Mysteries of Dropbox

Property-Based Testing of a Distributed Synchronization Service

John Hughes, Benjamin Pierce, Thomas Arts, Ulf Norell

CHALMERS

Penn
UNIVERSITY of PENNSYLVANIA

QuviQ

Synchronization Services

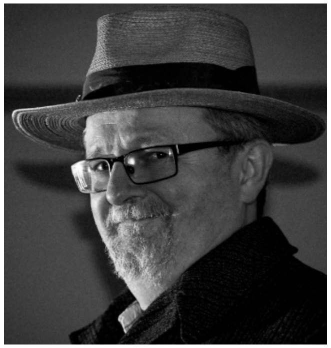
Dropbox	400 million (June 2015)
Google Drive	240 million (Oct 2014)
OneDrive	250 million (Nov 2014)

Are they trustworthy?

(exactly!)

What do they do?

Can we test them?





TEST
ING

Writing test cases by hand

(especially for testing distributed systems!)

Generate test cases from a model

Our Goals

- Develop a precise specification of the core behavior of a synchronization service
  - Phrased from the perspective of *users*
  - Applicable to a variety of different synchronizers
- Use property-based random testing to validate it against Dropbox's observed behavior

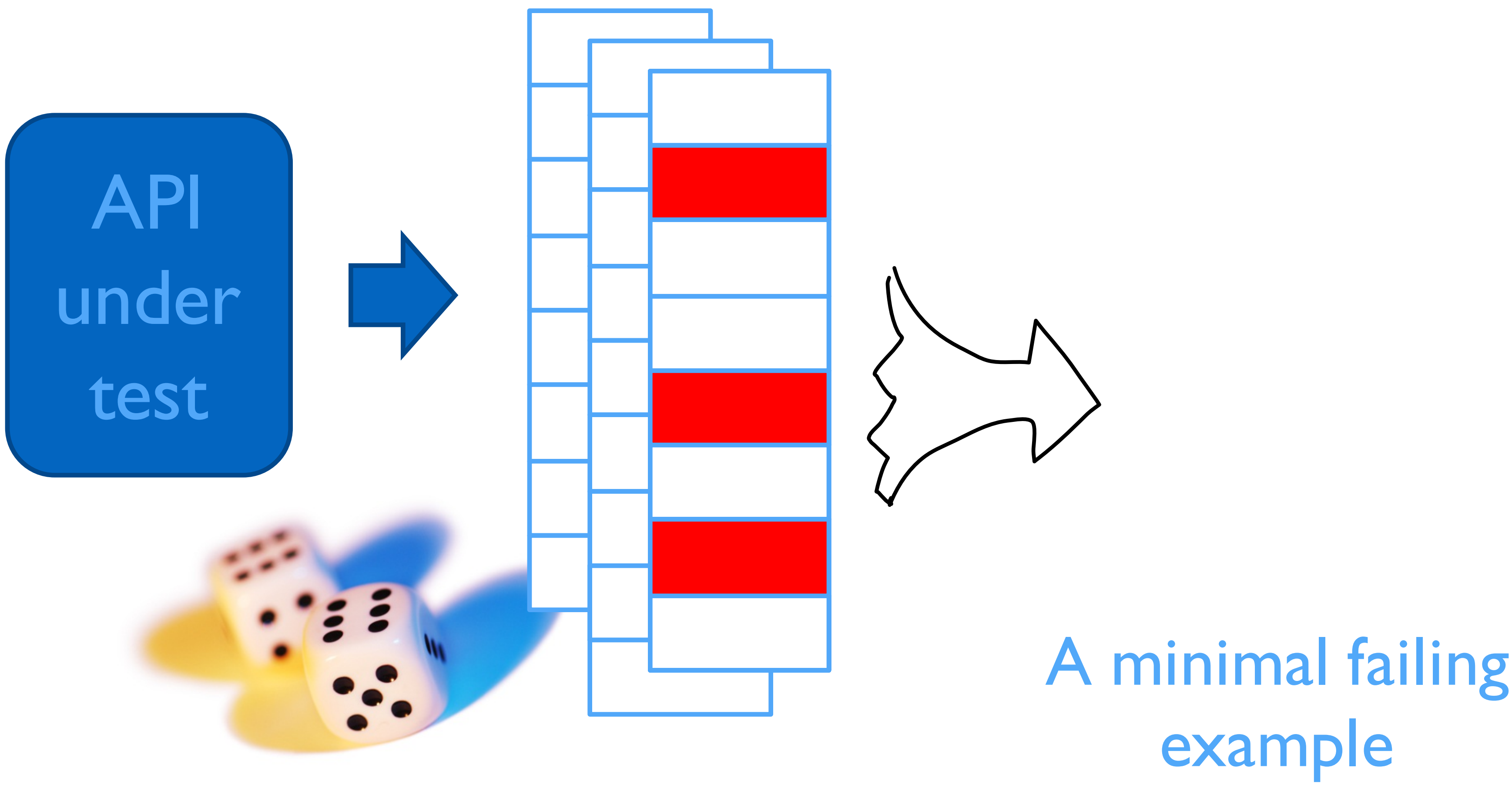
Why Generate Tests?

