

С# Асинхронное программирование

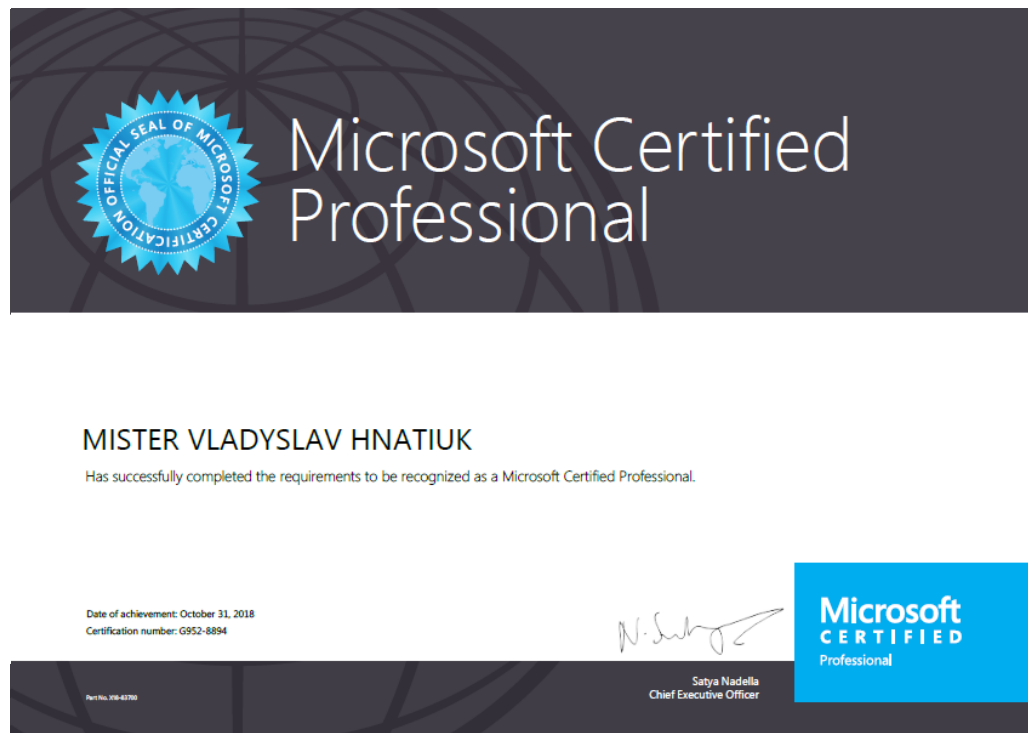
Работа контекста синхронизации с `async await`.
Роль `async await` в ASP.NET

C# Асинхронное программирование

Автор курса



Гнатюк Владислав



MCID:16354168

Работа контекста синхронизации в `async await`.
Роль `async await` в ASP.NET

C# Асинхронное программирование

План урока

- 1) Использование `async await` в WPF
- 2) Класс `SynchronizationContext` – контекст синхронизации
- 3) Продолжения оператора `await`
- 4) Управление ожиданием
- 5) Класс `ExecutionContext` – контекст выполнения
- 6) Модификатор `async` для `void`
- 7) Асинхронные лямбда выражения
- 8) Использование `async await` в ASP.NET

C# Асинхронное программирование

Использование `async await` в WPF

Использование ключевых слов `async await` упрощает асинхронное программирование в приложениях, написанных по технологии WPF.

Сложность асинхронного кода для приложений WPF всегда была в том, чтобы обращаться к элементам управления из потока пользовательского интерфейса.

Приходилось либо постоянно разными способами передавать данные из одного потока в другой, либо блокировать поток пользовательского интерфейса на время выполнения. Трудностей в передаче данных между потоками не было, но код становился грязным, тяжелым к рассмотрению и изменениям.

Использование ключевых слов `async await` делает работу с WPF достаточно простой. Асинхронный код выглядит как синхронный.

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    var operationResult = Operation();
    txtResult.Text = operationResult;
}
```

```
private async void Button_Click(object sender, RoutedEventArgs e)
{
    var operationResult = await OperationAsync();
    txtResult.Text = operationResult;
}
```

C# Асинхронное программирование

Использование `async await` в WPF

Использование ключевых слов `async await` упрощает асинхронное программирование в приложениях, написанных по технологии WPF.

Сложность асинхронного кода для приложений WPF всегда была в том, чтобы обращаться к элементам управления из потока пользовательского интерфейса.

Приходилось либо постоянно разными способами передавать данные из одного потока в другой, либо блокировать поток пользовательского интерфейса на время выполнения. Трудностей в передаче данных между потоками не было, но код становился грязным, тяжелым к рассмотрению и изменениям.

Использование ключевых слов `async await` делает работу с WPF достаточно простой. Асинхронный код выглядит как синхронный.

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    var operationResult = Operation();
    txtResult.Text = operationResult;
}
```

```
private async void Button_Click(object sender, RoutedEventArgs e)
{
    var operationResult = await OperationAsync();
    txtResult.Text = operationResult;
}
```

C# Асинхронное программирование

Использование `async await` в WPF


Использование ключевых слов `async await` упрощает асинхронное программирование в приложениях, написанных по технологии WPF.

Сложность асинхронного кода для приложений WPF всегда была в том, чтобы обращаться к элементам управления из потока пользовательского интерфейса.

Приходилось либо постоянно разными способами передавать данные из одного потока в другой, либо блокировать поток пользовательского интерфейса на время выполнения. Трудностей в передаче данных между потоками не было, но код становился грязным, тяжелым к рассмотрению и изменениям.

Использование ключевых слов `async await` делает работу с WPF достаточно простой. Асинхронный код выглядит как синхронный.

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    var operationResult = Operation();
    txtResult.Text = operationResult;
}
```



```
private async void Button_Click(object sender, RoutedEventArgs e)
{
    var operationResult = await OperationAsync();
    txtResult.Text = operationResult;
}
```

C# Асинхронное программирование

Использование `async await` в WPF

Использование ключевых слов `async await` упрощает асинхронное программирование в приложениях, написанных по технологии WPF.

Сложность асинхронного кода для приложений WPF всегда была в том, чтобы обращаться к элементам управления из потока пользовательского интерфейса.

Приходилось либо постоянно разными способами передавать данные из одного потока в другой, либо блокировать поток пользовательского интерфейса на время выполнения. Трудностей в передаче данных между потоками не было, но код становился грязным, тяжелым к рассмотрению и изменениям.

Использование ключевых слов `async await` делает работу с WPF достаточно простой. Асинхронный код выглядит как синхронный.

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    var operationResult = Operation();
    txtResult.Text = operationResult;
}
```

```
private async void Button_Click(object sender, RoutedEventArgs e)
{
    var operationResult = await OperationAsync();
    txtResult.Text = operationResult;
}
```


C# Асинхронное программирование

Использование `async await` в WPF

Использование ключевых слов `async await` упрощает асинхронное программирование в приложениях, написанных по технологии WPF.

Сложность асинхронного кода для приложений WPF всегда была в том, чтобы обращаться к элементам управления из потока пользовательского интерфейса.

Приходилось либо постоянно разными способами передавать данные из одного потока в другой, либо блокировать поток пользовательского интерфейса на время выполнения. Трудностей в передаче данных между потоками не было, но код становился грязным, тяжелым к рассмотрению и изменениям.

Использование ключевых слов `async await` делает работу с WPF достаточно простой. Асинхронный код выглядит как синхронный.

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    var operationResult = Operation();
    txtResult.Text = operationResult;
}
```

```
private async void Button_Click(object sender, RoutedEventArgs e)
{
    var operationResult = await OperationAsync();
    txtResult.Text = operationResult;
}
```

C# Асинхронное программирование

Использование `async await` в WPF

Использование ключевых слов `async await` упрощает асинхронное программирование в приложениях, написанных по технологии WPF.

Сложность асинхронного кода для приложений WPF всегда была в том, чтобы обращаться к элементам управления из потока пользовательского интерфейса.

Приходилось либо постоянно разными способами передавать данные из одного потока в другой, либо блокировать поток пользовательского интерфейса на время выполнения. Трудностей в передаче данных между потоками не было, но код становился грязным, тяжелым к рассмотрению и изменениям.

Использование ключевых слов `async await` делает работу с WPF достаточно простой. Асинхронный код выглядит как синхронный.

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    var operationResult = Operation();
    txtResult.Text = operationResult;
}
```

```
private async void Button_Click(object sender, RoutedEventArgs e)
{
    var operationResult = await OperationAsync();
    txtResult.Text = operationResult;
}
```

C# Асинхронное программирование

Использование `async await` в WPF

Использование ключевых слов `async await` упрощает асинхронное программирование в приложениях, написанных по технологии WPF.


Сложность асинхронного кода для приложений WPF всегда была в том, чтобы обращаться к элементам управления из потока пользовательского интерфейса.

Приходилось либо постоянно разными способами передавать данные из одного потока в другой, либо блокировать поток пользовательского интерфейса на время выполнения. Трудностей в передаче данных между потоками не было, но код становился грязным, тяжелым к рассмотрению и изменениям.

Использование ключевых слов `async await` делает работу с WPF достаточно простой. Асинхронный код выглядит как синхронный.

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    var operationResult = Operation();
    txtResult.Text = operationResult;
}
```

```
private async void Button_Click(object sender, RoutedEventArgs e)
{
    var operationResult = await OperationAsync();
    txtResult.Text = operationResult;
}
```



C# Асинхронное программирование

Использование `async await` в WPF

Использование ключевых слов `async await` упрощает асинхронное программирование в приложениях, написанных по технологии WPF.

Сложность асинхронного кода для приложений WPF всегда была в том, чтобы обращаться к элементам управления из потока пользовательского интерфейса.

Приходилось либо постоянно разными способами передавать данные из одного потока в другой, либо блокировать поток пользовательского интерфейса на время выполнения. Трудностей в передаче данных между потоками не было, но код становился грязным, тяжелым к рассмотрению и изменениям.

Использование ключевых слов `async await` делает работу с WPF достаточно простой. Асинхронный код выглядит как синхронный.

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    var operationResult = Operation();
    txtResult.Text = operationResult;
}
```

```
private async void Button_Click(object sender, RoutedEventArgs e)
{
    var operationResult = await OperationAsync();
    txtResult.Text = operationResult;
}
```

C# Асинхронное программирование

SynchronizationContext

SynchronizationContext – базовый класс для создания контекстов синхронизации. **Контекст синхронизации** – абстрактный механизм, который позволяет выполнить код вашего приложения в определенном месте.

Служит для обеспечения единого механизма распространения контекста синхронизации в различных моделях синхронизации.

SynchronizationContext позволяет расширять себя и предоставлять свои собственные реализации методов.

C# Асинхронное программирование

Основной функционал класса SynchronizationContext

Свойства:

- Current (*static*) – отдает контекст синхронизации, прикрепленный к текущему потоку.

Методы:

- CreateCopy (*virtual*) – при переопределении в производном классе создает копию контекста синхронизации.
- OperationStarted (*virtual*) – при переопределении в производном классе отвечает за уведомление о начале операции.
- OperationCompleted (*virtual*) – при переопределении в производном классе отвечает за уведомление о завершении операции.
- Send (*virtual*) – при переопределении в производном классе отправляет синхронное сообщение в контекст синхронизации.
- Post (*virtual*) – при переопределении в производном классе отправляет асинхронное сообщение в контекст синхронизации.
- SetSynchronizationContext (*static*) – задает текущий контекст синхронизации.

C# Асинхронное программирование

Post vs Send

В чем отличия этих двух методов?

Стандартная реализация метода Send:

```
public virtual void Send(SendOrPostCallback callback, object state)
{
    callback(state);
}
```

Стандартная реализация метода Post:

```
public virtual void Post(SendOrPostCallback callback, object state)
{
    ThreadPool.QueueUserWorkItem(new WaitCallback(callback.Invoke), state);
}
```

Метод Post производит асинхронный посыл сообщения. Он не ждет окончания работы делегата для собственного завершения, в отличие от метода Send.

При переопределении рекомендуется оставлять для метода Send синхронный посыл сообщения, для метода Post – асинхронный (но это является только рекомендацией, вам решать как они будут переопределены).

C# Асинхронное программирование

Использование контекста синхронизации

1. Создание своего класса, производного от `SynchronizationContext`.
2. Создание экземпляра класса `SynchronizationContext` или производного от него класса.
3. Регистрация этого контекста синхронизации с помощью метода **`SetSynchronizationContext`**.

Когда вы захотите обратиться к контексту синхронизации, вы должны запросить его с помощью статического свойства **`Current`**. На полученном контексте, с помощью метода `Post` или `Send`, вы можете отправить сообщение в контекст синхронизации.

C# Асинхронное программирование

Продолжения оператора await

Оператор `await` имеет установленные правила для выполнения своих продолжений. Он выполняет продолжение в контексте одного из заранее определенных механизмов. **Всегда выбирается только один из них.**

Поиск механизма, который может выполнить продолжение оператора `await`, идет в следующем порядке (по умолчанию):

1. Контекст синхронизации. Оператор `await` вначале пытается захватить контекст синхронизации. Если он захватил его, то продолжение будет отправлено на выполнение в контекст синхронизации.
2. Планировщик задач. Оператор `await` пытается получить текущий планировщик задач. Если он его получает, то продолжение будет отправлено на выполнение через планировщик задач.
3. Если предыдущие варианты не сработали, или если было рекомендовано отказаться от окружения вызывающего потока, тогда система будет выбирать где его выполнить. Обычно это означает одно из двух:
 - Выполнение продолжения синхронно там, где была завершена ожидаемая задача.
 - Выполнение продолжения в контексте пула потоков (ThreadPool).

