The Ethers.js Library

A complete and compact library for interacting with the Ethereum Blockchain and its ecosystem.

Designed for the Modern Ethereum Developer

Secure & Safe

- Keep your private keys in your client, safe and sound.
- Import and export BIP 39 mnemonic phrases and HD Wallets.
- Full support for standard JSON wallets (Geth, Parity).

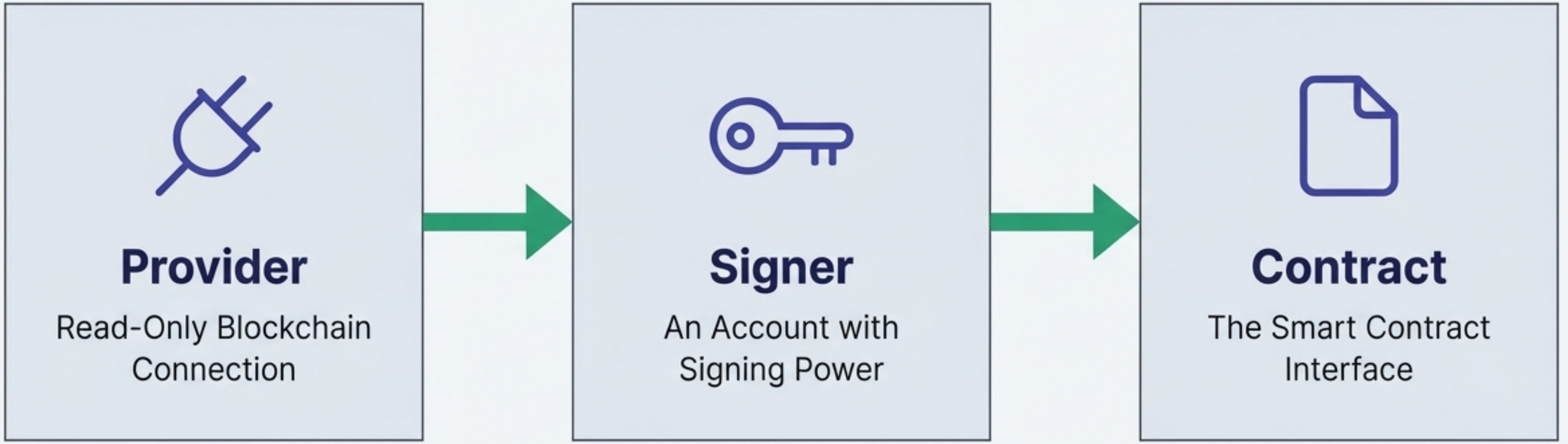
Complete & Comprehensive

- Full functionality for all Ethereum needs in a tiny footprint (~88kb compressed).
- Meta-classes for any contract ABI, including ABIv2.
- Extensive, well-maintained test cases provide reliability.

Excellent Developer Experience

- Fully TypeScript ready with complete source and definition files.
- ENS names are first-class citizens, usable anywhere an address is.
- Permissive MIT License for all code, including dependencies.

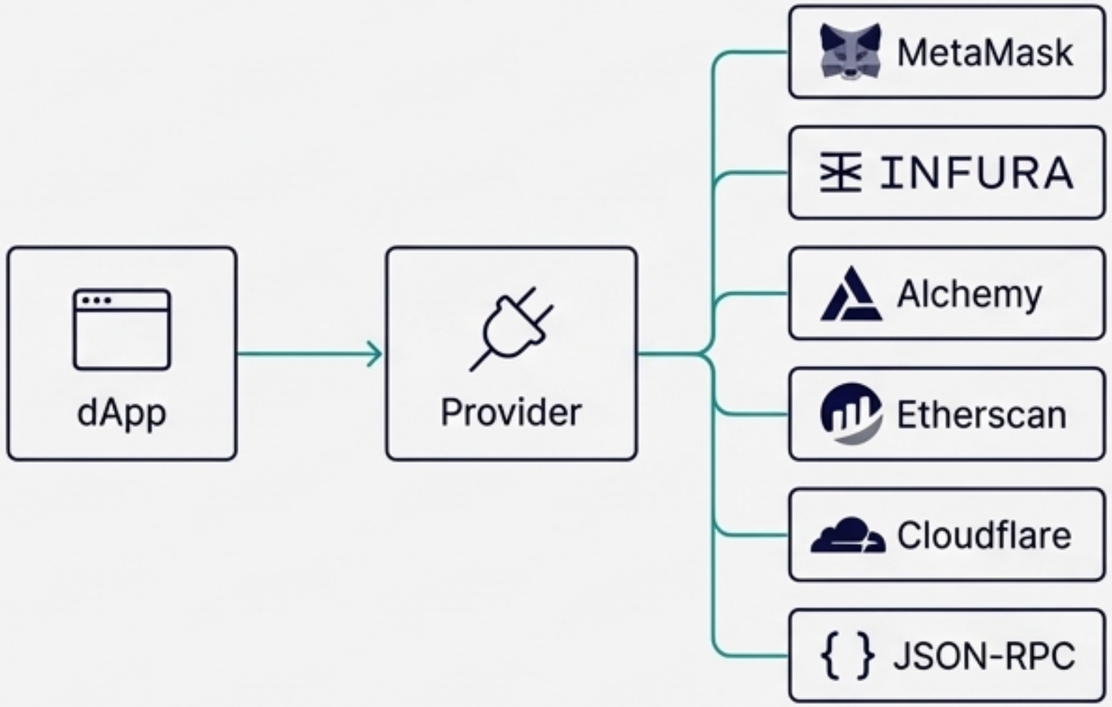
The Core Architecture: A Mental Model for dApp Development



