Яндекс

/ HolyJS 2026

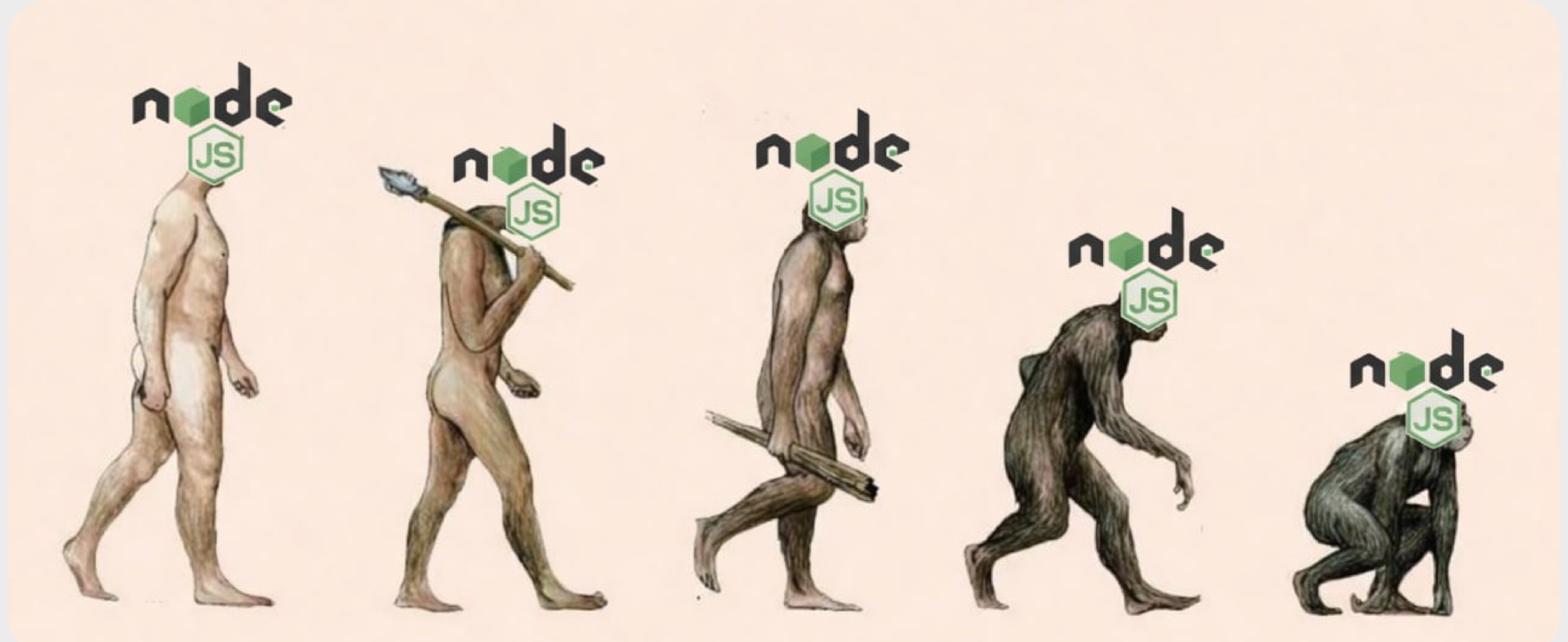
Degradation вместо Downtime

Node.js под пиковыми нагрузками и DDoS

Глеб Решетнёв

Старший разработчик,
Яндекс карты

Degradation вместо downtime



Graceful Degradation

Набор принципов и практик,
которые позволяют нашей системе
пережить пиковые нагрузки
и минимизировать её простой.

Способность системы сохранять часть функциональности
при сбоях и перегрузках — вместо полного отказа.

Кто я?

Глеб
Решетнёв

Старший разработчик,
Яндекс карты

- Разрабатываю веб-карты
- Пишу бэкэнды на Node.js

Уютный Дом

Три уровня защиты

01 Rate Limiter

02 Dynamic Config

03 AbortController

Три уровня защиты

01 Rate Limiter

02 Dynamic Config

03 AbortController

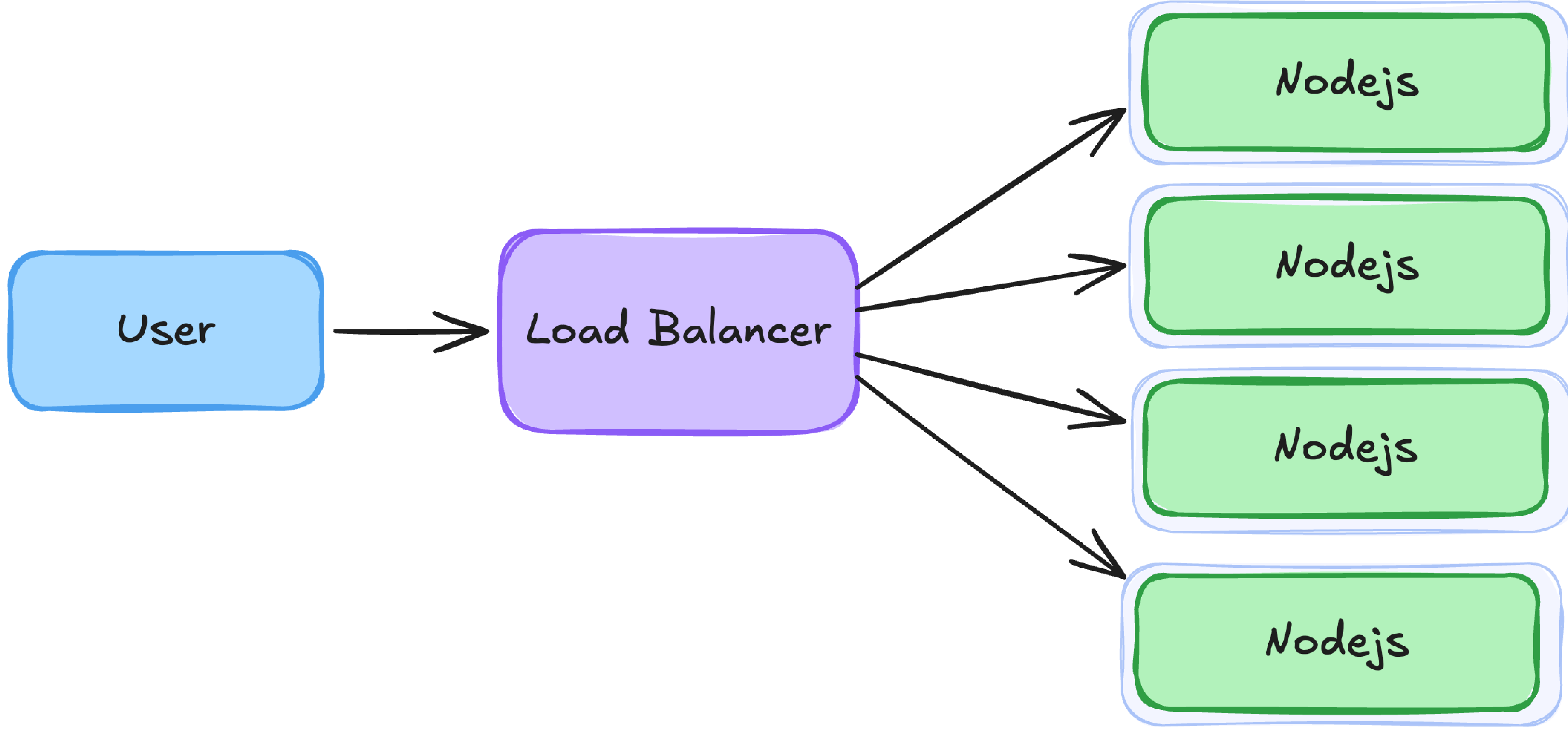
Три уровня защиты

01 Rate Limiter

02 Dynamic Config

03 AbortController

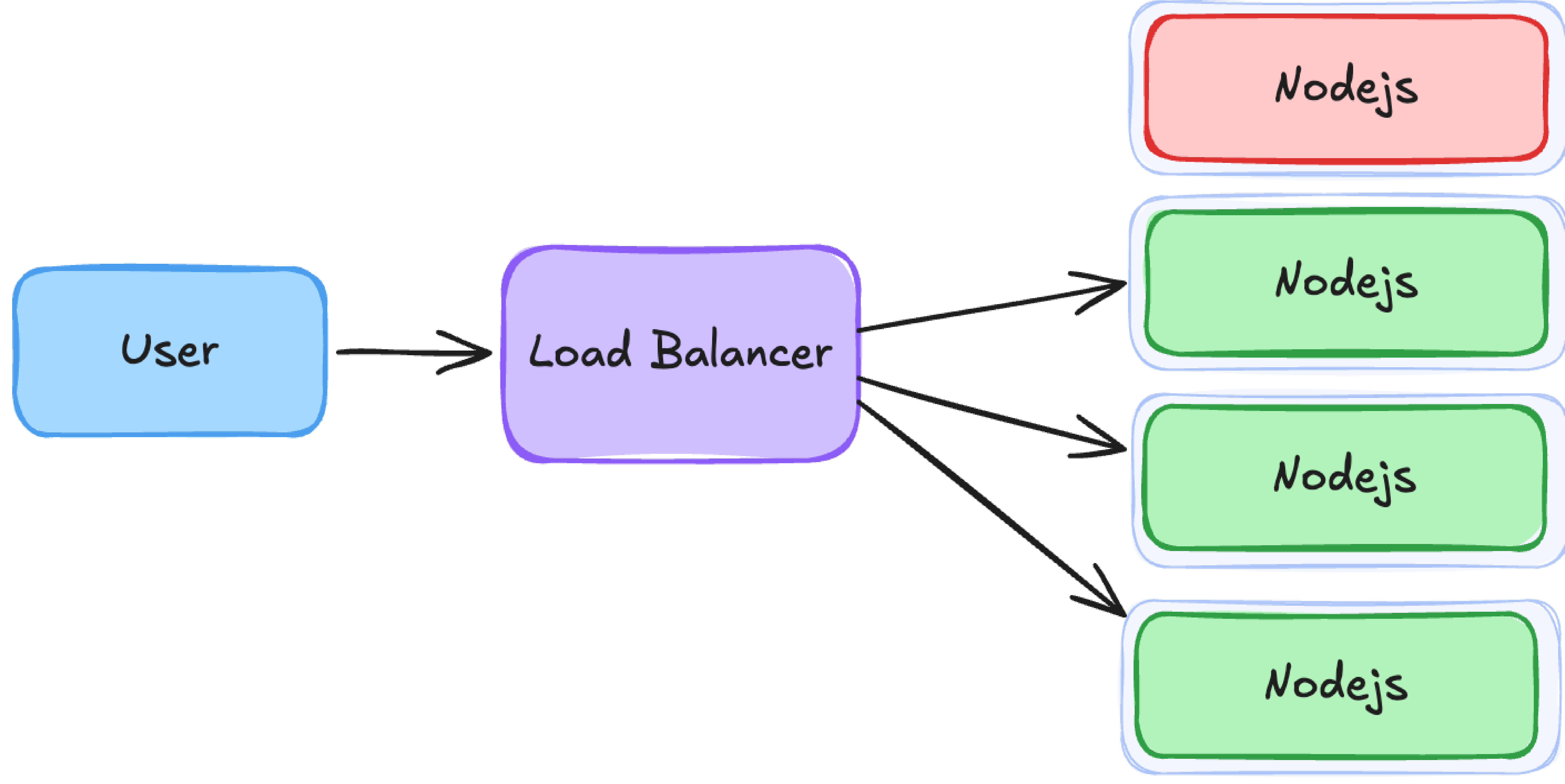
Сервис на Node.js



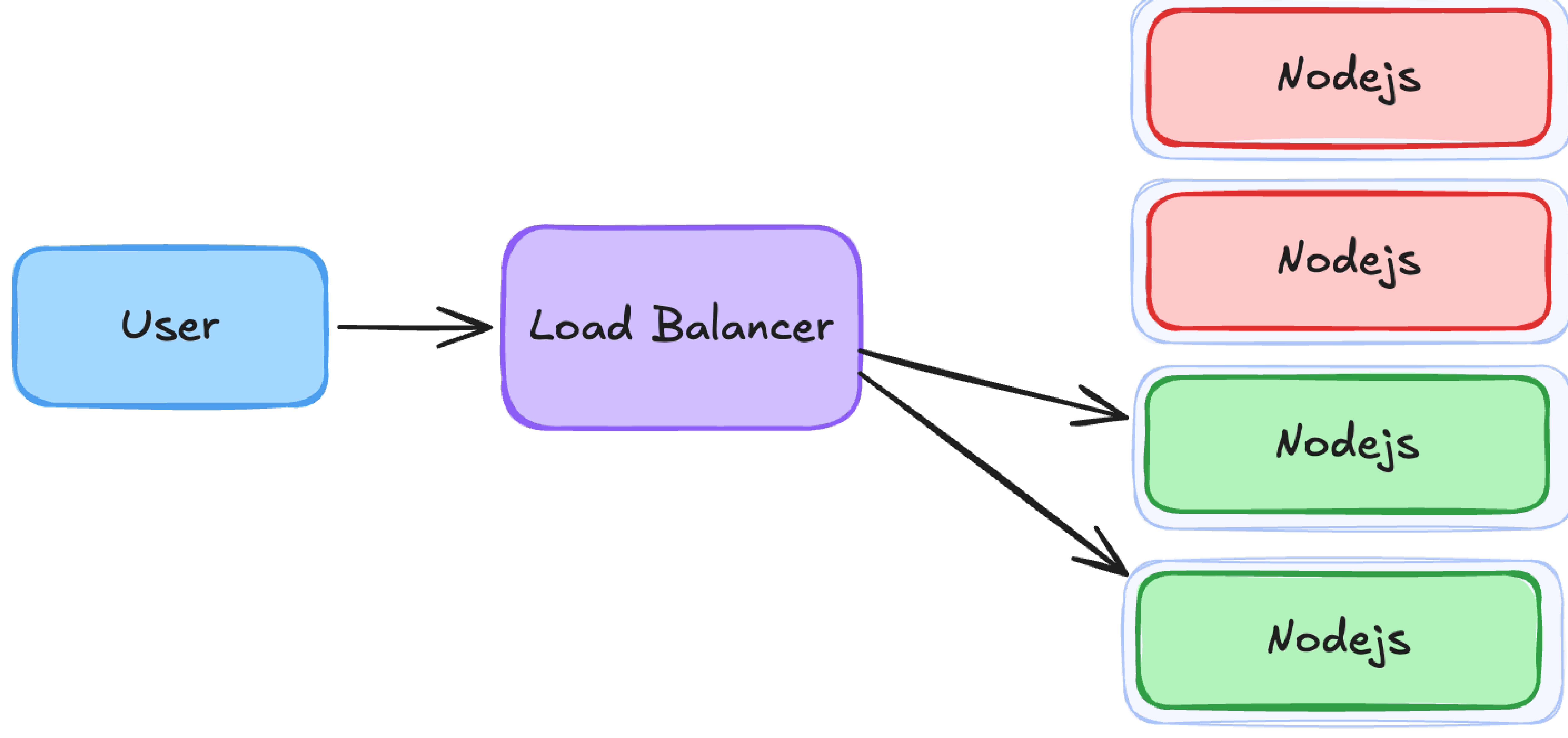
X10 трафика

Node.js выдержит или упадёт?

