Oracles:
Explained

Nikhil • Soso • Peter

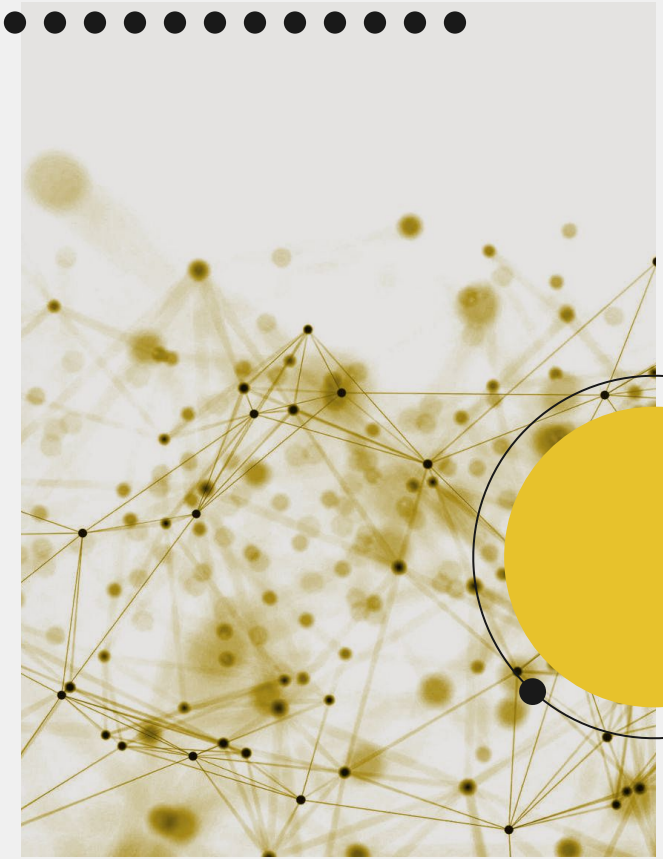
Table of Contents

Why Oracles?

Why should we use Oracles, what are they and why are they important?

Why Oracles?

Smart contracts running on blockchain networks have significant potential to **increase efficiencies** and **reduce transaction costs** across an array of industries.

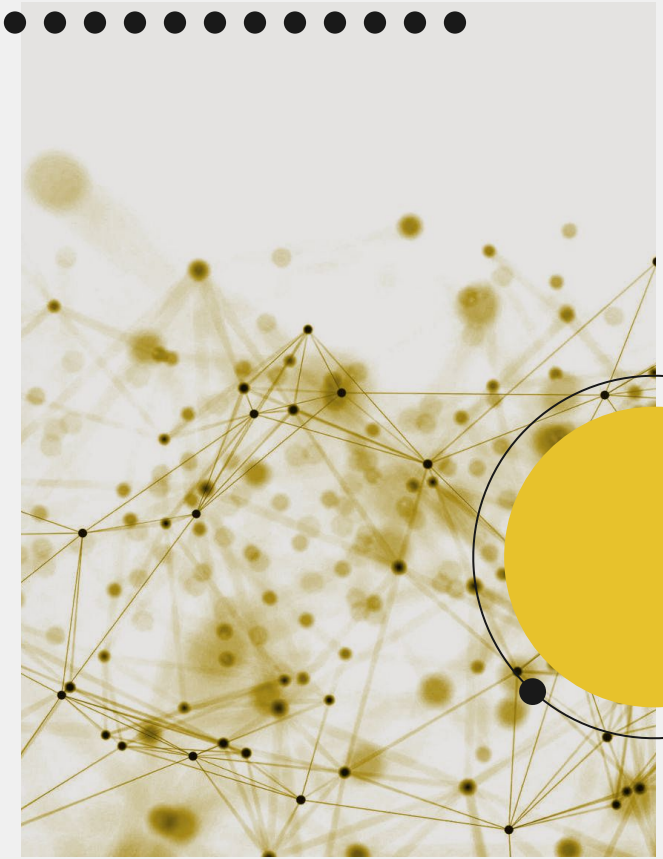


Why Oracles?

Smart contracts running on blockchain networks have significant potential to **increase efficiencies** and **reduce transaction costs** across an array of industries.

Smart contracts effectively **minimize counterparty risk** and provide **transparency**, accomplishing **trustless execution**, but they still face several **limitations** to their capacity.

