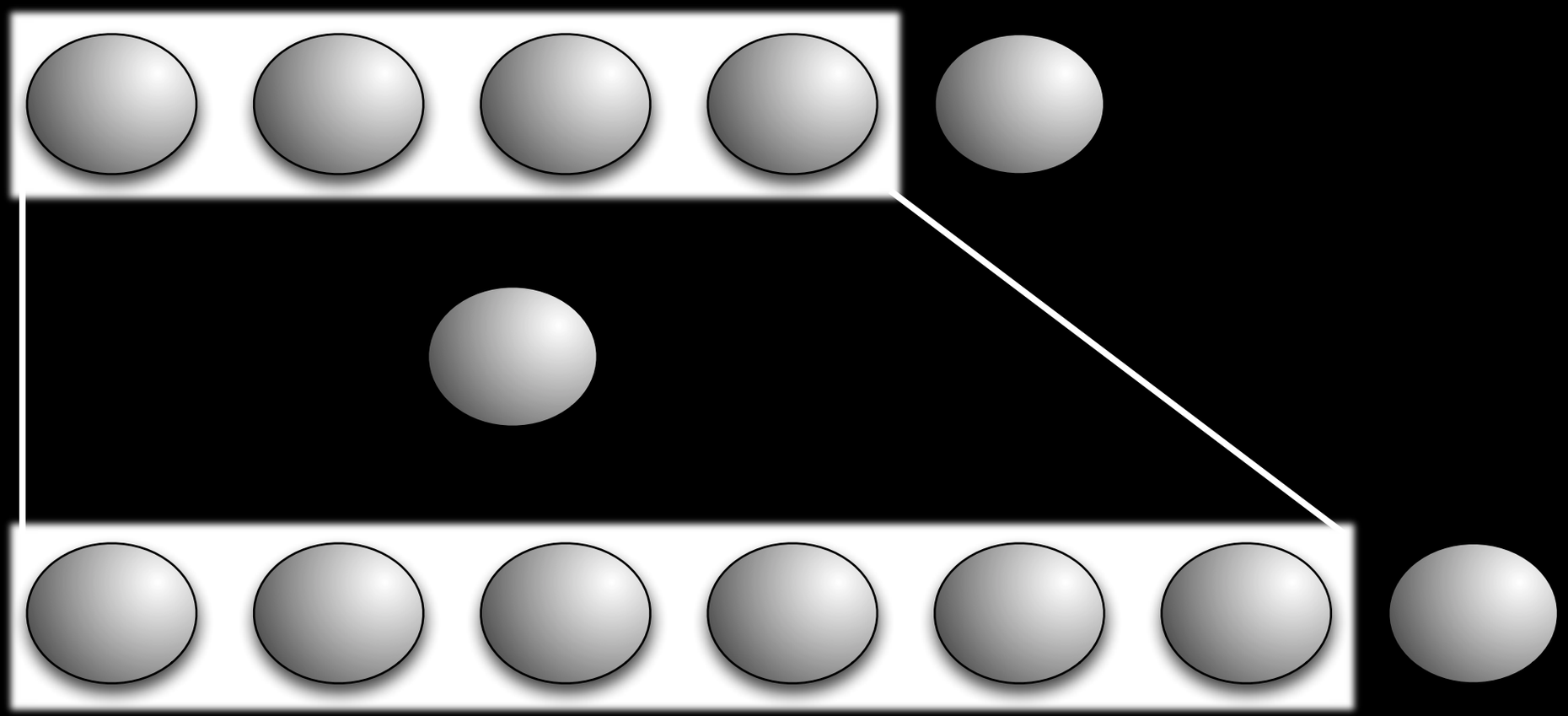
Stream Processors



Stream Processors



Stream Processors



Stream Processors

No Persistent State

Stream Processors

but also dynamic

No Persistent State

Λ

Incremental View Maintenance

QUERY

Incremental View Maintenance

ON CHANGE:

QUERY := R ⋈ S ⋈ T

Incremental View Maintenance

ON CHANGE:

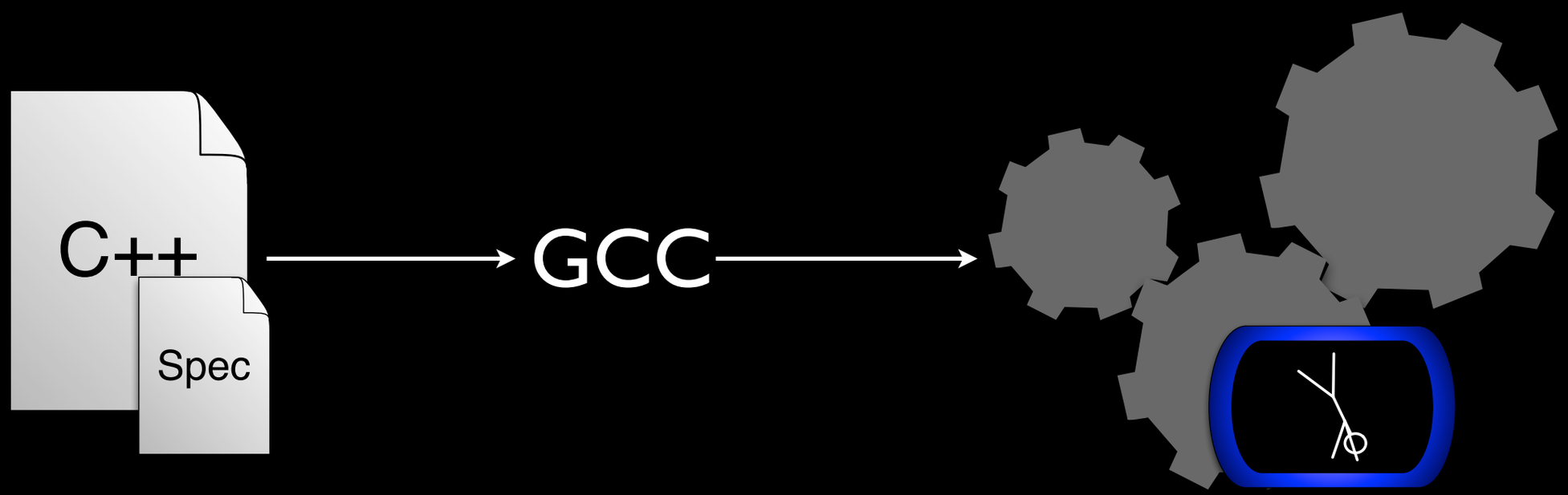
QUERY += Δ(R ⋈ S ⋈ T)