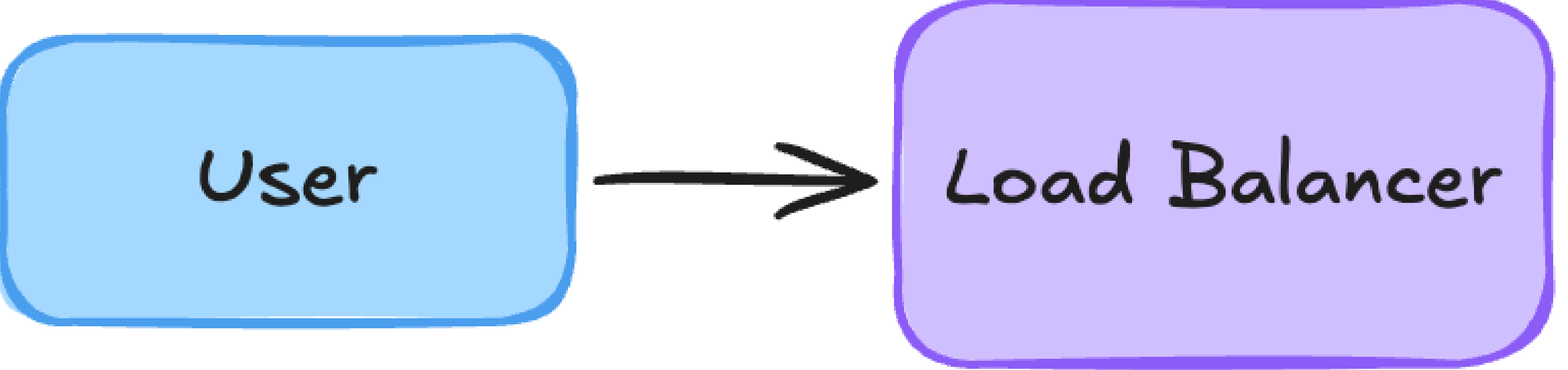
Падает первый инстанс

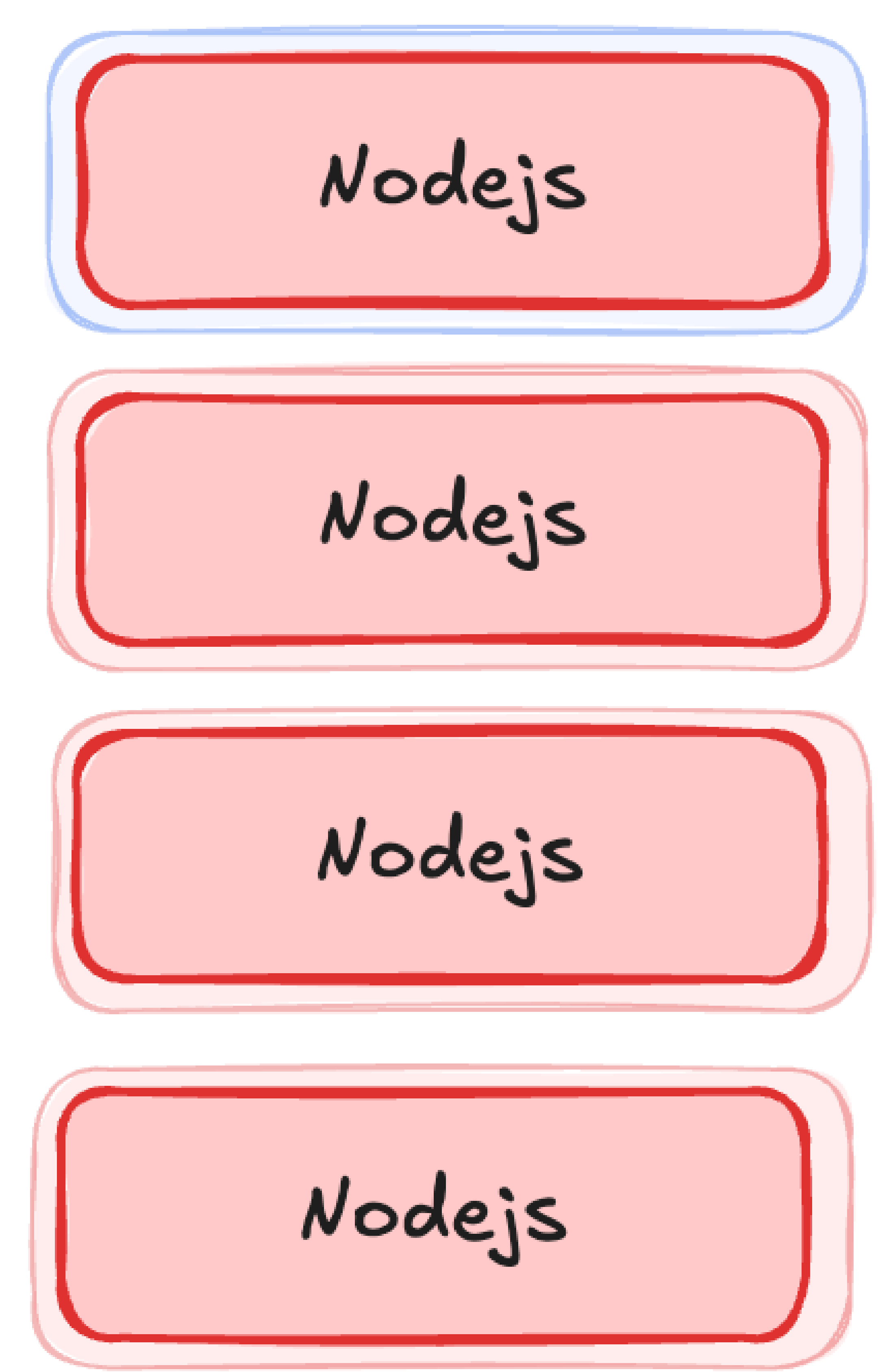


Падает следующий



Каскадный отказ





Что делать?

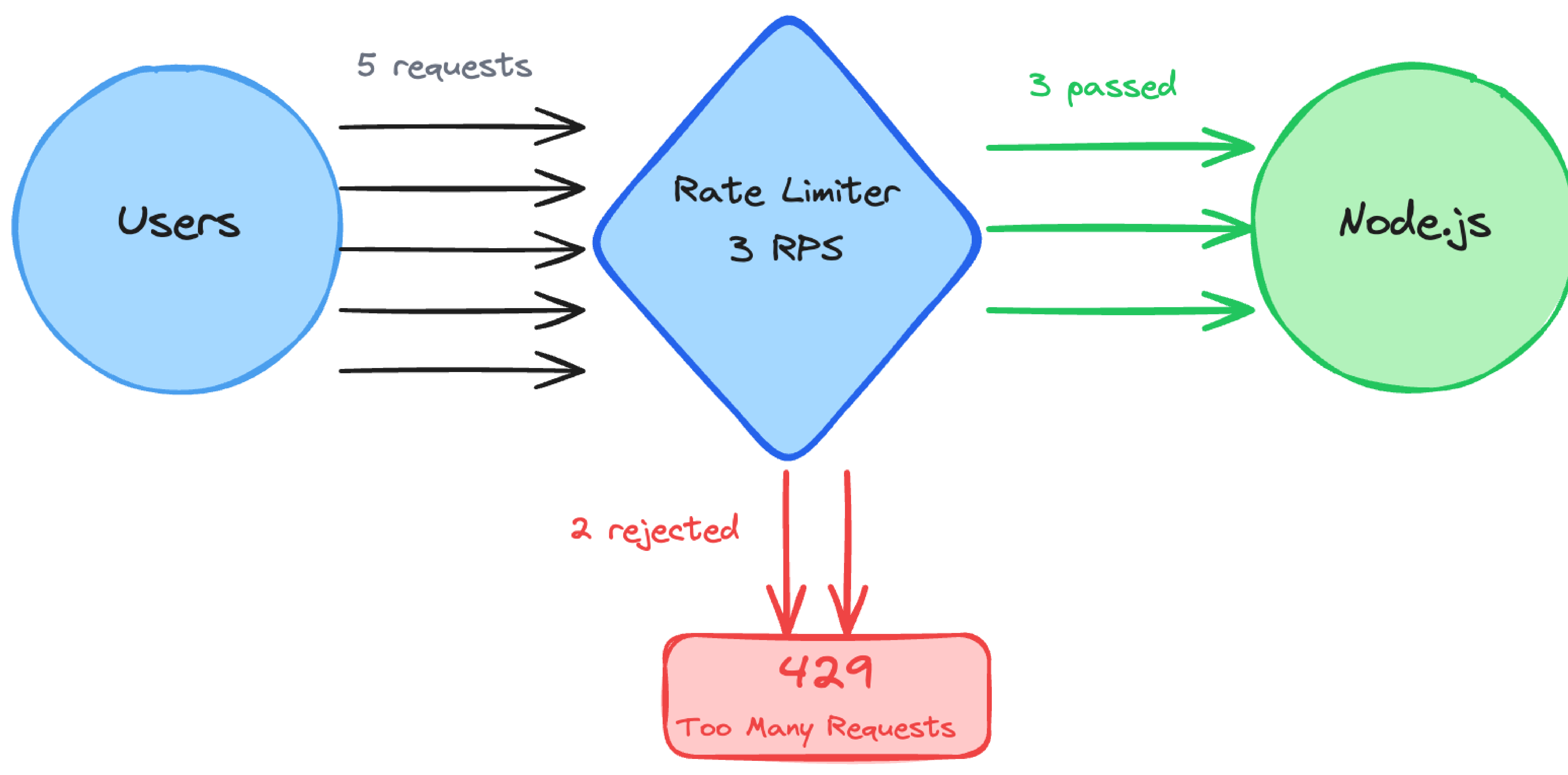
Трафика может прийти так много, что Node.js снова упадёт.

01 Добавить инстансов?

02 Сделать инстансы мощнее — CPU, память?