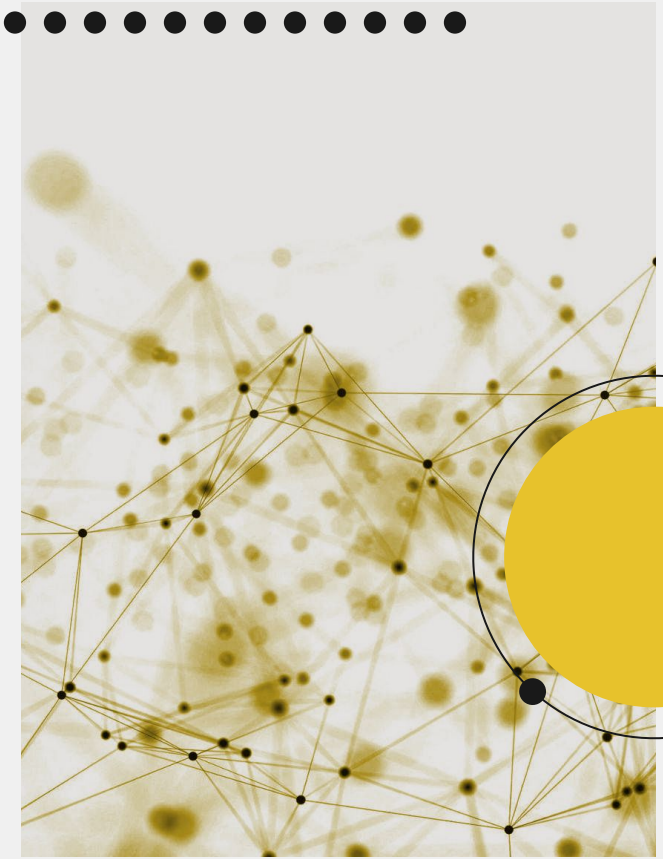
One of these limitations is access to **offchain data.**



What are Oracles?

Oracles act as the **bridge** between the blockchain and external real-world data.

Since blockchains are isolated systems, we have Oracles **retrieve** and **verify** external data for blockchains and smart contracts through web **APIs** or market **data feeds**, and any other form of external data.



02

Types of Oracles

What are the different types and kinds of oracles?

Types of Oracles

01 Hardware Oracles

02 Software Oracles

03 Consensus Oracles

04 Inbound Oracles

05 Outbound Oracles

Hardware Oracles are sensors that integrate with tangible physical objects.

eg: supply chain tracking with RFID tags, or sending environmental data.

Types of Oracles

01 Hardware Oracles

02 Software Oracles

03 Consensus Oracles

04 Inbound Oracles

05 Outbound Oracles

Software Oracles pull data from 3rd party sources, such as web APIs.

eg: flight information, or travel times.

Types of Oracles

01 Hardware Oracles

02 Software Oracles

03 Consensus Oracles

04 Inbound Oracles

05 Outbound Oracles

Consensus Oracles rely on aggregating data from multiple oracles to determine accuracy and authenticity.

eg: Seen in Chainlink's Protocol.

Types of Oracles

01 Hardware Oracles

02 Software Oracles

03 Consensus Oracles

04 Inbound Oracles

05 Outbound Oracles

Inbound Oracles are software oracles that trigger when conditions are met for a data request received.

eg: used for trading, and stock options.

Types of Oracles

01 Hardware Oracles

02 Software Oracles

03 Consensus Oracles

04 Inbound Oracles

05 Outbound Oracles

Outbound Oracles allow smart contracts to send data to sources outside the blockchain network.

eg: used for representation of physical real-world objects on the blockchain.

Source of Information

Centralized

Only have a **single source of information**. Hence, suffers from the “bottle-neck” problem such that a single point of failure exists.

Decentralized

Do not rely on a single source of information and **instead aggregate** information from **multiple data sources**.

03

The Oracle Problem

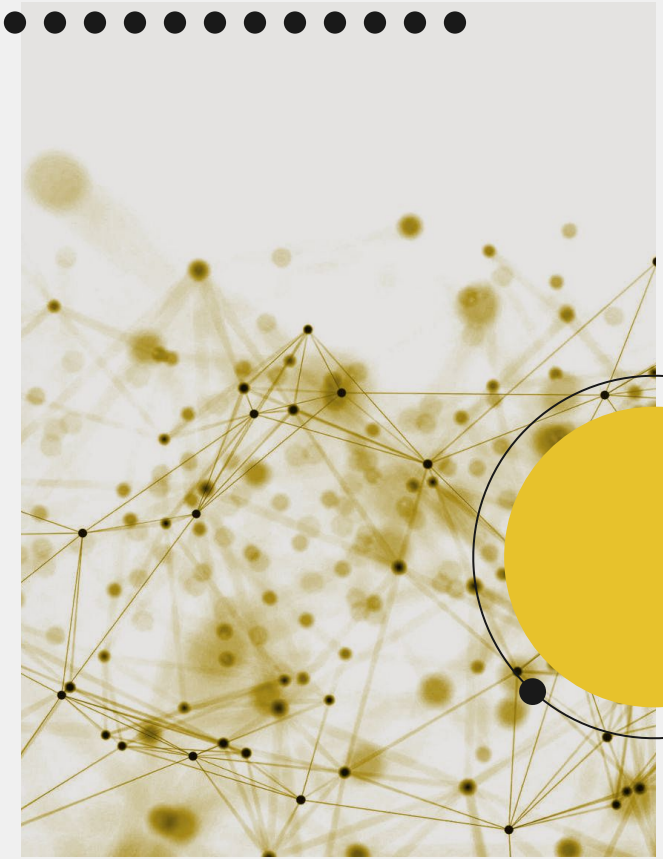
What is the Oracle Problem?