C# Асинхронное программирование

Управление ожиданием

Есть открытый API для управления ожиданием. Для управления ожиданием используется метод `ConfigureAwait(bool continueOnCapturedContext)`.

Метод позволяет вам порекомендовать системе, нужно ли выполняться продолжениям в захваченном контексте синхронизации или планировщике задач.

- Нужно указать значение `true` для параметра `continueOnCapturedContext`, чтобы разрешить выполнение продолжения в захваченном контексте синхронизации или планировщике задач.
- Нужно указать значение `false` для параметра `continueOnCapturedContext`, чтобы запретить выполнение продолжения в захваченном контексте синхронизации или планировщике задач.

Если вы не вызываете явно этот метод, то по умолчанию оператор `await` всегда будет пытаться захватить контекст синхронизации и выполнить продолжение в нем. Все равно, что вы вызвали метод `ConfigureAwait` и передали значение `true`.

C# Асинхронное программирование

ConfigureAwait

Метод **ConfigureAwait** возвращает структуру, которая имеет весь необходимый функционал по работе с оператором **await**.

Структура занимается конфигурированием оператора **await** на выполнение продолжения указанным вами способом.

```
public struct ConfiguredTaskAwaitable
{
    public ConfiguredTaskAwaiter GetAwaiter();

    public struct ConfiguredTaskAwaiter : ICriticalNotifyCompletion, INotifyCompletion
    {
        public bool IsCompleted { get; }

        public void GetResult();
        public void OnCompleted(Action continuation);
        public void UnsafeOnCompleted(Action continuation);
    }
}

public struct ConfiguredTaskAwaitable<TResult>
{
    public ConfiguredTaskAwaiter GetAwaiter();

    public struct ConfiguredTaskAwaiter : ICriticalNotifyCompletion, INotifyCompletion
    {
        public bool IsCompleted { get; }

        public TResult GetResult();
        public void OnCompleted(Action continuation);
        public void UnsafeOnCompleted(Action continuation);
    }
}
```

C# Асинхронное программирование

ConfigureAwait

Если на момент применения оператора `await` асинхронная задача уже завершена, то код продолжит выполняться в том же потоке синхронно.

Вызов метода `ConfigureAwait(false)` будет бесполезен. Поэтому, метод `ConfigureAwait()` с указанием значения `false` **не гарантирует**, что код после него **не будет выполнен** в оригинальном контексте синхронизации или планировщике задач.

C# Асинхронное программирование

Продолжения оператора await

По завершению ожидаемой задачи, среда выполнения будет запускать продолжение. Но перед этим она может проверить текущий контекст в возобновляющем потоке для определения возможности синхронного запуска продолжения. Если будет получен отказ, то продолжение будет выполнено асинхронно запланированным способом при его создании, как и подразумевалось.

Эта проверка может выполняться независимо от указаний метода `ConfigureAwaitAwait`.

C# Асинхронное программирование

ExecutionContext

ExecutionContext (контекст выполнения) – это объект, который представляет собой контейнер для хранения информации потока выполнения. В .NET Framework он хранит в себе другие контексты, к примеру, SecurityContext, SynchronizationContext, HostExecutionContext и другие...

С помощью ExecutionContext можно захватить состояние одного потока и восстановить его в другом.

Контекст выполнения в async await захватывается строителями асинхронных методов, если он не был подавлен до их работы.

C# Асинхронное программирование

Модификатор `async` для `void`

Для асинхронных методов с возвращаемым значением `void` существует потенциальное взаимодействие с контекстом синхронизации. Взаимодействие описано внутри строителя асинхронных методов `AsyncVoidMethodBuilder`.

Если контекст синхронизации будет захвачен, произойдет следующее:

- При создании строителя `AsyncVoidMethodBuilder` будет захвачен контекст синхронизации и если он не будет `null`, то среда вызовет метод `OperationStarted` на захваченном контексте.
- Если конечный автомат завершает работу с необработанным исключением (ловит его методом `SetException`), то исключение будет проброшено в захваченный контекст синхронизации.
- По завершению работы конечного автомата (успешном или провальном) будет вызван метод `OperationCompleted` на захваченном контексте синхронизации.

Если контекст синхронизации не будет захвачен, то вызовов методов `OperationStarted` и `OperationCompleted` не будет. Возникшее необработанное исключение, в свою очередь, будет выброшено через `ThreadPool`.

C# Асинхронное программирование

Асинхронные лямбда выражения

Лямбда выражения могут быть асинхронными. На них накладываются все правила и ограничения асинхронных методов.

Создание асинхронных лямбда выражений:

НЕПРАВИЛЬНЫЙ ВАРИАНТ

```
private async Task MethodAsync()
{
    // ....
    Func<Task> func = () =>
    {
        await DoSomethingAsync();
    };
    // ....
    await func.Invoke();
}
```

ПРАВИЛЬНЫЙ ВАРИАНТ

```
private async Task MethodAsync()
{
    // ....
    Func<Task> func = async () =>
    {
        await DoSomethingAsync();
    };
    // ....
    await func.Invoke();
}
```

Чтобы сделать лямбда выражение асинхронным – добавьте модификатор **async** перед указанием формальных параметров лямбда выражения. Пример: **async** () => { ... }

C# Асинхронное программирование

Асинхронные лямбда выражения

Лямбда выражения могут быть асинхронными. На них накладываются все правила и ограничения асинхронных методов.

Создание асинхронных лямбда выражений:

НЕПРАВИЛЬНЫЙ ВАРИАНТ

```
private async Task MethodAsync()
{
    // ....
    Func<Task> func = () =>
    {
        await DoSomethingAsync();
    };
    // ....
    await func.Invoke();
}
```

ПРАВИЛЬНЫЙ ВАРИАНТ

```
private async Task MethodAsync()
{
    // ....
    Func<Task> func = async () =>
    {
        await DoSomethingAsync();
    };
    // ....
    await func.Invoke();
}
```

Чтобы сделать лямбда выражение асинхронным – добавьте модификатор **async** перед указанием формальных параметров лямбда выражения. Пример: **async** () => { ... }

C# Асинхронное программирование

Асинхронные лямбда выражения

Лямбда выражения могут быть асинхронными. На них накладываются все правила и ограничения асинхронных методов.


Создание асинхронных лямбда выражений:

НЕПРАВИЛЬНЫЙ ВАРИАНТ

```
private async Task MethodAsync()
{
    // ....
    Func<Task> func = () =>
    {
        await DoSomethingAsync();
    };
    // ....
    await func.Invoke();
}
```

ПРАВИЛЬНЫЙ ВАРИАНТ

```
private async Task MethodAsync()
{
    // ....
    Func<Task> func = async () =>
    {
        await DoSomethingAsync();
    };
    // ....
    await func.Invoke();
}
```



Чтобы сделать лямбда выражение асинхронным – добавьте модификатор `async` перед указанием формальных параметров лямбда выражения. Пример: `async () => { ... }`

C# Асинхронное программирование

Асинхронные лямбда выражения

Лямбда выражения могут быть асинхронными. На них накладываются все правила и ограничения асинхронных методов.

Создание асинхронных лямбда выражений:

НЕПРАВИЛЬНЫЙ ВАРИАНТ

```
private async Task MethodAsync()
{
    // ....
    Func<Task> func = () =>
    {
        await DoSomethingAsync();
    };
    // ....
    await func.Invoke();
}
```

ПРАВИЛЬНЫЙ ВАРИАНТ

```
private async Task MethodAsync()
{
    // ....
    Func<Task> func = async () =>
    {
        await DoSomethingAsync();
    };
    // ....
    await func.Invoke();
}
```

Чтобы сделать лямбда выражение асинхронным – добавьте модификатор **async** перед указанием формальных параметров лямбда выражения. Пример: **async** () => { ... }

C# Асинхронное программирование

Делегаты с возвращаемыми значениями void

Будьте аккуратны при работе с асинхронными лямбда-выражениями. Вы можете не заметив использовать асинхронное лямбда-выражение с типом возвращаемого значения void.

Из-за этого, вы можете потерять преимущества использования задач (ожидание, результат, статус, отлов исключения) и получить непредсказуемый проброс исключения.

```
Action action = async () =>
{
    await Task.Run();
}
```

```
Func<Task> func = async () =>
{
    await Task.Run();
}
```

```
private async void Lambda1()
{
    await Task.Run();
}
```

```
private async Task Lambda2()
{
    await Task.Run();
}
```

C# Асинхронное программирование

Делегаты с возвращаемыми значениями void

Будьте аккуратны при работе с асинхронными лямбда-выражениями. Вы можете не заметив использовать асинхронное лямбда-выражение с типом возвращаемого значения void.

Из-за этого, вы можете потерять преимущества использования задач (ожидание, результат, статус, отлов исключения) и получить непредсказуемый проброс исключения.

```
Action action = async () =>
{
    await Task.Run();
}
```



```
private async void Lambda1()
{
    await Task.Run();
}
```

```
Func<Task> func = async () =>
{
    await Task.Run();
}
```

```
private async Task Lambda2()
{
    await Task.Run();
}
```

C# Асинхронное программирование

Делегаты с возвращаемыми значениями void

Будьте аккуратны при работе с асинхронными лямбда-выражениями. Вы можете не заметив использовать асинхронное лямбда-выражение с типом возвращаемого значения void.

Из-за этого, вы можете потерять преимущества использования задач (ожидание, результат, статус, отлов исключения) и получить непредсказуемый проброс исключения.

```
Action action = async () =>
{
    await Task.Run();
}
```

```
private async void Lambda1()
{
    await Task.Run();
}
```

```
Func<Task> func = async () =>
{
    await Task.Run();
}
```



```
private async Task Lambda2()
{
    await Task.Run();
}
```

C# Асинхронное программирование

Делегаты с возвращаемыми значениями void

Будьте аккуратны при работе с асинхронными лямбда-выражениями. Вы можете не заметив использовать асинхронное лямбда-выражение с типом возвращаемого значения void.

Из-за этого, вы можете потерять преимущества использования задач (ожидание, результат, статус, отлов исключения) и получить непредсказуемый проброс исключения.

```
Action action = async () =>
{
    await Task.Run();
}
```

```
Func<Task> func = async () =>
{
    await Task.Run();
}
```

```
private async void Lambda1()
{
    await Task.Run();
}
```

```
private async Task Lambda2()
{
    await Task.Run();
}
```

C# Асинхронное программирование

Асинхронность в ASP.NET

Асинхронное программирование является важной частью приложений ASP.NET. Потому, что благодаря ему вы можете увеличить пропускную способность входящих запросов для вашего приложения.



C# Асинхронное программирование

IIS сервер

Получением запросов в приложениях ASP.NET занимается IIS сервер. Именно через него запрос пользователя попадает в наше веб-приложение.

IIS (Internet Information Services) – расширяемый веб-сервер от компании Microsoft.

Веб-сервер IIS поддерживает технологию создания веб-приложений ASP.NET. Технология ASP.NET – это одно из основных средств для создания веб-приложений и веб-служб через IIS сервер.

C# Асинхронное программирование

Обработка запросов

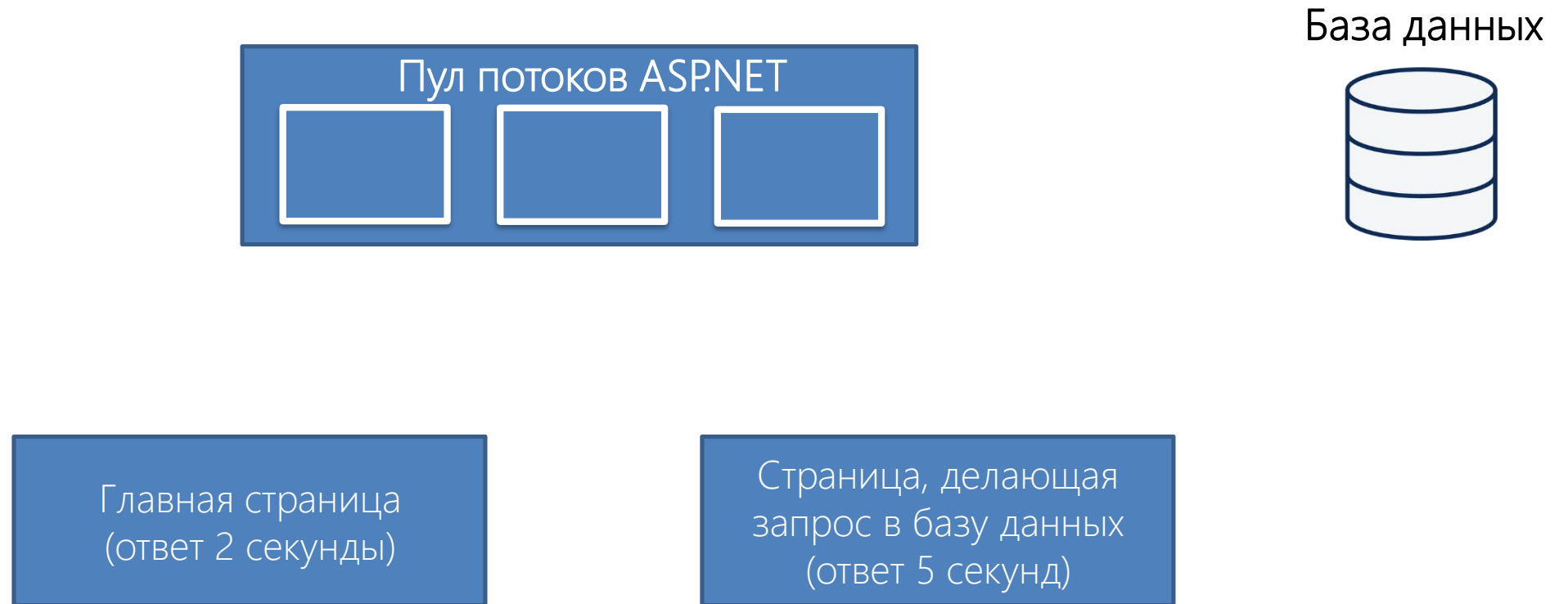
Каждый входящий запрос в веб-приложение ASP.NET обрабатывается потоком из пула потоков. Поток из пула потоков не будет освобожден, пока полностью не обработает пришедший запрос.

