Arbitrum:

Fast, Scalable, Private Smart Contracts

OFFCHAIN
LABS

OFFCHAIN LABS

Arbitrum is a Layer 2 cryptocurrency platform that makes smart contracts scalable, fast, and private. Arbitrum interoperates closely with Ethereum, so Ethereum developers can easily cross-compile their contracts to run on Arbitrum. Arbitrum achieves these goals through a unique combination of incentives, network protocol design, and virtual machine architecture.

This white paper explains what Arbitrum achieves for developers, how Arbitrum works, and where you can get more information.

WHY USE ARBITRUM?

Why should you use Arbitrum rather than sticking with plain Ethereum? Or from some other platform?

Arbitrum has four main advantages: scalability, privacy, the AnyTrust Guarantee of correct execution, and interoperation with Ethereum. No other solution combines these benefits.

Scalability

During normal Arbitrum operation, decentralized apps (dapps) only have to touch the main chain on startup or when they send or receive currency outside of Arbitrum. Other interactions can happen entirely offchain, with quick turnaround and guarantees of finality. Even in abnormal cases, the burden on the main chain is small compared to other solutions.

Privacy

Only the participants who validate a dapp's execution need to know what is in the dapp's code or storage. All that is published on-chain are cryptographic hashes of the dapp's state, and information about the dapp's public actions, such as sending and receiving calls, messages, and currency. The dapp developer can choose their own validators.

The dapp creator can choose to let the dapp's user see its internals if desired, and user's machines can verify the information if they receive it, but this disclosure is strictly optional, depending on the needs of each dapp.

AnyTrust Guarantee

Unlike many other state channel, sidechain, or private chain solutions, Arbitrum guarantees correct execution as long as any one validator of a dapp acts honestly—even if all of the other validators collude to (try to) cheat. Most other approaches require majority-honest or two-thirds-honest assumptions, or else require moving the entire state of a contract to the main chain in case of a dispute.

The AnyTrust Guarantee is especially nice if you are a validator. You can stand up for your own rights and force the dapp to give you the behavior and currency that you deserve, no matter what anybody else does. If all of the parties interested in a dapp are allowed to be validators, then everybody has a guarantee of correct behavior.

Interoperability with Ethereum

Arbitrum is interoperable with Ethereum. You write your dapp in Solidity, then use the open source Arbitrum compiler to generate Arbitrum-ready code. You can also transfer ether or any other Ethereum-based token back and forth between Ethereum and Arbitrum.

HOW TO RUN YOUR DAPP ON ARBITRUM

Suppose you're a developer who has written a dapp for the Ethereum platform. Arbitrum interoperates with Ethereum, so you can move your dapp over to Arbitrum and get better speed, scalability, and, if you desire, greater privacy.

You're starting with a dapp—or you're planning to develop one—that's made up of some contracts written in Solidity, along with a browser-based front end. Here's how use Arbitrum with your dapp.

First, you compile your Solidity contracts using the Arbitrum compiler. This creates an Arbitrum executable file. If you'd like, you can use Truffle to manage the compilation and deployment of your code.

Next, you identify a set of validators who will track your dapp's execution and make sure it is correct. Anybody can be a validator, and every contract creator gets to choose their own validators. The next section talks about how you might choose validators in some common use cases.

Lastly, you have the option of designating a set of observers. Observers get to see what the contract is doing but, unlike validators, observers aren't responsible for enforcing correctness.

Arbitrum's AnyTrust Guarantee says that as long as even a single one of your dapp's validators is online and acting honestly, your dapp will execute correctly. If you or someone you trust is a validator for a dapp, you can be confident that your dapp will execute correctly—even if all of the other validators are malicious.

When you're ready to launch your dapp, you send an Ethereum message to the EthBridge—the component that connects Arbitrum to Ethereum—telling it to launch your contract on Arbitrum, and identifying the contract's validators.