What is the Oracle Problem?

The Oracle Problem is defined as the **security, authenticity,** and **trust conflict** between third-party oracles and the trustless execution of smart contracts.

Oracles retain an enormous amount of **control** and power over smart contracts & how they are executed.

To ensure of this **trustless process**, we need a decentralized network of oracles to push and pull data from the external world.



04

Decentralized Oracles

What exactly are decentralized oracles and how do they work?

Decentralized Oracles

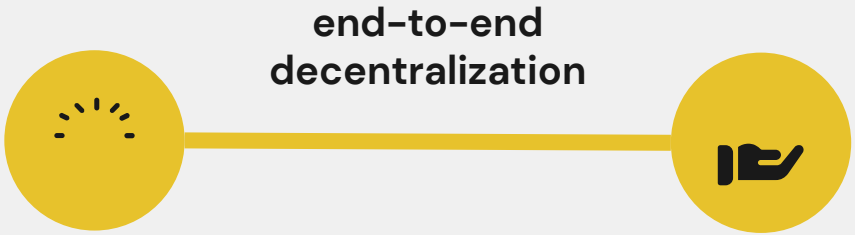
Distributed Data Sources

Decentralized oracles solve the off-chain data problem by granting blockchains access to real-world information by using distributed data sources, avoiding a **single point of vulnerability**.

Trustless & Deterministic

Aim to achieve trustless & deterministic results, and distribute trust amongst many network participants & provide security & fairness to smart contracts.

Decentralized Oracle Network (DON)



Independent Oracle Node Operators

Multiple Reliable Data Sources

Decentralized Oracle Network

Chainlink

05

Chainlink

How does Chainlink achieve their Decentralized Oracle Network?

How does Data get onto Chainlink?

Oracle Node Operators

Stake $LINK tokens to be considered as a trusted source of data, where nodes are rewarded for being reliable, and slashed otherwise.

Requesting Contract

Requesting contract is the beginning and register the request as an event, set up a matching smart contract, a Service Level Agreement (SLA), such that it is able to connect to off-chain data.

Service Level Agreement

01

Reputation Contract

Evaluates the track-record of an oracle and removes or keeps reliable/unreliable node, to ensure for trusted data.

02

Order Matching Contract

Sends the query to trustable nodes and checks their bids, automatically chooses suitable number of nodes & node types to handle the request.

03

Aggregation Contract

Validates data from single or multiple data sources. Reconciles them by taking an average, and their own consensus algorithm.

06

NaiveOracle: Overview

An overview of NaiveOracle: A Decentralized Oracle Network.

NaiveOracle vs Chainlink

💡 Interoperable Oracle Contract

We have an oracle contract which can be user defined an implementation agnostic whereas chain link does not.

Consensus Algorithm

Focus: Financial Industry & Market Data

01

Min/Max Stock Price

Minimum and Maximum stock price over the last 5 years, that updates when called by the requesting contract.

02

Create a Buffer

We take the maximum subtracted by the minimum and divide it by 2.

