Relational Cloud: a database service for the cloud

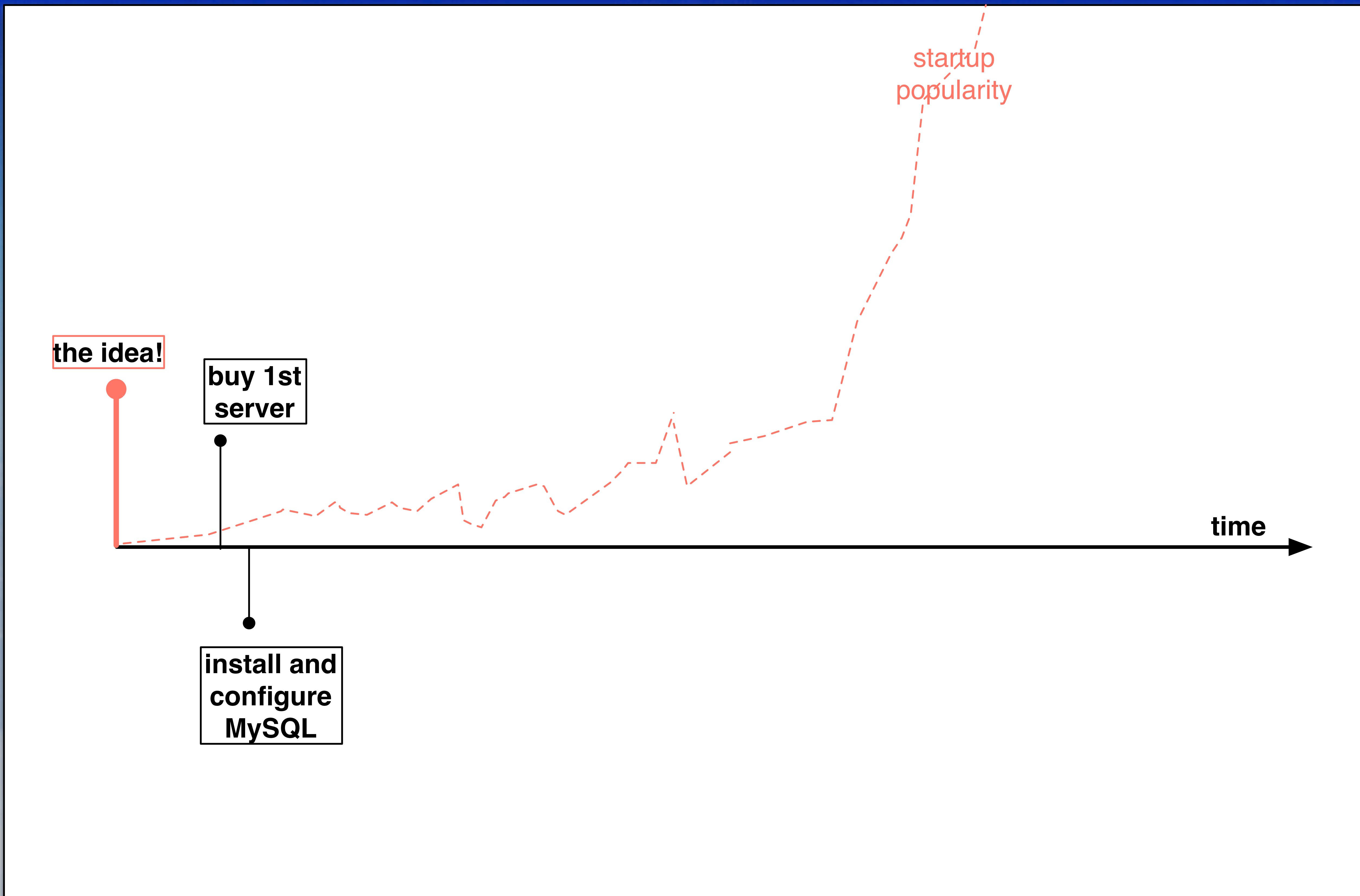
Carlo C[illegible]
Evan Jones, Raluca Ada Popa, Eugene Wu, Nirmesh Malviya
Sam Madden, Hari Balakrishnan, Nickolai Zeldovich

Relational Cloud: a database service for the cloud

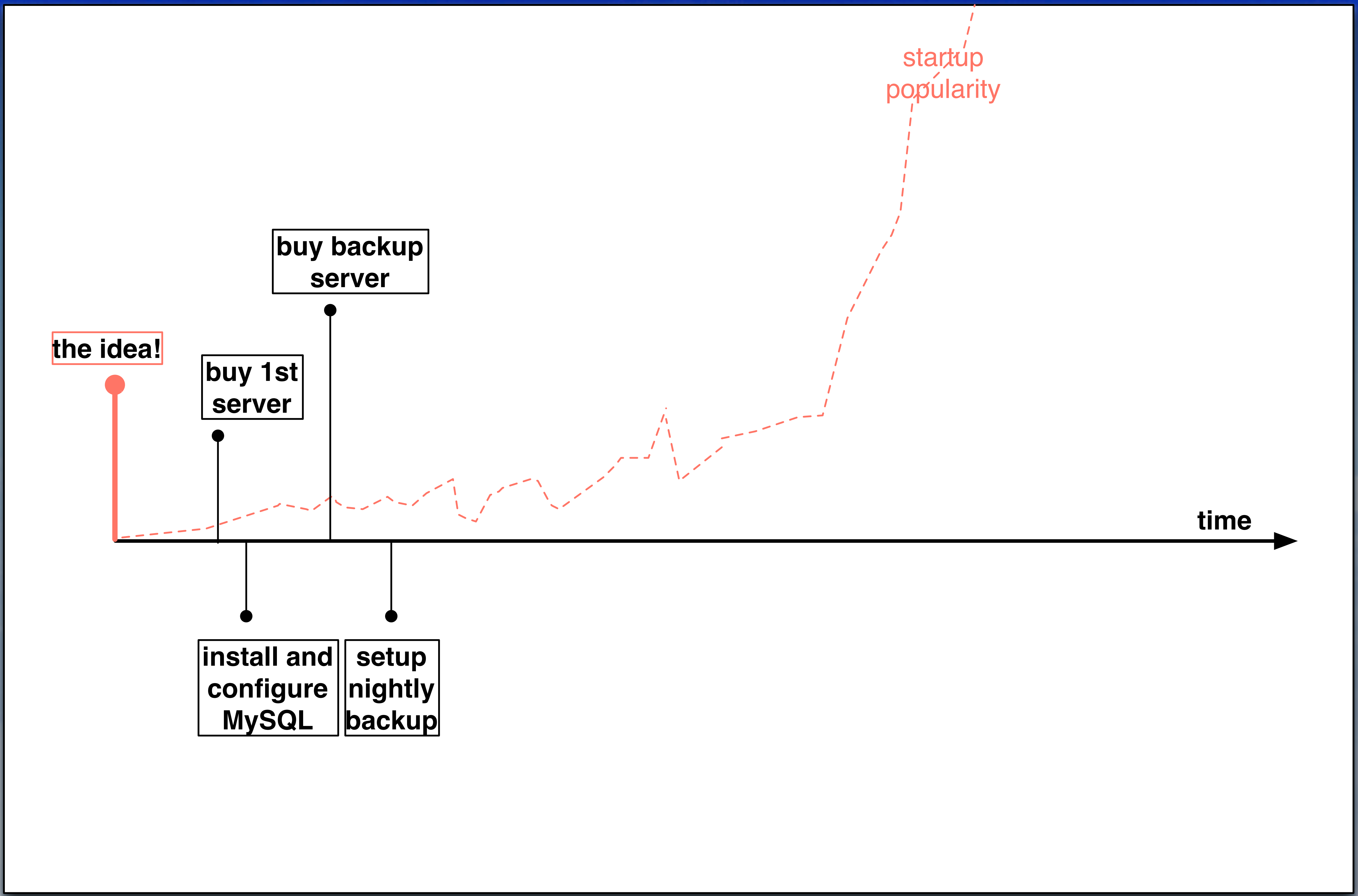
Carlo Curino,
Evan Jones, Raluca Ada Popa, Eugene Wu, Nirmesh Malviya
Sam Madden, Hari Balakrishnan, Nickolai Zeldovich