ВЫХОД ЕСТЬ
ВСЕГДА

Rate Limiter



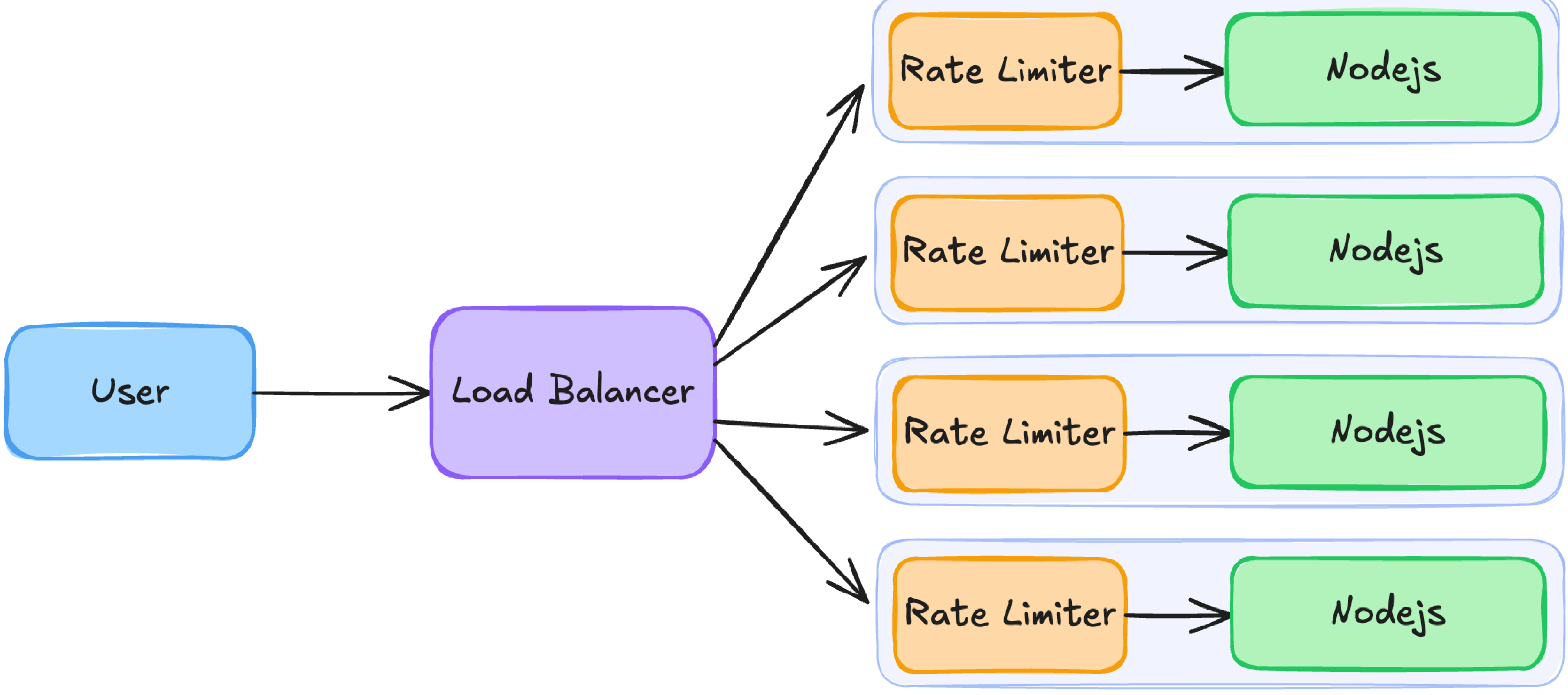
Local vs Global

Local Rate Limiter

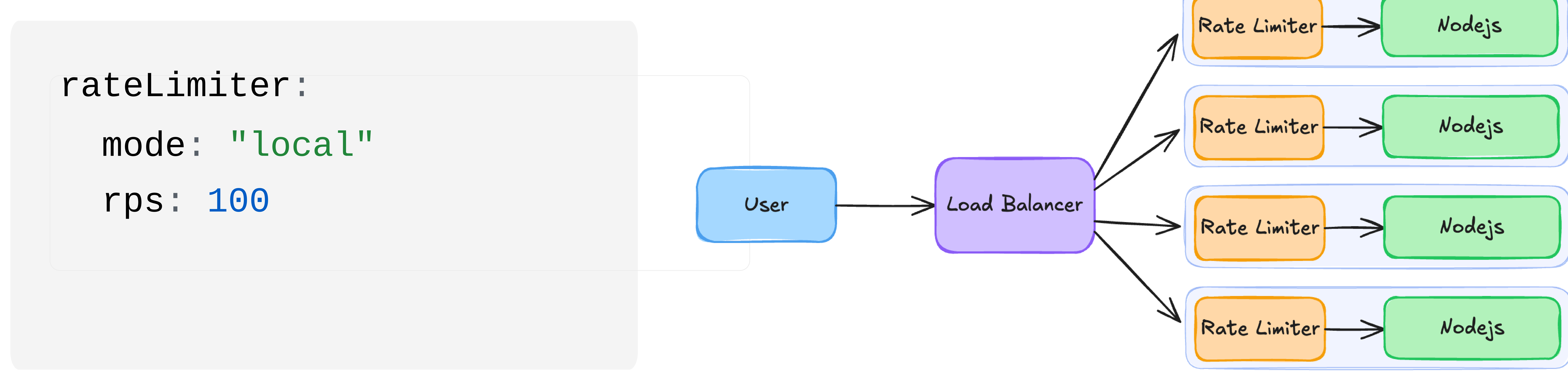
VS

Global Rate Limiter Service

Local Rate Limiter

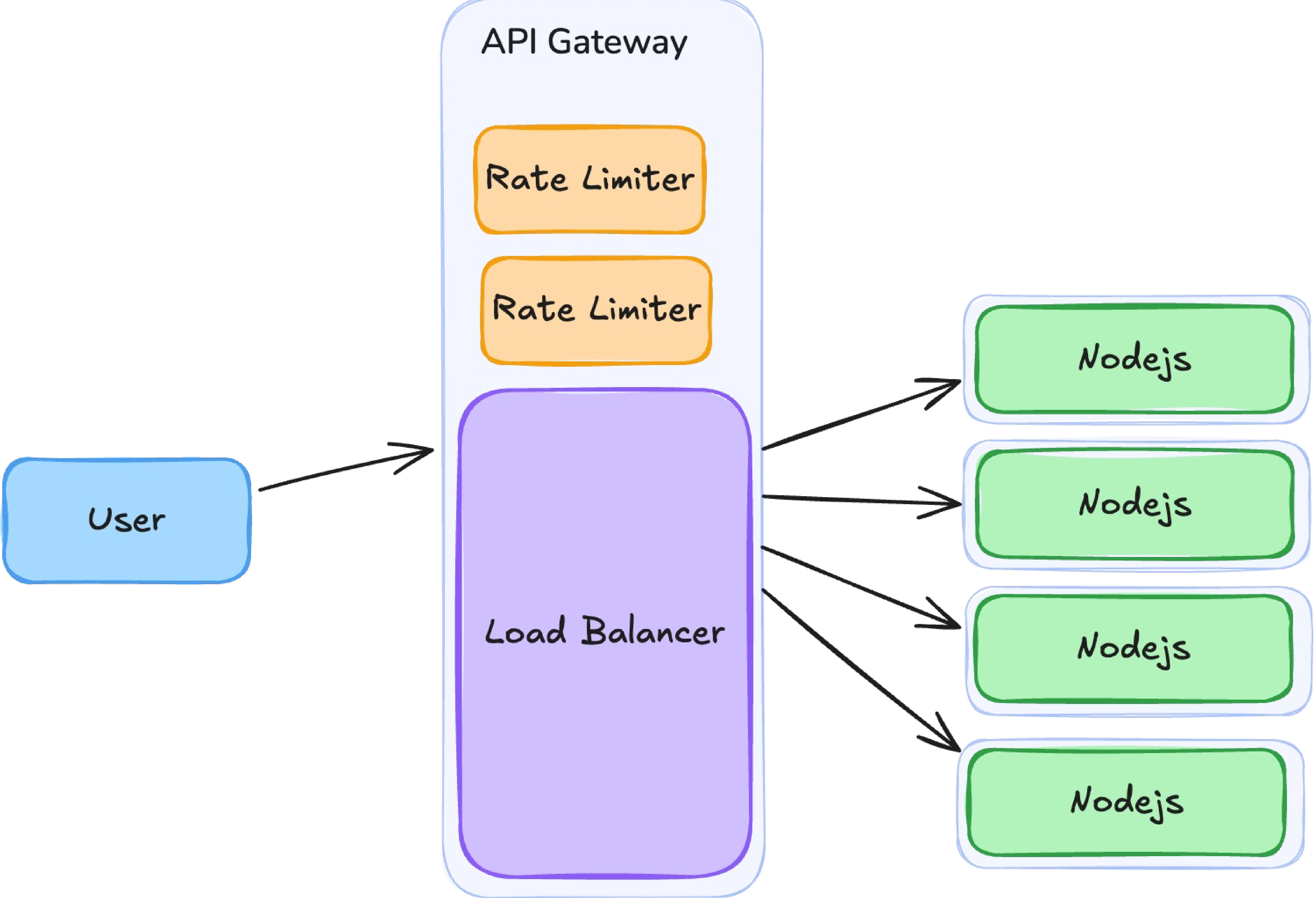


Local Rate Limiter



Каждый инстанс **100 rps** → на сервис целиком **400 rps**.

Local Rate Limiter



Local Rate Limiter

```yaml
rateLimiter:
  mode: "local"
  rps: 200
```

API Gateway
Rate Limiter
Rate Limiter
User
Load Balancer
Nodejs
Nodejs
Nodejs
Nodejs

Каждый инстанс **200 rps** → на сервис целиком **400 rps**.

Плюсы

1. Быстрый
2. Надёжный
3. Простой

Плюсы

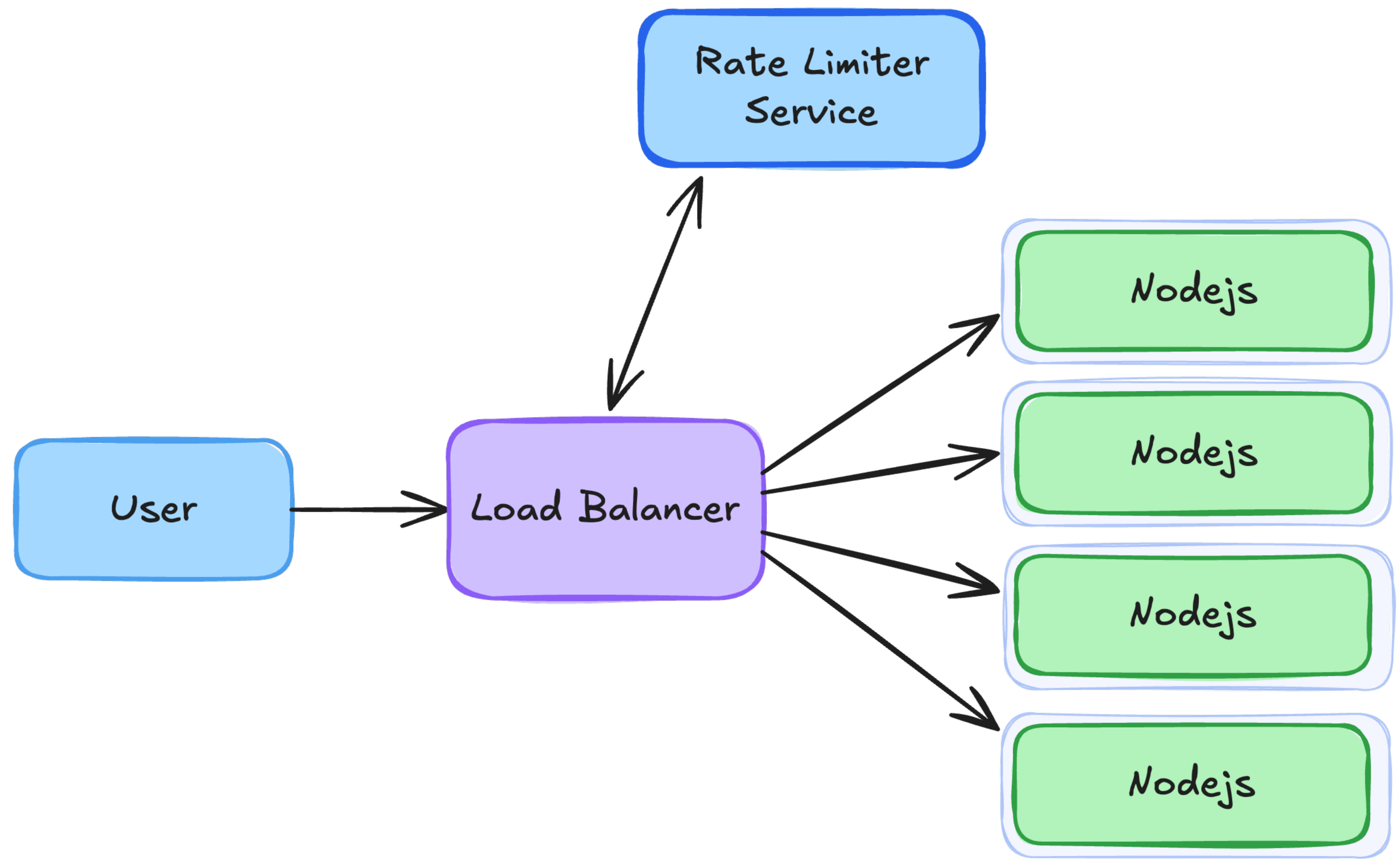
1. Быстрый
2. Надёжный
3. Простой

Минусы

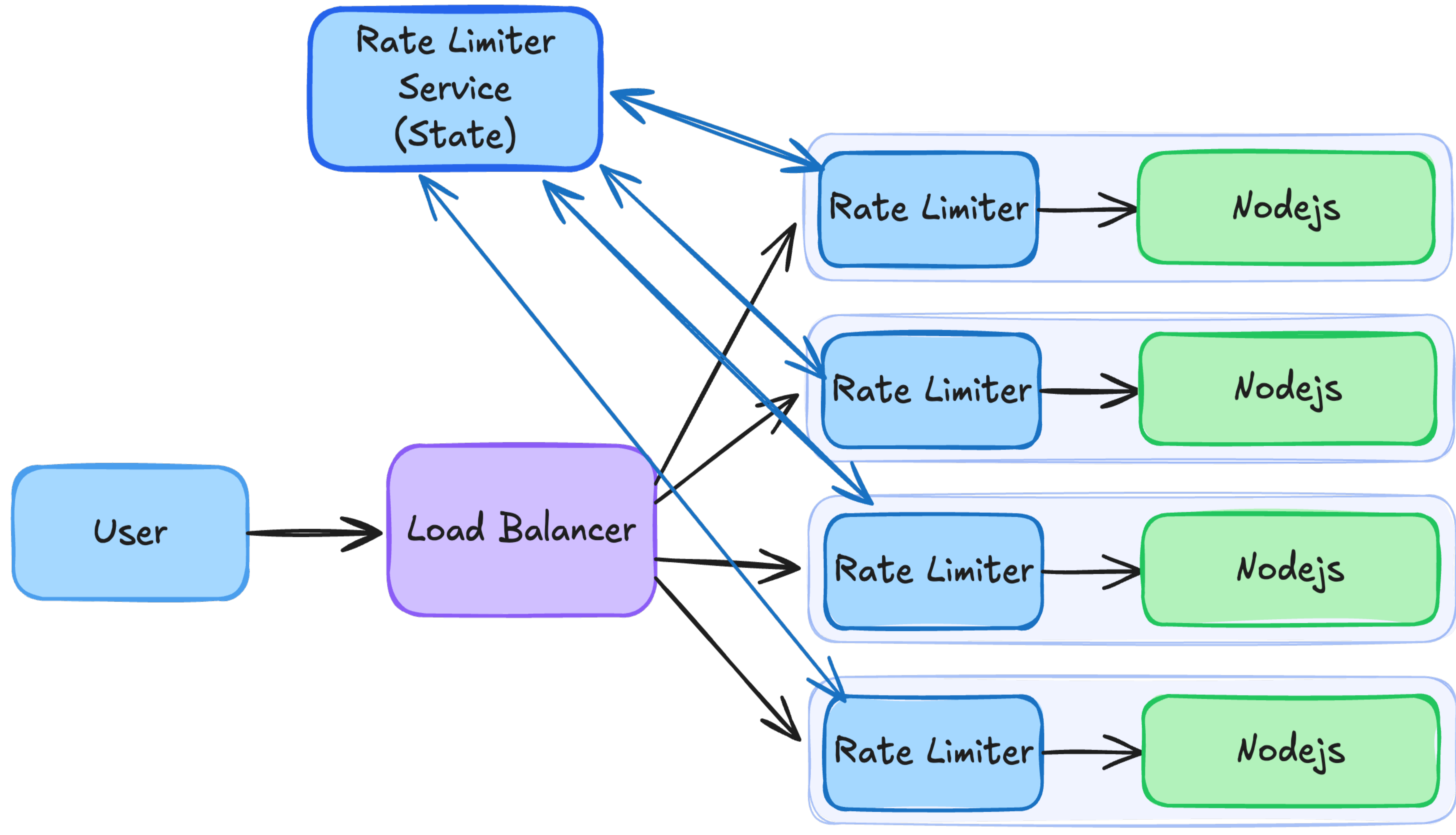
1. Не видим всю нагрузку на кластер
2. Точность зависит от балансировки трафика
3. Расчёт зависит от количества инстансов Rate Limiter