Если для запроса нет свободного потока в пуле для обработки входящего запроса, то запрос становится в очередь ожидания.

Очередь ожидания имеет свой предел (по умолчанию 1000 запросов), по достижению этого предела, пользователи начнут получать в ответ статус код 503 «Service Unavailable» или «Сервис недоступен».

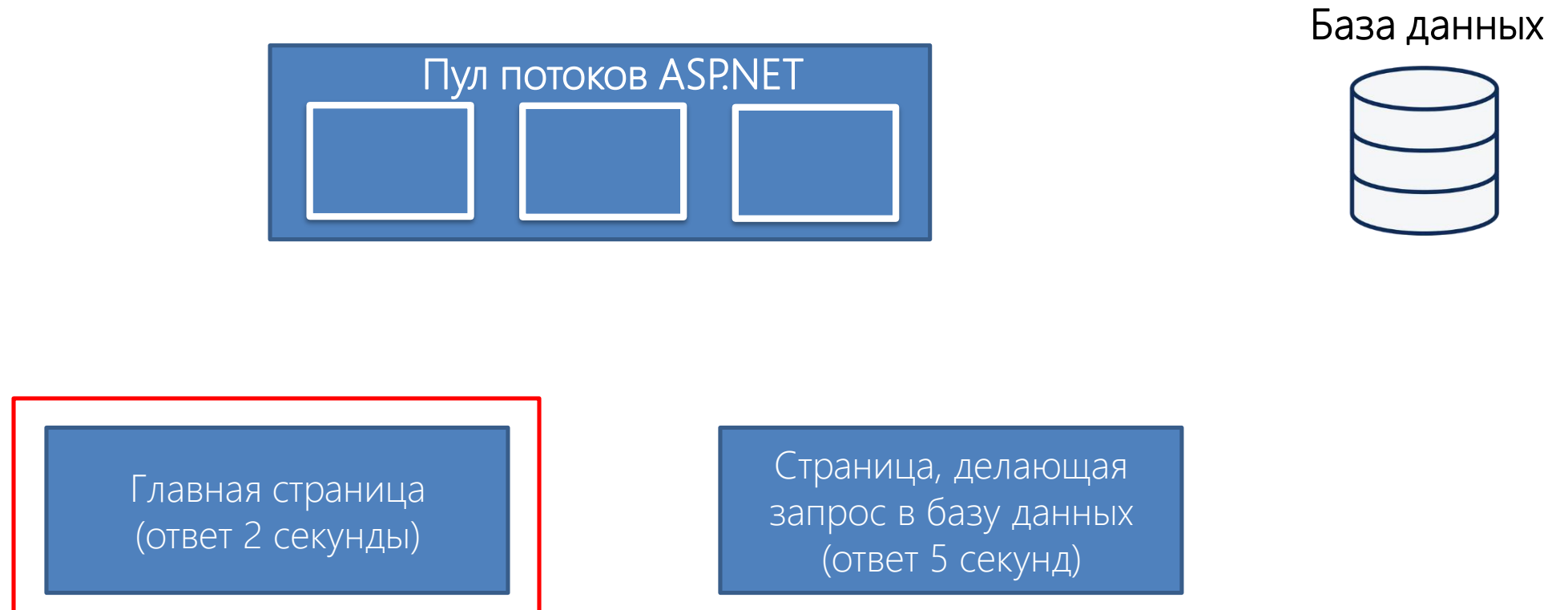
C# Асинхронное программирование

Синхронная обработка запросов в ASP.NET



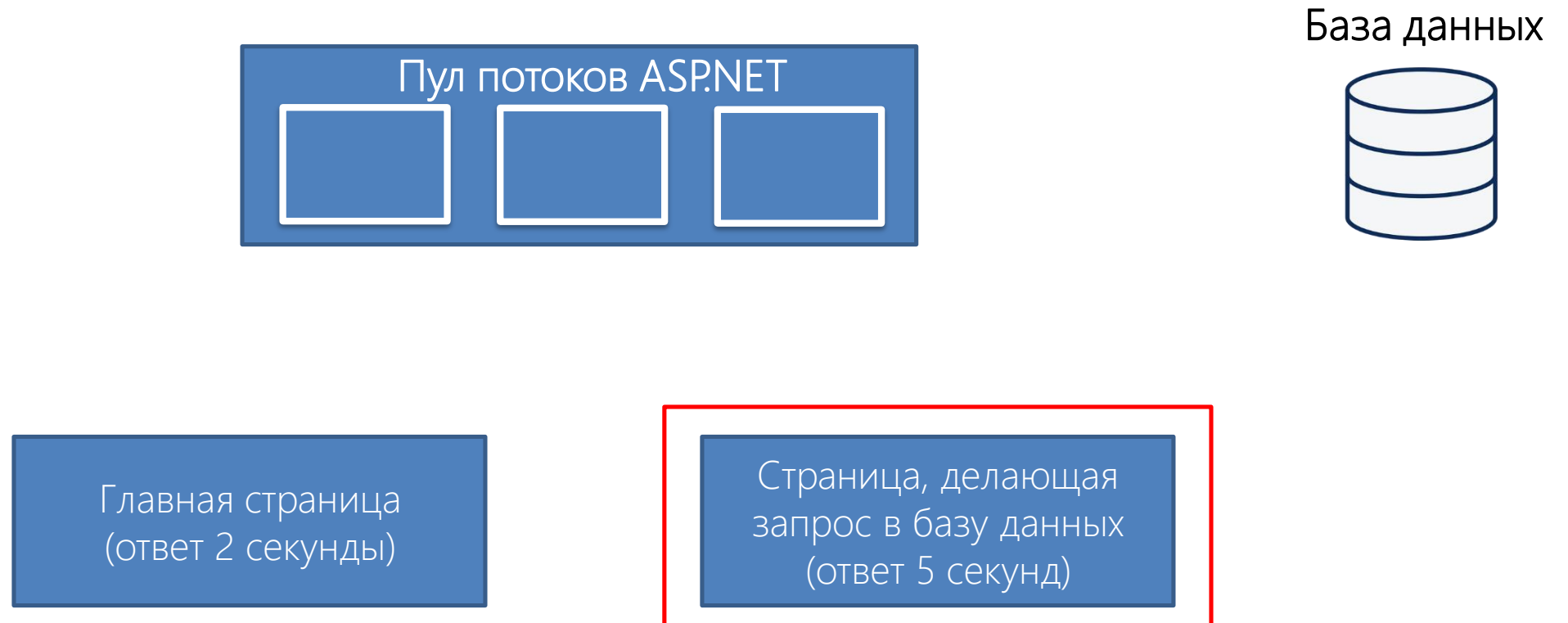
C# Асинхронное программирование

Синхронная обработка запросов в ASP.NET



