PAPOC '18

APRIL 23, 2018

FINE-GRAINED DISTRIBUTED CONSISTENCY GUARANTEES WITH EFFECT ORCHESTRATION

Kiarash Rahmani
Gowtham Kaki
Suresh Jagannathan

STRUCTURE OF THE TALK

- Background
- Problem Definition
- Our Solution
- Evaluation Results

BACKGROUND (1)

- Low latency and high availability are vital for modern web-scale applications

NETFLIX

VISA

YouTube

ebay

- Low latency and high availability are vital for modern web-scale applications
- 1 second delay in page response can result in a 7% reduction in sales

NETFLIX

VISA

YouTube

ebay

- Low latency and high availability are vital for modern web-scale applications
- 1 second delay in page response can result in a 7% reduction in sales
- Usually rely on the underlying data stores for consistency, integrity, durability and availability of the data.

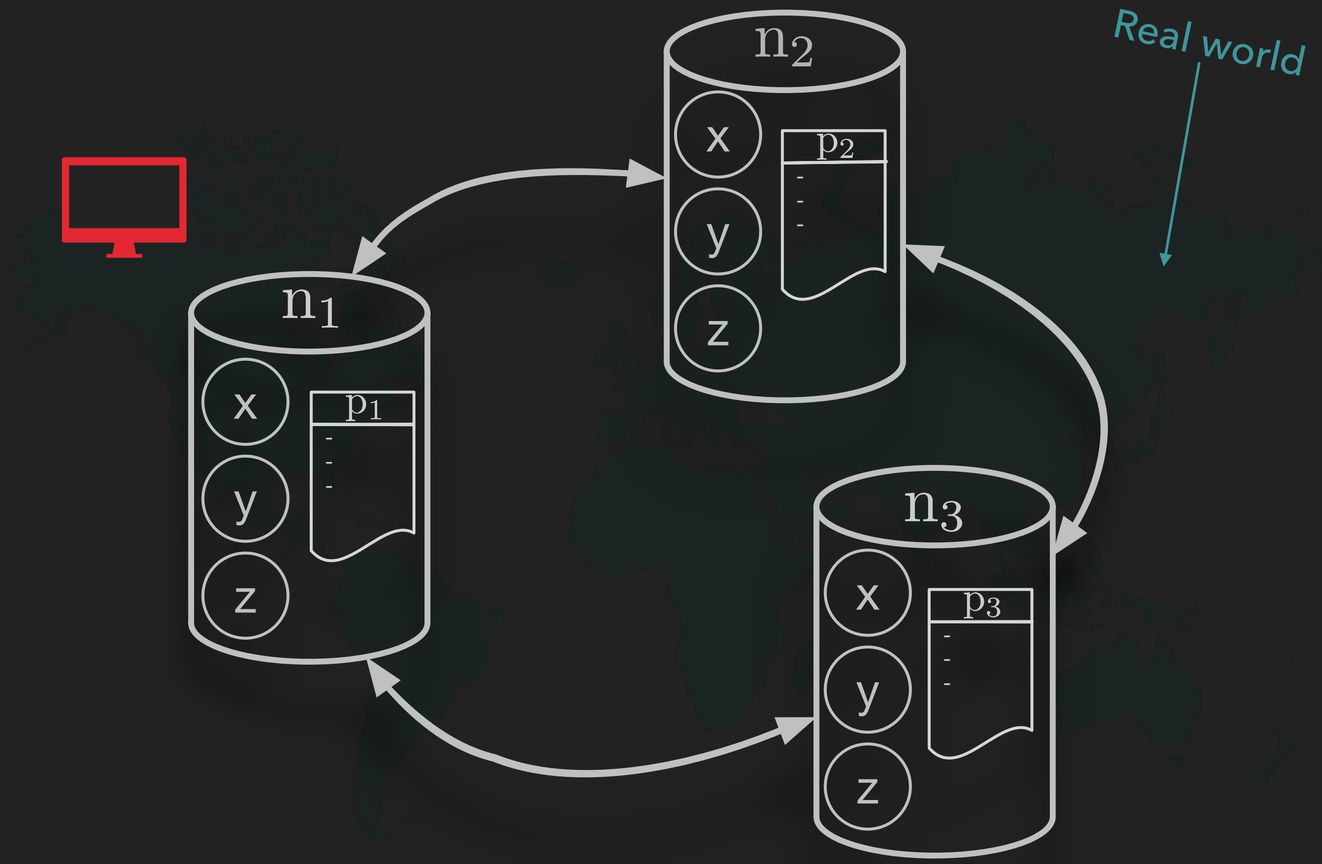
NETFLIX

VISA

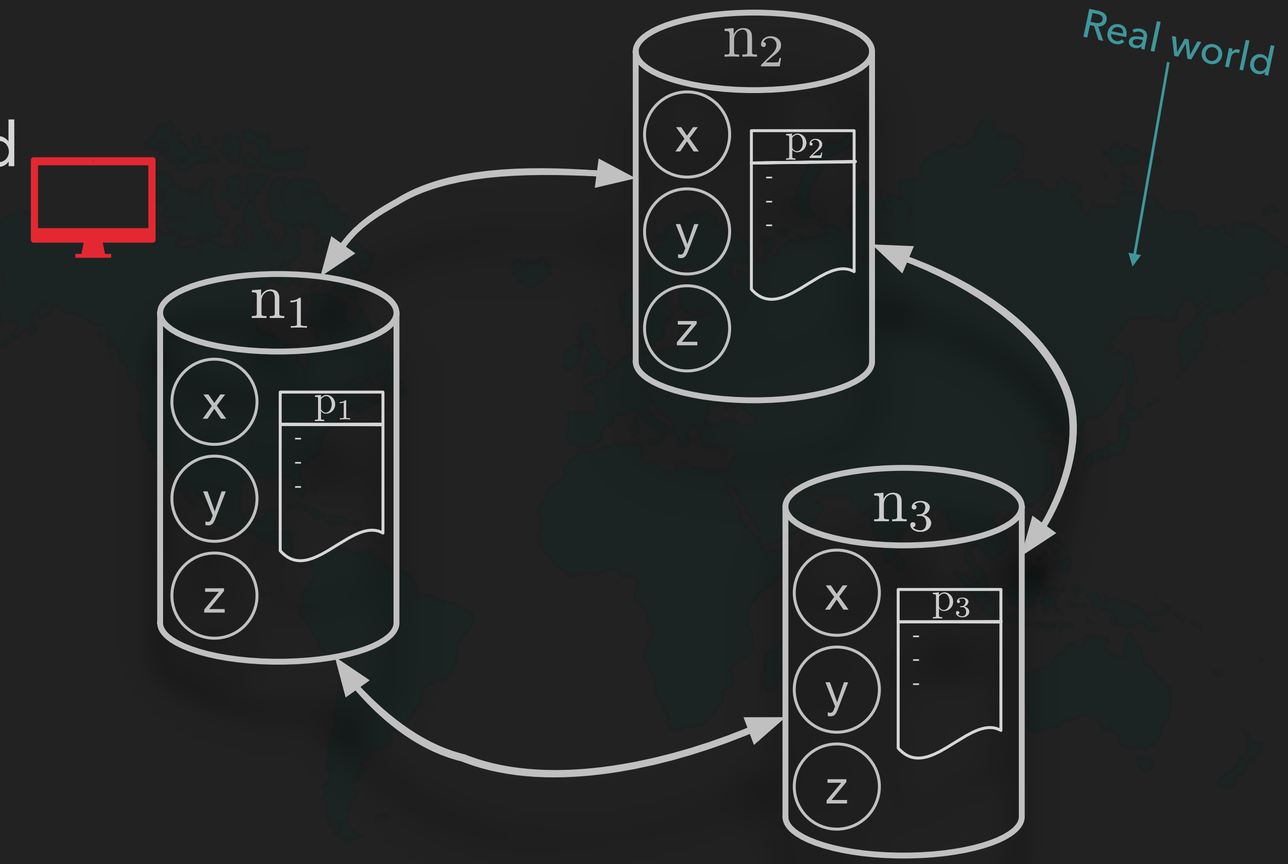
YouTube

ebay

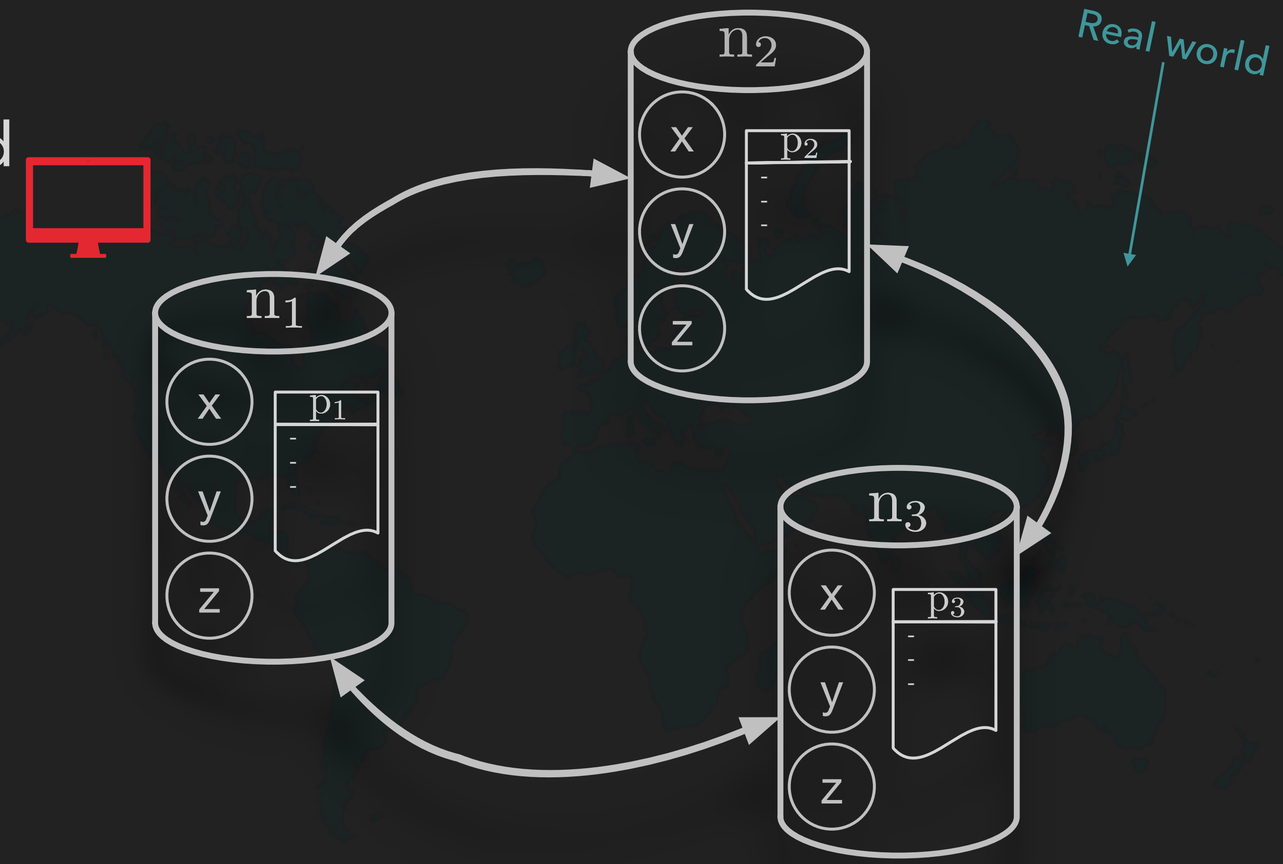
BACKGROUND (2)



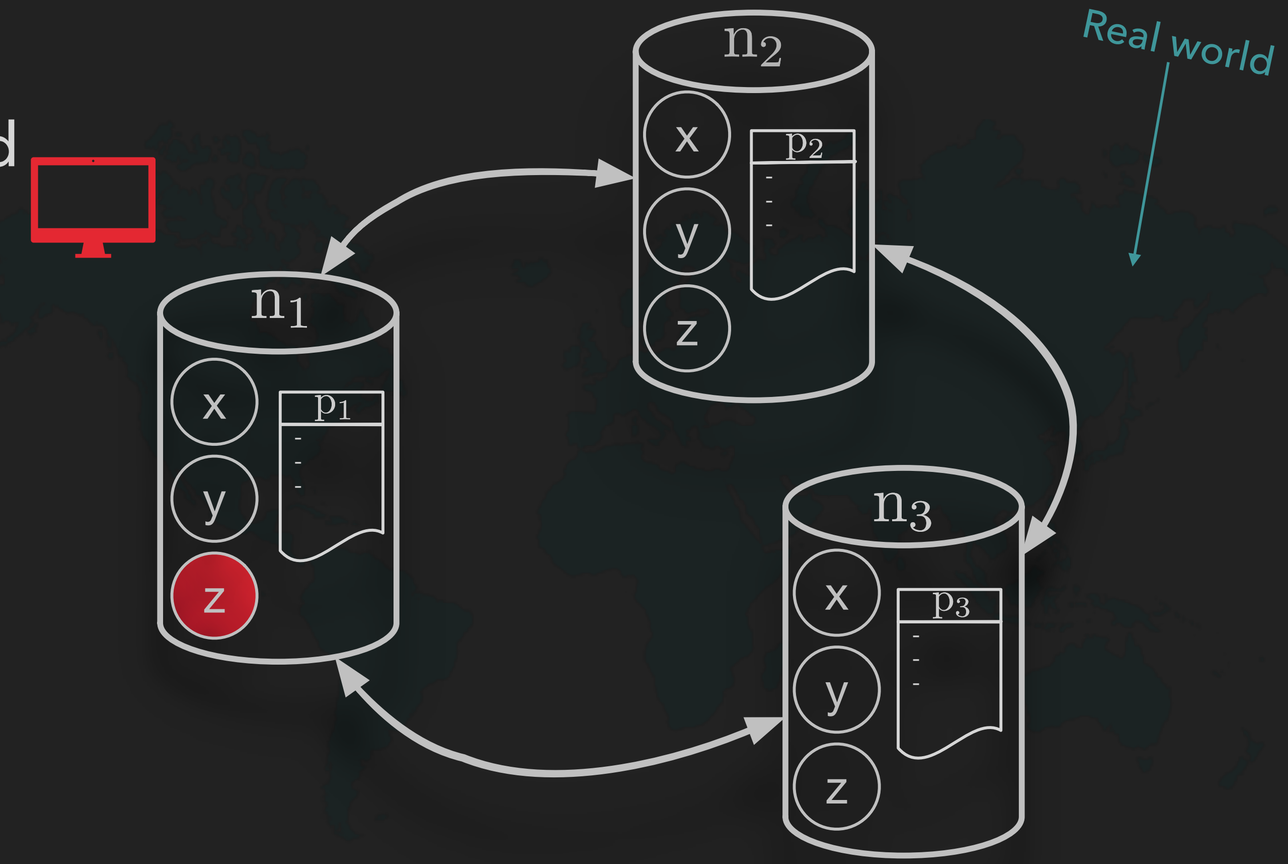
- Scalability and availability are achieved by geographic replication of data



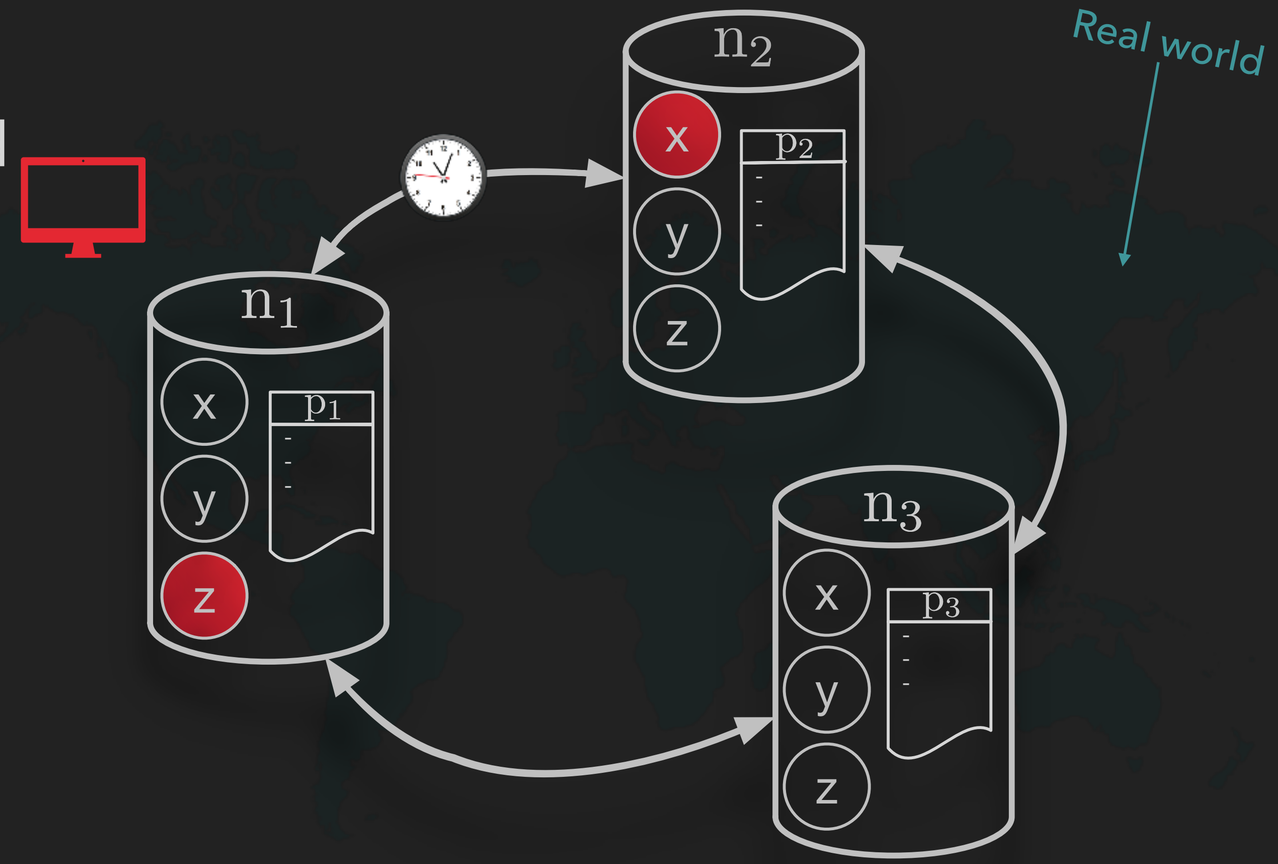
- Scalability and availability are achieved by geographic replication of data
- Unfortunately, in real world, inter-continent connections are very slow and links are broken frequently



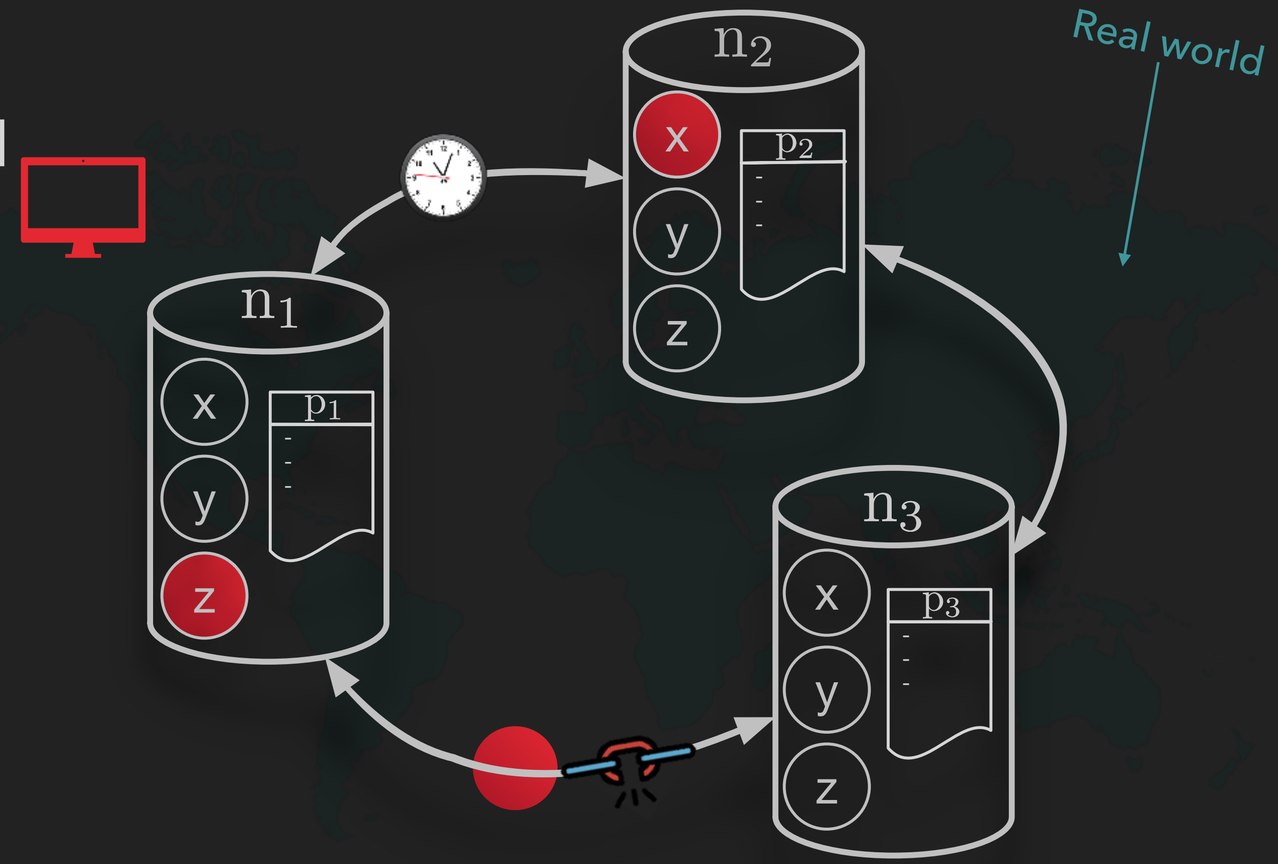
- Scalability and availability are achieved by geographic replication of data
- Unfortunately, in real world, inter-continent connections are very slow and links are broken frequently



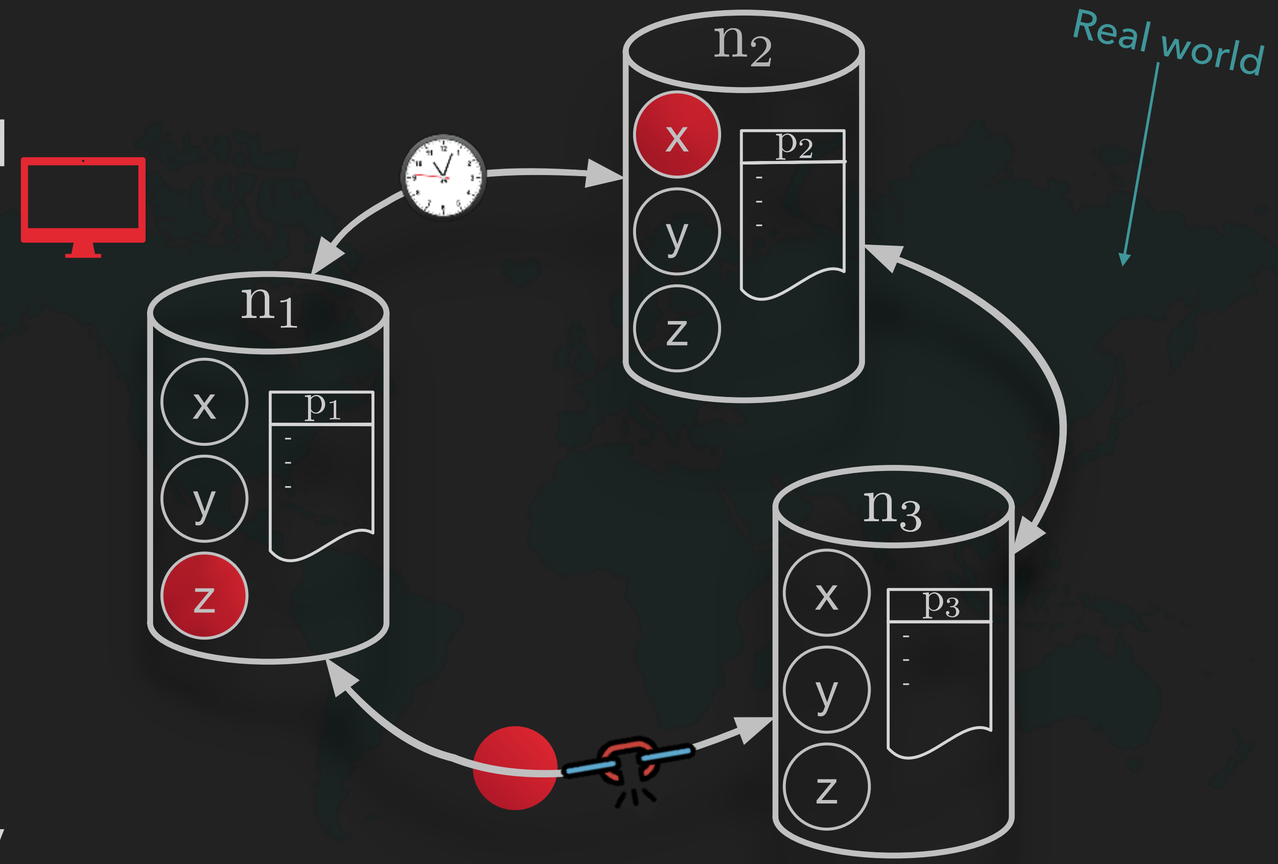
- Scalability and availability are achieved by geographic replication of data
- Unfortunately, in real world, inter-continent connections are very slow and links are broken frequently



- Scalability and availability are achieved by geographic replication of data
- Unfortunately, in real world, inter-continent connections are very slow and links are broken frequently



- Scalability and availability are achieved by geographic replication of data
- Unfortunately, in real world, inter-continent connections are very slow and links are broken frequently
- Modern data stores trade off strong forms of data consistency in favor of low response time and high availability (CAP theorem)



BACKGROUND (3)

- **E**ventual **C**onsistency (EC) has become a prominent consistency guarantee offered in many data stores by different vendors

- **E**ventual **C**onsistency (EC) has become a prominent consistency guarantee offered in many data stores by different vendors
- There has been a constant growth in popularity of EC data stores (and other NoSQL databases)

- **E**ventual **C**onsistency (EC) has become a prominent consistency guarantee offered in many data stores by different vendors
- There has been a constant growth in popularity of EC data stores (and other NoSQL databases)
- Building correct applications on top of EC stores, is “unnatural” and error-prone and sometimes impossible

- **E**ventual **C**onsistency (EC) has become a prominent consistency guarantee offered in many data stores by different vendors
- There has been a constant growth in popularity of EC data stores (and other NoSQL databases)
- Building correct applications on top of EC stores, is “unnatural” and error-prone and sometimes impossible
- Additional weaker forms of consistency guarantees are introduced, e.g. **C**ausal **C**onsistency (CC) and session guarantees (D. Terry et.al.)