03

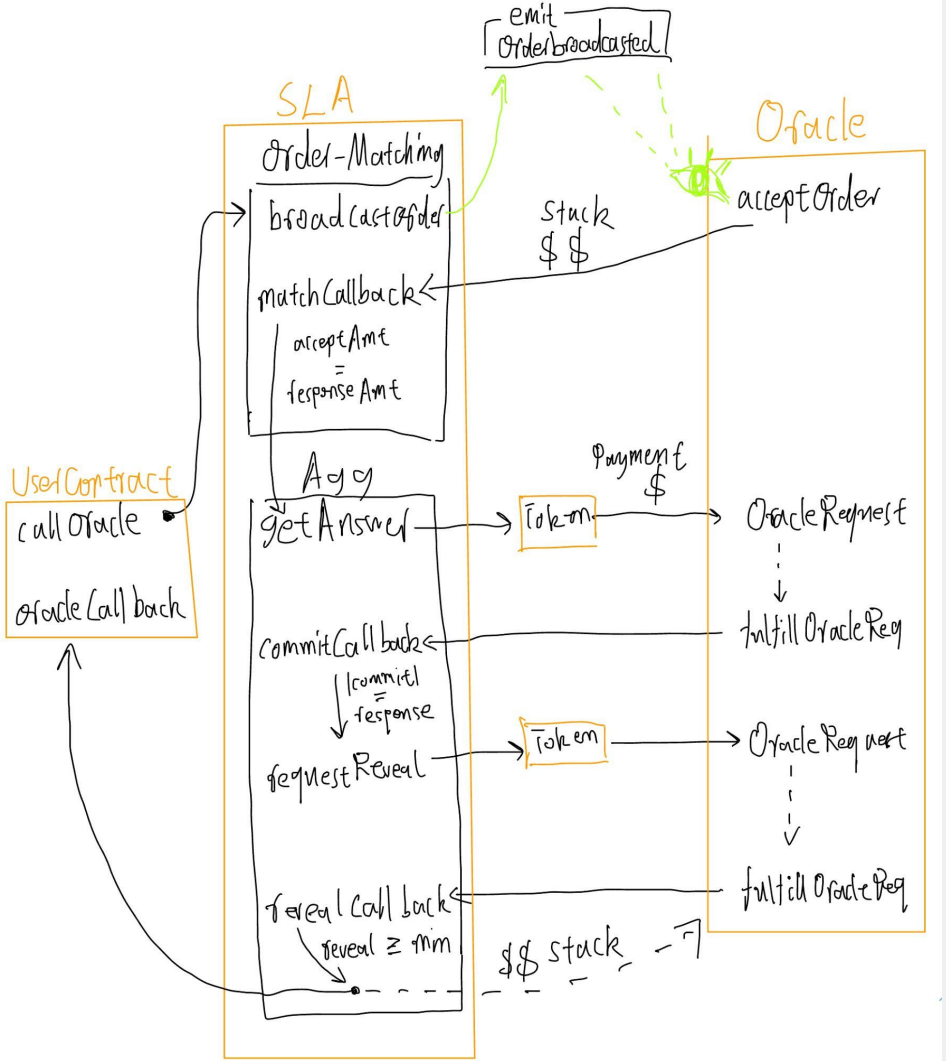
Slash / Reward

We take the median value, and check if it is above the median and buffer or below the median or buffer, and slash accordingly.

07

NaiveOracle:

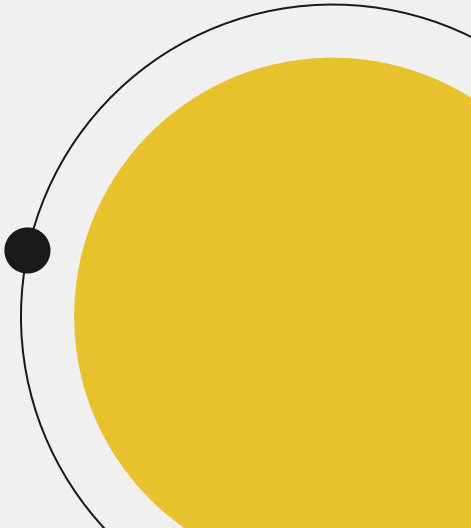
Story & Demo

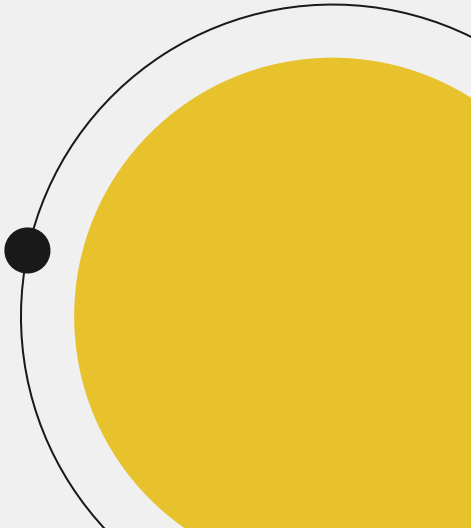



```
await token.connect(account1).mint(user.address, ONE_NAIVE.mul(20))
await token.connect(account1).mint(o1.address, ONE_NAIVE.mul(19))
await token.connect(account1).mint(o2.address, ONE_NAIVE.mul(19))
await token.connect(account1).mint(o3.address, ONE_NAIVE.mul(19))

// order matching
await user.callOracle(3, ONE_NAIVE.mul(12));

// Plan:
// o1, o2 didnt lie, o3 lied
// o1, o2, o3 all accept the order so stake 12 from each of them
// o1, o2, o3 all get 12/3 = 4 token award
// o1, o2 didnt lie so they get 12 token back: 19 - 12 + 4 + 12 = 23
// o3 lied so he get 0 token back: 19 - 12 + 4 = 11
// sla keeps that 12 token slashed from o3 yeey!

let OrderBroadcasted = await sla.queryFilter(sla.filters.OrderBroadcasted());
let orderArgs = OrderBroadcasted[0].args;
await o1.acceptOrder(orderArgs.requestId, orderArgs.callbackAddress, orderArgs.callbackFunctionId, orderArgs.payment);
await o2.acceptOrder(orderArgs.requestId, orderArgs.callbackAddress, orderArgs.callbackFunctionId, orderArgs.payment);
await o3.acceptOrder(orderArgs.requestId, orderArgs.callbackAddress, orderArgs.callbackFunctionId, orderArgs.payment);

const requestId = orderArgs.requestId;

// Oracle node recieves request and perpare commit hash
let requestRecievedO1C = await o1.queryFilter(o1.filters.RequestRecieved());
let requestRecievedO2C = await o2.queryFilter(o2.filters.RequestRecieved());
let requestRecievedO3C = await o3.queryFilter(o3.filters.RequestRecieved());
// let requestRecievedO2C = await o2.queryFilter(o2.filters.RequestRecieved());
let args01C = requestRecievedO1C[0].args;
let args02C = requestRecievedO2C[0].args;
let args03C = requestRecievedO3C[0].args;
await o1.commitOracleRequest(args01C._requestId, args01C._callbackAddress, args01C._callbackFunctionId, hash1)
await o2.commitOracleRequest(args02C._requestId, args02C._callbackAddress, args02C._callbackFunctionId, hash2)
await o3.commitOracleRequest(args03C._requestId, args03C._callbackAddress, args03C._callbackFunctionId, hash3)
// Oracle node recieves request and perpare reveal data
let requestRecievedO1R = await o1.queryFilter(o1.filters.RequestReveal());
let requestRecievedO2R = await o2.queryFilter(o2.filters.RequestReveal());
let requestRecievedO3R = await o3.queryFilter(o3.filters.RequestReveal());
let args01R = requestRecievedO1R[0].args;
let args02R = requestRecievedO2R[0].args;
let args03R = requestRecievedO3R[0].args;
// Oracles contract send the reveal data to Aggregator
await expect(o1.revealOracleRequest(args01R._requestId, args01R._callbackAddress, args01R._callbackFunctionId, data1, salt1))
  .to.emit(sla, "ResponseReceived");
await expect(o2.revealOracleRequest(args02R._requestId, args02R._callbackAddress, args02R._callbackFunctionId, data2, salt2))
  .to.emit(sla, "ResponseReceived");
await expect(o3.revealOracleRequest(args03R._requestId, args03R._callbackAddress, args03R._callbackFunctionId, data3, salt3))
  .to.emit(sla, "ResponseReceived")
  .to.emit(sla, "Answered");

// User contract check the response
const responseAns = ethers.BigNumber.from("1200");
expect(await user.getResponse()).to.equal(responseAns);

// check o3 in answers[requestId].slashOracles
const slashOracles = await sla.getSlashOracles(requestId);
expect(slashOracles[0]).to.equal(o3.address);

expect(await token.balanceOf(o1.address)).to.equal(ONE_NAIVE.mul(23)); // 19 - 12 + 4 + 12
expect(await token.balanceOf(o2.address)).to.equal(ONE_NAIVE.mul(23)); // 19 - 12 + 4 + 12
expect(await token.balanceOf(o3.address)).to.equal(ONE_NAIVE.mul(11)); // 19 - 12 + 4 slashed

expect(await token.balanceOf(sla.address)).to.equal(ONE_NAIVE.mul(12)); // slashed from o3

expect(await token.balanceOf(user.address)).to.equal(ONE_NAIVE.mul(8)); // 20 - 12 for order
```

Thanks!

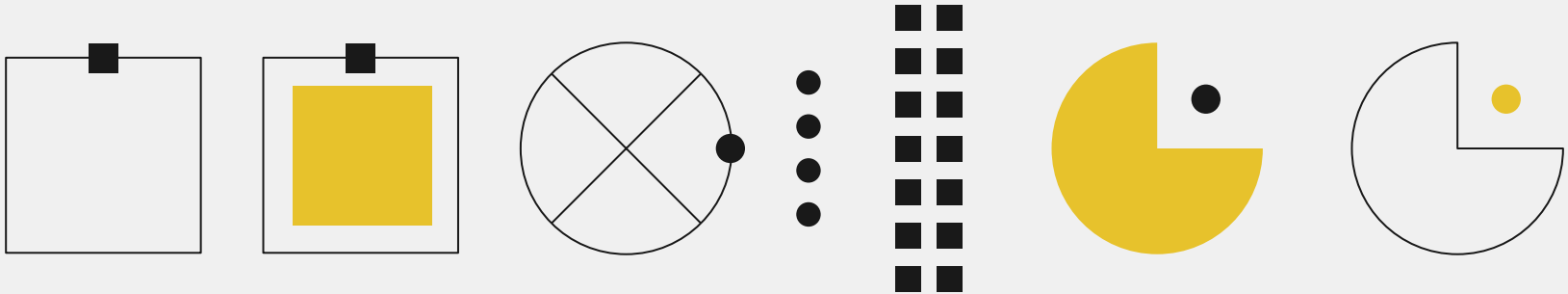
Any Questions?