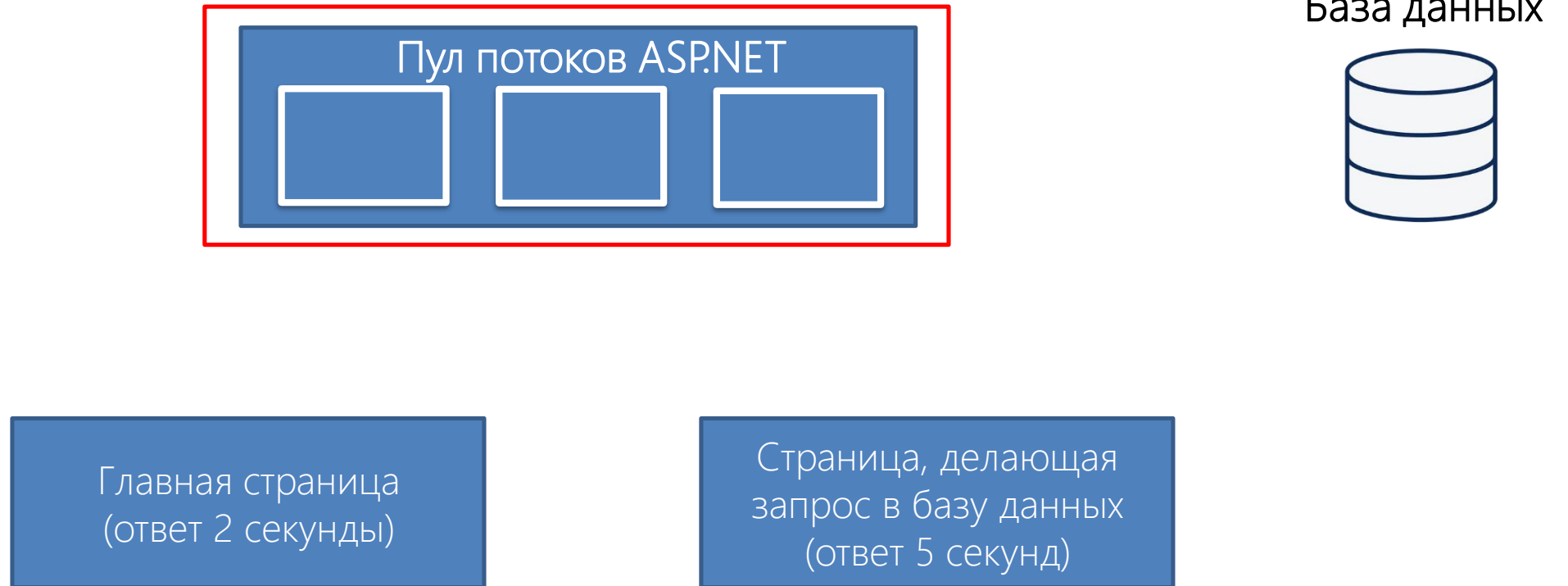
C# Асинхронное программирование

Синхронная обработка запросов в ASP.NET



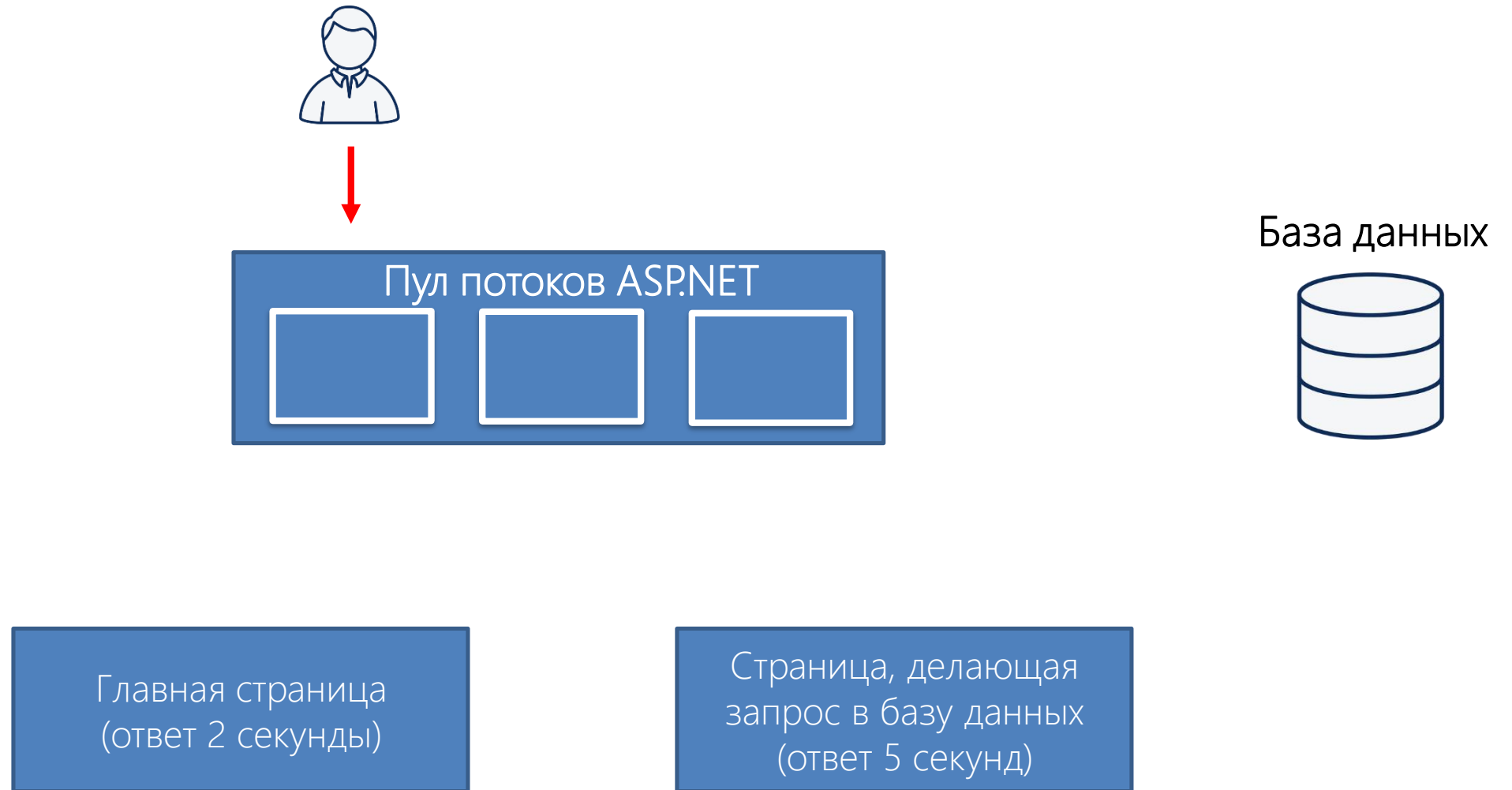
C# Асинхронное программирование

Синхронная обработка запросов в ASP.NET



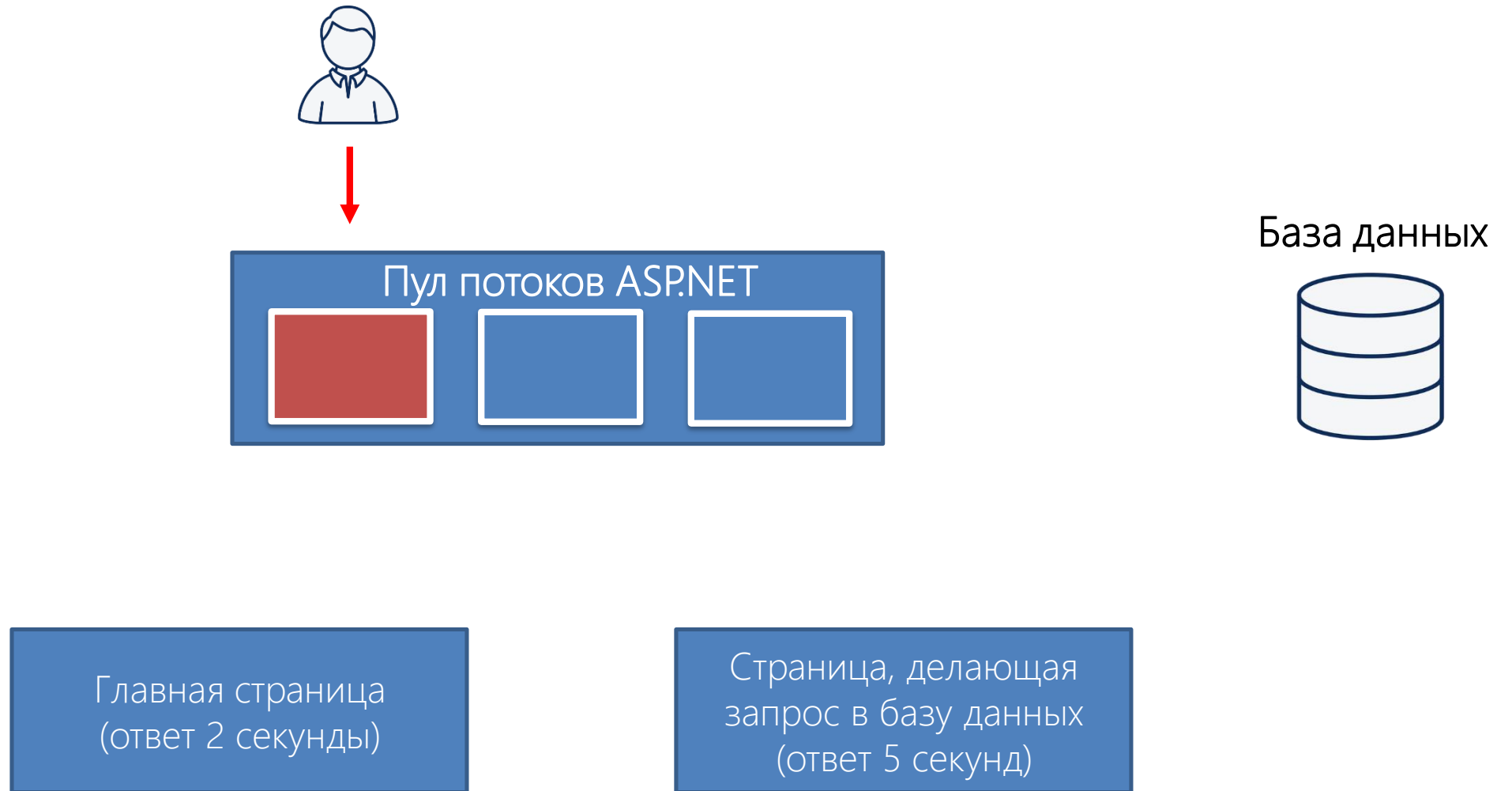
C# Асинхронное программирование

Синхронная обработка запросов в ASP.NET



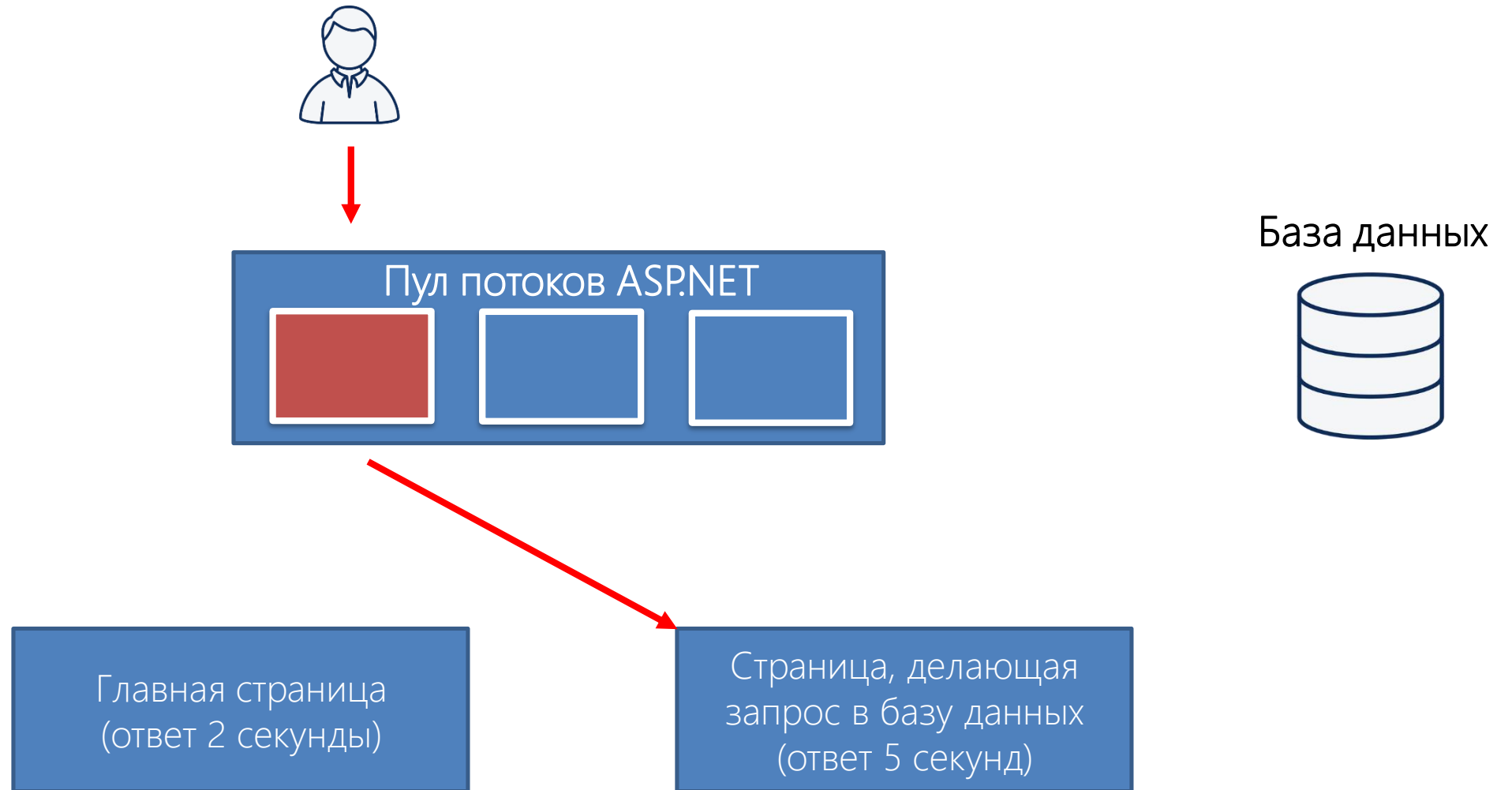
C# Асинхронное программирование

Синхронная обработка запросов в ASP.NET



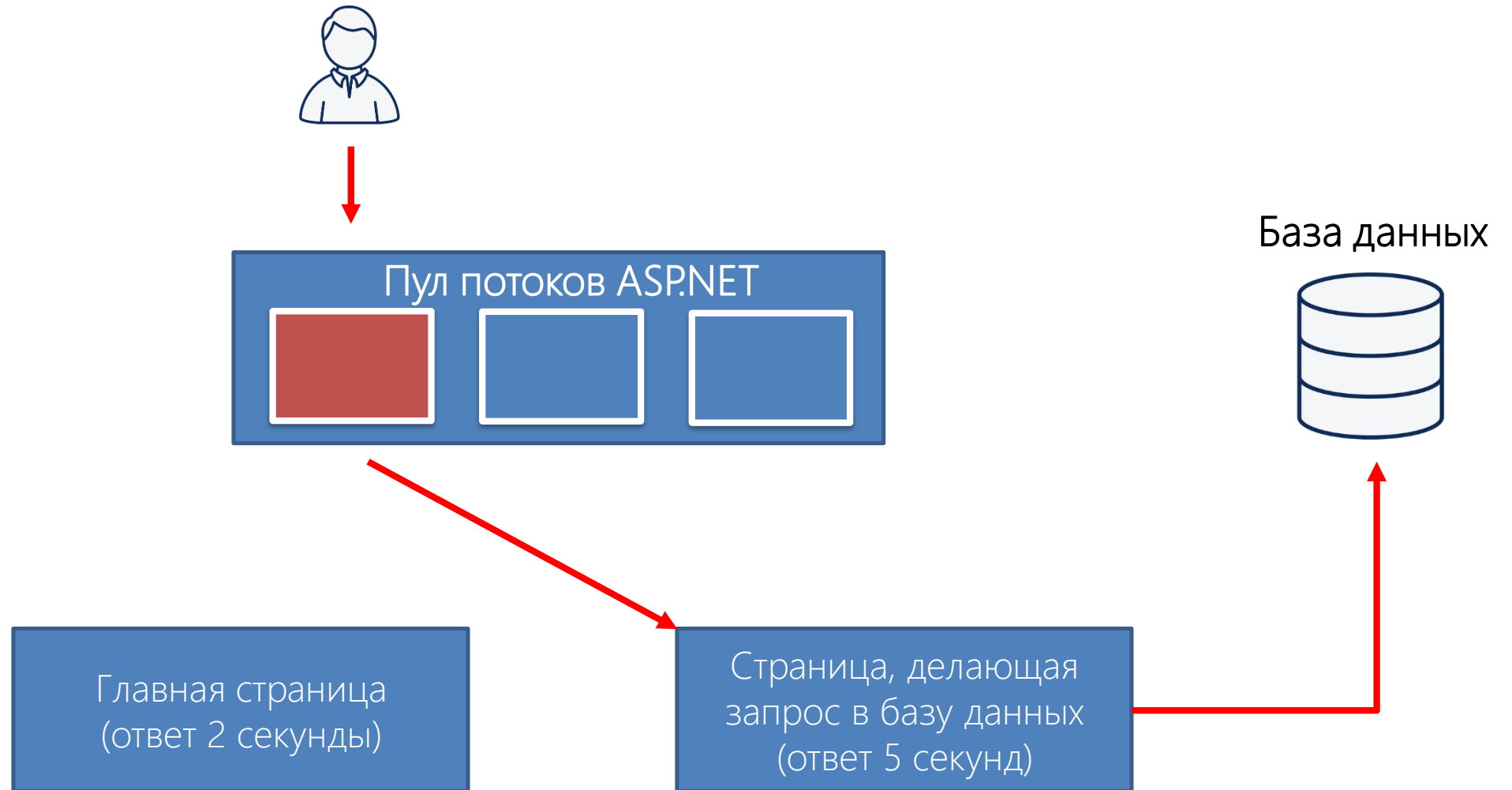