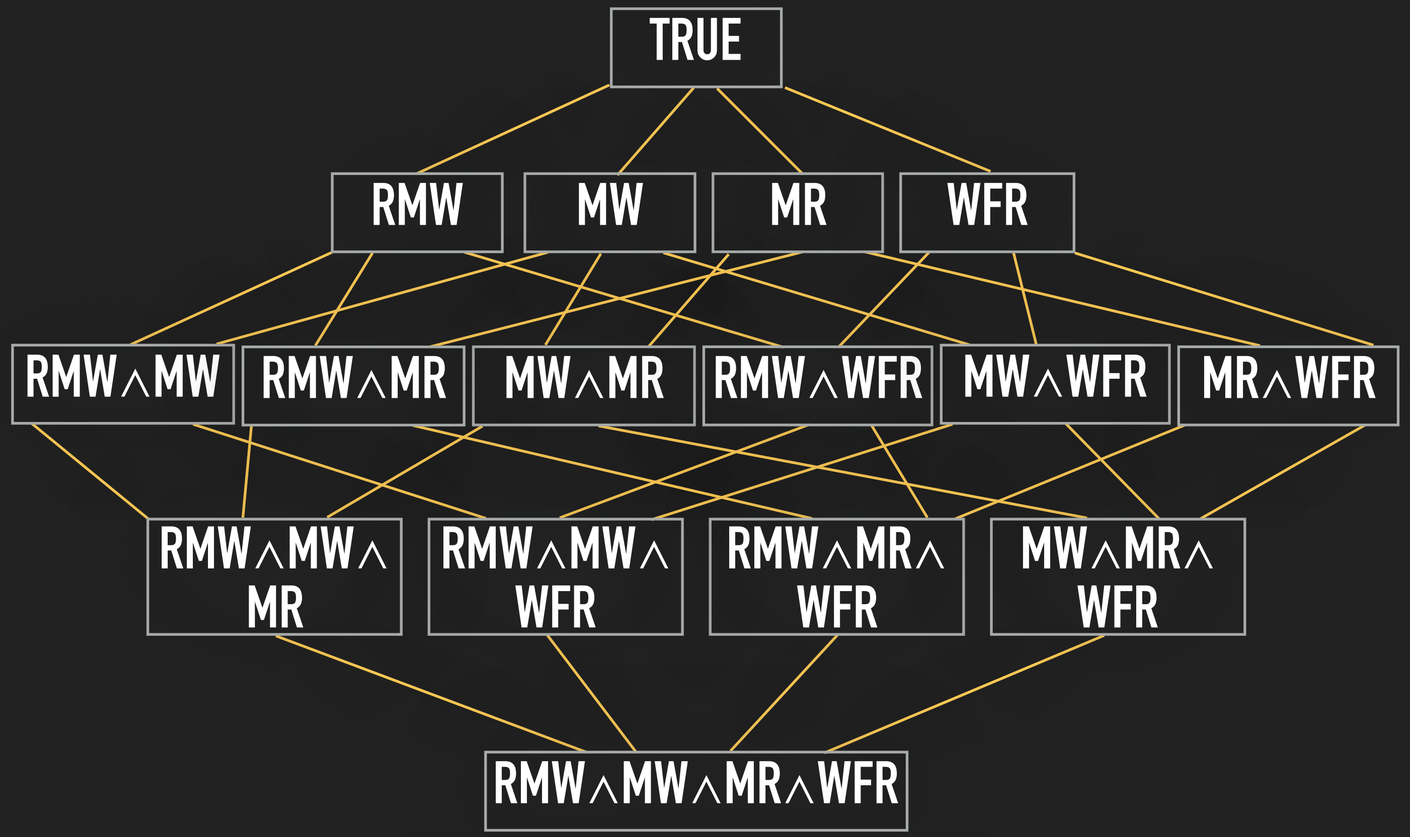
PROBLEM: LACK OF A GENERIC ENFORCEMENT MECHANISM

PROBLEM: LACK OF A GENERIC ENFORCEMENT MECHANISM

- There are many useful weak consistency guarantees

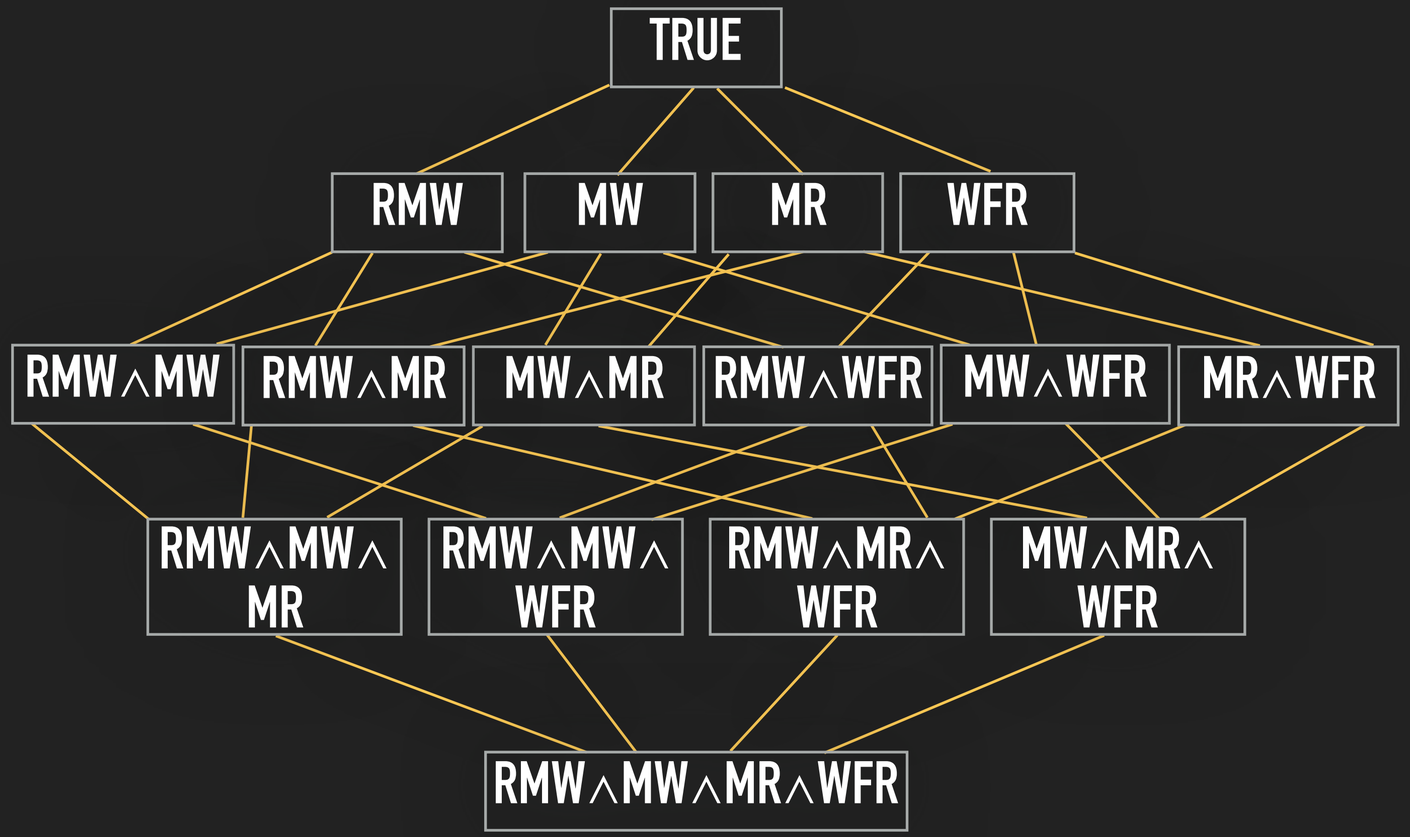
PROBLEM: LACK OF A GENERIC ENFORCEMENT MECHANISM

- There are many useful weak consistency guarantees
- Combinations of these guarantees are also required sometimes



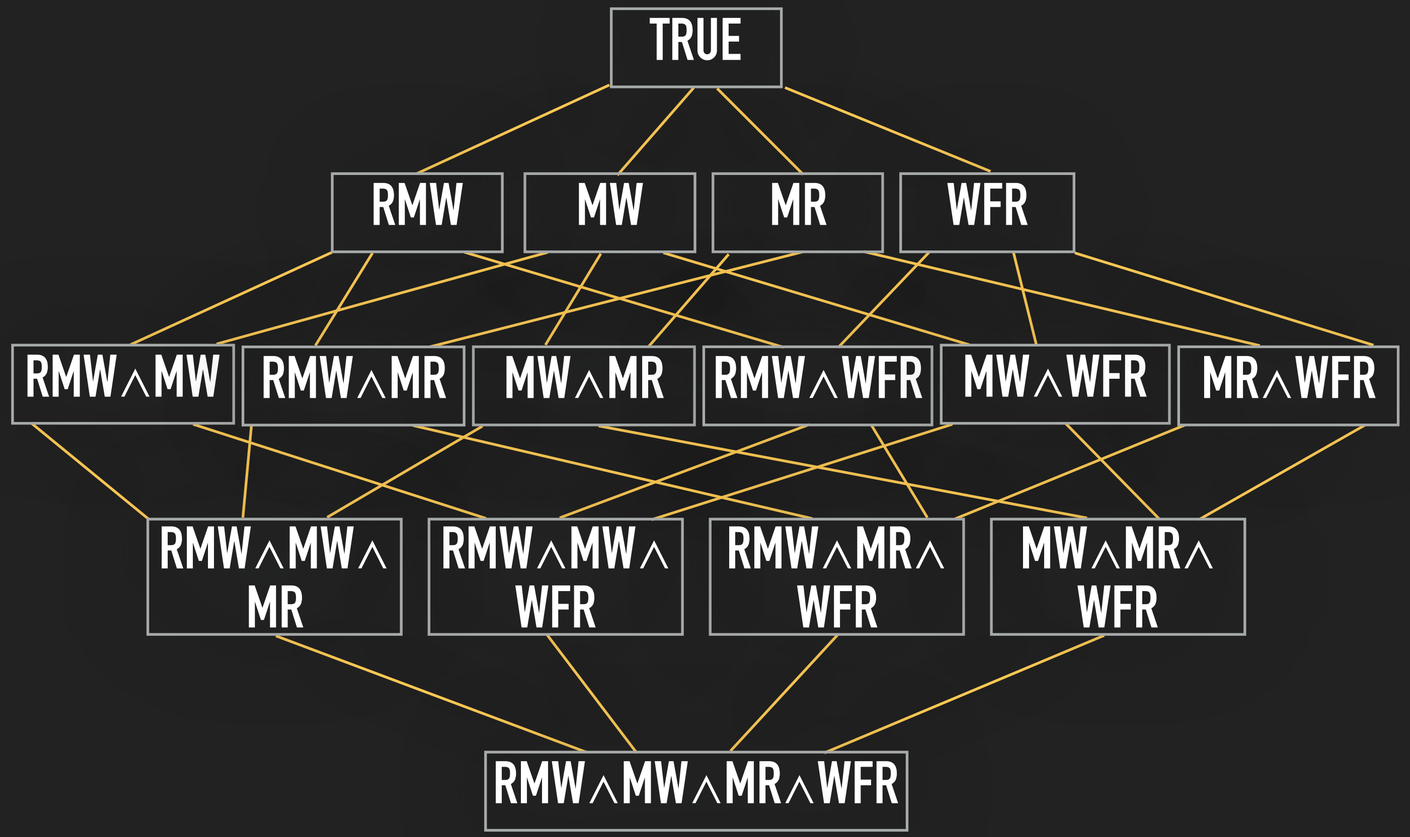
PROBLEM: LACK OF A GENERIC ENFORCEMENT MECHANISM

- There are many useful weak consistency guarantees
- Combinations of these guarantees are also required sometimes
- OTS data stores only offer a handful of them (at most)



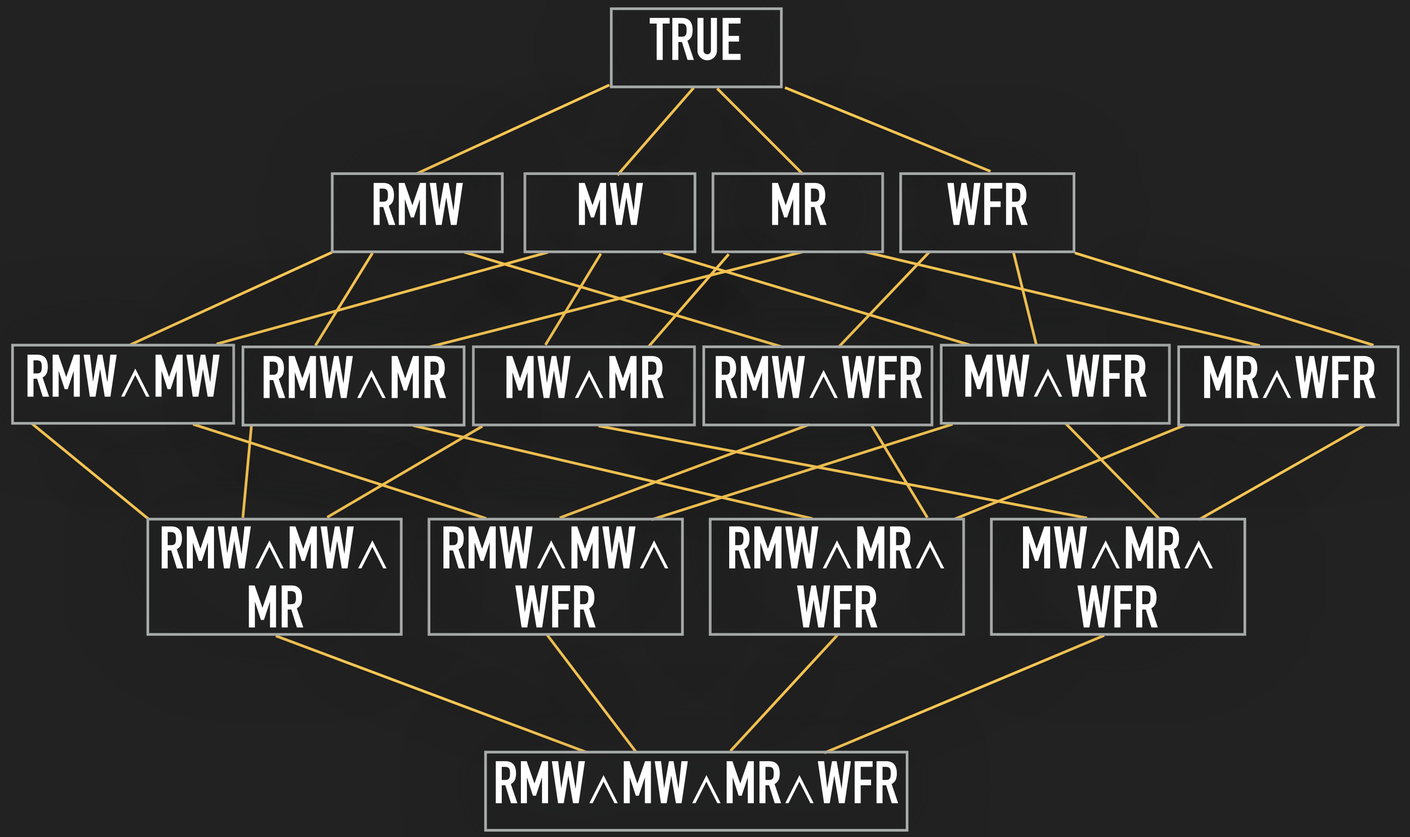
PROBLEM: LACK OF A GENERIC ENFORCEMENT MECHANISM

- There are many useful weak consistency guarantees
- Combinations of these guarantees are also required sometimes
- OTS data stores only offer a handful of them (at most)
- Designing new data stores offering every single combination is not feasible (e.g. Cassandra has 360’000 LOC)



PROBLEM: LACK OF A GENERIC ENFORCEMENT MECHANISM

- There are many useful weak consistency guarantees
- Combinations of these guarantees are also required sometimes
- OTS data stores only offer a handful of them (at most)
- Designing new data stores offering every single combination is not feasible (e.g. Cassandra has 360’000 LOC)
- System developers end up using guarantees stronger than what they need



PREVIOUS ATTEMPTS

PREVIOUS ATTEMPTS

- A generic tool offering fine-grained consistency guarantees does not exist

PREVIOUS ATTEMPTS

- A generic tool offering fine-grained consistency guarantees does not exist
- We introduce SYNCOPE to address this problem

PREVIOUS ATTEMPTS

- A generic tool offering fine-grained consistency guarantees does not exist
- We introduce SYNCOPE to address this problem
- SYNCOPE extends off the shelf EC data stores into a tunable multi-consistent data store, offering all known fine-grained consistency guarantees (and beyond!)

PREVIOUS ATTEMPTS

- A generic tool offering fine-grained consistency guarantees does not exist
- We introduce SYNCOPE to address this problem
- SYNCOPE extends off the shelf EC data stores into a tunable multi-consistent data store, offering all known fine-grained consistency guarantees (and beyond!)
- Previous attempts have offered weak consistency extensions with a much coarser granularity: e.g. Bolt-on Causal Consistency

- A generic tool offering fine-grained consistency guarantees does not exist
- We introduce SYNCOPE to address this problem
- SYNCOPE extends off the shelf EC data stores into a tunable multi-consistent data store, offering all known fine-grained consistency guarantees (and beyond!)
- Previous attempts have offered weak consistency extensions with a much coarser granularity: e.g. Bolt-on Causal Consistency
- Multi consistent shims have also been used before (e.g. Quelea) but only focusing on the relationship between the application-level requirements to a **pre-defined** set of weak consistency guarantees

TWO QUESTIONS

1. How prevalent these fine-grained consistency requirements are?

TWO QUESTIONS

1. How prevalent these fine-grained consistency requirements are?
2. Is it worth it to differentiate between all these fine-grained consistency levels?

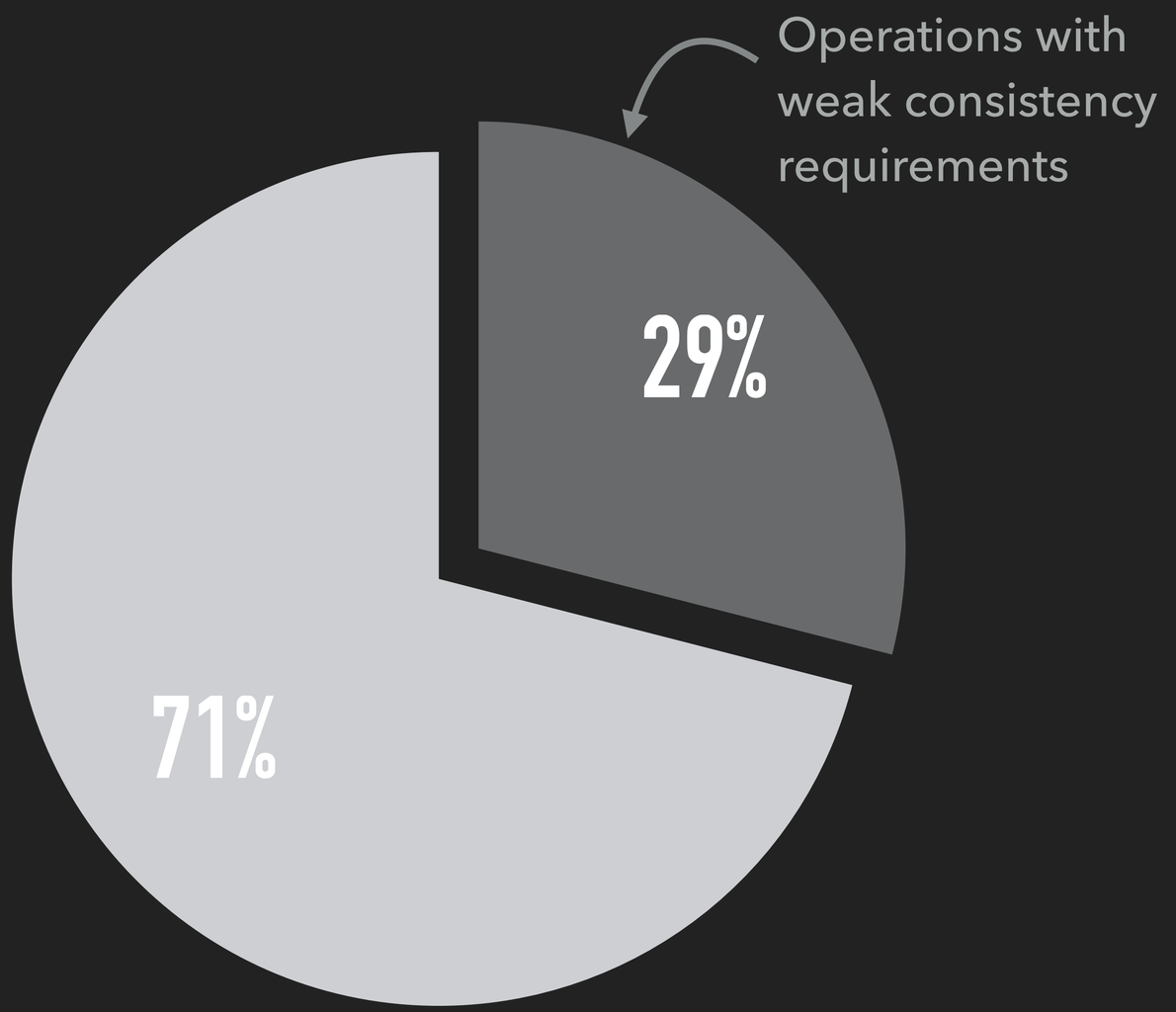
HOW FREQUENTLY ARE THEY USED? (1)

HOW FREQUENTLY ARE THEY USED? (1)

- Analyzed 7 benchmark applications

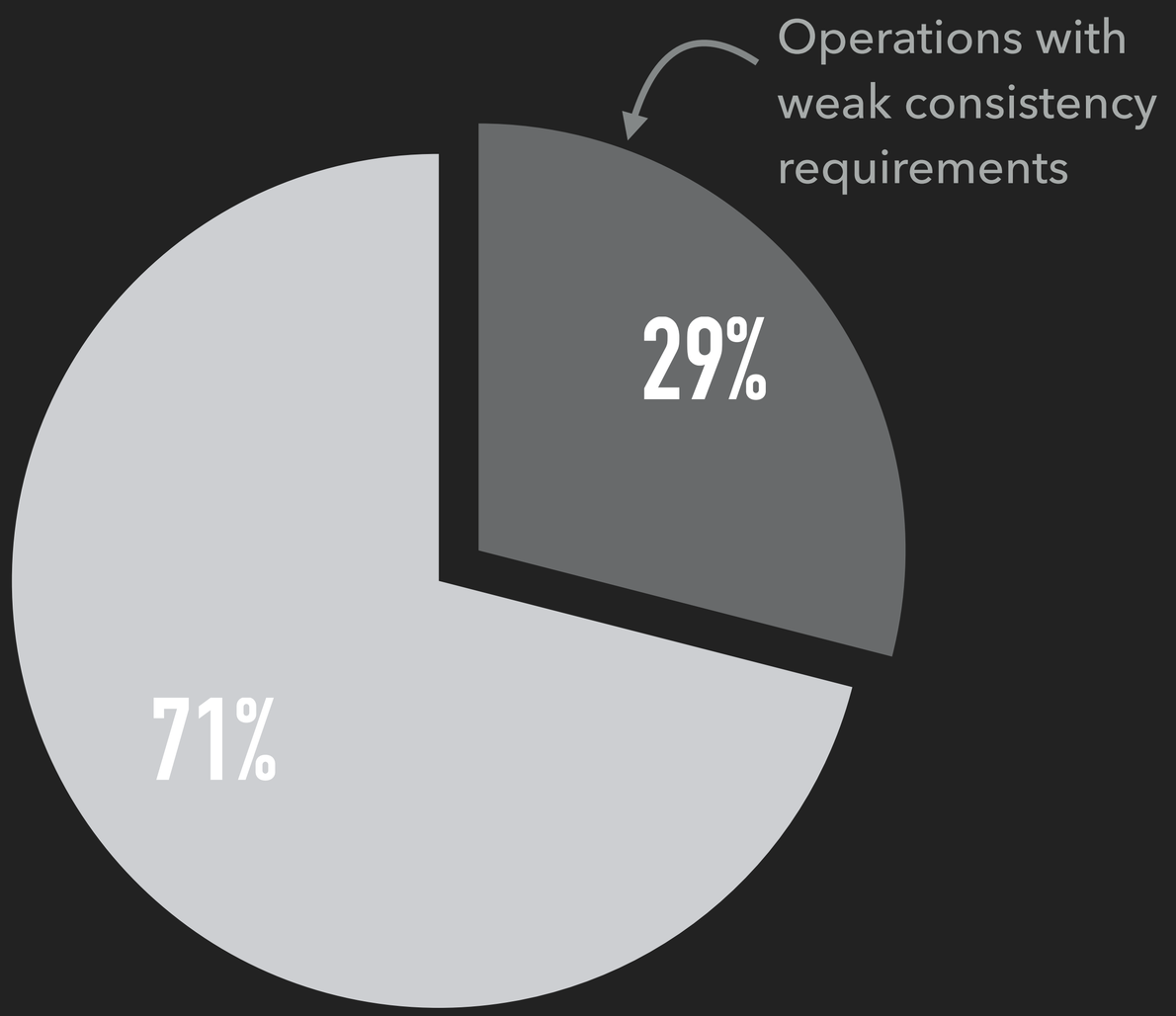
HOW FREQUENTLY ARE THEY USED? (1)

- Analyzed 7 benchmark applications
- All applications had operations that required *some* form of weak consistency guarantee



HOW FREQUENTLY ARE THEY USED? (1)

- Analyzed 7 benchmark applications
- All applications had operations that required *some* form of weak consistency guarantee
- Due to the lack of available underlying store implementations, all operations were originally mapped to CC



HOW FREQUENT ARE THEY USED? (2)

HOW FREQUENT ARE THEY USED? (2)

- *All* examined benchmark applications can be shown to require weaker forms of guarantees than CC

HOW FREQUENT ARE THEY USED? (2)

- *All* examined benchmark applications can be shown to require weaker forms of guarantees than CC
- The requirements are expressible as a logical combination of known weak consistency levels

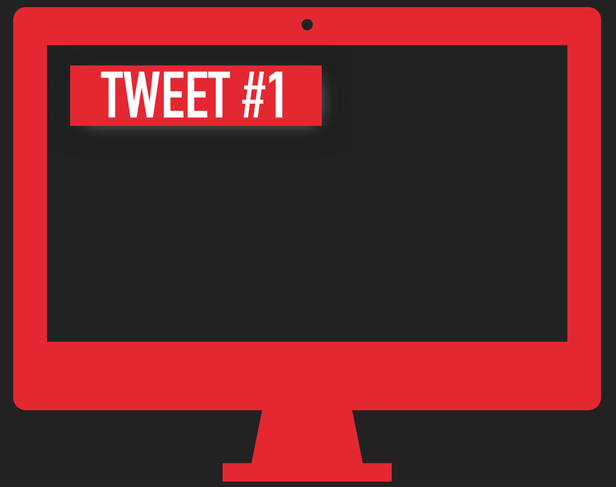
Benchmark	Consistency
Counter	MR
DynamoDB	RMW
Online Store	RMW
Bankaccount	2VIS ∧ RMW
Shopping List	MW ∧ RMW
Microblog	MW and RMW
Rubis	RMW and RMW∧2VIS

ARE THEY REALLY THAT DIFFERENT?

- Over faulty networks the difference between fine-grained consistency requirements can be substantial
- Example: Read-My-Writes (RMW) and CC

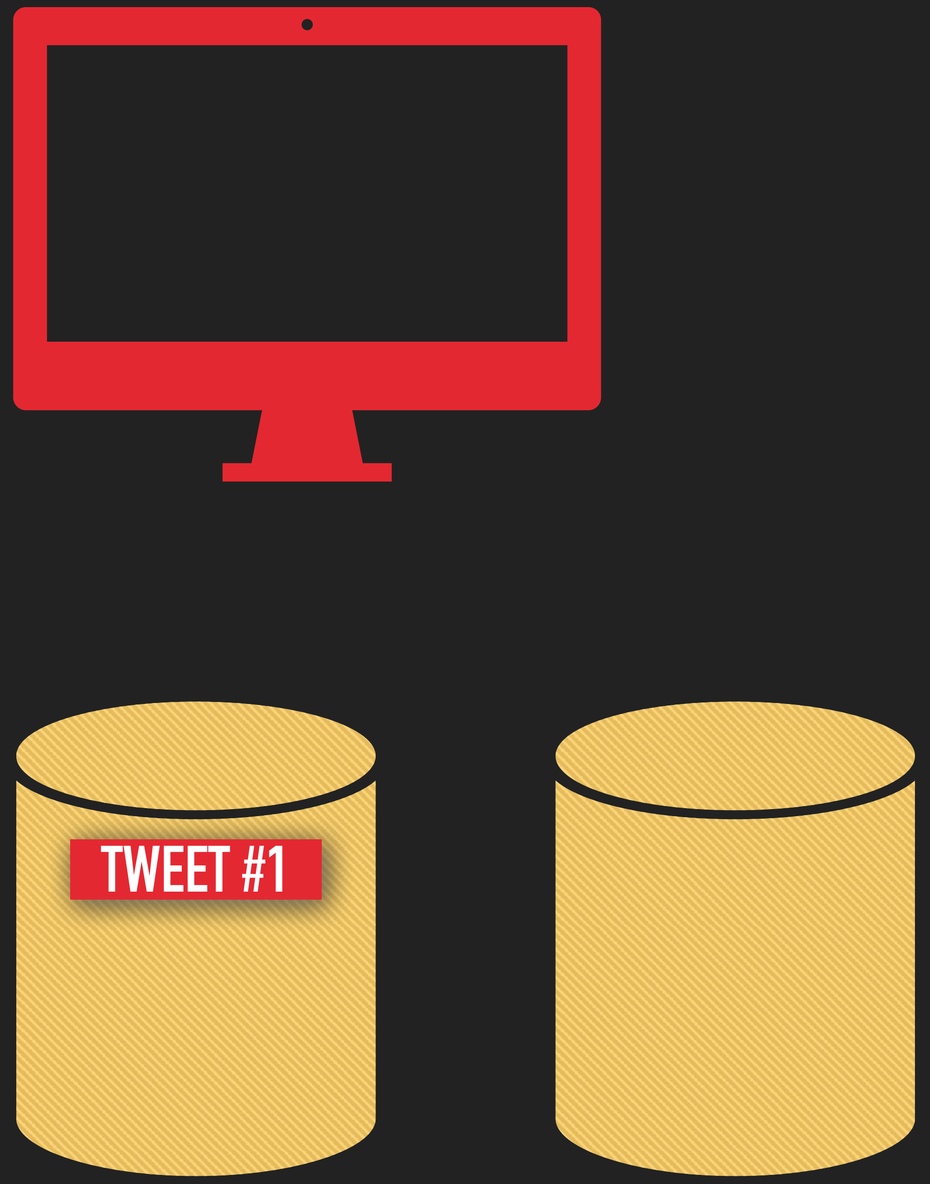
ARE THEY REALLY THAT DIFFERENT?

