Data consistency in 3D

(It's the invariants, stupid)

Marc Shapiro
Masoud Saieda Ardekani
Gustavo Petri

UPMC SORBONNE UNIVERSITÉS

Inria informatics mathematics

This talk is about…

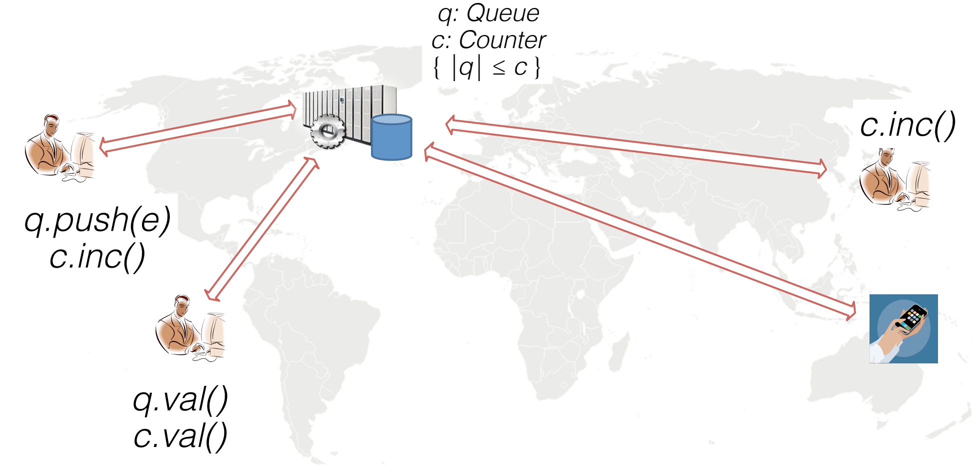
Understanding consistency
- Primitive consistency mechanisms
- How primitives compose models
- How models relate / differ
- What they cost

Understanding invariants
- Some interesting classes of invariants

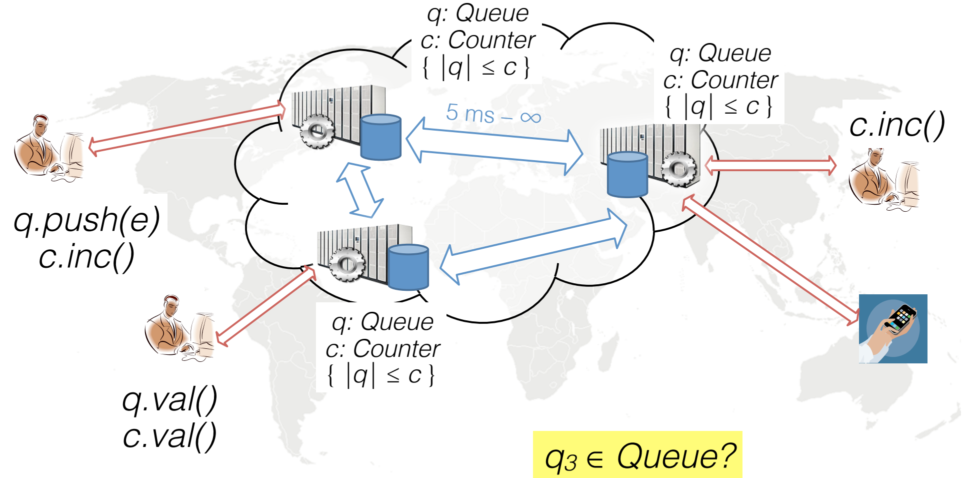
Relating consistency to invariants
- Which primitives guarantee which invariants

Useful intuitions for app. and system designers

Shared database



Geo-replicated database



$q_3 \in$ Queue?
$q_1 = q_2$?
$|q_1| \leq c_4$?

Consistency

More replicas:

- Better read availability, responsiveness, performance, etc.
- More work to keep replicas in sync

Consistent = behavior similar to sequential:

- Satisfies specs: does *q* behave like a queue?
- Replicas agree: is *q* identical everywhere?
- Objects agree: is $|q| \leq c$?
- Same flow of time? *q1.push()* before *q2.push()*

Consistency opportunities and costs

CAP

Availability

⇒ Parallelism keeps the hardware busy

⇒ More implem. options, scalable

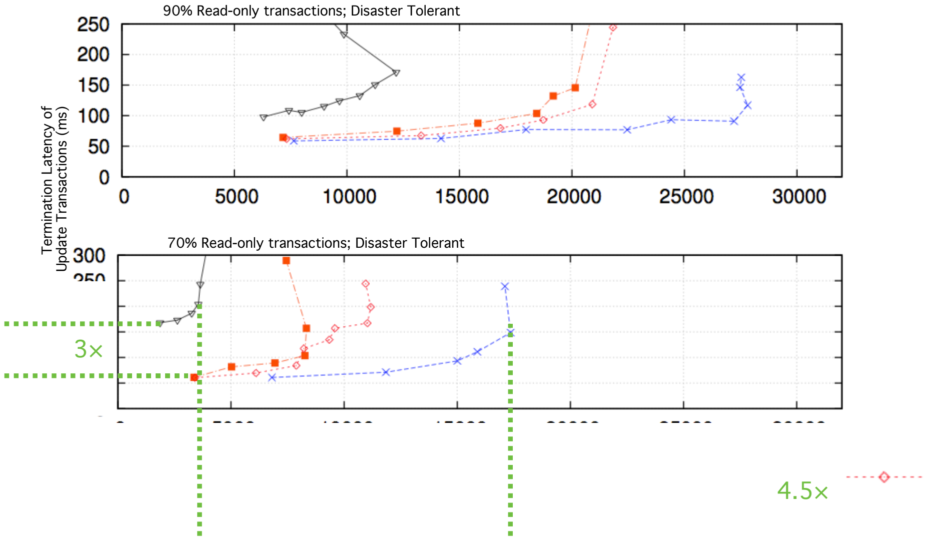
But consistency constrains order of events:

- Delay delivery
- Stale reads
- Waits, synchronisation (mutual wait)