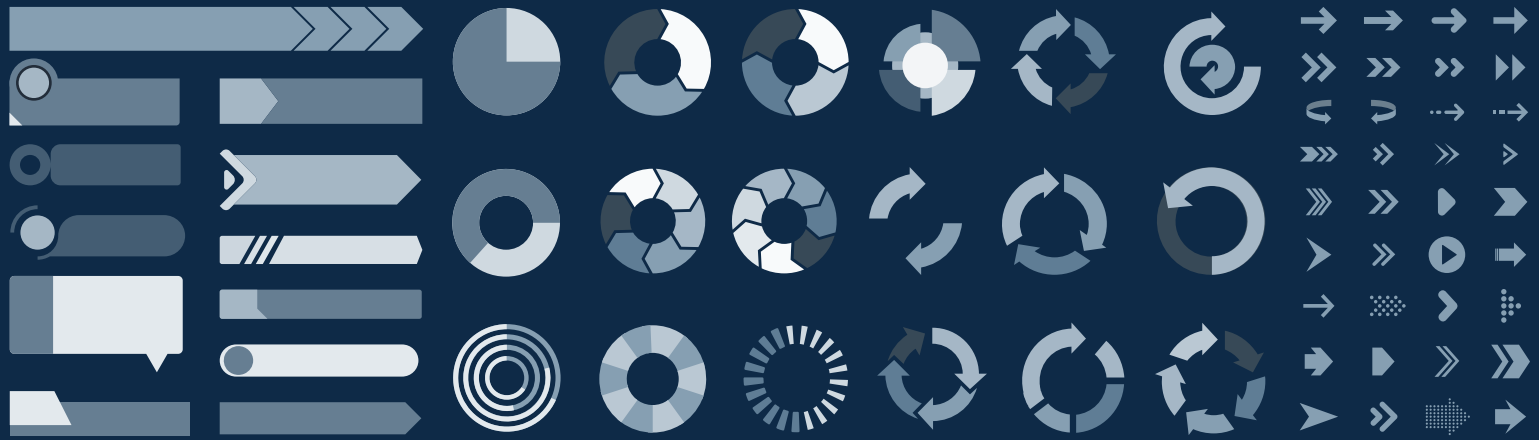




Alternative resources

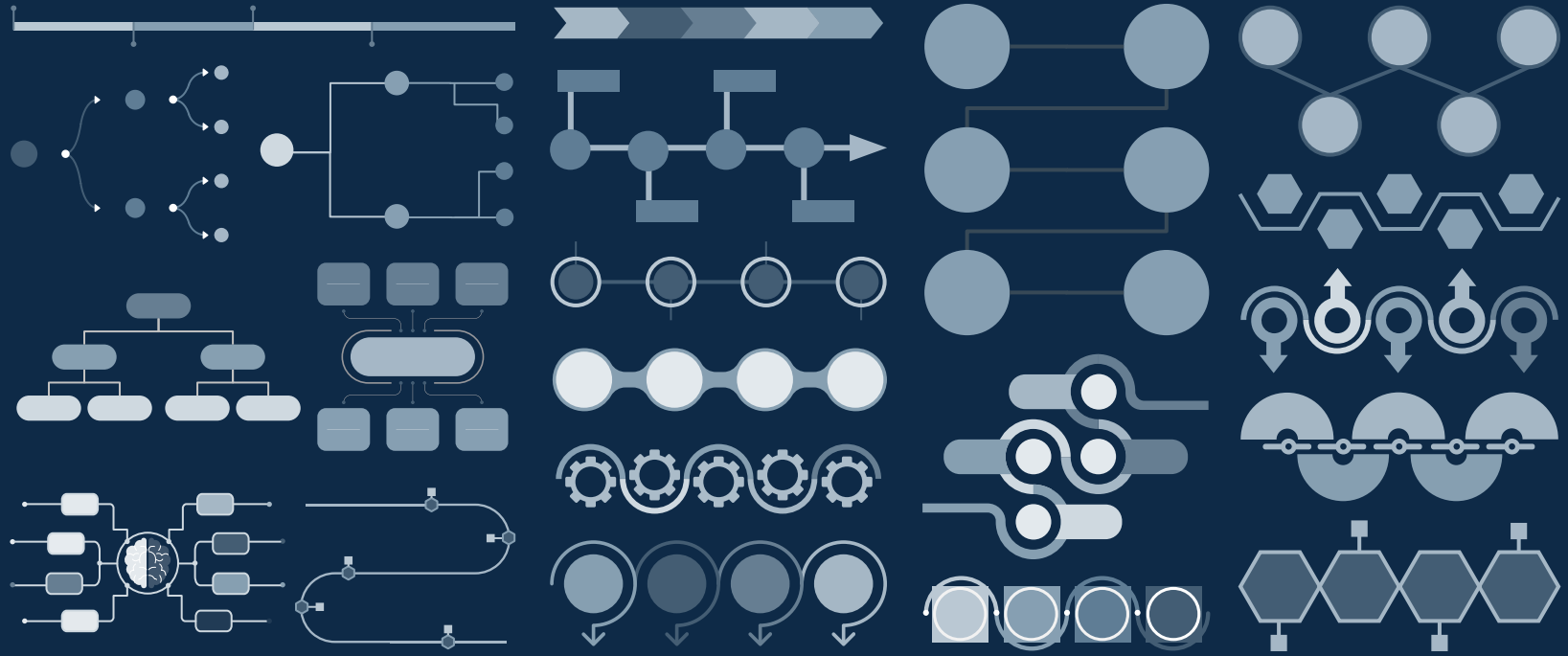
Here’s an assortment of alternative resources whose style fits that of this template:

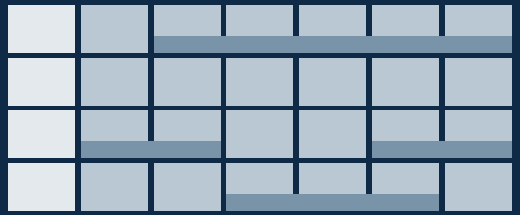


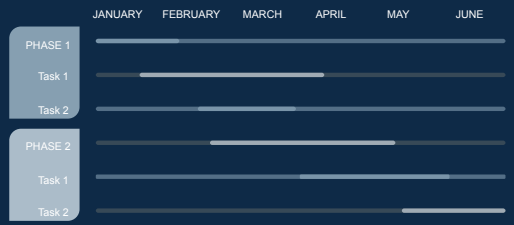


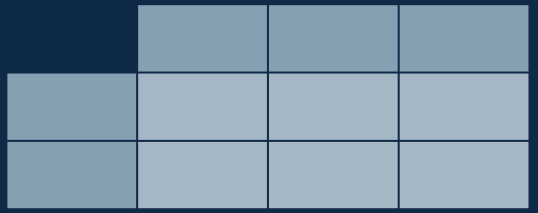


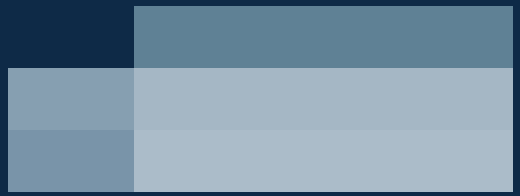


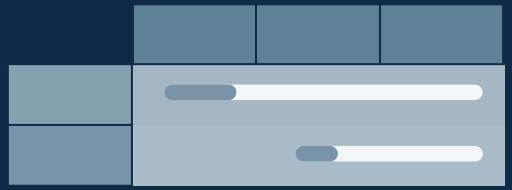



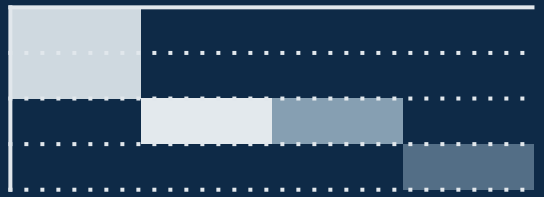


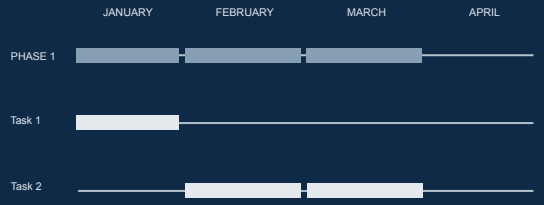

JANUARY
FEBRUARY
MARCH
APRIL
MAY
JUNE
PHASE 1
Task 1
Task 2
PHASE 2
Task 1
Task 2