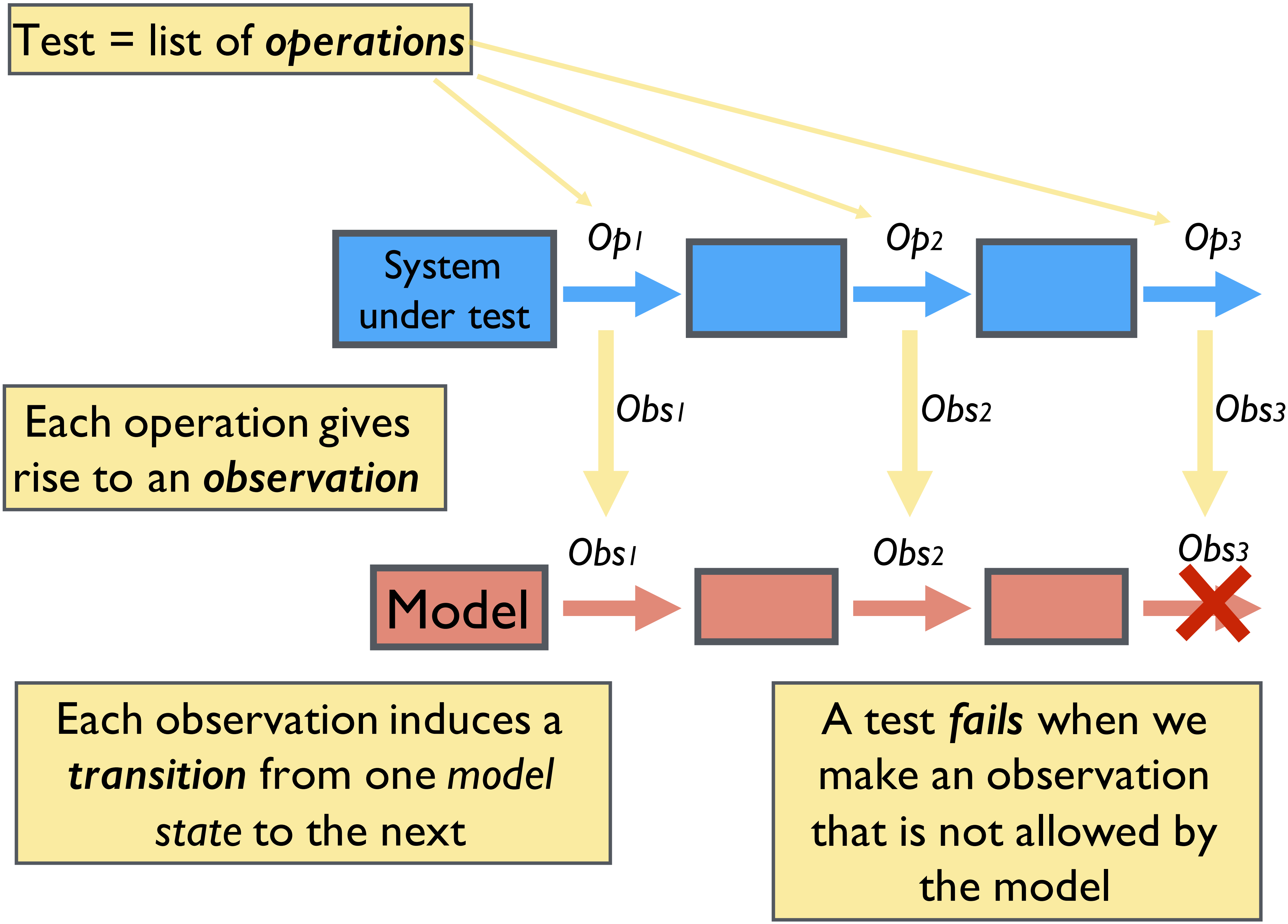
- Much wider variety!
  - Crucial for effective testing of distributed services
    - Subtle edge cases, timing dependencies, …
- More confidence!