Understanding ethers.js starts with three core concepts: the **Provider** to connect to the network, the **Signer** to represent an account, and the **Contract** to interact with your code on-chain. We'll explore each one.

Step 1: Connecting to the Blockchain with a Provider

A Provider is an abstraction for a connection to the Ethereum network, providing a read-only interface to the blockchain and its state.



Supported Connections

- JSON-RPC (via `JsonRpcProvider`)
- MetaMask (via `Web3Provider`)
- **API Providers**: INFURA, Etherscan, Alchemy, Cloudflare, Pocket, Ankr

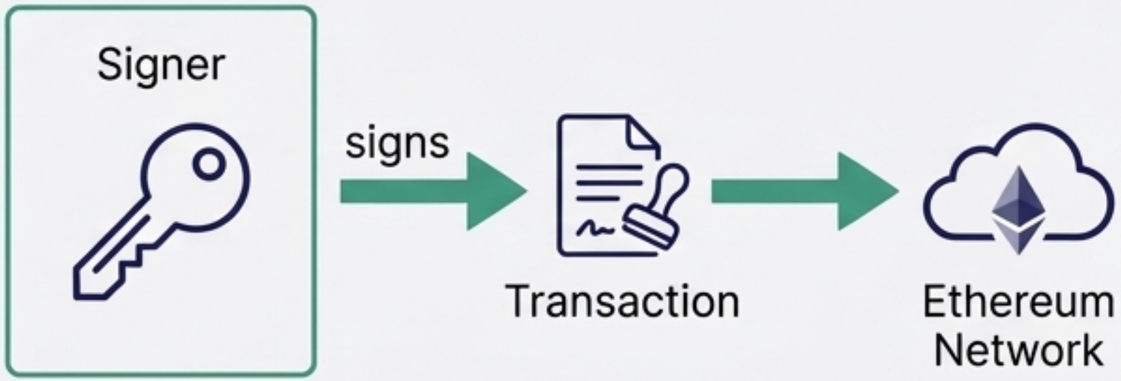
```
// Connect to the network using a default provider
const provider = ethers.getDefaultProvider("homestead");
```

Step 2: Gaining Write Access with a Signer

A Signer is an object that can sign messages and transactions and send them to the Ethereum Network to execute state-changing operations. It's an abstraction of an Ethereum account.

Key Classes

- **Signer**: The abstract class for any Ethereum account.
- **Wallet**: The most common implementation, which wraps a private key. It can be used to sign transactions and messages directly.
- **JsonRpcSigner**: Acquired from a `JsonRpcProvider` (like MetaMask) to use the node or service's account.




```
// Create a wallet instance from a private key and connect it to a provider
const signer = new ethers.Wallet(privateKey, provider);
```

Step 3: Interacting with Smart Contracts

The `Contract` object is an abstraction which represents a specific contract on the Ethereum Network. It allows you to call contract methods as if they were native JavaScript functions.

