# Exponential cake and consistent controllers

## Heliax Research Talk

**Murdoch J. Gabbay**
**(reporting on joint work with Isaac Sheff)**

17 July 2026

# Some links

This talk reports on joint work with Isaac Shell, and formalised by Anthony Hart in Lean. By the time I deliver this, the material should be online here:

- https://doi.org/10.5281/zenodo.21074359 (the ART report).
- https://doi.org/10.5281/zenodo.21074395 (Anthony's Lean formalisation).

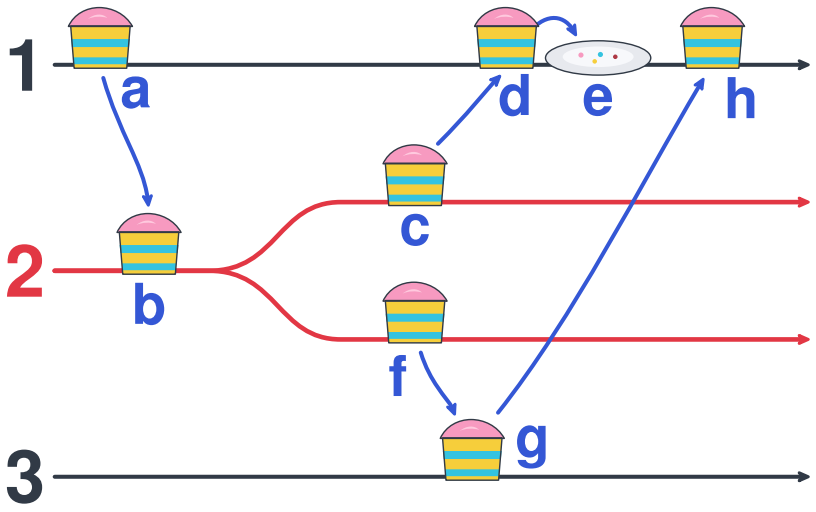
# The consistent histories problem

A controller is a (possibly distributed) independent machine that issues state. Examples include: a blockchain, a bank, a land registry, a university, a database, a parent.

The Real World has a multitude of controllers, and they communicate and send resources to each other.

Immediately, an issue arises. I give you . . . the exponential cake:

# The exponential cake: a cake you can have, and also eat




- Controller 1 has a cake . . .
- and sends it to Controller 2.
- Controller 2 forks.
- Controller 1 receives a cake back from Controller 2, and eats it.
- Controller 3 receives a cake from Controller 2, and forwards it to Controller 1.

Controller 1 has a cake, and has eaten it.

Controllers 1 and 3 are honest. Yet Controller 2's cake-duplicity has affected them. Not good. What can we do?

# Idea 1: complete histories

Label cake with a complete history of everything that happened to it.

Then Controller 1, on receiving the second cake, can check its history, see it's a forked copy, and refuse to accept it (and perhaps alert Controller 3 not to trust Controller 2).

Cf. David Chaum 1983 p.202 item 11 *Blind signatures for untraceable payments*: "Bank adds note to comprehensive list of cleared notes and stops if note already on list".

This works, but it's expensive: every time anybody does something with cake, this has to be remembered forever. In particular, this makes doing things with cakes expensive.

We want cake to move freely, and for this to be cheap and easy!