Keeping track of order requires metadata

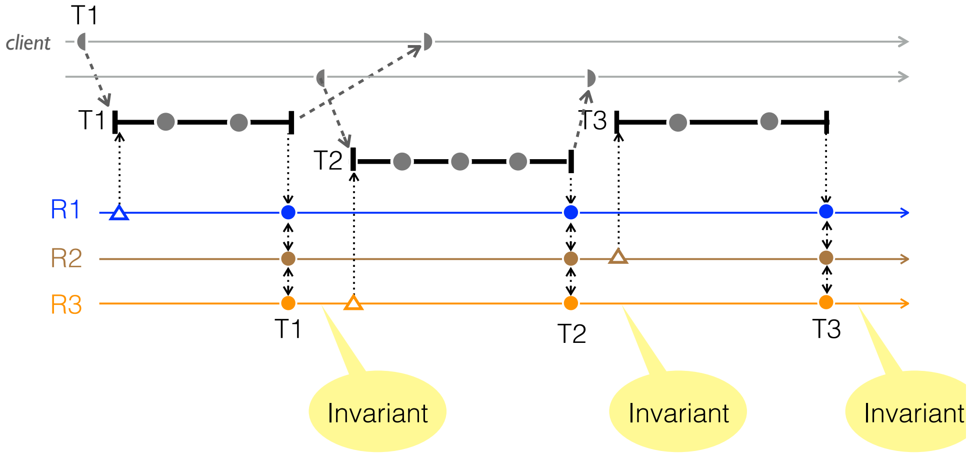
Significant!

Costs illustrated

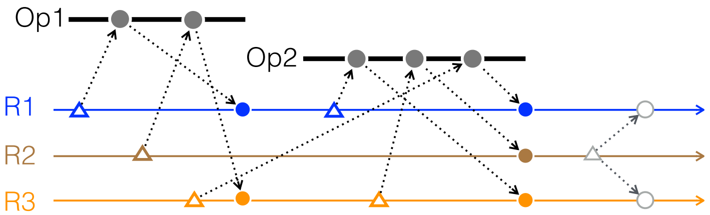


Credit: Masoud Saeida Ardekani

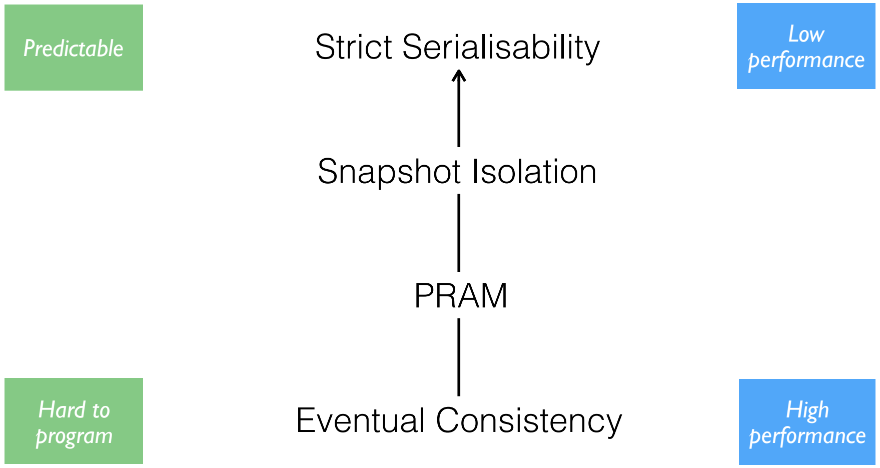
Strict Serialisability



Eventual consistency



Strong vs. weak?



Predictable

Strict Serialisability

Low performance

Snapshot Isolation

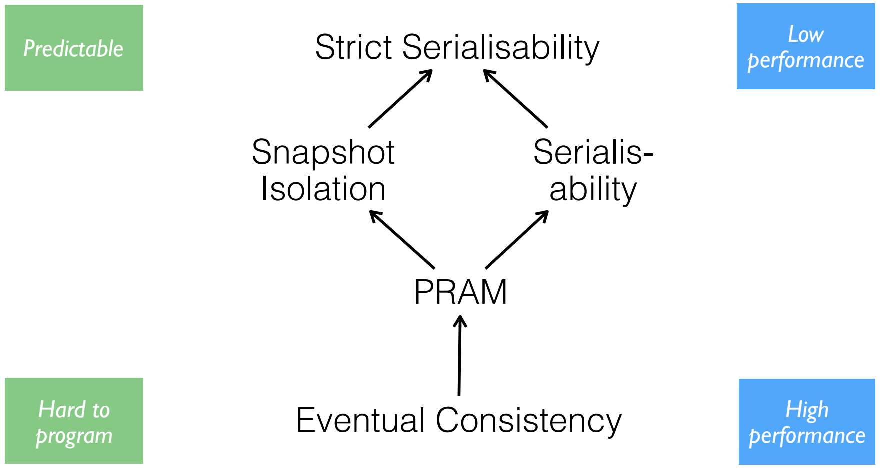
PRAM

Hard to program

Eventual Consistency

High performance

Strong vs. weak?



Predictable

Strict Serialisability

Low performance

Snapshot Isolation

Serialis-ability

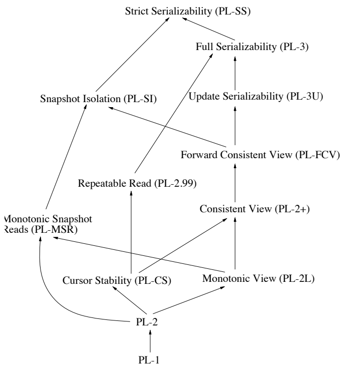
PRAM

Hard to program

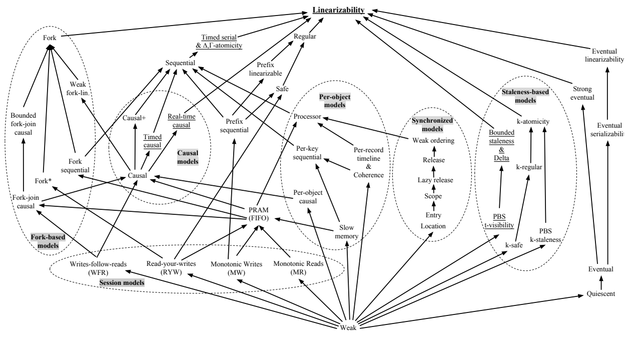
Eventual Consistency

High performance

Strong vs. weak?