Simpler

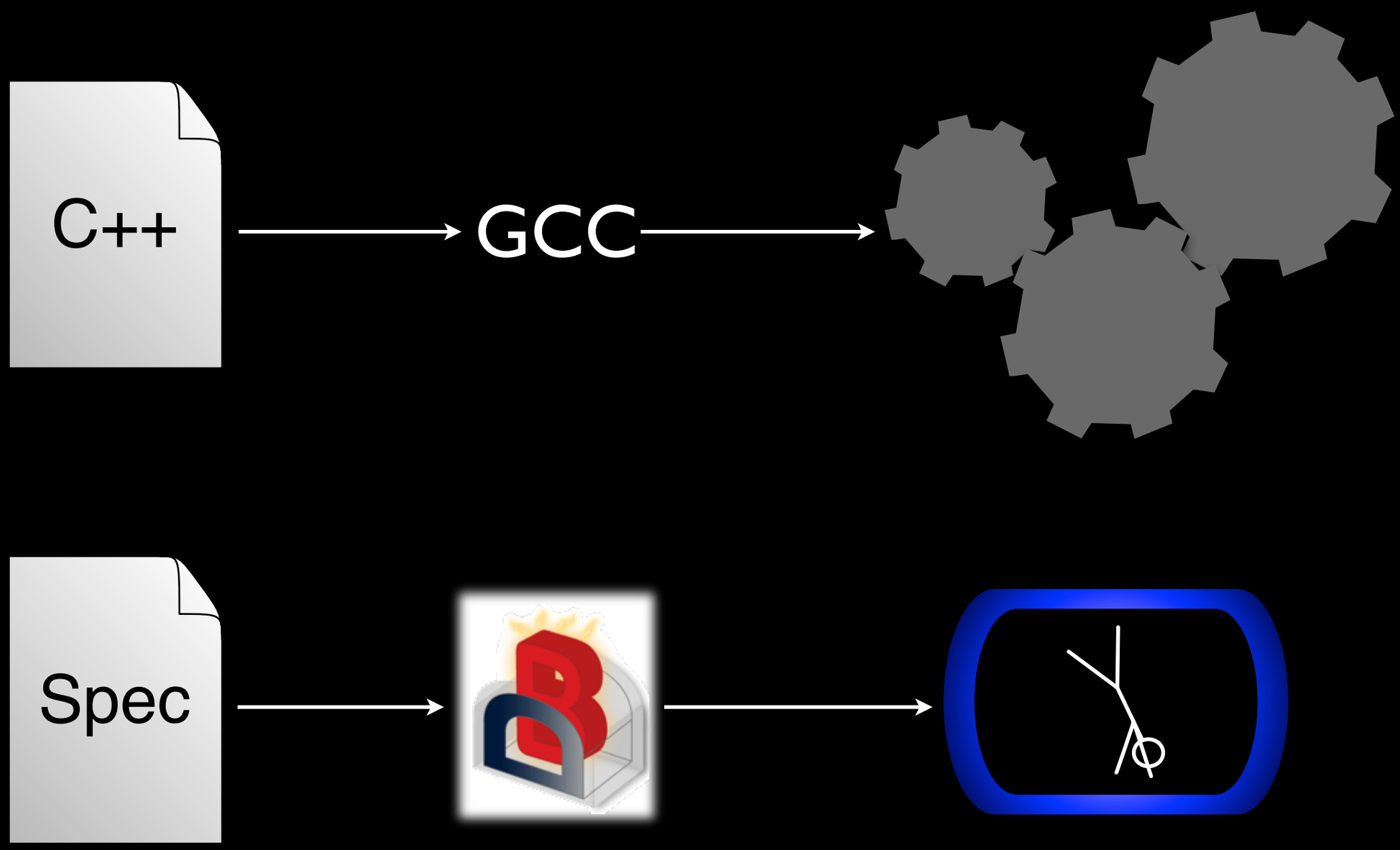
But still slow

- Existing Tools
- **DBToaster**
- Cumulus

DBToaster

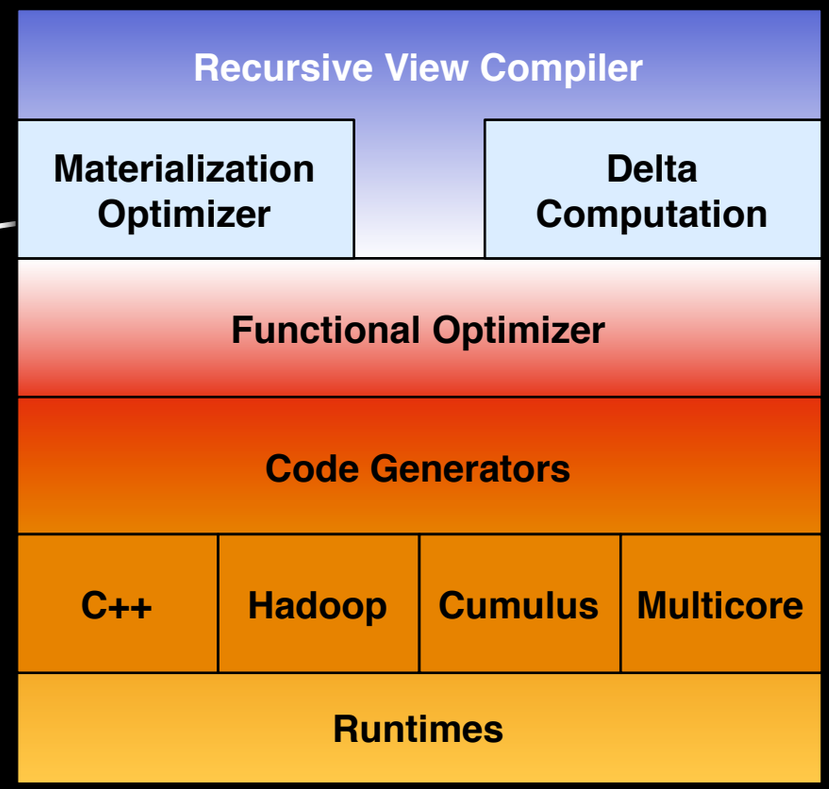


DBToaster



Toaster

- Exploit Incrementality ← Deltas
- Pick the Right Data Model
  - ... the right representation
  - ... understand the platform
- Borrow (Liberally) from Other Fields
  - ... use set-at-a-time optimizations (PL)
  - ... generate machine code (Compilers)
  - ... and others



Recursive Delta Compilation

ON ΔR :

QUERY $+= \Delta_{\Delta R}(R \bowtie S \bowtie T)$

Recursive Delta Compilation

ON ΔR :

$$\textbf{QUERY} \mathrel{+}= \Delta_{\Delta R}(R \bowtie S \bowtie T)$$

$\Delta_{\Delta R}(R \bowtie S \bowtie T)$ is Usually∧ **Simpler than** $R \bowtie S \bowtie T$

