Towards Property-Based Consistency Verification

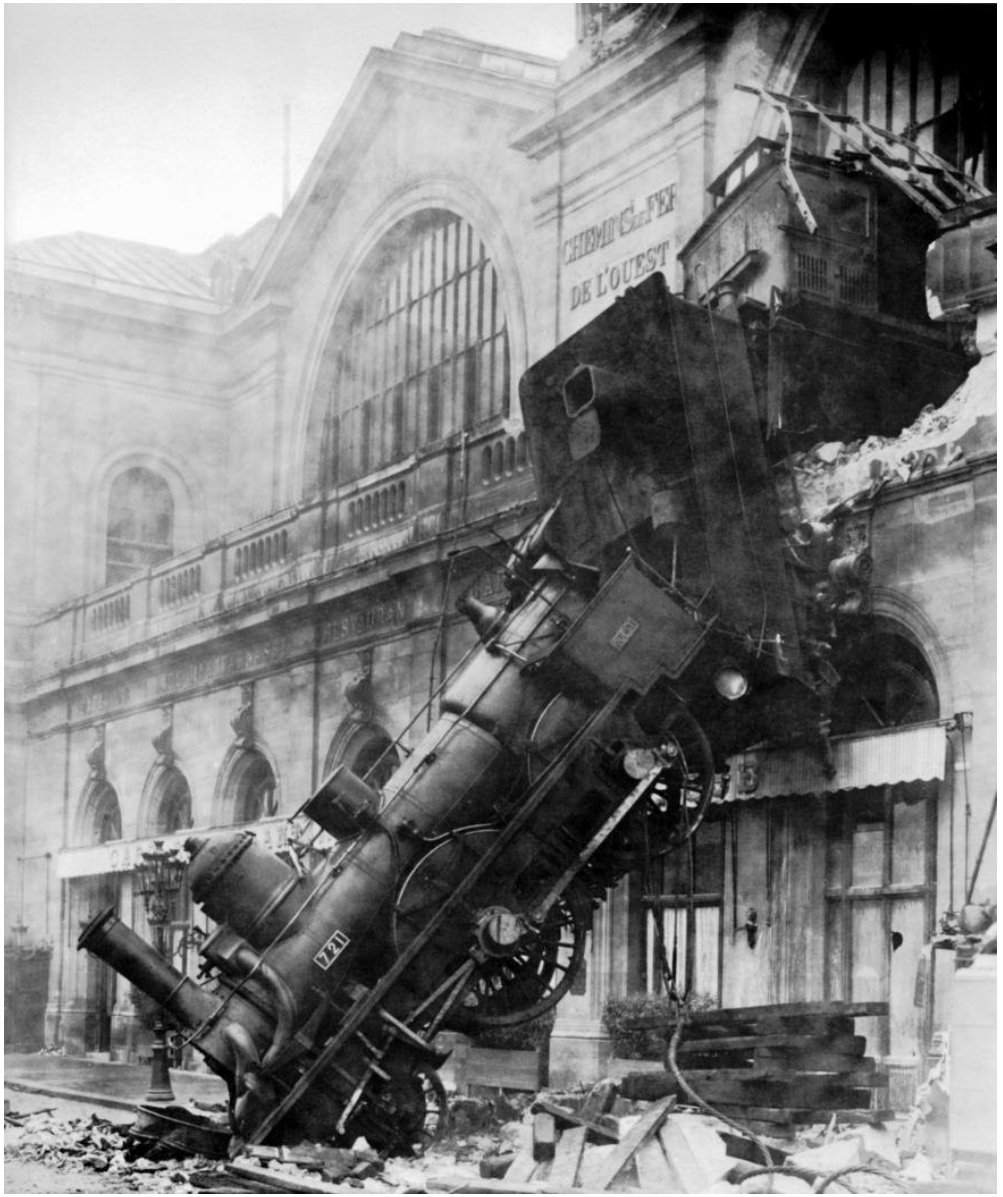
Paolo Viotti
EURECOM

Christopher Meiklejohn
Université catholique de Louvain

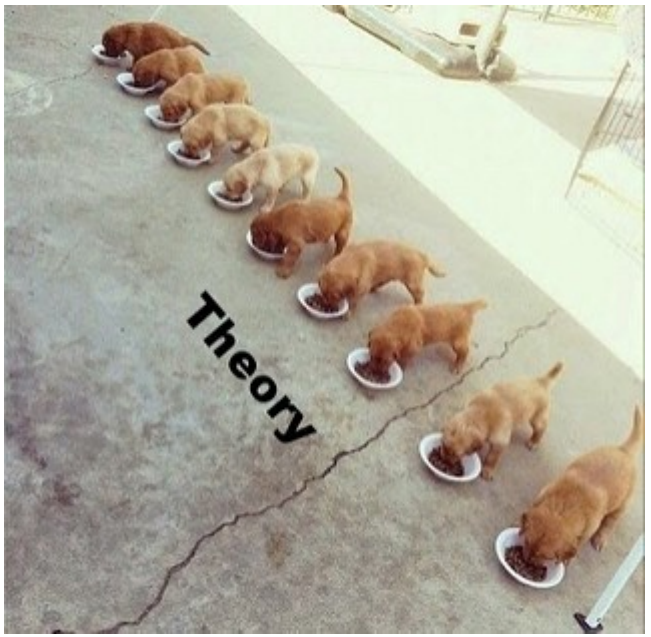
Marko Vukolić
IBM Research - Zurich

PaPoC - April 18-21, 2016, London

Testing can be useful.