Strict Serializability (PL-SS)
Full Serializability (PL-3)
Snapshot Isolation (PL-SI)
Update Serializability (PL-3U)
Forward Consistent View (PL-FCV)
Repeatable Read (PL-2.99)
Consistent View (PL-2+)
Monotonic Snapshot Reads (PL-MSR)
Cursor Stability (PL-CS)
Monotonic View (PL-2L)
PL-2
PL-1

Transactional
Adya 1999

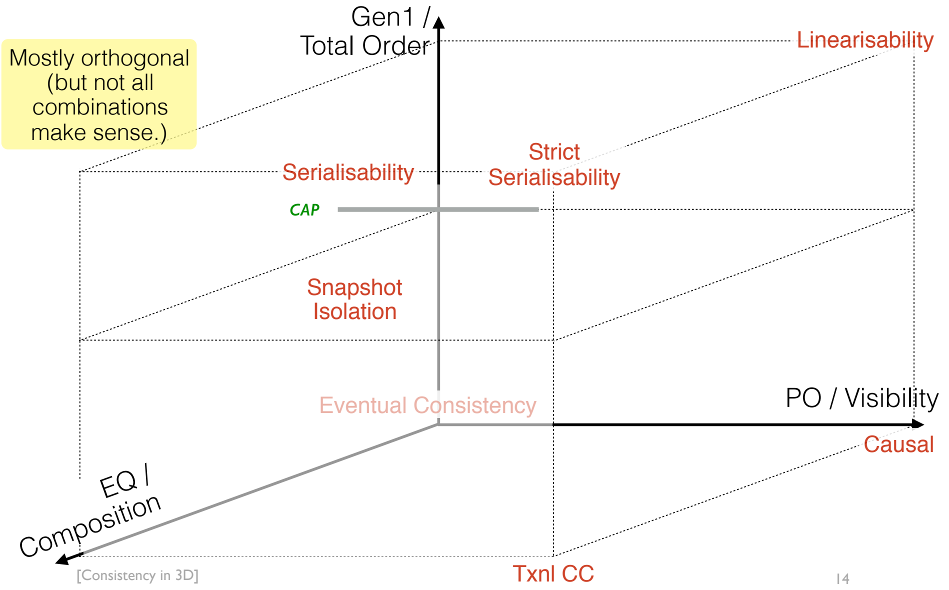


Non-transactional
Viotti & Vukolić 2016

Three classes…

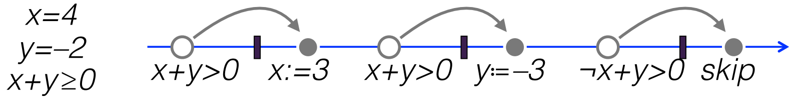
	…of invariant	… of protocol
Gen1	Object value	Total order of operations
PO	Relative ordering of operations	Visibility
EQ	State equivalence	Composition

Three dimensions



Mostly orthogonal (but not all combinations make sense.)

Operation



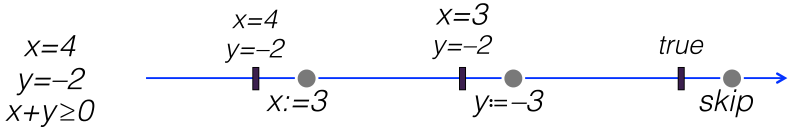
generator: read, compute, generate effector

effector: compute, write side-effect

Sequential execution:

- precondition ⟹ invariant
- each effector individually safe

Sequential correctness



generator: read, compute, generate effector

effector: compute, write side-effect

Sequential execution:

- precondition ⟹ invariant
- each effector individually safe

Guarantee vs. semantics

Guarantee:

- Class of invariants that is always true
- Regardless of application code
- Assuming sequentially correct

Application can compensate for absence of guarantee

- e.g. Inv={ *c≥0* }, app: *c.inc()*

Data types

Register

- Update: assign with constant
  - Not commutative
  - Absorbing

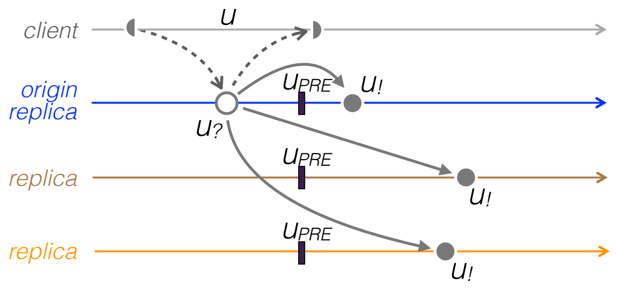
High-level types

- Counter, ORset, Sequence: effectors commute
- Stock, Account, Queue: ¬ commute

Composed data

- + structural invariants

Replicated operation

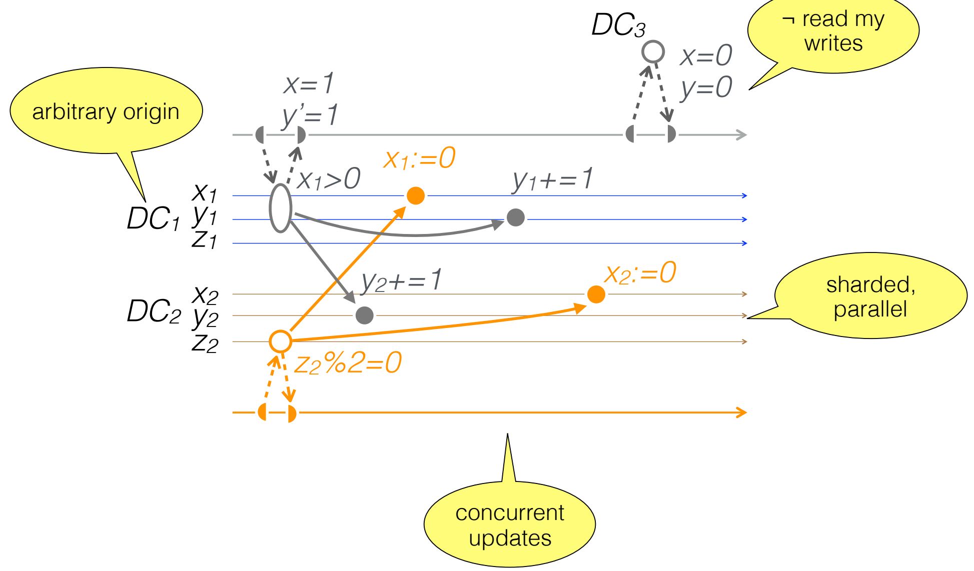


u: state ↪ (retval, (state ↪ state))

Read one, write all (ROWA)

Deferred-update replication (DUR)