Your dapp is now up and running on Arbitrum . Users of your dapp can launch your existing front-end interface in their browsers. The front end will automatically interact with the running dapp through behind-the-scenes messaging with your validators.

Thanks to the EthBridge, you and your users can interact with your dapp, or send ether or any other Ethereum-based token to your dapp, and the dapp can send ether or tokens to any Ethereum user or to any contract on Arbitrum or Ethereum. You do this by depositing funds into your Arbitrum wallet, which is managed by the EthBridge.

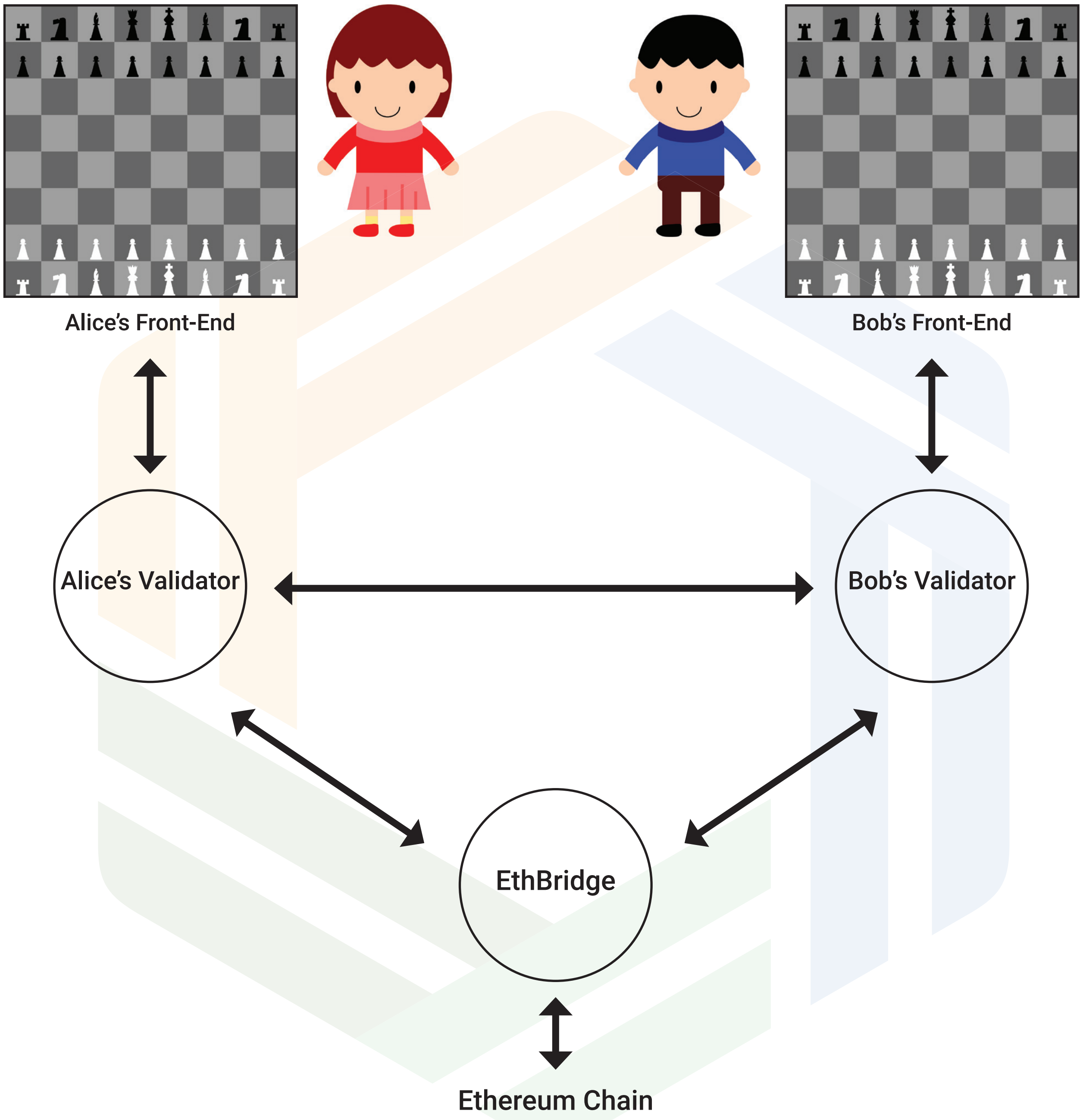
The software you need to do all of this—the Arbitrum compiler, the EthBridge, and a validator implementation—is all available from Offchain Labs via GitHub. Of course, it's all open source.