Key Classes

- `Contract`: For connecting to an existing, deployed contract.
- `ContractFactory`: A utility class for deploying new contracts.


```
const daiAddress = "0x6B175474E89094C44Da98b954EedeAC495271d0F";
const daiAbi = [ /* ABI content */ ];

// Create a contract instance
const daiContract = new ethers.Contract(daiAddress, daiAbi, signer);
```

Speaking the Language of Contracts: The ABI

Ethers provides a powerful `Interface` class for working with contract ABIs. It offers multiple formats, including a unique, highly intuitive Human-Readable format.

Solidity JSON ABI

```
json
[
  {
    "type": "function",
    "name": "balanceOf",
    "inputs": [{"name": "owner", "type":
"address"}],
    "outputs": [{"name": "balance", "type":
"uint256"}],
    "stateMatability": "view"
  }
]
```

Human-Readable ABI

```
typescript
[
  "function balanceOf(address owner) view
returns (uint256 balance)"
]
```

Simplify your code and improve readability with Human-Readable ABI.

ENS Names are First-Class Citizens

In ethers.js, Ethereum Name Service (ENS) names can be used anywhere an Ethereum address is required. The library automatically resolves them, simplifying dApp development and improving user experience.

Without ENS:

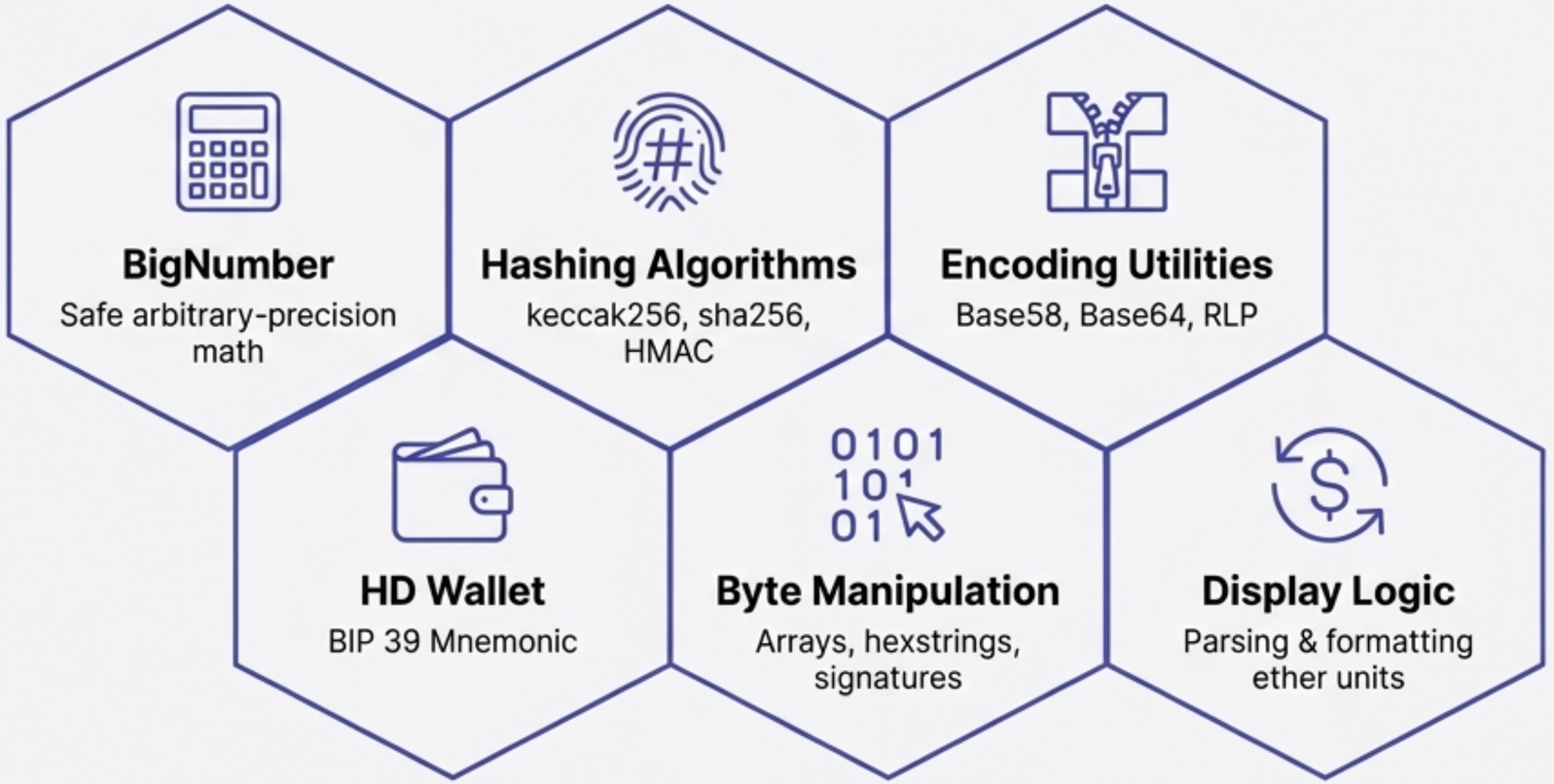
```
const balance = await dai.balanceOf("0
x5555555555555555555555555555555555555
5");
```