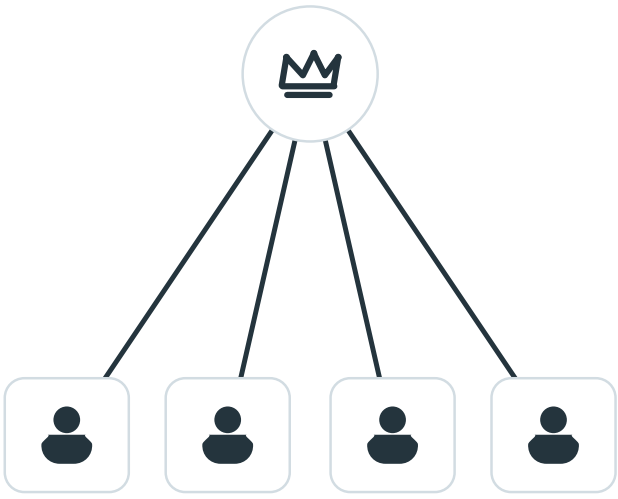
# Idea 2: One True Controller

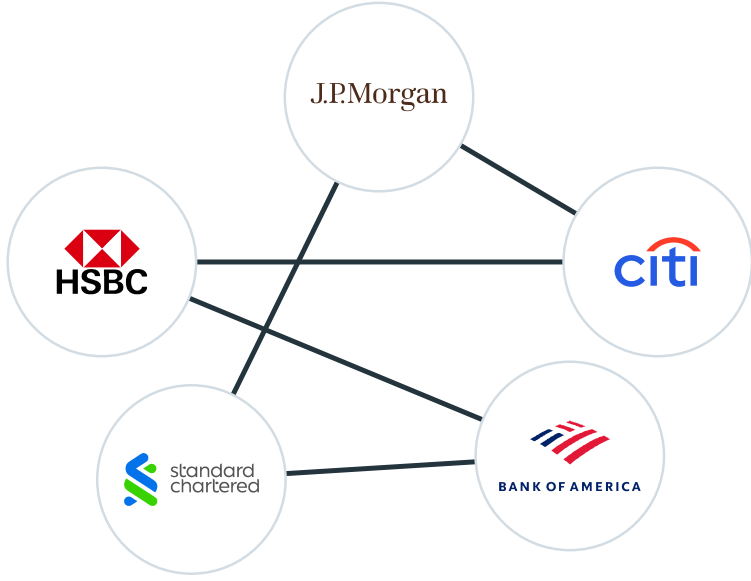
Ban multiple controllers. There can be only one!

This is not realistic:

1. It's technically expensive, requiring a single state machine to maintain consensus on truth for all objects.
2. It's also macroeconomically infeasible. The One True Controller would charge exorbitant fees for maintaining states (both because it can, and because if it did not then it would be clobbered by congestion).

# One true controller vs small clique of controllers



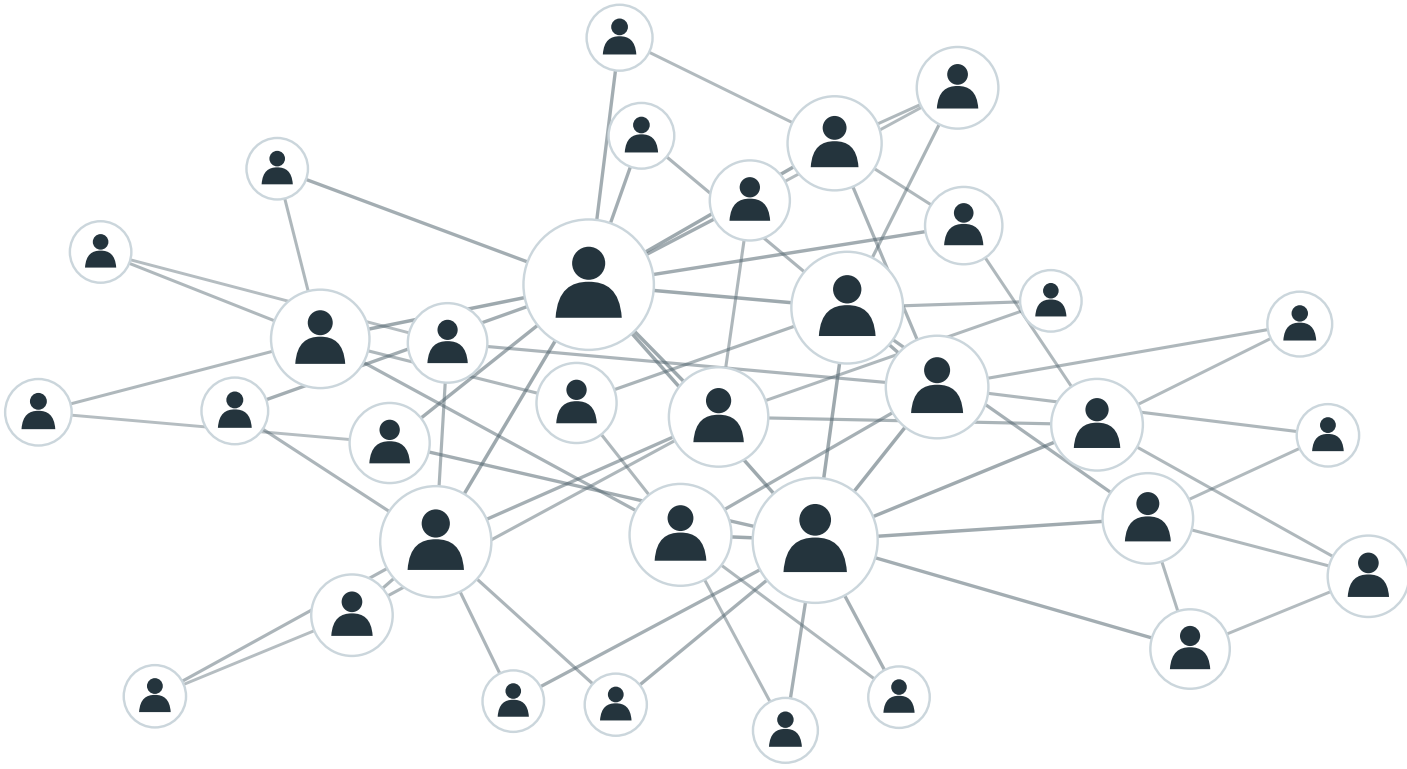


# Idea 2b: One Small Clique of Trusted Controllers

Make Controllers (very) permissioned, so they don't misbehave.