NYPD
Local Rate Limiter
Global Rate Limiter

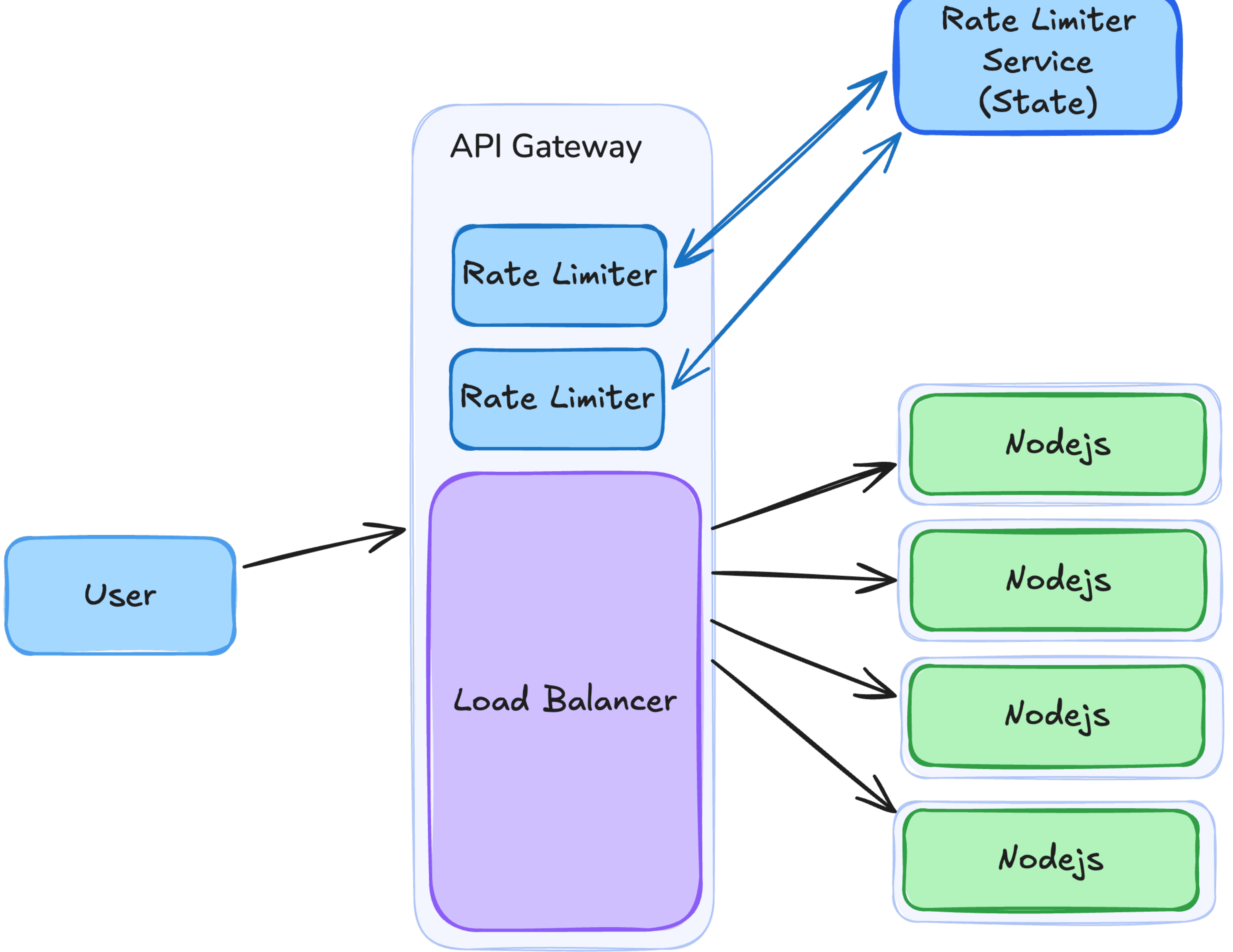
Global Rate Limiter



Global Rate Limiter

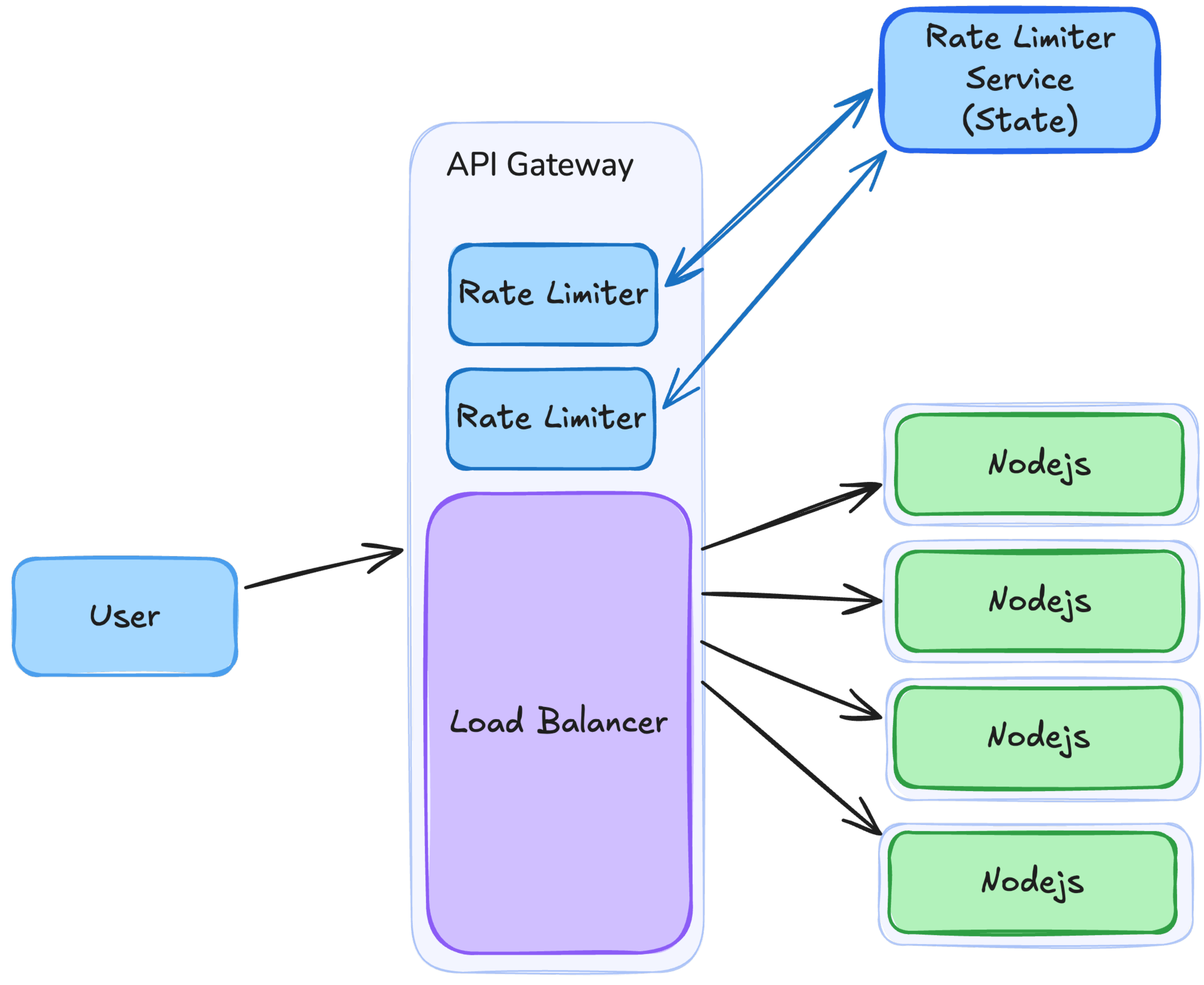


Global Rate Limiter



Global Rate Limiter

```yaml
rateLimiter:
  mode: "global"
  limit:
    rps: 400
  strategy: "redis"
  # не ждём ответ вечно
  timeout: "10ms"
  # fail-open | fail-closed
  # если хранилище недоступно
  failureMode: "fail-open"
```



Плюсы

1. Видим всю нагрузку на кластер
2. Лимиты точные, не зависят от балансировки

Плюсы

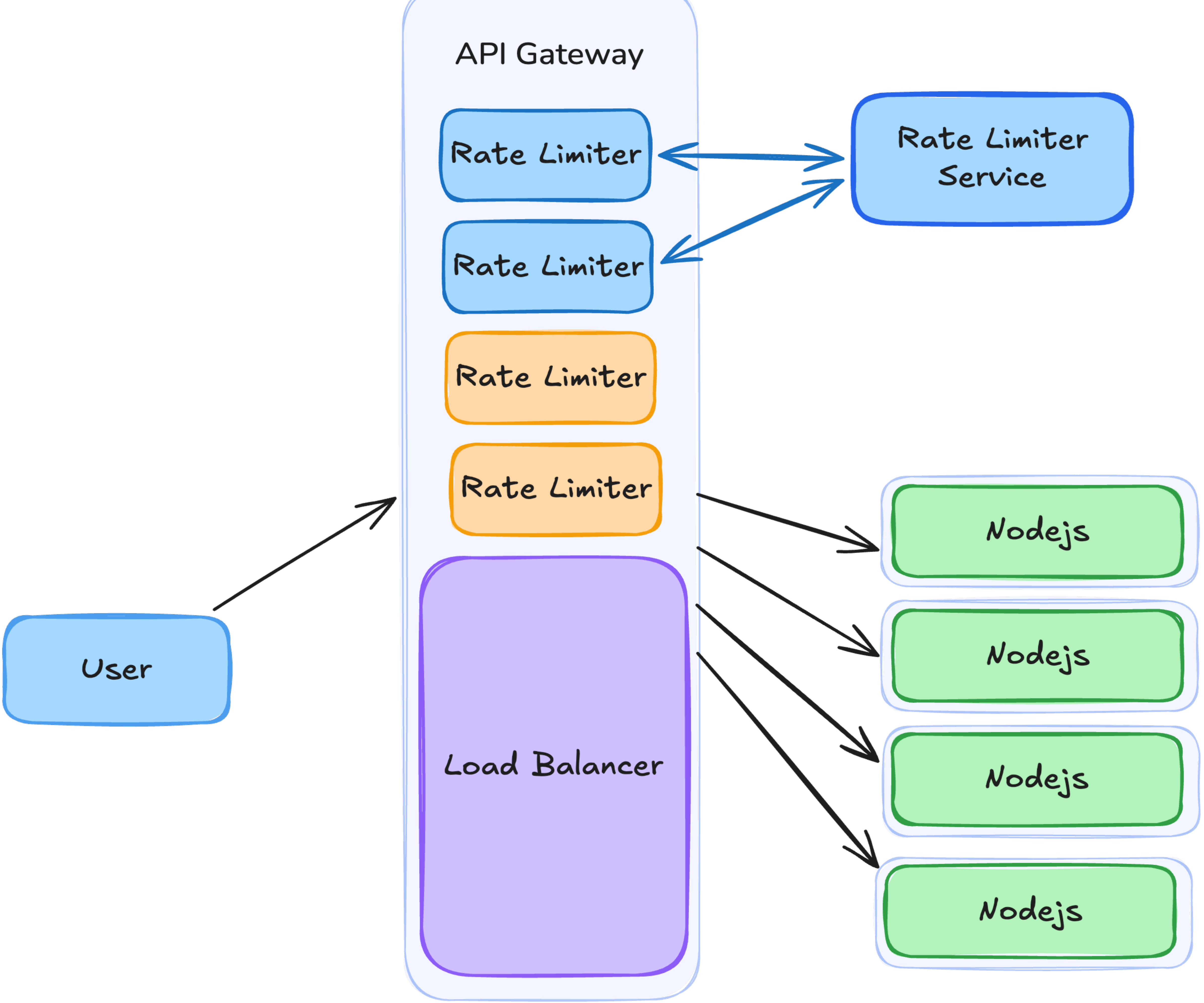
1. Видим всю нагрузку на кластер
2. Лимиты точные, не зависят от балансировки

Минусы

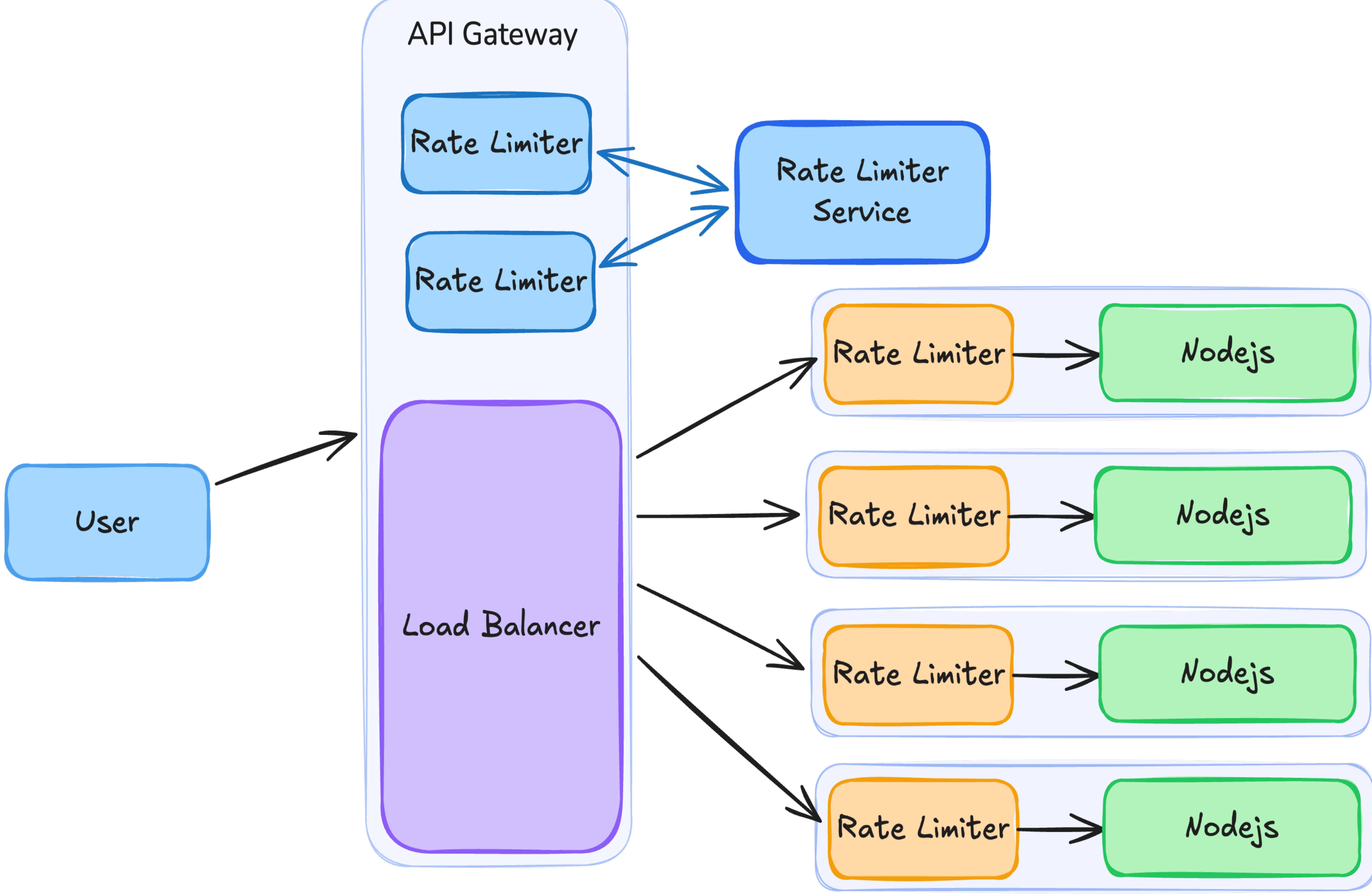
1. Не такой быстрый
2. Менее надёжный
3. +1 компонент к инфраструктуре

Что выбрать?