Δ is **Closed**

$\Delta_{\Delta R}(R \bowtie S \bowtie T)$ Usually∧ has **Finite Support** for ΔR

(Koch, PODS '10)

Recursive Delta Compilation

QUERY :=

SUM(R.A * T.D)

of

$R(A,B) \bowtie_B S(B,C) \bowtie_C T(C,D)$

Recursive Delta Compilation

ON +R(α,β) :

  QUERY += α * SUM(T.D)
  of
  S(β,C) ⋈$_C$ T(C,D)

Recursive Delta Compilation

ON $+R(\alpha, \beta)$:

QUERY $+= \alpha * \mathrm{SUM}(T.D)$ **of** $S(\beta, C) \bowtie_C T(C, D)$

$m_1[\beta] := \mathrm{SUM}(T.D)$ **of** $S(\beta, C) \bowtie_C T(C, D)$

Recursive Delta Compilation

```
ON +R(α,β) :
   QUERY += α * m1[β]
ON +S(β′,γ) :
   m1[β′]+= SUM(T.D)
              of
              T(γ,D)
```

Recursive Delta Compilation

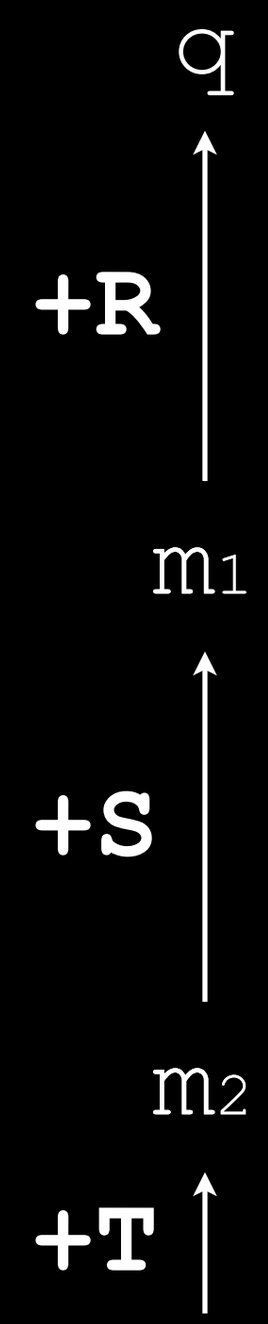
```
ON +R(α,β) :
   QUERY += α * m1[β]
ON +S(β′,γ) :
   m1[β′] += m2[γ]

m2[γ]:= SUM(T.D) of T(γ,D)
```

Recursive Delta Compilation

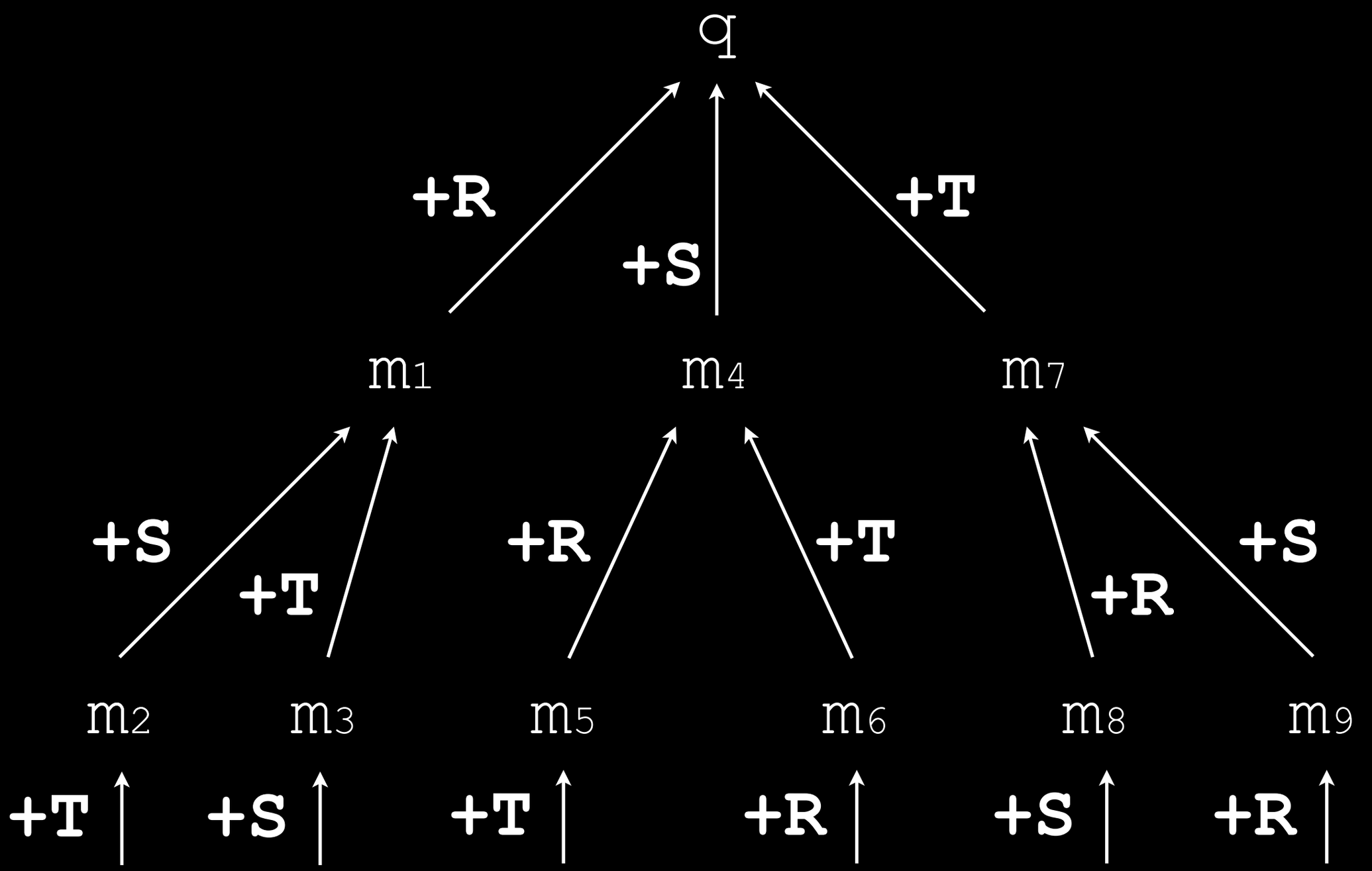
```
ON +R(α,β) :
   QUERY += α * m1[β]
ON +S(β′,γ) :
   m1[β′]+= m2[γ]
ON +T(γ′,δ) :
   m2[γ′]+= SUM(δ)
```