Coming soon: multi-language support: The first release of Arbitrum will run contract code written in Solidity. We know that many developers would prefer to write their code in a different programming language. That's why we're developing a compiler toolchain that allows standard languages such a C/C++, Python, Go, and Rust to be compiled to Arbitrum.

USE CASES

Arbitrum's design is versatile and can be used to run just about any dapp or smart contract efficiently. When setting up your dapp on Arbitrum, the key decision is how to choose the validators. You'll probably want to use different approaches for different categories of smart contracts.

Let's look at some use cases.



Use case: A two-player game. For many applications, it's natural to have everyone who interacts with the contract be a validator. Say Alice and Bob want to bet on a game of chess. They create a dapp, written in Solidity and compiled for Arbitrum, that encodes the rules of chess: it checks that moves are legal and when the game ends, authorizes payment to the winner. They deploy it as an Arbitrum Virtual Machine (VM).

Alice and Bob are the natural validators for this VM. Each of them trusts at least one validator—themselves—to be honest, so thanks to AnyTrust they both know the VM will execute correctly. Alice knows that Bob tries to cheat and falsely claim that he won the game, she can force the VM to pay out the coin correctly. Bob has a similar guarantee. If Alice or Bob doesn't want to be a validator, they can name somebody they trust to be a validator instead. They can each name two or three validators if they want, just to be extra sure.

A two-player game is a good example of an Arbitrum State channel, which we will describe in the appendix.

Use case: An open contest. Suppose Alice is stumped on a Sudoku puzzle and announces, "I will hereby pay 1 ETH to the first person who provides the solution." This can easily be run as an Arbitrum VM. Alice encodes her puzzle as well as the rules of Sudoku in Solidity and compiles it into an Arbitrum VM. She deposits 1 ETH into the VM, and the VM will send that prize to whoever first proposes the correct solution. If Bob wants to submit a solution, he first sends the VM a cryptographic commitment to both his identity and his answer (to ensure his place in line). He then sends another message opening the commitment. The VM will check Bob's solution, and if it is valid, it will send Bob the prize.

How should Alice choose the set of validators? It is no longer feasible to allow every contract participant to name a validator, because anybody in the world can potentially participate by proposing a solution.

Once again this is much easier thanks to Arbitrum's AnyTrust guarantee, which says that the execution of the VM can only go wrong if all of the validators collude. Alice can choose a diverse set of validators with real known identities and reputations who are highly unlikely to all collude. In Arbitrum you can always add more validators to mitigate the risk of universal collusion.

Now Bob can look at the list of validators, and as long as he trusts at least one of them, he knows that the contract will give him the payment he earned solving Alice's Sudoku.

This contest use case is a good example of an AnyTrust sidechain, which we will look at further in the appendix.

Use case: Private blockchain. While public blockchains like Ethereum are powerful, many enterprises are now turning to private, or permissioned, blockchains to coordinate with other players in their sector. Some banks, for example, are considering consortium blockchains as an interbank settlement solution. Other sectors are using permissioned blockchains to track products through their supply chain.

Private blockchains, as their name suggests, have significant privacy benefits. Only the participating nodes may see what happens on the private chain; it remains shielded from the rest of the world.