QuickCheck

- 1999—invented by Koen Claessen and John Hughes, for Haskell
- 2006—Quviq founded marketing Erlang version
- Many extensions
- Finding deep bugs for Ericsson, Volvo Cars, Basho, etc…

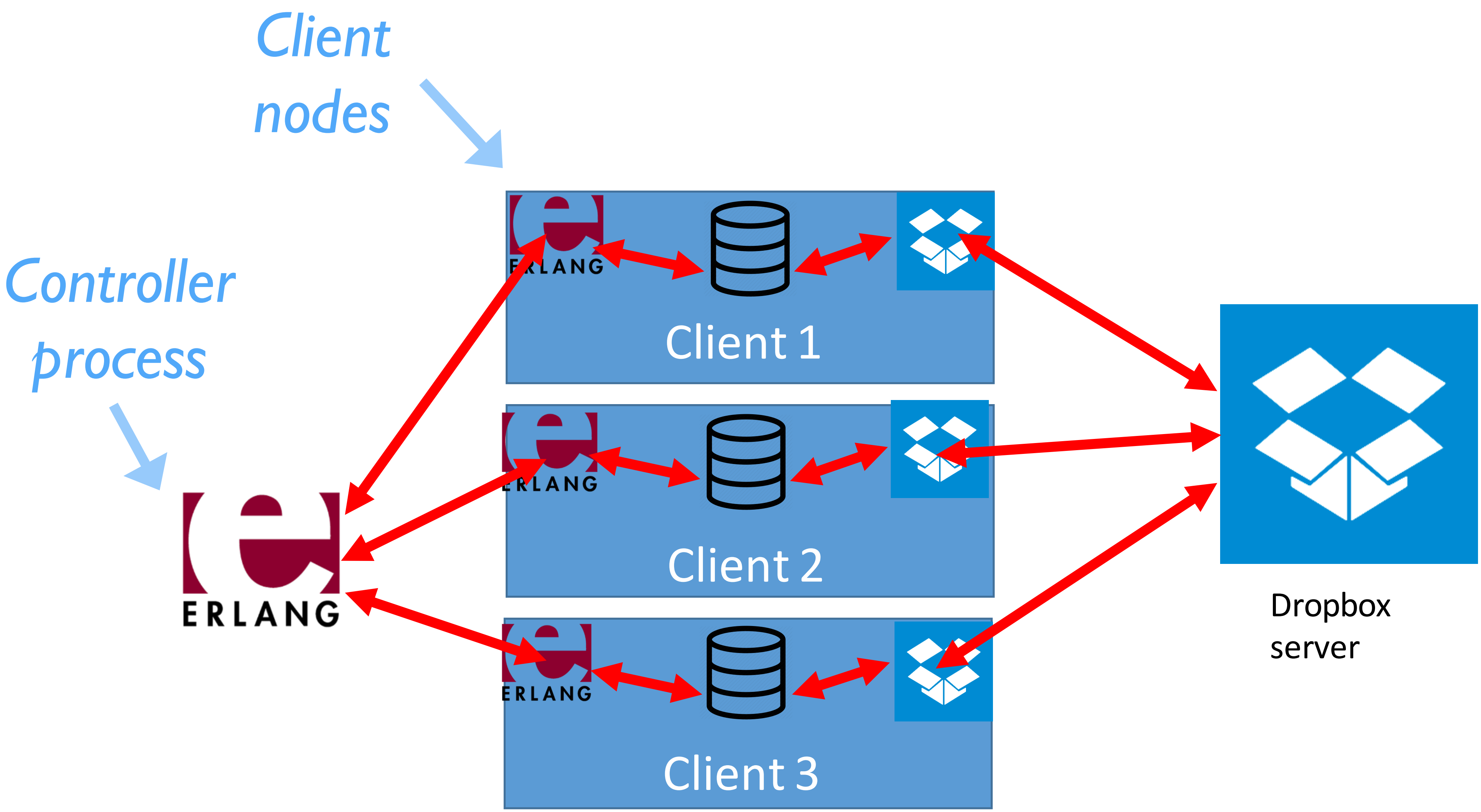
QuickCheck





Test = list of *operations*
System under test
Op_1
Op_2
Op_3
Obs_1
Obs_2
Obs_3
Each operation gives rise to an *observation*
Obs_1
Obs_2
Obs_3
Model
Each observation induces a *transition* from one *model state* to the next
A test *fails* when we make an observation that is not allowed by the model

Test Harness for Dropbox



What operations and observations do we need?