C# Асинхронное программирование

Синхронная обработка запросов в ASP.NET



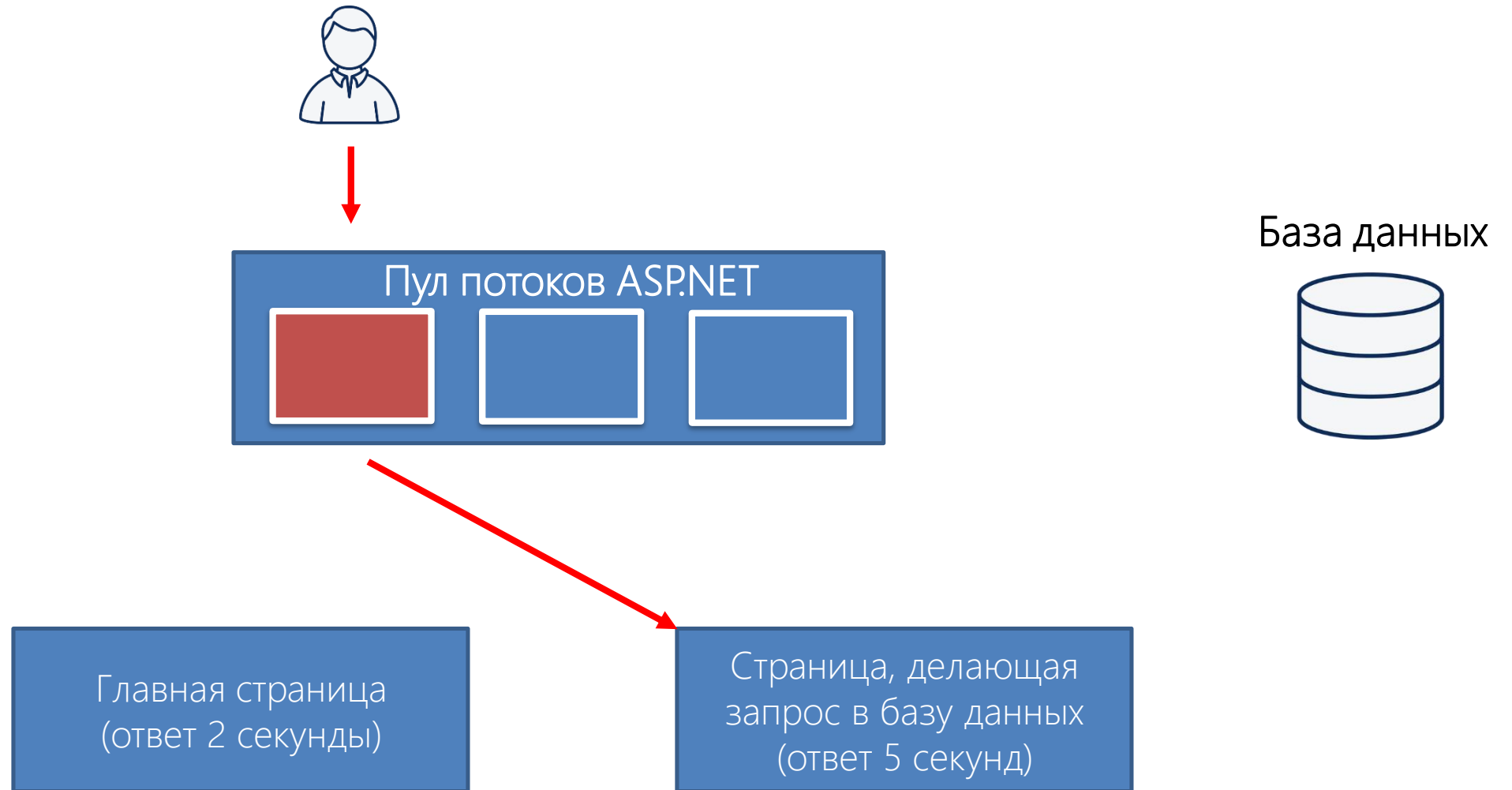
C# Асинхронное программирование

Синхронная обработка запросов в ASP.NET



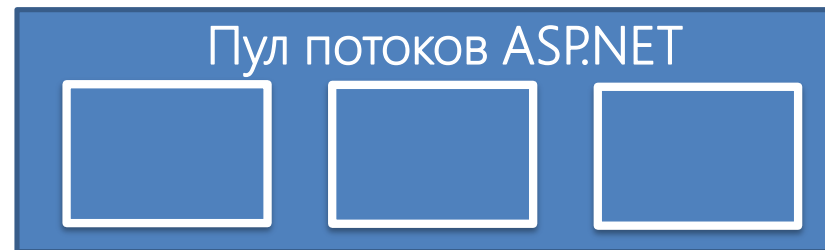
С# Асинхронное программирование

Синхронная обработка запросов в ASP.NET



C# Асинхронное программирование

Синхронная обработка запросов в ASP.NET



База данных

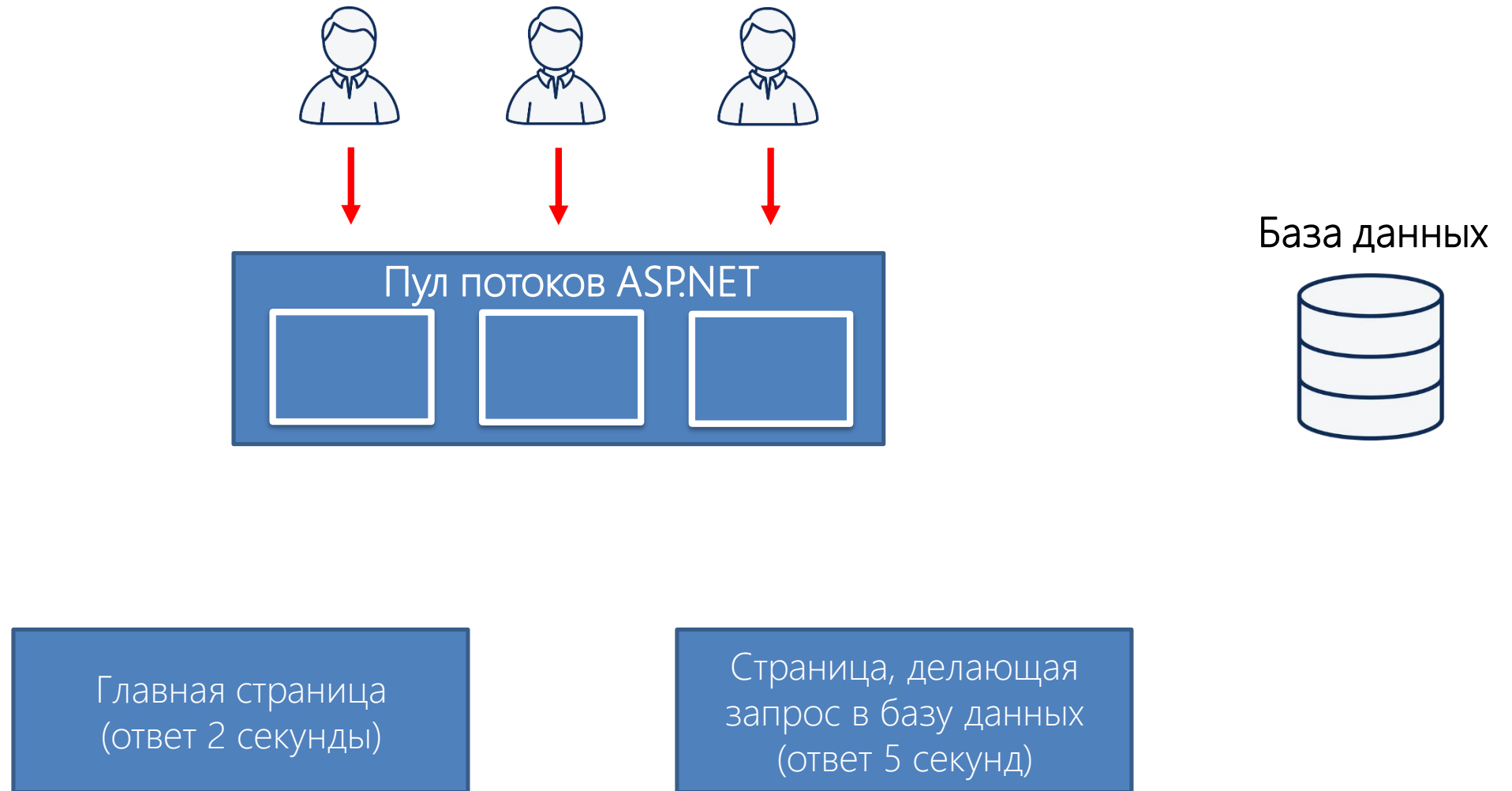


Главная страница
(ответ 2 секунды)

Страница, делающая
запрос в базу данных
(ответ 5 секунд)

