Testing ejabberd with QuickCheck

John Hughes, Ulf Norell

Chalmers University/Quviq

Jérôme Sautret

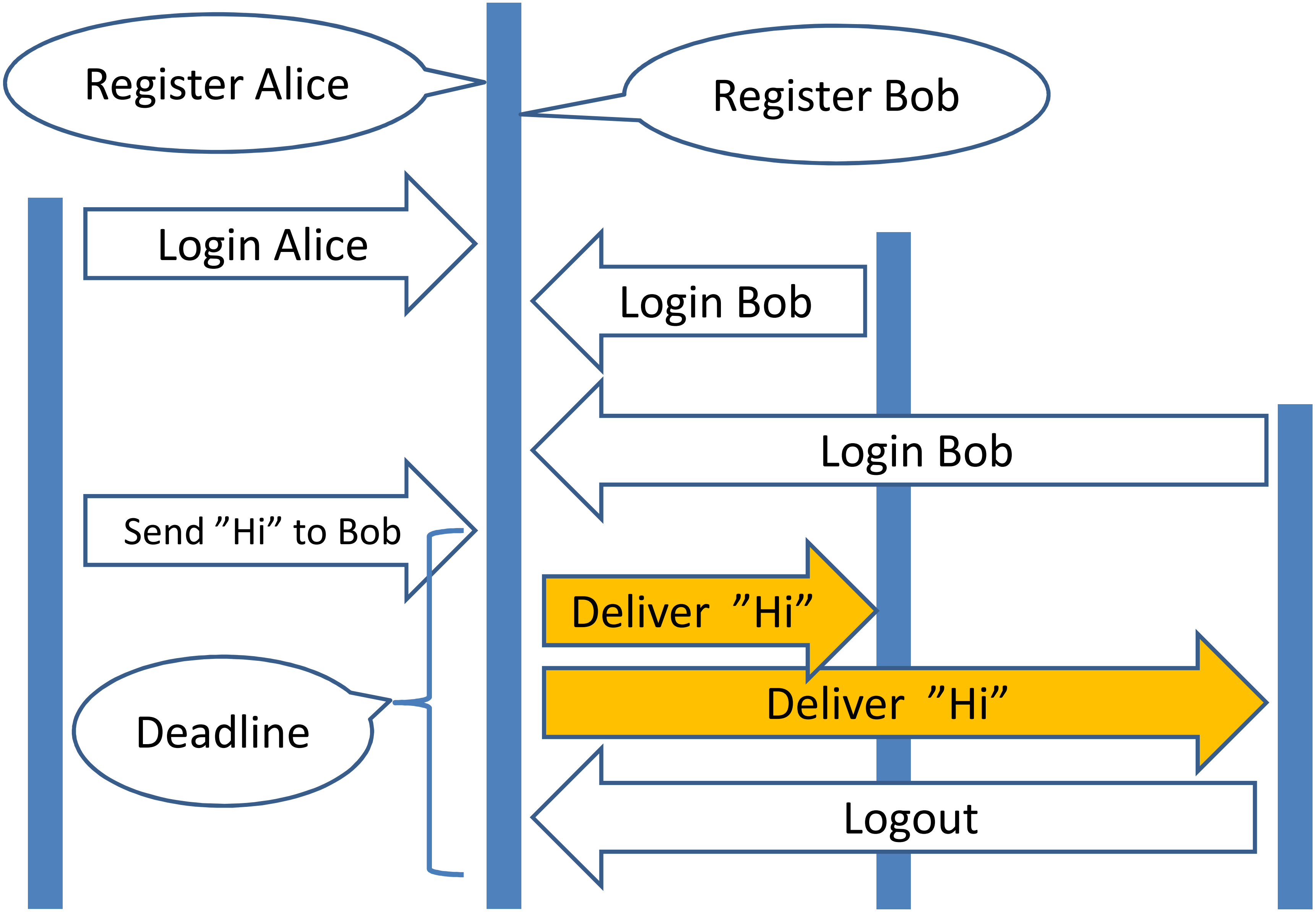
Process One

ejabberd

- Instant Messaging server
- XMPP-based
  - Runs 38% of XMPP servers
- Forthcoming release is a major refactoring
  - Testing is a priority!