One Simplification…

- Real filesystem APIs are complex
  - Files, directories, timestamps, permissions, extended attributes, symlinks, hard links, …

- We make a small restriction…

Filesystem = single file

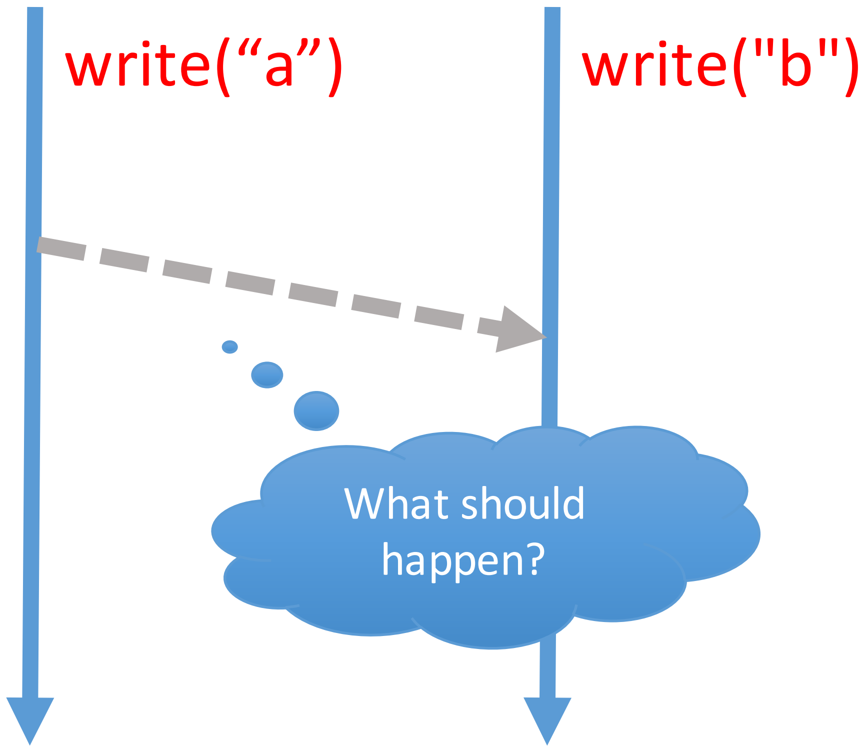
Operations	Observations
READ*N*	READ*N* $\longrightarrow$ "*current value*"
WRITE*N* ("*new value*")	WRITE*N* ("*new value*") $\longrightarrow$ "*old value*"

Use special value $\perp$ for "no file"

READ*N* $\longrightarrow$ $\perp$ means that the file is missing

WRITE*N* ($\perp$) means delete the file

Challenge #1: Conflicts



Dropbox's answer:
The first value to reach the server wins; other values are moved to *conflict files* in the same directory.

However, these conflict files may not appear for a little while!

Second try…

Operations	Observations
READN	READN ⟶ "*current value*"
WRITEN ("*new value*")	WRITEN ("*new value*") ⟶ "*old value*"
STABILIZE	STABILIZE ⟶ ("*value*", {"*conflict values*"})

Same value in the file on all clients

Same set of values in conflict files on all clients

Challenge #2: Background operations

- The Dropbox client communicates with the test harness via the filesystem.

But…

- The Dropbox client *also* communicates with the Dropbox servers!
  - Timing of these communications is unpredictable

Invisible, unpredictable activity ⟶ Nondeterminism!