- Over faulty networks the difference between fine-grained consistency requirements can be substantial
- Example: Read-My-Writes (RMW) and CC



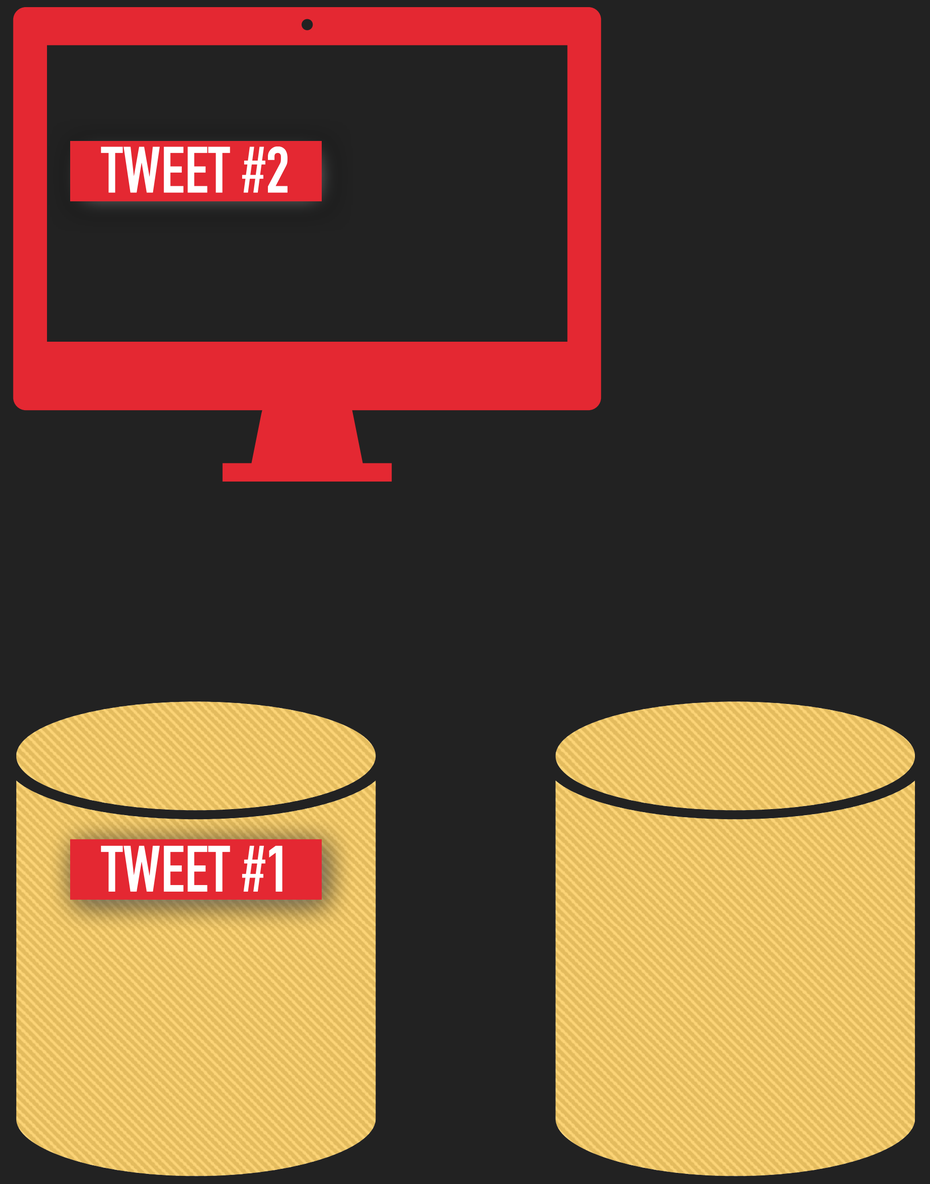
ARE THEY REALLY THAT DIFFERENT?

- Over faulty networks the difference between fine-grained consistency requirements can be substantial
- Example: Read-My-Writes (RMW) and CC



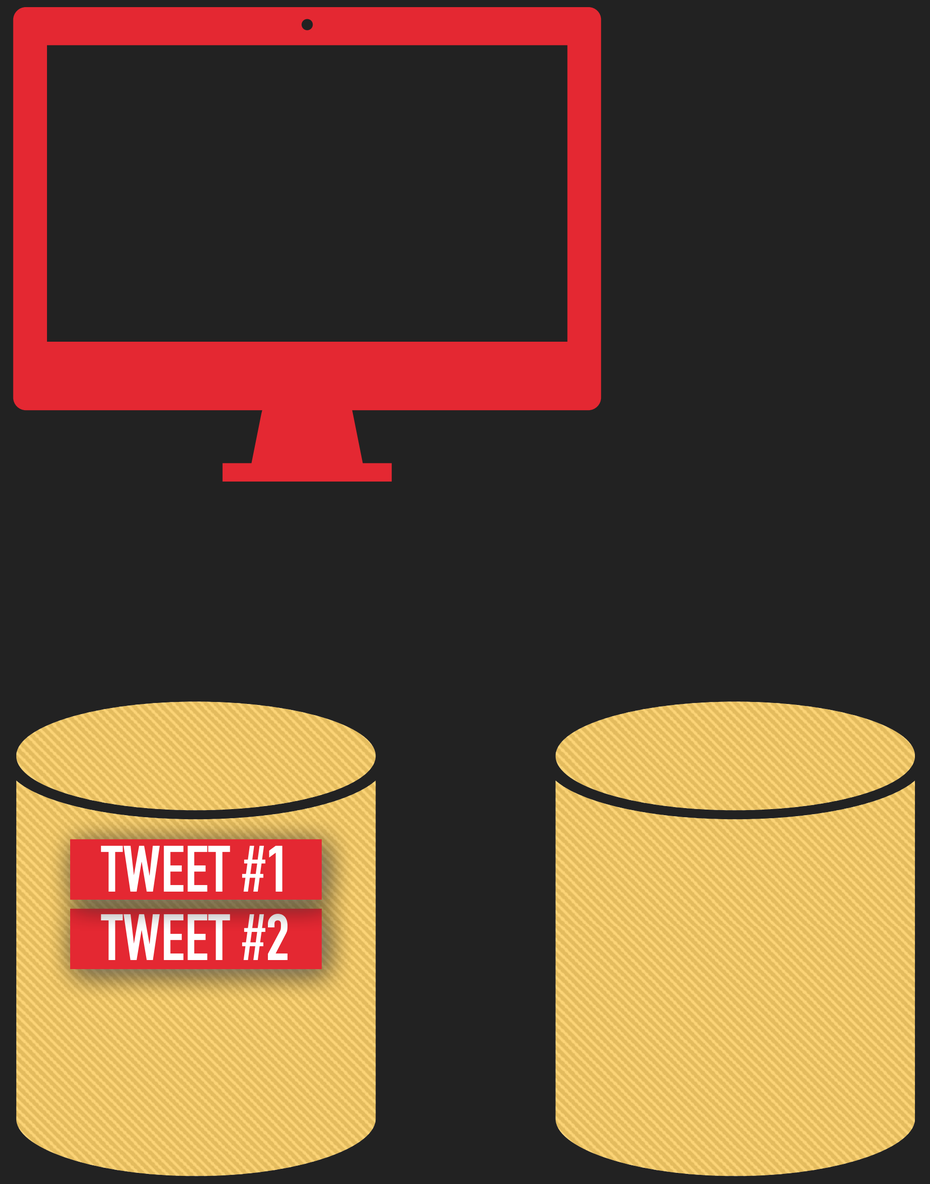
ARE THEY REALLY THAT DIFFERENT?

- Over faulty networks the difference between fine-grained consistency requirements can be substantial
- Example: Read-My-Writes (RMW) and CC



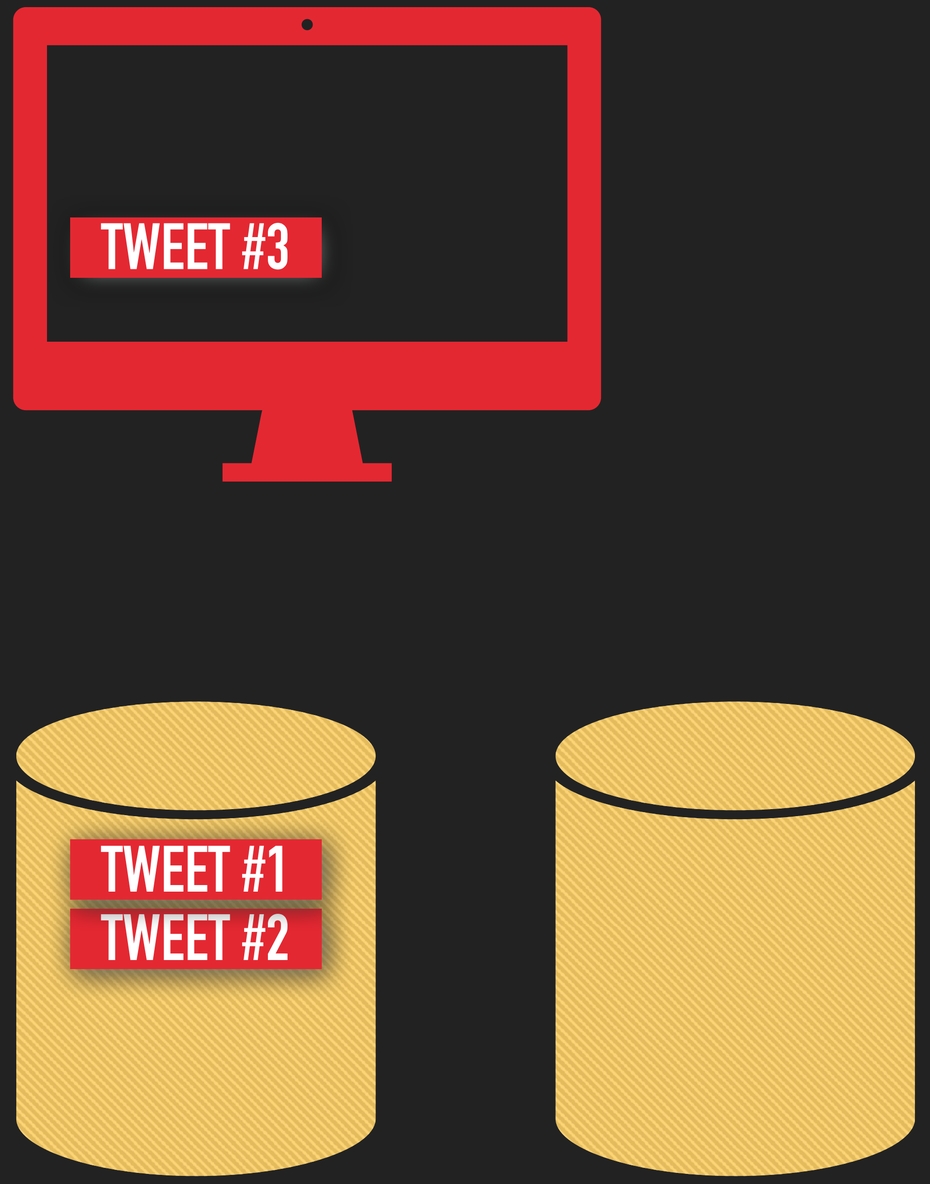
ARE THEY REALLY THAT DIFFERENT?

- Over faulty networks the difference between fine-grained consistency requirements can be substantial
- Example: Read-My-Writes (RMW) and CC



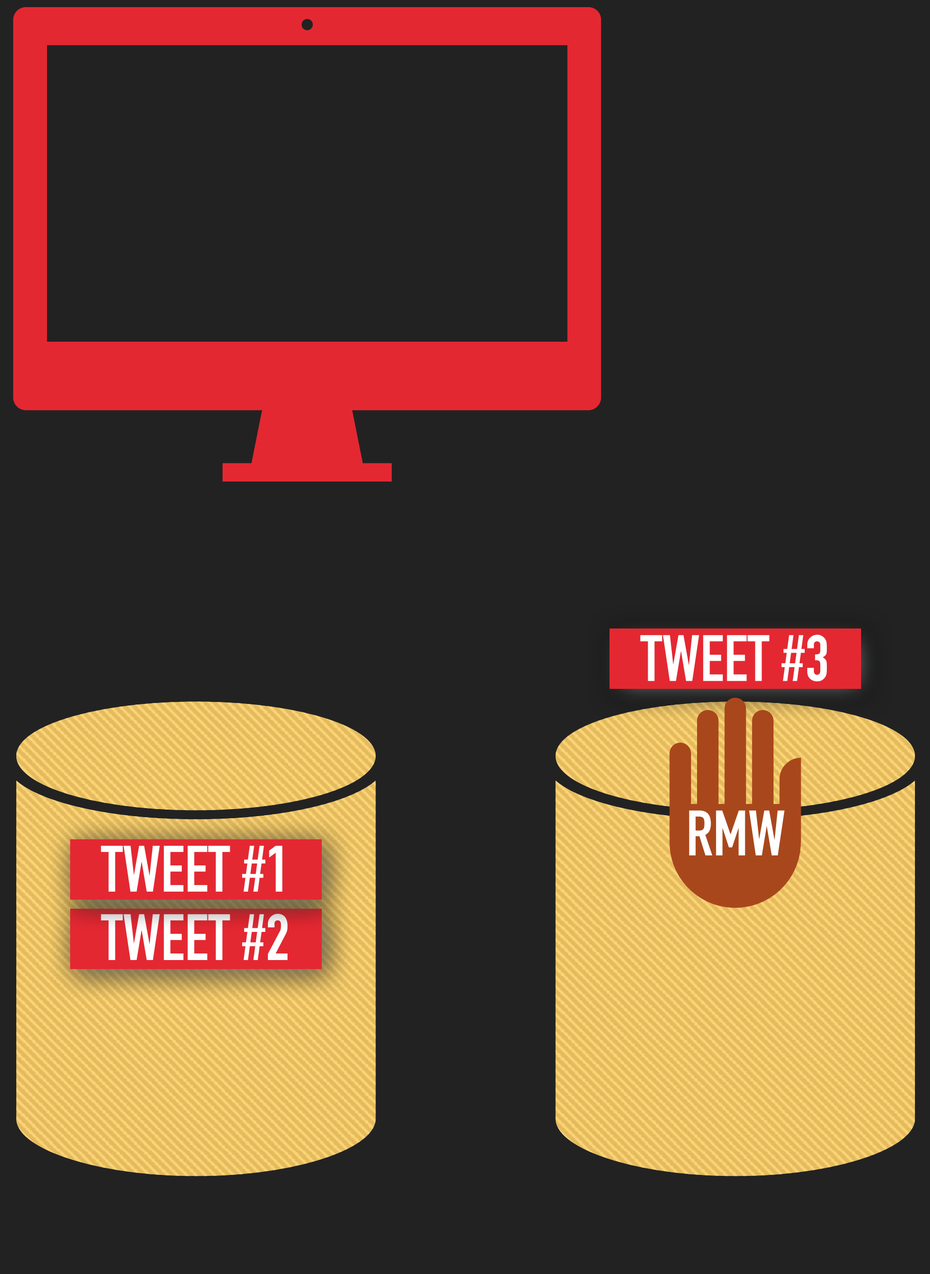
ARE THEY REALLY THAT DIFFERENT?

- Over faulty networks the difference between fine-grained consistency requirements can be substantial
- Example: Read-My-Writes (RMW) and CC



ARE THEY REALLY THAT DIFFERENT?

- Over faulty networks the difference between fine-grained consistency requirements can be substantial
- Example: Read-My-Writes (RMW) and CC

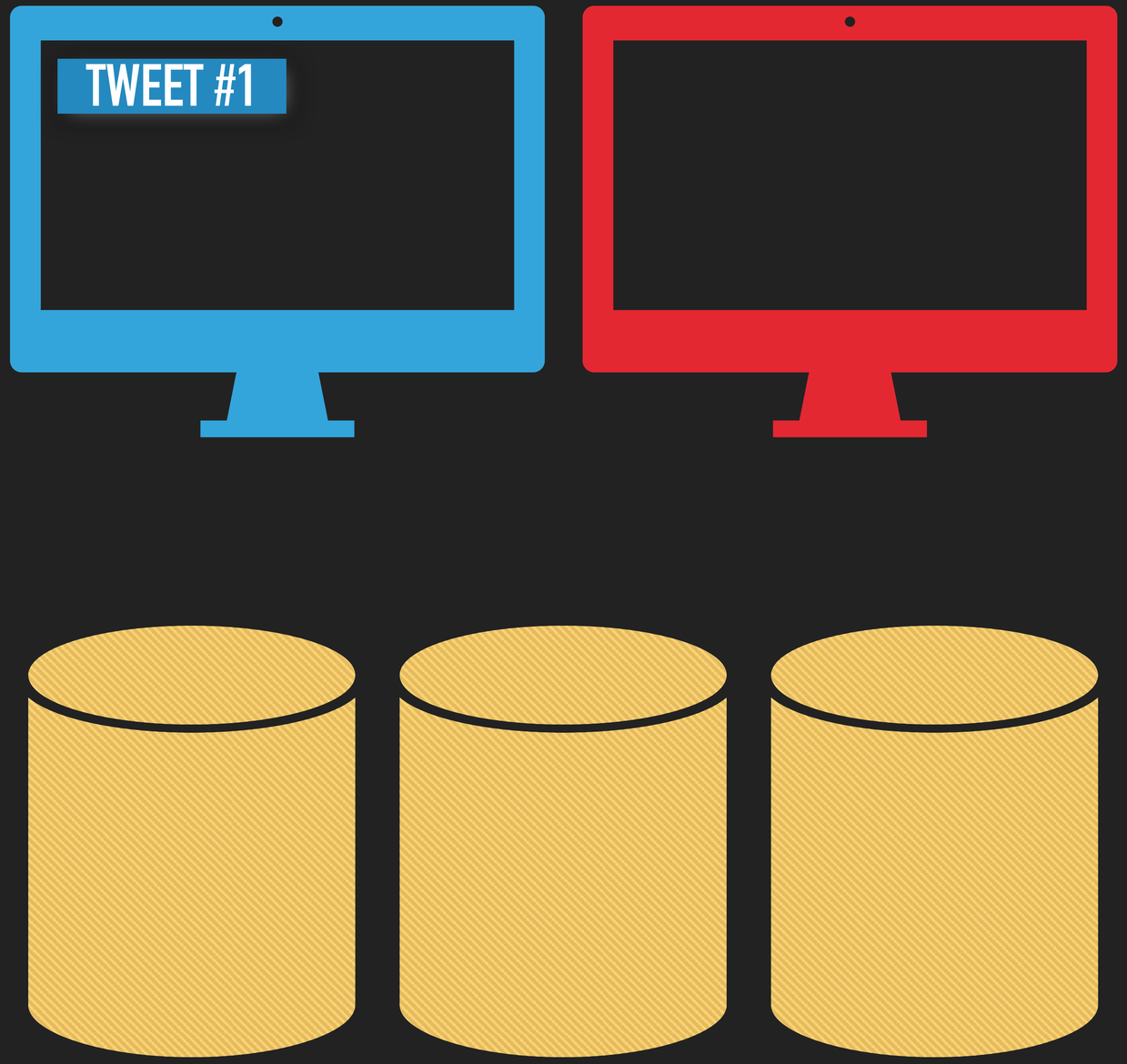


APPLICATION AND DATABASE CONSISTENCY MISMATCH

- Over faulty networks the difference between fine-grained consistency requirements can be substantial
- Example: Read-My-Writes (RMW) and CC
- CC incurs additional constraints
- Not only my previous writes, but everything visible to them should also be visible to me!

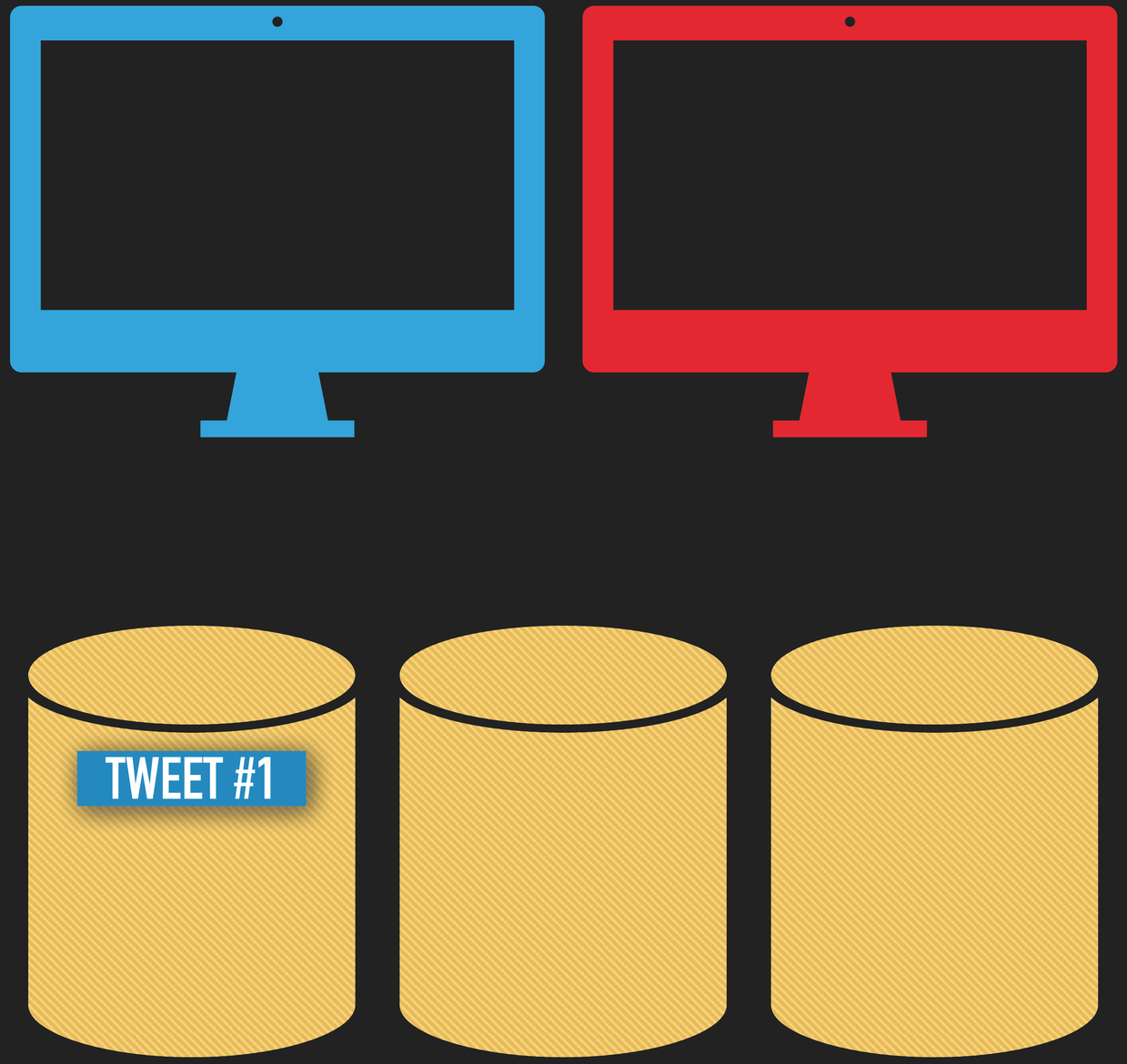
APPLICATION AND DATABASE CONSISTENCY MISMATCH

- Over faulty networks the difference between fine-grained consistency requirements can be substantial
- Example: Read-My-Writes (RMW) and CC
- CC incurs additional constraints
- Not only my previous writes, but everything visible to them should also be visible to me!



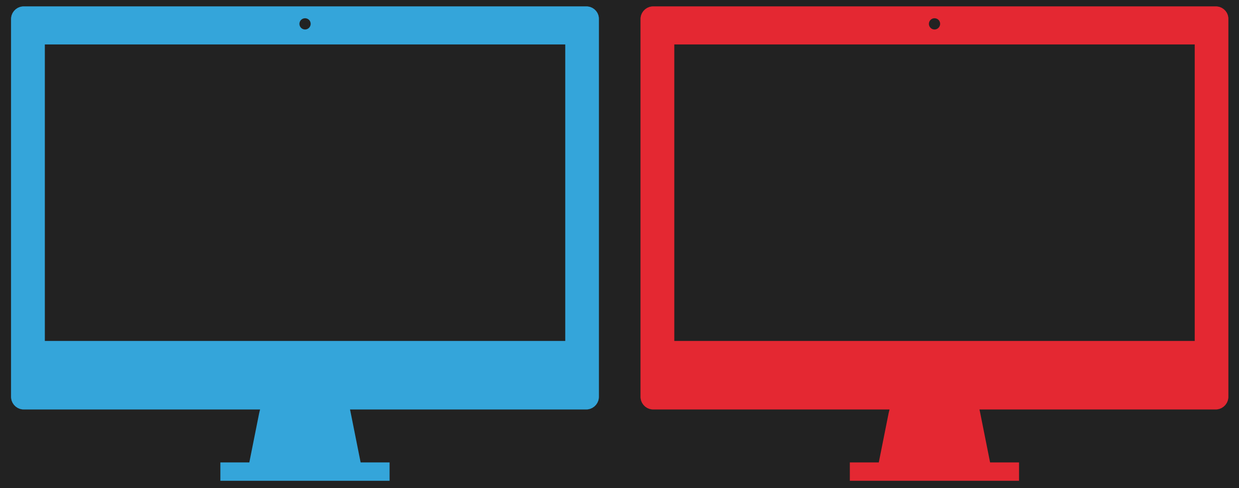
APPLICATION AND DATABASE CONSISTENCY MISMATCH

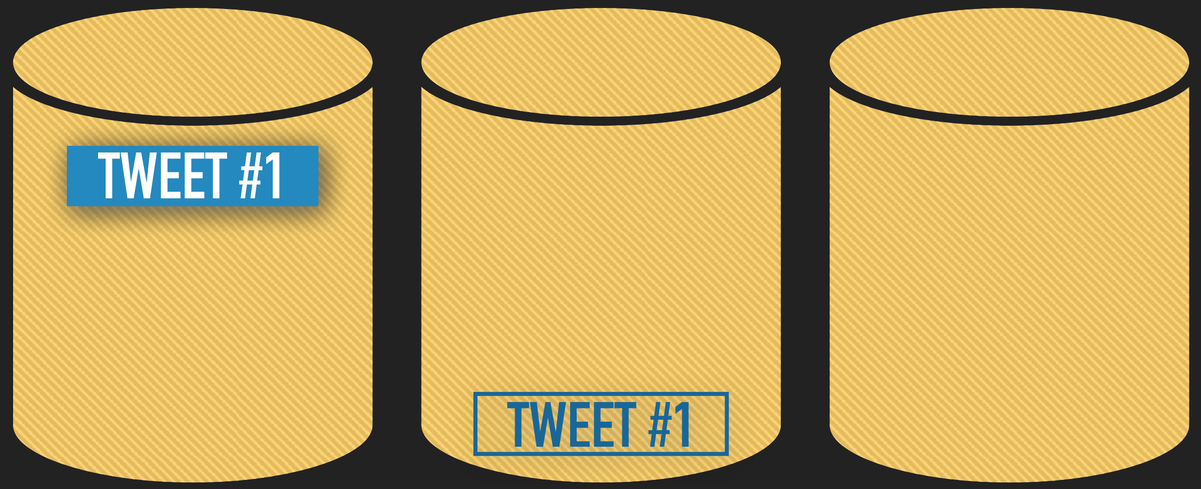
- Over faulty networks the difference between fine-grained consistency requirements can be substantial
- Example: Read-My-Writes (RMW) and CC
- CC incurs additional constraints
- Not only my previous writes, but everything visible to them should also be visible to me!