Testing distributed systems can be especially useful (and fun!).



Consistency implementations

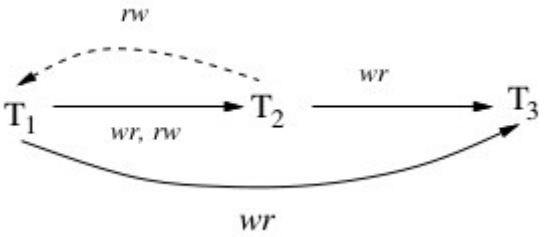
*: terms and conditions may apply.

Verifying consistency: state of the art

- Strong consistency checkers
  - Binary decision problem

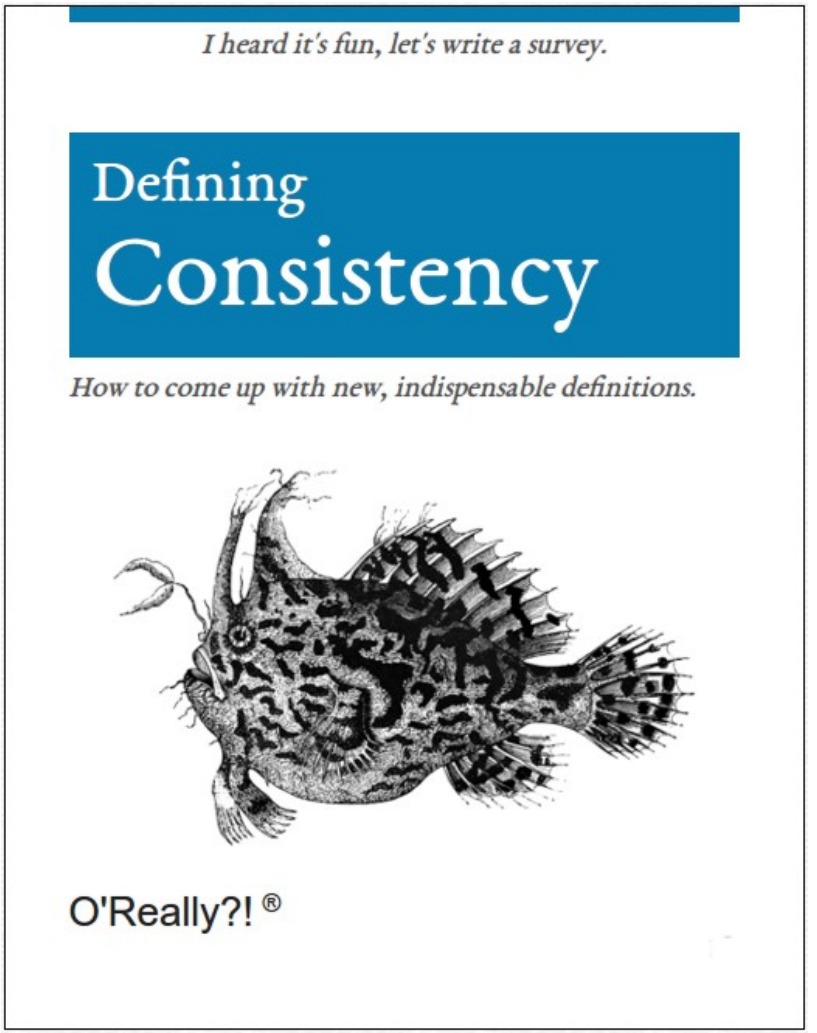
- Staleness
  - In EC systems