Global + Local Rate Limiter



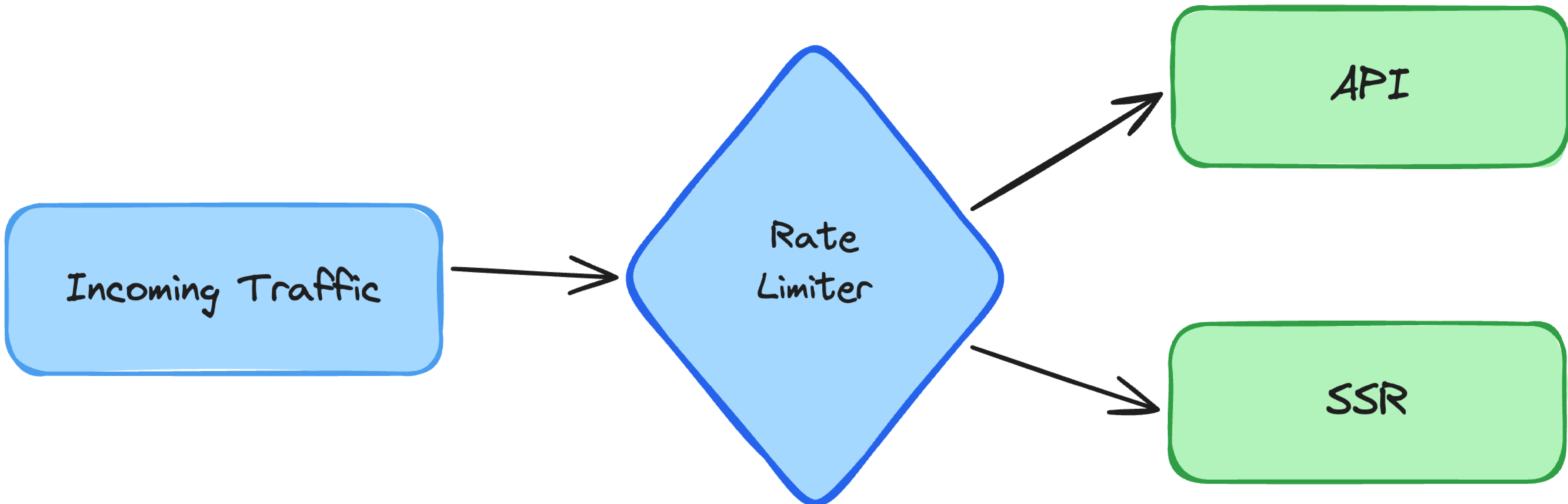
Global + Local Rate Limiter



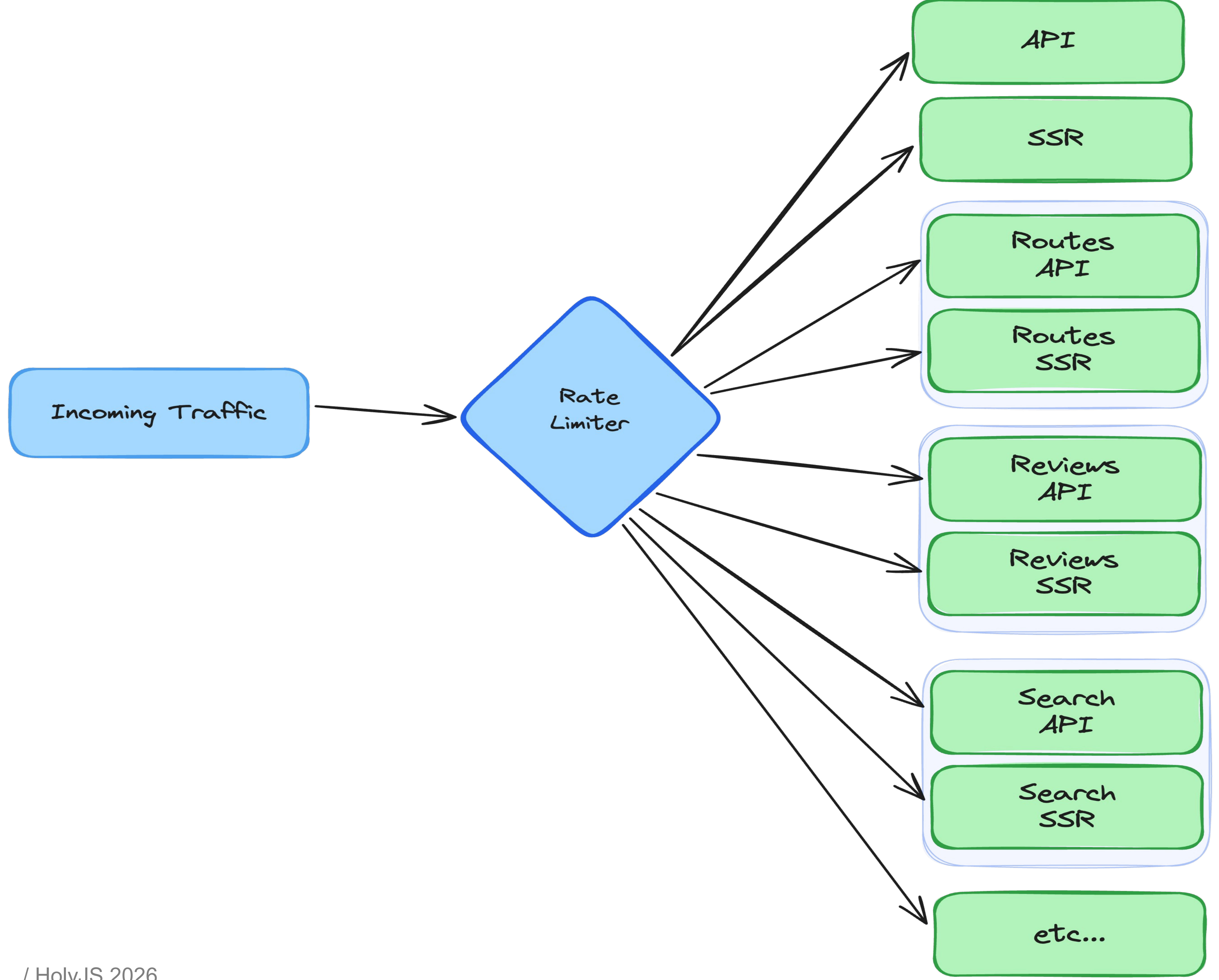
Как-то всё в куче….

Так разделяй и властвуй!

Группа: API + SSR

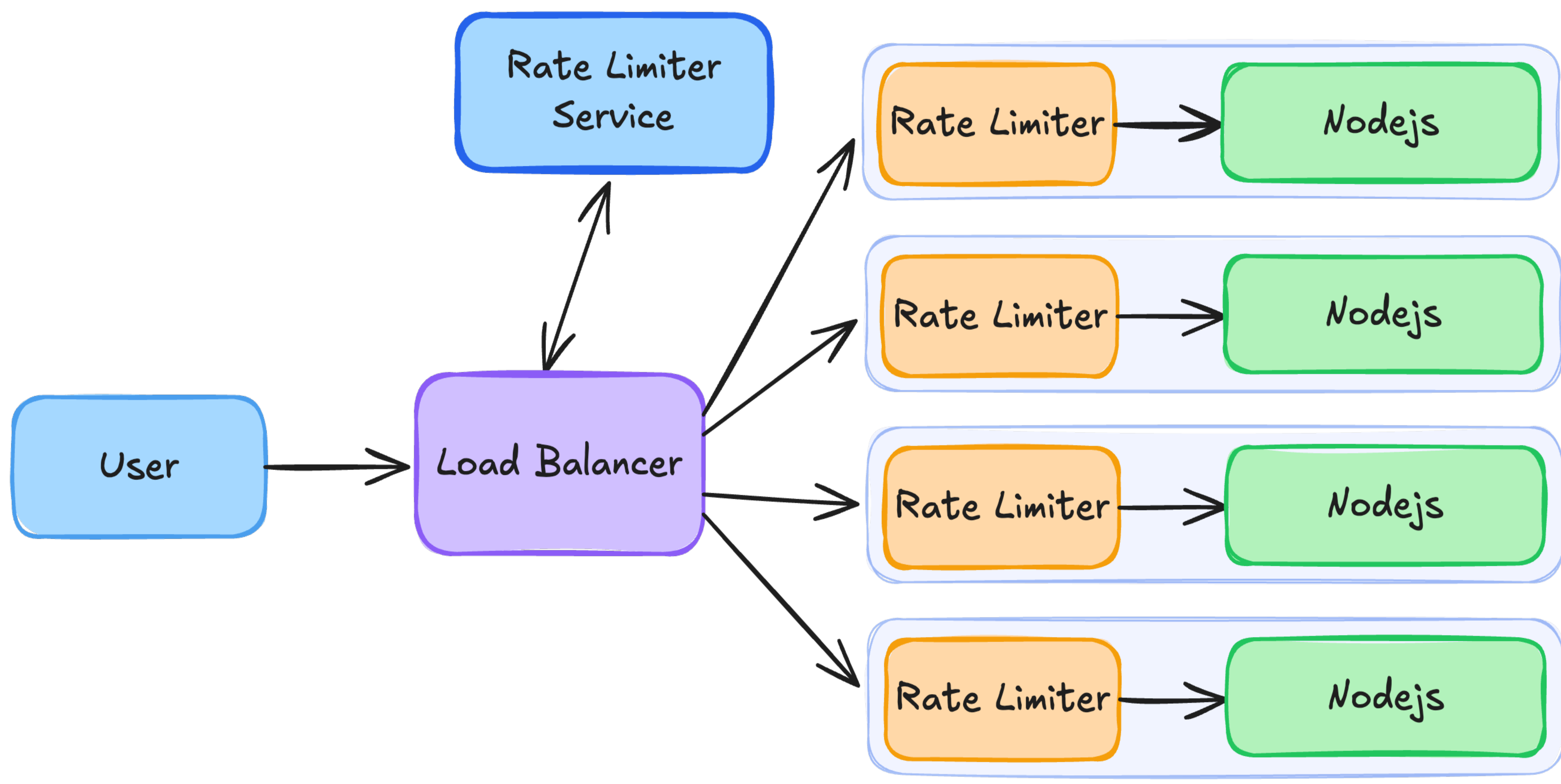


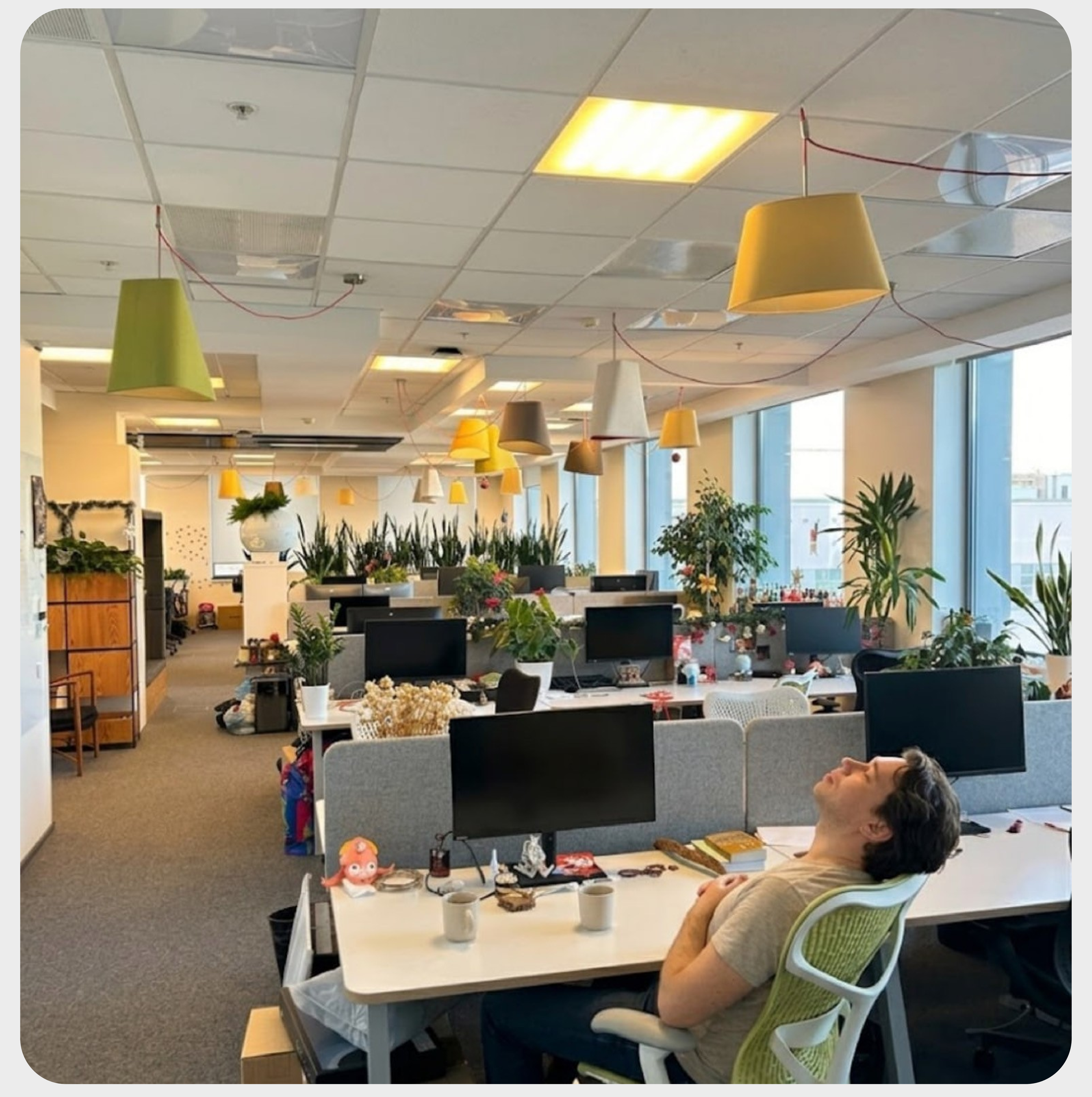
Группы по функциональности



node
JS

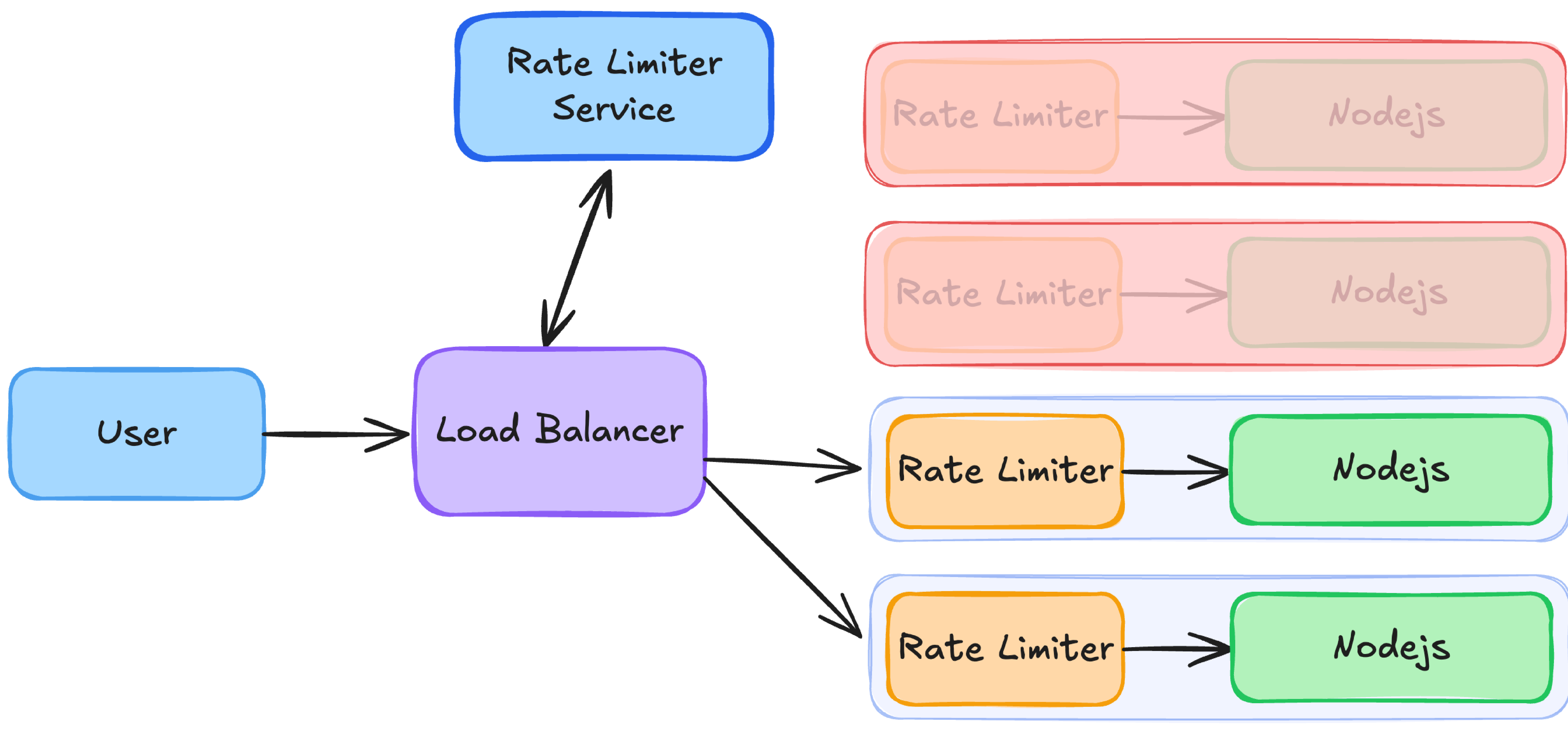
Теперь сервис переживает всплески трафика





ALERT!