This is an oligopoly, and structurally it just pushes the problem down one level.

# True decentralisation



But how to sensibly program this without just retaining total histories?

# Idea 3: multiple controllers with reducible histories

Multiple controllers are a feature, not a bug, but we need some notion of history that is safe (can detect double-spends), yet also editable.

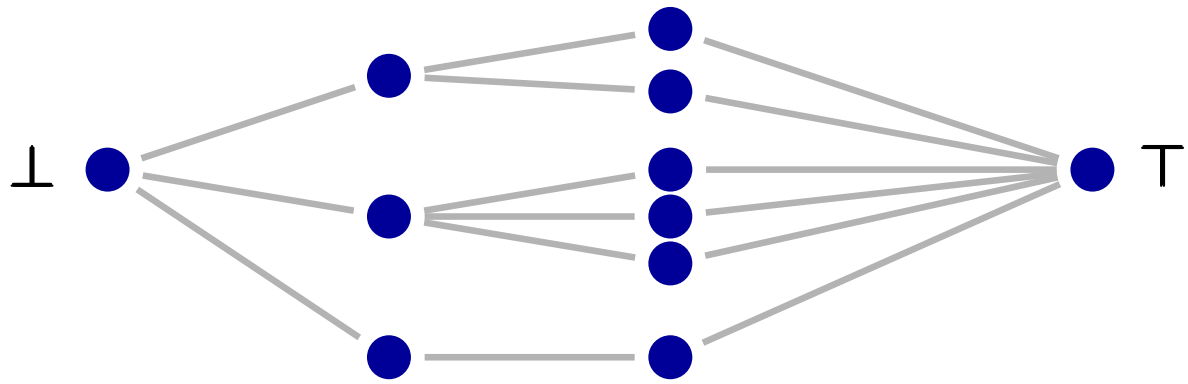
This is *reducible* histories, or as Isaac calls them, *cuttable controller tags*.

In order to build these, we need three ingredients:

1. A notion of state for a single controller.
2. A notion of state for a collection of controllers, along with a notion of *attestation* allowing them to communicate.
3. A sophisticated *flag* system allowing controllers to reduce the history of a resource.

Easy as 1, 2, 3. Let's do it.

# 1. Top-trees: sequential state with byzantine behaviour




A bounded semilattice is a tuple $\mathcal{L} = (\mathsf{X}, \vee, \bot, \top)$ where $\mathsf{X}$ is a set of times and $x \vee x'$ is a binary join, such that $x \leq x'$ when $x \vee x' = x'$ and $\bot \leq x \leq \top$ always.

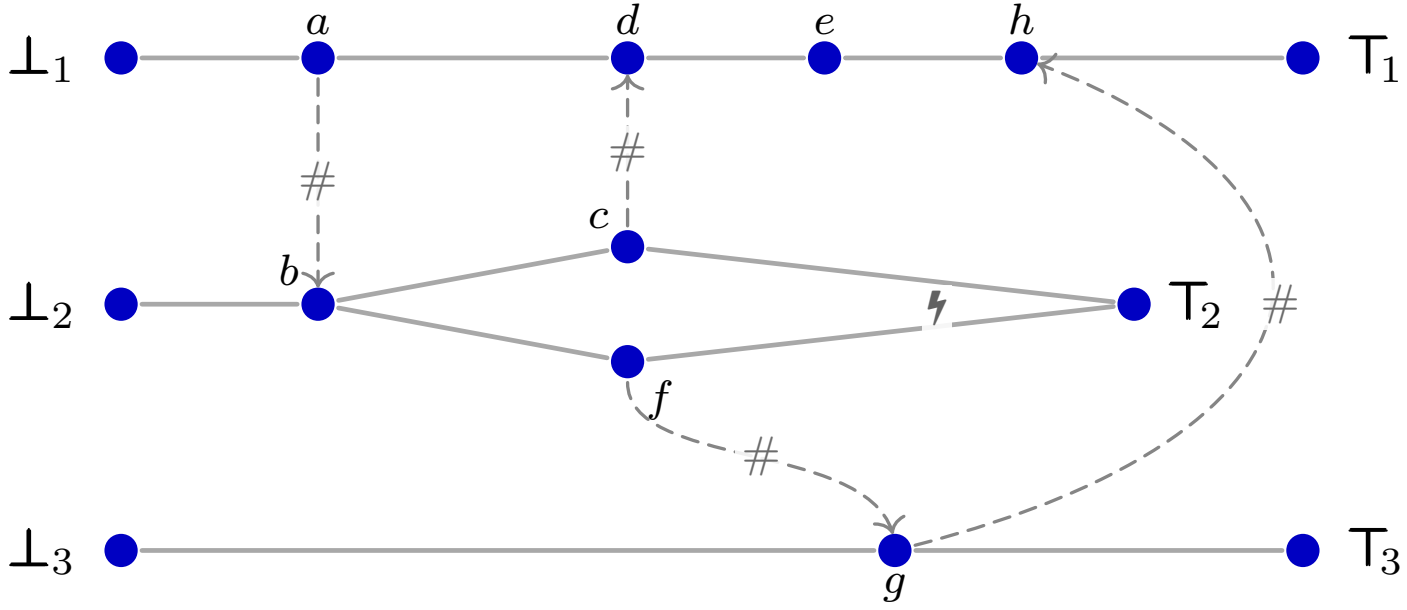
Call $t$ and $t'$ incomparable when $t \not\leq t' \wedge t' \not\leq t$ (equivalently: $t \vee t' = \top$).