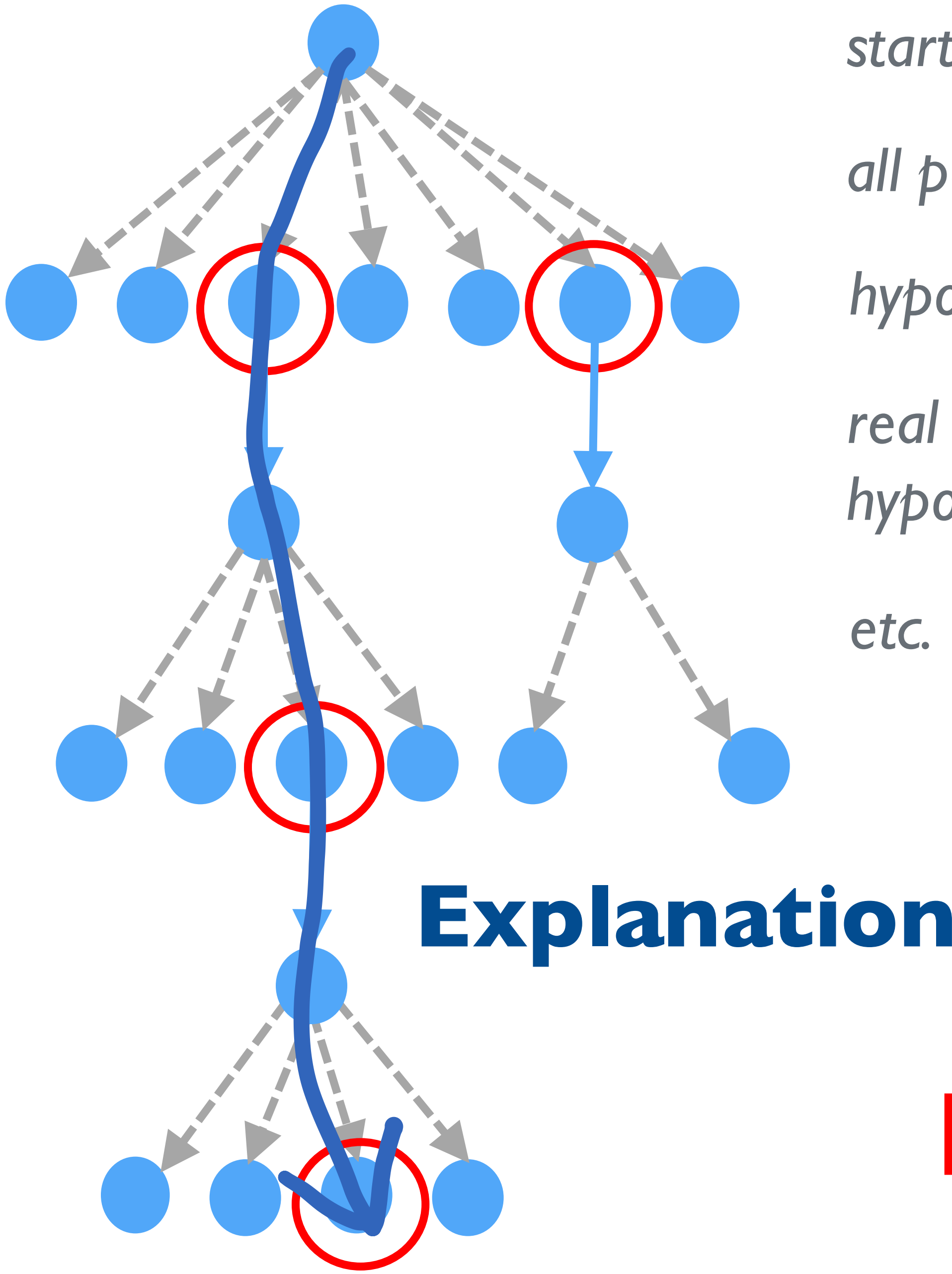
Approach

- Model the whole system state *including the (invisible) state of the server*
- Add ”conjectured observations” to the ones we actually observe when running tests…

Operations	Observations
READ$_N$	READ$_N$ $\longrightarrow$ "*current value*"
WRITE$_N$ ("*new value*")	WRITE$_N$ ("*new value*") $\longrightarrow$ "*old value*"
STABILIZE	STABILIZE $\longrightarrow$ ("*value*", {"*conflict values*"})
	UP$_N$
	DOWN$_N$

node *N* uploads its value to the server

node *N* is refreshed by the server



starting state

all possible sequences of Up/Downs

hypothetical states

real observation (invalid in most hypothetical states)

etc.