Часть инстансов недоступна



Что делать?

01 Rate Limiter

02 Больше инстансов

03 Выключить ненужное

Что делать?

01 Rate Limiter

02 Больше инстансов

03 Выключить ненужное

Что делать?

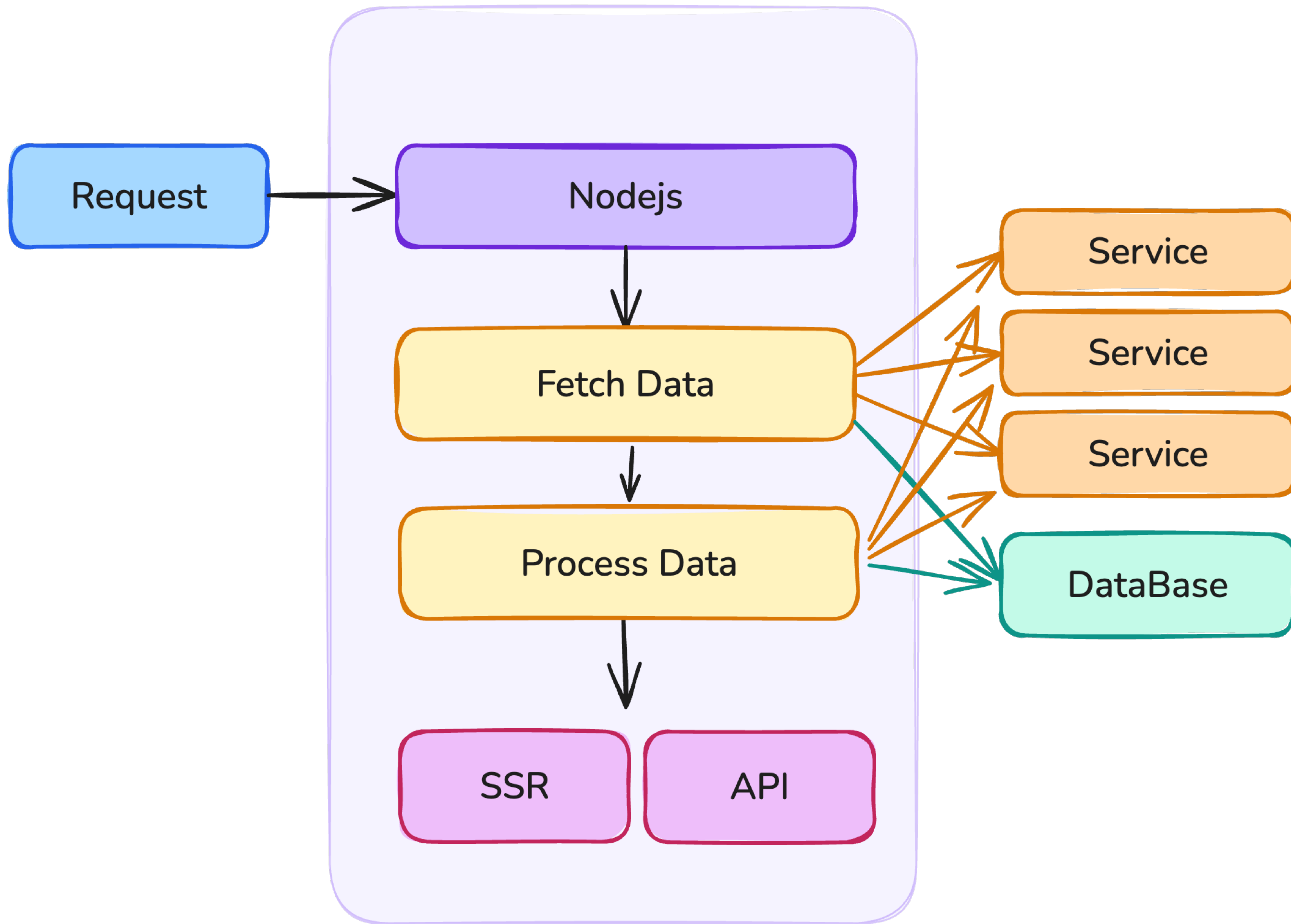
01 Rate Limiter

02 Больше инстансов

03 Выключить ненужное

Чтобы продать что-то не нужкное
надо сначала купить что-то не нужное

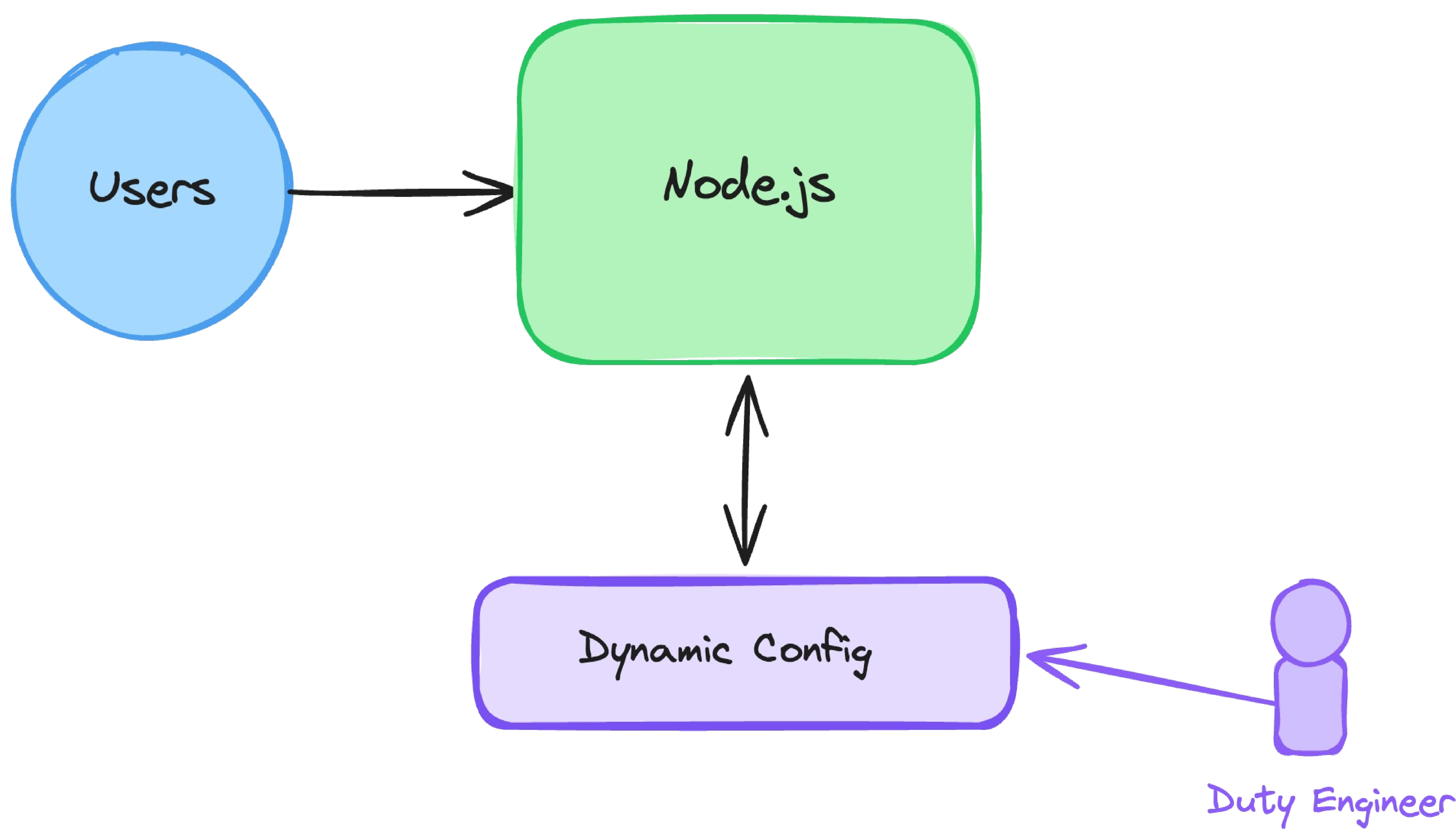
Запрос



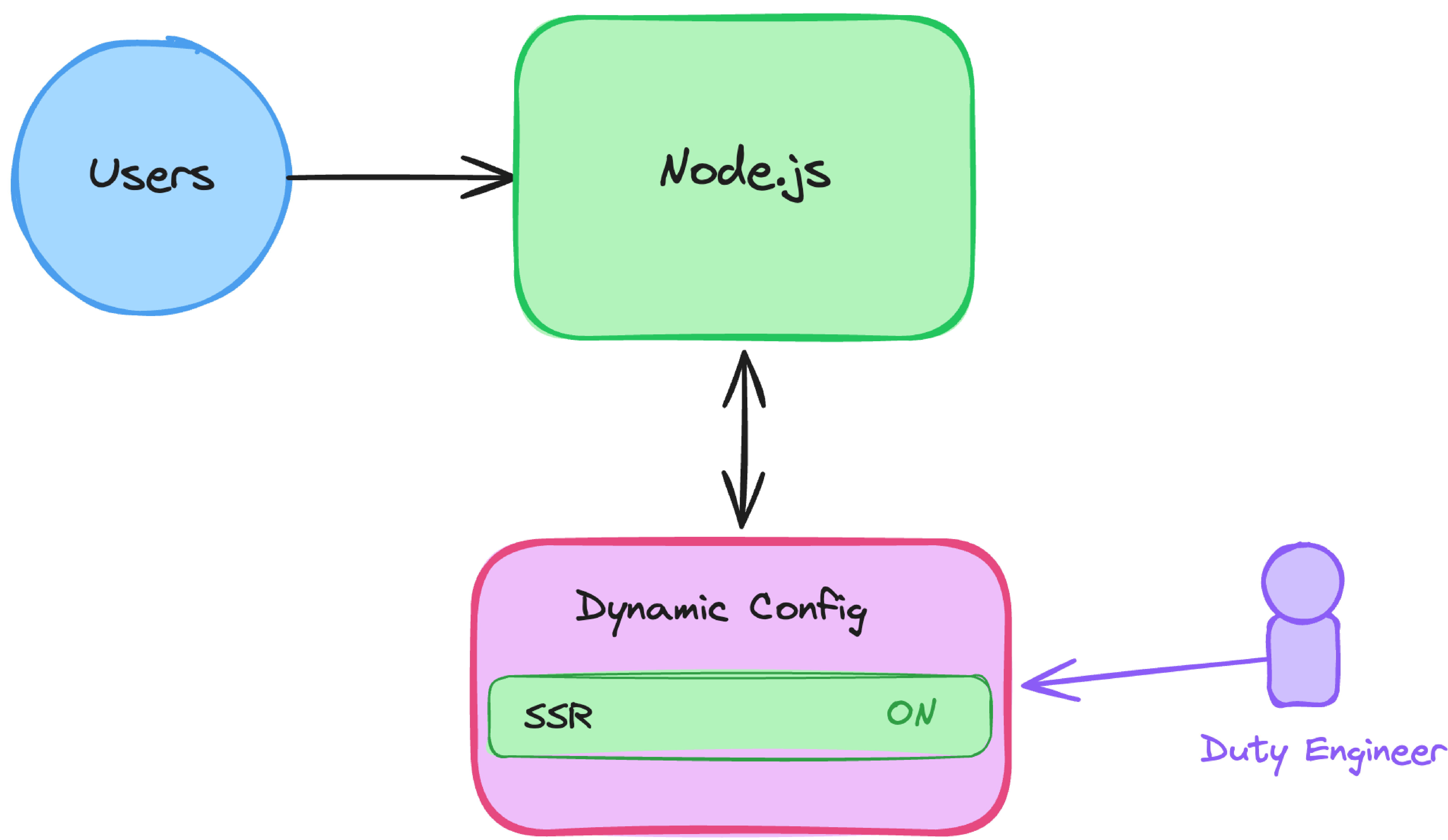
А как быстро это сделать?

Писать новый код
и катить релиз — это
долго.