- Precedence graph
  - Transactional systems



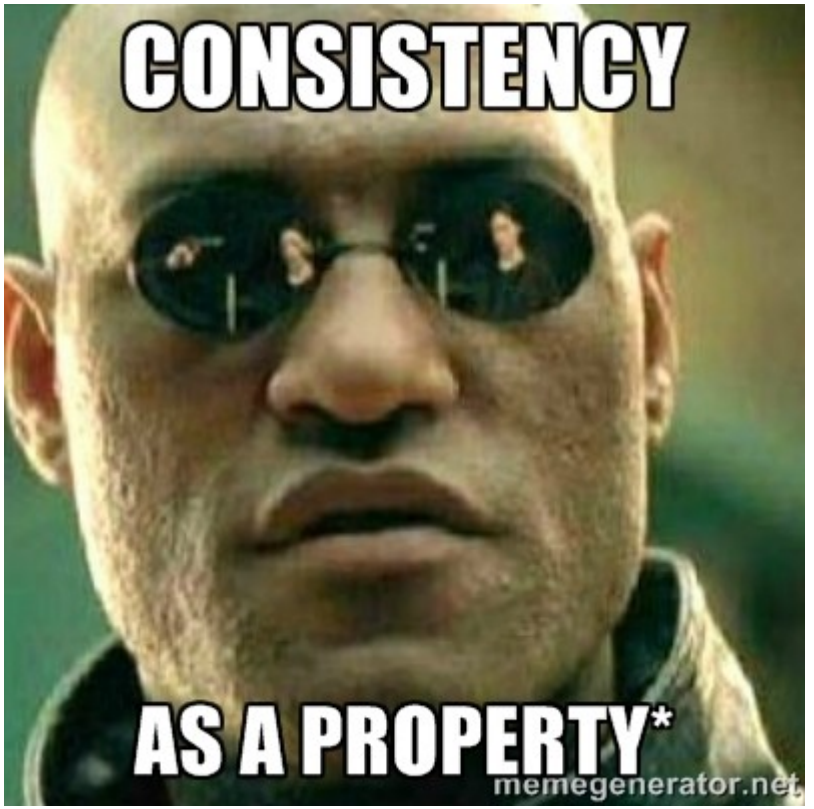
Definitions of consistency

- Most of the time:
  - Informal, imprecise
  - Ad-hoc, for specific settings
  - Conflicting
  - Incompatible, incomparable



Declarative consistency

What if we could define...

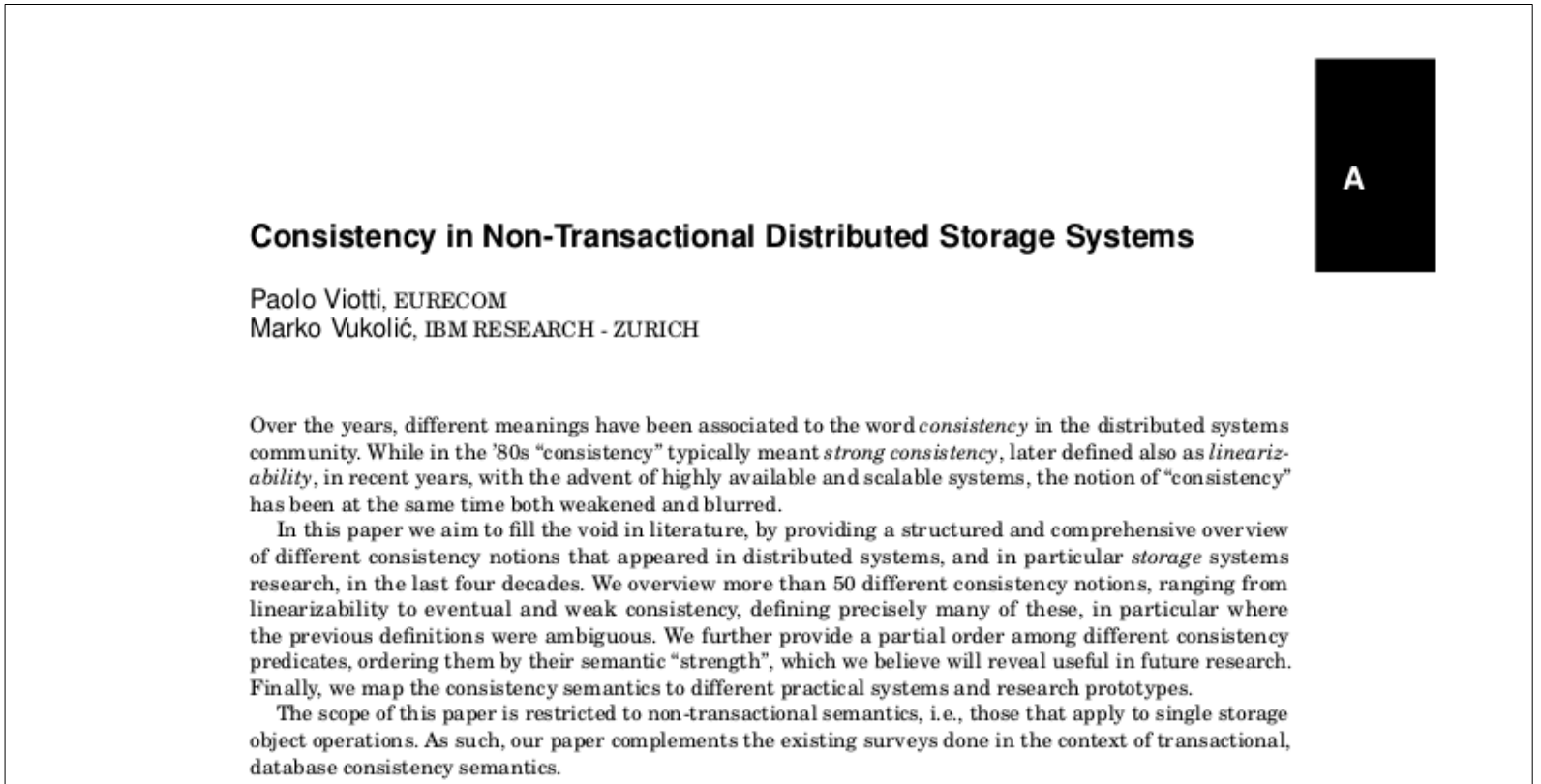


*: for real.