Sharded, geo-replicated



Type EQ invariants

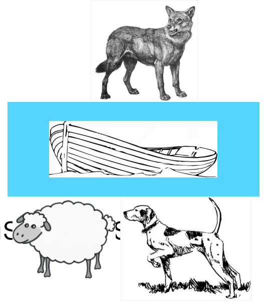
- *A = B*
- *x.friendOf (y) ⟺ y.friendOf (x)*
- *x + y = constant*
- *South ⊎ Boat ⊎ North*
  = { sheep, dog, wolf }

Joint update to two objects

Atomicity (all-or-nothing) property of trans

Protocol: single update message

- Asynchronous



EQ: transactional composition

Airplane reservation

- Allocate a seat to me
- Pay for the flight

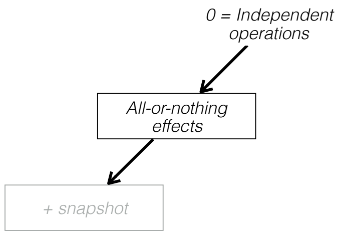
Two EQ relations:

- paid = have_seat
- my $$ + airline $$ = constant

Ad-hoc grouping

(This txn also needs TO + snapshot)

EQ/Composition axis



Transaction groups operations

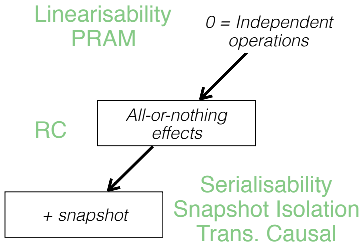
All-or-nothing effects:

- Deliver effectors indivisibly
  - packaged together
- + same TOE
  - ≈ 2-phase commit

Snapshot reads:

- all generators read from same set of effectors
  - maintain versions
- + same TO, VIS guarantees
  - coordination

EQ/Composition axis



Transaction groups operations

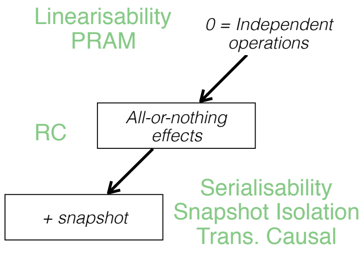
All-or-nothing effects:

- Deliver effectors indivisibly
  - packaged together
- + same TOE
  - ≈ 2-phase commit

Snapshot reads:

- all generators read from same set of effectors
  - maintain versions
- + same TO, VIS guarantees
  - coordination

EQ/Composition axis



Transaction groups operations

All-or-nothing effects:

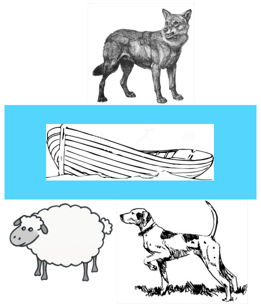
- Deliver effectors indivisibly
  - packaged together
- + same TOE
  - ≈ 2-phase commit

Snapshot reads:

- all generators read from same set of effectors
  - maintain versions
- + same TO, VIS guarantees
  - coordination

Type PO invariants

- *employee.manager.salary* ≥ *employee.salary*
- $S1; S2; S3 \equiv S1 \Leftarrow S2 \Leftarrow S3$
- $dog \in S \Leftarrow sheep \in S \wedge wolf \in S$
- Referential integrity
  "inode references disk block"
- $ACL(u, p) \Leftarrow access(u, p)$



Demarcation Protocol:

1. increase LHS by c
2. increase RHS by $c' \leq c$

⟹ *ordered delivery*

No synchronisation: Available

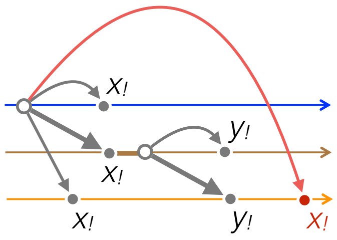
PO: transitive / causal visibility

$x = 100; y = 100$

Inv = $\{ x \geq y \}$

Ex 1:

- P1: x += 100
- P2: if $x > y$ then y += $(x–y)/2$
- P3: $x \geq y$?
- Transitive visibility $vis^* \subseteq vis$



PO: transitive / causal visibility

$x = 100; y = 100$

Inv = $\{ x \geq y \}$

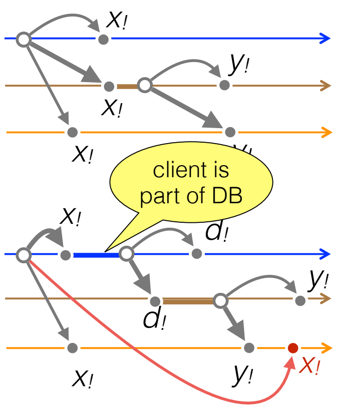
Ex 1:

- P1: x += 100
- P2: if $x > y$ then y += $(x–y)/2$
- P3: $x \geq y$?
- Transitive visibility $vis^* \subseteq vis$

Ex 2:

- P1: x += 100; d ≔ 100
- P2: if $d > 0$ then y += $d/2$
- P3: $x \geq y$?

Causal visibility $(vis; po)^* \subseteq vis$



PO/Visibility axis

Visibility

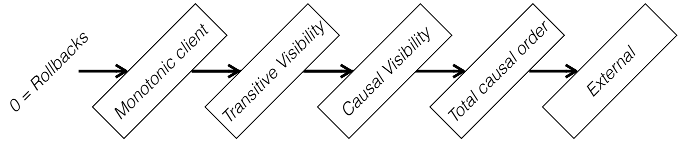
- Which *writes* visible to *reads*

Transitive closure property

- Metadata
- System-wide

Sender not delayed ⟹ writes available

Stale data ⟹ reads available

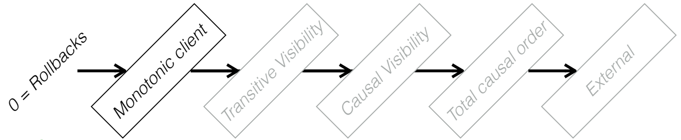


Monotonic client

- Read My Writes
- Monotonic Reads

Often assumed

- Buffer

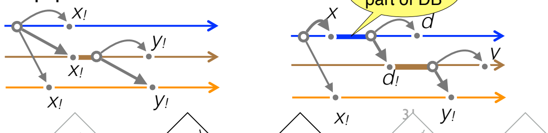


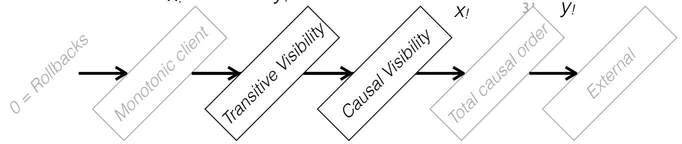
Eventual Consistency
“Not reasonable”

Transitive, causal vis.