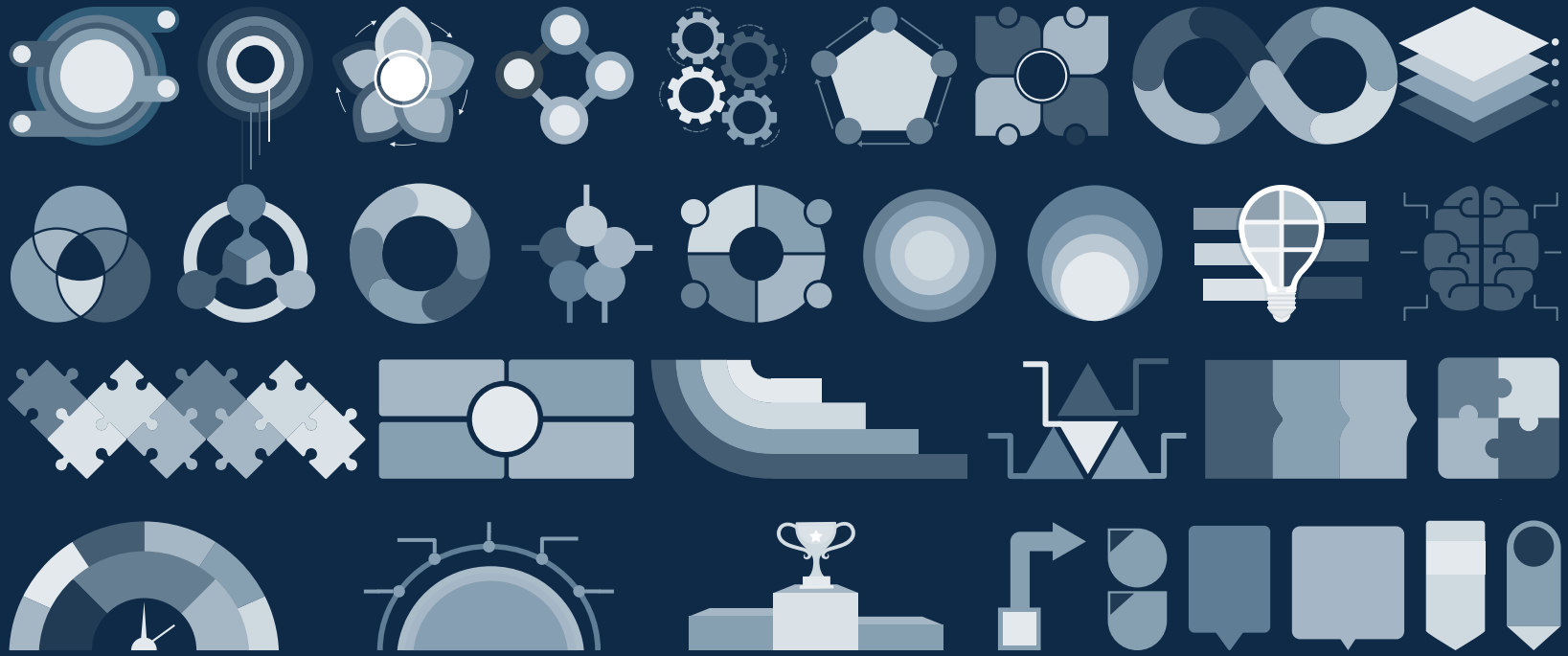









JANUARY
FEBRUARY
MARCH
APRIL
PHASE 1
Task 1
Task 2



Educational Icons

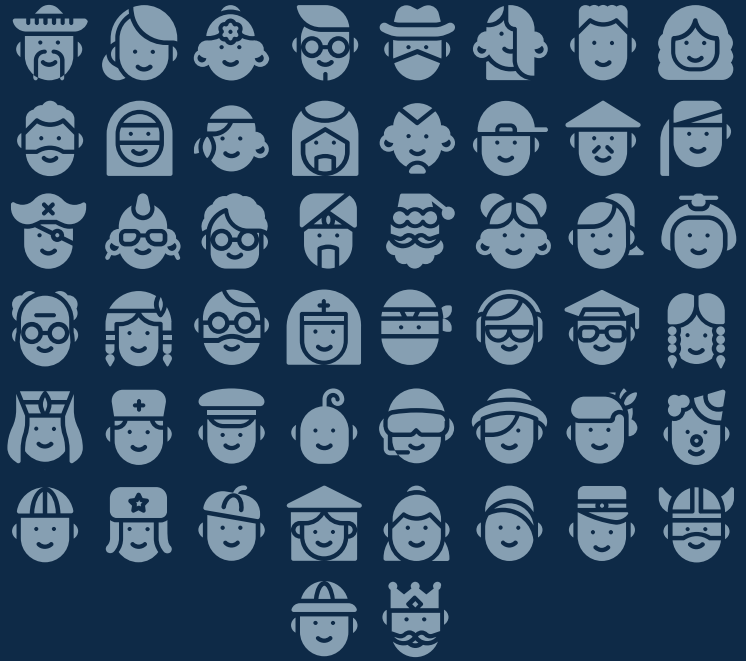
Medical Icons

Business Icons

Teamwork Icons

Help & Support Icons

Avatar Icons



Creative Process Icons

Performing Arts Icons

Nature Icons

SEO & Marketing Icons