But why is it interesting?

XMPP Server



Register Alice

Register Bob

Login Alice

Login Bob

Login Bob

Send ”Hi” to Bob

Deliver ”Hi”

Deliver ”Hi”

Deadline

Logout

Asynchrony!

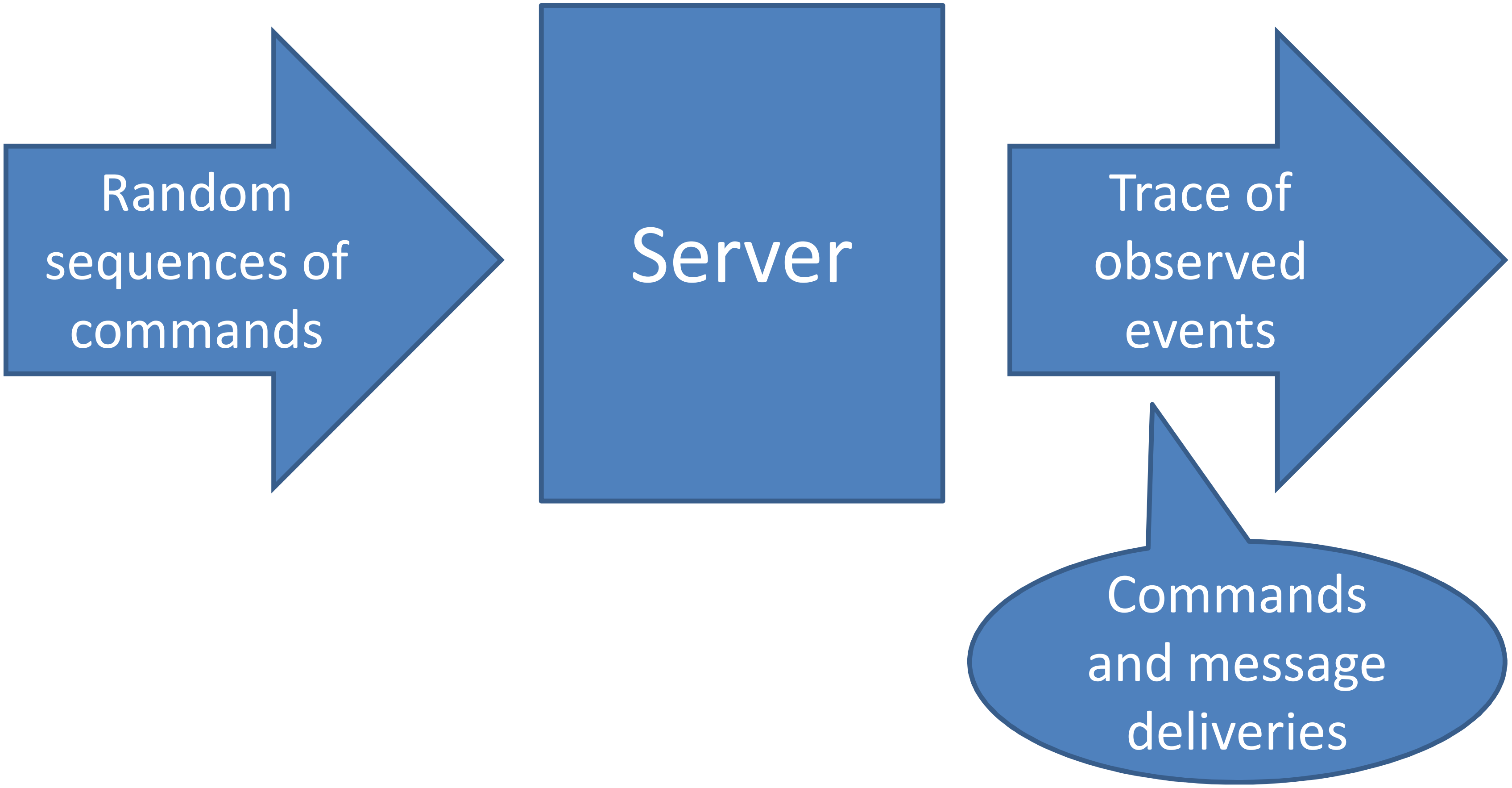
Makes testing hard!