APPLICATION AND DATABASE CONSISTENCY MISMATCH

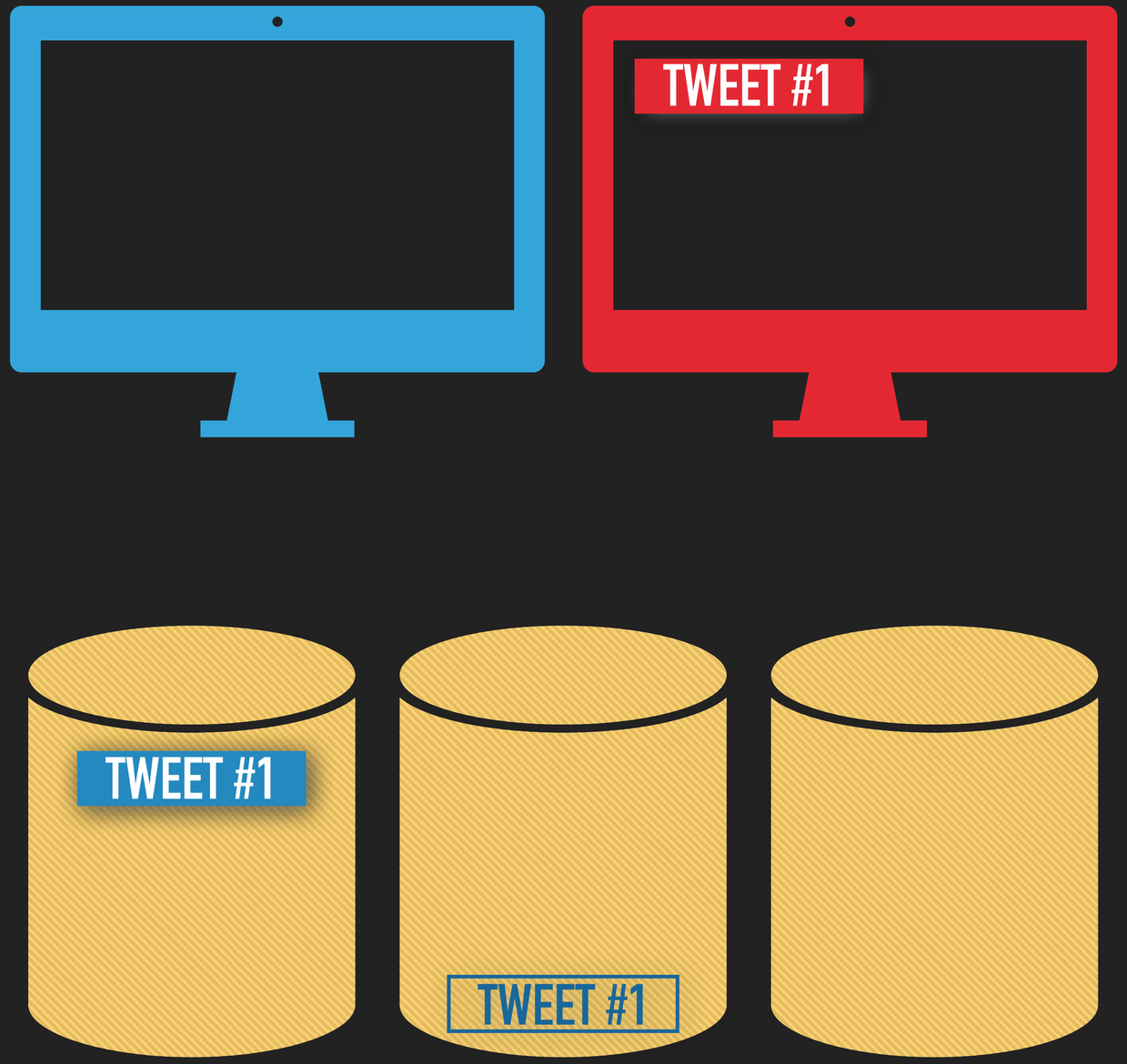
- Over faulty networks the difference between fine-grained consistency requirements can be substantial
- Example: Read-My-Writes (RMW) and CC
- CC incurs additional constraints
- Not only my previous writes, but everything visible to them should also be visible to me!





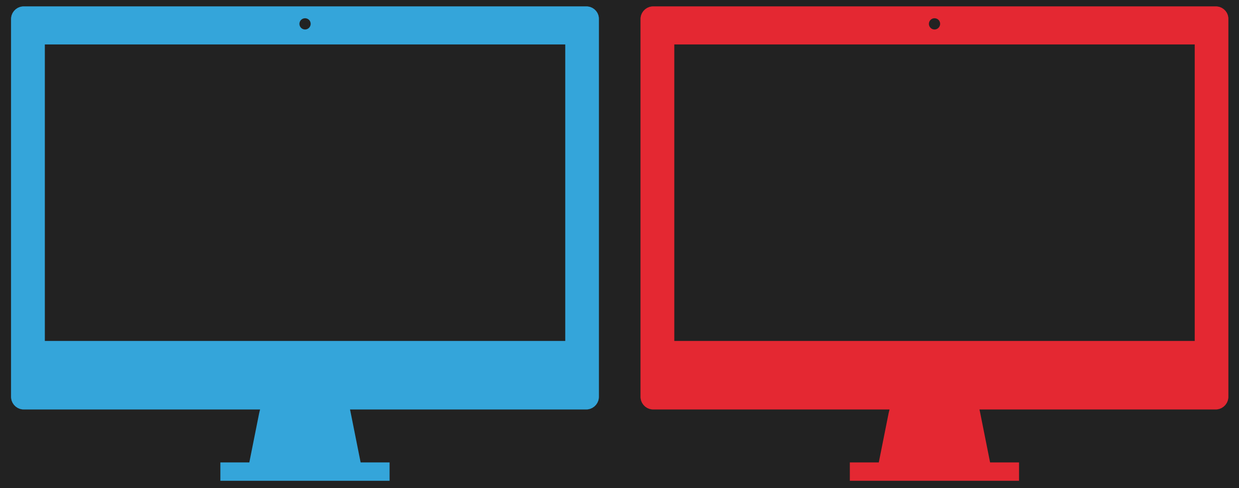
APPLICATION AND DATABASE CONSISTENCY MISMATCH

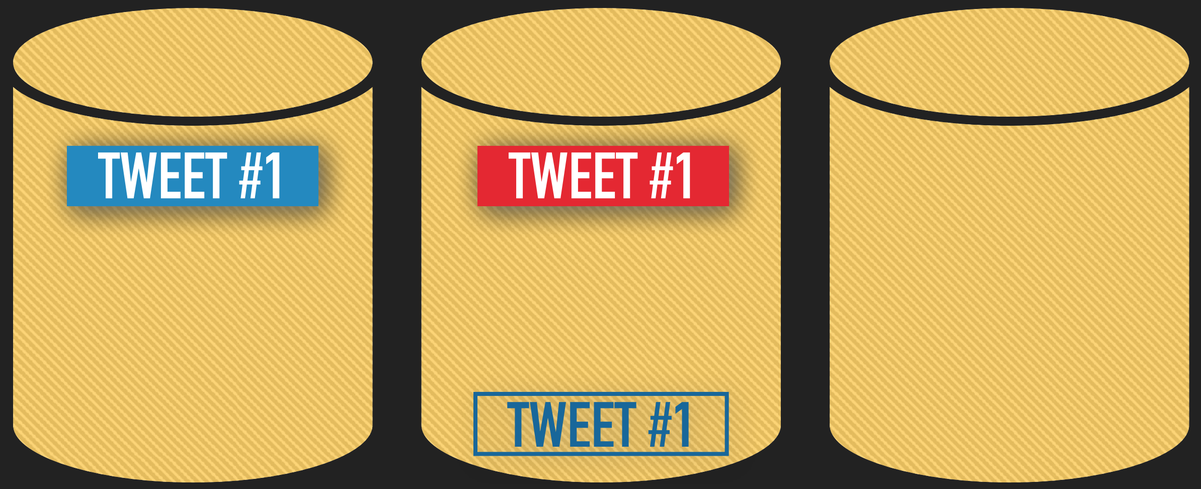
- Over faulty networks the difference between fine-grained consistency requirements can be substantial
- Example: Read-My-Writes (RMW) and CC
- CC incurs additional constraints
- Not only my previous writes, but everything visible to them should also be visible to me!



APPLICATION AND DATABASE CONSISTENCY MISMATCH

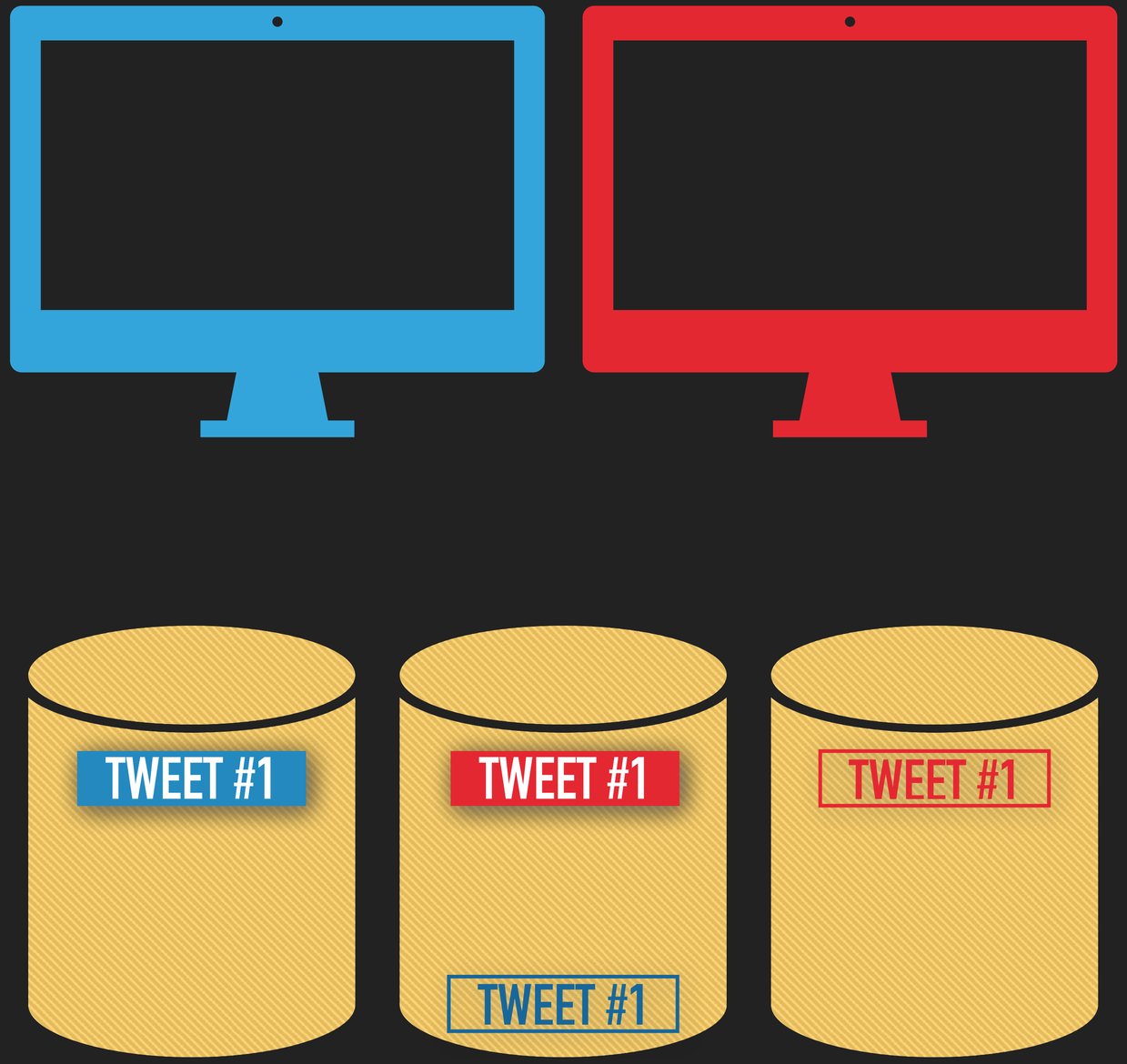
- Over faulty networks the difference between fine-grained consistency requirements can be substantial
- Example: Read-My-Writes (RMW) and CC
- CC incurs additional constraints
- Not only my previous writes, but everything visible to them should also be visible to me!





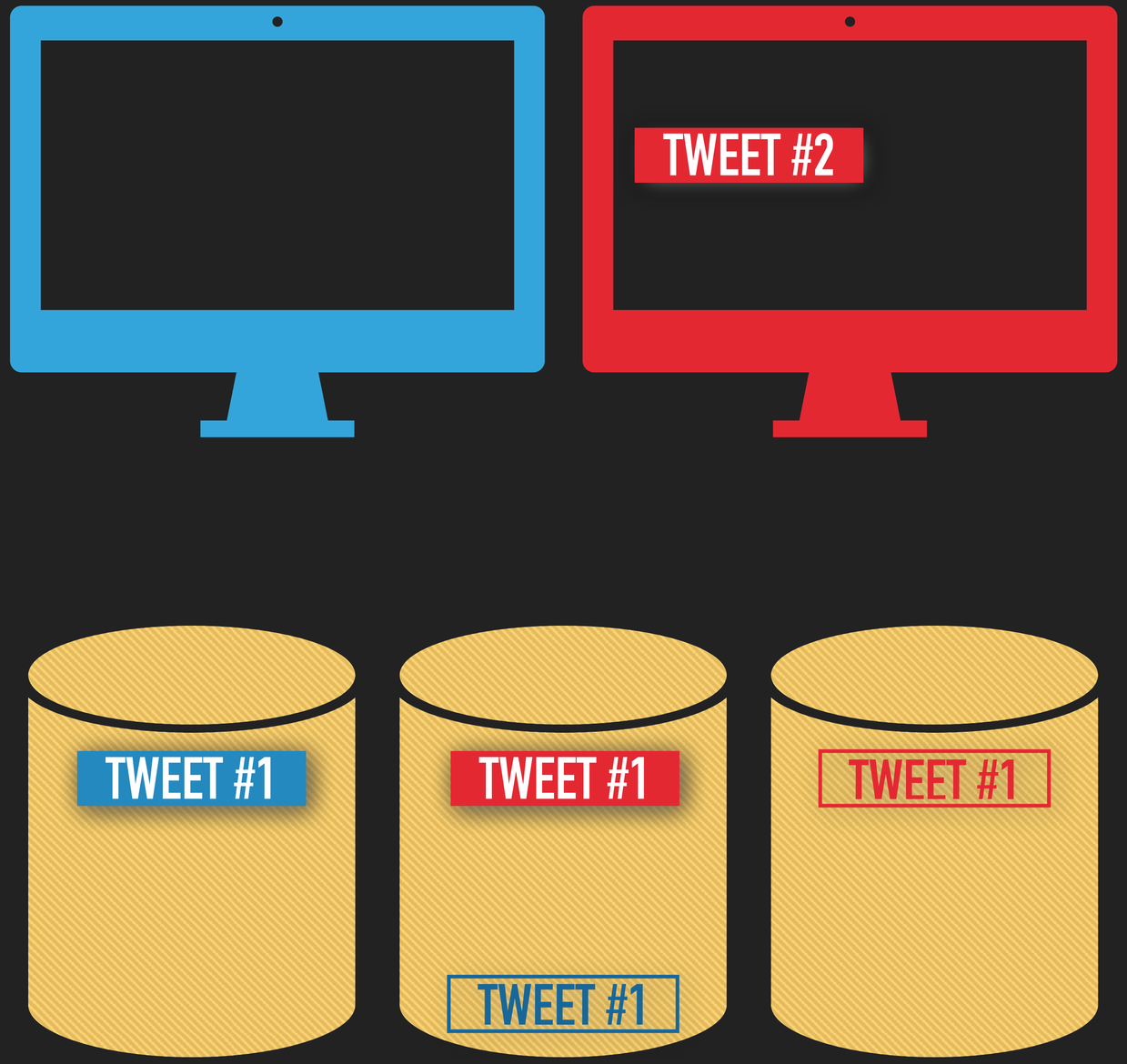
APPLICATION AND DATABASE CONSISTENCY MISMATCH

- Over faulty networks the difference between fine-grained consistency requirements can be substantial
- Example: Read-My-Writes (RMW) and CC
- CC incurs additional constraints
- Not only my previous writes, but everything visible to them should also be visible to me!



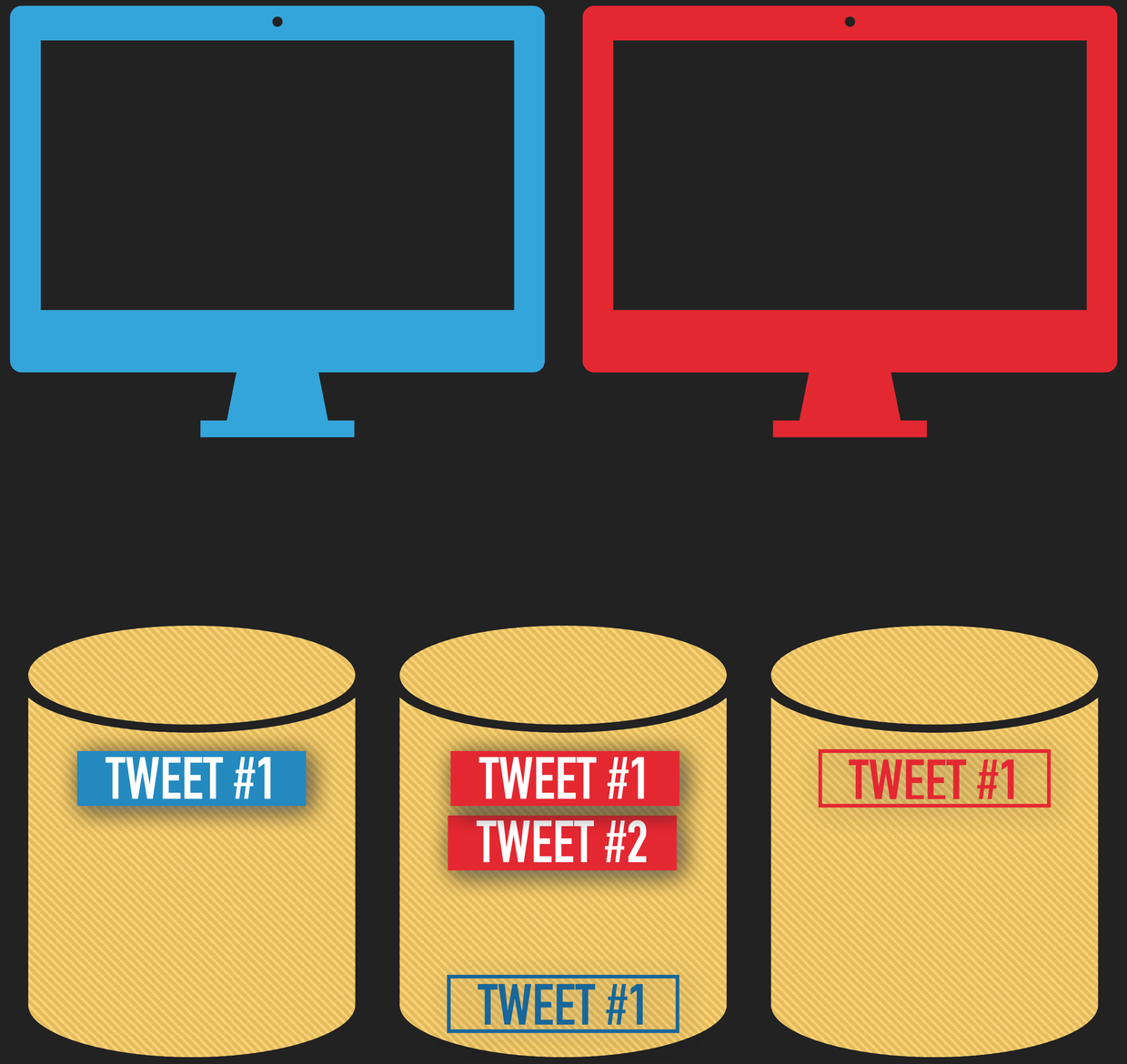
APPLICATION AND DATABASE CONSISTENCY MISMATCH

- Over faulty networks the difference between fine-grained consistency requirements can be substantial
- Example: Read-My-Writes (RMW) and CC
- CC incurs additional constraints
- Not only my previous writes, but everything visible to them should also be visible to me!



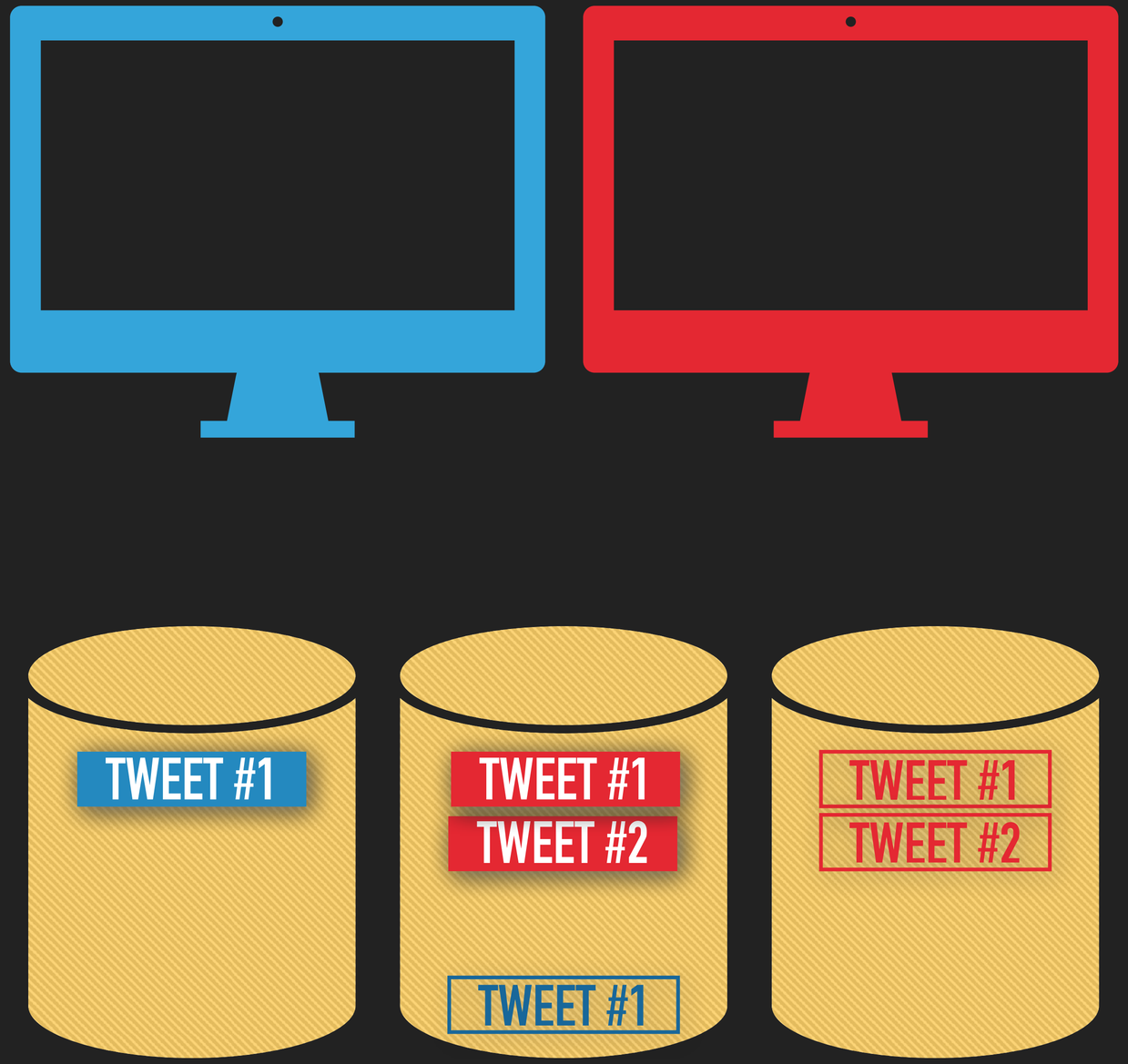
APPLICATION AND DATABASE CONSISTENCY MISMATCH

- Over faulty networks the difference between fine-grained consistency requirements can be substantial
- Example: Read-My-Writes (RMW) and CC
- CC incurs additional constraints
- Not only my previous writes, but everything visible to them should also be visible to me!



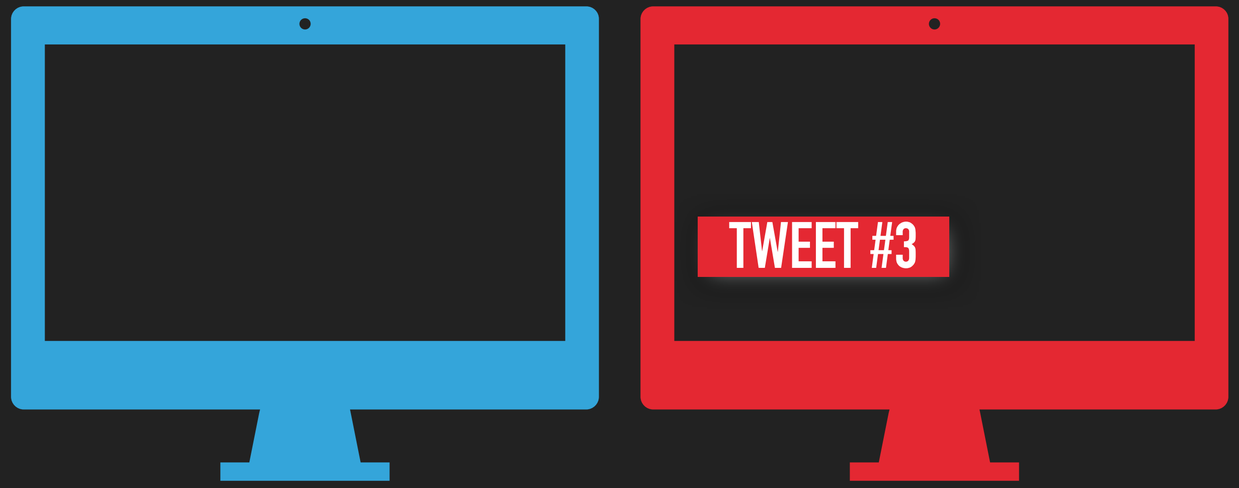
APPLICATION AND DATABASE CONSISTENCY MISMATCH

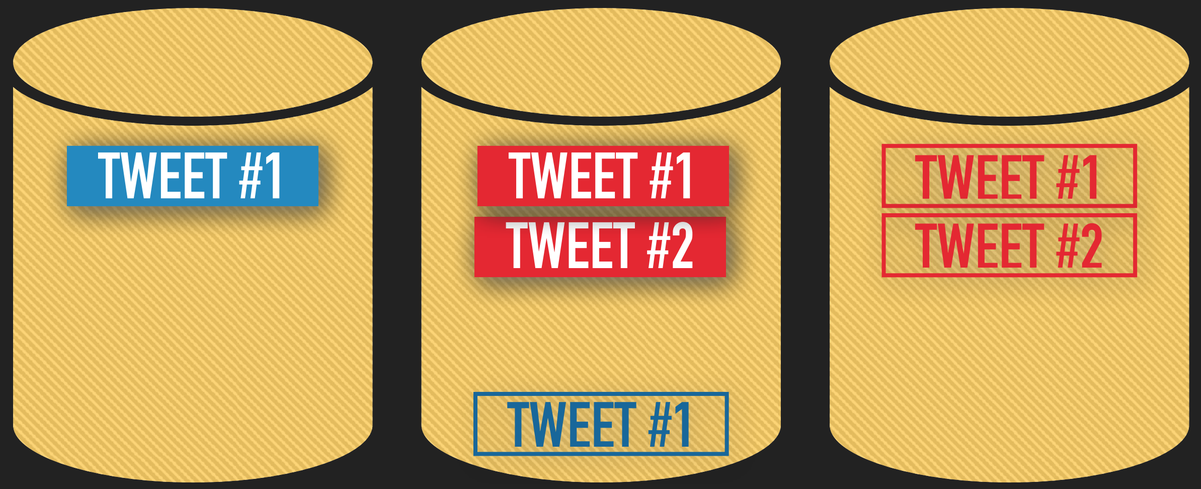
- Over faulty networks the difference between fine-grained consistency requirements can be substantial
- Example: Read-My-Writes (RMW) and CC
- CC incurs additional constraints
- Not only my previous writes, but everything visible to them should also be visible to me!