With Ethers & ENS:

```
const balance = await dai.balanceOf(
  "ricmoo.eth"
);
```

Beyond the Core: The Developer's Utility Belt

Ethers includes a vast collection of utilities to handle the common (and uncommon) data types and cryptographic needs of Ethereum development.



A Mature and Extensible Ecosystem

The ethers.js project extends beyond the core library, offering powerful tooling and support for a professional development workflow.

Hardware Wallets

Direct support for hardware signers like Ledger (`LedgerSigner`).

Command Line Interfaces (CLI)

Includes a suite of tools for common tasks: Sandbox, Assembler, ENS lookup, and more.

Cookbook & Guides

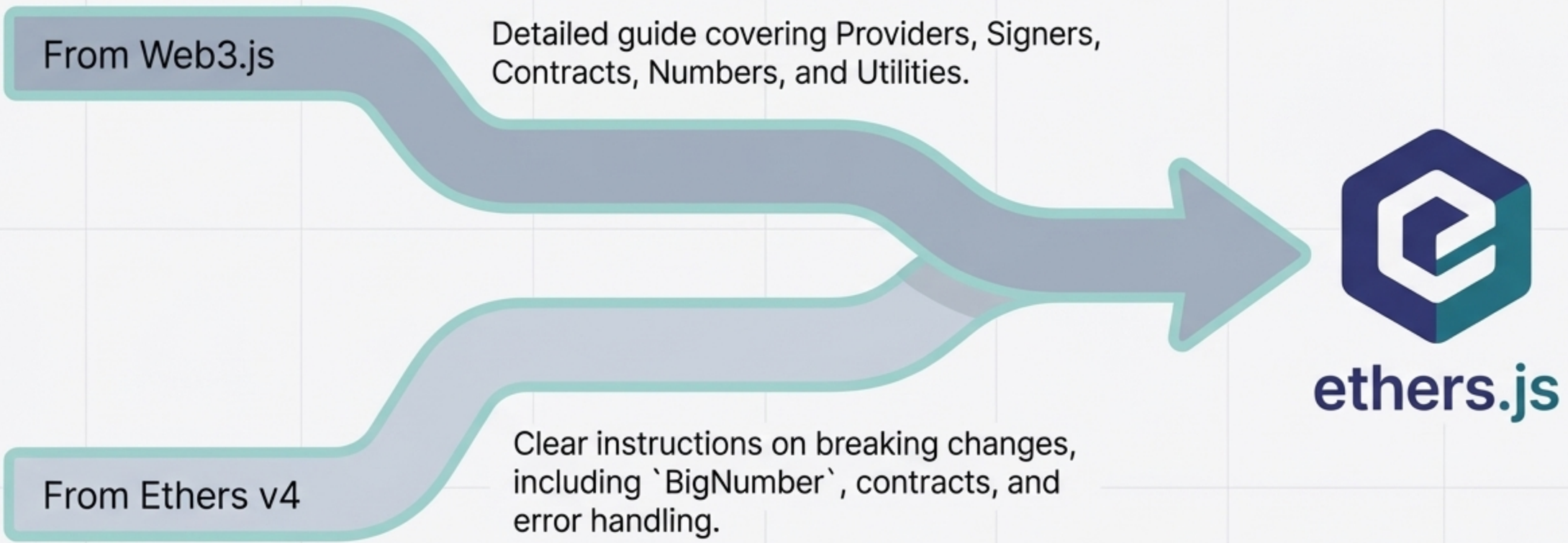
Practical recipes for specific platforms like React Native.

Plugin Support

Extensible architecture for creating your own plugins.