Private blockchains generally abandon the energy inefficient proof-of-work consensus that is used to secure public blockchains such as Bitcoin and Ethereum. Since the set of nodes is known and fixed, quorum-based solutions such as Byzantine Fault Tolerant (BFT) protocols from the distributed systems world can be used to gain consensus. These protocols usually have an honesty requirement that some fraction of the validators (generally ⅔) must remain honest in order to reach consensus and make progress.

The ⅔ honesty assumption can be troublesome when choosing the validators for a consortium blockchain. Consider the supply chain example. On the one hand, it makes sense to include a wide variety of businesses, both big and small, from every step of the supply chain, as validators. On the other hand, some of the larger, more reputable players may not want to include too many lesser-known players, lest ⅓ of the participants form a cartel and prevent correct execution. In such a setting, one has to be very careful in which nodes to allow as validators.

Here again Arbitrum's AnyTrust guarantee is just what is needed. In an Arbitrum private blockchain, the logic and rules of the private chain are encoded inside an Arbitrum VM, and the nodes of the private chain become the validators of the Arbitrum VM. Choosing which nodes to allow into the chain is no longer a big issue. Arbitrum's AnyTrust property allows you to add as many validators as you want. As long as your node (or any other) remains honest, the addition of more validators cannot hurt you and correct execution is guaranteed.

Arbitrum's privacy model is identical to that provided by a private blockchain: the permissioned set of participants can see all of the blockchain data, but it is completely opaque to the outside world.

Arbitrum thus provides an ideal solution for enterprises launching private blockchains, as it significantly reduces risks associated with choosing participants. You no longer need to be convinced that two-thirds of your peers will be honest. Actually, you don't have to trust any of your peers. Thanks to AnyTrust, you can unilaterally enforce correct behavior.

HOW ARBITRUM WORKS

If you're interested in the nuts and bolts of how Arbitrum really works, you can read the code; or you can dive into some of the deep details by reading the academic paper[1] that Arbitrum is based on, although that's a snapshot from May 2018, so it's missing some important features of today's Arbitrum. Or … keep reading and we'll summarize it.

We'll explain the nuts and bolts of Arbitrum in terms of its three main components: the compiler, the EthBridge, and validators.

The Arbitrum **compiler** takes a group of contracts written in Solidity, and compiles and links them together into a single executable file that can run on the Arbitrum Virtual Machine (AVM) architecture. (Why did we design our own VM architecture? Because it lets us reduce the on-chain footprint of a VM, which makes things go faster. See below for a detailed explanation.)

Because many dapps are written as a group of contracts rather than a single contract, the Arbitrum compiler can take all of the contracts in your dapp and compile them together into a single deployable unit. To avoid confusion between a compiled Arbitrum program and the multiple contracts that make it up, we call the whole thing a Virtual Machine (or VM) when it's running.

The **EthBridge** is a dapp running on Ethereum. As the name implies, its job is to serve as a bridge between Ethereum-land and Arbitrum-land. Anyone in Ethereum-land can call the EthBridge to interact with Arbitrum-land, for example to launch an Arbitrum VM, to make a call to a contract running on Arbitrum, or to send ether or any token to an Arbitrum VM. If an Arbitrum VM sends ether or tokens to you, the EthBridge will pass them on to you. And the EthBridge knows the status of everything in Arbitrum so you can make non-mutating calls to it (costing zero gas) to follow what's happening in Arbitrum-land (except for the internals of private VMs).

[1] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S. Matthew Weinberg, and Edward W. Felten. "Arbitrum: Scalable, private smart contracts." In 27th USENIX Security Symposium, pp. 1353-1370. 2018. Available at https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-kalodner.pdf

The EthBridge's other important job is to referee disputes between validators. As described below, the Arbitrum protocol and AVM architecture make dispute resolution very fast and cheap, as well as imposing strong disincentives against entering disputes in the first place. But if a dispute does occur, the EthBridge will referee it to preserve the AnyTrust Guarantee.

