In the context of Rollups (both optimistic and ZK), it can be the case that two implementations of the Rollups Virtual Machine (VM) can be developed that carry complementary properties. Let us call these implementations A

and AB

.

While implementation A

is fast to execute, implementation AB

is easier to use in the settlement algorithm (either fraud or validity proof). Having hybrid Rollups architectures is not a new concept, as can be seen in the cases of Arbitrum (EVM + WASM), Optimism (EVM + MIPS) and Cartesi (RISC-V and micro-RISC-V).

In such situations, using a single implementation would result in a loss for the Rollus solution:

- while using A

alone would mean that the settlement could become complex to implement (harming the security of the chain);

- using only AB

would impact the performance and consequently the block size of the system.

It is in this context that we introduce the idea of Machine Swapping, as a method to reconcile the properties of both implementations, in an attempt to preserve both security and scalability.

Running example

- throughout the post it is helpful to keep in mind an example where this could be applied. For instance A

could be thought of as a machine that processes one EVM block at each step, while AB

could be for example a MIPS or RISC-V machine emulating an EVM interpreter.

Basic Notation

To be more concrete, let us start defining the desired computation in terms of the slow machine AB

. This architecture is defined in terms of a step function (that we also call AB

) acting on a state R

. In other words, the state R_t

at time t

is updated to $R_{t + 1} = AB(R_t)$

. The objective of the overall computation is to perform a large number of steps, which we write as $2^{a + b}$

, for reasons that will become apparent soon (one of the reasons to consider powers of 2

is to facilitate interactive fraud proofs, but it not strictly necessary).

Of course the first idea to perform the execution is to implement the function AB

and iterate it $2^{a + b}$

times as required, but to avoid the performance impact we introduce the second architecture.

We denote both the machine and the step function of the second architecture with the letter A

. This machine acts on a different state (denoted here by S

) through a similar formula: $S_{t + 1} = A(S_t)$

. As it will become clear later, applying the function A

will correspond to “fast forwarding” the AB

machine by 2^a

steps.

At this point the two architectures seem completely unrelated, performing different computations on states that have different types. Let us now describe the connection between the two.

State cleaning

The most important relation between these two architectures relates their state.

We suppose that the type of the state S

(for the A

architecture) is a field in the type of R

. We can write this as R = (Q, S)

and from now on, we refer to the second field of S

as the essential state

and the Q

component as the auxiliary state

.

Running example

If our EVM vs MIPS example, one can think of S

as the EVM state, while Q

as the internal state of MIPS being used to operate on S

.

The auxiliary state will of course change during the execution of the machine, but we will assume that at all steps that are a multiple of 2^a

, the machine's auxiliary state happens to be at a fixed special value Q*

. It may be necessary to modify the architecture of AB

in order to achieve this requirement in a proces that we call state cleaning

, described below.

Suppose that the machine AB

needs to spend a number t

of steps (t

large, but no larger than 2^a

) to perform a meaningful change to the (higher level) state S

.

Of course, due to the simplicity of the architecture A

, during this process it may need to use the auxiliary state Q

for book-keeping. Suppose also that at this point the machine no longer needs the information contained in Q

, which could be safely dropped before the machine moves on to change S

again.

In this situation, the machine evolves by simply updating a clock contained in Q

, until it reaches a multiple of 2^a

. At this moment the machine makes an atomic operation that cleans the auxiliary state, effectively changing it to Q^*

.

Fast forwarding

Another important connection between the two machines relates their time evolutions.

We assume that for any essential state S

, it is always equivalent to apply a single step of the A

machine (giving $S' = A(S)$

) or to apply the procedure below:

1. attach a pristine auxiliary state Q^*

to S

: $R = (Q^*, S)$

;

1. execute the AB

architecture by 2^a

steps (obtaining $R' = (Q^*, S')$

and

1. remove the auxiliary state by extracting the second component S'

of R'

.

If these implementations are equivalent in the above sense (the diagram comutes), then we are in a good position to implement Machine Swapping.

Guaranteeing speed and security

At this point it becomes clear that the two machines can be used in tandem, but the reader could be questioning whether we can extract the advantages of the two architectures, or if we will end up with a system that is both unsafe and slow. To extract the advantages of both architectures, we are going to perform their execution in a careful way.

We start by taking the initial state S_0

and running 2^b

steps of the A

machine, while saving snapshots at each step: S_0

, S_1

, …, S_{2^b}

. We can now run the architecture AB

for each of the above states (by first attaching them with Q^*

) and double checking that S_t

indeed evolves to $S_{t + 1}$

also according to AB

.

This double checking of the transitions proposed by the architecture A

is what guarantees that the system did not lose any security (when compared with simply running AB

directly). On the other hand, note that this double cheking procedure can be done in parallel, justifying the claims on scalability gain.

Further comments

A Rollups team could decide to bypass the (intensive) double checking describe above, if they are absolutely convinced about the equivalence of the two machines. This can be done by either: a rigorous audit, a proof using formal methods or by having multiple independent implementations of the fast machine A

running together.

Another observation is that the idea of Machine Swapping can involve more layers (think of running $2^{a + b + c}$

steps on a state of type (P, Q, S)

. There are contexts for which these multiple layers can be beneficial.

PS: To the best of my knowledge, this idea was born at Cartesi during the implementation of our microarchitecture.