Dynamic Config



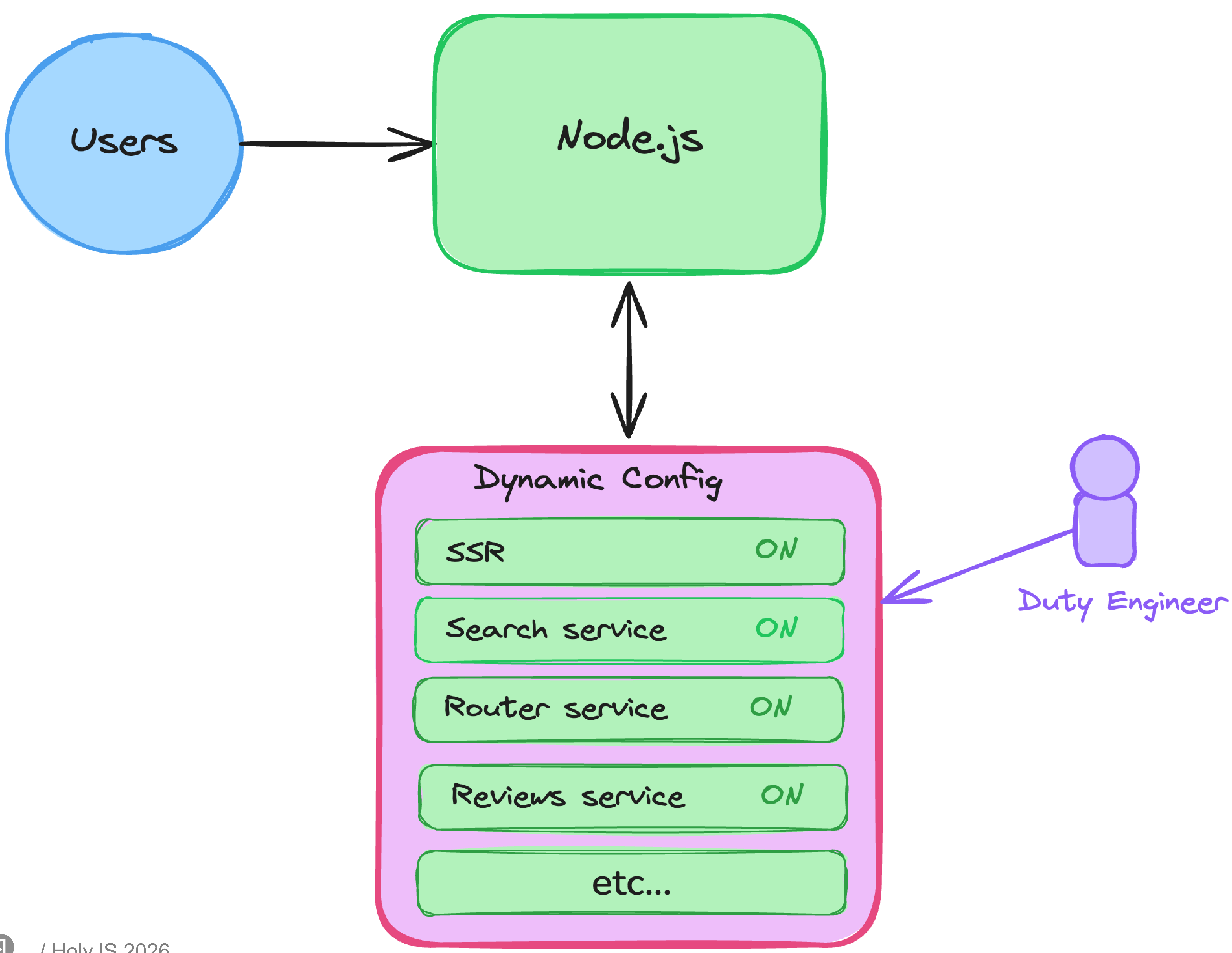
Флаг disableSSR



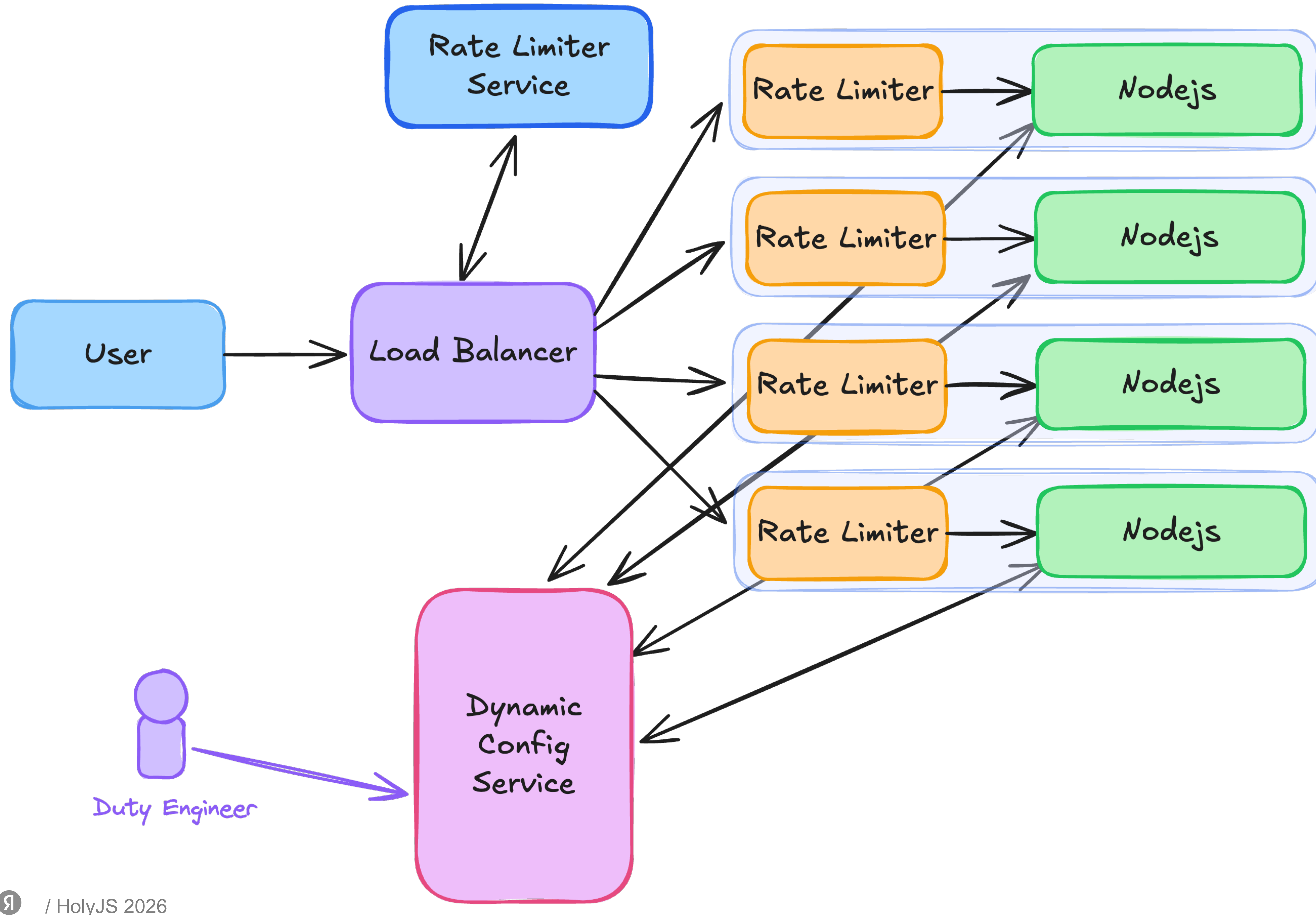
Один if в коде

```
if (dynamicConfig.disableSSR) {
    return pageWithoutSSR();
} else {
    return pageSSR();
}
```

Dynamic Config



Dynamic Config



Принципы Dynamic Config

01 Безопасность

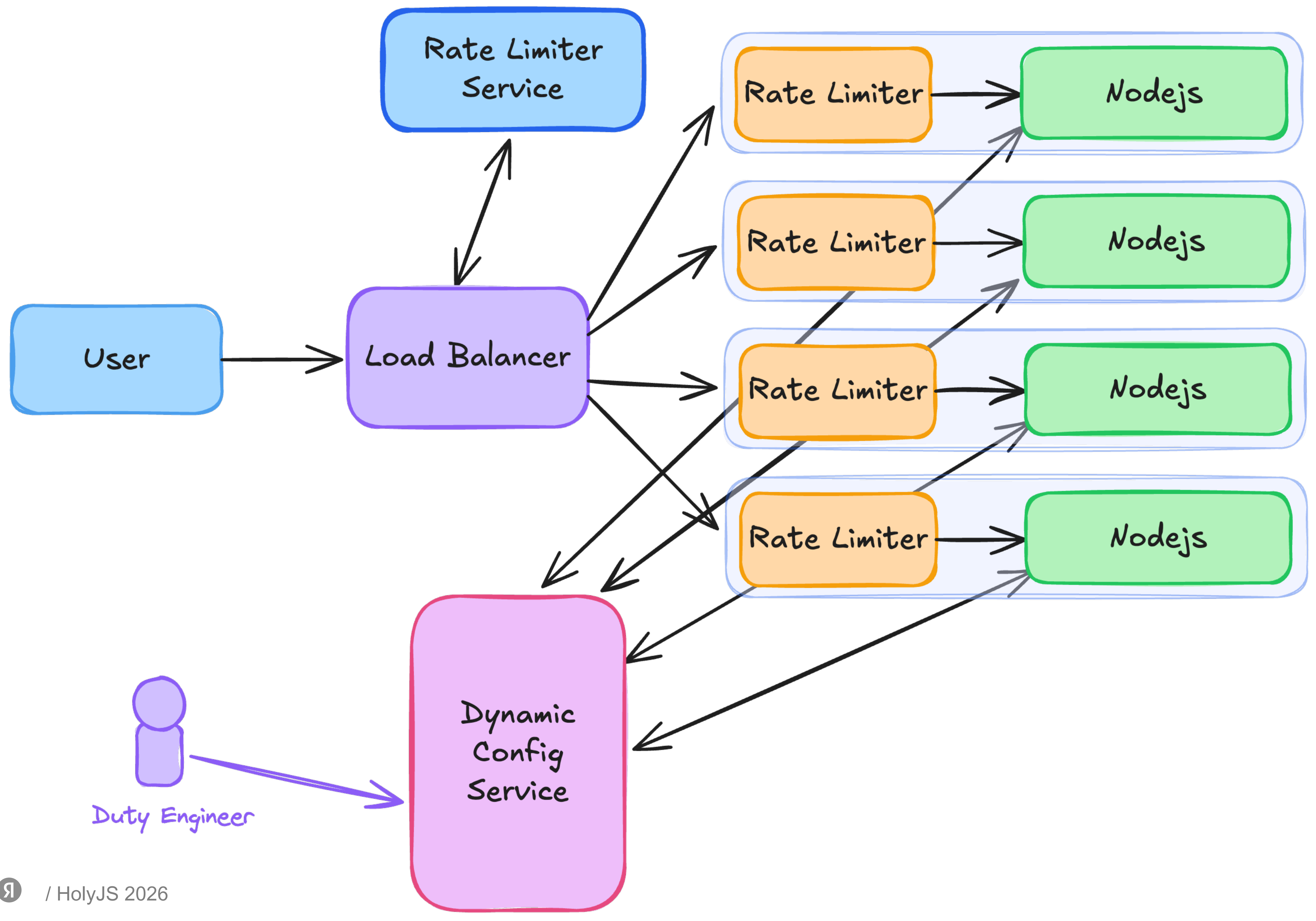
02 Аудит

03 Простота

04 Defaults

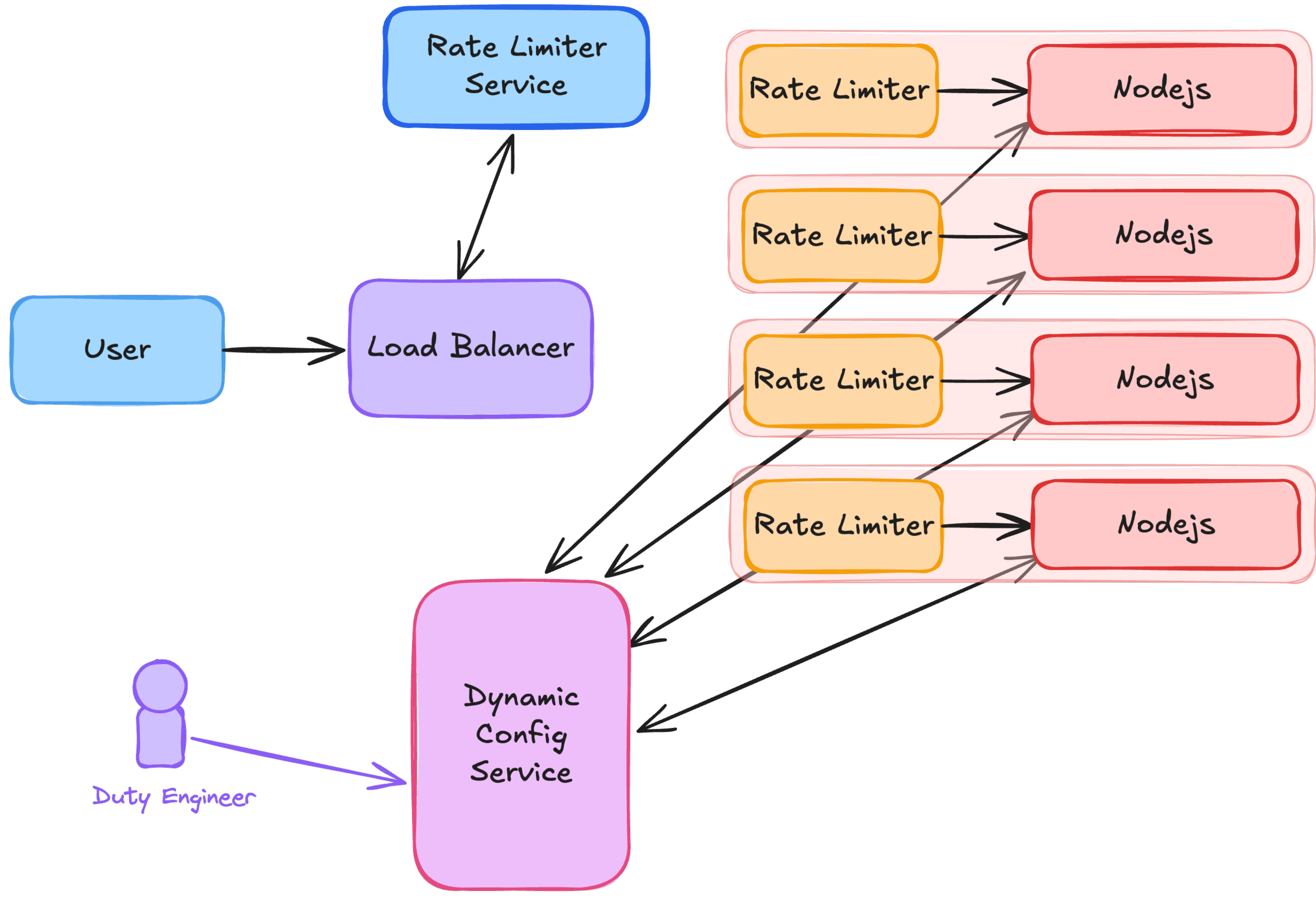
05 Время распространения

Теперь всё хорошо



Однажды пришли тяжёлые запросы

Снова каскадное падение

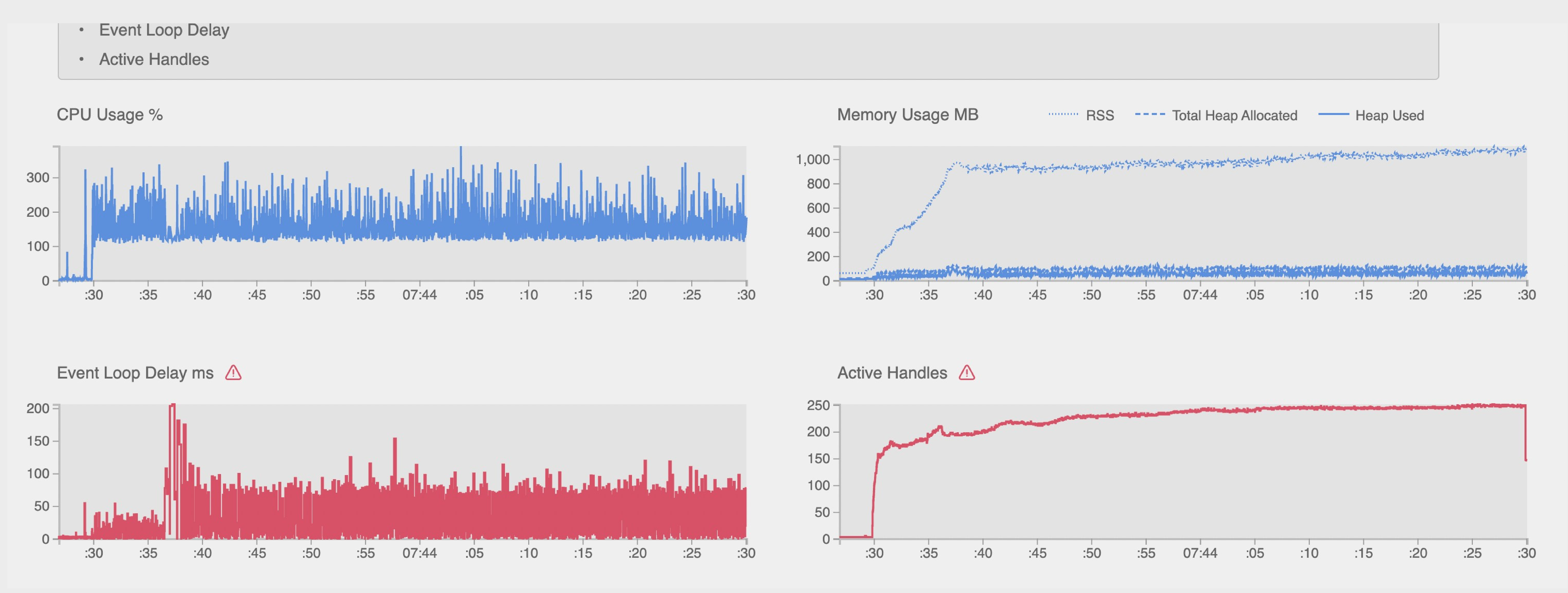


Что происходит?