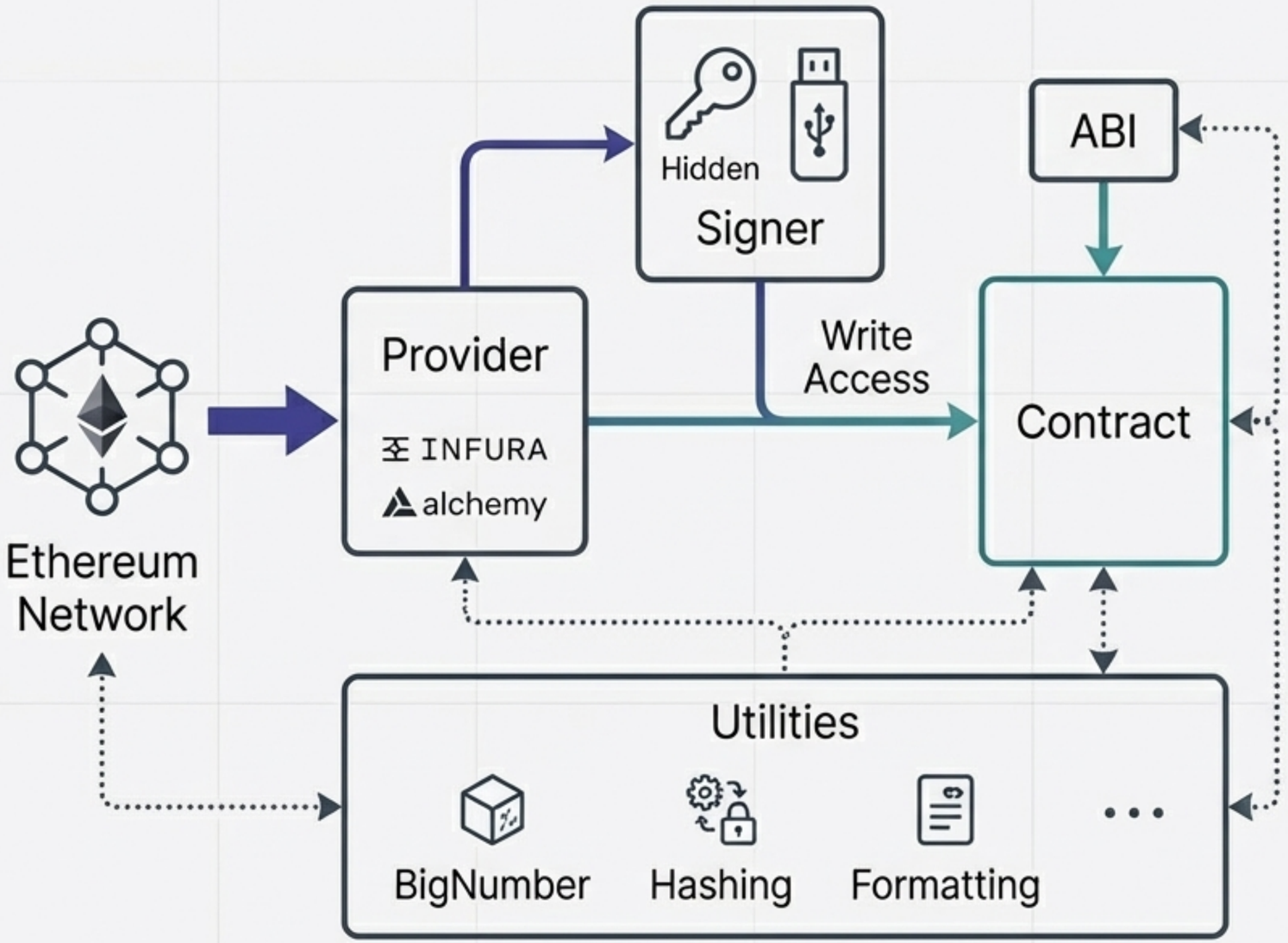
Seamless Integration and Migration

Ethers provides comprehensive guides and a logical API structure to make it easy to adopt, whether you're starting fresh or migrating from another library.



A well-documented migration path reduces friction and accelerates development.

Ethers.js: The Complete & Compact Ethereum Toolkit



Intuitive Architecture

A logical mental model based on Providers, Signers, and Contracts.

Developer-First Features

TypeScript, Human-Readable ABI, and First-Class ENS.

Comprehensive & Reliable

A complete set of utilities backed by extensive testing.