Declarative consistency

P. Viotti, M. Vukolić. *Consistency in non-transactional distributed storage systems.* ACM Comput. Surv. (to appear)

- extend and refine model in [Burckhardt 2014]
- (also available as pre-print on ArXiv)



A

Consistency in Non-Transactional Distributed Storage Systems

Paolo Viotti, EURECOM
Marko Vukolić, IBM RESEARCH - ZURICH

Over the years, different meanings have been associated to the word *consistency* in the distributed systems community. While in the '80s "consistency" typically meant *strong consistency*, later defined also as *linearizability*, in recent years, with the advent of highly available and scalable systems, the notion of "consistency" has been at the same time both weakened and blurred.

In this paper we aim to fill the void in literature, by providing a structured and comprehensive overview of different consistency notions that appeared in distributed systems, and in particular *storage* systems research, in the last four decades. We overview more than 50 different consistency notions, ranging from linearizability to eventual and weak consistency, defining precisely many of these, in particular where the previous definitions were ambiguous. We further provide a partial order among different consistency predicates, ordering them by their semantic "strength", which we believe will reveal useful in future research. Finally, we map the consistency semantics to different practical systems and research prototypes.

The scope of this paper is restricted to non-transactional semantics, i.e., those that apply to single storage object operations. As such, our paper complements the existing surveys done in the context of transactional, database consistency semantics.

Declarative consistency

- Consistency semantics as logic *predicates* about ***ordering*** and mutual ***visibility*** of events
- Composable, extensible
- Implementation agnostic

Declarative consistency

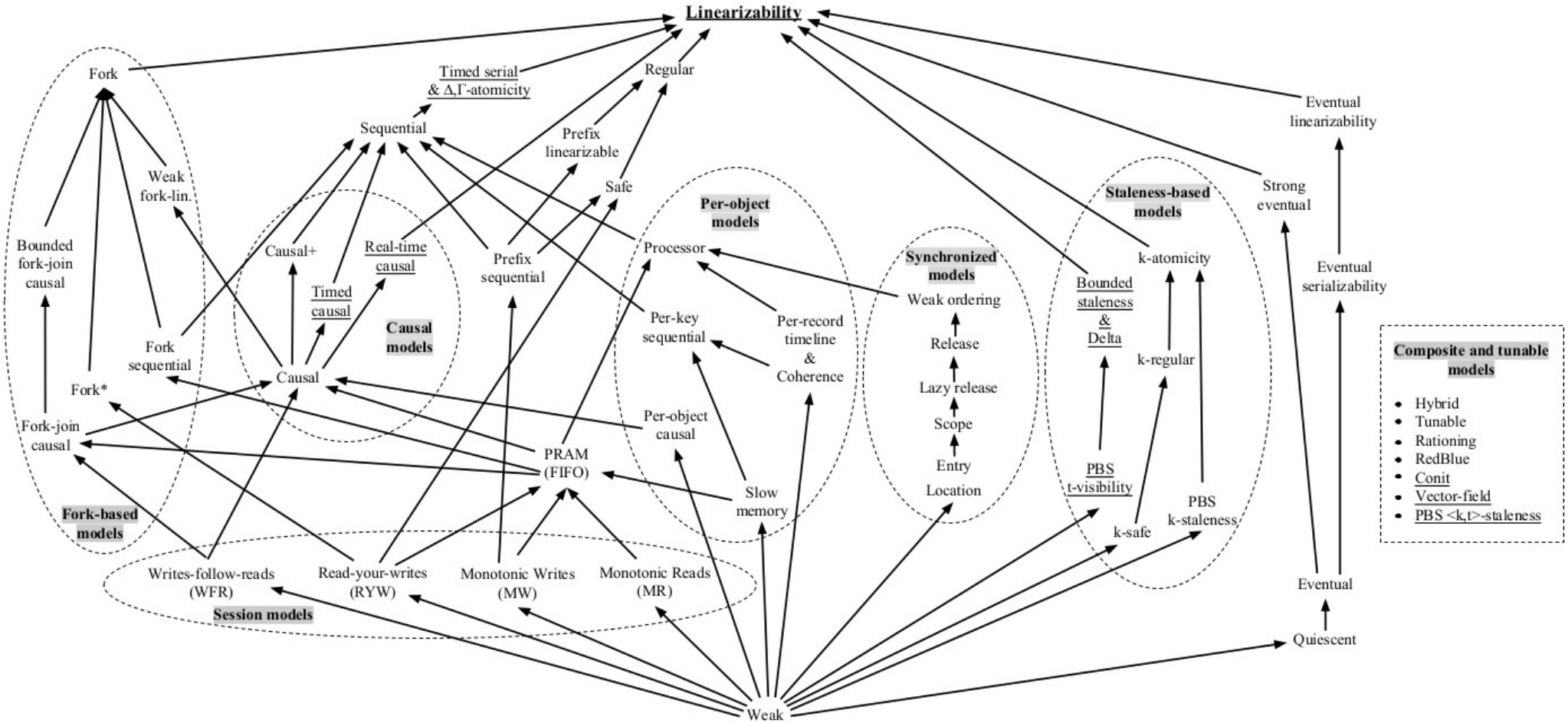
Entity	Description
Operation (op)	Single operation. Includes: process and object id, type, input and output values, start and end time.
History (H)	The set of operations of an execution. Described by: returns-before partial order, same-session and same-object equivalence relations.
Visibility (vis)	Acyclic partial order on operations. Accounts for propagation of write operations.
Arbitration (ar)	Total order on operations. Specifies how the system resolves conflicts.