Write $t \lightning t'$ when $t \vee t' = \top$. Call $\mathcal{L}$ sequential when $t \vee t' \in \{t, t', \top\}$ always ($t \leq t'$ or $t' \leq t$ or $t \lightning t'$).

Call $\top$ the inconsistent element, because it is the join of any two non-top elements that cannot be put in sequence, and call $t \neq \top$ consistent.

Call a sequential bounded semilattice a top-tree.

# 2. Exponential cake as a forest of attesting top-trees . . .




Attestation $\#$ takes a time $t$ at controller $n$ and a time $s$ at controller $m$, and forms a time $t\#s \geq t$ at controller $n$ that is "$t$ *combined with* or *attesting to* the information in $s$".

# Attestation

Attestation $\#$ satisfies various axioms, including:

1. Expansive: $t \leq t\#s$ always.
2. Contradiction-preserving: $s \lightning s'$ implies $t\#s \lightning t'\#s'$.
   $t$ and $t'$ are at the same controller.

Attestation is *not* transitive.

1. $s \lightning s'$ does not imply $t\#s \lightning t'\#(v'\#s')$.
2. $s \lightning s'$ does not imply $t\#(w\#s) \lightning t'\#(v'\#s')$, where $w$ and $v'$ are at *different* controllers.

# Attestation

1. $s \lightning s'$ implies $t\#s \lightning t'\#s'$.
2. $s \lightning s'$ does not imply $t\#s \lightning t'\#(v'\#s')$.
3. $s \lightning s'$ does not imply $t\#(w\#s) \lightning t'\#(v'\#s')$, for $w$ and $v'$ at different controllers.

Which of the following contradict? Which are simultaneously consistent?

1. Marmite's great.
2. Marmite's foul.
3. My son says that Marmite's great.
4. My wife says that my son says that Marmite's foul.
5. My wife says that I say that Marmite's great.
6. My wife says that my son says that Marmite's great.

# Attestation

1. $s \lightning s'$ implies $t\#s \lightning t'\#s'$.
2. $s \lightning s'$ does not imply $t\#s \lightning t'\#(v'\#s')$.
3. $s \lightning s'$ does not imply $t\#(w\#s) \lightning t'\#(v'\#s')$, for $w$ and $v'$ at different controllers.

Which of the following contradict? Which are simultaneously consistent?

1. Marmite's great.
2. Marmite's foul.
3. My son says that Marmite's great.
4. My wife says that my son says that Marmite's foul.
5. My wife says that I say that Marmite's great.
6. My wife says that my son says that Marmite's great.

# Summary so far

We have a notion of interacting controllers:

1. Each controller is modelled as a top-tree, to represent its state.
2. Controllers communicate via an expansive contradiction-preserving *attestation operation*.
3. We can think of a *controller tag* on a resource just as a list of the states of the controllers that the resource passed through: $(t_n, t_{n\text{-}1}, \ldots, t_2, t_1)$ (time here flows right-to-left).
   (In general the tag may be a DAG, because resources can be combined, but this is not relevant for detecting double-spends: we can just 'explode' the DAG as a bag of linear traces.)

So far, so good. How shall we manage tag reduction?