CSAIL MIT
Cambridge, MA

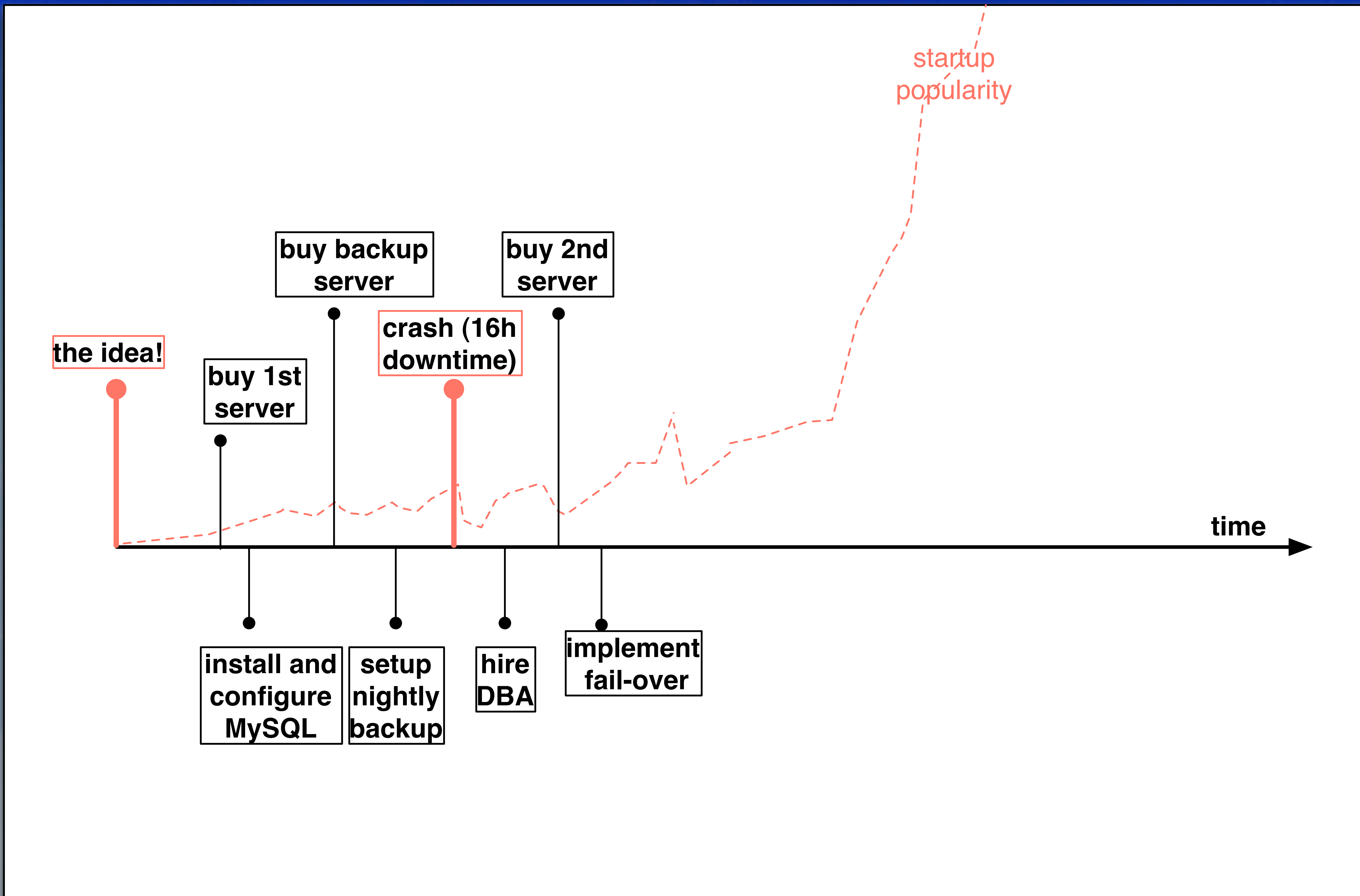
Startup success: a story of DB drama



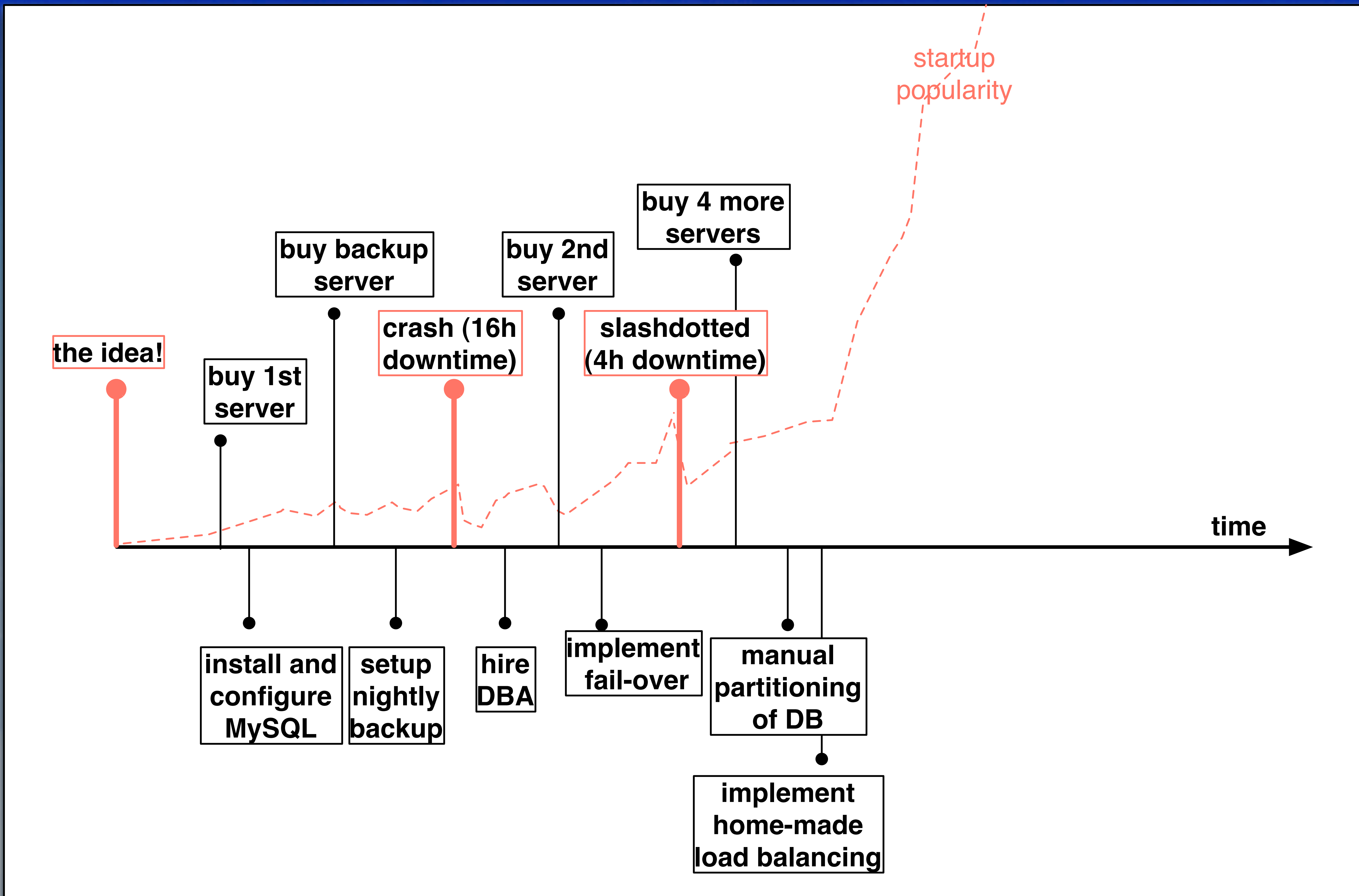
Startup success: a story of DB drama



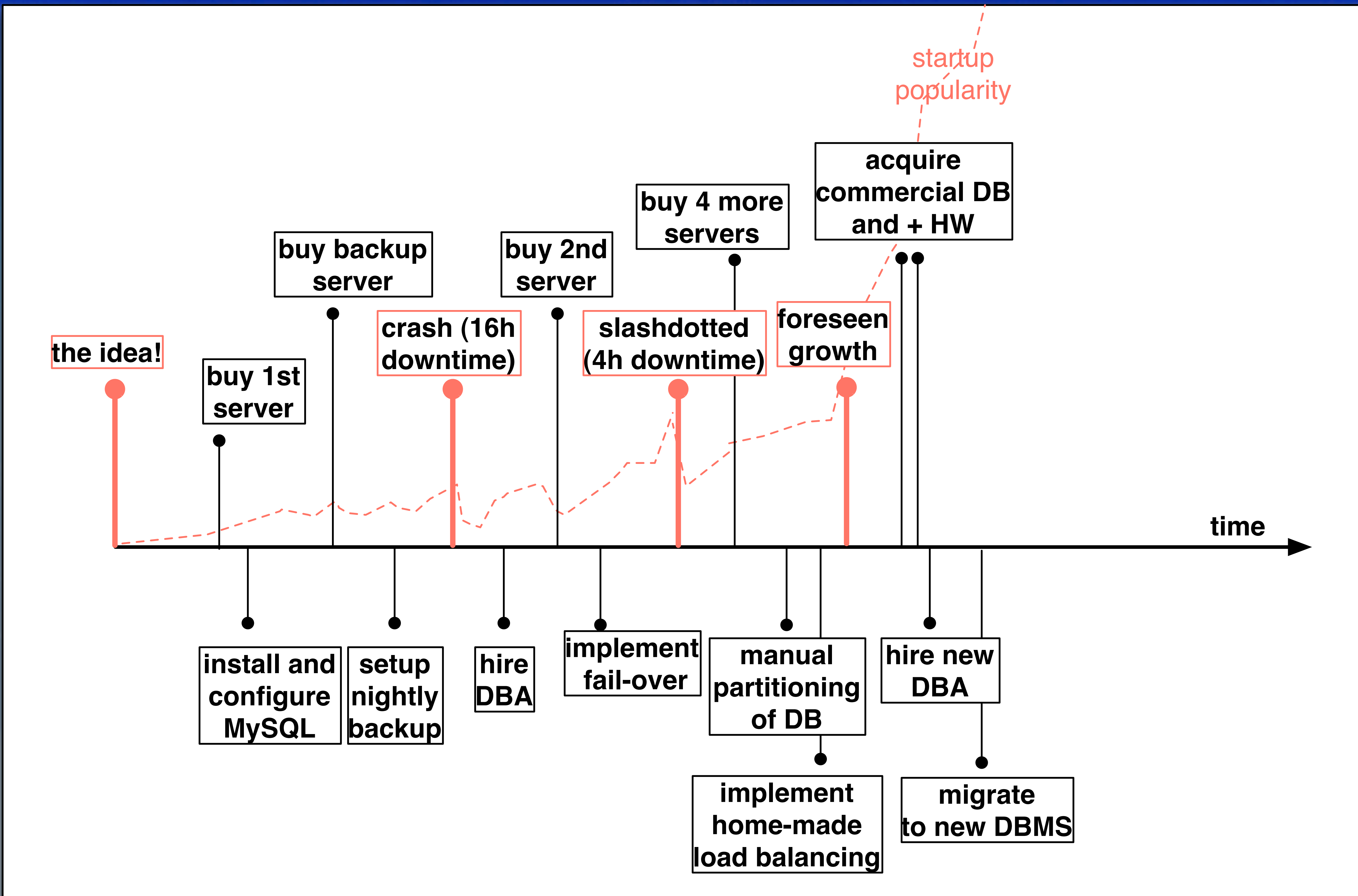
Startup success: a story of DB drama



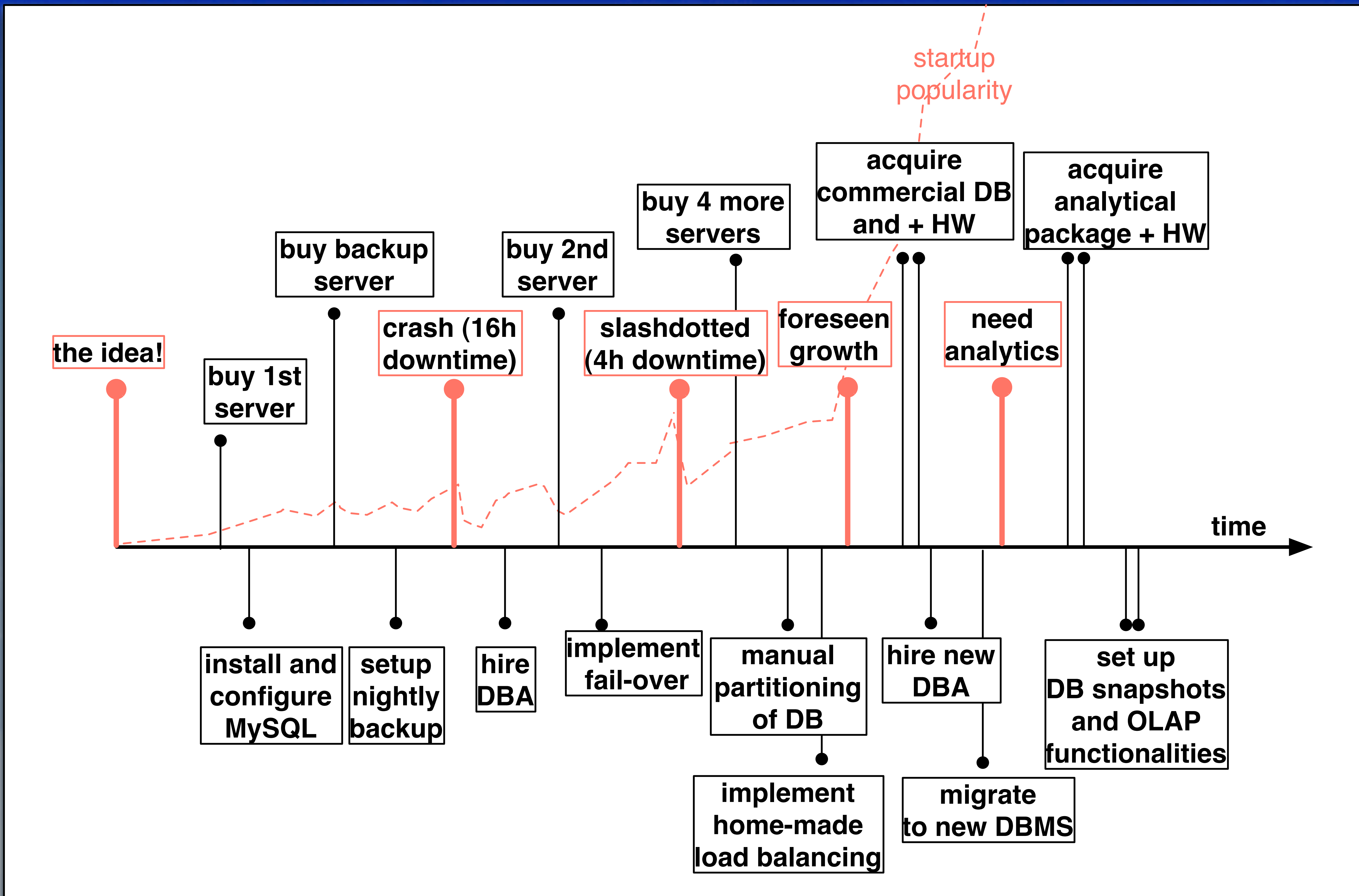
Startup success: a story of DB drama



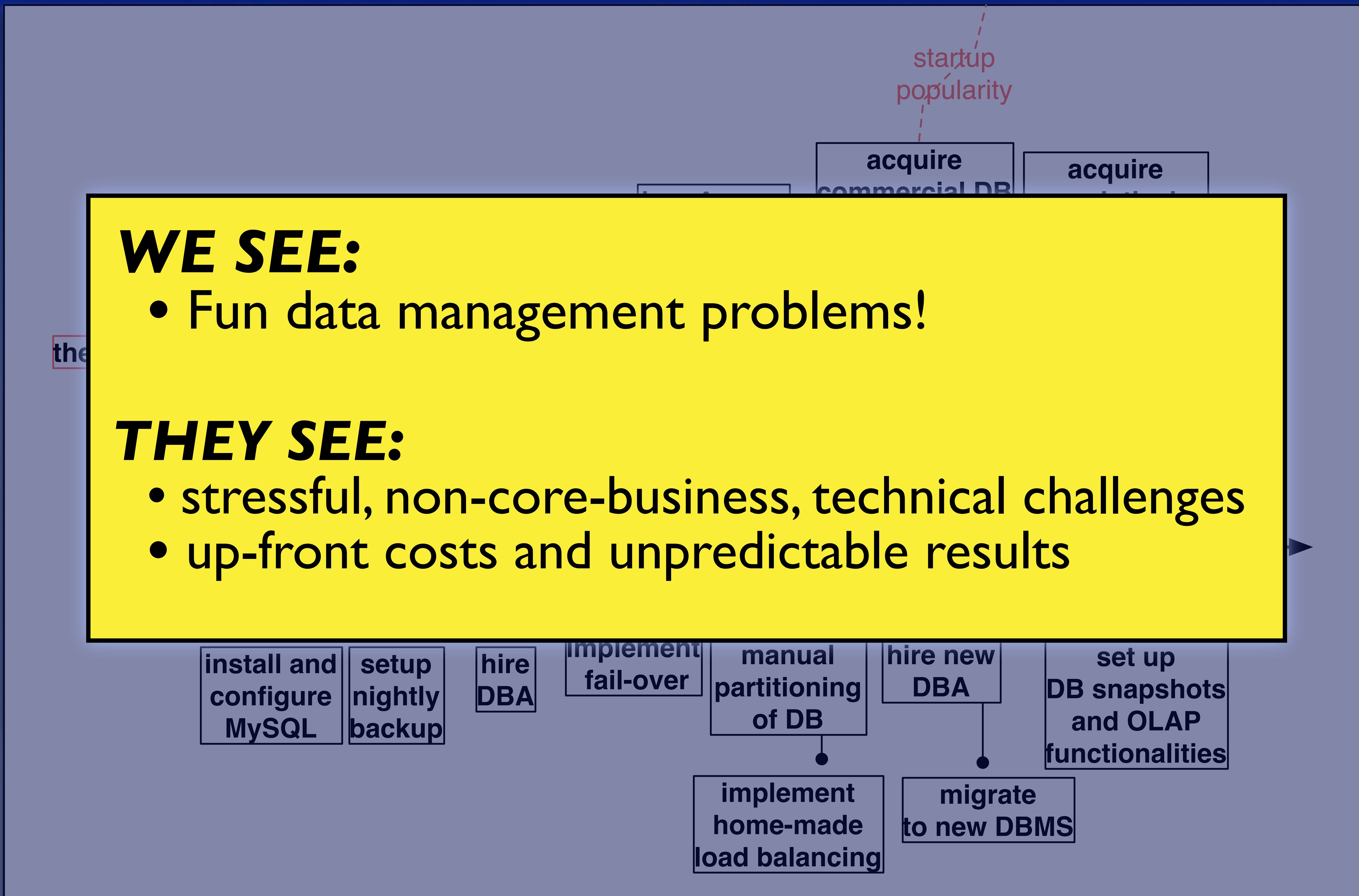
Startup success: a story of DB drama



Startup success: a story of DB drama



Startup success: a story of DB drama



WE SEE:

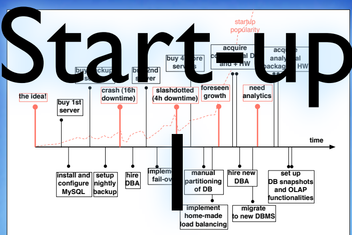
- Fun data management problems!

THEY SEE:

- stressful, non-core-business, technical challenges
- up-front costs and unpredictable results

Startup success: a story of DB drama

Start-up



Startup success: a story of DB drama

Startup success: a story of DB drama

WHAT IS WRONG WITH THIS PICTURE?

HW resources are under utilized:

- peak-provisioning
- HW for infrequent tasks
- low power-efficiency

Same problems solved over and over:

- hw/sw selection