Declarative consistency

LINEARIZABILITY($\mathcal{F}$)	SINGLEORDER $\wedge$ REALTIME $\wedge$ RVAL($\mathcal{F}$)
SINGLEORDER	$\exists H' \subseteq \{op \in H : op.oval = \nabla\} : vis = ar \setminus (H' \times H)$
REALTIME	$rb \subseteq ar$
REGULAR($\mathcal{F}$)	SINGLEORDER $\wedge$ REALTIMEWRITES $\wedge$ RVAL($\mathcal{F}$)
SAFE($\mathcal{F}$)	SINGLEORDER $\wedge$ REALTIMEWRITES $\wedge$ SEQRVAL($\mathcal{F}$)
REALTIMEWRITES	$rb\vert_{wr \to op} \subseteq ar$
SEQRVAL($\mathcal{F}$)	$\forall op \in H : Concur(op) = \emptyset \Rightarrow op.oval \in \mathcal{F}(op, cxt(A, op))$
EVENTUALCONSISTENCY($\mathcal{F}$)	EVENTUALVISIBILITY $\wedge$ NOCIRCULARCAUSALITY $\wedge$ RVAL($\mathcal{F}$)
EVENTUALVISIBILITY	$\forall a \in H, \forall [f] \in H/\approx_{ss} : \vert\{b \in [f] : (a \xrightarrow{rb} b) \wedge (a \not\xrightarrow{vis} b)\}\vert < \infty$
NOCIRCULARCAUSALITY	$acyclic(hb)$
STRONGCONVERGENCE	$\forall a, b \in H\vert_{rd} : vis^{-1}(a)\vert_{wr} = vis^{-1}(b)\vert_{wr} \Rightarrow a.oval = b.oval$
STRONGEVENTUALCONS.($\mathcal{F}$)	EVENTUALCONSISTENCY($\mathcal{F}$) $\wedge$ STRONGCONVERGENCE
QUIESCENTCONSISTENCY($\mathcal{F}$)	$\vert H\vert_{wr}\vert < \infty \Rightarrow \exists C \in \mathcal{C} : \forall [f] \in H/\approx_{ss} : \vert\{op \in [f] : op.oval \notin \mathcal{F}(op, C)\}\vert < \infty$
PRAM	$so \subseteq vis$
SEQUENTIALCONSISTENCY($\mathcal{F}$)	SINGLEORDER $\wedge$ PRAMCONSISTENCY $\wedge$ RVAL($\mathcal{F}$)

...etc. (40+ definitions)

Declarative consistency



Property-based testing

Automatic testing technique focused on *properties*

- Input: properties, input format
- Abstract away test case generation
  - generates and runs random test cases
- If an error is found, test case is shrunk to a minimal complexity

[Hughes and Claessen 2000]
tools: QuickCheck, PropEr, etc...

Property-based testing

A simple example (in Erlang):

Function to reverse a list

```
reverse([]) ->
    [];
reverse([X|Xs]) ->
    reverse(Xs) ++ [X].
```

Property:

for every list Xs, reverse(reverse(Xs)) == Xs

```
prop_reverse() ->
    ?FORALL(Xs, list(int()),
            reverse(reverse(Xs))==Xs).
```

```
3> proper:quickcheck(qc_test:prop_reverse()).
....................................................
OK: Passed 100 test(s).
true
```

Property-based consistency verification

- Property-based paradigm to verify consistency
  - ✔ declarative consistency semantics as properties