# Flags: nextindex, cutyou, cutme

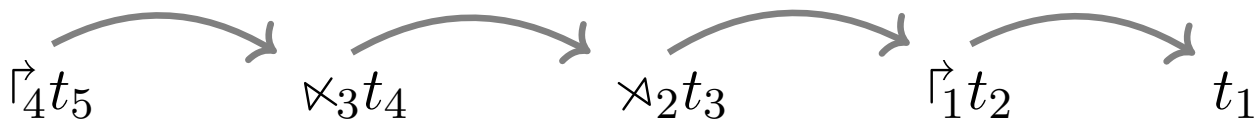
To each controller, associate a flag, which is one of

$$\vec{\lceil}_i, \quad \ltimes_i, \quad \rtimes_i.$$

Intuition is:

- $\vec{\lceil}_i$ (nextindex) means 'The next index is $i$'.
- $\ltimes_i$ (cutyou) means 'The next index is $i$ and I am ready to cut $i$'.
- $\rtimes_i$ (cutme) means 'The next index is $i$, and I am ready to be cut by somebody to my left so that their next index becomes $i$'.

Any time $t$ may have the form $\vec{\lceil}_i t'$, $\ltimes_i t'$, or $\rtimes_i t'$ for some $t'$.

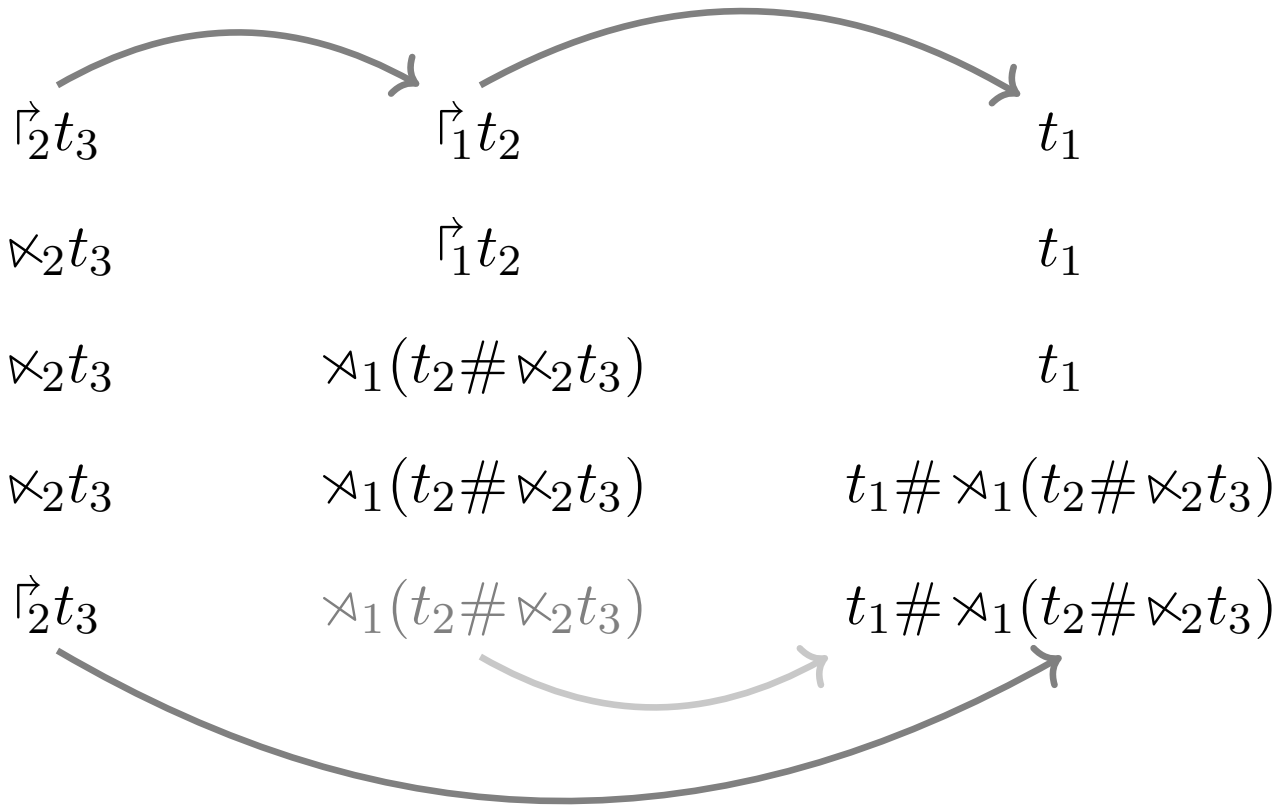



For $t$ and $t'$ times at the same controller, we assume $\ltimes_i t \lightning \rtimes_j t'$ for any $t$ and $t'$ and any $i \leq j$.

# Schematically, reduction works as follows

$$\frac{(\ldots,\ \vec{\lceil}_j t_k,\ \vec{\lceil}_i t_j,\ t_i,\ \ldots)}{(\ldots,\ \vec{\lceil}_i t_k,\ \ t_i \# \rtimes_i (t_j \# \ltimes_j t_k),\ \ldots)}\ (\textbf{Reduce})$$

Here is how it works under the hood:



# How to implement reduction (intuition)

If I'm a controller with a tag containing times of active controllers and I want to reduce that tag, then I can go talk to the controllers concerned and ask them to perform the attestation dance just described — and then update to a reduced history.