No explanation
= failing test

Example:

Test

Client 1	Client 2
WRITE 'a'	
	WRITE 'b'
READ	
	WRITE 'c'
STABILIZE	

Test

Client 1	Client 2
WRITE 'a'	
	WRITE 'b'
READ	
	WRITE 'c'
STABILIZE	

Observations

Client 1	Client 2
WRITE 'a' $\rightarrow \perp$	
	WRITE 'b' $\rightarrow$ 'a'
READ $\rightarrow$ 'b'	
	WRITE 'c' $\rightarrow$ 'b'
STABILIZE $\rightarrow$ ('c', $\emptyset$)	

Example:

Test

Client 1	Client 2
WRITE ‘a’	
	WRITE ‘b’
READ	
	WRITE ‘c’
STABILIZE	

Observations

Client 1	Client 2
WRITE ‘a’ $\rightarrow \perp$	
	WRITE ‘b’ $\rightarrow$ ‘a’
READ $\rightarrow$ ‘b’	
	WRITE ‘c’ $\rightarrow$ ‘b’
STABILIZE $\rightarrow$ (‘c’, $\emptyset$)	

Explanation

Client 1	Client 2
WRITE ‘a’ $\rightarrow \perp$	
UP	
	DOWN
	WRITE ‘b’ $\rightarrow$ ‘a’
	UP
DOWN	
	WRITE ‘c’ $\rightarrow$ ‘b’
	UP
READ $\rightarrow$ ‘b’	
DOWN	
STABILIZE $\rightarrow$ (‘c’, $\emptyset$)	

Using the model for testing

1. Generate a random sequence of operations $Op_1 \ldots Op_n$

2. Apply them to the system under test, yielding observations $Obs_1 \ldots Obs_n$

3. Calculate *all* ways of interleaving Up and Down observations with $Obs_1 \ldots Obs_n$

4. For each of these, check whether

$$\textbf{init-state} \xrightarrow{\text{(Ups and Downs)}} \xrightarrow{Obs_1} \xrightarrow{\text{(Ups and Downs)}} \xrightarrow{Obs_2} \xrightarrow{\text{(Ups and Downs)}} \cdots \xrightarrow{Obs_n}$$

is a valid sequence of transitions of the model

4. If the answer is “no” for every possible interleaving, we have found a failing test; otherwise, repeat

Model states

- *Stable value* (i.e., the one on the server)
- *Conflict set* (only ever grows)
- For each node:
  - Current *local value*
  - "FRESH" or "STALE"
  - "CLEAN" or "DIRTY"

i.e., has the global value changed since this node's last communication with the server

i.e., has the local value been written since this node was last refreshed by the server

Modeling the operations

$\boxed{\textsc{Read} \rightarrow V}$

Precondition: $LocalVal_N = V$

Effect: none

$\boxed{\textsc{Write}\ V_{new} \rightarrow V_{old}}$

Precondition: $LocalVal_N = V_{old}$

Effect: $LocalVal_N \leftarrow V_{new}$

$Clean?_N \leftarrow \text{DIRTY}$

Modeling the operations

$\boxed{\textsc{Stabilize} \rightarrow (V, C)}$

$$\begin{aligned}
\textit{Precondition:}\ & \mathit{ServerVal} = V \\
& \mathit{Conflicts} = C \\
& \text{for all } N,\ \ \mathit{Fresh?}_N = \text{FRESH} \\
& \qquad\qquad\quad\ \mathit{Clean?}_N = \text{CLEAN}
\end{aligned}$$

Effect: none

Modeling the operations

$\boxed{\textsc{Down}}$

Precondition: $Fresh?_N = \text{STALE}$

$Clean?_N = \text{CLEAN}$

Effect: $LocalVal_N \leftarrow ServerVal$

$Fresh?_N \leftarrow \text{FRESH}$

Modeling the operations

$\boxed{\text{UP}}$

$$
\begin{array}{l}
\textit{Precondition:}\ \textit{Clean?}_N = \text{DIRTY} \\
\textit{Effect:}\ \textit{Clean?}_N \leftarrow \text{CLEAN} \\
\qquad\quad \text{if } \textit{Fresh?}_N = \text{FRESH} \text{ then} \\
\qquad\qquad \text{if } \textit{LocalVal}_N \neq \textit{ServerVal} \text{ then} \\
\qquad\qquad\quad \textit{Fresh?}_{N'} \leftarrow \text{STALE} \quad \text{for all } N' \neq N \\
\qquad\qquad\quad \textit{ServerVal} \leftarrow \textit{LocalVal}_N \\
\qquad\quad \text{else} \\
\qquad\qquad \text{if } \textit{LocalVal}_N \neq \textit{ServerVal} \text{ then} \\
\qquad\qquad\quad \textit{Conflicts} \leftarrow \textit{Conflicts} \cup \{\textit{LocalVal}_N\}
\end{array}
$$