—a common problem!

But we succeeded…

Three problems and their solutions

Our Approach using QuickCheck


Random sequences of commands

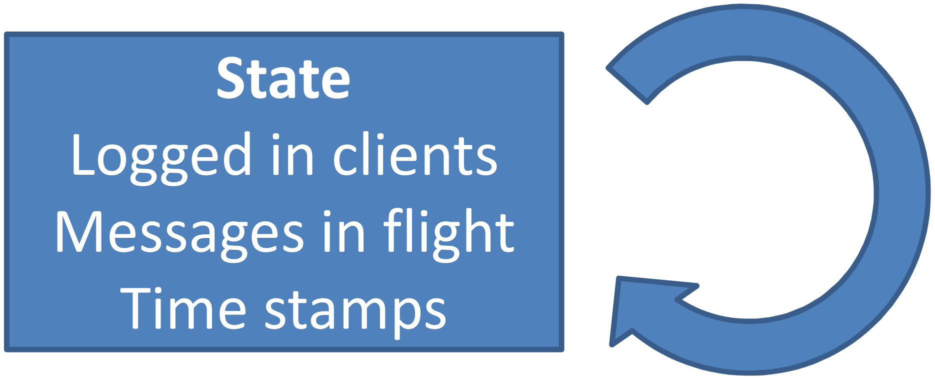
Server

Trace of observed events

Commands and message deliveries

Trace Verification

- Examples:
  - Is a message send followed by appropriate receives within the right deadline?
  - Are messages delivered uncorrupted?



State

Logged in clients

Messages in flight

Time stamps

Is each trace event acceptable?

Transform state appropriately.

Do we end in an acceptable state? (e.g. no messages in flight)