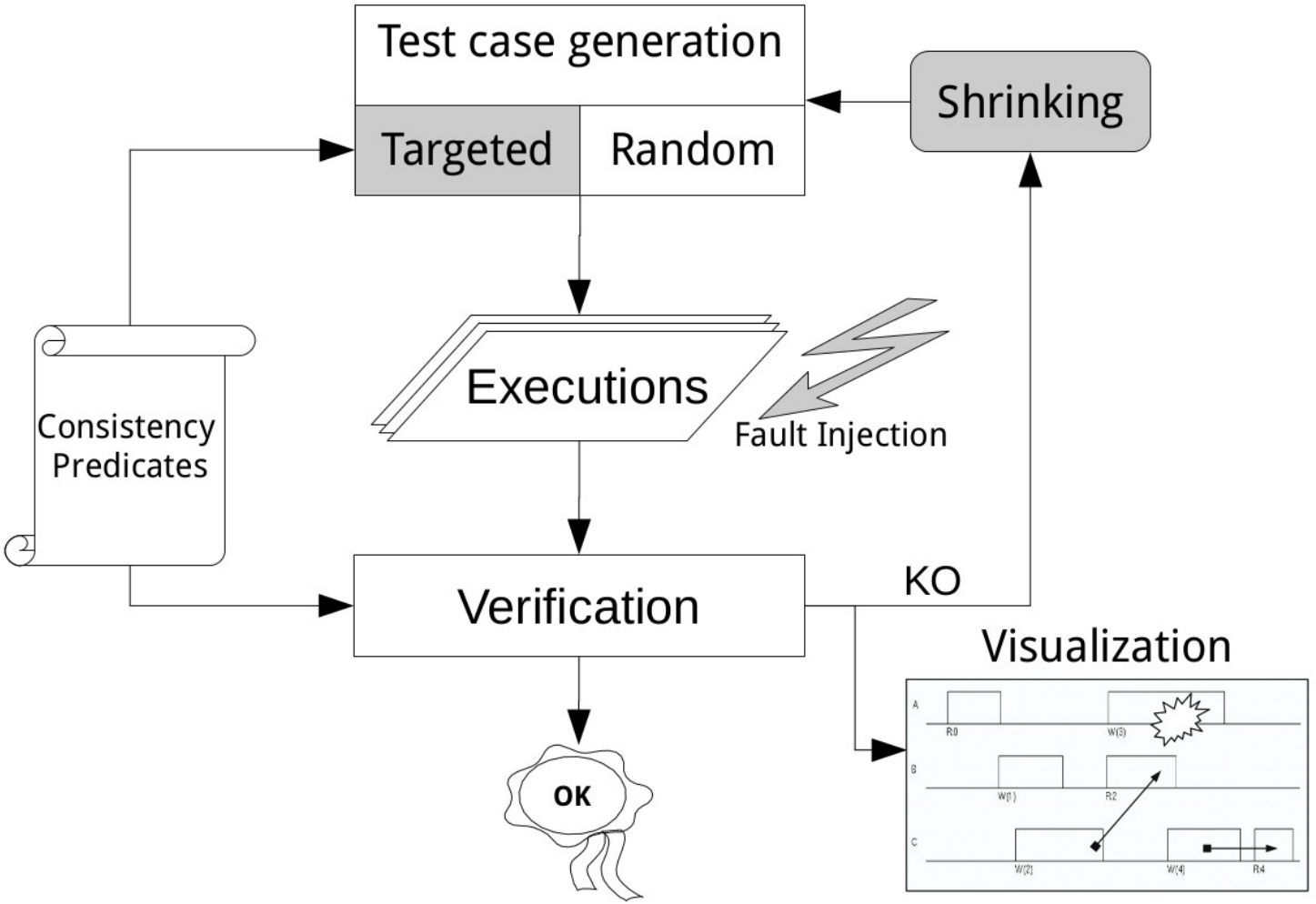
- ...but distributed systems:
  - deal with non-determinism
    - ➔ shrinking test cases may be challenging
  - are cumbersome and slow to test
  - are heterogeneous

Conver

A property-based consistency verification framework

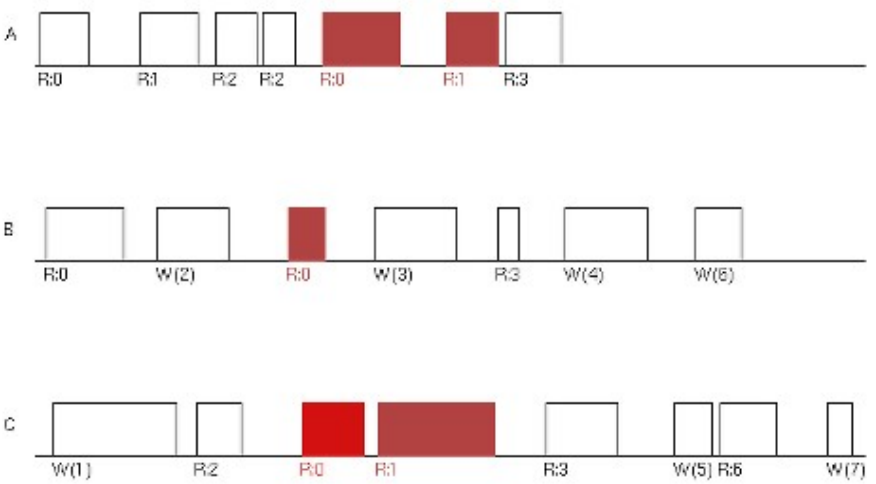
- Prototype in Erlang
  - https://github.com/pviotti/conver
- Currently, it can verify
  - 7 consistency models
  - 2 data stores (Riak, ZooKeeper)
    - easily extensible