Профилируем через Clinicjs

Event Loop забит

Новые запросы тоже **ждут**



Что такое «ненужный запрос»?

01

Запрос на который клиент
уже **не ждёт** ответ

02

Но внутри Node.js запрос
продолжает выполнять код
таких запросов

AbortController

01

Стандартный объект Node.js
для отмены асинхронной
операции.

02

Многие API уже умеют
работать с сигналом
отмены.

AbortController

```
app.get('/page', async (req, res) => {
       const controller = new AbortController();
       req.on('close', () => controller.abort());

       const data = await fetchData({
            signal: controller.signal,
       });
       if (controller.signal.aborted) return;
       renderPage(res, data, controller);
});
```

makeRequest с signal

```
async function makeRequest(url, abortSignal) {
	if (abortSignal.aborted) return null;
	const response = await fetch(url, {signal: req.abortSignal})
		.catch((error) => {
			if (error.name === 'AbortError') return null;
			throw error;
		});

	return response;
}
```

SSR с AbortController

```
if (req.abortSignal.aborted) return;

const { pipe, abort: abortSSR } = renderToPipeableStream(<App />);

req.abortSignal.addEventListener('abort', () => {
 abortSSR();
});
```

X2

Пропускная способность Node.js —
после нескольких
`if` в коде

Демо: 3 типа запросов

01

Light

отвечают быстро
= 0.5 MB

02

Medium

чуть больше
времени
= 3.5 MB

03

Heavy

заметная нагрузка
на Node.js

= 10 MB

Сервер демо

```
const PATTERN = ['light', 'light', 'medium', 'heavy', 'light', 'light'];
let counter = 0;
app.get('/route', async (_req, res) => {
        counter += 1;
        const mode = PATTERN[counter % PATTERN.length];
        …
        const data = await makeRequest(`${ROUTES_URL}/${mode}.json`, {
            timeout: 4000,
        });
      …
      res.json({ mode, data });
```

makeRequest

```
async function makeRequest(url, { timeout }) {
        const timeoutPromise = new Promise((_, reject) => {
              setTimeout(() => reject(new Error('timeout')), timeout);
        });
        const response = await Promise.race([fetch(url), timeoutPromise]);
        const data = await response.json();
        return data;
}
```

Нагружаем без AbortController

```
—— autocannon: v1  (timeout 5s) ——
$ npx autocannon -c 100 -d 60 -t 5 http://localhost:4000/route

Running 60s test @ http://localhost:4000/route
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	371 ms	1293 ms	4643 ms	4803 ms	1321.23 ms	786.24 ms	4996 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	7	18	27	58	30.65	12.09	7
Bytes/Sec	2.72 MB	14.9 MB	18.4 MB	79.6 MB	26.9 MB	17.9 MB	2.72 MB

```
Req/Bytes counts sampled once per second.
# of samples: 60

1833 2xx responses, 5 non 2xx responses
3k requests in 60.05s, 1.61 GB read
677 errors (677 timeouts)
```

`autocannon`

— 100 подключений
× 60 сек, таймаут 5 сек.

1833 2xx

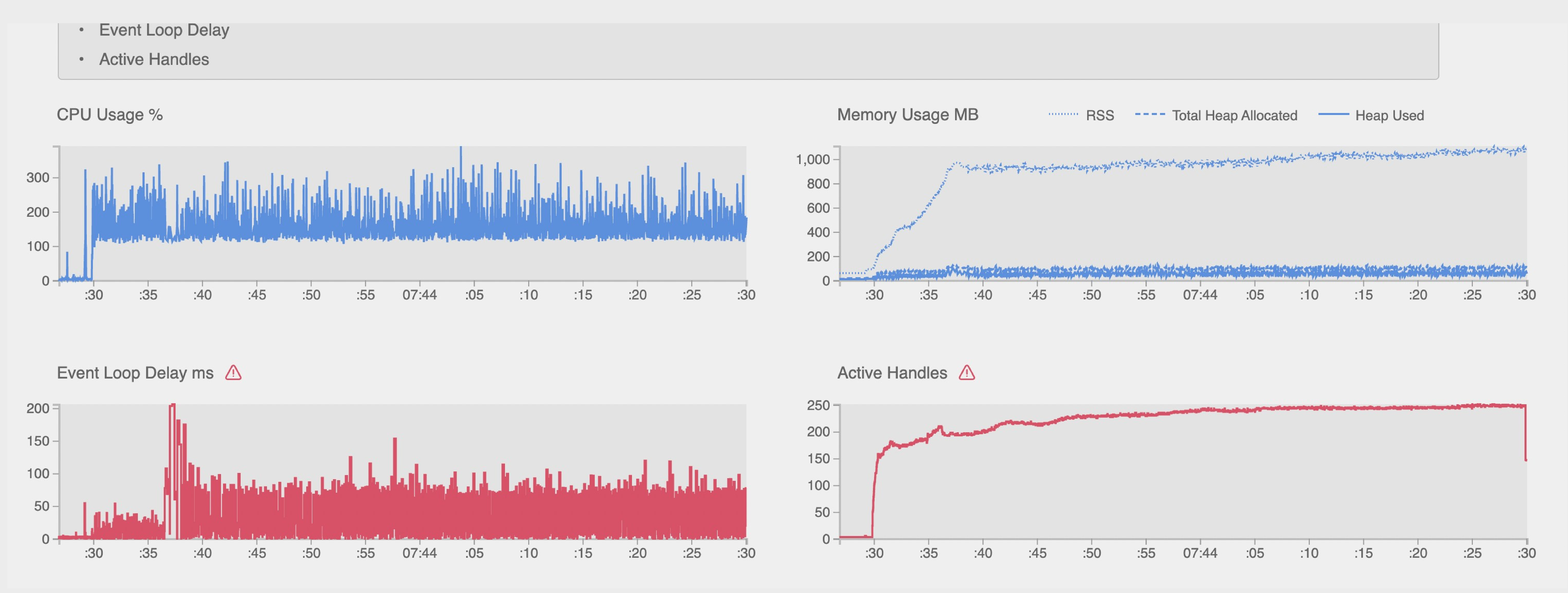
5 non 2xx

677 timeouts

1.6 GB read

Event Loop без AbortController

Event loop **загружен**. Обычные запросы страдают вместе с тяжёлыми.



Сервер с AbortController

```
app.get('/route', async (req, res) => {
    counter += 1;
    const mode = PATTERN[counter % PATTERN.length];
    const ac = new AbortController();
    req.ac = ac;
    req.on('close', () => req.ac.abort());

    const data = await makeRequest(`${ROUTES_URL}/${mode}.json`, {
        timeout: 4000,
        req,
    });
    …
```

makeRequest с signal

```
async function makeRequest(url, { timeout, req }) {
      const timeoutPromise = new Promise((_, reject) => {
          setTimeout(() => reject(new Error('timeout')), timeout);
      });
      const response = await Promise.race([
             fetch(url, { signal: req.ac.signal }).catch((err) => {
                 if (err.name === 'AbortError') return null;
                 throw err;
             }),
            timeoutPromise,
      ]);
        ...
```

Тот же сценарий с AbortController

```
─── autocannon: v2  (timeout 5s) ───
$ npx autocannon -c 100 -d 60 -t 5 http://localhost:4000/route

Running 60s test @ http://localhost:4000/route
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	347 ms	421 ms	2013 ms	2266 ms	714.43 ms	604.6 ms	4046 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	49	52	59	65	59.09	5.1	49
Bytes/Sec	39.4 MB	61.1 MB	72.1 MB	87.7 MB	71.6 MB	6.85 MB	39.4 MB

```
Req/Bytes counts sampled once per second.
# of samples: 60

3540 2xx responses, 5 non 2xx responses
4k requests in 60.03s, 4.29 GB read
659 errors (659 timeouts)
```

3540 2xx

5 non 2xx

659 timeouts

4.3GB read

Event Loop с AbortController

Активных обработчиков меньше, задержка EventLoop меньше.
И при этом в 2 раза больше запросов.

Сервер с отменой, но без AbortController

```
app.get('/route', async (req, res) => {
        counter += 1;
        const mode = PATTERN[counter % PATTERN.length];
        req.cancelled = false;
        req.on('close', () => req.cancelled = true);

        const data = await makeRequest(`${ROUTES_URL}/${mode}.json`, {
                timeout: 4000,
                req,
        });
        …
```

makeRequest с отменой, но без signal

```
async function makeRequest(url, { timeout, req }) {
      if (req.cancelled) return null;
      const timeoutPromise = new Promise((_, reject) => {
          setTimeout(() => reject(new Error('timeout')), timeout);
      });
      const response = await Promise.race([
            fetch(url),
            timeoutPromise,
      ]);
      if (req.cancelled) return null;
        ...
```

Тот же сценарий с отменой, но без AbortController

```
── autocannon: v3  (timeout 5s) ──
$ npx autocannon -c 100 -d 60 -t 5 http://localhost:4000/route

Running 60s test @ http://localhost:4000/route
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	413 ms	907 ms	4298 ms	4445 ms	1419.93 ms	1226.49 ms	4652 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	19	29	40	62	41	8.16	19
Bytes/Sec	17.7 MB	36 MB	48.3 MB	85.7 MB	49.2 MB	9.79 MB	17.7 MB

```
Req/Bytes counts sampled once per second.
# of samples: 60

2459 2xx responses, 1 non 2xx responses
3k requests in 60.04s, 2.95 GB read
466 errors (466 timeouts)
```

2459 2xx

1 non 2xx

466 timeouts

2.9GB read

Event Loop с отменой, но без AbortController

Активных обработчиков меньше, задержка EventLoop меньше.
И при этом на 30% больше запросов чем без отмены.

Сравнение

Без AbortController

1833 запросов

677 timeouts

4.8 сек p99

1.6 GB read

С AbortController

3540 запросов

659 timeouts

2.3 сек p99

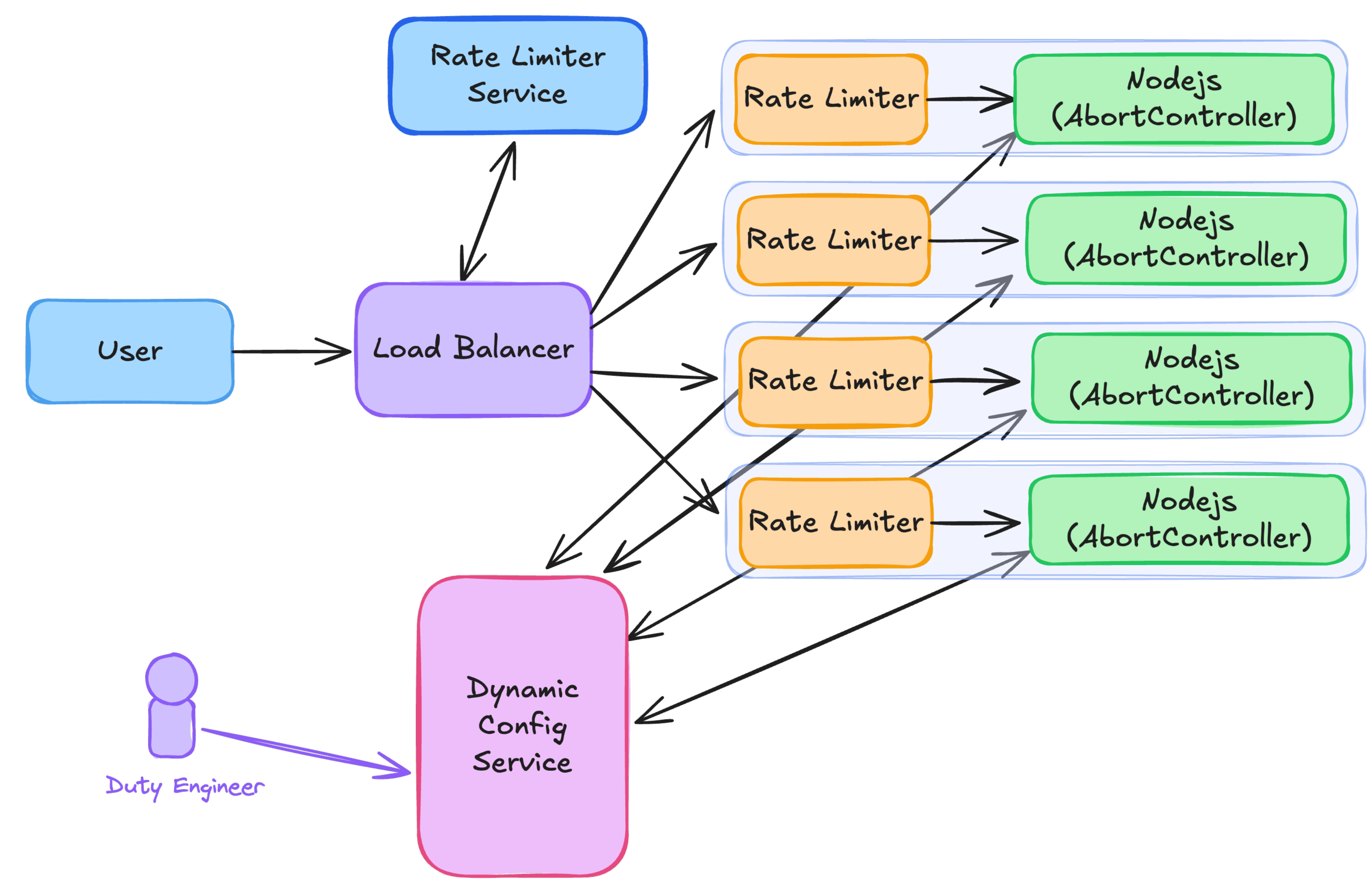
4.3 GB read

Три уровня защиты

01 Rate Limiter

02 Dynamic Config

03 AbortController



Сервис деградирует, а не падает

Слои защиты

Ни один уровень не идеален — каждый можно обойти.

Но когда они собраны вместе, защита становится сильнее.

Каждый уровень дополняет другой.

Яндекс

Спасибо!

Глеб Решетнев

Старший разработчик,
Яндекс Карты