- configuration and tuning
- scalability and load balancing

Database as a Service

- **Transactional, Relational DB service**
  - hide complexity
  - exploit resource pooling
  - increase automation
  - (both for *private* and *public* cloud)

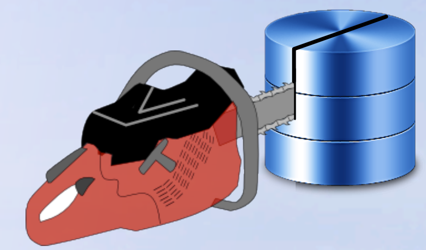
Existing Services

- **Existing Commercial DB Services:**
  - Amazon RDS, SQL Azure *(and many others)*
- **What they got right:**
  - simplified provisioning/deployment
  - reduced administration/tuning headaches
- **What is still missing?**
  - workload placement (to reduce hw cost)
  - automatic partitioning (to scale out)
  - encryption (to achieve data privacy)

Relational Cloud: Key Contributions

Workload Placement *[under submission]*

- *consolidation up to 17:1*



Automatic Partitioning *[pvldb2010]*

- matches or outperforms manual sharding

Provable Data Privacy *[under submission]*

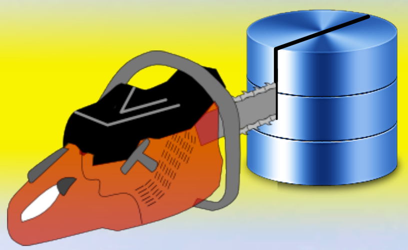
- run SQL over encrypted data
- low overhead (22.5% impact on TPC-C throughput)