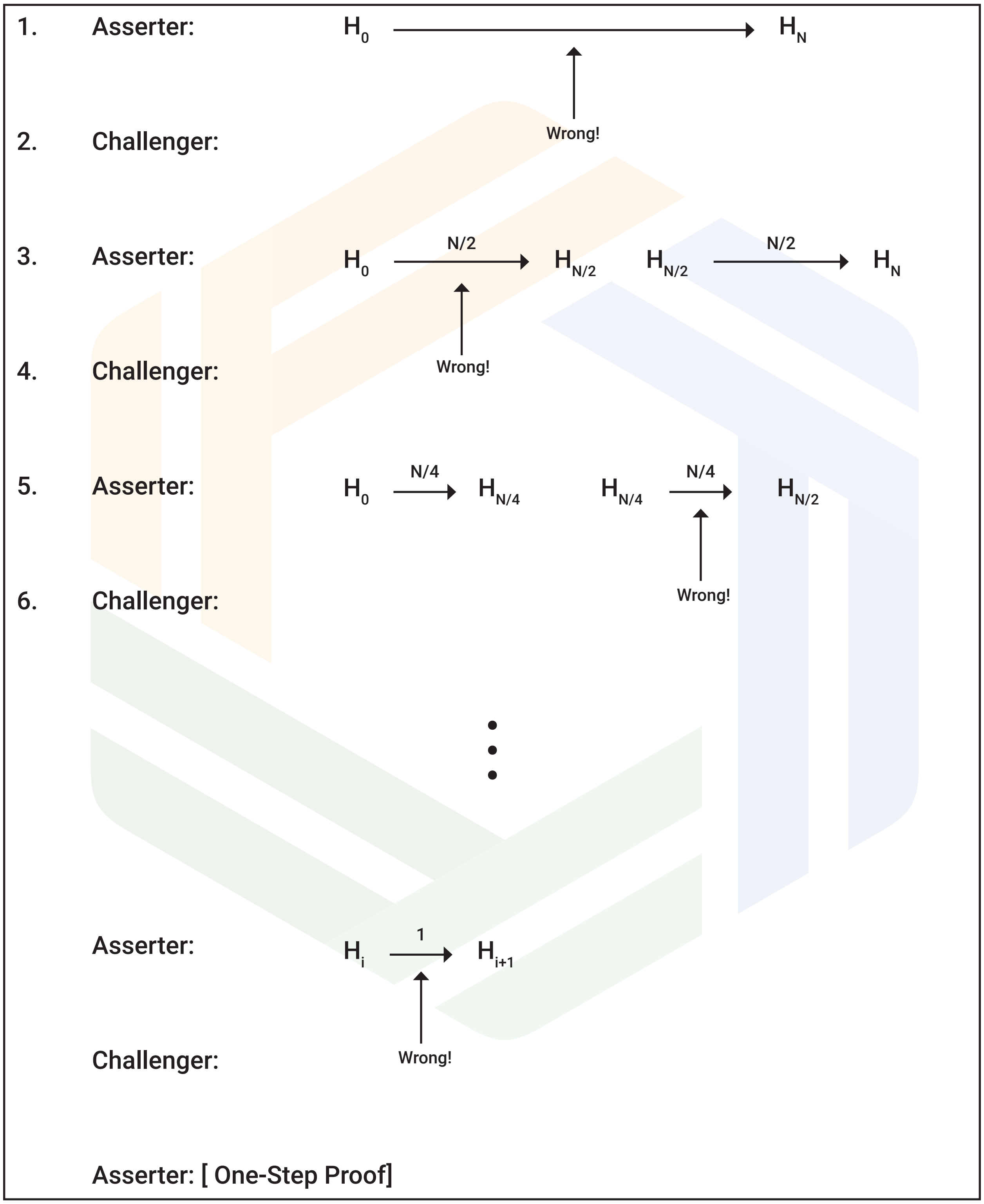
Validators are the workhorses of Arbitrum—and they do that work almost entirely off-chain. Recall that every running VM has its own validators who are chosen by the dapp creator. Those validators keep track of the state of the VM and advance the VM's state by emulating the instructions in the VM's code.