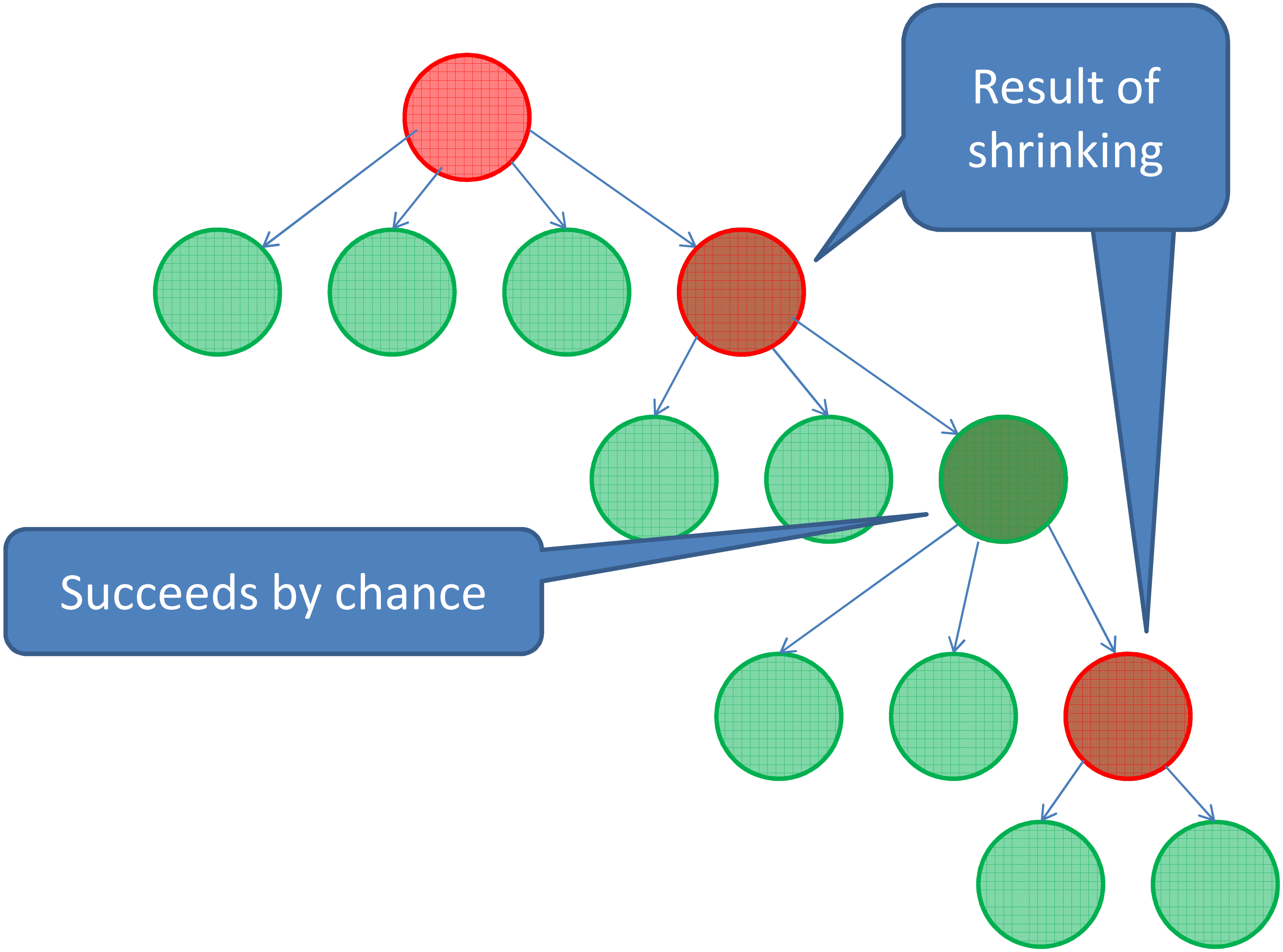
Tests were failing…

- But only longer tests—about 60 commands!
  - Impossible to diagnose!

- Random tests are like failures from the field
  - Lots of irrelevant stuff!

- First task: *simplify* the failing test…
  - And QuickCheck does!

How Shrinking Works



Result of shrinking

Succeeds by chance

Problem #1

The *same test* may succeed or fail in different runs!

A *real pain* when you’re trying to simplify a test case

Solution #1

Repetition!

- Should we consider a test to pass if it *always* passes, or if it *sometimes* passes?

- ?ALWAYS(N,Property) or ?SOMETIMES(N,Property)

?SOMETIMES(10,...)

- Search for test cases which *fail repeatably*...
- ...yields failing tests of about 30 commands!

PROGRESS!

Problem #2

Shrinking leaves lots of commands

…and many seem to do very little
(e.g. update presence information)

Insight #2

Duff commands are needed *to take time*

…because tests fail when timeouts are exceeded

Solution #2

Shrink commands to sleeps!

Shrinks to the shortest sleep necessary to provoke failure

Any command → {call,timer,sleep,[choose(1,1000)]}

- Tests shrank to small counterexamples!

- Found several bugs in the *trace verifier*
  - Didn’t recognise that unregister(Bob) also logs Bob out!

But tests still failed when they were fixed!

Problem #3

Event time-stamps are recorded inaccurately.

Sometimes even event *order* is recorded inaccurately!