Relational Cloud: Key Contributions

Workload Placement *[under submission]*

- *consolidation up to 17:1*

Automatic Partitioning *[pvldb2010]*



- matches or outperforms manual sharding

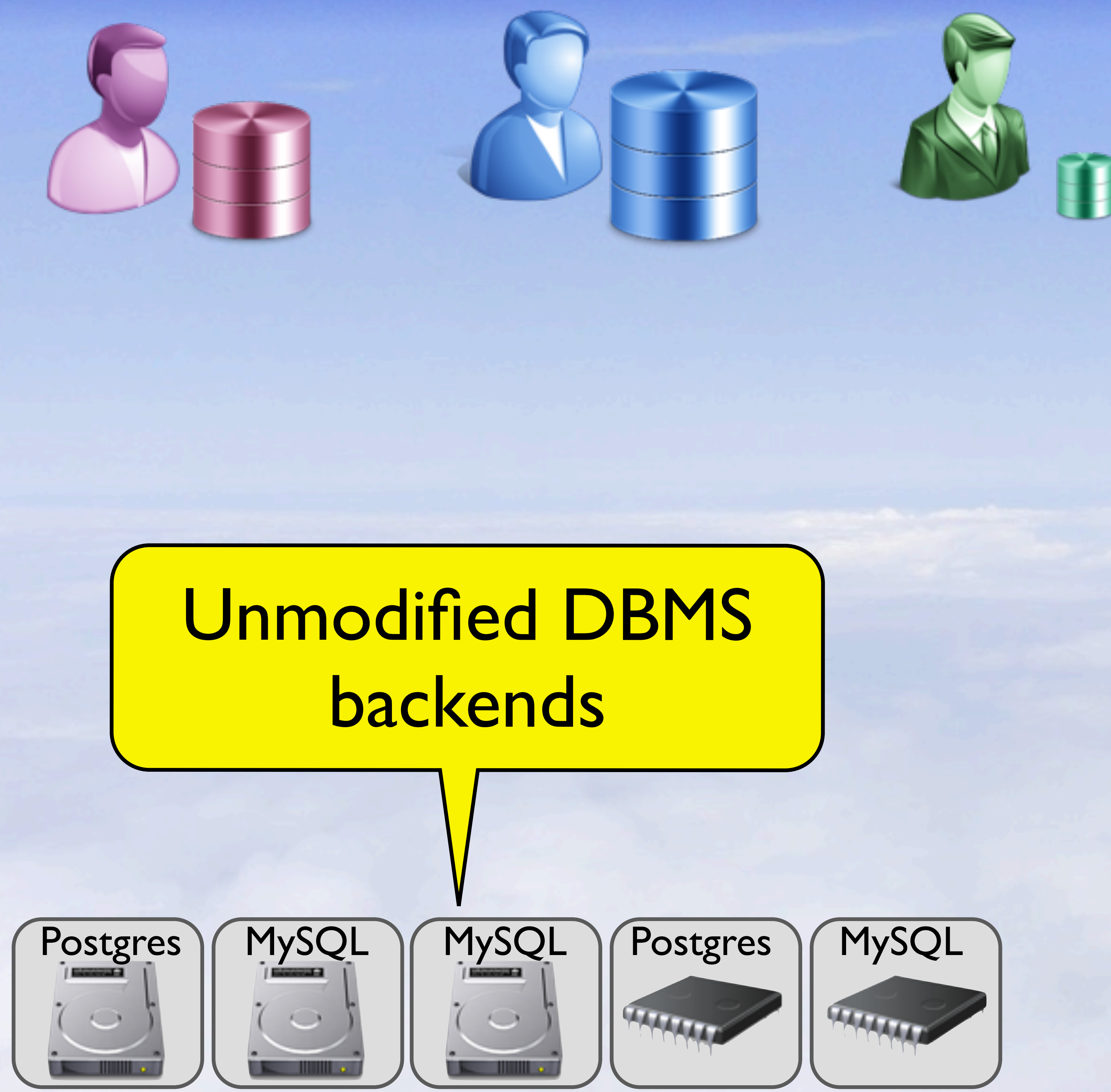
Provable Data Privacy *[under submission]*

- run SQL over encrypted data
- low overhead (22.5% impact on TPC-C throughput)

Relational Cloud Architecture

Users
(and their workloads)