So the tag is an abstraction of history.

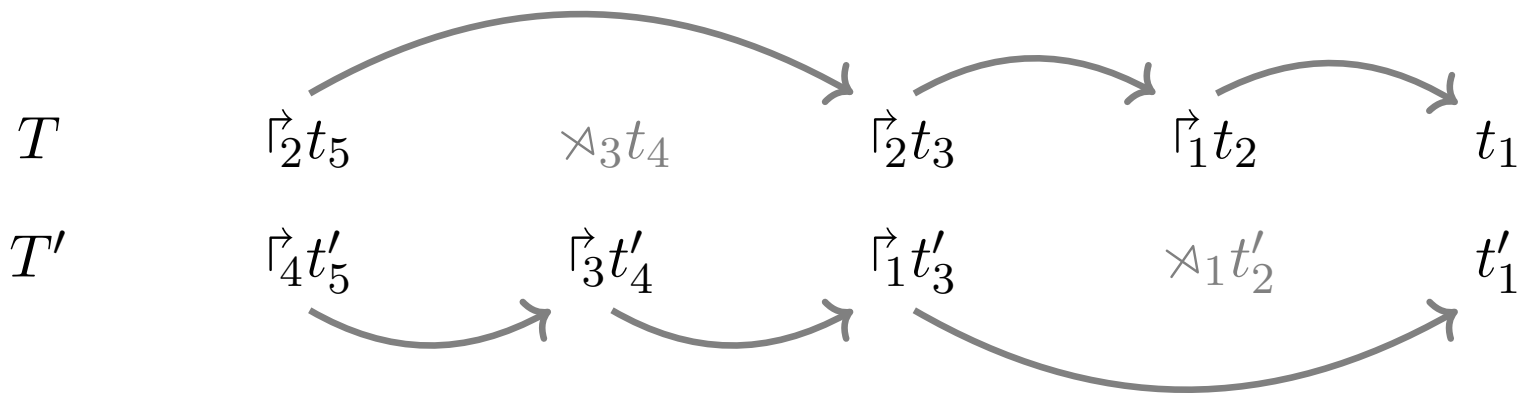
Now suppose a controller double-spends and the two halves of the double-spend find their way to the same controller (in the manner of the exponential cake). Can a later controller detect a contradiction? And if history has been reduced?

# Circuit (pair of paths)

A circuit consists of a pair of paths $T = (t_n, t_{n\text{-}1}, \ldots, t_1)$ and $T' = (t'_n, t'_{n\text{-}1}, \ldots, t'_1)$ where $t_i$ and $t'_i$ have the same controller for each $1 \leq i \leq n\text{-}1$.

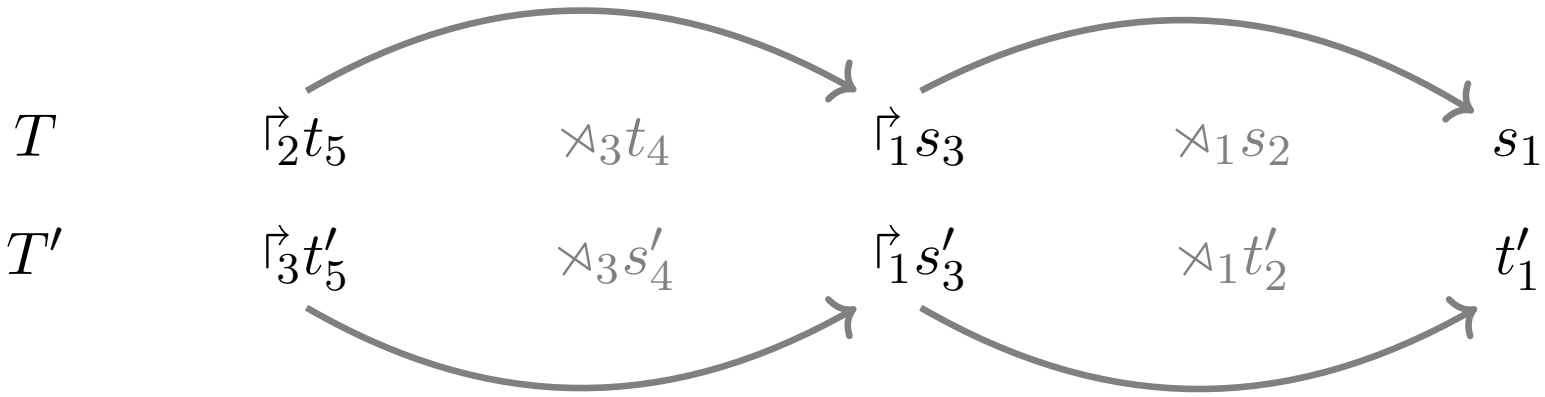
If the two paths in the circuit are reduced, can we still detect the double spend? Suppose controller $4$ has a double-spend.