Conver


Test case generation
Targeted
Random
Shrinking
Executions
Fault Injection
Consistency Predicates
Verification
KO
OK
Visualization

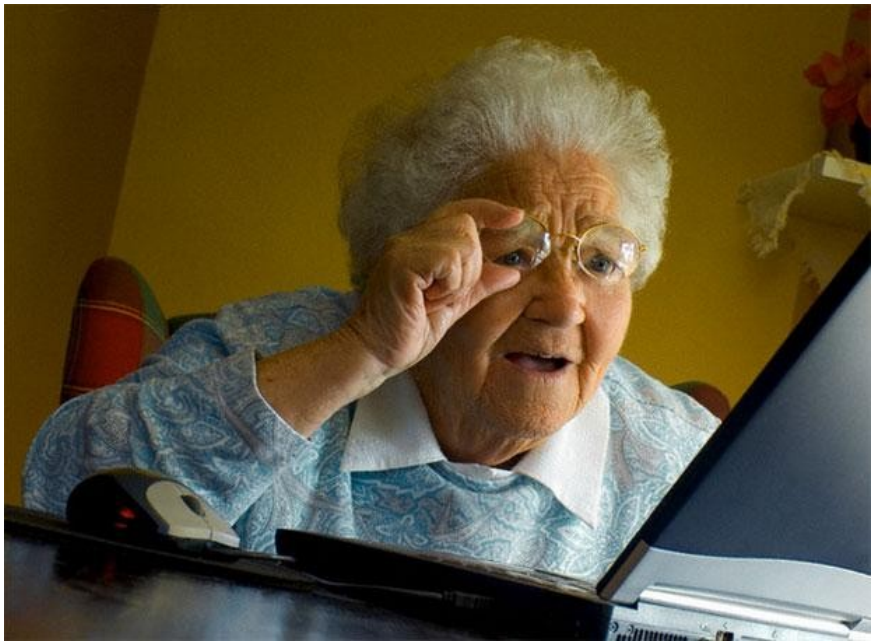
Conver

```
→  conver git:(master) ✗ ./conver run 3 mock
14:19:36.506 [info] Pa (<0.63.0>) init, n. ops: 12
14:19:36.506 [info] Pb (<0.64.0>) init, n. ops: 7
14:19:36.506 [info] Pc (<0.65.0>) init, n. ops: 11
Pa:R:0. Pc:R:0. Pb:R:0. Pa:W(1). Pb:W(2). Pc:R:0. Pa:R:2. Pb:R:2. Pc:R:2
:2. Pb:W(4). Pa:R:2. Pb:W(5). Pa:R:5. Pc:R:3. Pa:R:7. Pb:W(6). Pc:W(7).
Pc:W(8). Pa:R:7. Pc:R:8. Pa:R:6. Pc:W(9). Pa:W(11). Pc:W(10). Pa:W(12).
Pa:R:12. Pa ended.

Consistency check: [{a,[{op,a,read,18365784,473572369,0,[]},
                        {op,a,write,523396326,790537546,1,[]},
                        {op,a,read,809451162,1223526263,2,[]},
                        {op,a,write,1255406995,1607493902,3,[]},
                        {op,a,read,1639397013,1915492592,2,[ryw,rval]},
                        {op,a,read,1939397279,2063462375,5,[]},
                        {op,a,read,2101391052,2254486513,7,[]},
                        {op,a,read,2296396816,2612555337,7,[]},
                        {op,a,read,2660415843,2952494165,6,[rval]},
                        {op,a,write,2993366185,3425619839,11,[]},
                        {op,a,write,3460423293,3588512996,12,[]},
                        {op,a,read,3596393095,3881608490,12,[]}]},
                    {b,[{op,b,read,49402382,504493793,0,[]},
                        {op,b,write,535402991,802559044,2,[]},
                        {op,b,read,832461428,1246555624,2,[]},
                        {op,b,read,1278453886,1483508328,3,[]},
                        {op,b,write,1496402673,1794535937,4,[]},
                        {op,b,write,1820368287,2056500055,5,[]},
                        {op,b,write,2066391935,2287477894,6,[]}]},
                    {c,[{op,c,read,20355322,475466139,0,[]},
                        {op,c,read,508394301,868494261,0,[]},
                        {op,c,read,905453037,1361593170,2,[]},
                        {op,c,read,1380451076,1645483528,2,[]},
                        {op,c,read,1681405896,2092490062,3,[]},
                        {op,c,write,2095390788,2316486221,7,[]},
                        {op,c,write,2353428824,2494459524,8,[]},
                        {op,c,read,2509371338,2825526425,8,[]},
                        {op,c,write,2839399341,2962468709,9,[]},
                        {op,c,write,2981395922,3494497532,10,[]},
                        {op,c,read,3510398555,3686512350,10,[]}]}]

Monotonic Reads..................... PASS
Read-Your-Writes.................... FAIL
Monotonic Writes.................... PASS
Writes-Follow-Reads................. PASS
PRAM................................ FAIL
Causal.............................. FAIL
RealTime............................ PASS
Regular............................. FAIL
```