Recursive Delta Compilation

```
ON +R(α,β) :
   QUERY += α * m1[β]
ON +S(β′,γ) :
   m1[β′] += m2[γ]
ON +T(γ′,δ) :
   m2[γ′] += δ
```



View Hierarchy

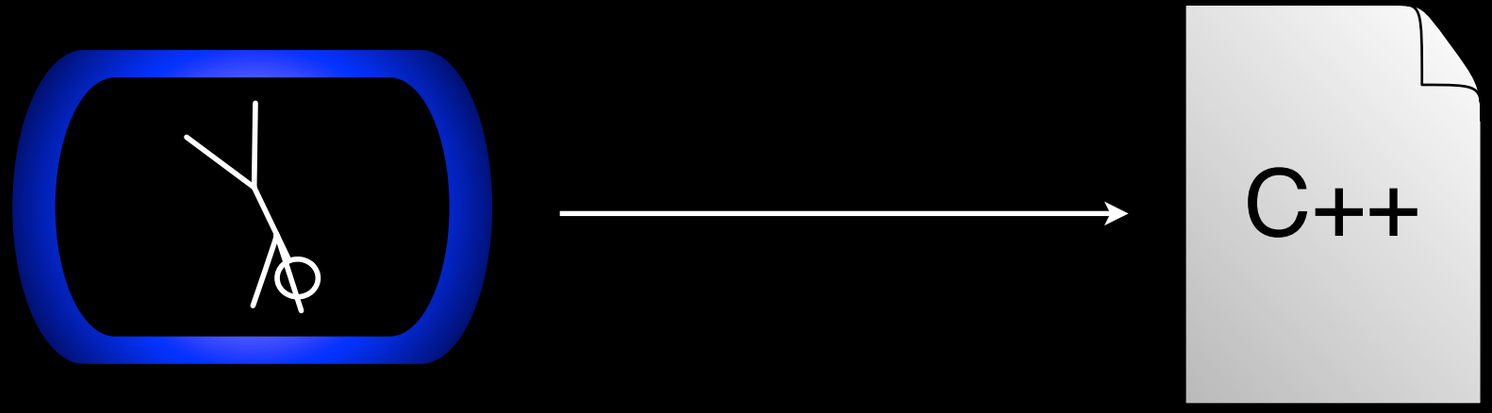


Maintenance Program

```
ON +R[ A, B ]:
 QUERY[   ]          += ( A * QUERY_dR[ B ] )
 QUERY_dT[ C ]       += FORALL C:( A * QUERY_dR_dT[ B, C ] )
 QUERY_dS[ B ]       += A

ON +S[ B, C ]:
 QUERY[   ]          += ( QUERY_dS[ B ] * QUERY_dR_dS[ C ] )
 QUERY_dT[ C ]       +=
 QUERY_dR[ B ]       +=                                  C ]
 QUERY_dR_dT[ B, C ] += 1.

ON +T[ C, D ]:
 QUERY[   ]          += ( QUERY_dT[ C ] * D )
 QUERY_dR[ B ]       += FORALL B:( D * QUERY_dR_dT[ B; C ] )
 QUERY_dR_dS[ C ]    += D
```

Maintenance Program

(DBToaster; CIDR '11)



But...

Usually
$\Delta_{\Delta R}(R \bowtie S \bowtie T)$ is **Simpler** than $R \bowtie S \bowtie T$

Nested Subqueries

Usually
$\Delta_{\Delta R}(R \bowtie S \bowtie T)$ has **Finite Support** for ΔR

Non-Equi-Joins

Nested Subqueries

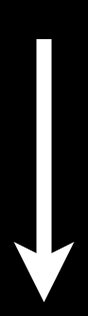
Nested Subqueries

```
QUERY :=
  COUNT() of R(A,B)
  where A = (SUM(C) of S(C))
```

Nested Subqueries

Step 1:

```
SUM(C) of [$(C)
```



Step 2:

```
COUNT() of R(A,B)
where  A = [result of step 1]
```

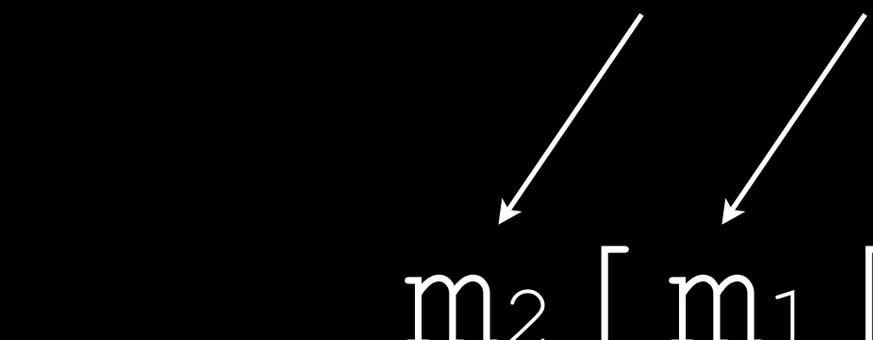
Nested Subqueries

Step 2:

m_2[A]:= COUNT() **of** R(A,B)
where A = [result of step 1]

Partial Materialization

Materialize the query in parts



Perform computations at maintenance-time

Non-Equality Predicates

```
QUERY :=
  COUNT() of R(A) ⨉ S(B,C)
  where A < B
```

Non-Equality Predicates

```
ON +R(α):
  QUERY += SUM(m1[]*S(B,C)) where α < B
```

Partial Materialization

```
m1[B] := COUNT() of S(B,C) group by B
```

Materialization Optimizer

- Nested Subqueries
- Non-equality predicates
- Memory Constraints
- High Maintenance Cost
- Specialized Datastructures

VWAP

```
SELECT sum(b1.price
           * b1.volume)
  FROM bids b1
 WHERE 0.25 *
   (SELECT sum(b3.volume)
      FROM bids b3
     WHERE b3.price >
           b1.price) >
   (SELECT sum(b2.volume)
      FROM bids b2
     WHERE b2.price >
           b1.price);
```