If the reductions are compatible, like this . . .




. . . then we complete, like this:

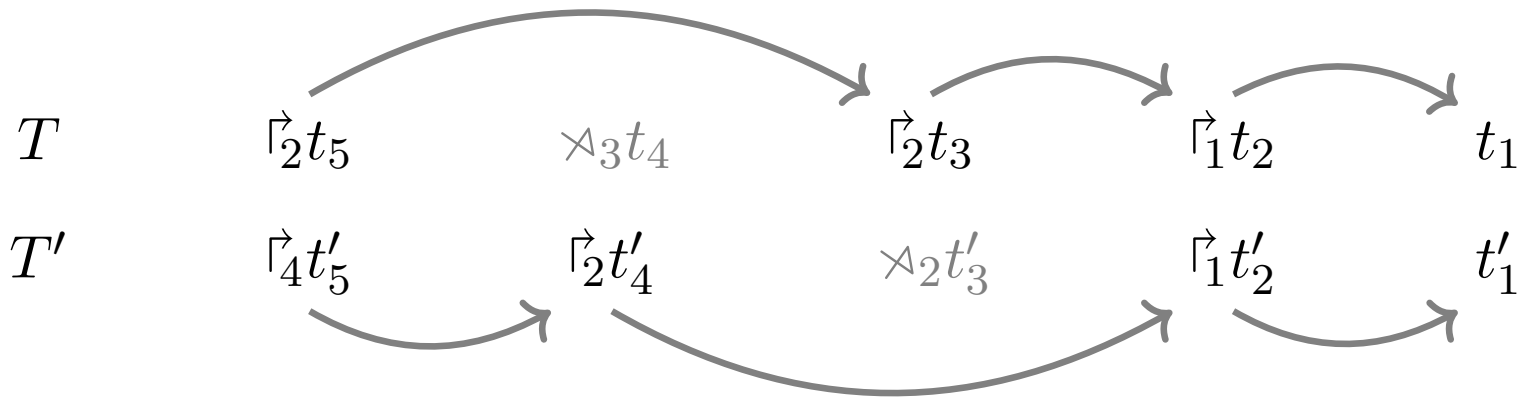
# Circuits




Now both sides of the contradictory double-spend from index $4$ will be attested to at index $3$, because index $3$ is forced to attest to $t_4$ (in the top path) and $t'_4$ (in the bottom path). So after reduction, $s_3 \lightning s'_3$.

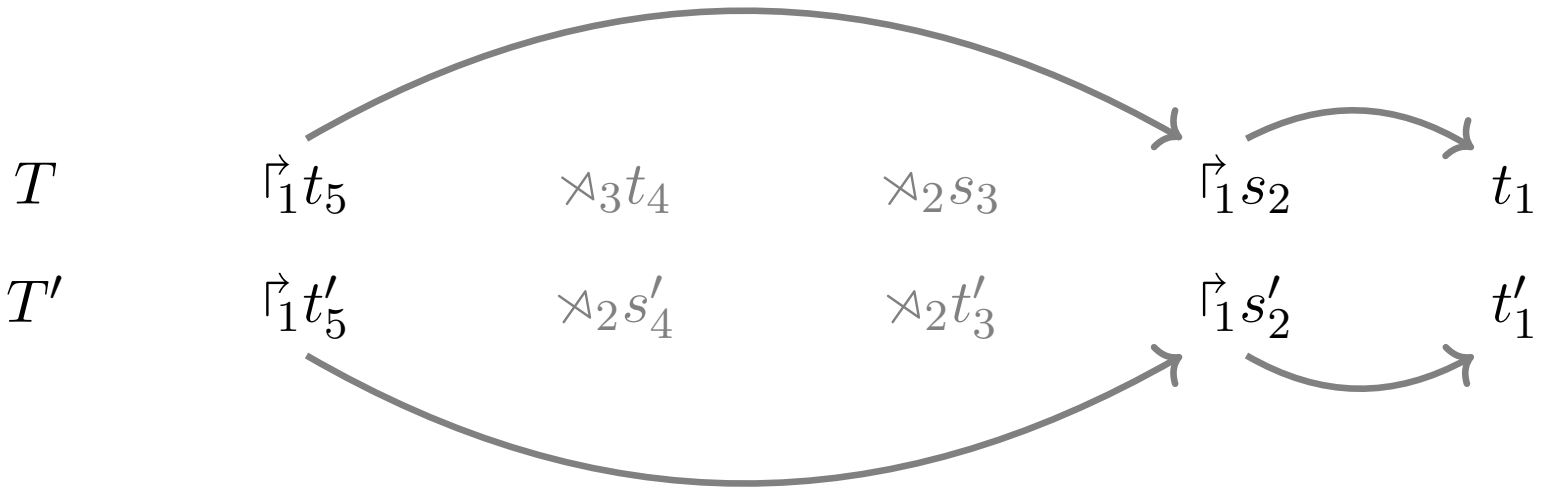
# Circuits

If the reductions are not compatible, like this ...




