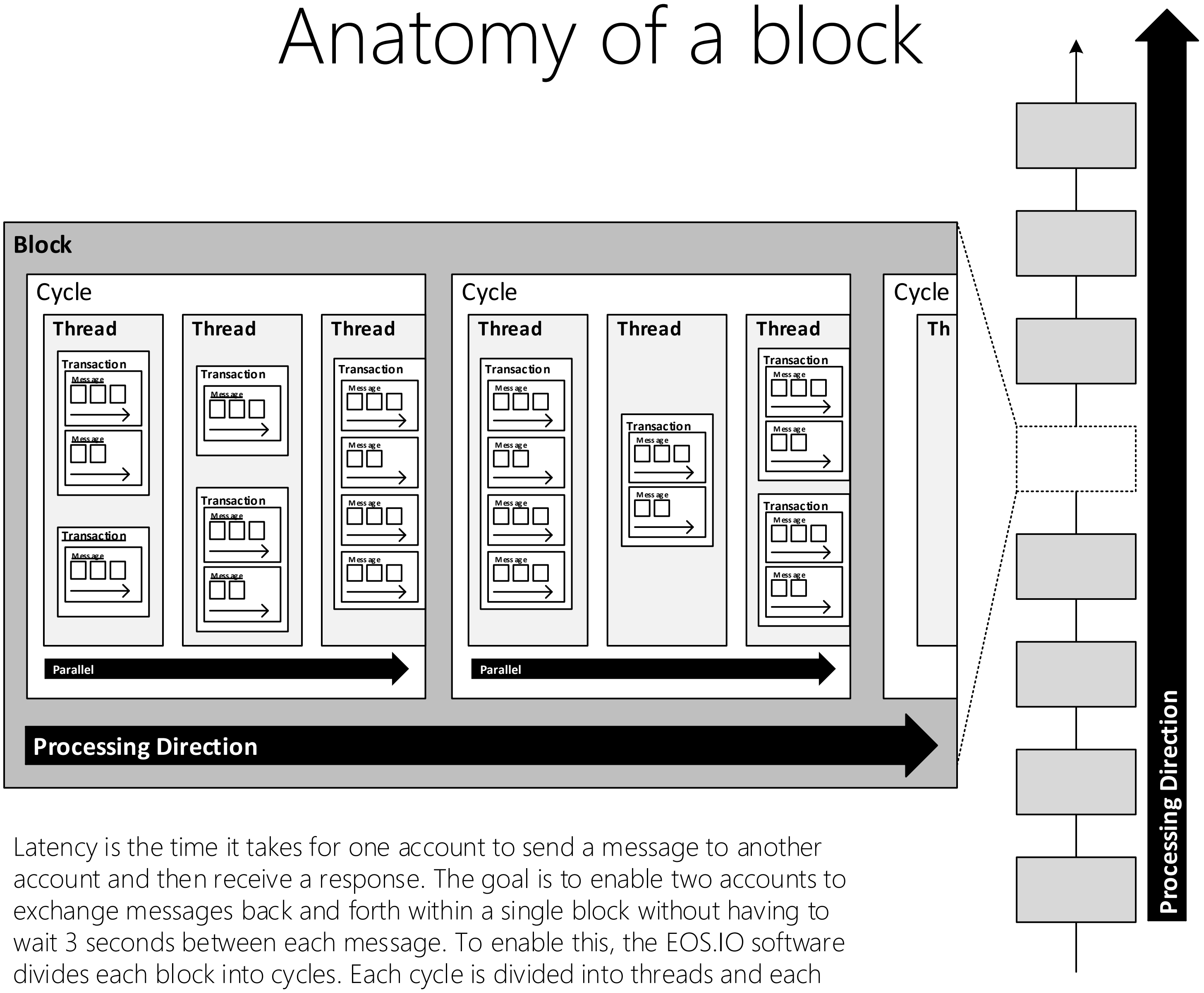
EOS
Visualized

EOS

Anatomy of a block

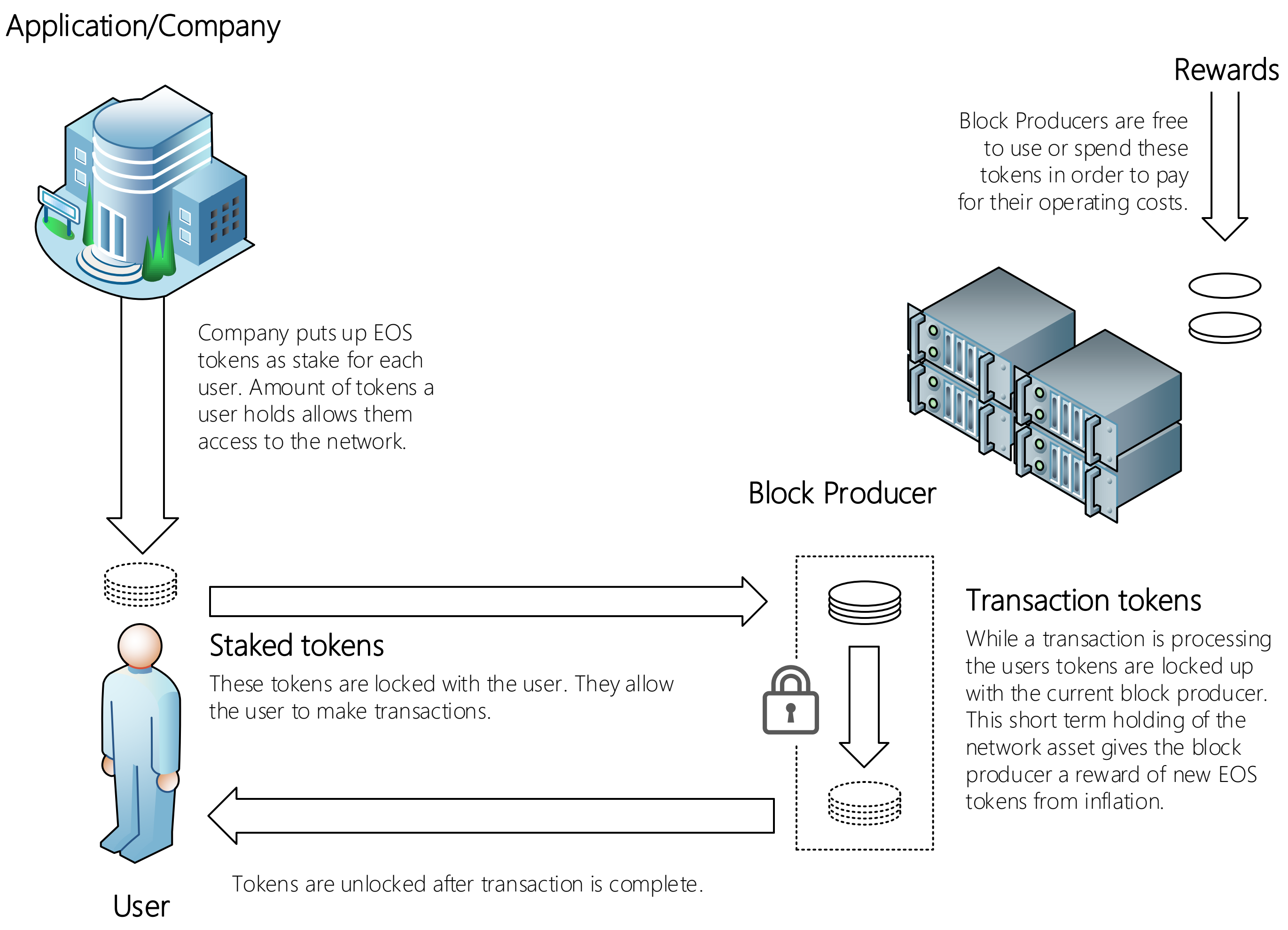


Latency is the time it takes for one account to send a message to another account and then receive a response. The goal is to enable two accounts to exchange messages back and forth within a single block without having to wait 3 seconds between each message. To enable this, the EOS.IO software divides each block into cycles. Each cycle is divided into threads and each thread contains a list of transactions. Each transaction contains a set of messages to be delivered. This structure can be visualized as a tree where alternating layers are processed sequentially and in parallel.

Transactions generated in one cycle can be delivered in any subsequent cycle or block. Block producers will keep adding cycles to a block until the maximum wall clock time has passed or there are no new generated transactions to deliver.
It is possible to use static analysis of a block to verify that within a given cycle no two threads contain transactions that modify the same account. So long as that invariant is maintained a block can be processed by running all threads in parallel.

EOS

Staking/Tokens



Block Producers are free to use or spend these tokens in order to pay for their operating costs.

Company puts up EOS tokens as stake for each user. Amount of tokens a user holds allows them access to the network.

Staked tokens

These tokens are locked with the user. They allow the user to make transactions.