Dealing with deletion

- Deletion can easily be added to the model:

  DELETE$_N$ just means WRITE$_N$ $\perp$

- Try adding this and run some tests…

Still not quite right…

Client 1	Client 2	Client 3
WRITE ‘a’ $\rightarrow \perp$		
	READ $\rightarrow$ ‘a’	
WRITE $\perp \rightarrow$ ‘a’		
	READ $\rightarrow \perp$	
		WRITE ‘b’ $\rightarrow$ ‘a’
READ $\rightarrow$ ‘b’		

Write “a” on client 1

Client 2 sees 1’s value

Delete the file

2 sees “missing” (so stable value at server is “missing”)

Now client 3 writes "b". Observes previous value ‘a’ (n.b.: not $\perp$).

We now observe "b", so the stable value on the server must have been overwritten, despite the fact that ‘b’ was in conflict

Refining the specification…

- Add special cases for “missing” in Up and Down actions:
  - When “missing” encounters another value during an up or down, the other value always wins
  - I.e., when a write and a delete conflict, the delete gets undone

$\boxed{\textsc{Up}}$

Precondition: $\mathit{Clean?}_N = \text{DIRTY}$

Effect: $\mathit{Clean?}_N \leftarrow \text{CLEAN}$

$$
\begin{array}{l}
\text{if } \mathit{Fresh?}_N = \text{FRESH then} \\
\quad \text{if } \mathit{LocalVal}_N \neq \mathit{ServerVal} \text{ then} \\
\quad\quad \mathit{Fresh?}_{N'} \leftarrow \text{STALE} \quad \text{for all } N' \neq N \\
\quad\quad \mathit{ServerVal} \leftarrow \mathit{LocalVal}_N \\
\text{else} \\
\quad \text{if } \mathit{LocalVal}_N \notin \{\mathit{ServerVal}, \perp\} \text{ then} \\
\quad\quad \mathit{Conflicts} \leftarrow \mathit{Conflicts} \cup \{\mathit{LocalVal}_N\}
\end{array}
$$

Surprises…

Surprise: Dropbox can (briefly) delete a newly created file…

	Client 1	Client 2	
Create file	WRITE ‘a’ $\to \bot$		
Delete it	WRITE $\bot \to$ ‘a’		
		WRITE ‘b’ $\to$ ‘a’	Observe creation
Create it again	WRITE ‘c’ $\to \bot$		
File is gone!	READ $\to \bot$		

Timing is critical! → Add SLEEP operations in tests

Surprise: Dropbox can (permanently) re-create a deleted file…

	Client 1
Create file	$\textsc{Write}$ 'b' $\rightarrow \perp$
Delete it	$\textsc{Write}$ $\perp \rightarrow$ 'b'
File is back!	$\textsc{Read} \rightarrow$ 'b'

(other clients idle)

(Again, timing is critical)

Surprise: Dropbox can lose data

Client 1	Client 2
	WRITE 'b' $\rightarrow \perp$
WRITE 'a' $\rightarrow$ 'b' READ $\rightarrow$ 'a'	
STABILIZE $\rightarrow$ { ('a', $\emptyset$),	('b', $\emptyset$) }

Create file

Overwrite it

(Again, timing is important)

New value persists on client 1

Old value persists on client 2!

Client 1 believes it is still Fresh, so if we later write a new value on client 2, it will silently overwrite client 1's value and no conflict file will be created

Wrapping up…

What did we do?

- Tested a non-deterministic system by *searching for explanations* using a model with hidden actions

- Used QuickCheck's minimal failing tests to *refine* the model, until it matched the intended behaviour

- Now minimal failing tests reveal *unintended* system behaviour

What do Dropbox say?

- The synchronization team has reproduced the buggy behaviours
- They're *rare* failures which occur under very special circumstances
- They're developing fixes

Synchronization is subtle!

There's much more to do…

- Add directories!
  - Directories and files with the same names
  - Conflicts between deleting a directory and writing a file in it
  - …

- More file synchronizers!

Thank you!

(Any questions?)

CHALMERS

Penn
UNIVERSITY of PENNSYLVANIA

QuviQ