APPLICATION AND DATABASE CONSISTENCY MISMATCH

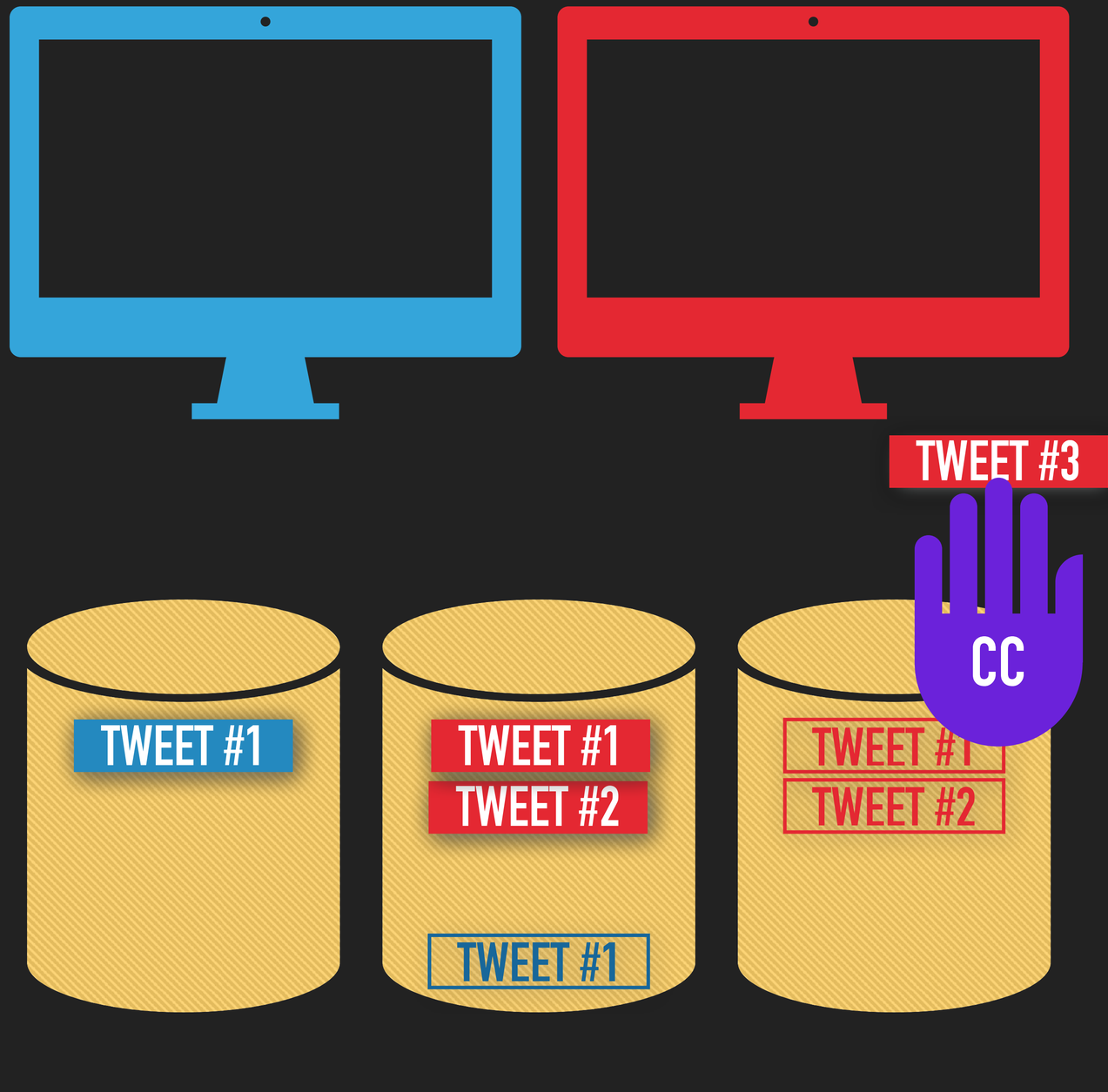
- Over faulty networks the difference between fine-grained consistency requirements can be substantial
- Example: Read-My-Writes (RMW) and CC
- CC incurs additional constraints
- Not only my previous writes, but everything visible to them should also be visible to me!





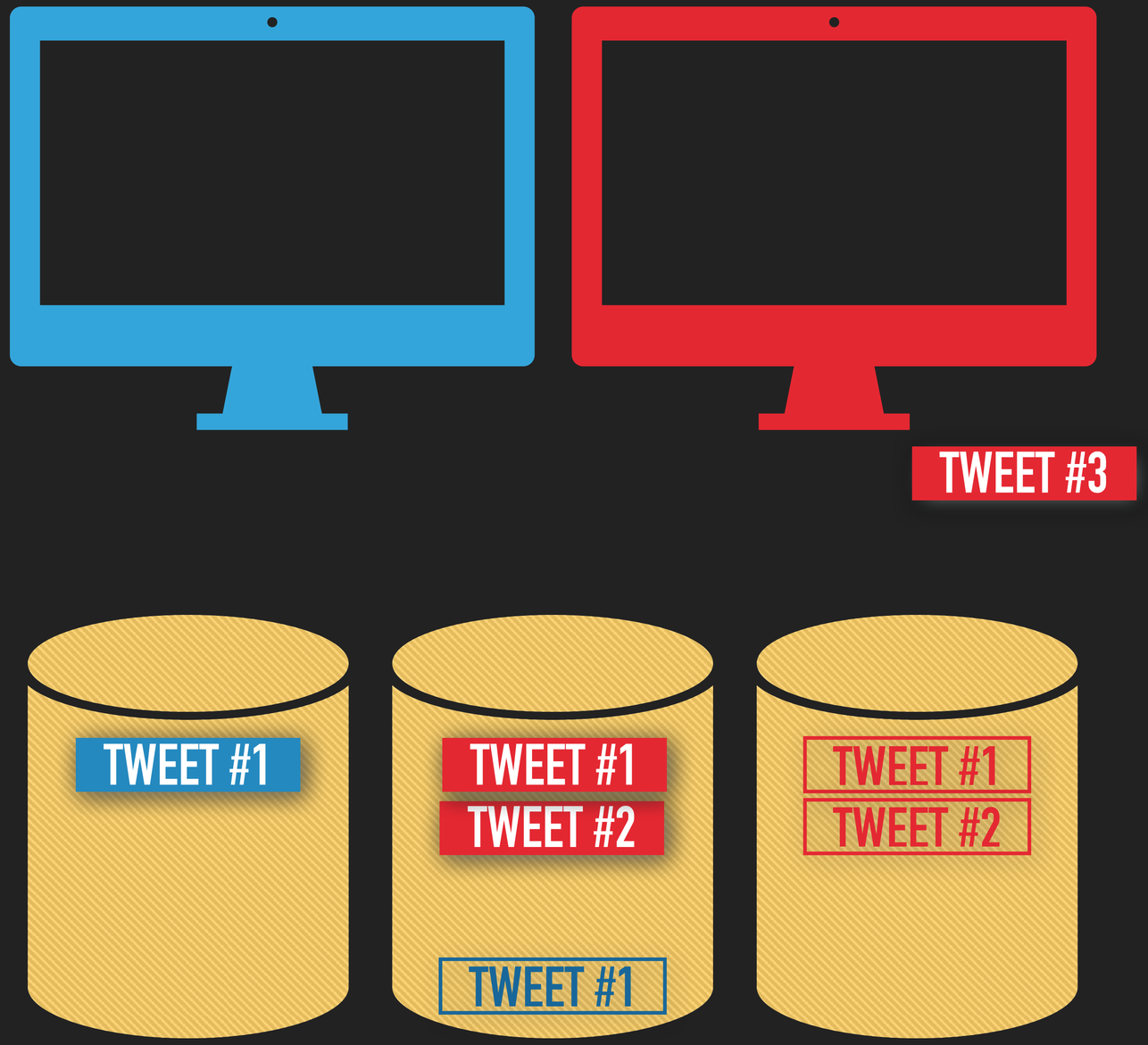
APPLICATION AND DATABASE CONSISTENCY MISMATCH

- Over faulty networks the difference between fine-grained consistency requirements can be substantial
- Example: Read-My-Writes (RMW) and CC
- CC incurs additional constraints
- Not only my previous writes, but everything visible to them should also be visible to me!



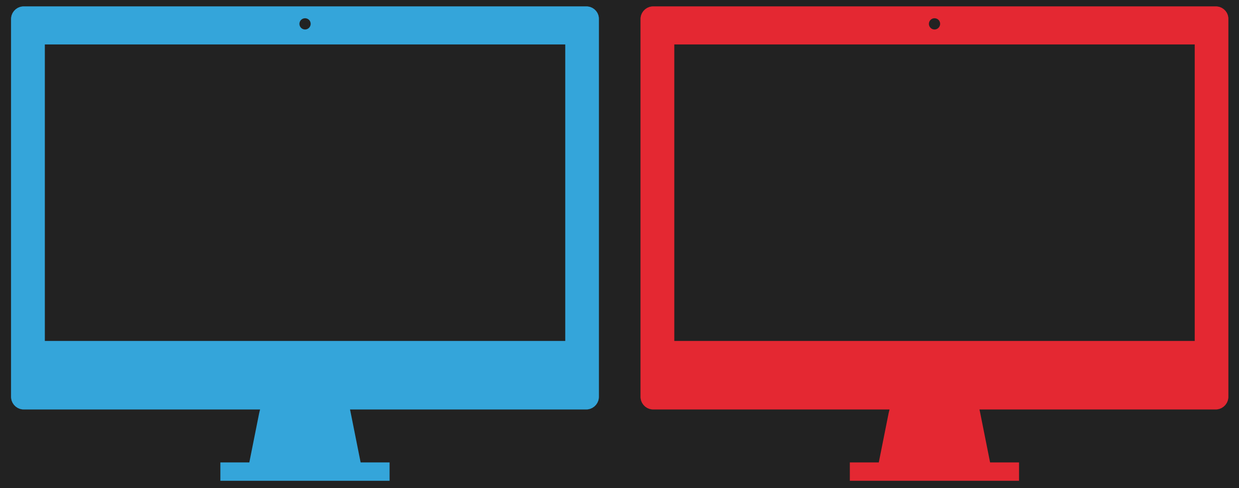
APPLICATION AND DATABASE CONSISTENCY MISMATCH

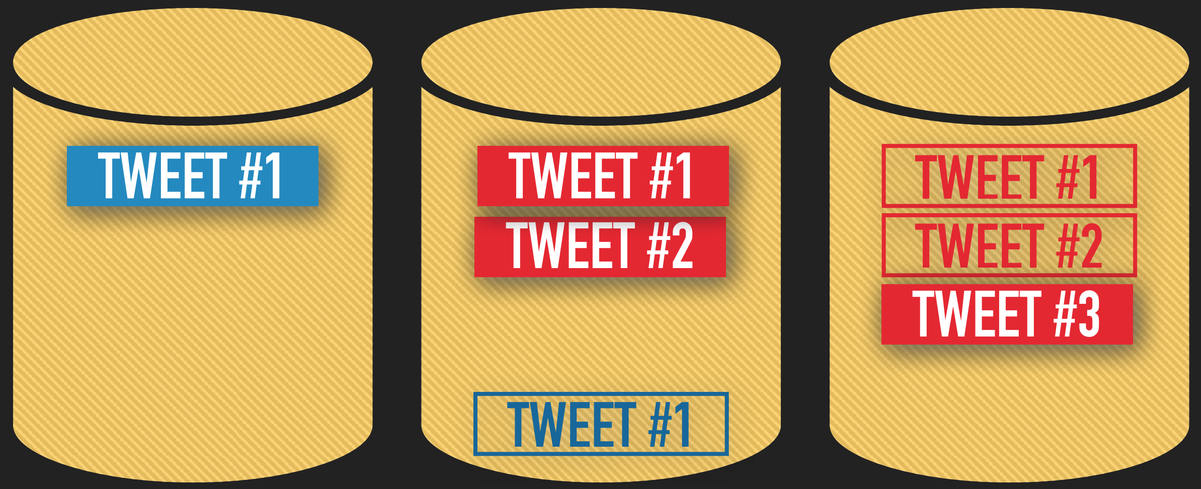
- Over faulty networks the difference between fine-grained consistency requirements can be substantial
- Example: Read-My-Writes (RMW) and CC
- CC incurs additional constraints
- Not only my previous writes, but everything visible to them should also be visible to me!



APPLICATION AND DATABASE CONSISTENCY MISMATCH

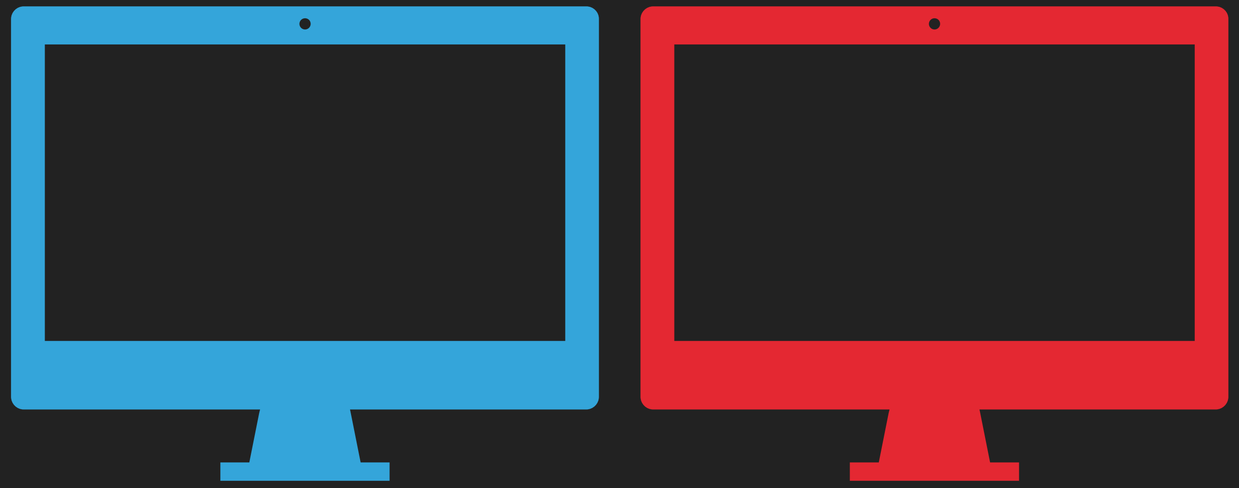
- Over faulty networks the difference between fine-grained consistency requirements can be substantial
- Example: Read-My-Writes (RMW) and CC
- CC incurs additional constraints
- Not only my previous writes, but everything visible to them should also be visible to me!

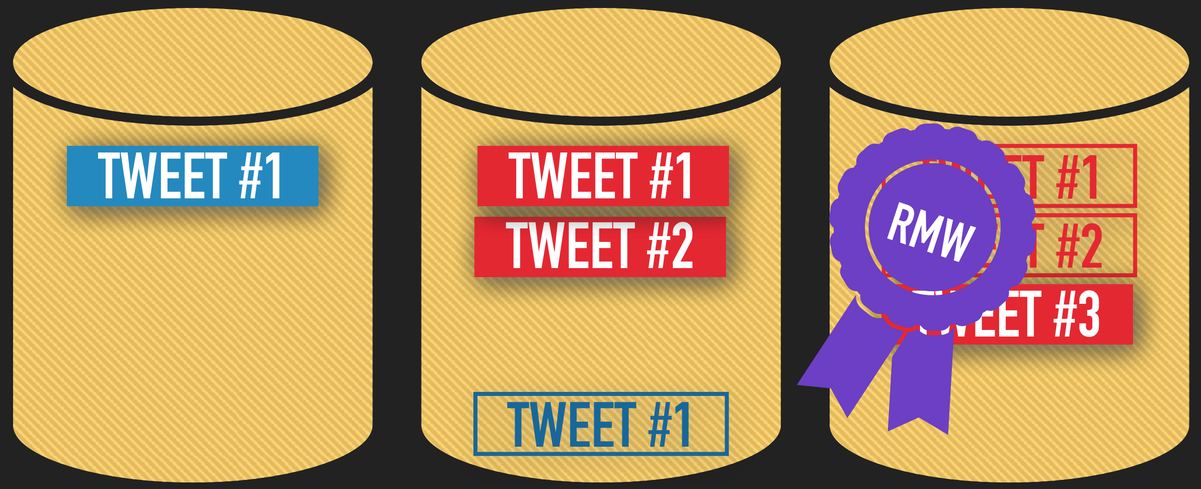




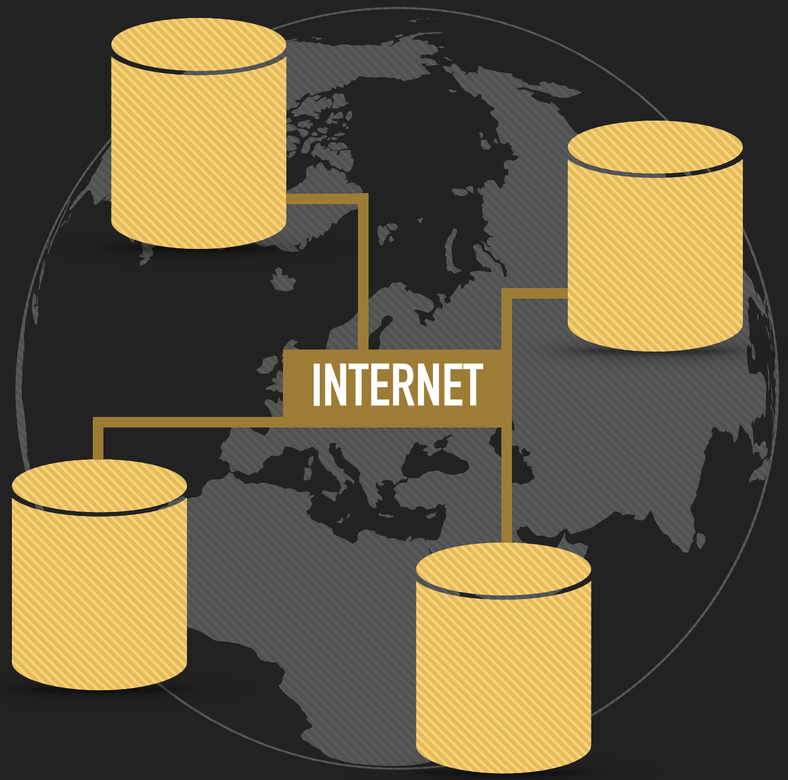
APPLICATION AND DATABASE CONSISTENCY MISMATCH

- Over faulty networks the difference between fine-grained consistency requirements can be substantial
- Example: Read-My-Writes (RMW) and CC
- CC incurs additional constraints
- Not only my previous writes, but everything visible to them should also be visible to me!





EVENTUALLY CONSISTENT DATA STORES

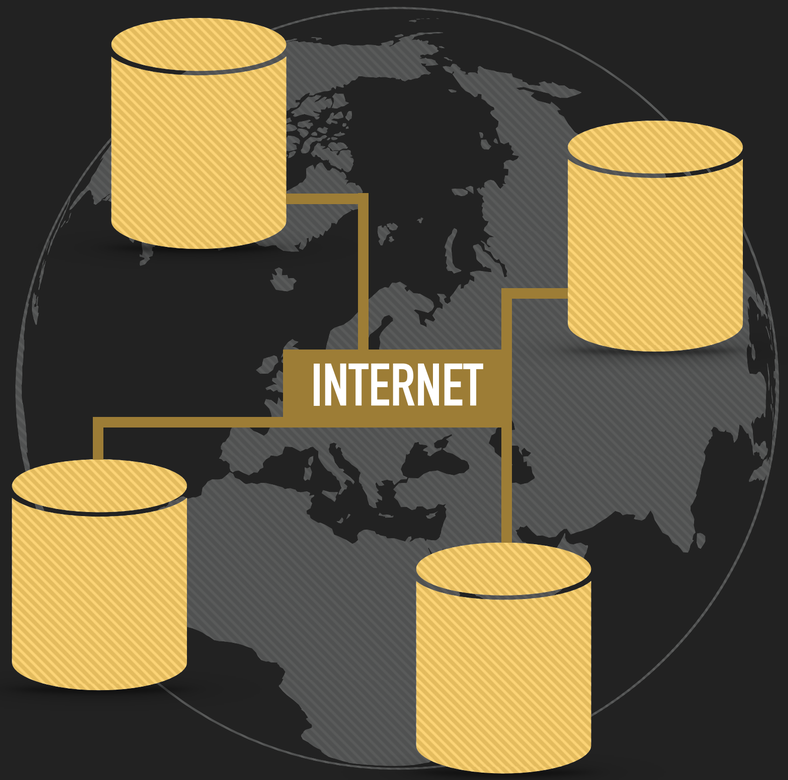


riak

mongoDB

cassandra

EVENTUALLY CONSISTENT DATA STORES



riak

mongoDB

cassandra

ADDITIONAL CONSISTENCY GUARANTEES

Startup #1

- BASIC EC
- STRONG

Project #2

- BASIC EC
- CAUSAL
- STRONG

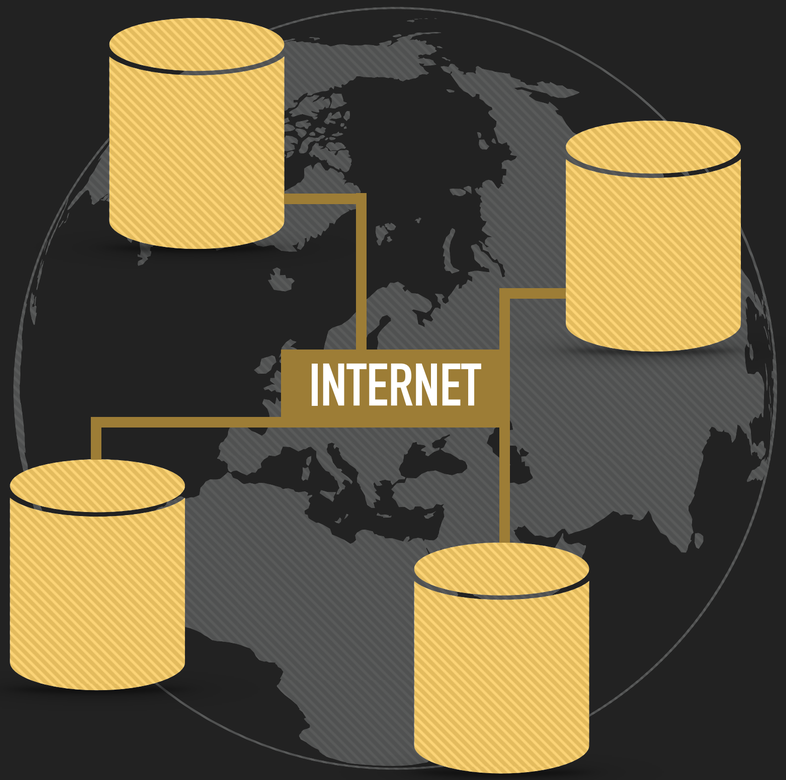
Company #3

- MONOTONIC WRITES
- MONOTONIC READS
- CAUSAL CONSISTENCY

Paper #4

- BASIC EC
- MONOTONIC WRITES
- MONOTONIC READS
- READ-MY-WRITES
- WRITES FOLLOW READS

EVENTUALLY CONSISTENT DATA STORES



riak

mongoDB

cassandra

ADDITIONAL CONSISTENCY GUARANTEES

Startup #1

- BASIC EC
- STRONG

Project #2

- BASIC EC
- CAUSAL
- STRONG

Company #3

- MONOTONIC WRITES
- MONOTONIC READS
- CAUSAL CONSISTENCY

Paper #4

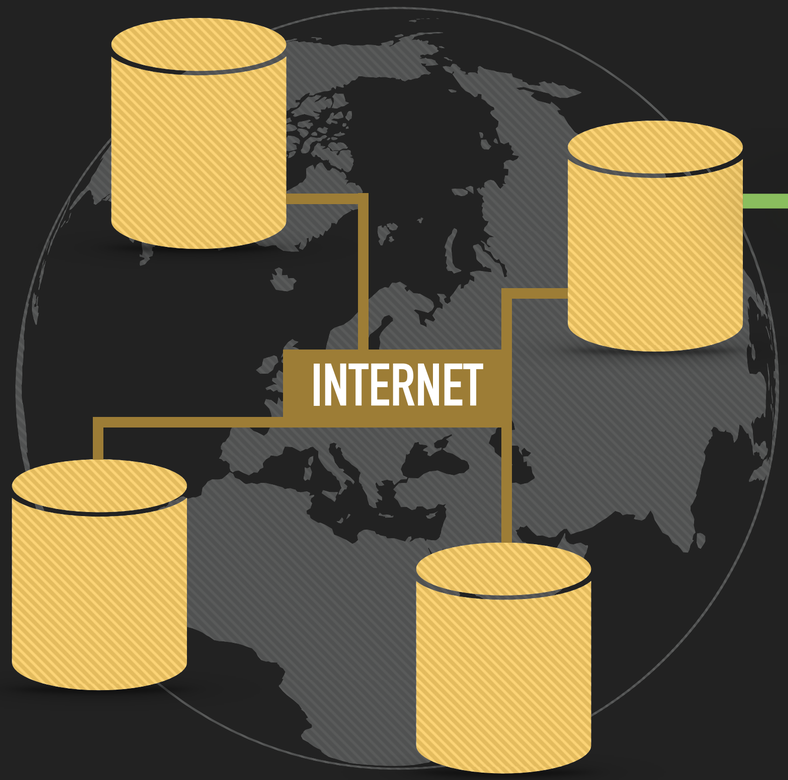
- BASIC EC
- MONOTONIC WRITES
- MONOTONIC READS
- READ-MY-WRITES
- WRITES FOLLOW READS

SYSTEM DEVELOPER

EVENTUALLY CONSISTENT DATA STORES

AUTOMATIC ENFORCEMENT OF FINE-GRAINED CONSISTENCY REQUIREMENTS

SYSTEM DEVELOPER



SYNCOPE

- Light weight run-time system on top of Cassandra
- Multi-consistent shim layer which is specifically maintained for each operation
- Automatic enforcement of consistency requirements per operation
- Provably optimal and correct

Generic programming model

Per operation consistency declaration

riak

mongoDB

cassandra

IT’S ALL ABOUT ENFORCING VISIBILITY

IT'S ALL ABOUT ENFORCING VISIBILITY

- There are two fundamental relations between such effects: **vis** and **so**

IT'S ALL ABOUT ENFORCING VISIBILITY

- There are two fundamental relations between such effects: **vis** and **so**
- **vis** relates an effect to all other effects present at the local replica at the time of its creation

IT'S ALL ABOUT ENFORCING VISIBILITY

- There are two fundamental relations between such effects: **vis** and **so**
- **vis** relates an effect to all other effects present at the local replica at the time of its creation
- **so** relates operations from the same client session

IT'S ALL ABOUT ENFORCING VISIBILITY

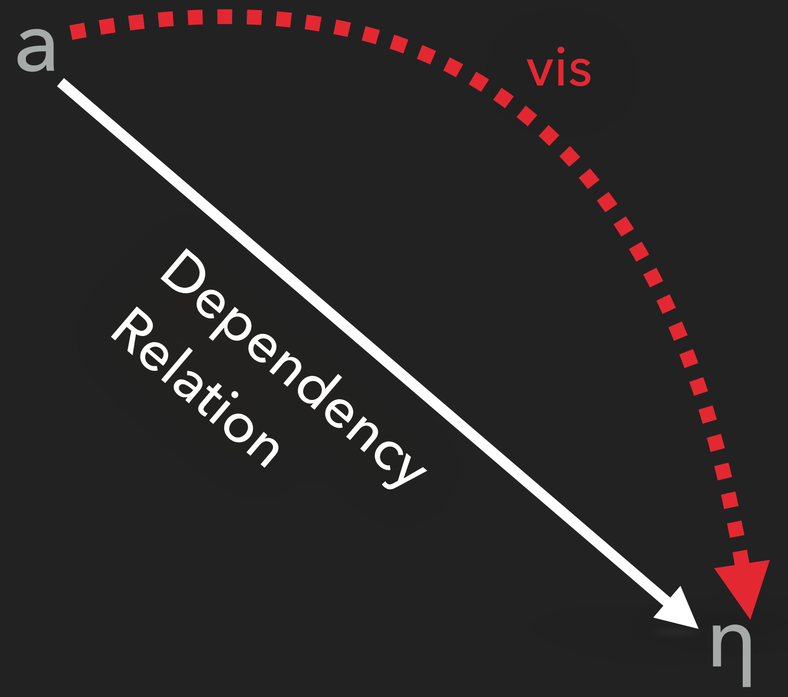
- There are two fundamental relations between such effects: **vis** and **so**
- **vis** relates an effect to all other effects present at the local replica at the time of its creation
- **so** relates operations from the same client session
- **vis** might be altered at the executing replica by changing the visible snapshot of the system to each operation