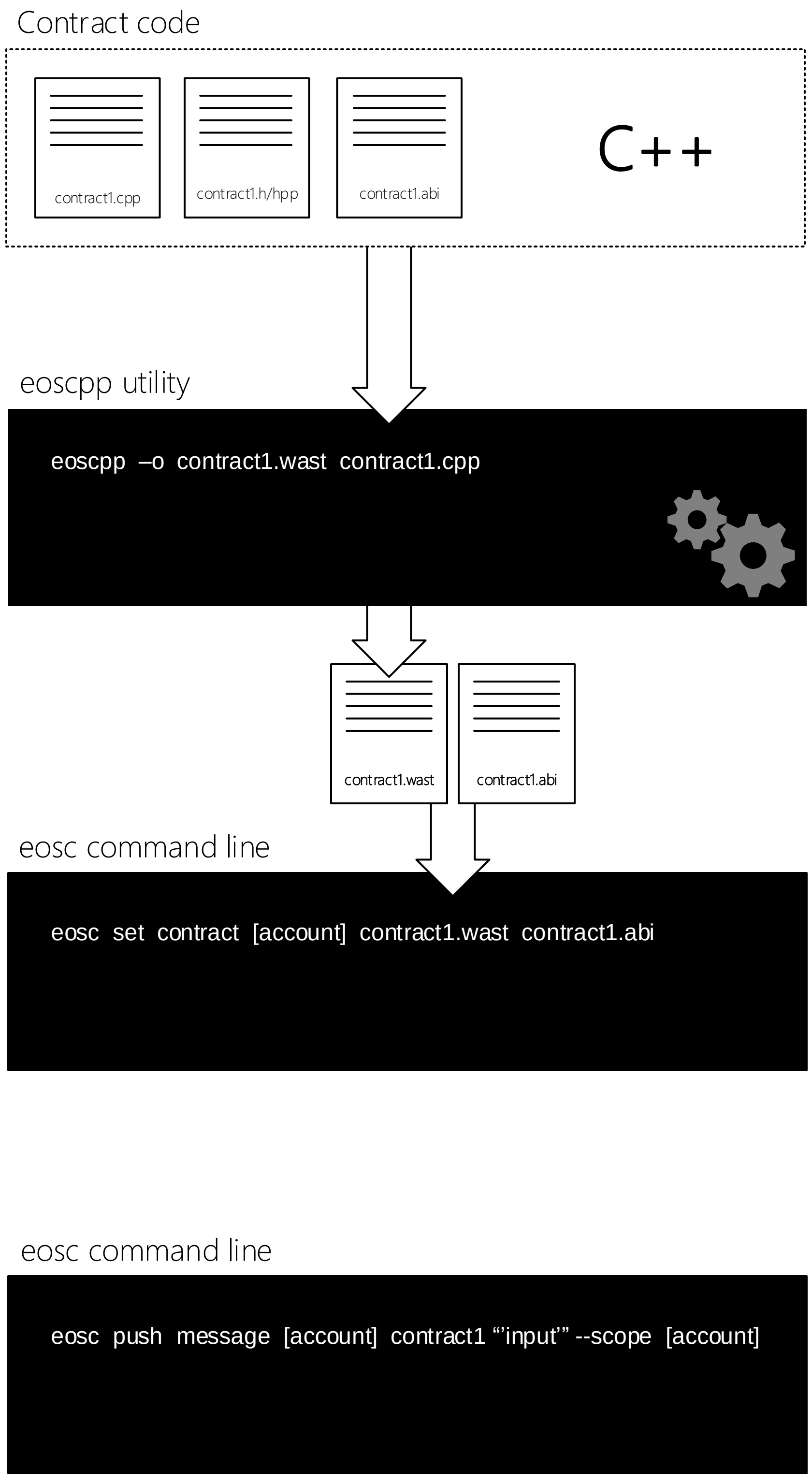
Transaction tokens

While a transaction is processing the users tokens are locked up with the current block producer. This short term holding of the network asset gives the block producer a reward of new EOS tokens from inflation.

Tokens are unlocked after transaction is complete.

EOS

Anatomy of a contract



```
eoscpp  –o  contract1.wast  contract1.cpp
```

```
eosc  set  contract  [account]  contract1.wast  contract1.abi
```

```
eosc  push  message  [account]  contract1 “’input’” --scope  [account]
```

1. Contracts are written in C++
(or any language capable of being compiled into WebAssembly)

2. Contracts are compiled into .wast files using the eoscpp utility

3. The compiled contract along with the .abi file can then be uploaded to the blockchain using the eos command line utility (eosc).

4. Contracts on the chain can now be executed using the command line.