A VM's validators know everything about the current state of that VM. The EthBridge just keeps track of the cryptographic hash of each VM's state. We can think of that hash as a commitment by the VM's validators to the complete state.

The common case: fast path

Most of the time, a VM's validators will agree on what the VM does. Each validator will emulate some steps of execution by the VM, and then the validators will compute the cryptographic hash of the resulting state. If the validators are all behaving correctly, they will agree on the new state hash. They will then make a joint digital signature recording the VM's evolution from the old state hash to the new state hash. That's called a unanimous assertion. The EthBridge will accept any **unanimous assertion** as correct. (That's consistent with the AnyTrust Guarantee, because assuming that at least one validator is honest, anything that was endorsed by all validators must have been endorsed by an honest validator.)

Usually a VM's validators will be able to string together a whole sequence of unanimous assertions, without needing to bother the EthBridge or put anything on the main chain. A unanimous assertion is like a voucher that can always be taken to the main chain (via the EthBridge); or you can hang onto a voucher until you get a newer, more up-to-date voucher. Honest validators will do this for you, so you have a guarantee of finality in unanimously asserted transactions, even before they're submitted to the EthBridge to be put on the main chain. This mode of execution can give finality very quickly—it requires only agreement and a digital signature from your VM's chosen validators—while still supporting the AnyTrust Guarantee of correctness.


1. Asserter: H_0 → H_N
2. Challenger: Wrong!
3. Asserter: H_0 N/2 → $H_{N/2}$ $H_{N/2}$ N/2 → H_N
4. Challenger: Wrong!
5. Asserter: H_0 N/4 → $H_{N/4}$ $H_{N/4}$ N/4 → $H_{N/2}$
6. Challenger: Wrong!
Asserter: H_i 1 → H_{i+1}
Challenger: Wrong!
Asserter: [One-Step Proof]

If a validator isn't available

Occasionally a validator might be unavailable or offline. In this case, unanimous agreement can't be reached. To make progress, any validator can make a disputable assertion: an on-chain claim, submitted via the EthBridge, that asserts what the correct next state hash and actions of the VM are. After this disputable assertion, the other validators have a time window, whose length is determined by the dapp creator, to dispute the assertion if they believe it incorrect. If the problem is just that one validator is unavailable, the assertion will be honest and correct, and nobody will dispute it. After the dispute window closes, the EthBridge will accept the assertion as correct, allowing the VM to make progress.

As soon as the unavailable validator comes back online, the validators will go back to the normal, unanimous, off-chain operation of the VM.

If a validator is dishonest

The most challenging case arises when a dishonest validator tries to cheat. This is the case where many state channel approaches get into trouble and have to move the full state of the dapp onto the main chain, which can be very expensive and slow. This case is where Arbitrum really shines.

There are two things a dishonest validator might try to do. First, they might try to corrupt the execution of the VM by making a disputable assertion that is false in the hopes that the EthBridge will accept that assertion. Second, they might let somebody else make a truthful disputable assertion and then start a bogus dispute about it.

Arbitrum deters misbehavior by requiring validators to put down a deposit or "stake" that is held by the EthBridge. If a validator gets caught cheating, it will forfeit its stake. (Half of the cheater's stake goes to the other party in the dispute. The other half is burned (that is, destroyed). When the VM completes, validators who haven't been caught cheating will get their stake back.

Arbitrum's dispute resolution protocol is designed to resolve disputes very efficiently, by identifying a cheating party while requiring a minimum of on-chain activity. The dispute plays out as a contest between an asserter (who claims their disputable assertion is correct) and a challenger (who claims that the same assertion is wrong), with the EthBridge acting as referee.