Relational Cloud Architecture



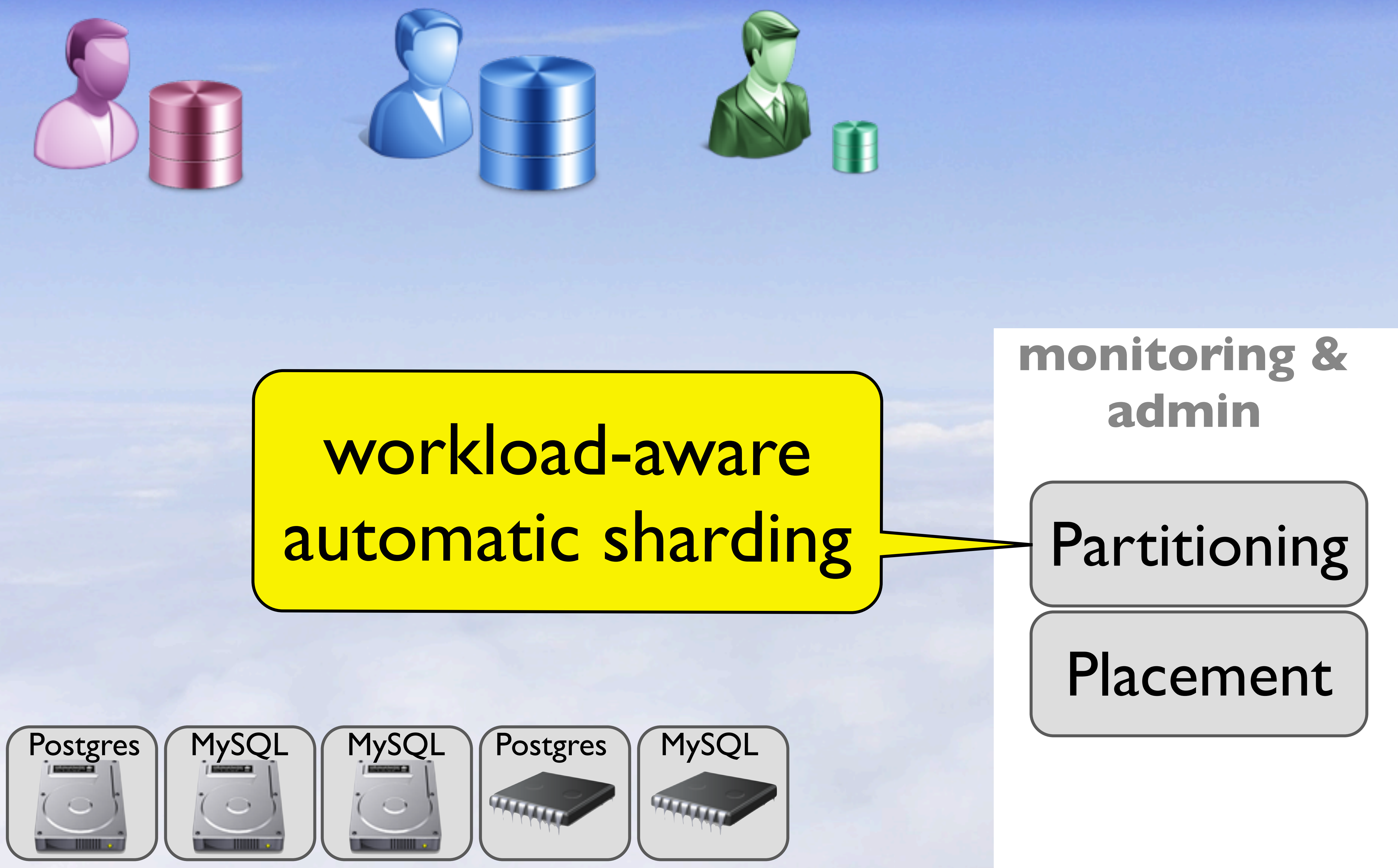
Relational Cloud Architecture

monitoring &
admin

workload-aware
consolidation

Placement

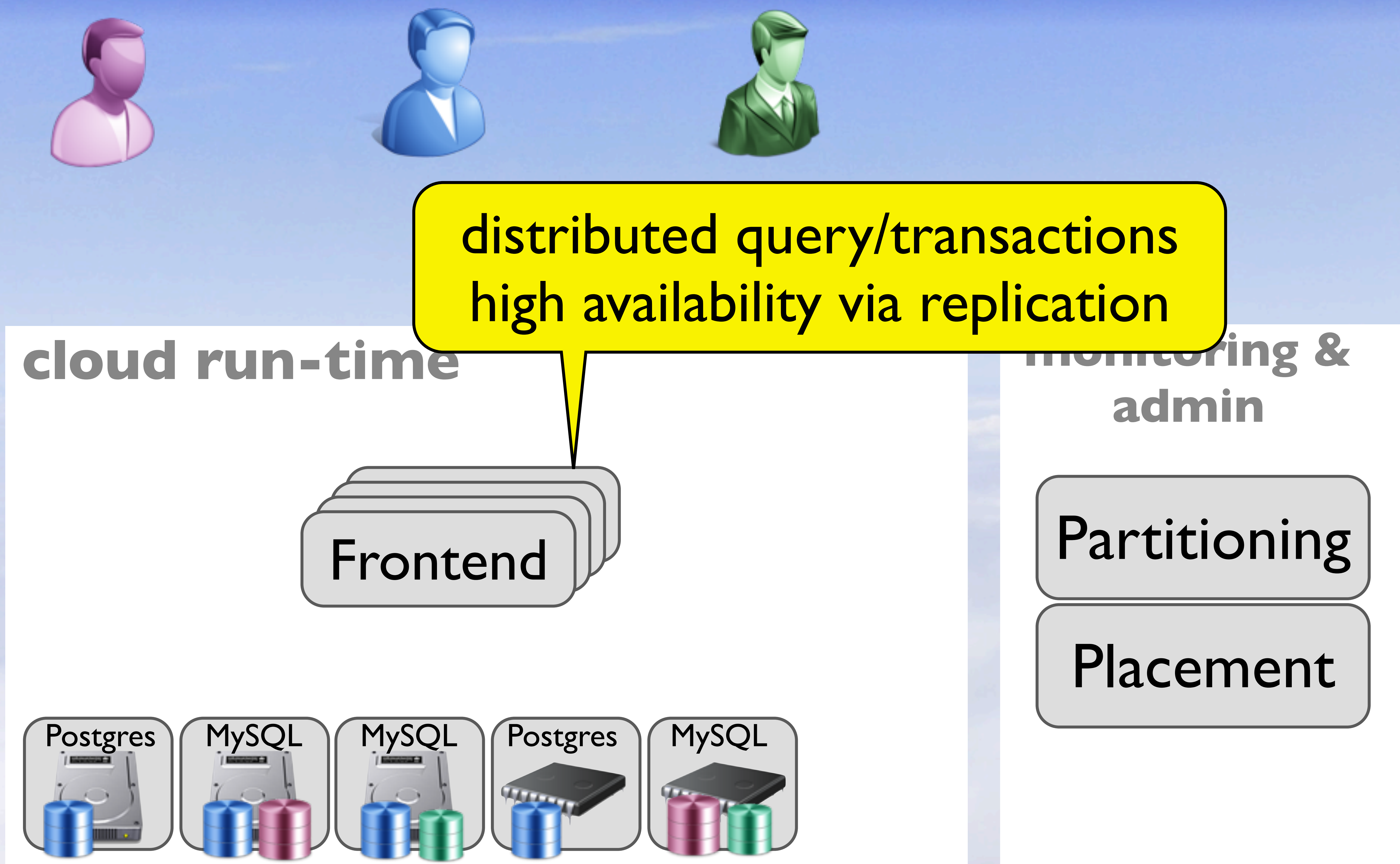
Relational Cloud Architecture



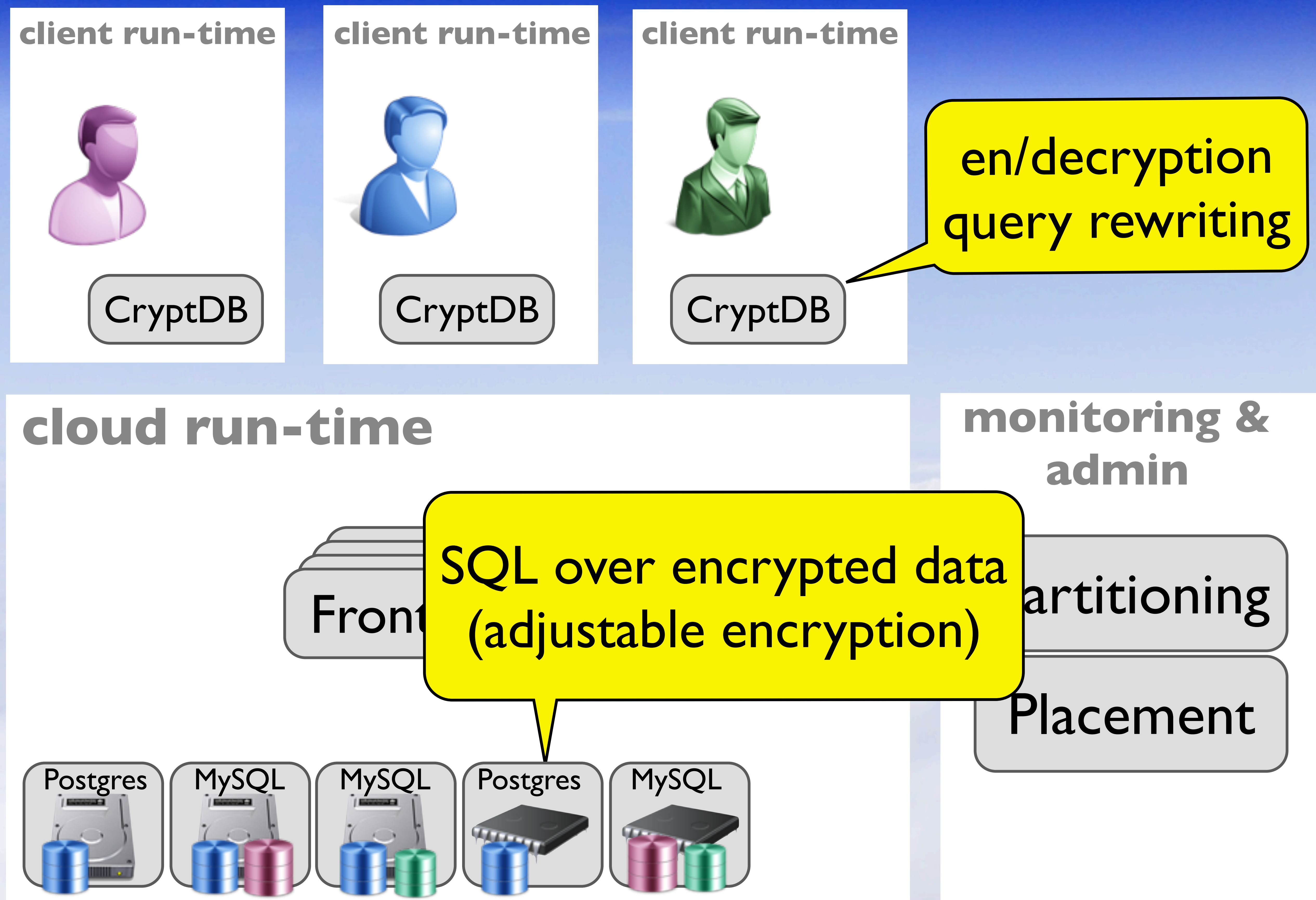
Relational Cloud Architecture

Relational Cloud Architecture

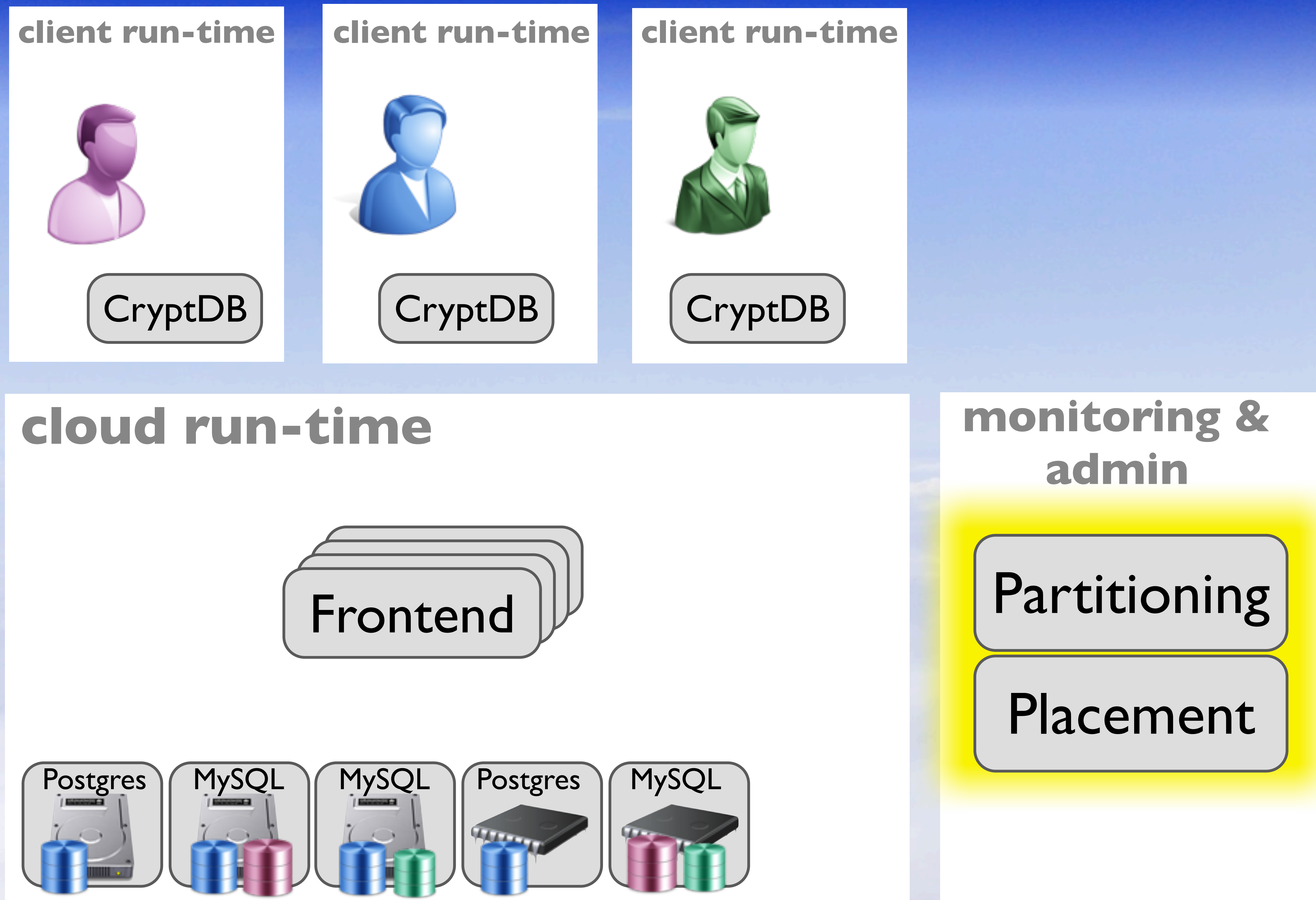
Relational Cloud Architecture



Relational Cloud Architecture



Relational Cloud Architecture



Workload Placement

Scenario:

- Each workload initially run on a dedicated server
- Consolidate DB machines

Problem Definition:

Allocate workloads to servers in a way that:

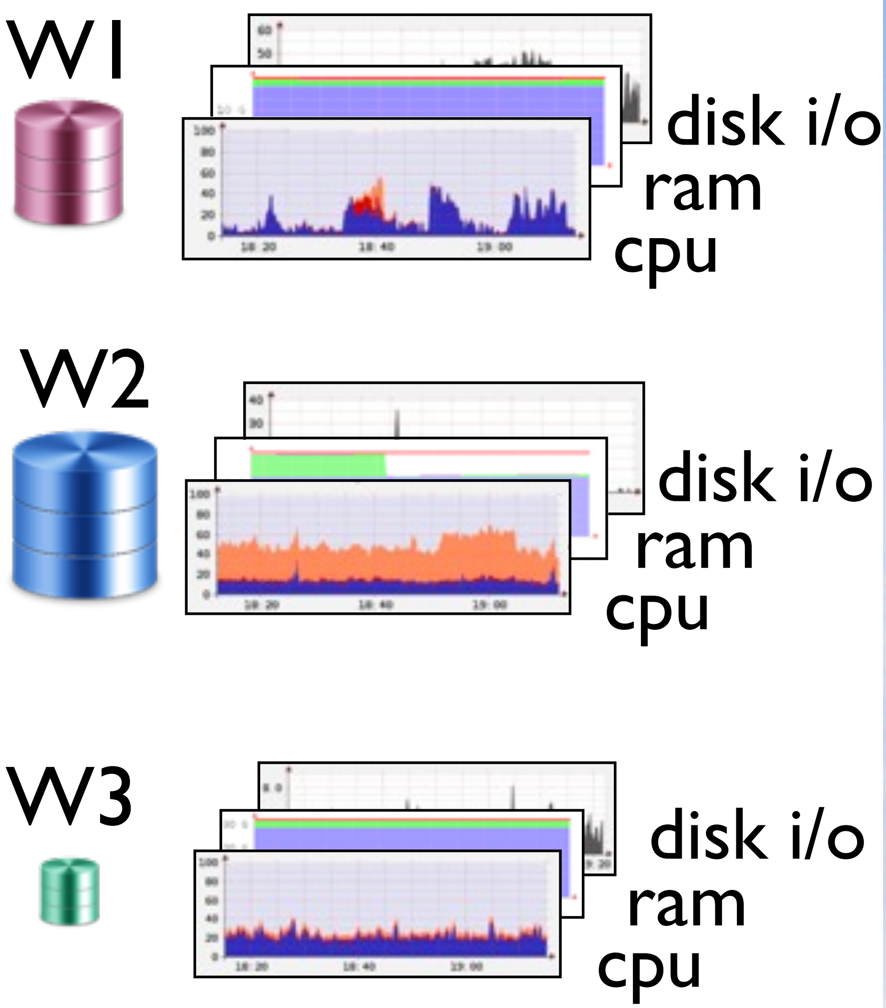
1) minimizes number of servers used

2) balances load across servers

3) maintains performance unchanged

Workload Placement

measure resource utilization

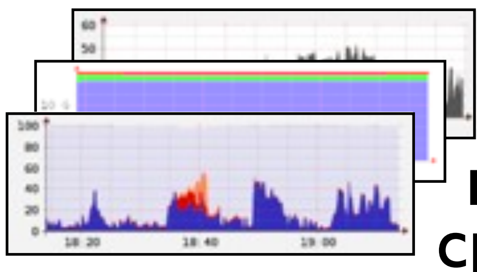


DBMSs tend to use all available resources

Workload Placement

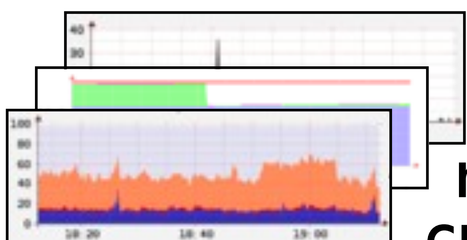
measure resource utilization

W1



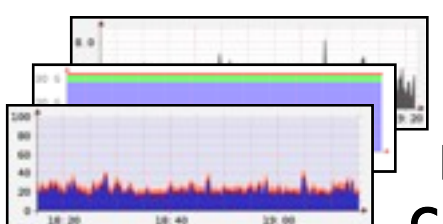
disk i/o
ram
cpu

W2



disk i/o
ram
cpu

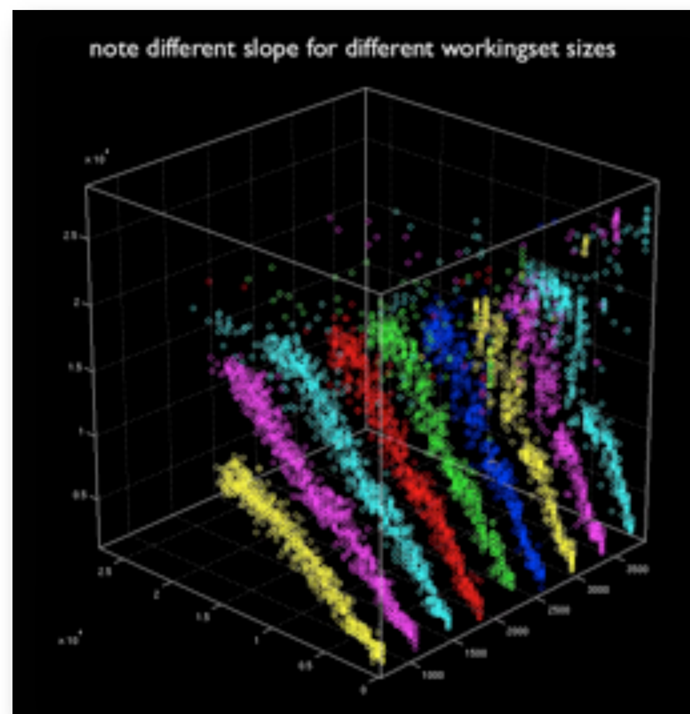
W3



disk i/o
ram
cpu

estimate combined load

numerical models

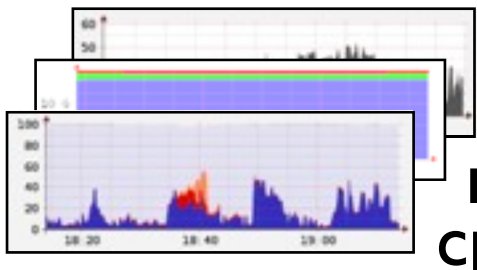


resource
non-linearities

Workload Placement

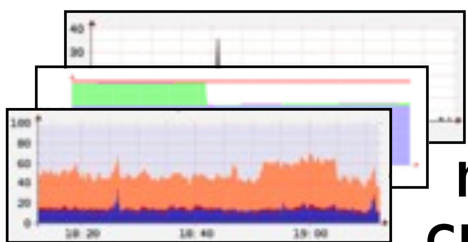
measure resource utilization

W1



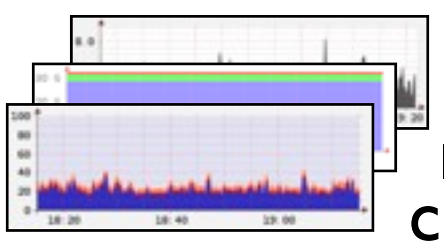
disk i/o
ram
cpu

W2



disk i/o
ram
cpu

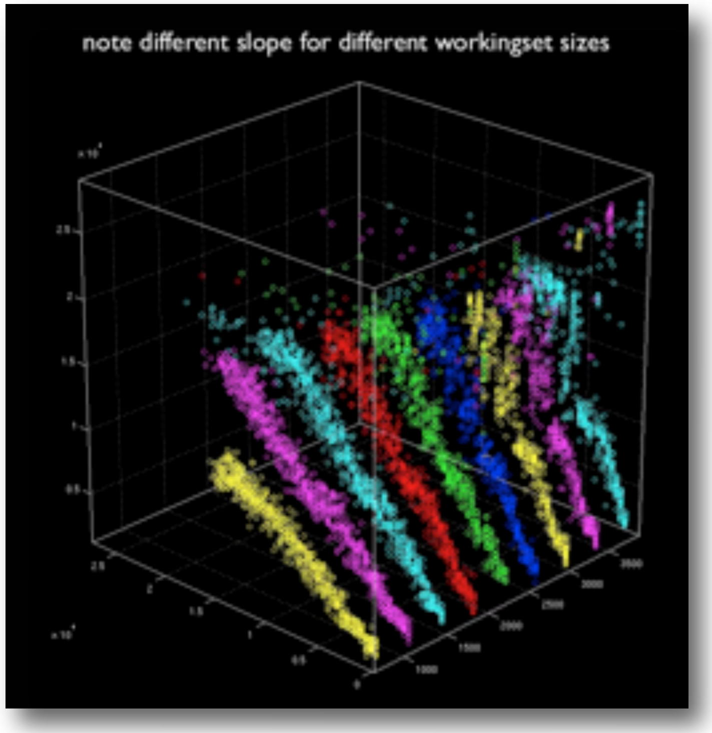
W3



disk i/o
ram
cpu

estimate combined load

numerical models



find optimal assignment

non-linear programming

$$
\begin{aligned}
\min_{x} \quad & \sum_{tj}(e^{(\sum_i C_{ti}*x_{ij})} * signum(\sum_i x_{ij})); \\
\text{subject to} \quad & \forall i \sum_j x_{ij} = R_i; \\
& \forall j\ max_t(\sum_i CPU_{ti} * x_{ij}) < MaxCPU_j; \\
& \forall j\ max_t(\sum_i MEM_{ti} * x_{ij}) < MaxMEM_j; \\
& \forall j\ diskModel(DISK_{ti}, x_{ij}) < MaxDISK_j; \\
& \dots \\
& additional\ placement\ constraints \\
& \dots \\
& \forall i, j x_{ij} \in N; 0 \le x_{ij} \le 1
\end{aligned}
$$

non-linear constraints and objective function

Workload Placement

Non-Linear Integer Constraints:

- No overcommit of HW using:
    - historical resource time-series
    - combined resource estimation
- Each workload is assigned
- HA via replication (e.g., W2)

servers

workloads

	S1	S2	S3	S4
W1	0	0	0	1
W2	1	0	1	0
W3	1	0	0	0
W4	0	0	1	0
W5	0	0	0	1
W6	0	0	1	0
W7	1	0	0	1

Workload Placement

Non-Linear Integer Constraints:

- No overcommit of HW using:
  - historical resource time-series
  - combined resource estimation
- Each workload is assigned
- HA via replication (e.g., W2)

servers

workloads	S1	S2	S3	S4
W1	0	0	0	1
W2	1	0	1	0
W3	1	0	0	0
W4	0	0	1	0
W5	0	0	0	1
W6	0	0	1	0
W7	1	0	0	1

Workload Placement

Non-Linear Integer Constraints:

- No overcommit of HW using:
  - historical resource time-series
  - combined resource estimation
- Each workload is assigned
- HA via replication (e.g., W2)

servers

workloads

	S1	S2	S3	S4
W1	0	0	0	1
W2	1	0	1	0
W3	1	0	0	0
W4	0	0	1	0
W5	0	0	0	1
W6	0	0	1	0
W7	1	0	0	1

Workload Placement

Non-Linear Integer Constraints:

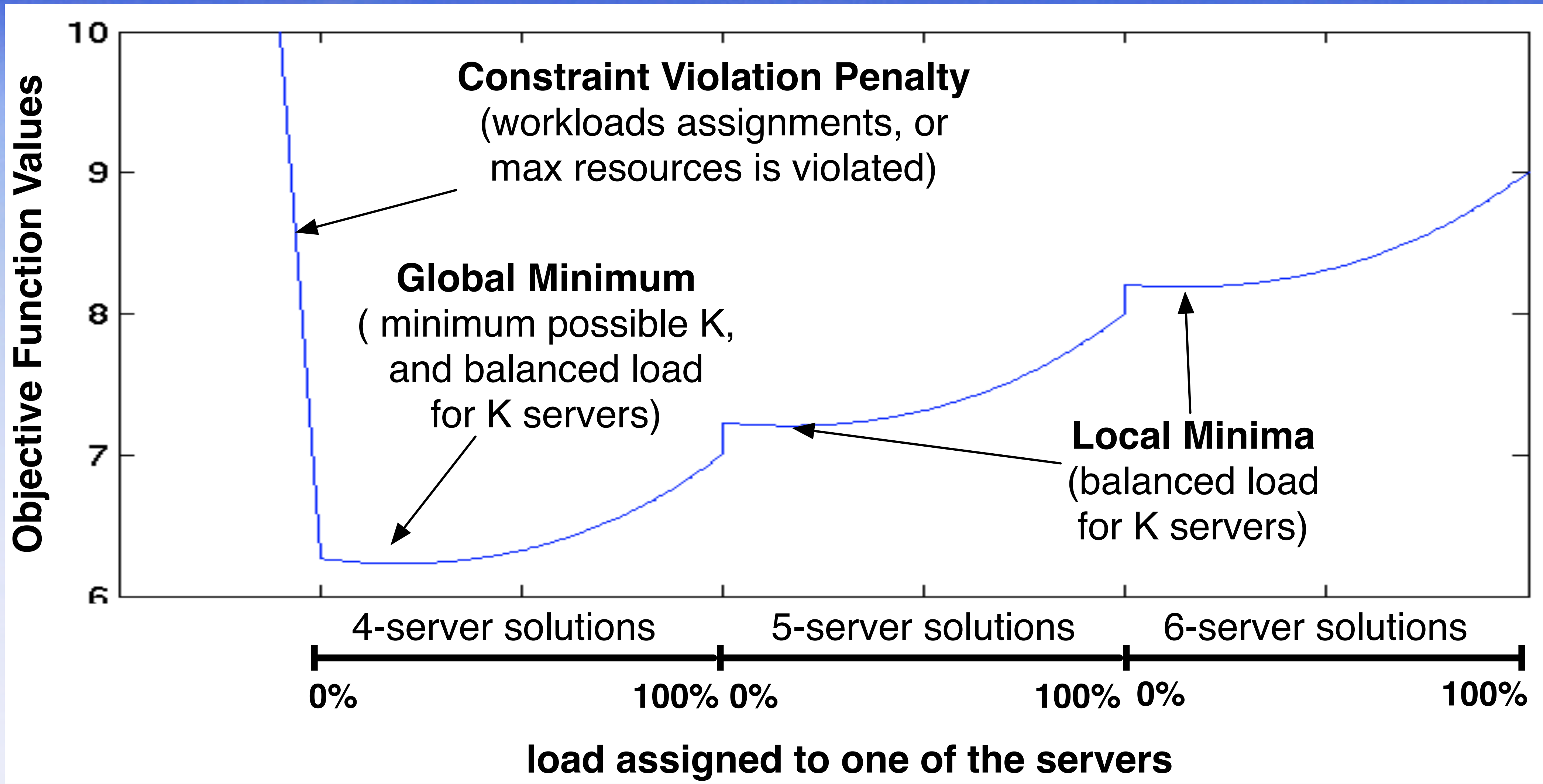
- No overcommit of HW using:
  - historical resource time-series
  - combined resource estimation
- Each workload is assigned
- HA via replication (e.g., W2)

servers

workloads

	S1	S2	S3	S4
W1	0	0	0	1
W2	1	0	1	0
W3	1	0	0	0
W4	0	0	1	0
W5	0	0	0	1
W6	0	0	1	0
W7	1	0	0	1

Workload Placement



Objective function:

- minimize servers (use *SIGNUM*)
- maximize balance (use *EXP*)

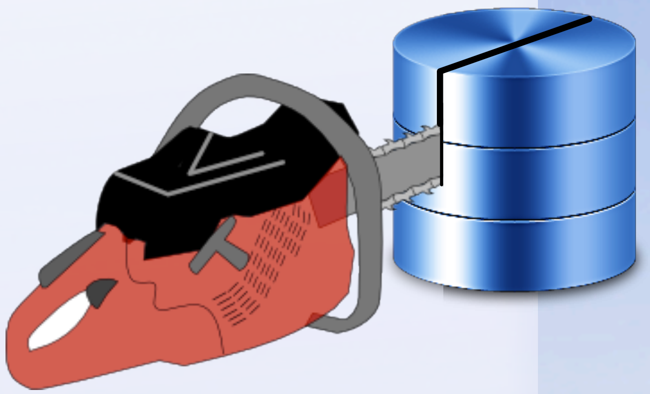
Workload Placement Results

- **Validated our approach on synthetic data**
- **Estimated real-world impact:**
  - Load statistics from production data-centers: *Wikipedia, Wikia, SecondLife, MIT*
  - Huge potential consolidation: **6:1** to **17:1**

Partitioning

Why:

- scalability
- manageability



Problem Definition:

Partition the database into N chunks in a way that maximizes the workload performance

Partitioning

Why:

- scalability

KEY TO SCALABILITY (OLTP/Web):

- Limit percentage of distributed transaction

Partition the database into N chunks in a way that maximizes the workload performance

Graph-based Partitioning

INPUT

Database

Workload Trace

```
BEGIN
 UPDATE account SET bal=60k WHERE id=2;
 SELECT * FROM account WHERE id=5;
COMMIT
```

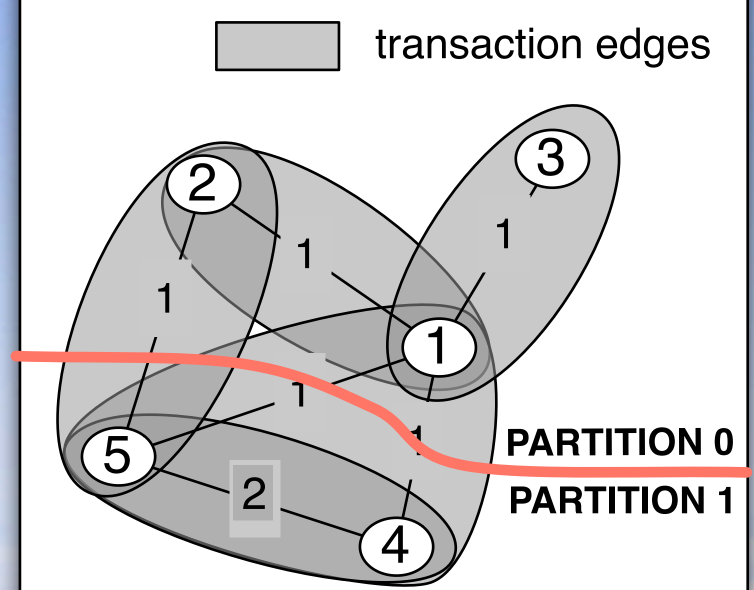
```
BEGIN
 UPDATE account SET bal=bal-1k WHERE id=1;
 UPDATE account SET bal=bal+1k WHERE id=2;
COMMIT
```

```
BEGIN
 SELECT * FROM account WHERE id IN {1,3}
ABORT
```

```
BEGIN
 UPDATE SET bal=bal+1k WHERE bal < 100k;
COMMIT
```

```
BEGIN
 UPDATE SET bal=bal*1.1 WHERE id >= 4;
COMMIT
```

GRAPH REPRESENTATION



EXPLANATION

P: 0	ID < 4
P: 1	ID >= 4

Graph-based Partitioning

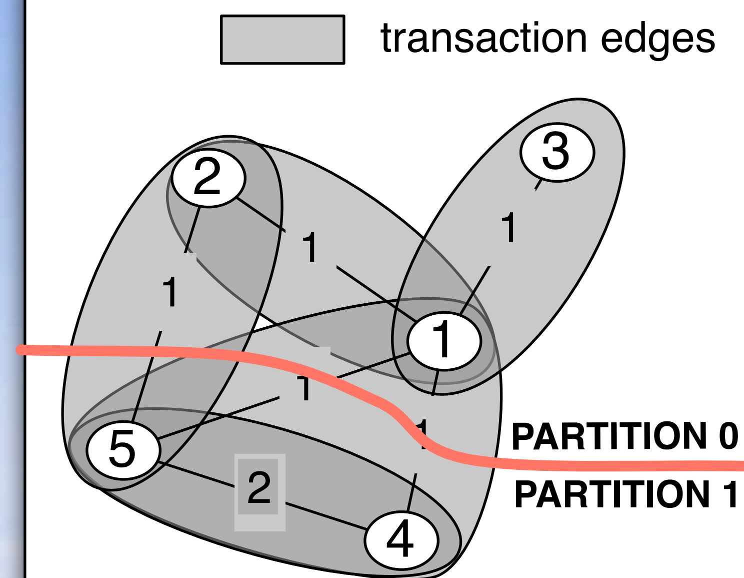
INPUT

Database

Workload Trace

```
BEGIN
 UPDATE account SET bal=60k WHERE id=2;
 SELECT * FROM account WHERE id=5;
COMMIT

BEGIN
 UPDATE account SET bal=bal-1k WHERE id=1;
 UPDATE account SET bal=bal+1k WHERE id=2;
COMMIT

BEGIN
 SELECT * FROM account WHERE id IN {1,3}
ABORT

BEGIN
 UPDATE SET bal=bal+1k WHERE bal < 100k;
COMMIT

BEGIN
 UPDATE SET bal=bal*1.1 WHERE id >= 4;
COMMIT
```