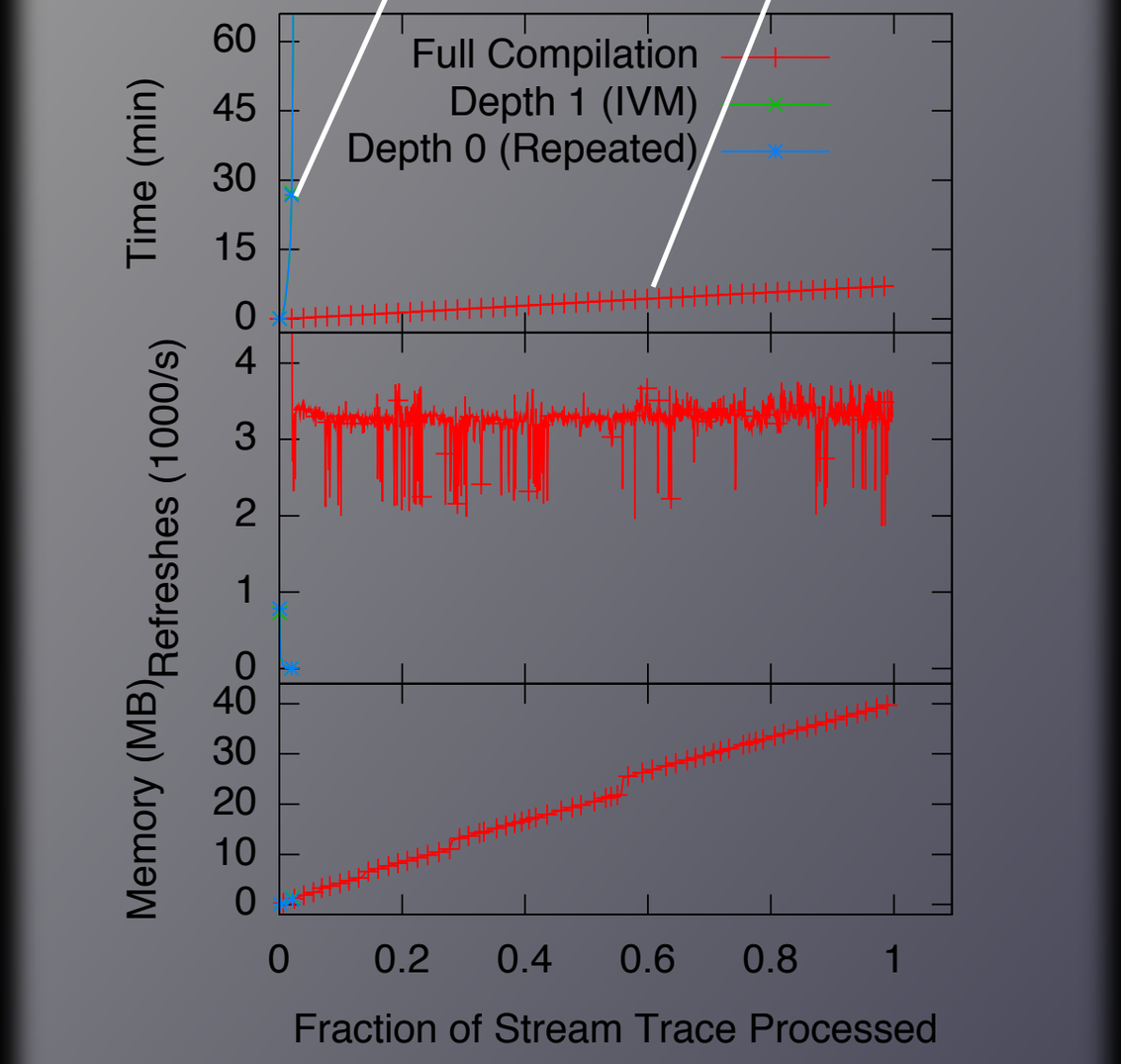
Cumulative Time

Rate of View Refreshing

Memory Usage

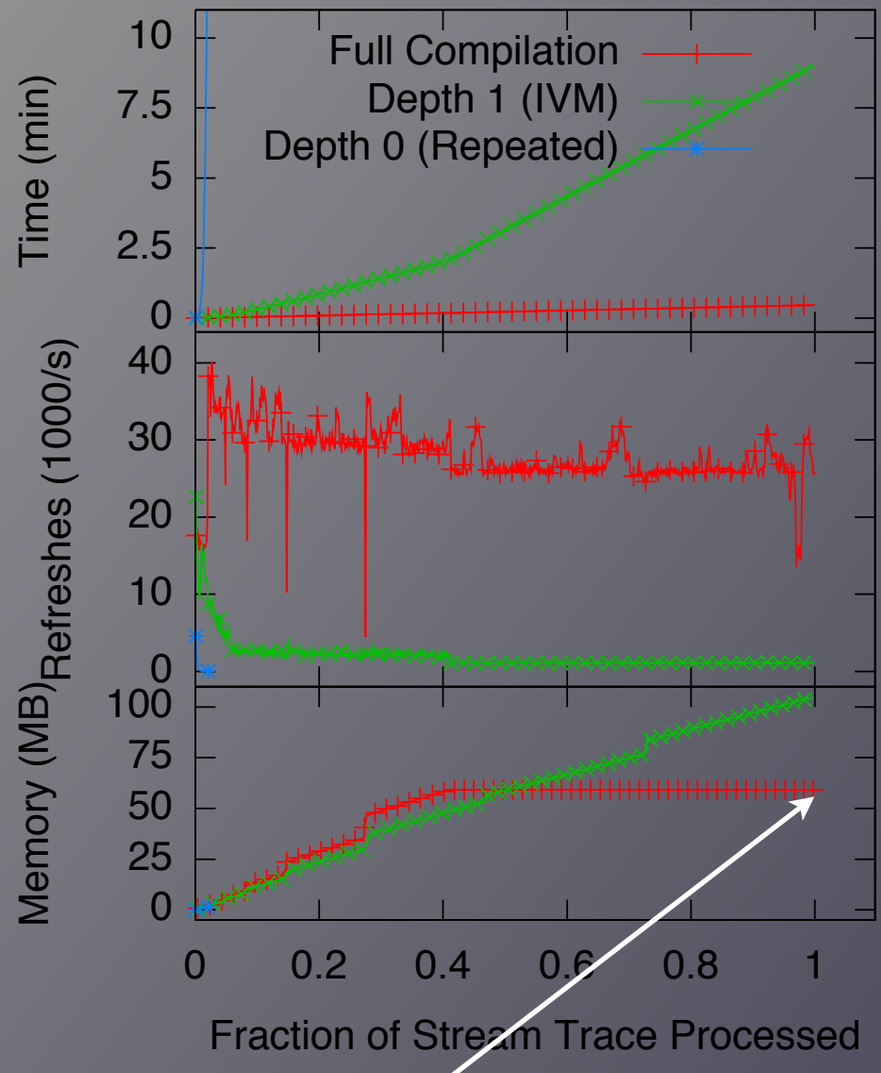
IVM & Naive

DBToaster



TPC-H Q3

```
SELECT ORDERS.orderkey,
       ORDERS.orderdate,
       ORDERS.shippriority,
       SUM(extendedprice
           * (1 - discount))
  FROM CUSTOMER, ORDERS, LINEITEM
 WHERE CUSTOMER.mktsegment
                = 'BUILDING'
   AND ORDERS.custkey
                = CUSTOMER.custkey
   AND LINEITEM.orderkey
                = ORDERS.orderkey
   AND ORDERS.orderdate
                < DATE('1995-03-15')
   AND LINEITEM.SHIPDATE
                > DATE('1995-03-15')
GROUP BY ORDERS.orderkey,
         ORDERS.orderdate,
         ORDERS.shippriority;
```