Dispute resolution

The dispute resolution protocol goes in two stages. First, the players narrow down their disagreement, using an on-chain bisection protocol, until they are disagreeing about a single step of the VM's computation. Then the asserter sends a one-step proof—a proof of correctness for the execution of a single VM instruction—to the EthBridge which checks that tiny proof.

The bisection protocol starts when the asserter claims that a VM starting with state-hash X can execute N instructions, resulting in state-hash Y, and the challenger responds that this is false.

The asserter is required to bisect their assertion into two half-size assertions, each involving the execution of N/2 instructions, which fit together to give the initial assertion. Then the challenger has to pick one those two halves to challenge. If either player fails to act within a time limit, they lose. After one round of bisection, the size of the dispute has been cut in half to N/2 steps.

The process continues with further bisections. After a logarithmic number of bisections (logarithmic in the number of instructions executed, N), the dispute has been narrowed to a single step: the dispute will be over whether or not a VM starting with state-hash Y can execute a single instruction to get to the state-hash Z.

At this point the asserter has to give a one-step proof to the EthBridge, containing the information that the EthBridge needs to quickly verify that the one-step assertion is correct. Or perhaps the asserter will fail to provide a valid one-step proof. Either way, the dispute is resolved.

We won't go into the details of one-step proofs here—we're pretty deep in the weeds already. Suffice it to say that Arbitrum's custom VM architecture comes into play here, making the one-step proof small (a few hundred bytes at most) and quick to check (costing about 90,000 Ethereum gas, or about four cents at current prices).

By making disputes relatively cheap to resolve, and imposing a substantial penalty on the loser, Arbitrum strongly disincentivizes attempts to cheat, but even if a dispute occurs this doesn't impose a huge on-chain impact. In the common case, validators will agree and progress will occur off-chain, with only occasional touches to the main chain.

THE BOTTOM LINE

Arbitrum can do what other smart contract scalability solutions can't. Arbitrum makes your dapp scalable, by moving almost all of its activity off-chain and making your dapp's cost independent of the size of its code and data. Arbitrum preserves privacy by letting you choose who gets to see what your dapp is doing. Arbitrum gives you confidence in your dapp's correctness due to the AnyTrust Guarantee which lets you pick your validators and requires only one honest validator to ensure correct behavior. And Arbitrum is compatible with Ethereum.

If you're developing a dapp and you care about scalability, privacy, security, and ease of development, we invite you to try Arbitrum.

GETTING STARTED WITH ARBITRUM

For information on the status of Arbitrum, and instructions on how to download the Arbitrum code and get started, see the developer quickstart guide at https://developer.offchainlabs.com.

ABOUT OFFCHAIN LABS

Offchain Labs, Inc. is a New Jersey based startup that aims to make smart contracts more private, secure, and scalable using a combination of protocol design, incentives, and virtual machine architecture. Led by Princeton Professor of Computer Science Ed Felten and a team of academics and PhD technologists, Offchain Labs has invented Arbitrum, a blockchain-agnostic Layer 2 scaling solution that incentivizes parties to agree off-chain how a virtual machine would act in order to improve transaction throughput, speed, and privacy.

For more information on Offchain Labs, see https://offchainlabs.com

APPENDIX: ARBITRUM, STATE CHANNELS, AND SIDE CHAINS

Blockchain experts sometimes ask us, "Is Arbitrum a state channel, or is it a sidechain?" The answer is that it isn't exactly either one. We think of Arbitrum as a more general approach that can be used in a state-channel style, or in a sidechain style.

Using Arbitrum in State Channel Style

State channels. We talked above about the example of a two-player chess game, where each player themselves served as a validator. Choosing the validators this way is not limited to chess and it's not even limited to two-party VMs. Any VM that has a fixed set of participants who are known ahead of time can operate in this style by allowing each participant to name (or themselves be) a validator.