Conver

[DEMO]

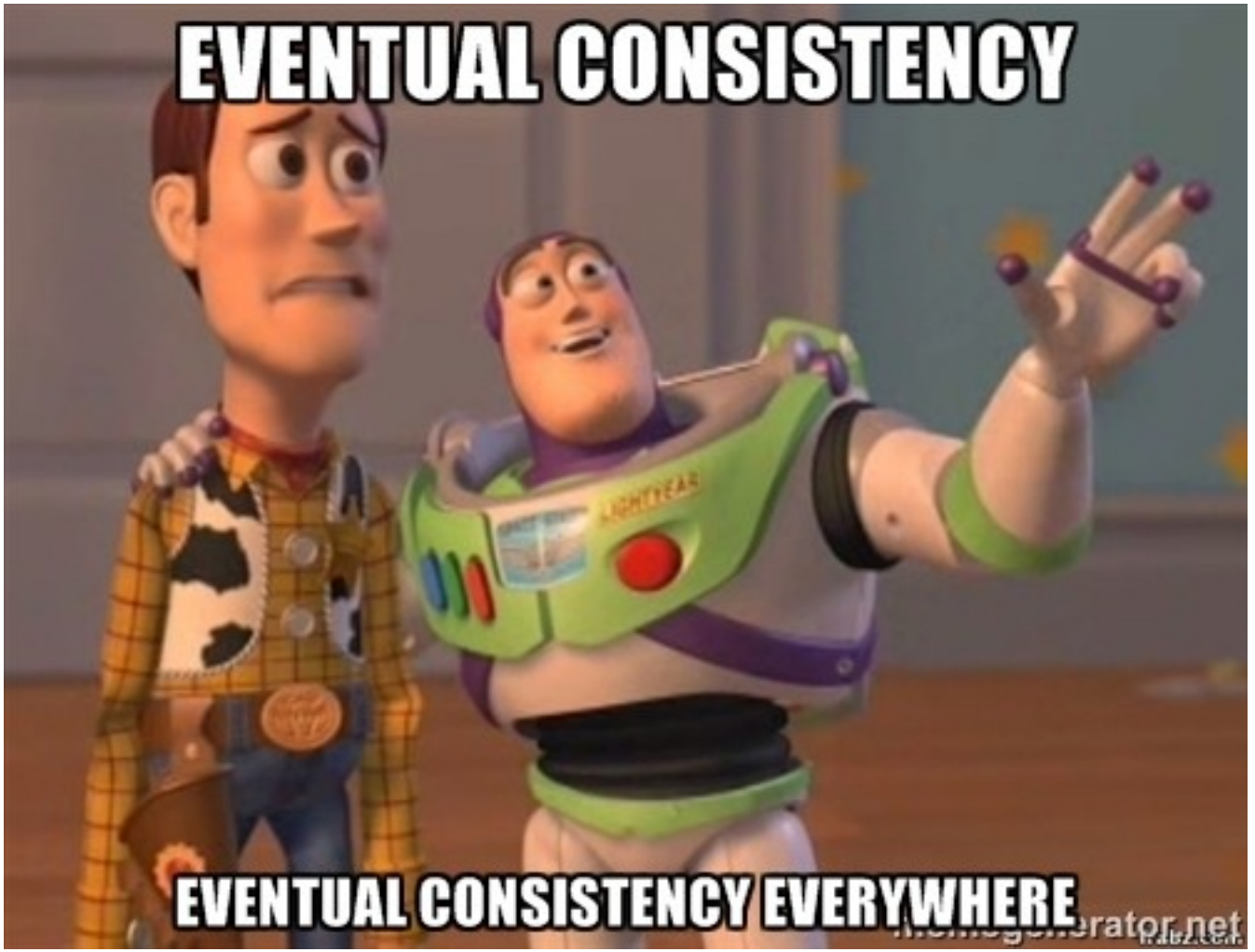


Conver

- Future work / possible extensions
  - Transactional models
  - Client distribution
  - Targeted executions
  - Test case shrinking
  - Fault injection
  - Dynamic verification
  - ...

Summary

- Express consistency as *composable* **properties** (logic predicates)
- Test consistency properties as **invariants** of executions
- An open source prototype: Conver



Thanks for your attention! - Q&A

Backup slide:
Comparison with CISE Tool

	Conver	**CISE**
Abstraction level / generality / expressiveness	Storage system	Application
Required integration effort	External plugin (~30 LOC)	Add annotations to application's source code; SMT solver
Debug info provided	Visualization and data of faulty executions	Counter-example