- Effector: metadata identifies set of predecessor effectors
- Delay delivery after predecessors
  - Read stale data
- Graph: unbounded
- Vector clock: 10^4—10^6 entries × 8 bytes!
- Approximate VC: stron ler

client is part of DB





SER
NMSI

PSI

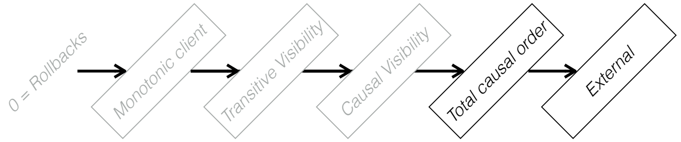
Total/external causal

Total order extends causal order

Metadata: 1 single scalar

- but cost of total order

External: real-time clock



Gentle Rain

Linearisable
SSER

Gen1 invariants

Inv = "$0 \leq x$"
$u_!$ = "$x \coloneqq x{-}1$"
$\{\, Inv \wedge 1 \leq x \,\}\ u_!\ \{\, Inv \,\}$

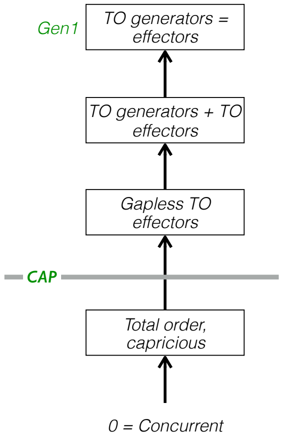
Predict that *Inv* will be true after $u_!$:

- Sequential: weakest precondition
- Generalises to bounded concurrency

Unbounded concurrency: no sufficient precondition

- Invariant is not stable
- Limit concurrency: escrow
- No concurrency: <u>order updates</u>

Gen1: total order



Gen1

TO generators = effectors

TO generators + TO effectors

Gapless TO effectors

CAP

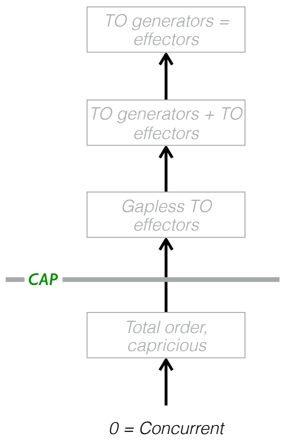
Total order, capricious

0 = Concurrent

Do replicas observe events in the same order?

Pick a unique number

0 = unordered



TO generators = effectors

TO generators + TO effectors

Gapless TO effectors

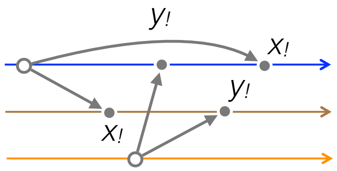
CAP

Total order, capricious

0 = Concurrent

Do replicas observe events in the same order?

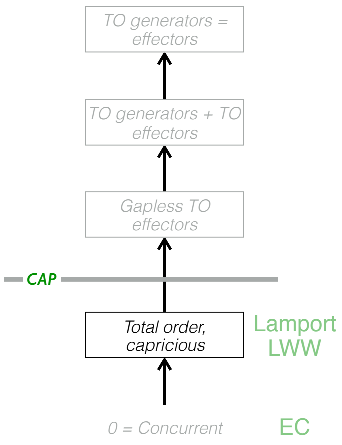
Pick a unique number



No: concurrent

- Commute ⟹ converge
- Stable precondition ⟹ Invariant

Capricious TO effectors



TO generators = effectors

TO generators + TO effectors

Gapless TO effectors

CAP

Total order, capricious

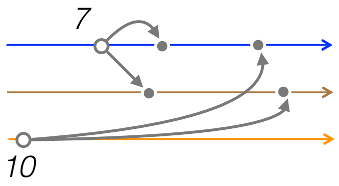
Lamport LWW

0 = Concurrent

EC

Do replicas observe events in the same order?

Pick a unique number

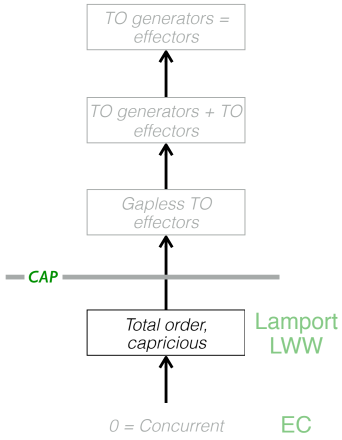


Pick a number locally: capricious

Gap: will arrive later?

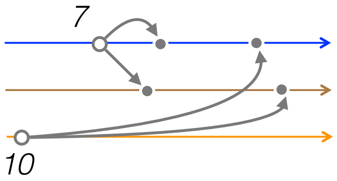
- Non-monotonic: rollback
- Monotonic
  - Wait for gap to fill (Lamport 78)
  - Lost updates (LWW)

Capricious TO effectors



Do replicas observe events in the same order?

Pick a unique number

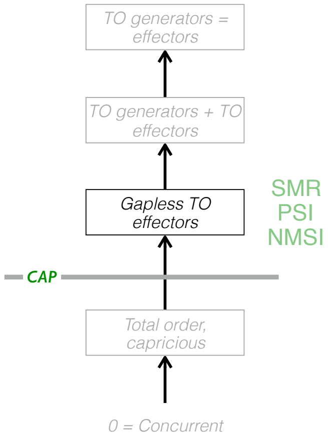


Pick a number locally: capricious

Gap: will arrive later?

- Non-monotonic: rollback
- Monotonic
  - Wait for gap to fill (Lamport 78)
  - Lost updates (LWW)

Gapless TO effectors



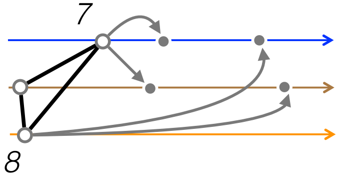
Do replicas observe events in the same order?