then we complete by cutting index $3$ in path $T$, as follows:

# Circuits




Now index 2 is contradictory: $s_2 \lightning s'_2$.

Observe that $s_2$ has attested via index 3 to $\rtimes_3 t_4$, and *also* via index 3 to $\ltimes_2 t'_4$. As discussed, $\rtimes_3(\text{-}) \lightning \ltimes_2(\text{-})$.

It's not trivial to prove, but it can be shown that *however* we play this game, we will arrive at a contradiction.

# A theorem (Thm 5.6.2, Corr 5.6.3)

Theorem: Reduction reduces a contradictory path to a contradictory path.

Proof:

- either the contradiction is propagated in both branches via compatible cuts, in which case the contradiction is propagated along attestations,
- or *incompatible* cuts are introduced, which introduces a new contradiction.

Intuitively: if a crime is covered up, then that itself just creates a new crime. It's not that contradictions can't be reduced and removed out of history — they can! But that just introduces a new contradiction.

# Key technical ideas of the proof

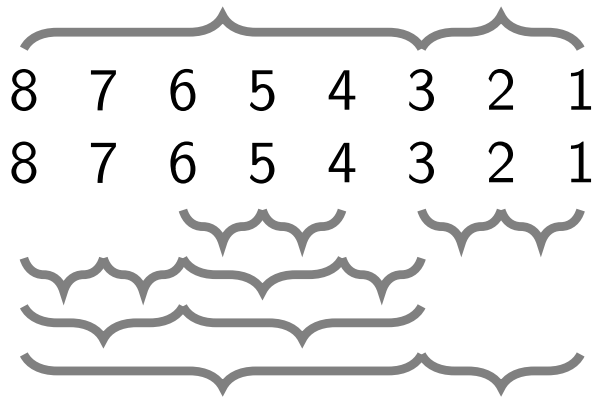
Making this all rigorous was hard work. It took nine months, sixty pages of maths, and a Lean development. The proof is technical, involving a delicate interplay between geometry, logic, and induction:

1. Geometry. Matryoshka cuts.
2. Logic. $\asymp_i t \lightning \asymp_j t'$ for any $t$ and $t'$ and any $i \leq j$.
3. Induction. The key inductive quantity is to look at the rightmost inconsistent controller.

Let me show you a picture of the Matryoshka cuts result, just because it's quite pretty:

# Geometry: Matryoshka cuts

Suppose we know that controller 8 has cut controller 3, so its next index is controller 1.




Then within this Cut must be nested — all the way down, like a series of Russian (Matryoshka) dolls — some structure of cuts cutting every index between $8$ and $3$, and every index between $3$ and $1$.

A key technical point here is that cuts nest *and also* that this nesting is 'space-filling' in the sense that every index between the outermost cut indexes ($8$ to $1$, in this example) is cut.

# Conclusions

This paper directly addresses (solves?) a deep problem in digitised assets: how to *allow* controllers to communicate assets while *preventing* misbehaviour by one controller from degrading the integrity of these assets, in a *scalable* manner.

Insisting on a complete record of histories satisfies the first two requirements but not the last. Insisting that all controllers never misbehave — essentially, restricting controllers to be permissioned national institutions — is a systems solution that does not scale (but treats this as a feature, not a bug).

I would argue that reducible controller tags point the way to something much better.

# The message: multiple controllers with reducible histories

Multiple controllers are possible. 'Blockchain fragmentation' is not a bug, it's a feature. This is a complex distributed system, so we just need to appropriate maths — e.g. reducible histories — to control it.

*This kind of challenge has been overcome before. Consider that telephone networks have multiple controllers. Remember when you could hear long-distance being connected, because the control signals used the audio lines? Remember Captain Crunch the phone phreaker?*

*'Telephony fragmentation' was a thing, and now it's not, because we developed the mathematics and the engineering to make telephony work smoothly and reliably in practice, in the presence of multiple controllers.*

# Future work

Future work is to implement both the tags and their reduction,

1. first in mathematical form as a 'real' mathematical system with real tags and resources (another paper), and
2. then as a prototype implementation.