Half the Memory Usage

- Existing Tools
- DBToaster
- Cumulus

Maintenance Programs

Data-Parallel Computations

```
ON +R[ A, B ]:
 QUERY[   ]          += ( A * QUERY_dR[ B ] )
 QUERY_dT[ C ]       += FORALL C:( A * QUERY_dR_dT[ B, C ] )
 QUERY_dS[ B ]       += A

ON +S[ B, C ]:
 QUERY[   ]          += ( QUERY_dS[ B ] * QUERY_dR_dS[ C ] )
 QUERY_dT[ C ]       += QUERY_dS[ B ]
 QUERY_dR[ B ]       += QUERY_dR_dS[ C ]
 QUERY_dR_dT[ B, C ] += 1.

ON +T[ C, D ]:
 QUERY[   ]          += ( QUERY_dT[ C ] * D )
 QUERY_dR[ B ]       += FORALL B:( D * QUERY_dR_dT[ B; C ] )
 QUERY_dR_dS[ C ]    += D
```

Key/Value Style Datastructures

Execution Model

ON Event(param1, param2, …)

Statement 1

Statement 2

Statement 3

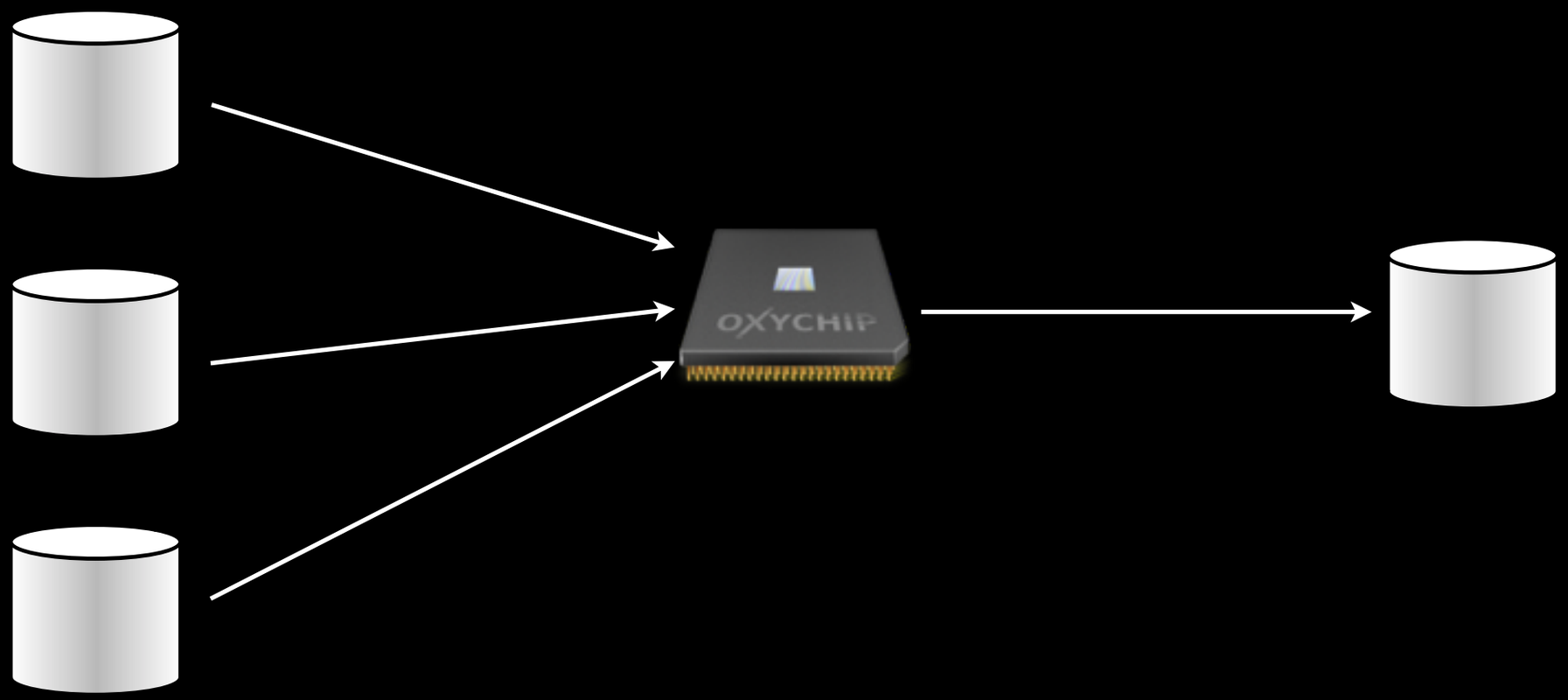
…

Statement Execution

Read

Compute

Write



[Old Version]

[New Version]