IT'S ALL ABOUT ENFORCING VISIBILITY

- There are two fundamental relations between such effects: **vis** and **so**
- **vis** relates an effect to all other effects present at the local replica at the time of its creation
- **so** relates operations from the same client session
- **vis** might be altered at the executing replica by changing the visible snapshot of the system to each operation
- **so** is pre-determined by the clients

IT'S ALL ABOUT ENFORCING VISIBILITY

- There are two fundamental relations between such effects: **vis** and **so**
- **vis** relates an effect to all other effects present at the local replica at the time of its creation
- **so** relates operations from the same client session
- **vis** might be altered at the executing replica by changing the visible snapshot of the system to each operation
- **so** is pre-determined by the clients
- The difference between all weak consistency guarantees is on how they **enforce** visibility relations between effects



SPECIFICATION LANGUAGE: DEFINITION

SPECIFICATION LANGUAGE: DEFINITION

$$\texttt{r} \in \texttt{rel.seed} := vis \mid so \mid \texttt{r} \cup \texttt{r}$$

SPECIFICATION LANGUAGE: DEFINITION

$(\mathrm{a} \xrightarrow{r_1;r_2;\ldots;r_k} b)$ is interpreted as

$\exists c.(a \xrightarrow{r_1;r_2;\ldots;r_{k-1}} c \wedge c \xrightarrow{r_k} b)$

$$\begin{array}{lcll} \mathtt{r} & \in & \mathtt{rel.seed} & := & vis \mid so \mid \mathtt{r} \cup \mathtt{r} \\ \mathtt{R} & \in & \mathtt{relation} & := & \mathtt{r} \mid \mathtt{R}; \mathtt{r} \mid \mathtt{null} \end{array}$$

SPECIFICATION LANGUAGE: DEFINITION

$(\mathrm{a} \xrightarrow{r_1;r_2;\ldots;r_k} b)$ is interpreted as

$\exists c.(a \xrightarrow{r_1;r_2;\ldots;r_{k-1}} c \wedge c \xrightarrow{r_k} b)$

$$
\begin{array}{lcll}
\mathtt{r} & \in & \mathtt{rel.seed} & := \quad vis \mid so \mid \mathtt{r} \cup \mathtt{r} \\
\mathtt{R} & \in & \mathtt{relation} & := \quad \mathtt{r} \mid \mathtt{R}; \mathtt{r} \mid \mathtt{null} \\
\pi & \in & \mathtt{prop} & := \quad \forall a.\ a \xrightarrow{R} \hat{\eta} \ \Rightarrow \ a \xrightarrow{vis} \hat{\eta}
\end{array}
$$

SPECIFICATION LANGUAGE: DEFINITION

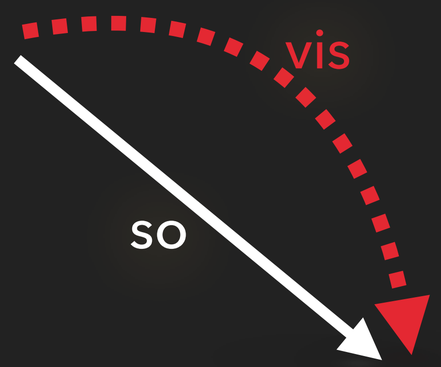
$(\mathrm{a} \xrightarrow{r_1;r_2;\ldots;r_k} b)$ is interpreted as

$\exists c.(a \xrightarrow{r_1;r_2;\ldots;r_{k-1}} c \wedge c \xrightarrow{r_k} b)$

$$
\begin{array}{lcll}
\mathtt{r} & \in & \mathtt{rel.seed} & := \quad vis \mid so \mid \mathtt{r} \cup \mathtt{r} \\
\mathtt{R} & \in & \mathtt{relation} & := \quad \mathtt{r} \mid \mathtt{R}; \mathtt{r} \mid \mathtt{null} \\
\pi & \in & \mathtt{prop} & := \quad \forall a.\ a \xrightarrow{R} \hat{\eta} \ \Rightarrow \ a \xrightarrow{vis} \hat{\eta} \\
\psi & \in & \mathtt{spec} & := \quad \pi \mid \pi \wedge \pi
\end{array}
$$

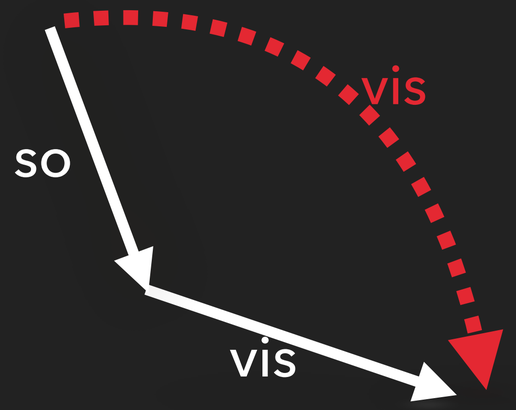
GENERALITY: SESSION GUARANTEES

READ MY WRITES



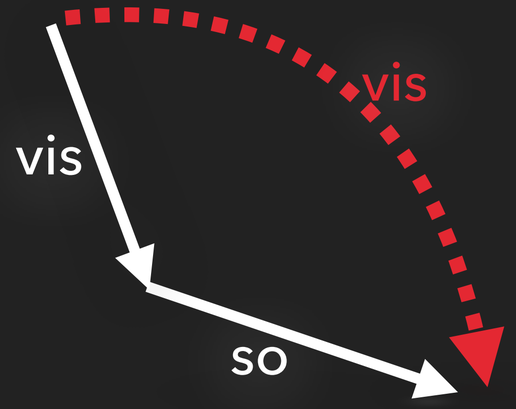
$$\forall a.a \xrightarrow{so} \hat{\eta} \Rightarrow a \xrightarrow{vis} \hat{\eta}$$

MONOTONIC WRITES



$$\forall a.a \xrightarrow{so;vis} \hat{\eta} \Rightarrow a \xrightarrow{vis} \hat{\eta}$$

MONOTONIC READS



$$\forall a.a \xrightarrow{vis;so} \hat{\eta} \Rightarrow a \xrightarrow{vis} \hat{\eta}$$

TWO CLASSES

TWO CLASSES

- Guarantees can be classified into two groups: **L**ower **B**ound (LB) and **U**pper **B**ound (UB) contracts

TWO CLASSES

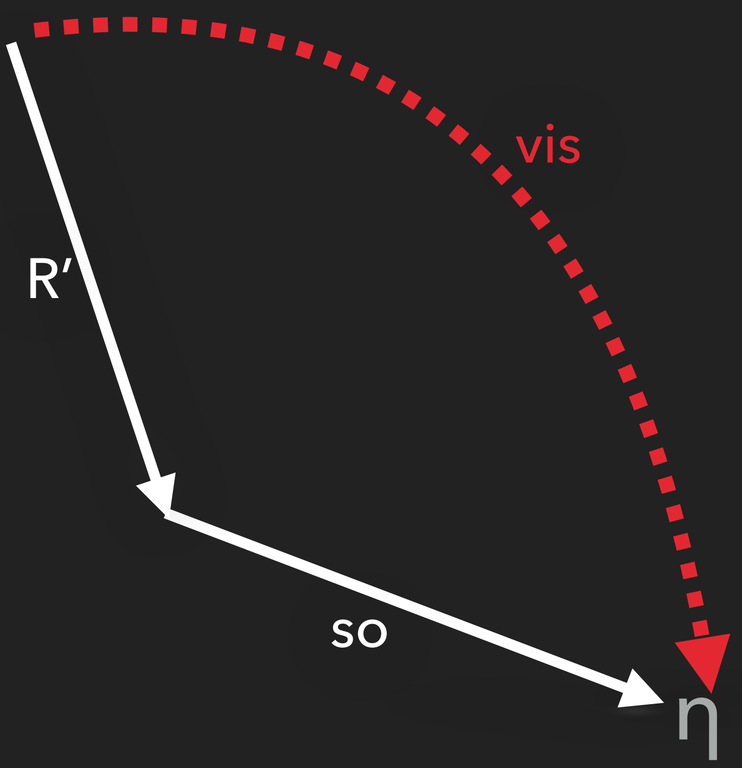
- Guarantees can be classified into two groups: **L**ower **B**ound (LB) and **U**pper **B**ound (UB) contracts
- Each class naturally maps to an enforcement mechanism

THE CORE OF LB

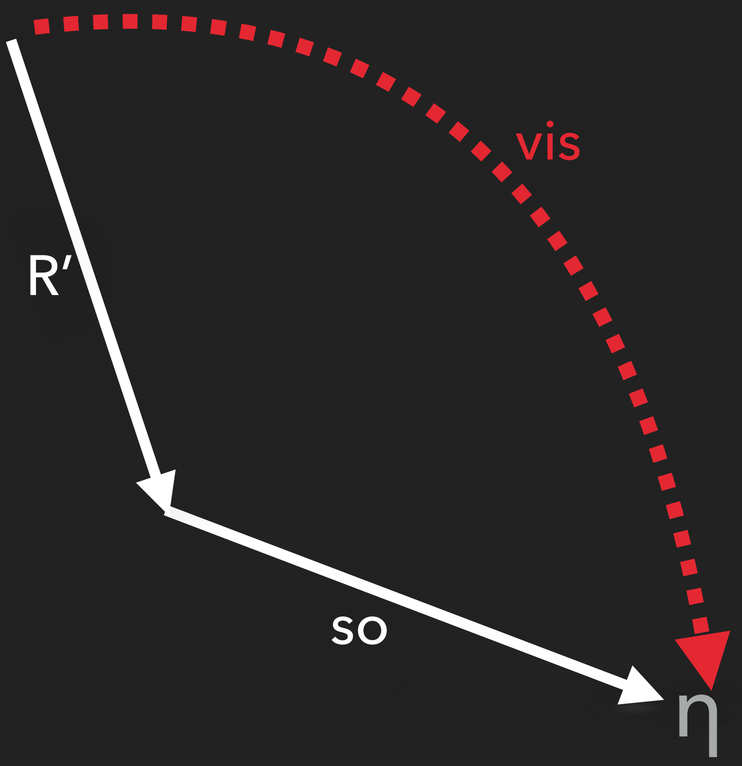
THE CORE OF LB

- LB guarantees specify a *minimal set* of effects that must be visible to an effect at the time of its creation

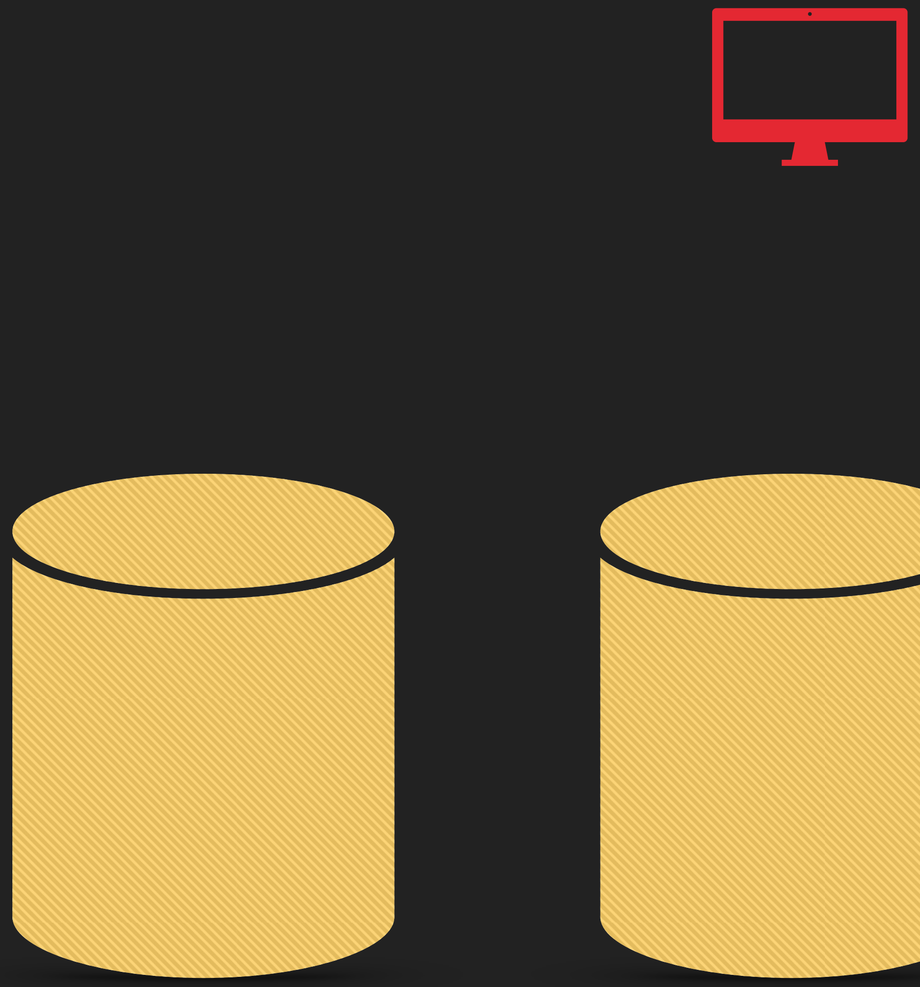
- LB guarantees specify a *minimal set* of effects that must be visible to an effect at the time of its creation
- The dependence relation of LB guarantees ends with an **so** edge: R′;so



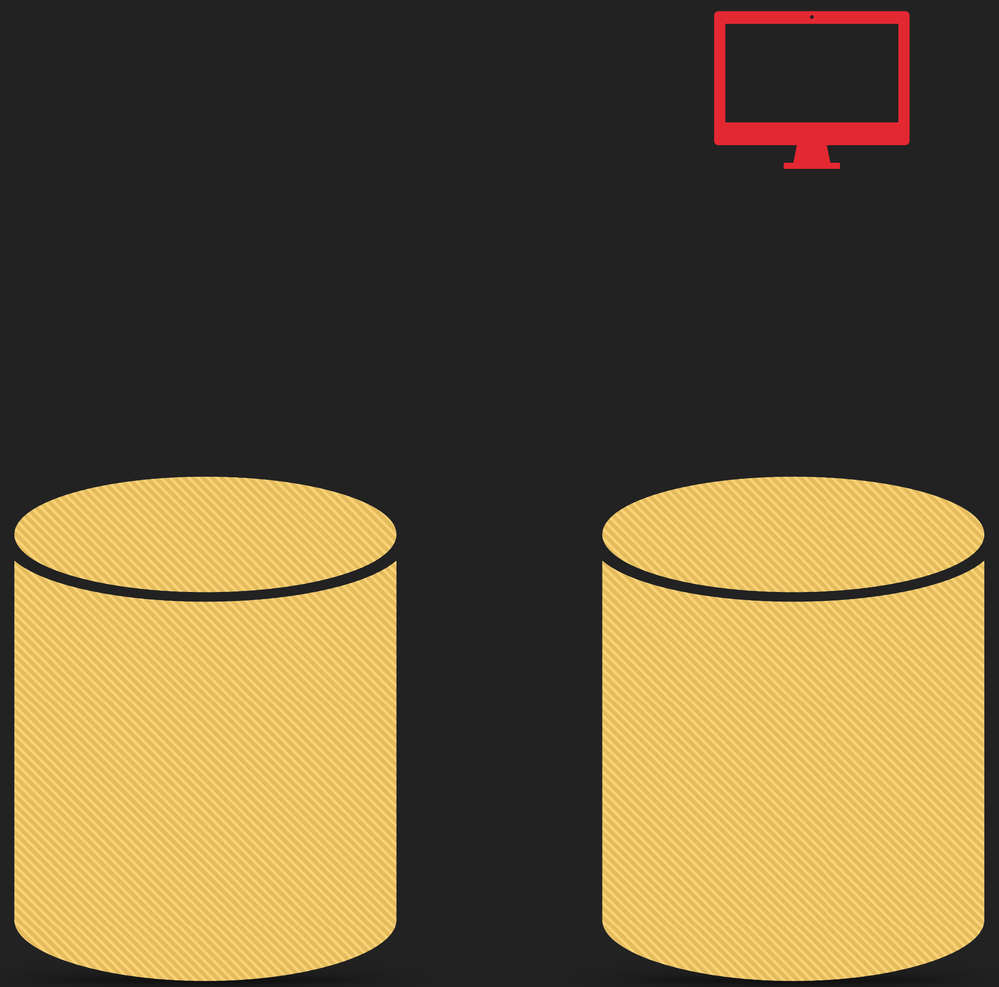
- LB guarantees specify a *minimal set* of effects that must be visible to an effect at the time of its creation
- The dependence relation of LB guarantees ends with an **so** edge: R′;so
- The dependency set of the current operation is *pre-determined* at the time of its execution



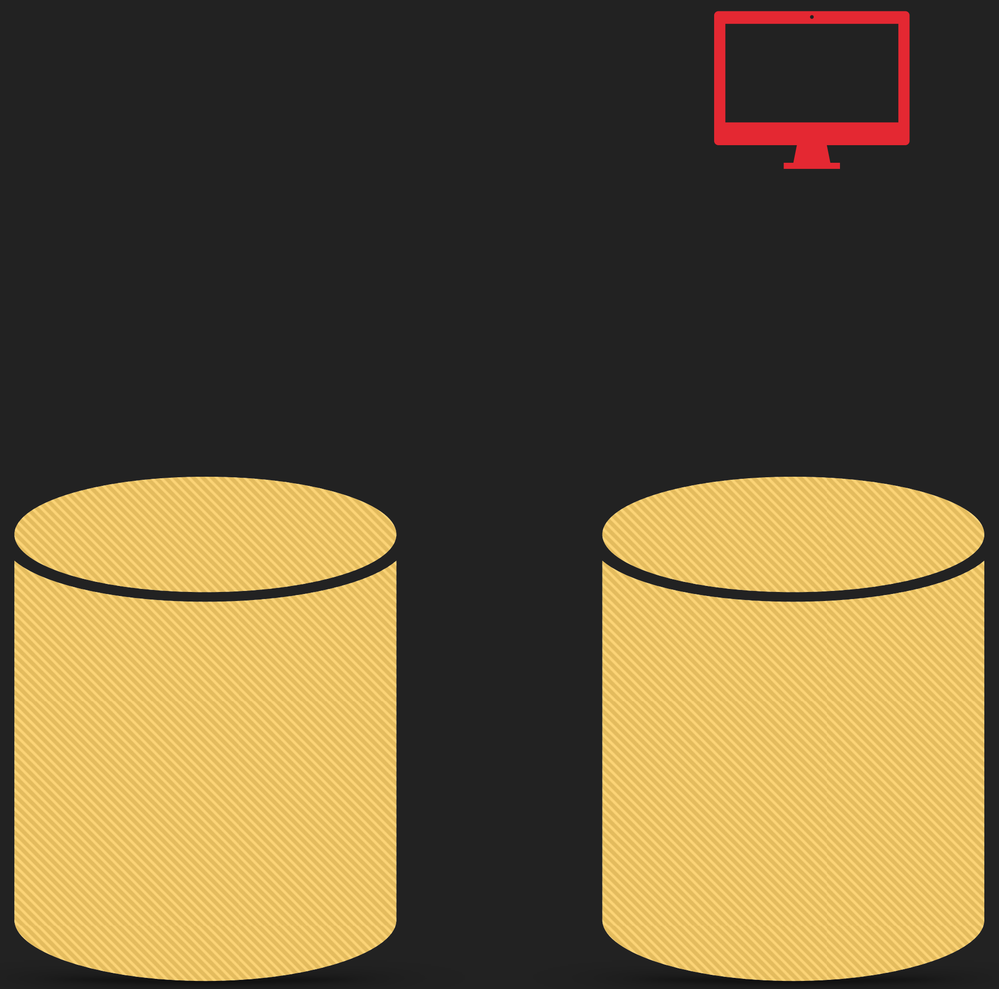
HOW TO ENFORCE LB



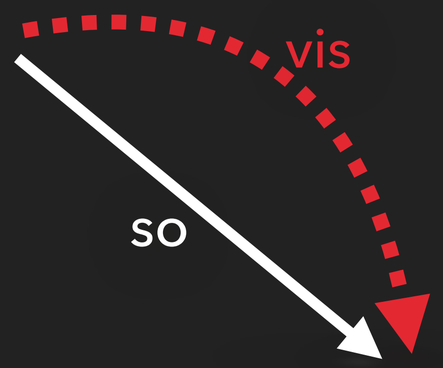
- LB contracts can be enforced by ***blocking*** the client requests temporarily



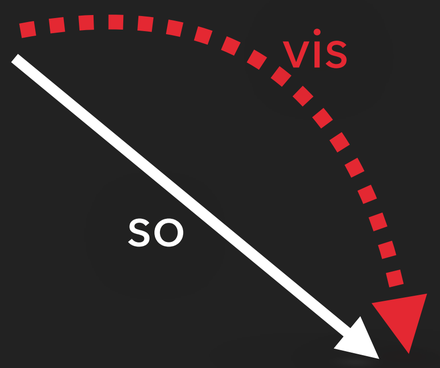
- LB contracts can be enforced by ***blocking*** the client requests temporarily
- Assuming EC, the dependency set will eventually become available locally



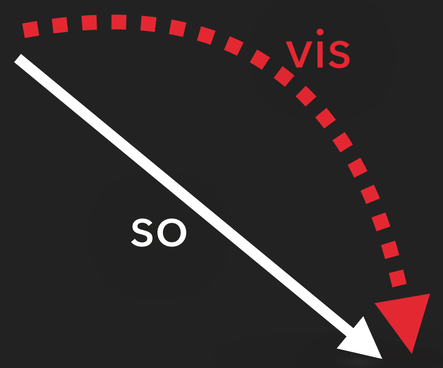
- LB contracts can be enforced by ***blocking*** the client requests temporarily
- Assuming EC, the dependency set will eventually become available locally
- Example: RMW

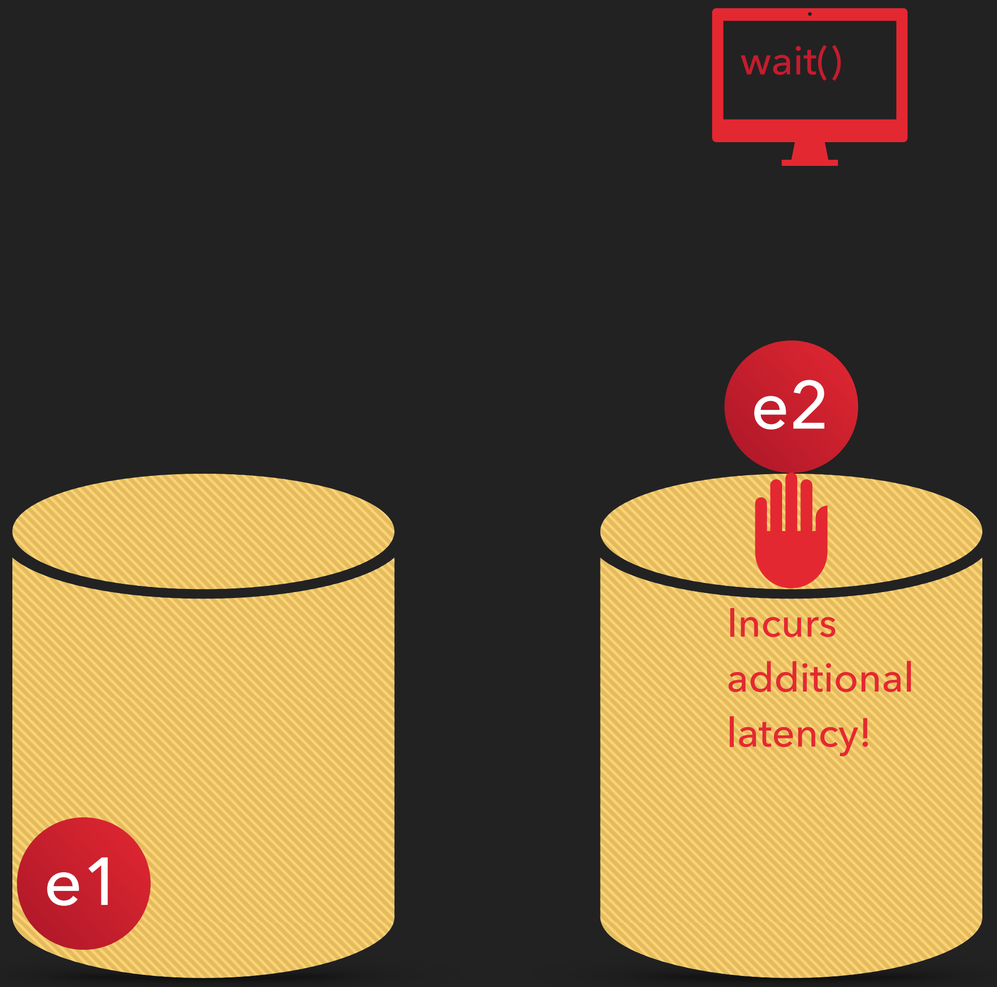


- LB contracts can be enforced by ***blocking*** the client requests temporarily
- Assuming EC, the dependency set will eventually become available locally
- Example: RMW

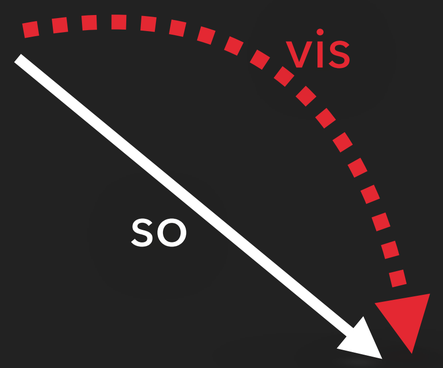


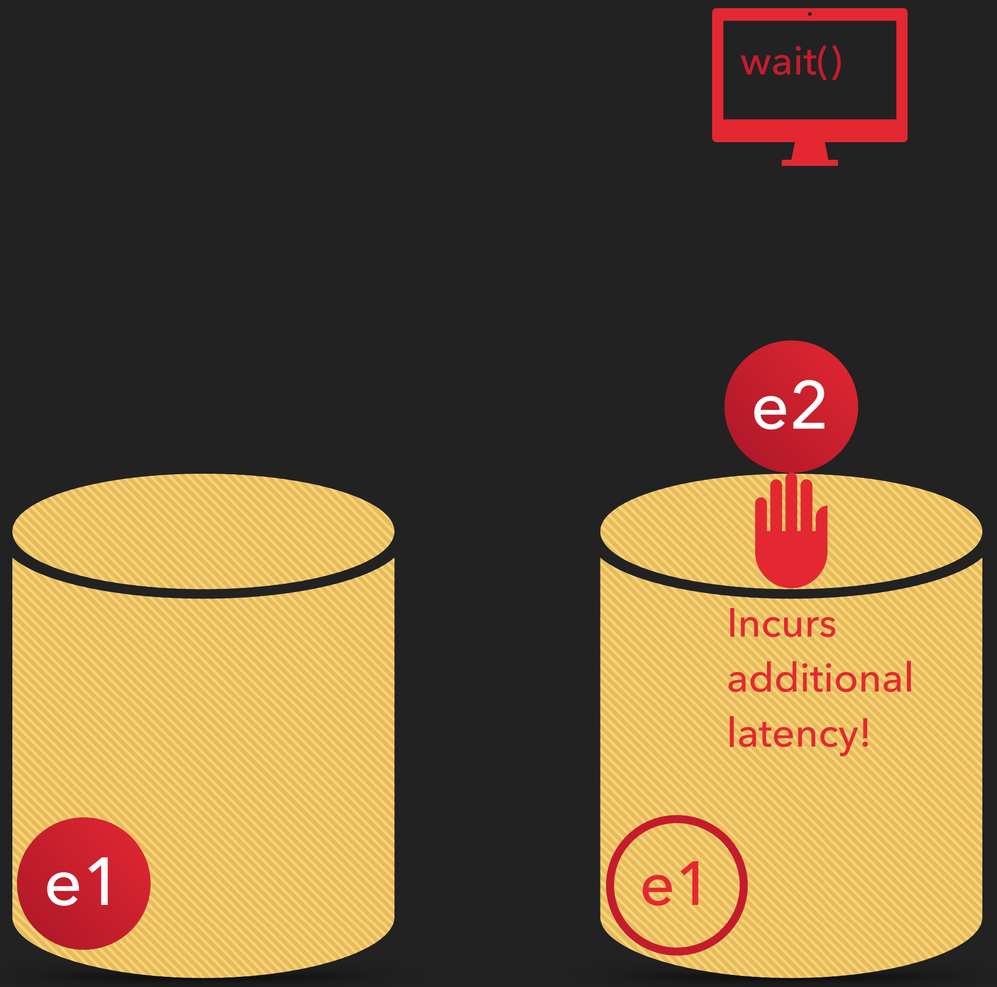
- LB contracts can be enforced by ***blocking*** the client requests temporarily
- Assuming EC, the dependency set will eventually become available locally
- Example: RMW