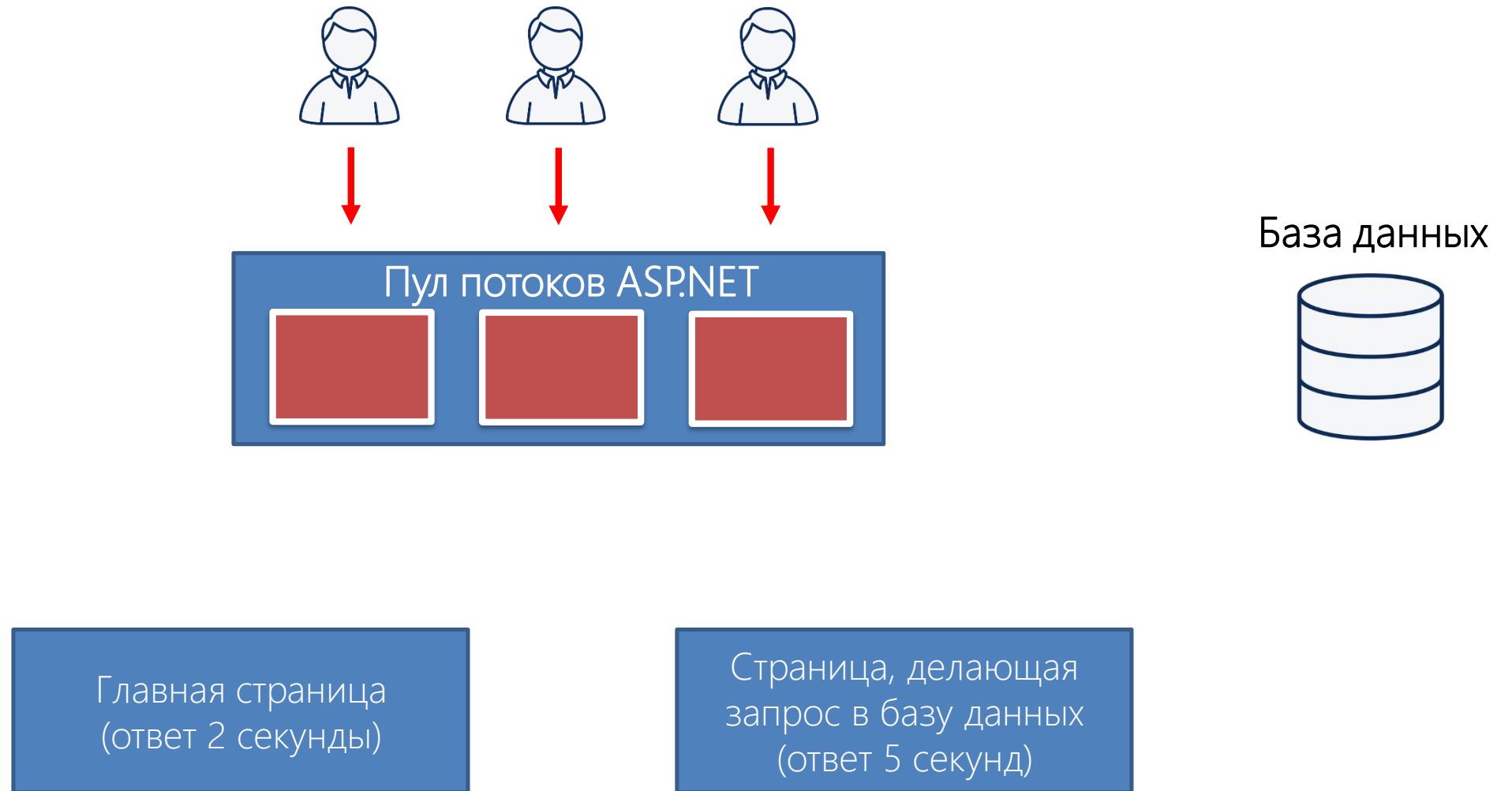
C# Асинхронное программирование

Синхронная обработка запросов в ASP.NET



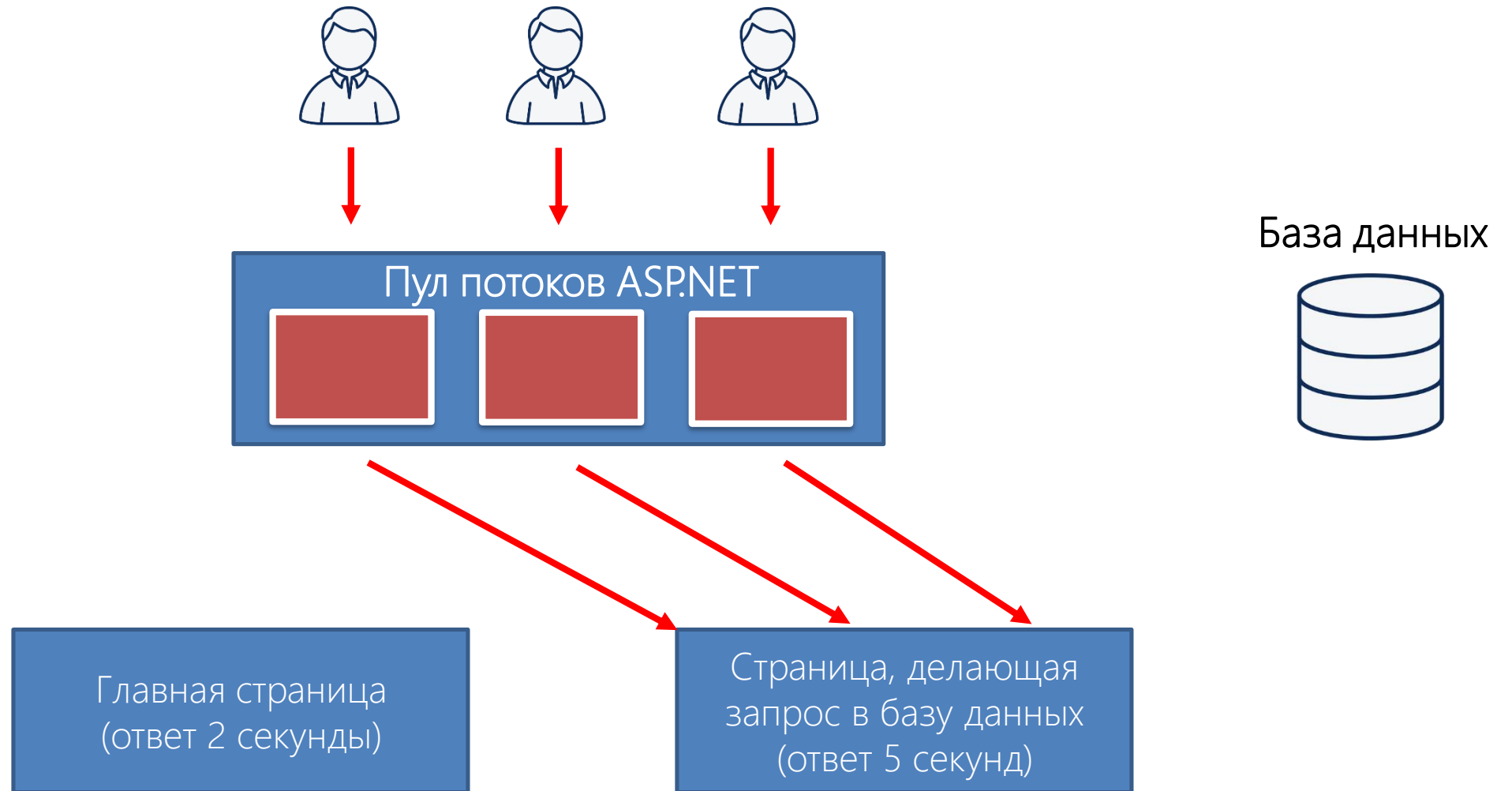
C# Асинхронное программирование

Синхронная обработка запросов в ASP.NET



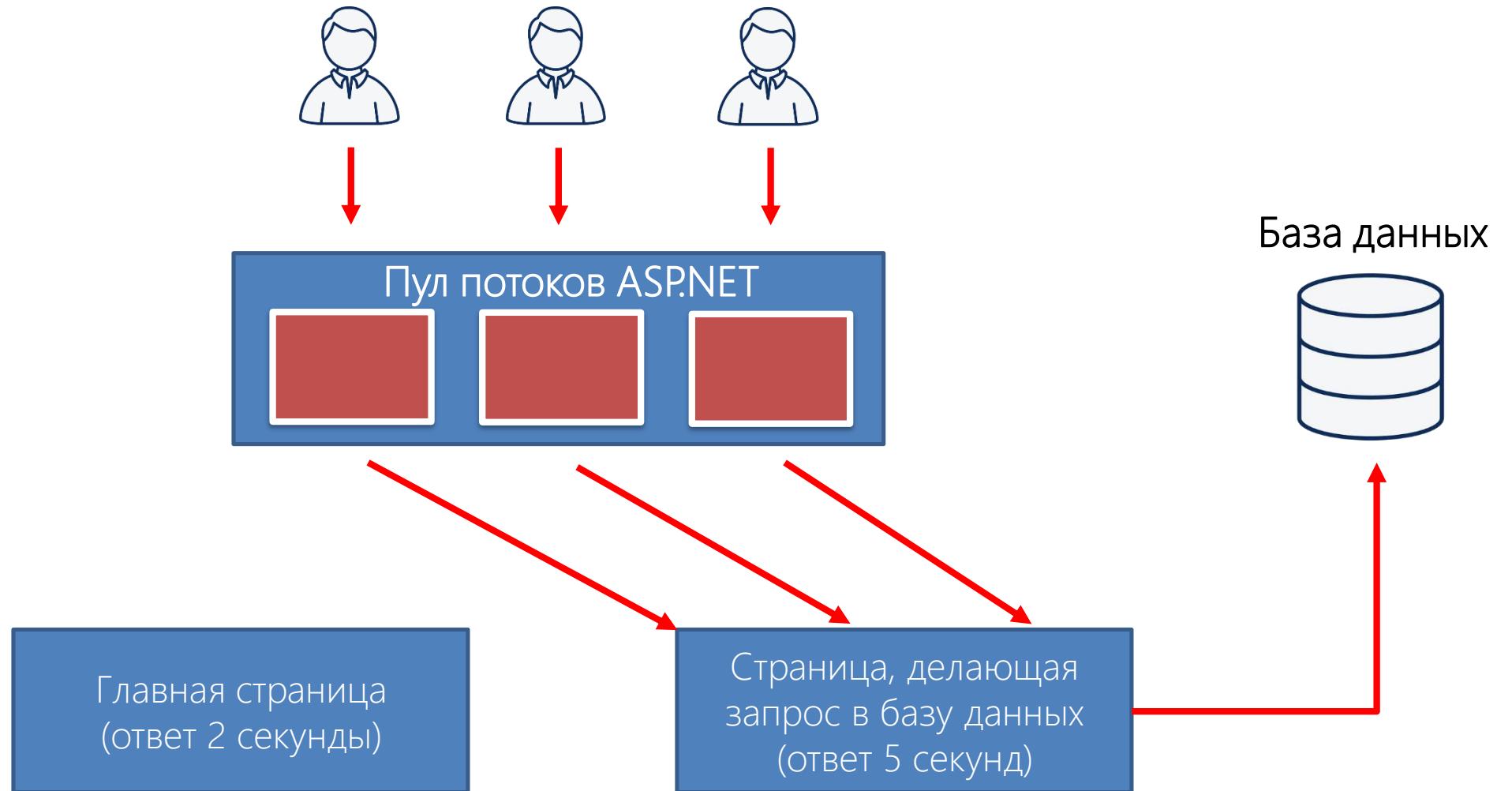
C# Асинхронное программирование

Синхронная обработка запросов в ASP.NET



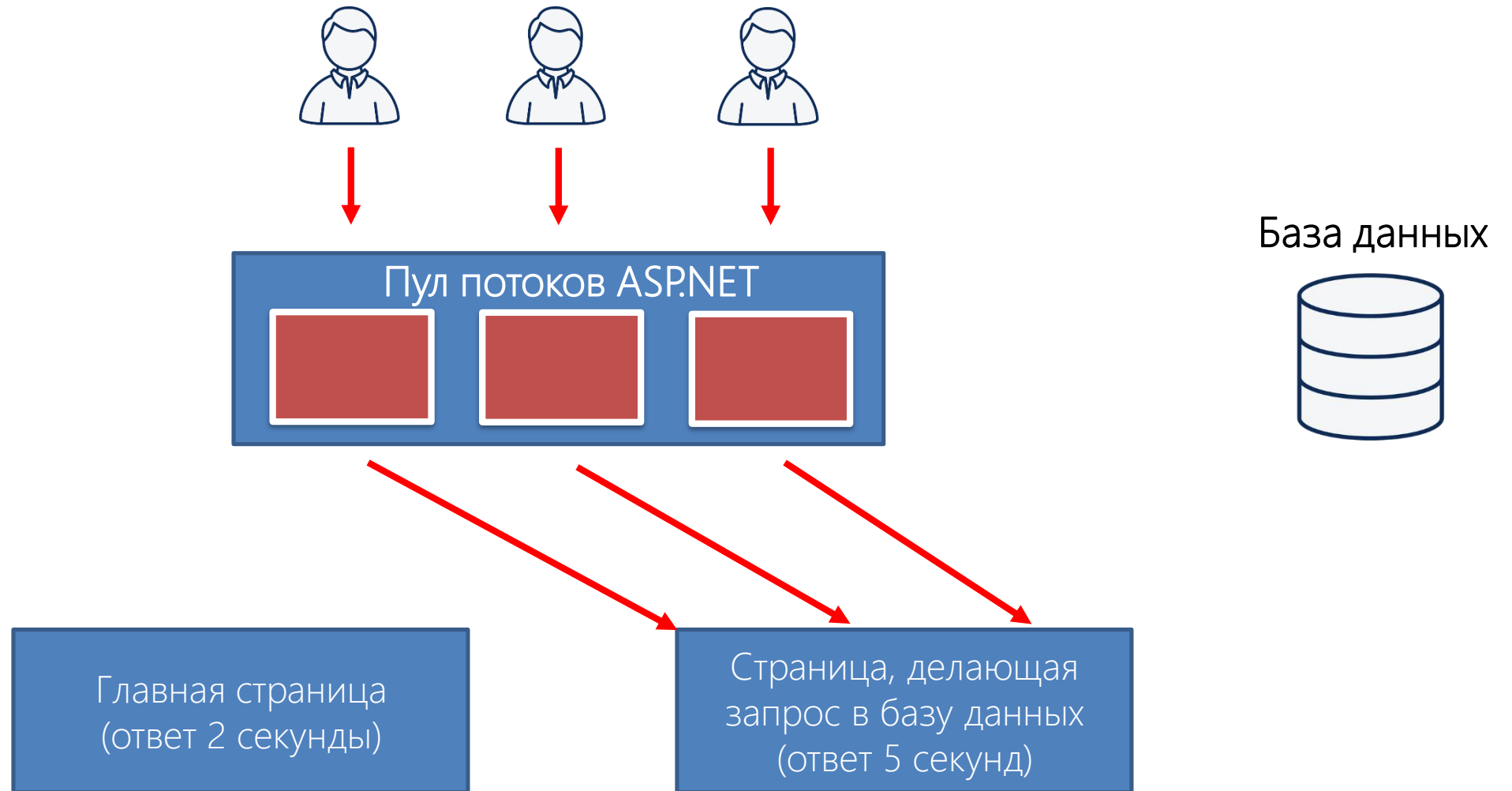
С# Асинхронное программирование

Синхронная обработка запросов в ASP.NET



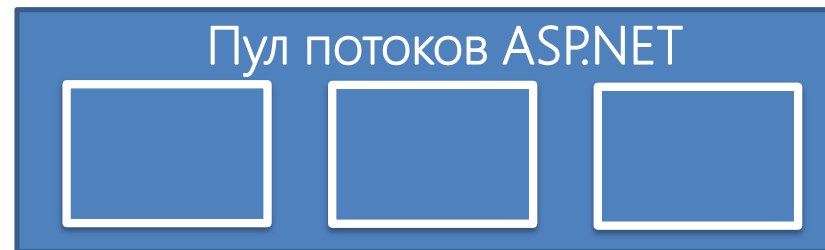
C# Асинхронное программирование

Синхронная обработка запросов в ASP.NET



C# Асинхронное программирование

Синхронная обработка запросов в ASP.NET



База данных

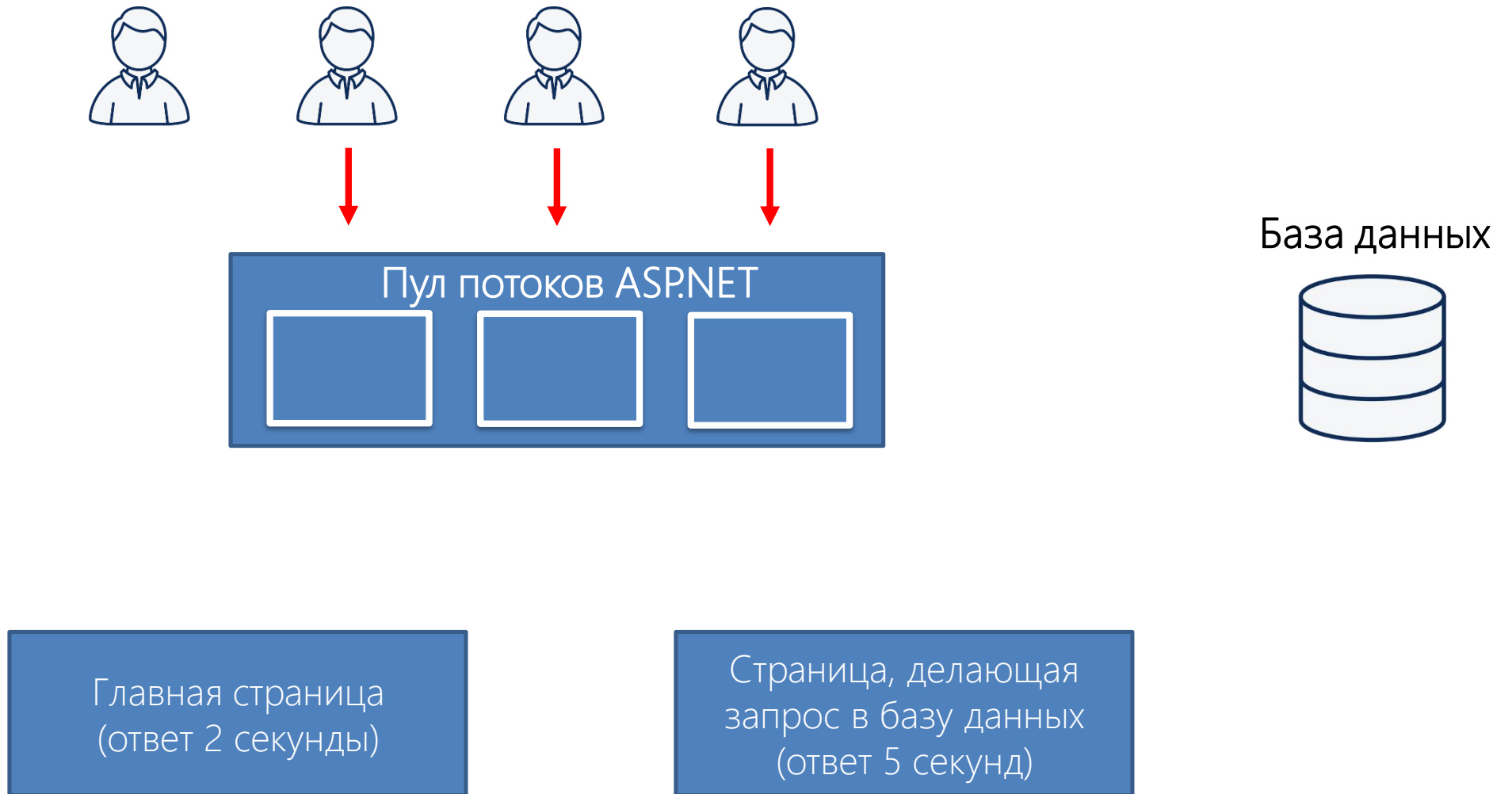


Главная страница
(ответ 2 секунды)