The trace verifier must cope with this…

Problem #3’

The trace verifier becomes *horribly* complex

- We don’t *know* the state
  - We don’t know the event order!
  - We must represent *many possible* states…

…exponentially many

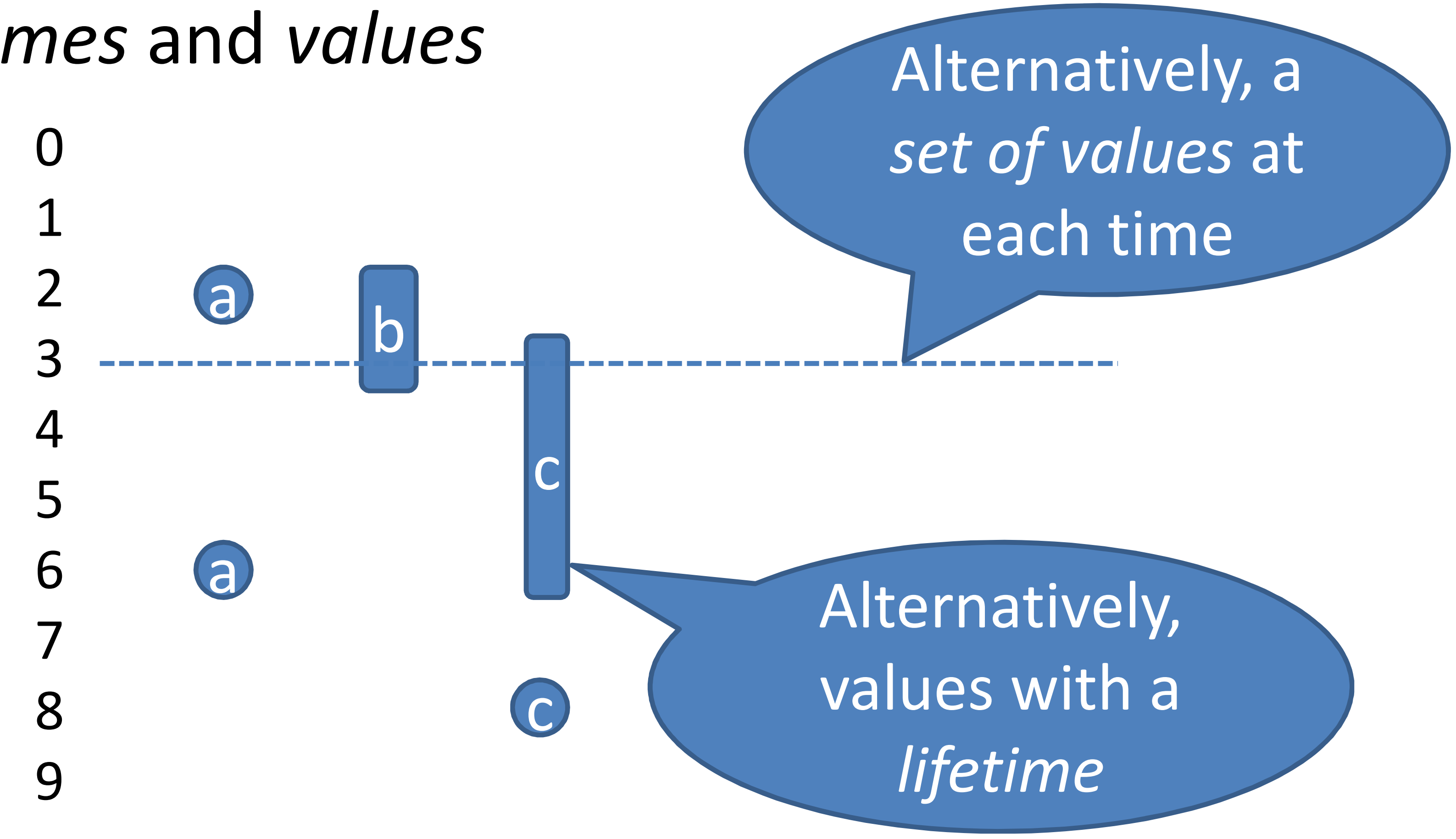
Solution #3

A new datatype of *temporal relations*, used to represent temporal information

Details in our paper at *Automation of Software Test* 2010.