Pick a unique number

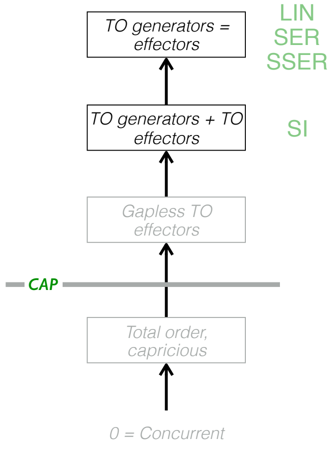
Gapless:

- No lost updates
- Consensus, 2PC to uniquely allocate next free number

⟹ not available



TO generators



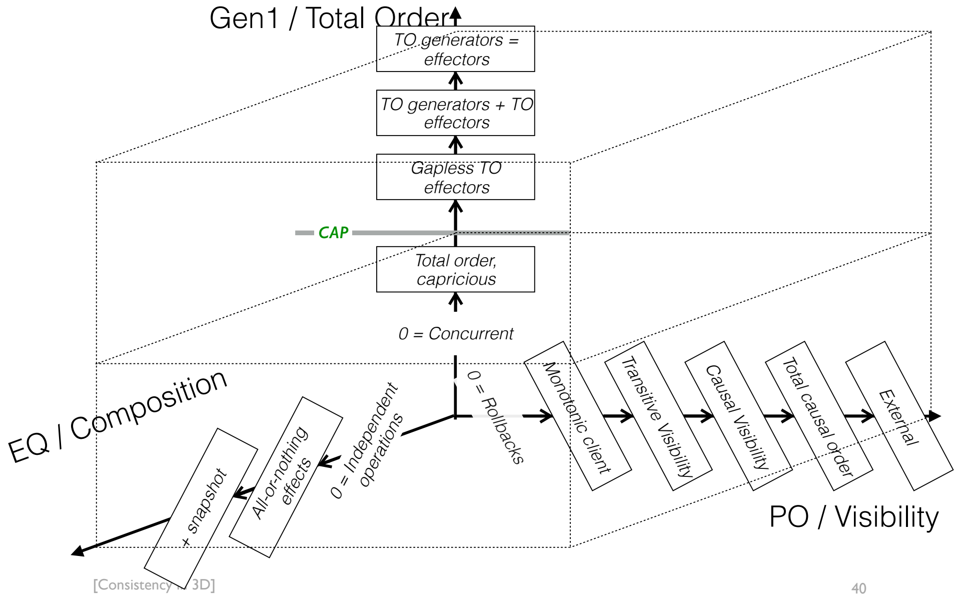
Do replicas observe events in the same order?