Страница, делающая
запрос в базу данных
(ответ 5 секунд)

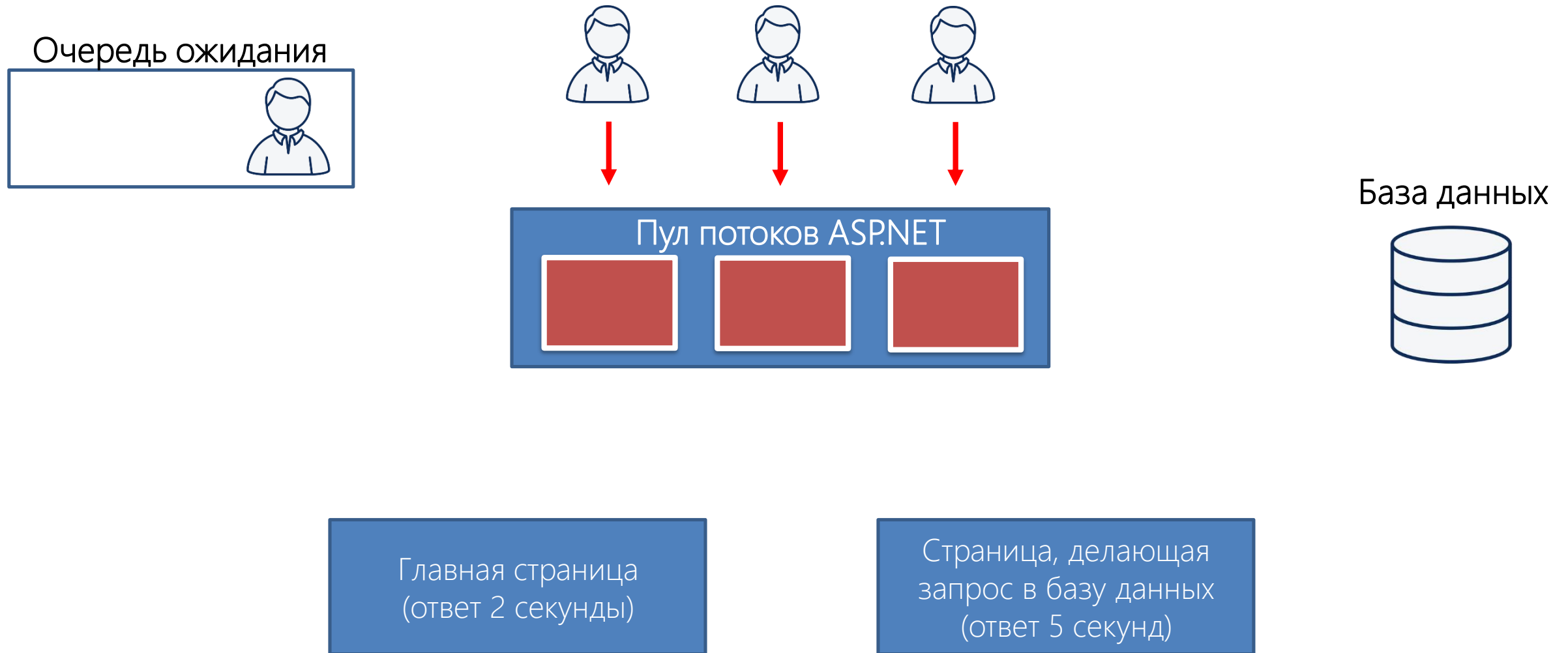
C# Асинхронное программирование

Синхронная обработка запросов в ASP.NET



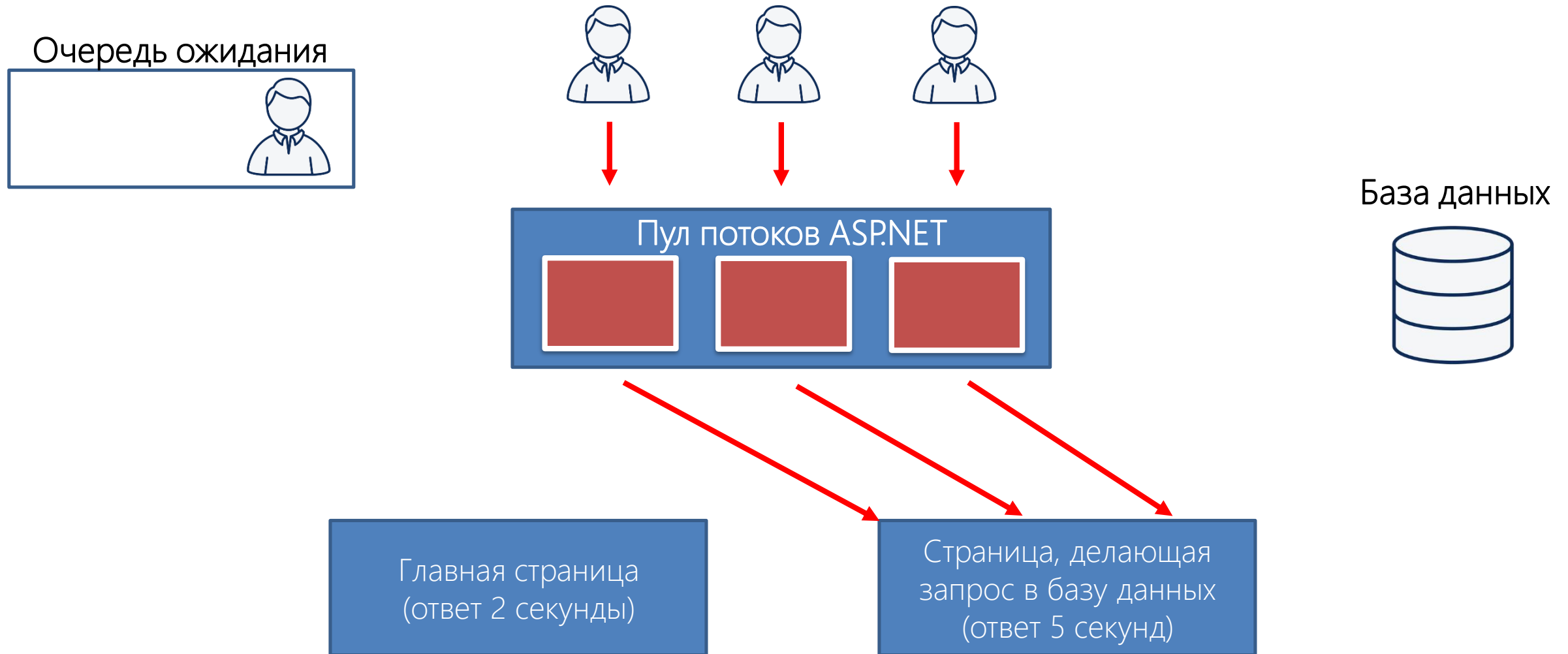
C# Асинхронное программирование

Синхронная обработка запросов в ASP.NET



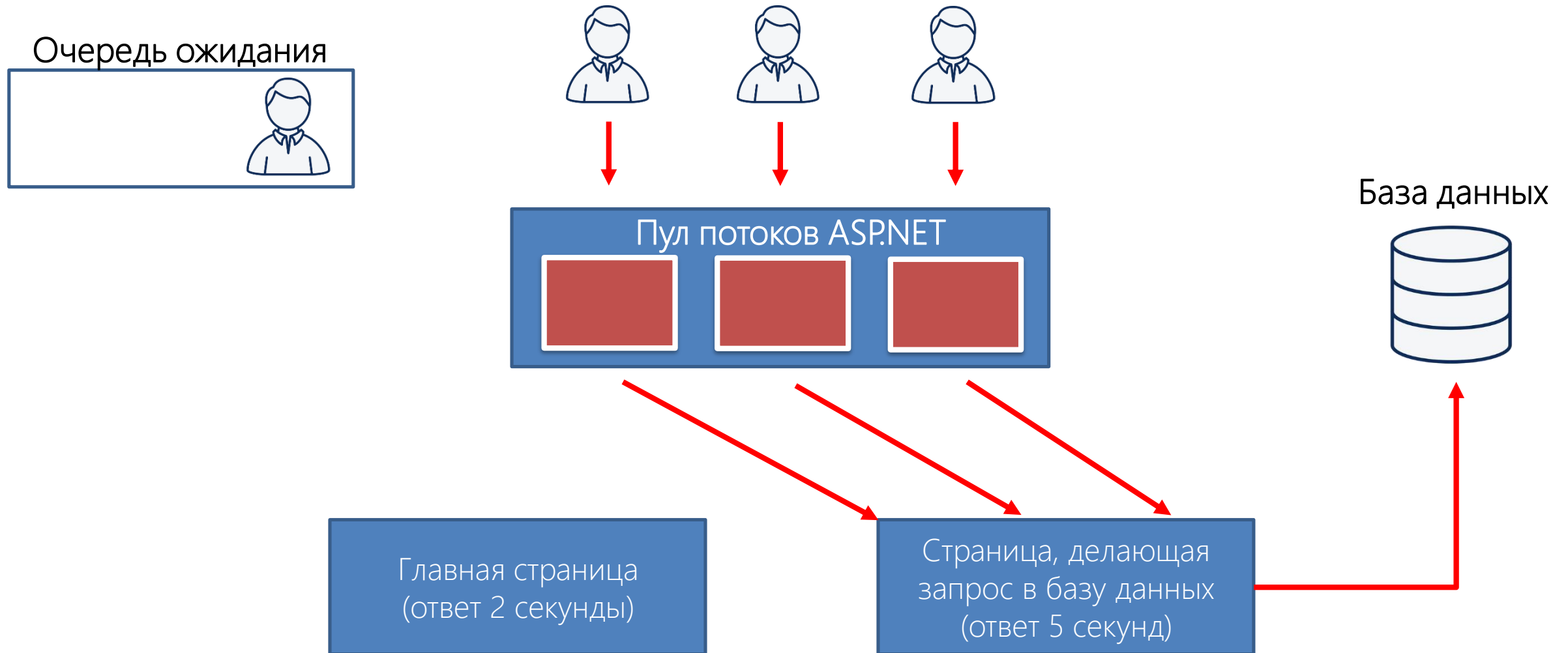
C# Асинхронное программирование

Синхронная обработка запросов в ASP.NET



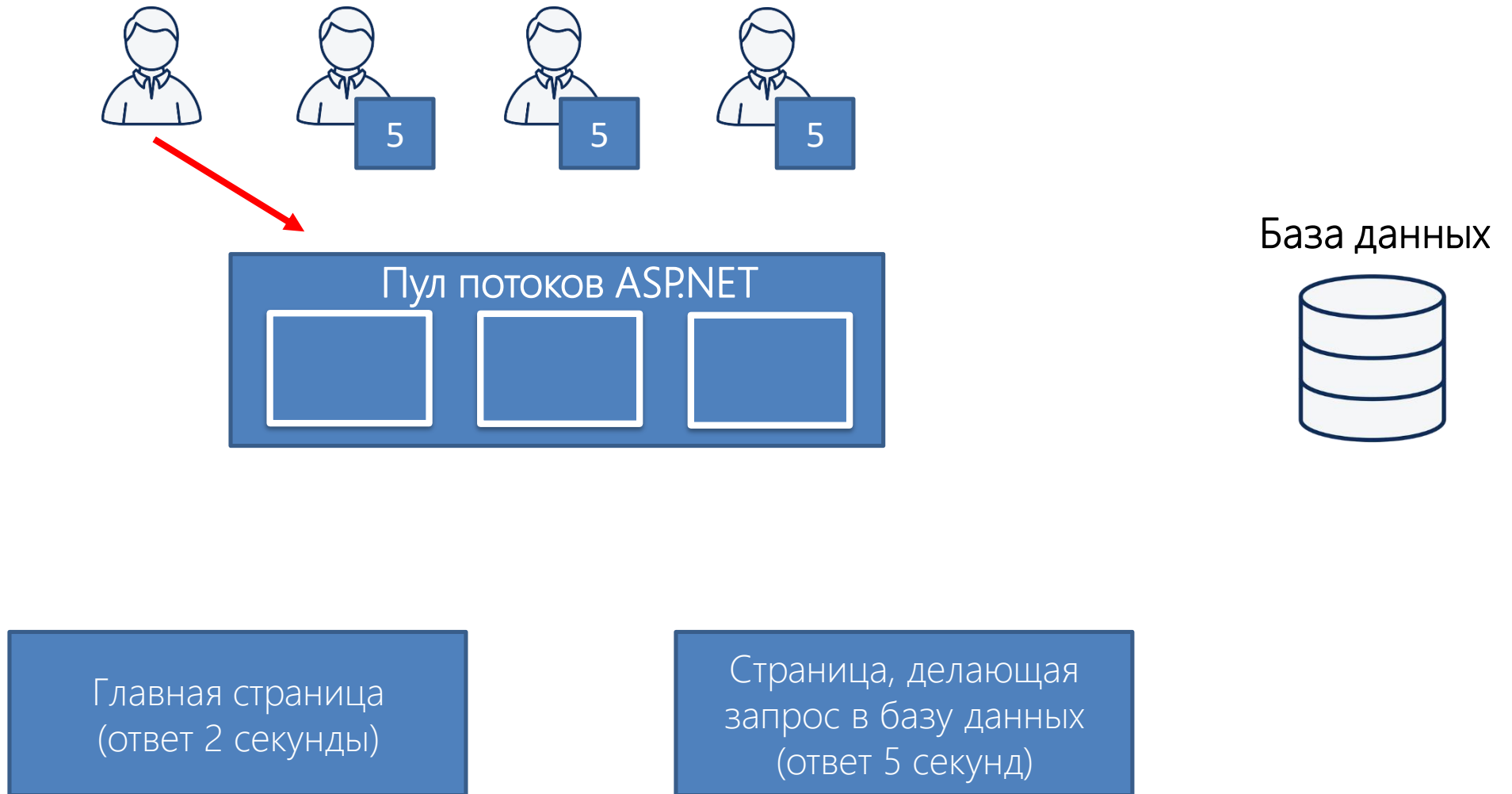
C# Асинхронное программирование

Синхронная обработка запросов в ASP.NET



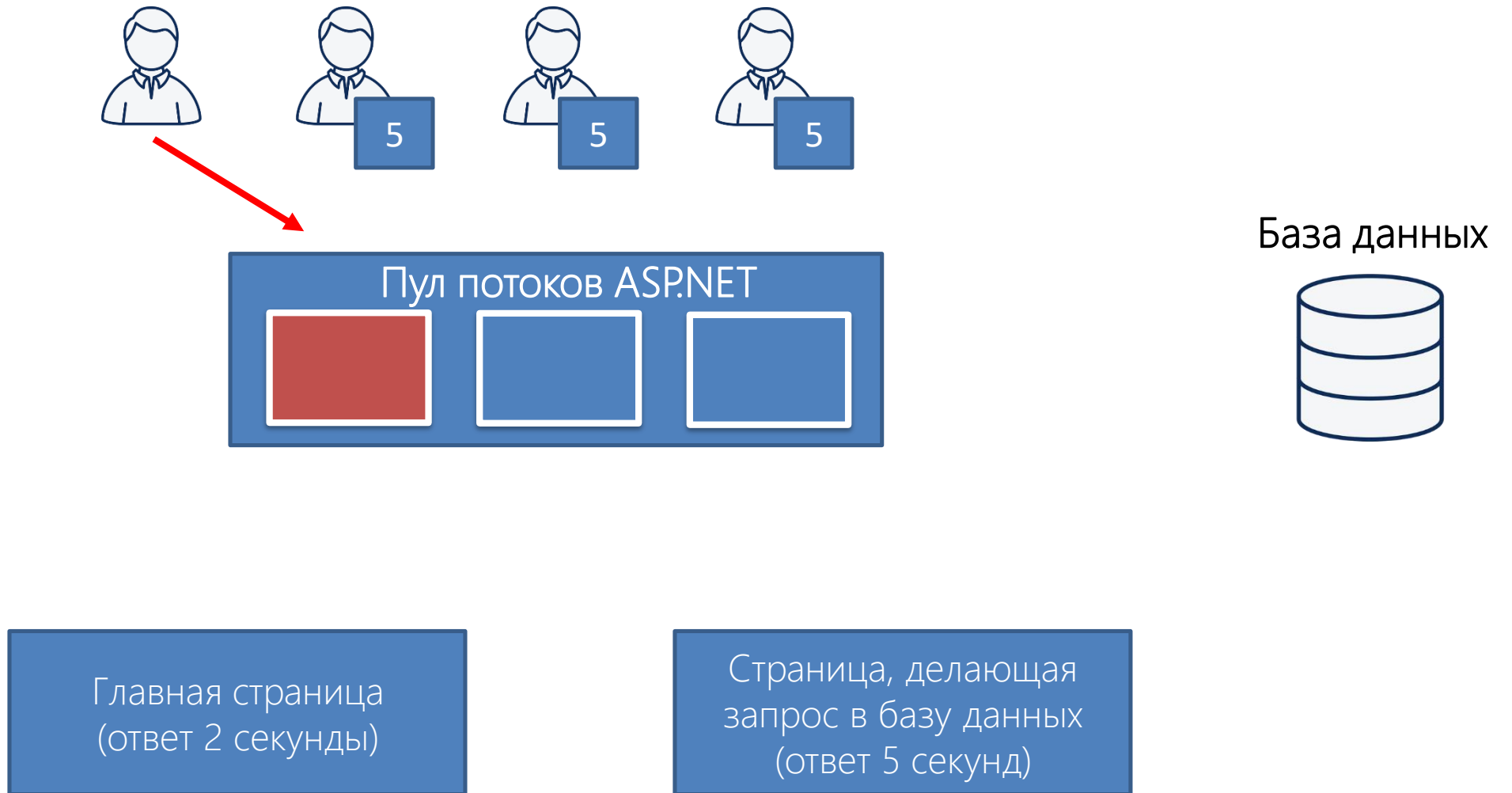