Temporal Relations

- A *temporal relation* is a relation between *times* and *values*



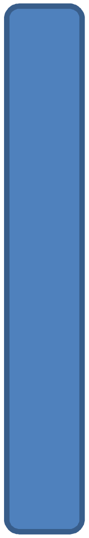
Alternatively, a *set of values* at each time

Alternatively, values with a *lifetime*

Example

Events as a temporal relation

10	{login,alice,laptop}
11	{login,bob,desktop}
15	{login,bob,phone}
26	{send,alice,bob,”Hi”}
31	{delivery,alice,bob,desktop,”Hi”}
33	{logout,bob,phone}



{logged_in,
bob,
phone}

States as a temporal relation

Stateful Relations

```
LoggedIn = stateful(fun logging_in/1,
                    fun logging_out/2,
                    Events)
```

- Enter a *list of states* on a matching event

```
logging_in({login,Uid,ResourceId}) ->
  [{logged_in,Uid,ResourceId}].
```

- Transform a state on a matching event

```
logging_out({logged_in,Uid,Rid},Ev) ->
  case Ev of
    {logout,Uid,Rid} -> [];
    {unregister,Uid} -> []
  end.
```

Relational Operations

```
MessageCreations =
  map(fun message_creation/1,
      product(Events,LoggedIn))
```

Apply this function…

…to every pair of an event and logged-in user

- On matching events, create a message-in-flight

```
message_creation({{send,From,To,Msg},
                  {logged_in,To,Rid}}) ->
  {message,From,To,Rid,Msg}.
```

Messages as a Temporal Relation

```
Messages = stateful(fun start_message/1,
                    fun stop_message/2,
               union(MessageCreations,
                     Events))
```

```
start_message({message,From,To,R,Msg}) ->
  [{message,From,To,R,Msg}].
```

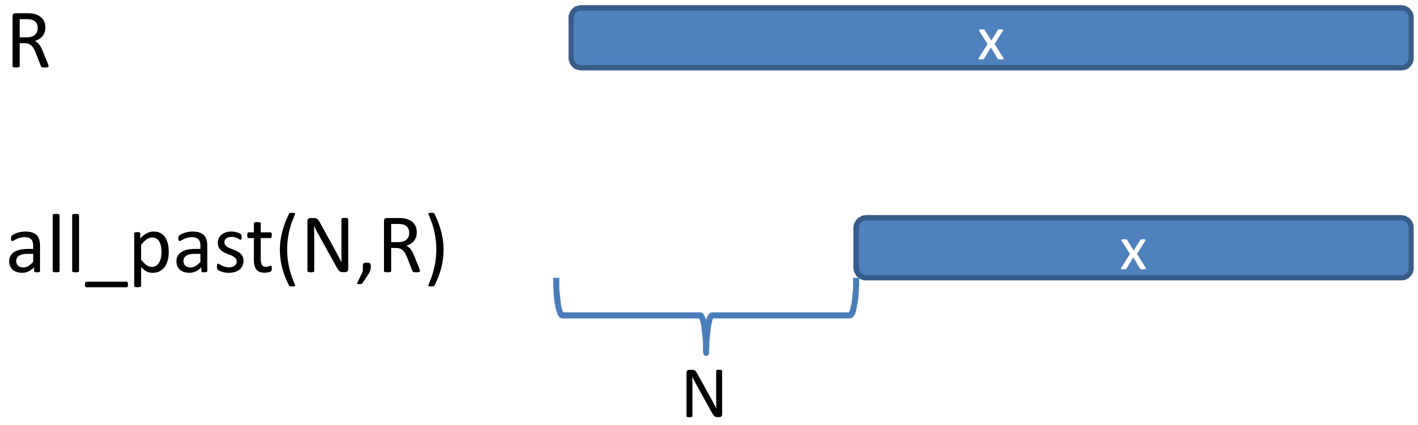
```
stop_message({message,From,To,R,Msg},Ev) ->
  case Ev of
    {delivery,From,To,R,Msg} -> [];
    {logout,To,R}            -> [];
    {unregister,To}          -> []
  end.
```