Pick a unique number

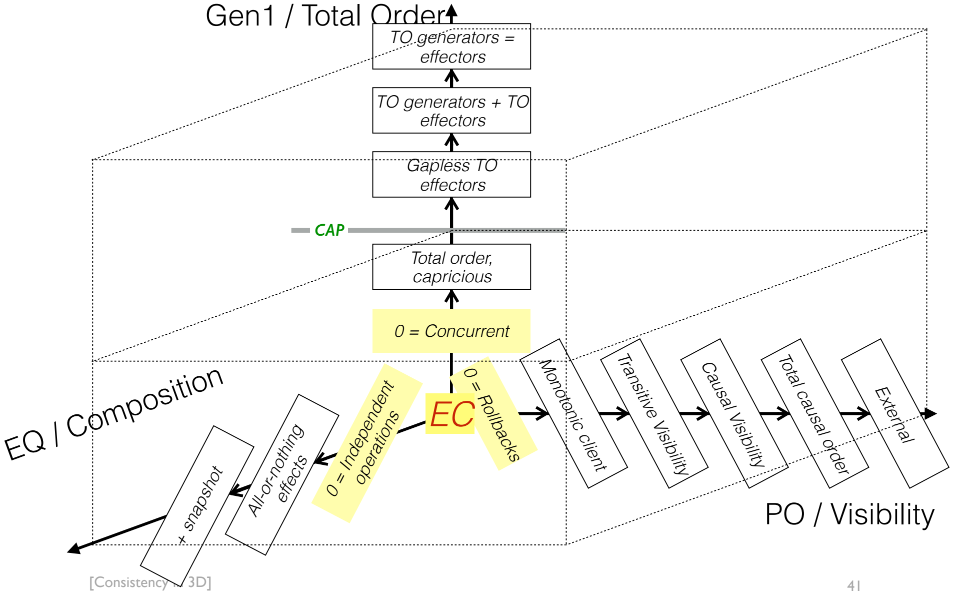
TO effectors

+ TO generators
  - separate from effectors
  - same order as effectors

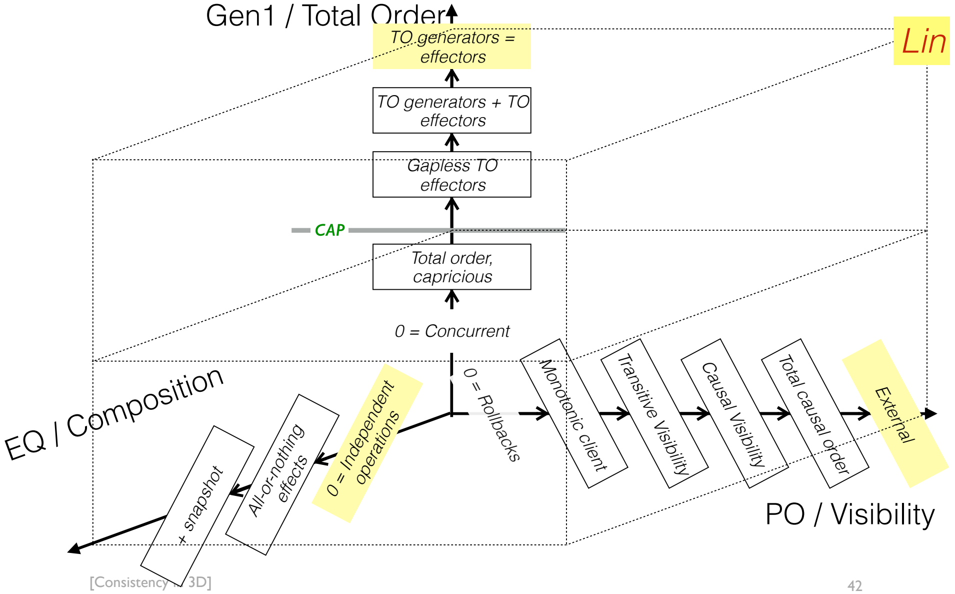
Three dimensions



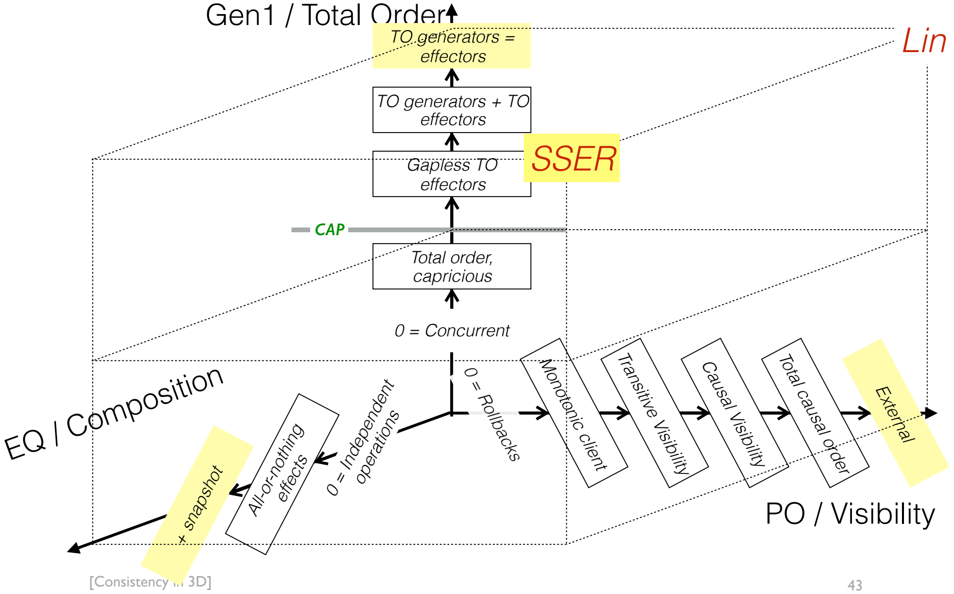
Three dimensions



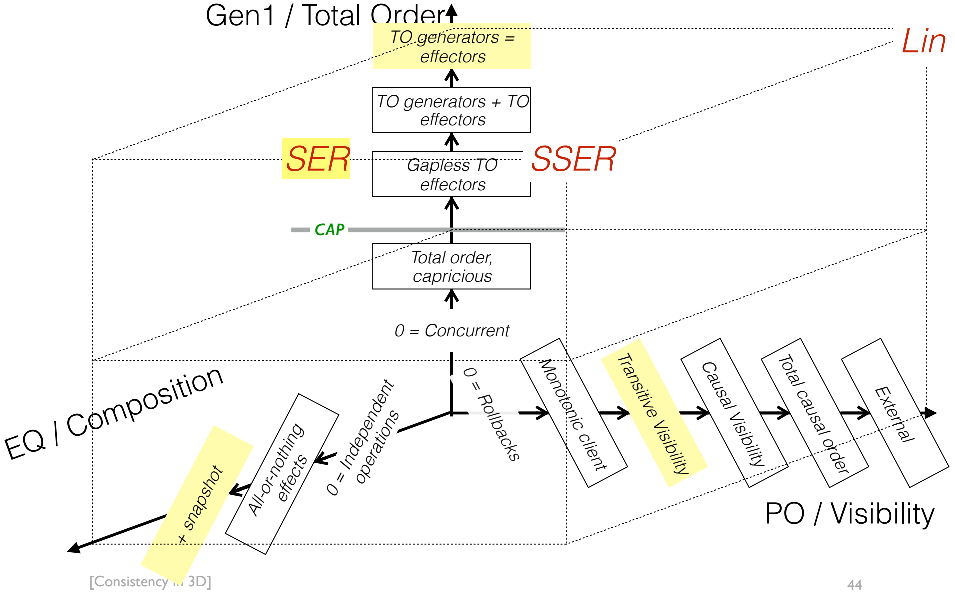
Three dimensions



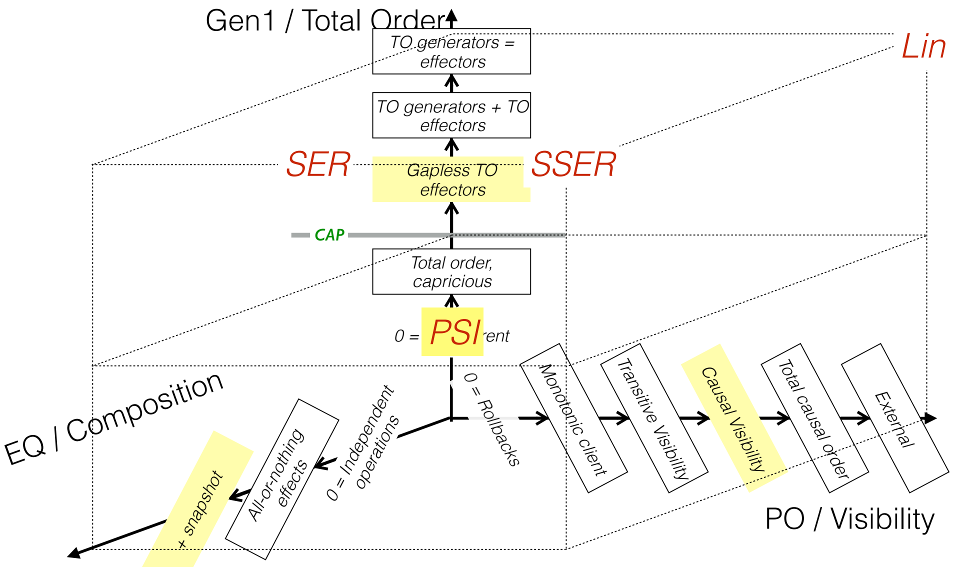
Three dimensions



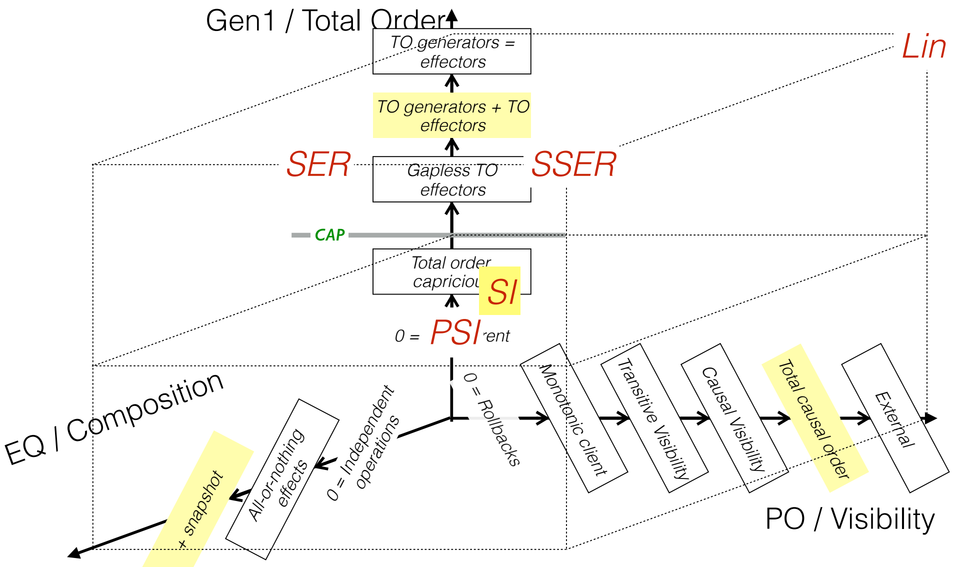
Three dimensions



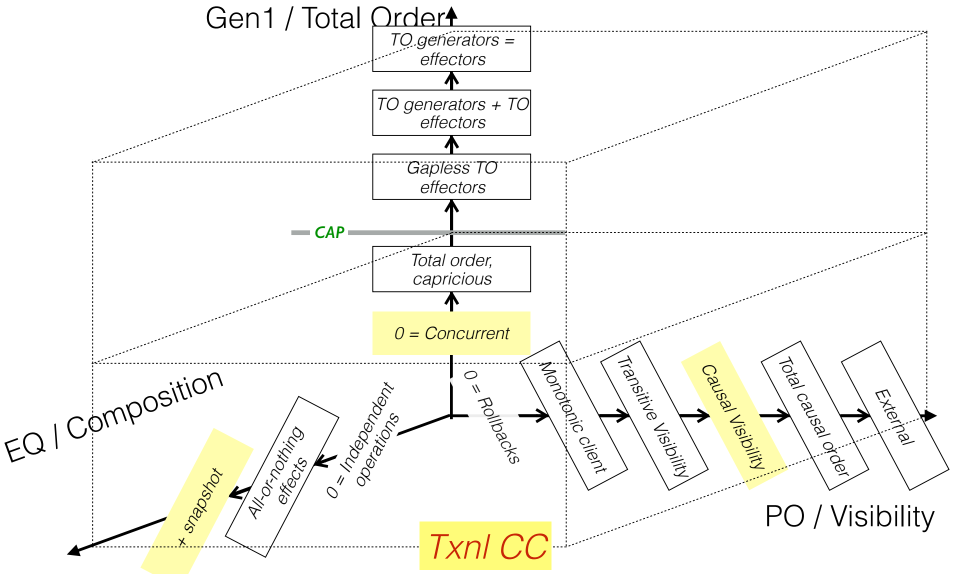
Three dimensions



Three dimensions



Three dimensions



		Visibility				
Total Order	**Composition**	**Rollbacks**	**Monotonic**	**Transitive**	**Causal**	**External**
TOG=TOE	All-or-Nothing + Snapshot			SER		SSER
	All-or-Nothing Effectors					
	Single Operation				SC	LIN
Gapless TOE	All-or-Nothing + Snapshot			NMSI	PSI	SSI
	All-or-Nothing Effectors					
	Single Operation					
Capricious TOE	All-or-Nothing + Snapshot	Bayou				∅
	All-or-Nothing Effectors					∅
	Single Operation		LWW			∅
Concurrent Ops	All-or-Nothing + Snapshot				Causal HAT	∅