GRAPH REPRESENTATION



EXPLANATION

P: 0	ID < 4
P: 1	ID >= 4

Input
(logs+preprocessing)

Graph-based Partitioning

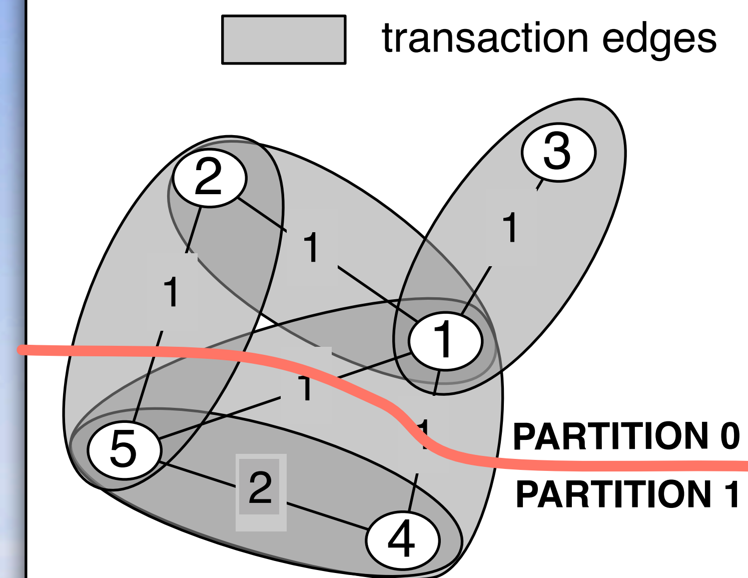
INPUT

Database

Workload Trace

```
BEGIN
 UPDATE account SET bal=60k WHERE id=2;
 SELECT * FROM account WHERE id=5;
COMMIT

BEGIN
 UPDATE account SET bal=bal-1k WHERE id=1;
 UPDATE account SET bal=bal+1k WHERE id=2;
COMMIT

BEGIN
 SELECT * FROM account WHERE id IN {1,3}
ABORT

BEGIN
 UPDATE SET bal=bal+1k WHERE bal < 100k;
COMMIT

BEGIN
 UPDATE SET bal=bal*1.1 WHERE id >= 4;
COMMIT
```

GRAPH REPRESENTATION



EXPLANATION

P: 0	ID < 4
P: 1	ID >= 4

Graph Partitioning (METIS)

Graph-based Partitioning

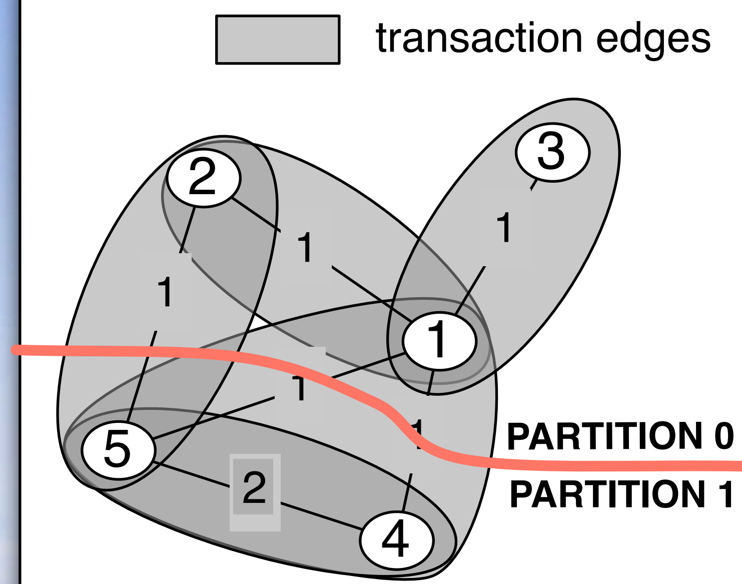
INPUT

Database

Workload Trace

```
BEGIN
 UPDATE account SET bal=60k WHERE id=2;
 SELECT * FROM account WHERE id=5;
COMMIT

BEGIN
 UPDATE account SET bal=bal-1k WHERE id=1;
 UPDATE account SET bal=bal+1k WHERE id=2;
COMMIT

BEGIN
 SELECT * FROM account WHERE id IN {1,3}
ABORT

BEGIN
 UPDATE SET bal=bal+1k WHERE bal < 100k;
COMMIT

BEGIN
 UPDATE SET bal=bal*1.1 WHERE id >= 4;
COMMIT
```

GRAPH REPRESENTATION



EXPLANATION

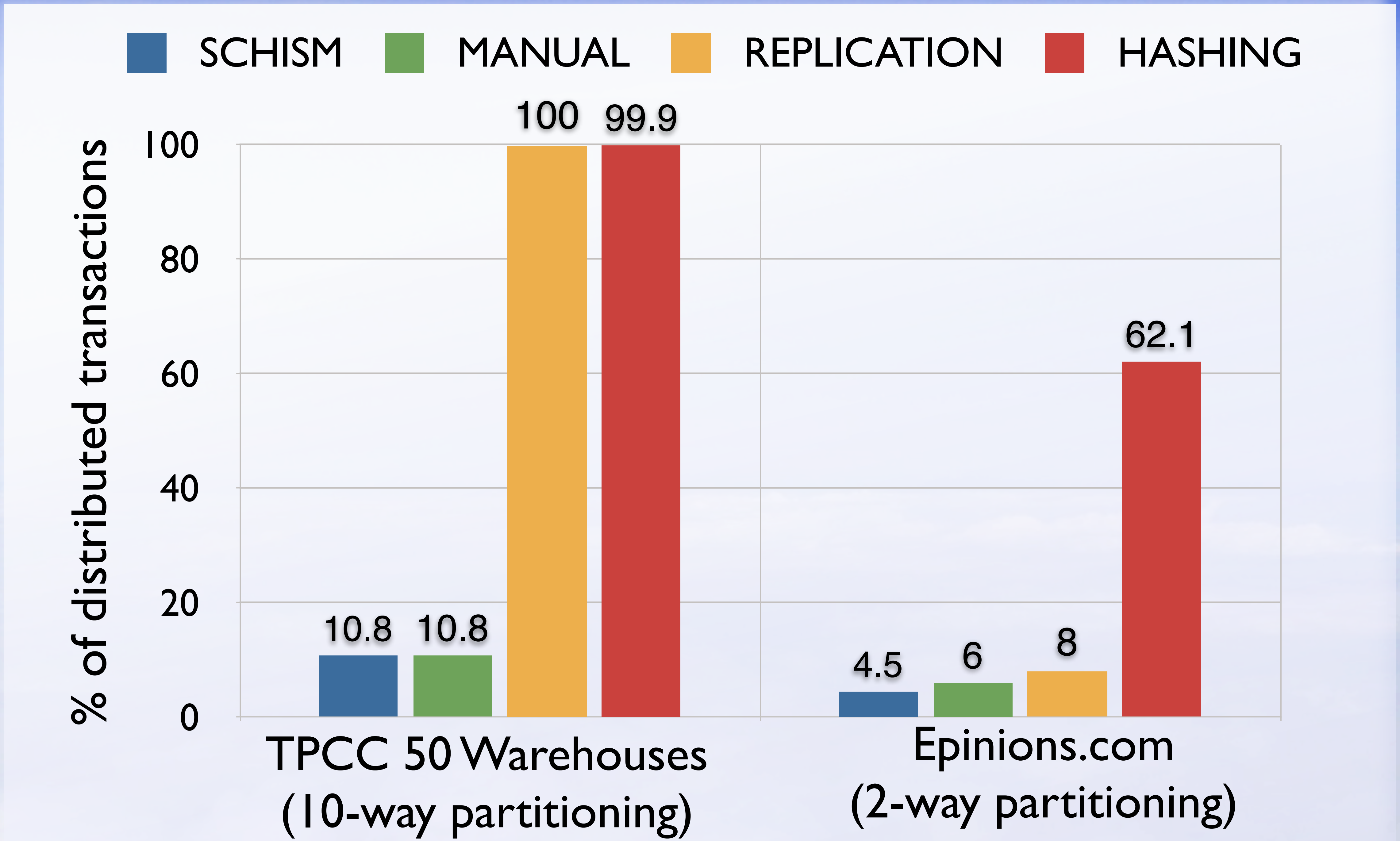
P: 0	ID < 4
P: 1	ID >= 4

Classification (Decision Tree)

Graph Partitioning Demo

- Example inspired by YCSB
- Single table, short scans

Partitioning Results



Conclusions

- **Database as a Service has *real* potential**
- **Key Features to fully enable DBaaS**
  - Workload Placement *(up to 17:1 consolidation)*
  - Automatic Partitioning *(matches manual partitioning)*
  - Provable Privacy *(22.5% performance impact)*
- **What's next?**
  - Live Migration
  - Dynamic reallocation/repartitioning

Conclusions

- **Database as a Service has *real* potential**
- **Key Features to fully enable DBaaS**
  - Workload Placement *(up to 17:1 consolidation)*
  - Automatic Partitioning *(matches manual partitioning)*
  - Provable Privacy *(22.5% performance impact)*
- **What's next?**
  - Live Migration
  - Dynamic reallocation/repartitioning

For follow-up comments and job-offers: **curino@mit.edu**