S

0

1

2

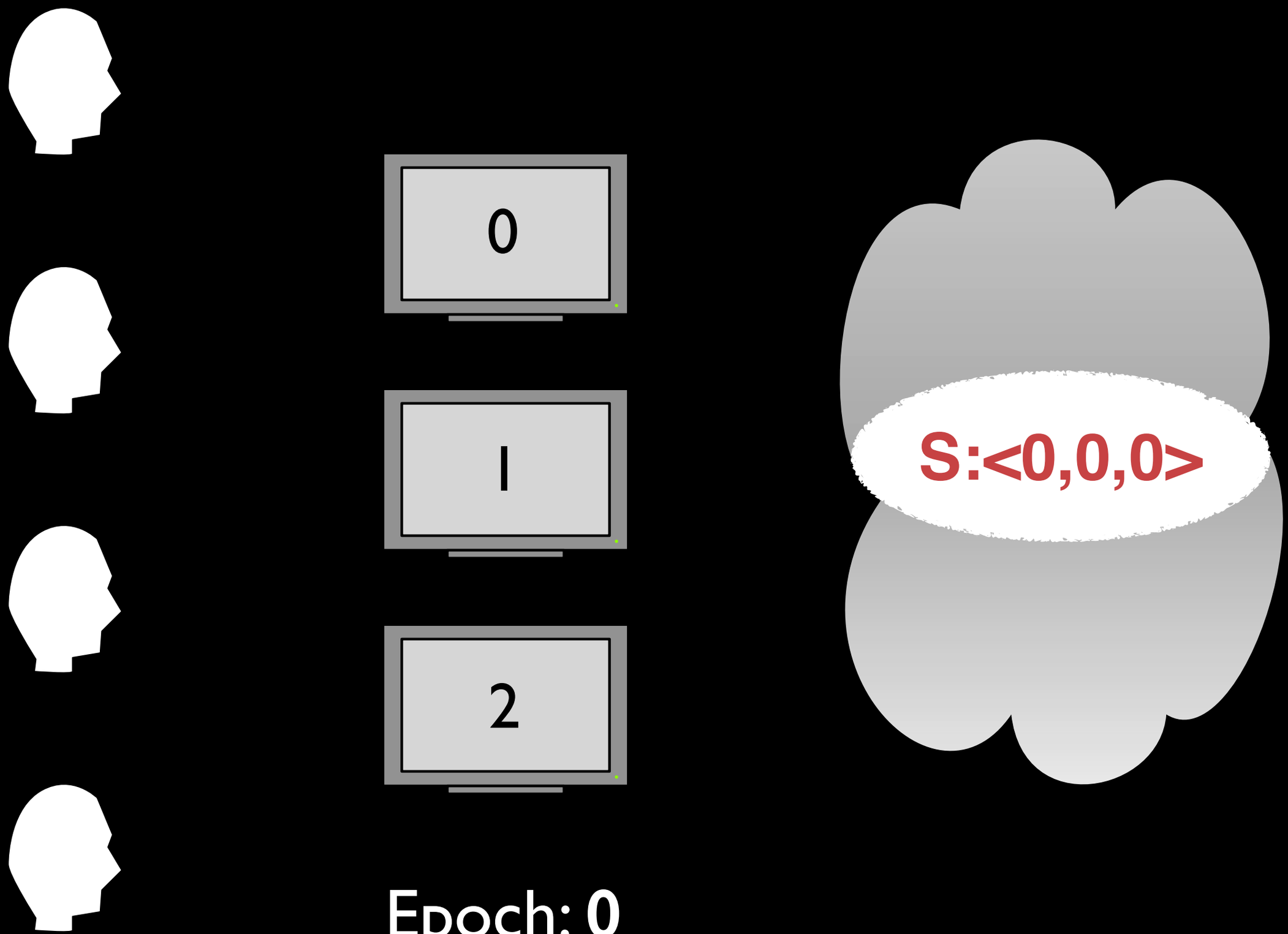
Epoch: **0**

S:< S ,0>

1

2

Epoch: **0**



Epoch: **0**

R

T

0

1

2

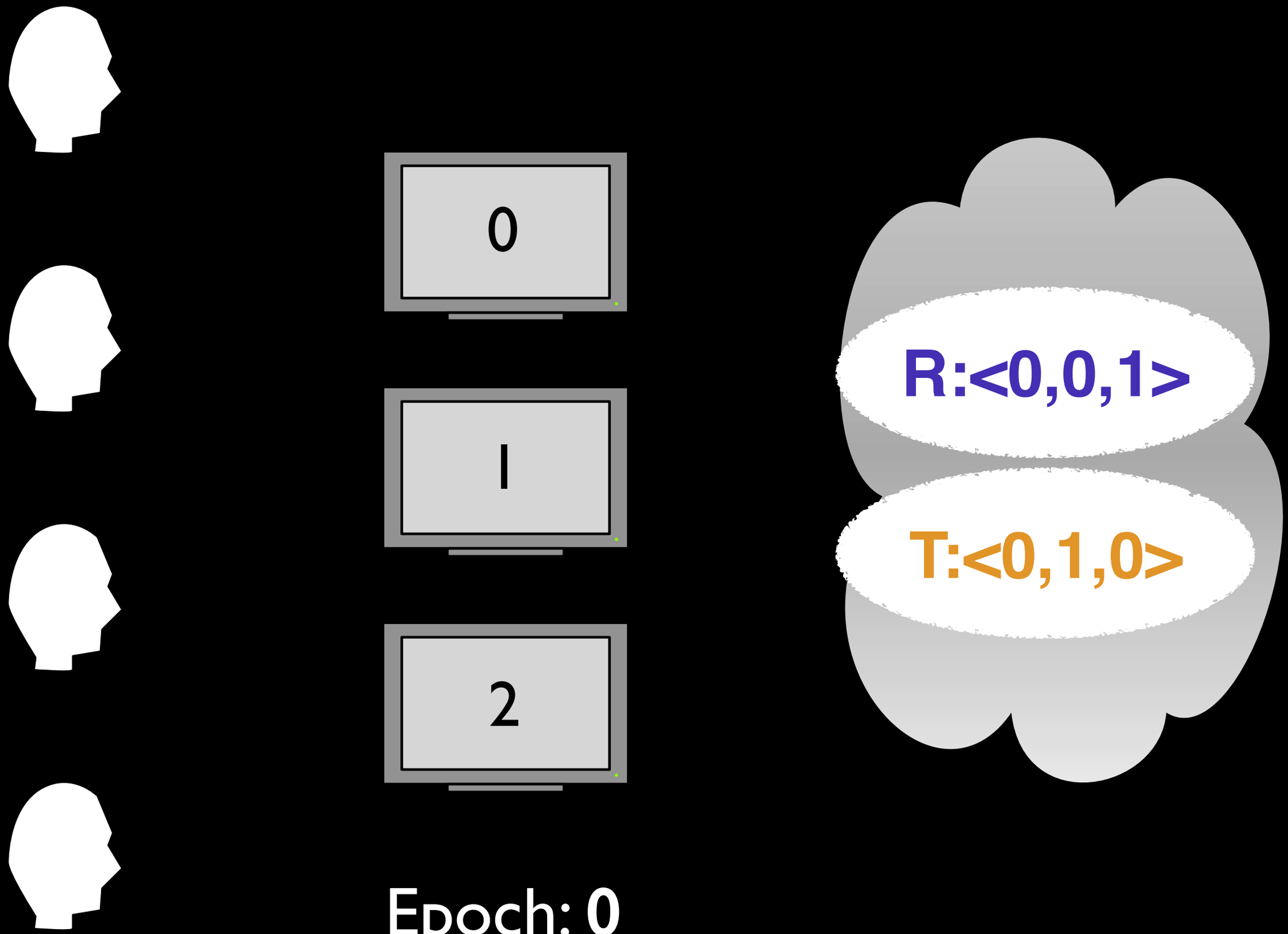
Epoch: **0**

R:< R ,1>

T:< T ,0>

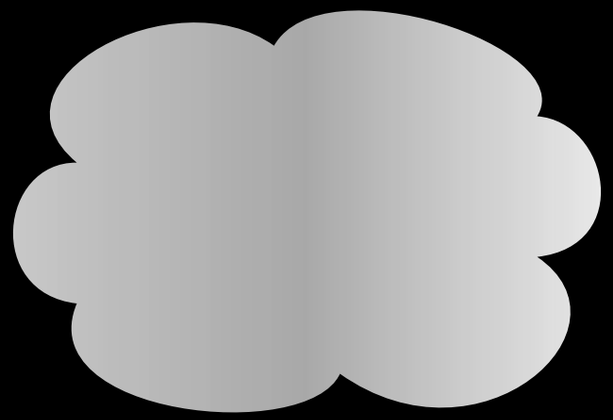
2

Epoch: 0


0
1
2
R:<0,0,1>
T:<0,1,0>
Epoch: 0

$M_2[1,3]$ += 2

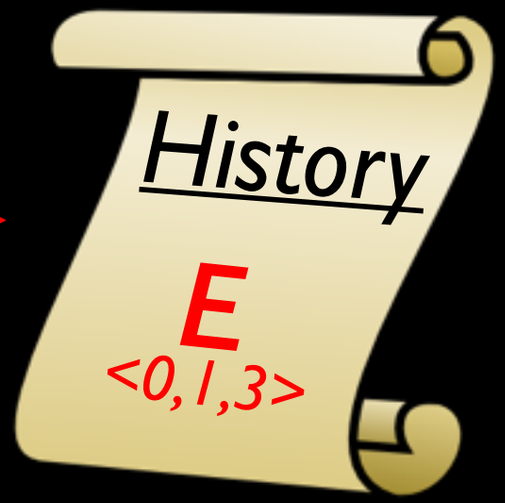
<0,2,4>



R

<0,1,3>

R
<0,1,3>



$M_2[\ldots]$ += $M_3[\ldots] * M_4[\ldots]$