Temporal Operations

- all_past(N,R) contains *x* at time *t*

  ⇔ R contains *x* at *t*

  and at the N previous times



Message Delivery

- A relation containing messages overdue for delivery…

```
Overdue = all_past(100,Messages)
```

  - In flight for the last 100 ms

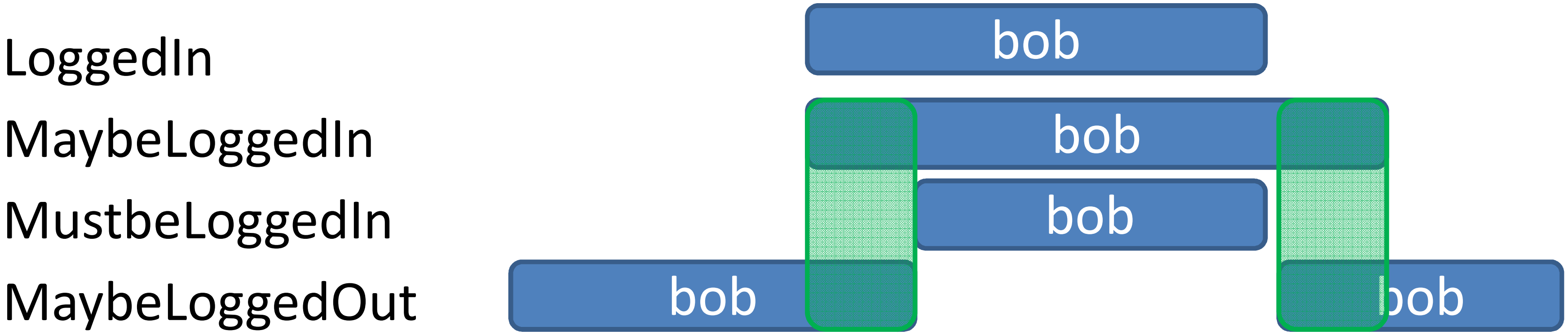
- In the test case, check

```
is_empty(Overdue)
```

Timing Uncertainty

- If a user logs in on a second resource *just before* a message is sent, it need not be delivered…login may not be complete

```
MustbeLoggedIn  = all_past(15,LoggedIn),
MaybeLoggedOut  = complement(MustbeLoggedIn),
MaybeLoggedIn   = any_past(15,LoggedIn)
```



LoggedIn

MaybeLoggedIn

MustbeLoggedIn

MaybeLoggedOut

- Relational trace verifier is much more modular and declarative than the state-machine one
  - Messages *may* be delivered after a logout—for a short time
    - State machine: 26 LOC at 4 separate locations
    - Relational: MaybeLoggedIn
  - Message delivery deadline
    - 5 places in state-machine spec
    - 1 place in relational spec
- And it works!

Bugs in ejabberd

- Send M to Bob & Bob logs in close together
  - M *should* be delivered to Bob
  - M only delivered on Bob’s *next* login

- Send M to Bob & Bob logs out close together
  - M *should* be delivered to Bob now, or on next login
  - M may be lost altogether

Conclusions

- Automating testing of asynchronous systems is *hard…*
- …but doable, and the ideas in this talk can help.

More information

- Paper on temporal relations at *Automation of Software Test* 2010
- Try QuickCheck Mini (free version, on your CD)