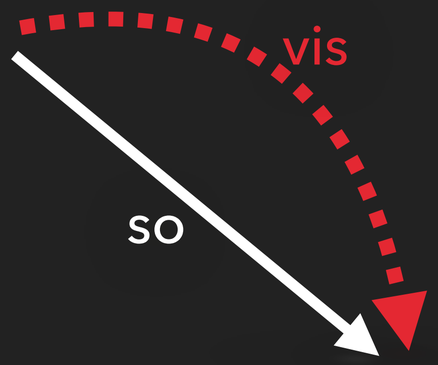


- LB contracts can be enforced by ***blocking*** the client requests temporarily
- Assuming EC, the dependency set will eventually become available locally
- Example: RMW





- LB contracts can be enforced by ***blocking*** the client requests temporarily
- Assuming EC, the dependency set will eventually become available locally
- Example: RMW

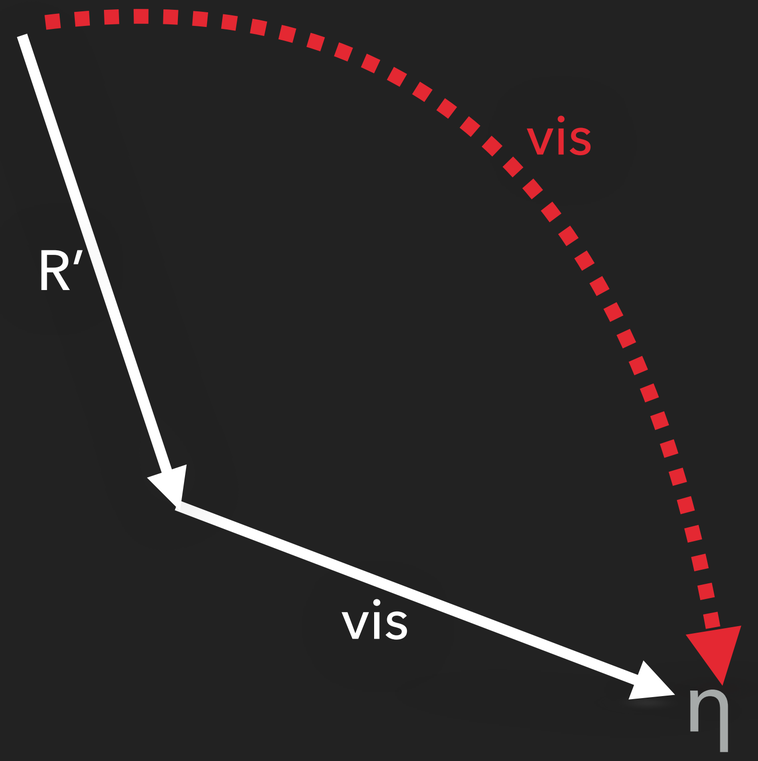


WHAT IS THE CORE OF UB

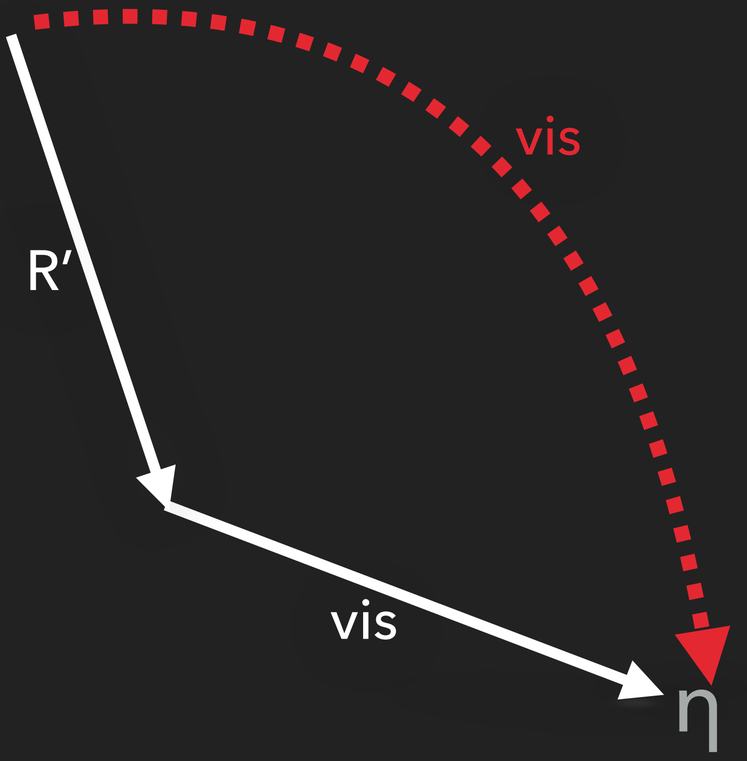
WHAT IS THE CORE OF UB

- UB guarantees specify a *constraint* on the set of effects that ***might*** *be visible to an effect*

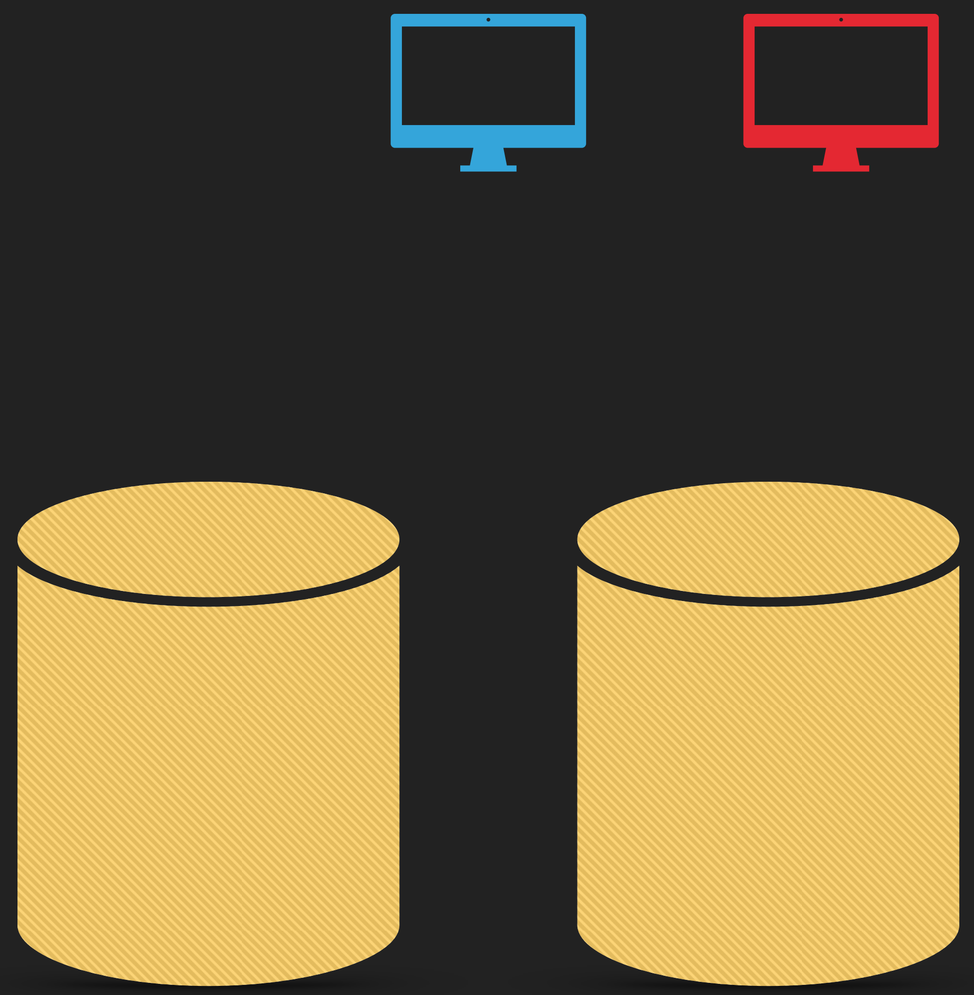
- UB guarantees specify a *constraint* on the set of effects that ***might*** *be visible to an effect*
- The dependence relation of UB guarantees ends with **vis**:
  R′;vis



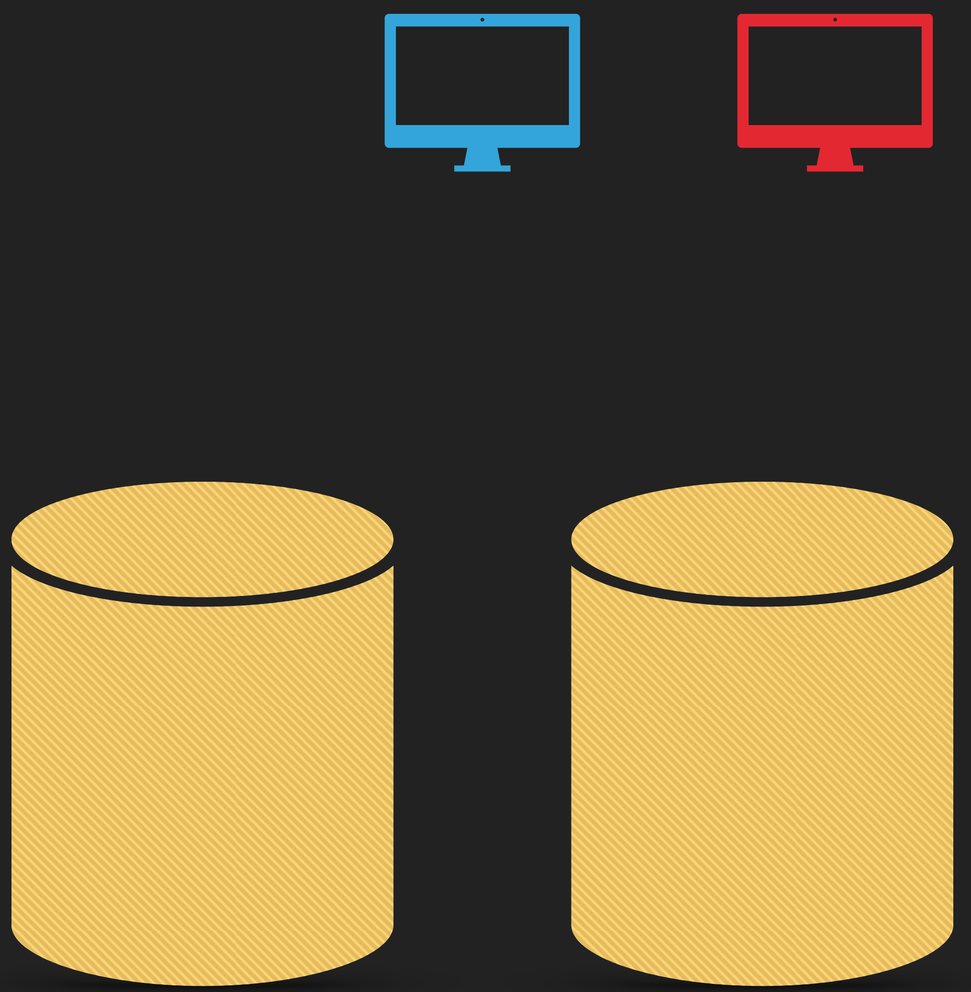
- UB guarantees specify a *constraint* on the set of effects that ***might*** *be visible to an effect*
- The dependence relation of UB guarantees ends with **vis**:
  R′;vis
- The dependency set of the current operation can be determined at the running replica by changing the visible snapshot to the current operation



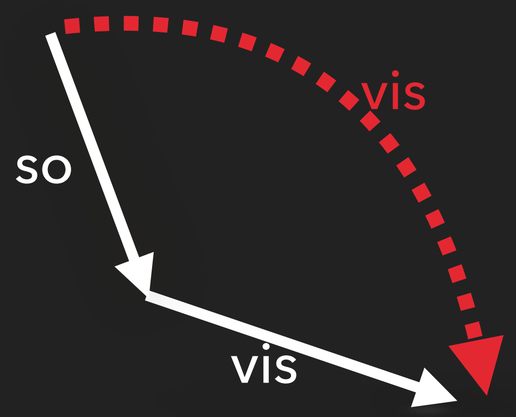
HOW TO ENFORCE UB

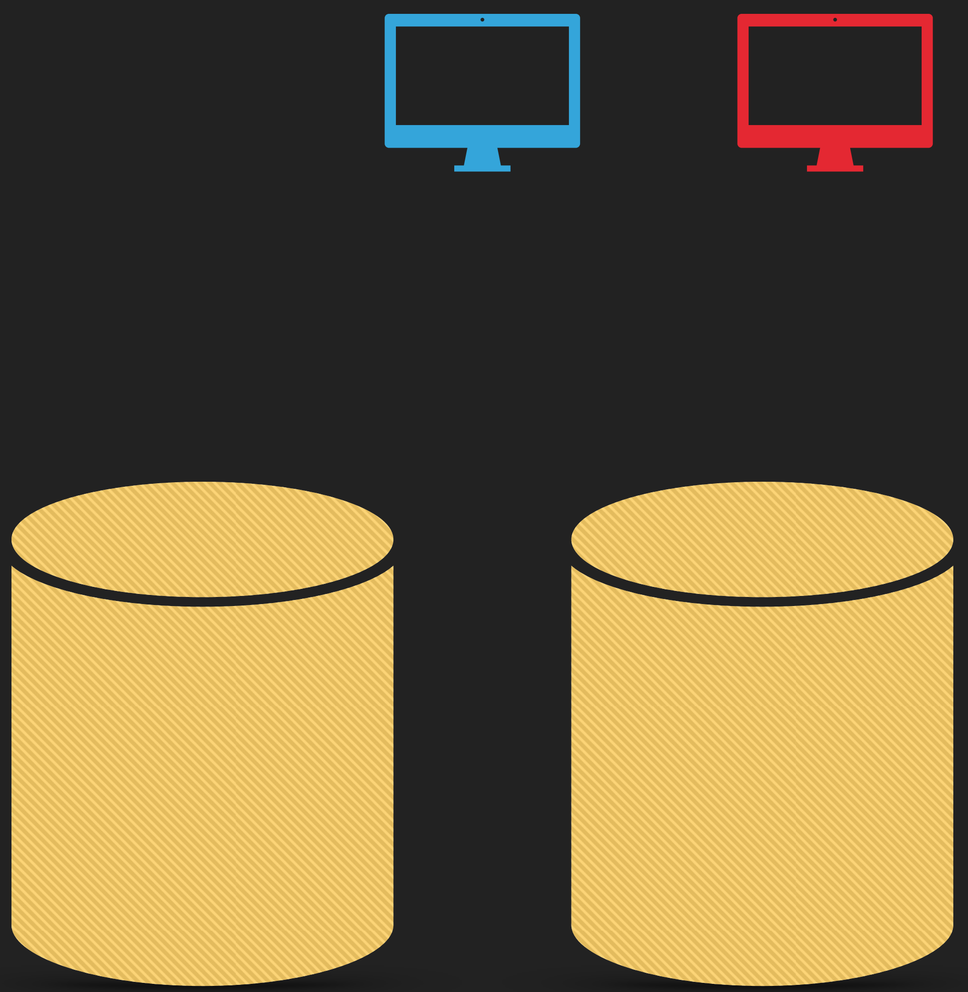


- UB contracts can be satisfied by showing a ***filtered*** snapshot of the locally available updates to each operation

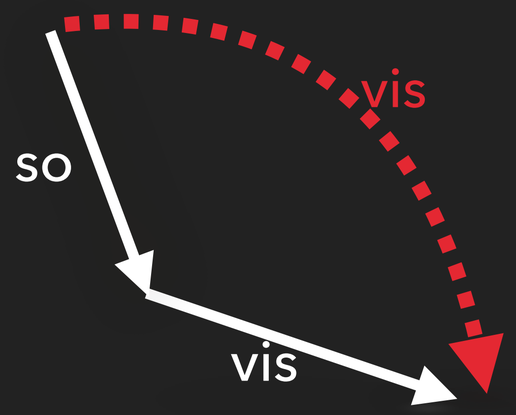


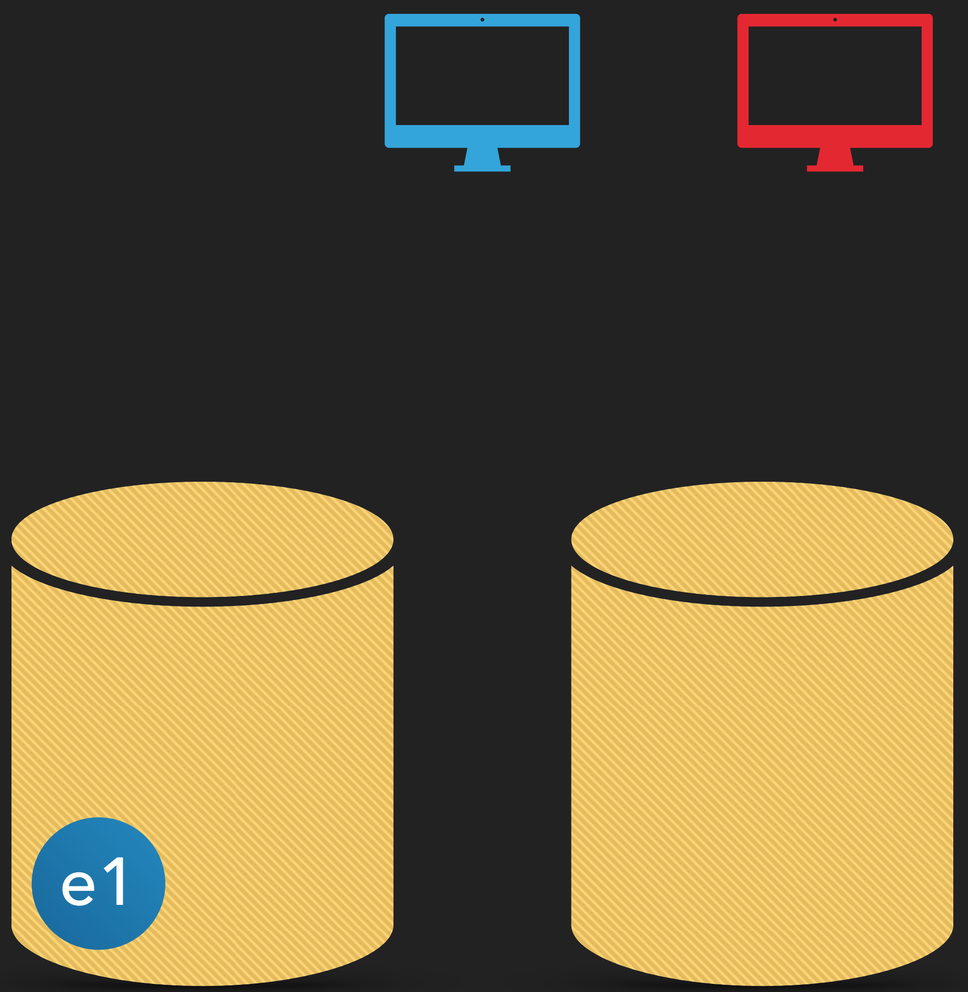
- UB contracts can be satisfied by showing a ***filtered*** snapshot of the locally available updates to each operation
- Example: MW



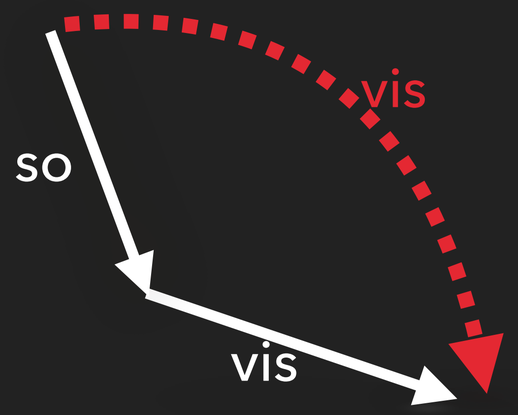


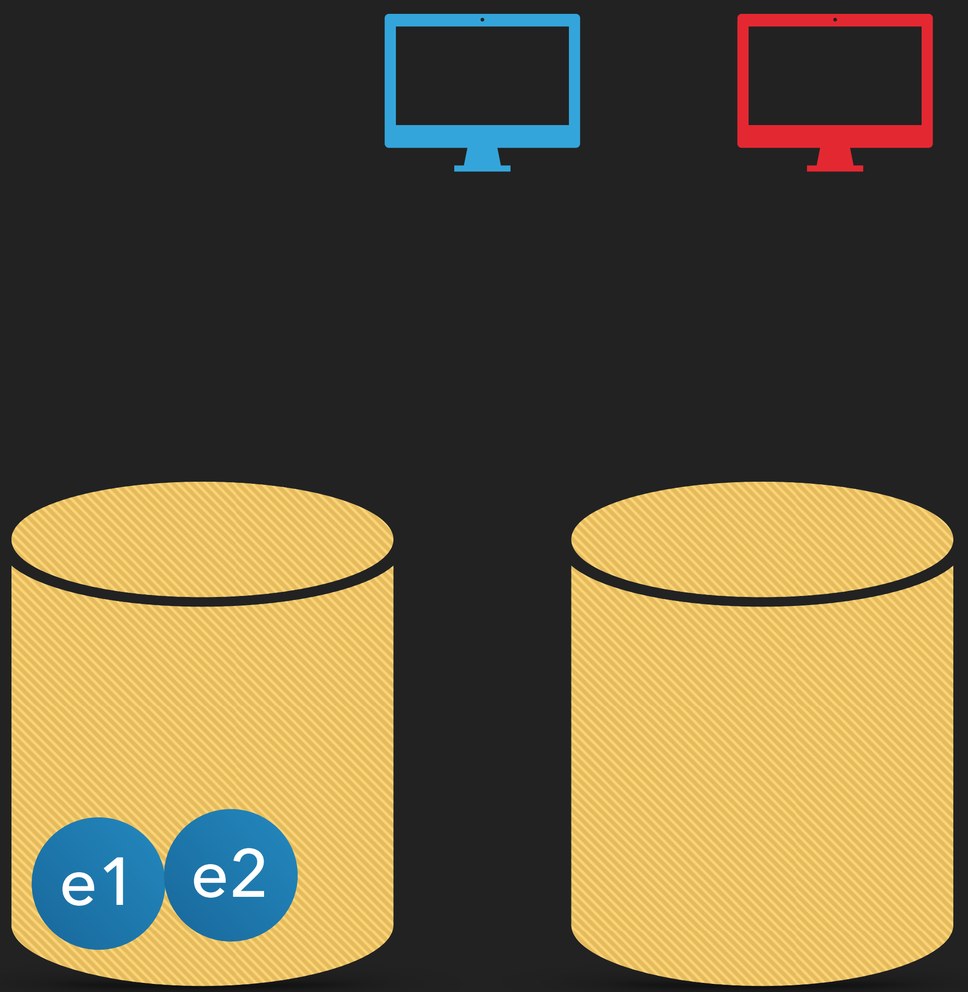
- UB contracts can be satisfied by showing a ***filtered*** snapshot of the locally available updates to each operation
- Example: MW



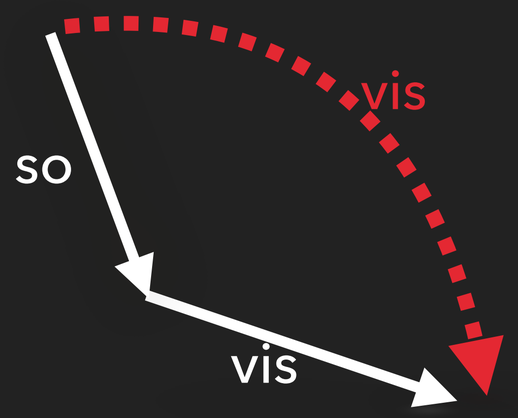


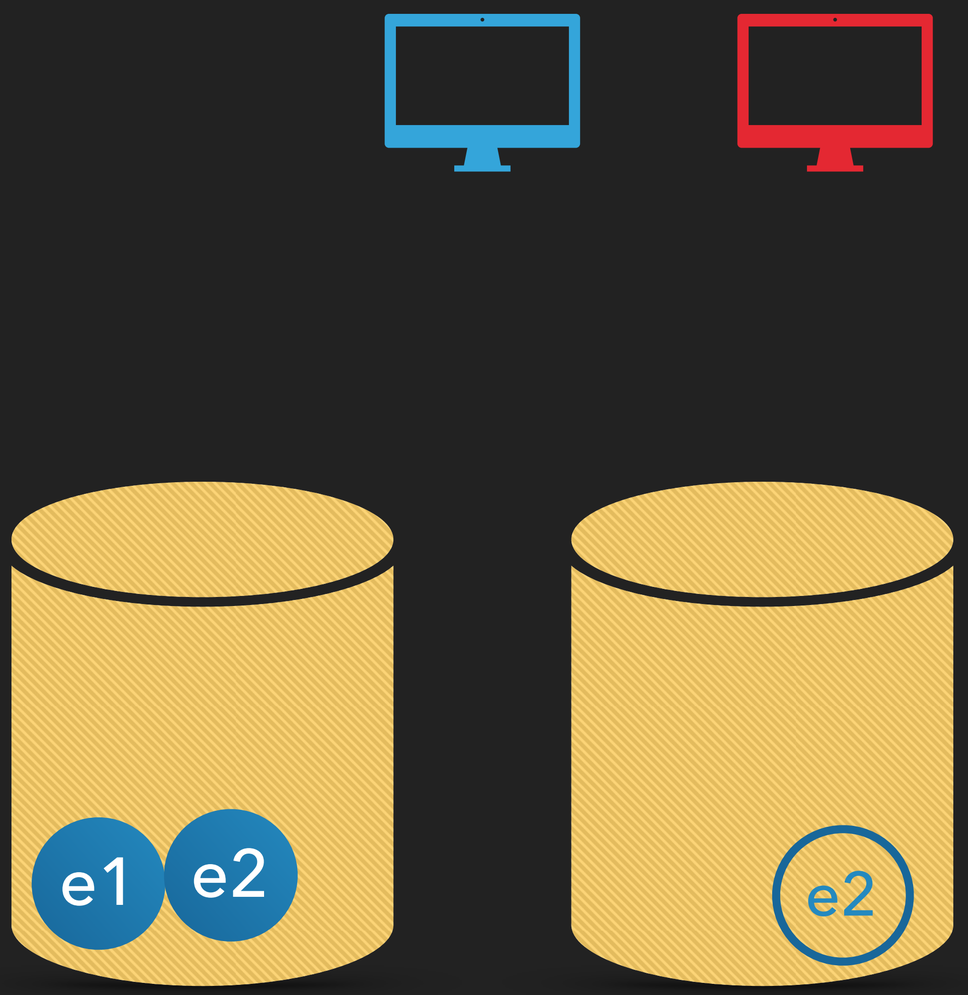
- UB contracts can be satisfied by showing a ***filtered*** snapshot of the locally available updates to each operation
- Example: MW