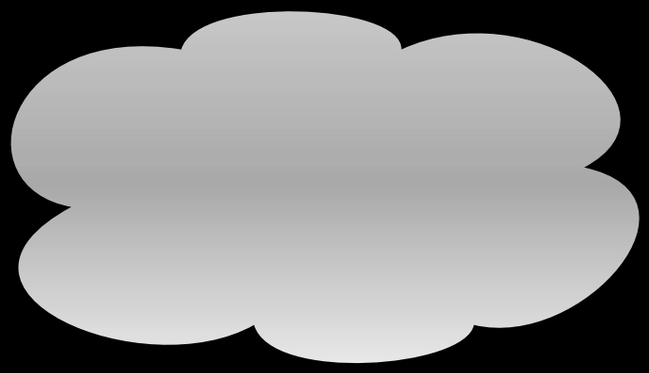
$\Sigma\delta$

$M_3[\ldots]$

$M_4[\ldots]$

$M_3[\ldots] * M_4[\ldots]$

$M_2[\ldots]$ += ...
<0,1,3>

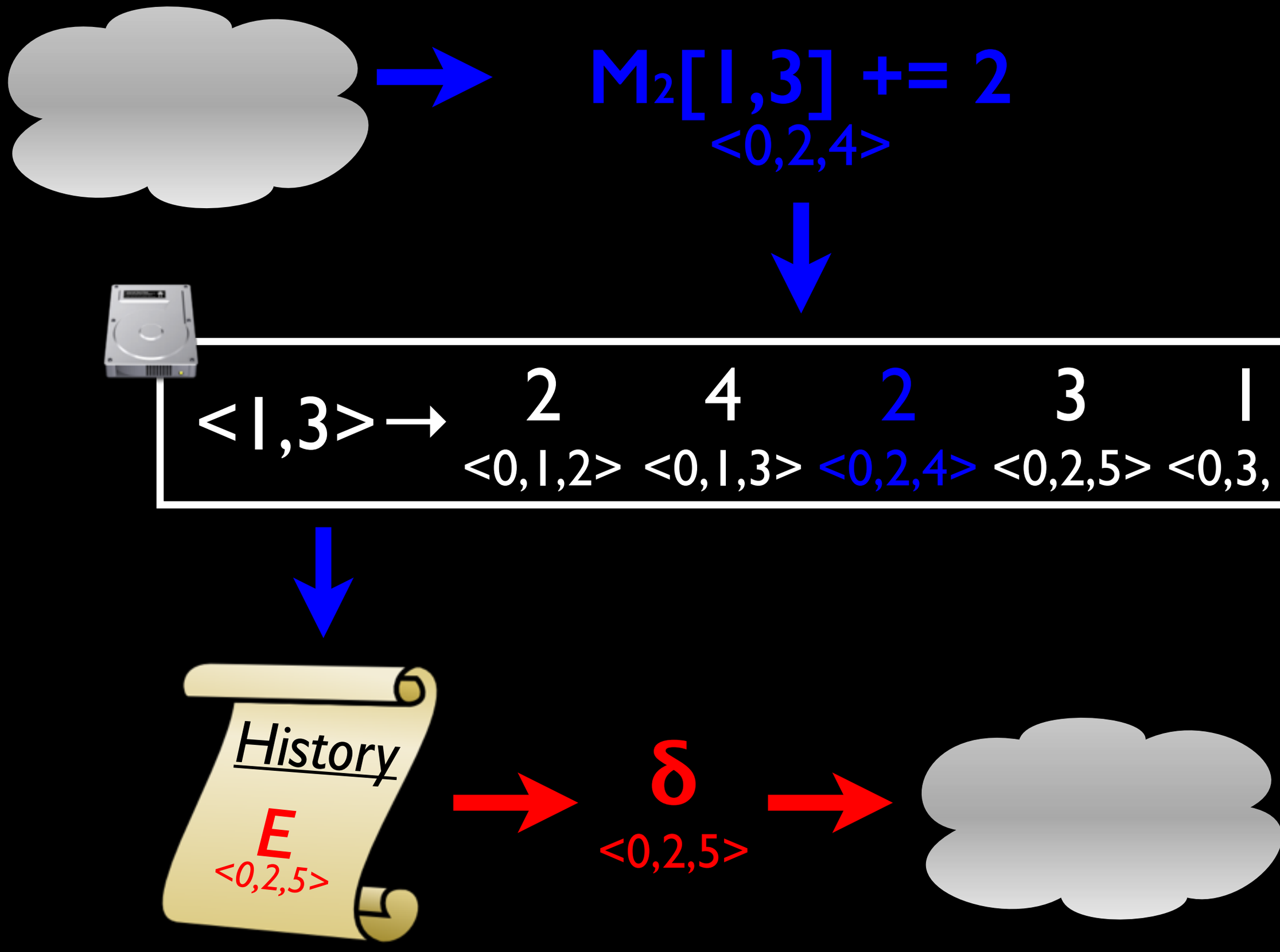


$M_2[1,3]$ += 2

<0,2,4>

<1,3> → 2 4 3 1

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


M2[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>

Open Challenges

- Data Placement
  - Migration
- Batch Processing
- Live Program Management