This approach, where a fixed number of participants each represent their own interest by unanimously agreeing to off-chain activity and reporting the results on-chain, bears a close resemblance to *state channel* constructions, and indeed we sometimes call this an *Arbitrum state channel.*

Arbitrum state channels have significant benefits over most other state channel approaches:

> **Progress doesn't require unanimity.** Traditional state channels are highly efficient when all participants can reach unanimous agreement about what a contract did off-chain. Indeed, in this case Arbitrum state channels behave exactly like traditional state channels.
>
> It's not always possible though for validators to unanimously agree off-chain. There might be a dispute among the validators as to what occurred off-chain, or one party might crash or lose internet connectivity. Traditional state channels cannot differentiate between these two failure cases. Whenever unanimous agreement can't be reached—for any reason—traditional state channels can no longer make progress off-chain, and require the contract's code to be executed on-chain.
>
> Arbitrum differentiates between non-unanimity due to lack of availability and non-unanimity due to a dispute. Only in case of a true dispute does Arbitrum's lightweight dispute resolution protocol run. If one validator goes offline, Arbitrum still allows for execution without having running the code on-chain.

Ultimately, it's every validator's responsibility to ensure that they are available enough to monitor and ensure the correctness of a smart contract's execution by deterring bad behavior by other validators. However, if one validator shirks this responsibility, Arbitrum allows the other validators to continue executing the VM. Crucially, Arbitrum guarantees that other validators will not be able to abuse this feature by pretending that an available validator is absent.

Efficient dispute resolution. In any state channel system disputes between validators may arise Arbitrum makes dispute resolution highly efficient. The cost of resolving a dispute is small, and is independent of how big a VM's code is or how much storage it uses. (See the main text of this white paper for details.) Traditional state channels give no such guarantee. The on-chain dispute resolution can be large, complex, and expensive to run; it also sacrifices VM privacy.

Ease of development. Traditional state channels can put a significant burden on the developer by requiring smart contracts to be coded so that state transitions can be run either on- or off-chain. in a very specific way that specifies state transitions in a way that allows for each transition to be run on-chain when off-chain agreement is not reached. kDevelopers are restricted as they must make sure that each state transition in their protocol is simple enough to run on-chain if necessary. Complex state transitions that would require too much gas can't easily be supported.

Arbitrum, in contrast, allows the developer to write code in the normal way. In fact, the Arbitrum compiler can take existing Ethereum contracts and convert them to Arbitrum contracts with no extra development work.

Whereas in traditional state channels the state transitions are higher level protocol transitions, in Arbitrum state channels, the state transitions are the virtual machine instructions in the compiled code. These are generated by the compiler, not specified by the user. Moreover, the set of virtual machine instructions is identical for every contract, no matter what the higher level code is doing. Thus, the on-chain component for Arbitrum needs only to be able to execute the virtual machine instructions, but it does not need to be customized for each individual application.

Arbitrum AnyTrust Sidechains

Running an open contest on Arbitrum, as we saw in the main body of this white paper, is an example of "open world" contracts, in which there are no fixed sets of participants and sets of designated validators must be chosen to represent the varied interests of all participants. Other examples include betting and prediction markets in which anyone in the world is free to bet on the outcome of a particular event. In these settings, the set of validators operates a sidechain. Anybody else can interact with the sidechain, and it is guaranteed to operate correctly so long as any single validators is honest and available. The AnyTrust property distinguishes Arbitrum sidechains from many other proposals. Complex protocols and threshold honesty assumptions are not needed. All that's needed to guarantee correct execution of contract code is that a single validator remains honest and available.

Contract isolation. No matter how hard one tries to choose a robust set of high-reputation validators, it's unlikely that everyone in the world will be satisfied. It's important to remember that the validators can only affect the execution of the particular contract for which they are designated. In particular, even if they are all dishonest, they cannot harm any other contract or system component, so you can safely ignore any contract whose validators you do not trust.