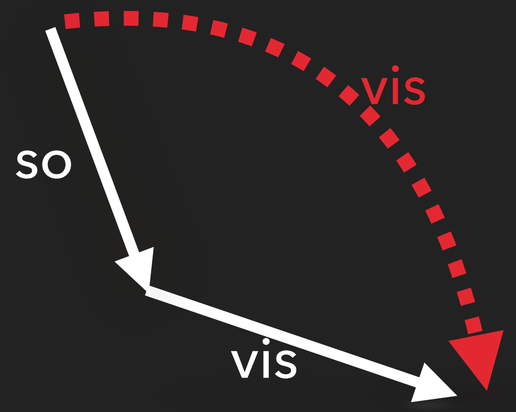


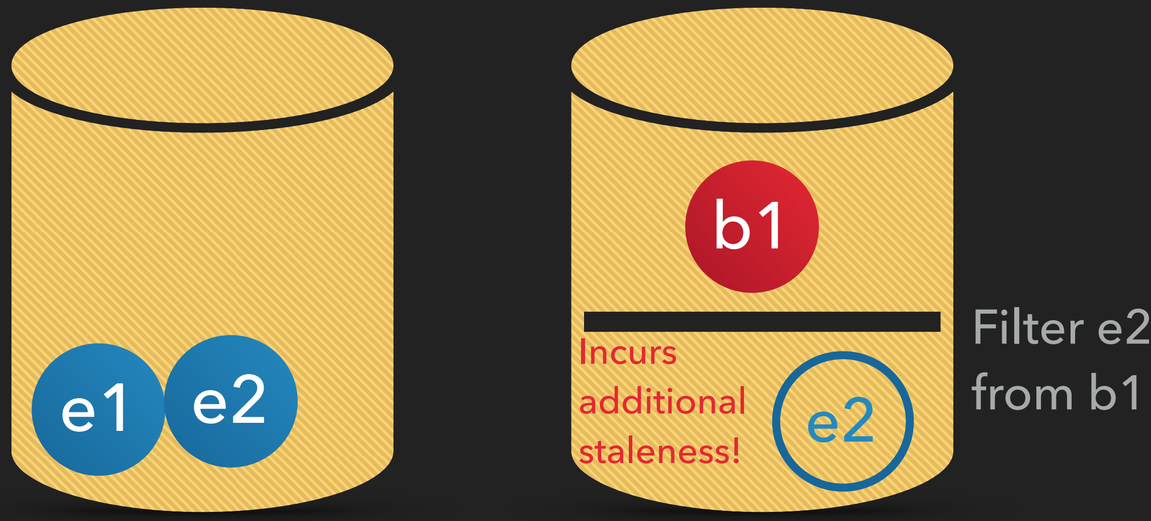
- UB contracts can be satisfied by showing a ***filtered*** snapshot of the locally available updates to each operation
- Example: MW





- UB contracts can be satisfied by showing a ***filtered*** snapshot of the locally available updates to each operation
- Example: MW

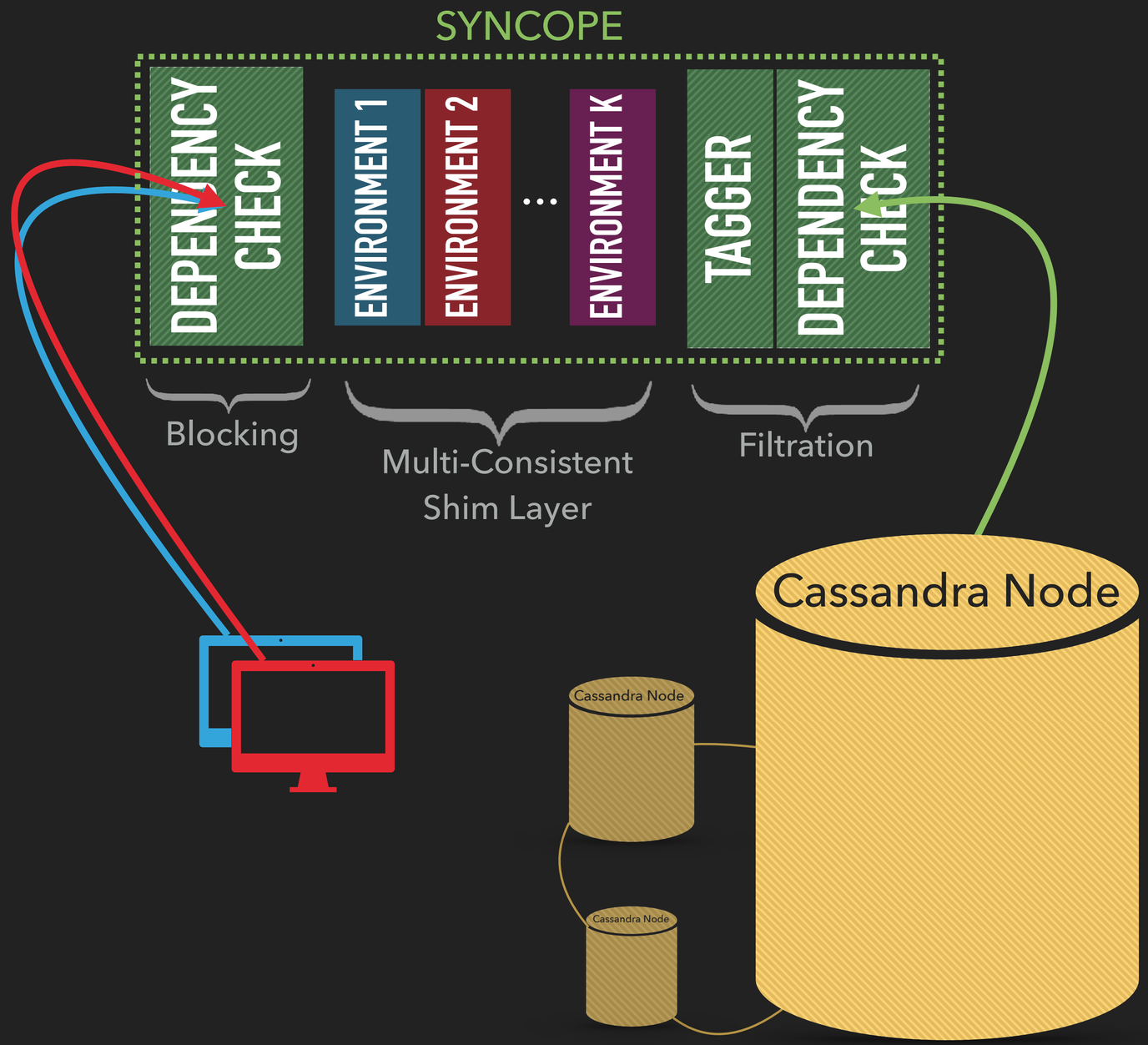




OVERVIEW OF THE IMPLEMENTATION

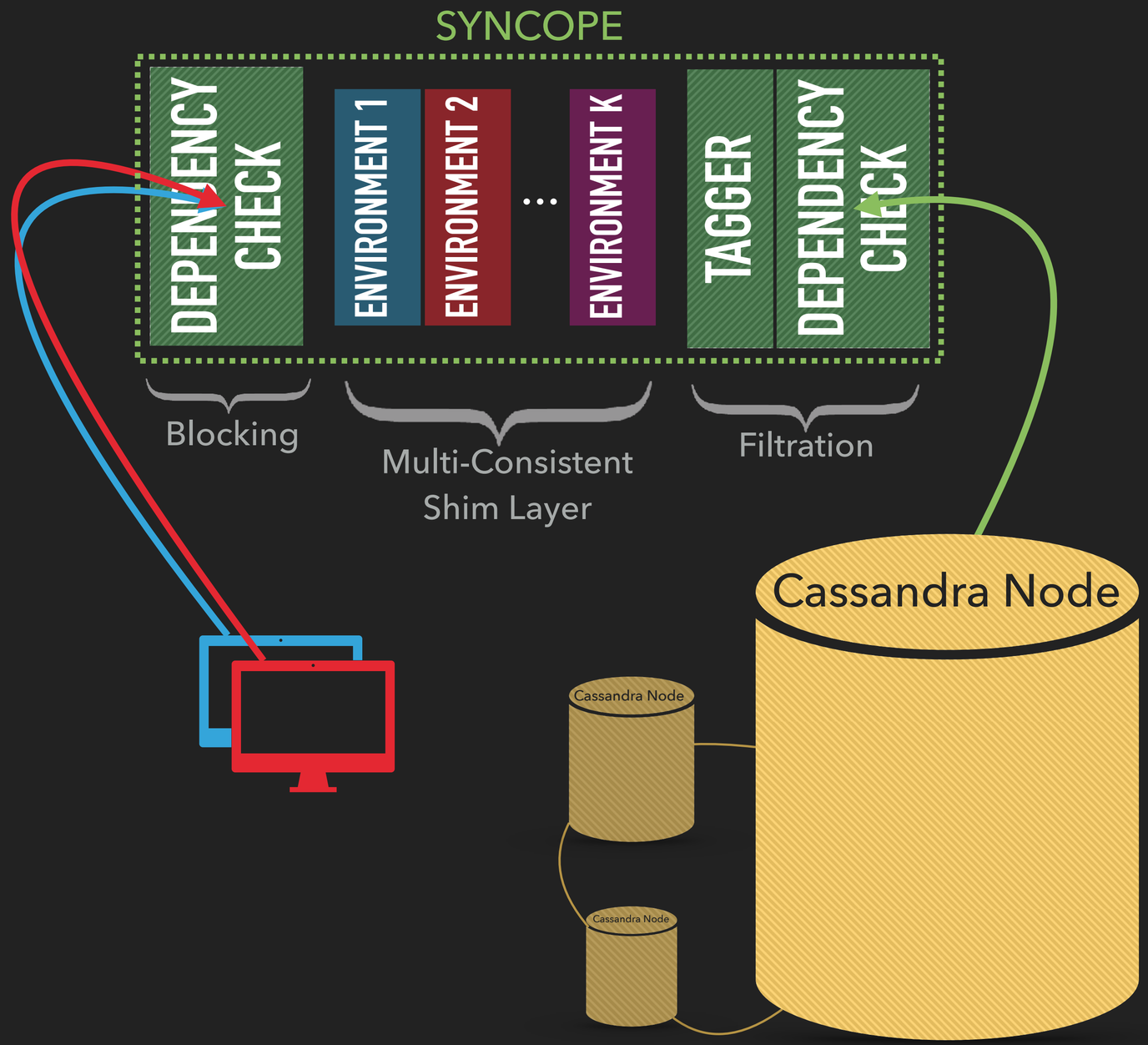
OVERVIEW OF THE IMPLEMENTATION

- Implemented in Haskell



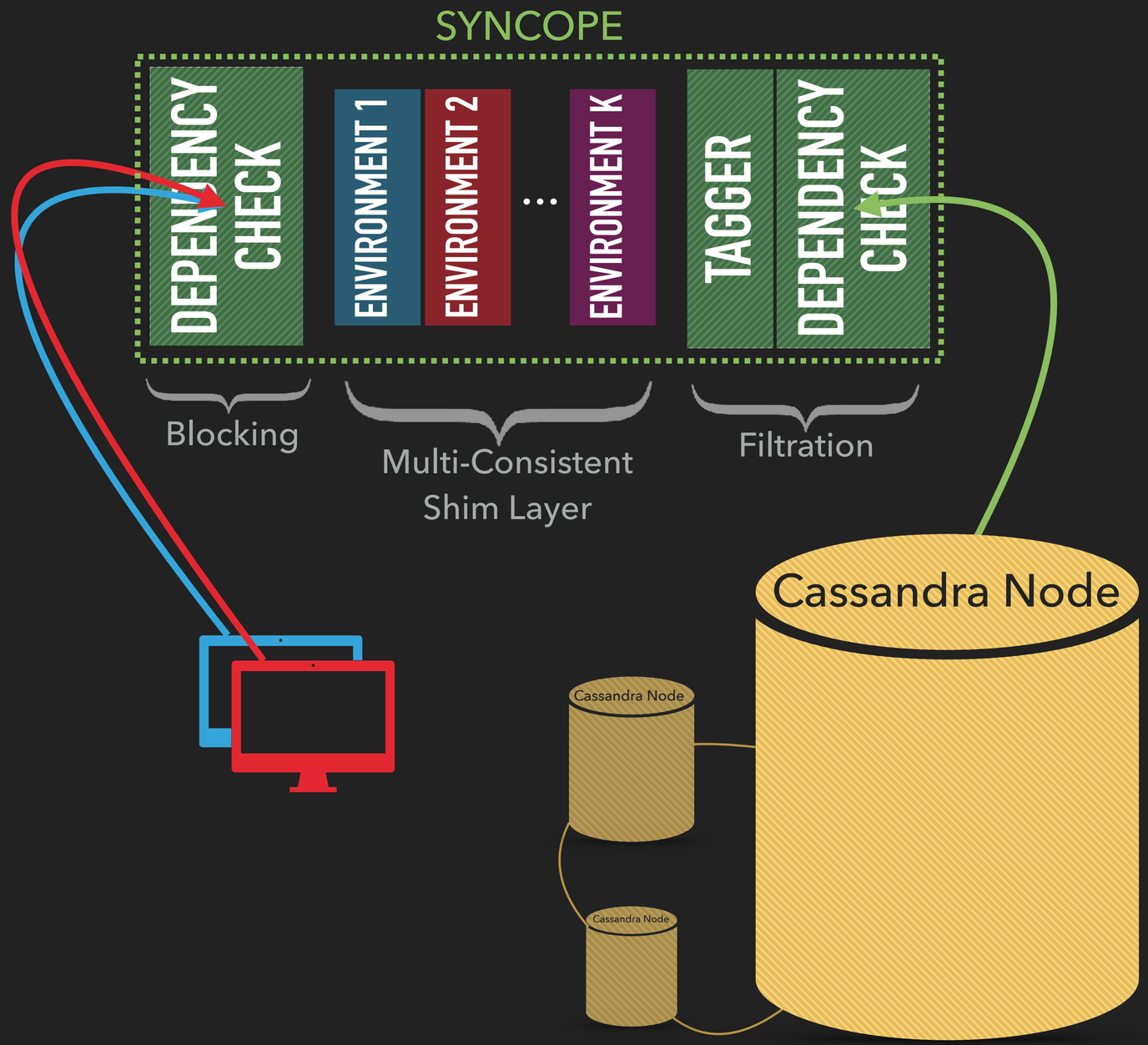
OVERVIEW OF THE IMPLEMENTATION

- Implemented in Haskell
- Backed up by Cassandra



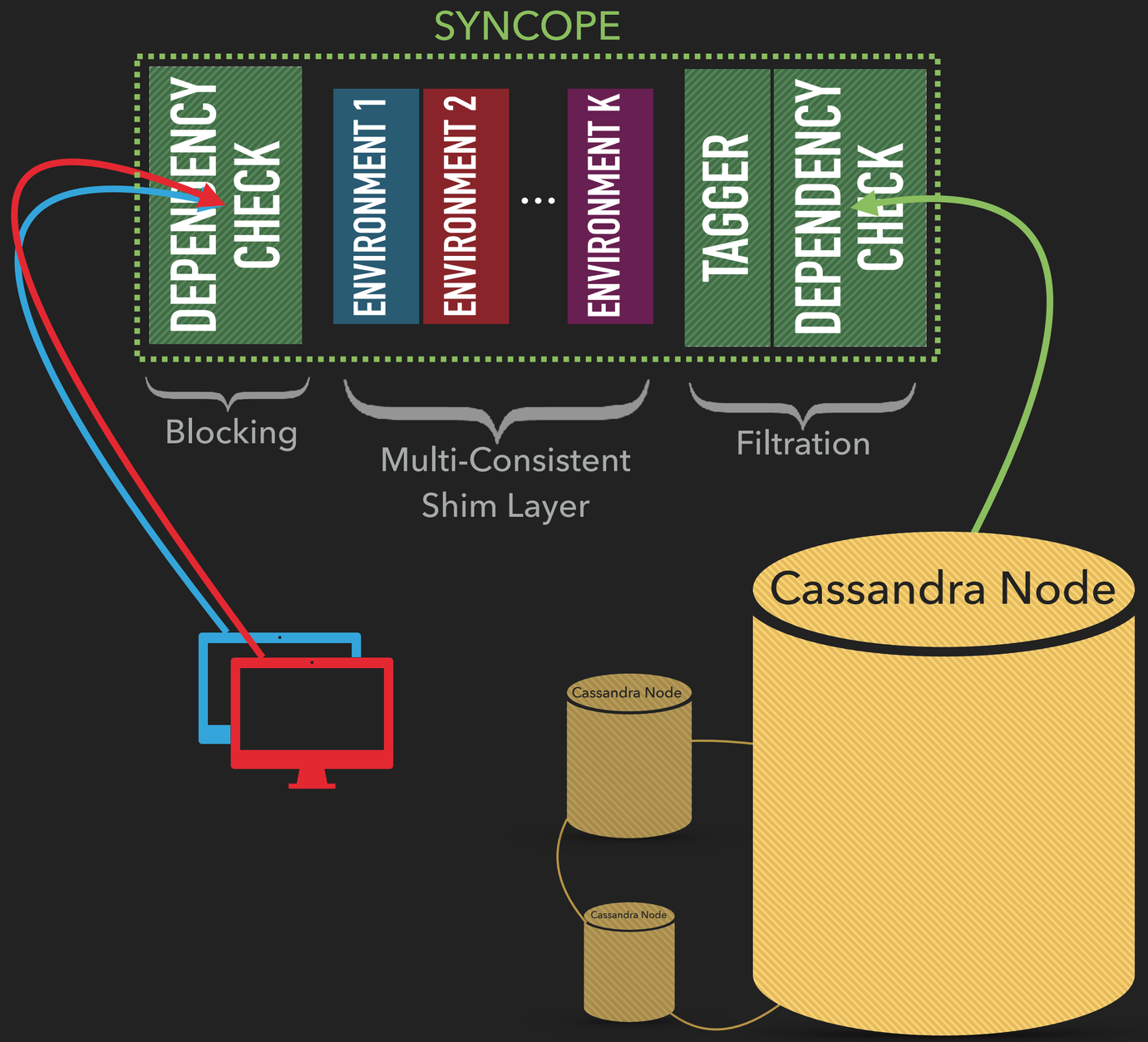
OVERVIEW OF THE IMPLEMENTATION

- Implemented in Haskell
- Backed up by Cassandra
- Each operation is executed on it’s own snapshot of the local replica



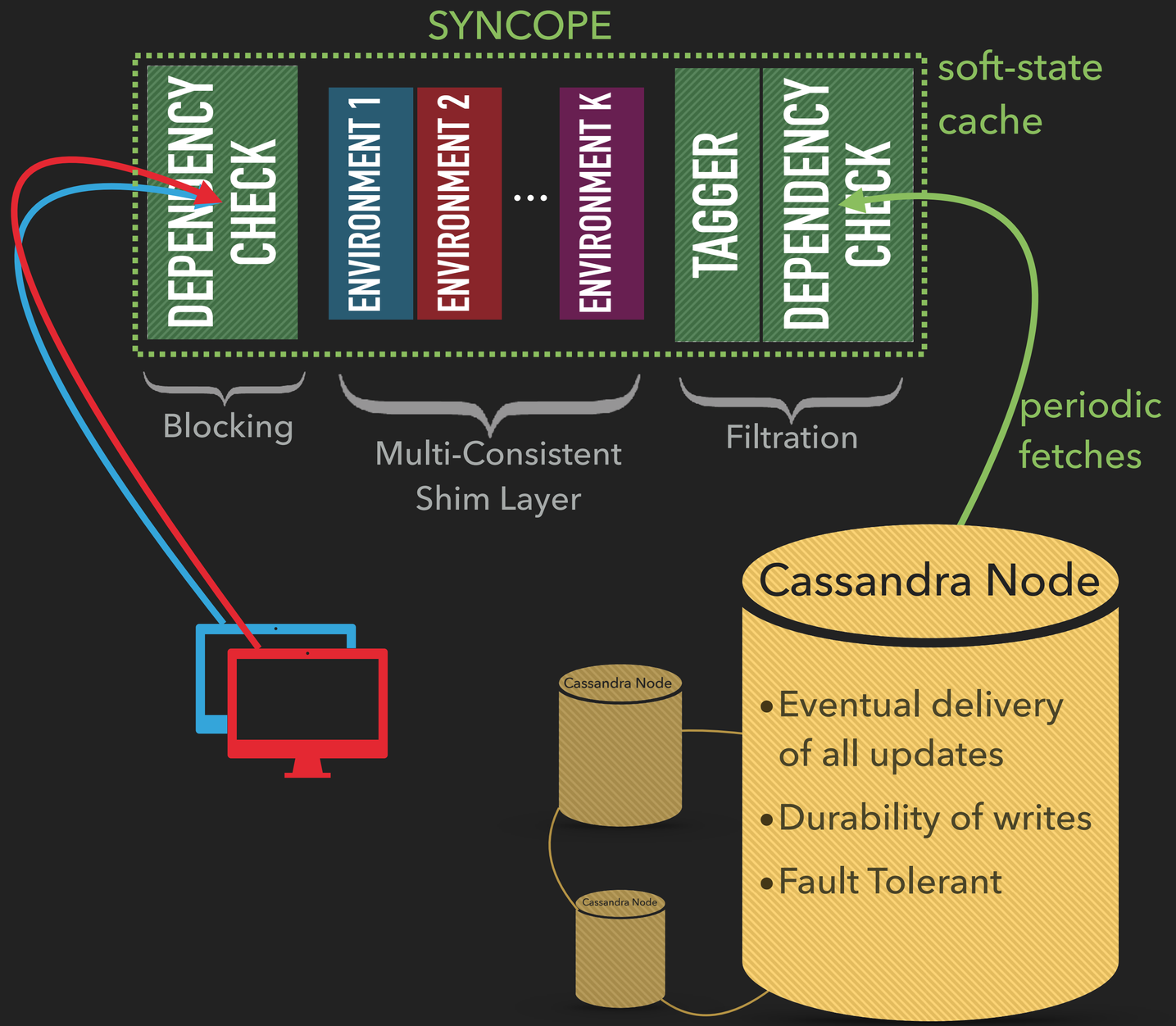
OVERVIEW OF THE IMPLEMENTATION

- Implemented in Haskell
- Backed up by Cassandra
- Each operation is executed on it’s own snapshot of the local replica
- Can be realized by keeping multiple local copies of data or by a simple tagging mechanism



OVERVIEW OF THE IMPLEMENTATION

- Implemented in Haskell
- Backed up by Cassandra
- Each operation is executed on it’s own snapshot of the local replica
- Can be realized by keeping multiple local copies of data or by a simple tagging mechanism



EVALUATION: SETTING

EVALUATION: SETTING

- A 3-node cluster backed by Cassandra deployed on Amazon EC2

EVALUATION: SETTING

- A 3-node cluster backed by Cassandra deployed on Amazon EC2
- Artificial network fault injection: messages are randomly delayed for 1s

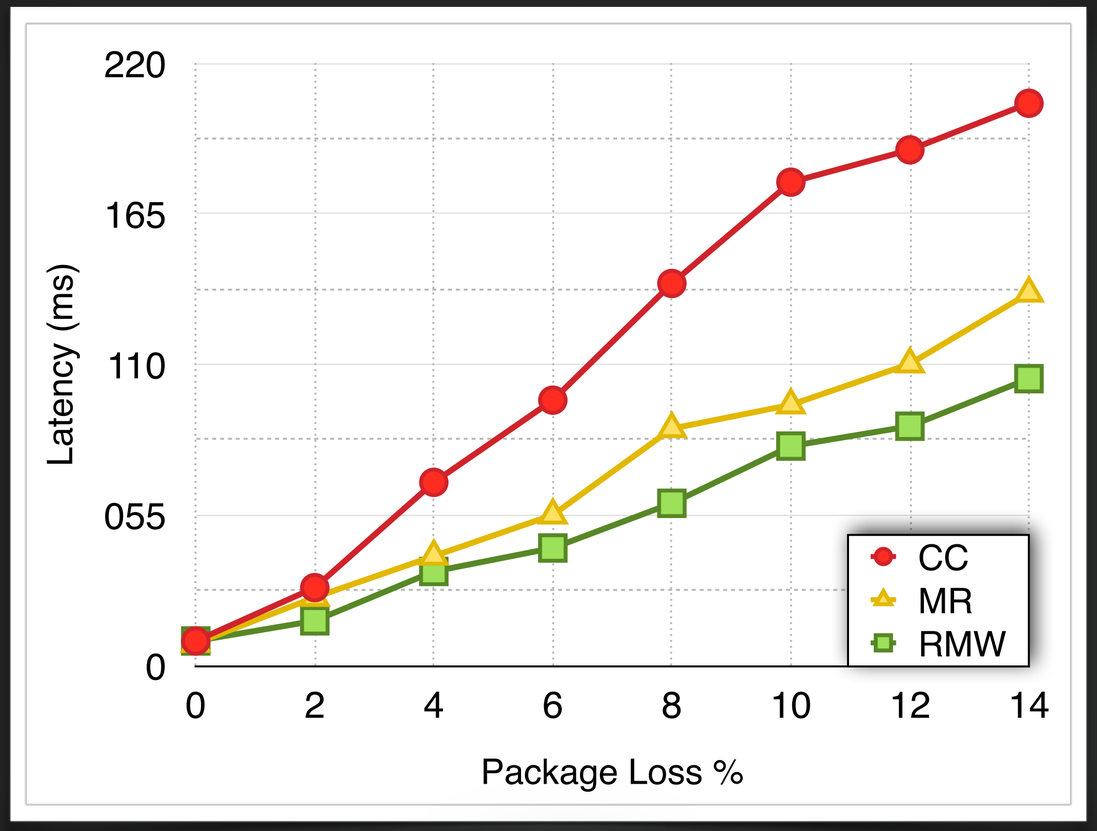
EVALUATION: SETTING

- A 3-node cluster backed by Cassandra deployed on Amazon EC2
- Artificial network fault injection: messages are randomly delayed for 1s
- User-Perceived latency (for LB guarantees) and visible snapshot staleness (for UB guarantees) are measured

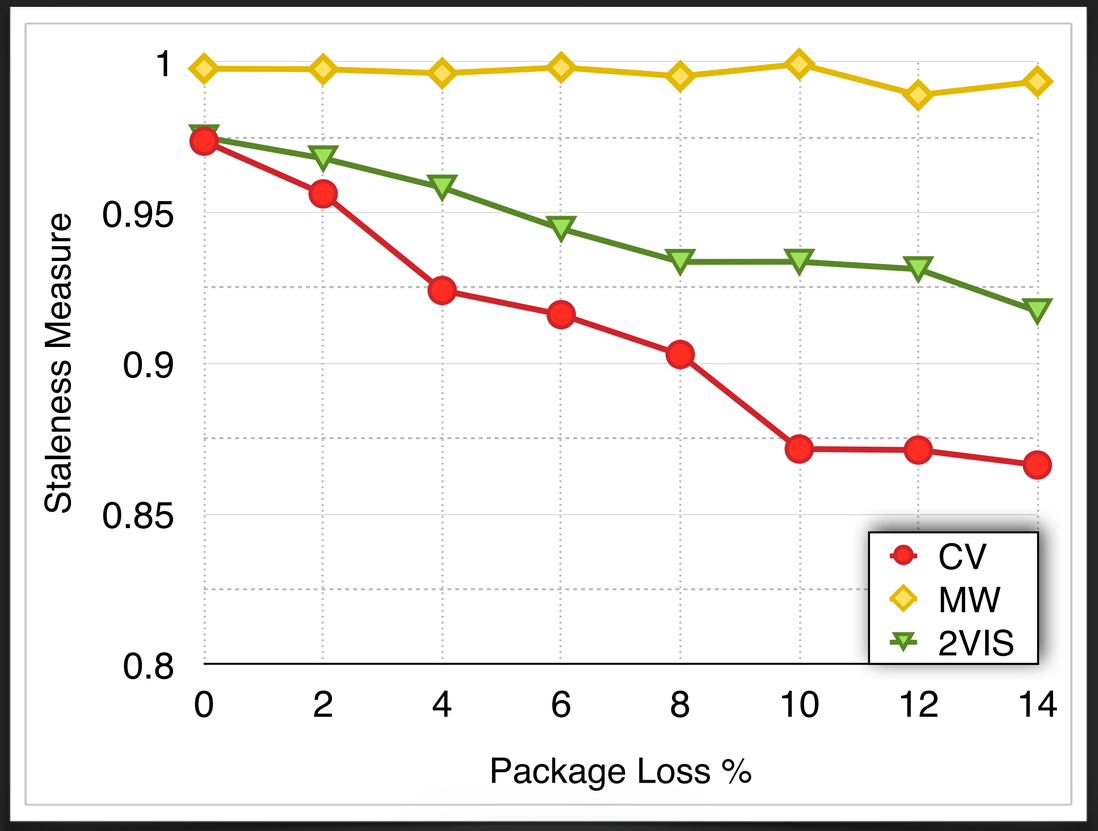
EVALUATION: SETTING

- A 3-node cluster backed by Cassandra deployed on Amazon EC2
- Artificial network fault injection: messages are randomly delayed for 1s
- User-Perceived latency (for LB guarantees) and visible snapshot staleness (for UB guarantees) are measured
- 50 Concurrent clients performing sessions of reads and write operations to random replicas on a shared counter object

EVALUATION: RESULTS



LATENCY



STALENESS

CONCLUSIONS

- We offer a generalized platform for specifying and enforcing fine-grained application-level consistency requirement
- Fine-grained consistency enforcement can result in considerable performance and availability gain in faulty networks
- The next goal is to build more fault resilient distributed systems, where SYNCOPE can be deployed as a secondary defense mechanism to fight unreliable networks without sacrificing correctness

THANK YOU!

QUESTIONS?