	All-or-Nothing Effectors		RC			∅
	Single Operation	EC	PRAM		CC	∅

Summary

Distributed, replicated data

- Improves read availability
- Parallel updates may violate invariants
- Guarantee: invariants maintained by system
- System vs. application cost trade-off
  - Tools needed

3D consistency design space

- Total order (effectors, generators)
- Visibility order
- Transactional Composition

Work in progress

Creative Commons Attribution-ShareAlike 4.0 Intl. License

You are free to:

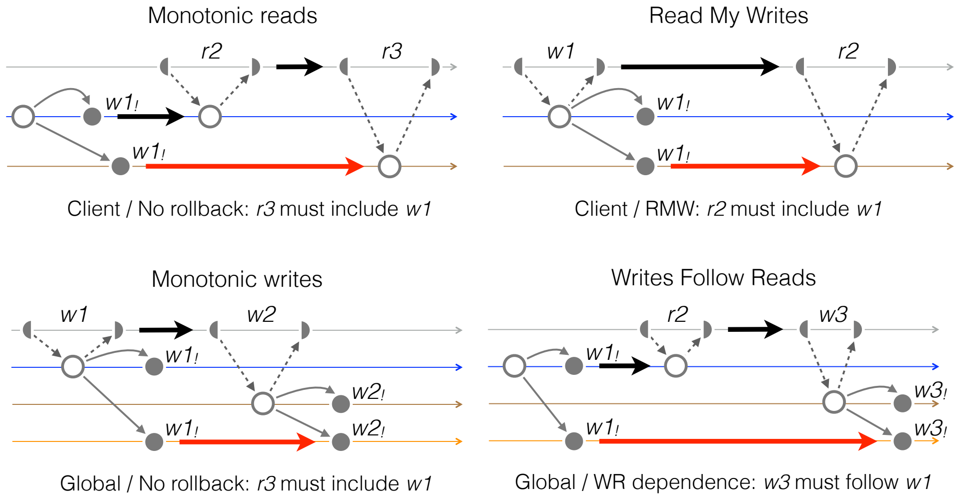
- Share — copy and redistribute the material in any medium or format
- Adapt — remix, transform, and build upon the material

for any purpose, even commercially, under the following terms:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

4 session guarantees ≣ causal



Monotonic reads

Client / No rollback: *r3* must include *w1*

Read My Writes

Client / RMW: *r2* must include *w1*

Monotonic writes

Global / No rollback: *r3* must include *w1*

Writes Follow Reads

Global / WR dependence: *w3* must follow *w1*