C# Асинхронное программирование

Синхронная обработка запросов в ASP.NET



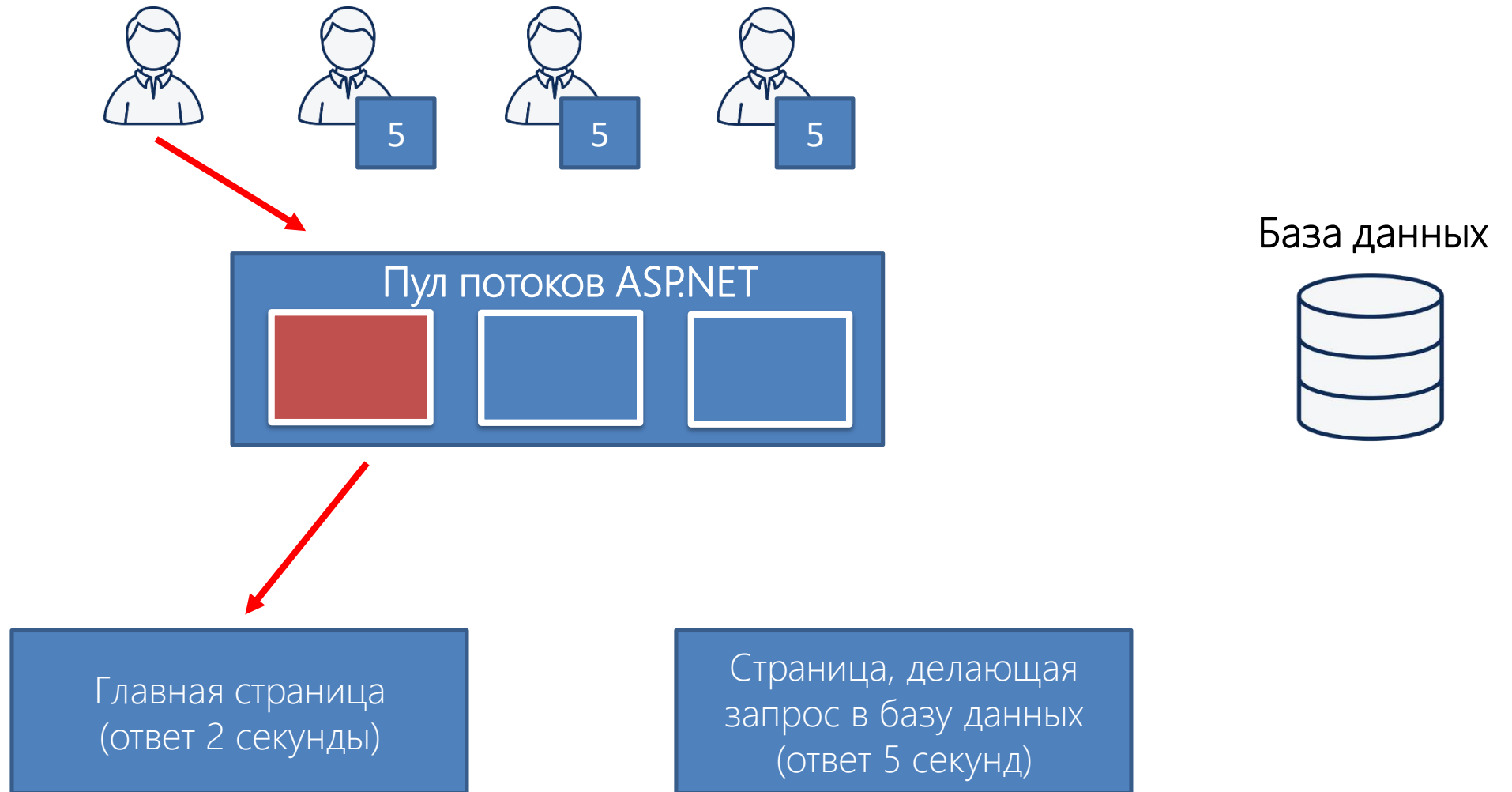
С# Асинхронное программирование

Синхронная обработка запросов в ASP.NET



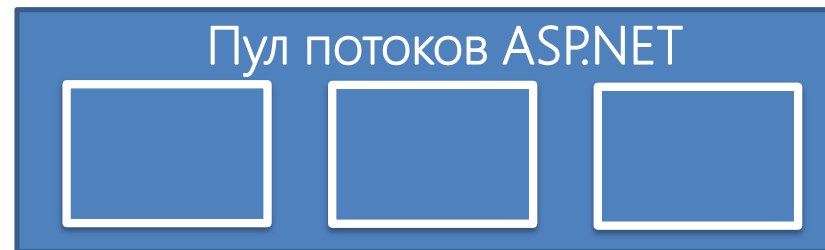
С# Асинхронное программирование

Синхронная обработка запросов в ASP.NET



C# Асинхронное программирование

Синхронная обработка запросов в ASP.NET



База данных

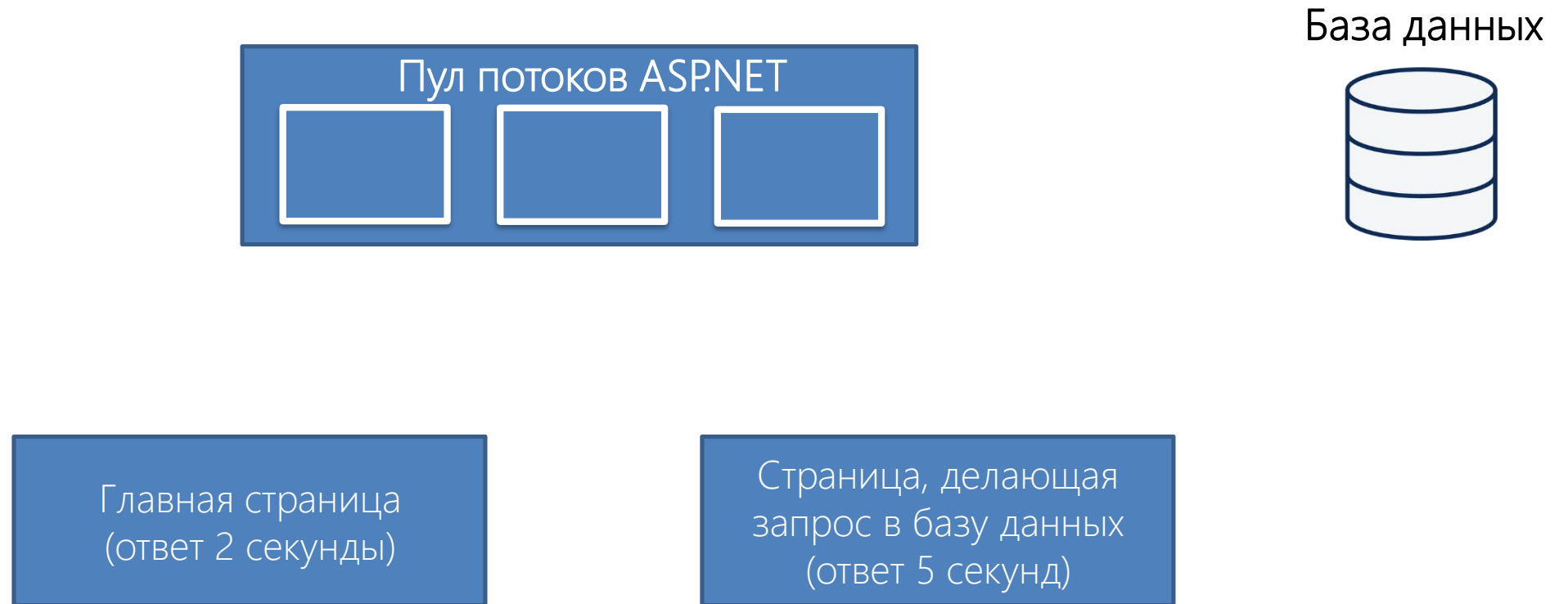


Главная страница
(ответ 2 секунды)

Страница, делающая
запрос в базу данных
(ответ 5 секунд)

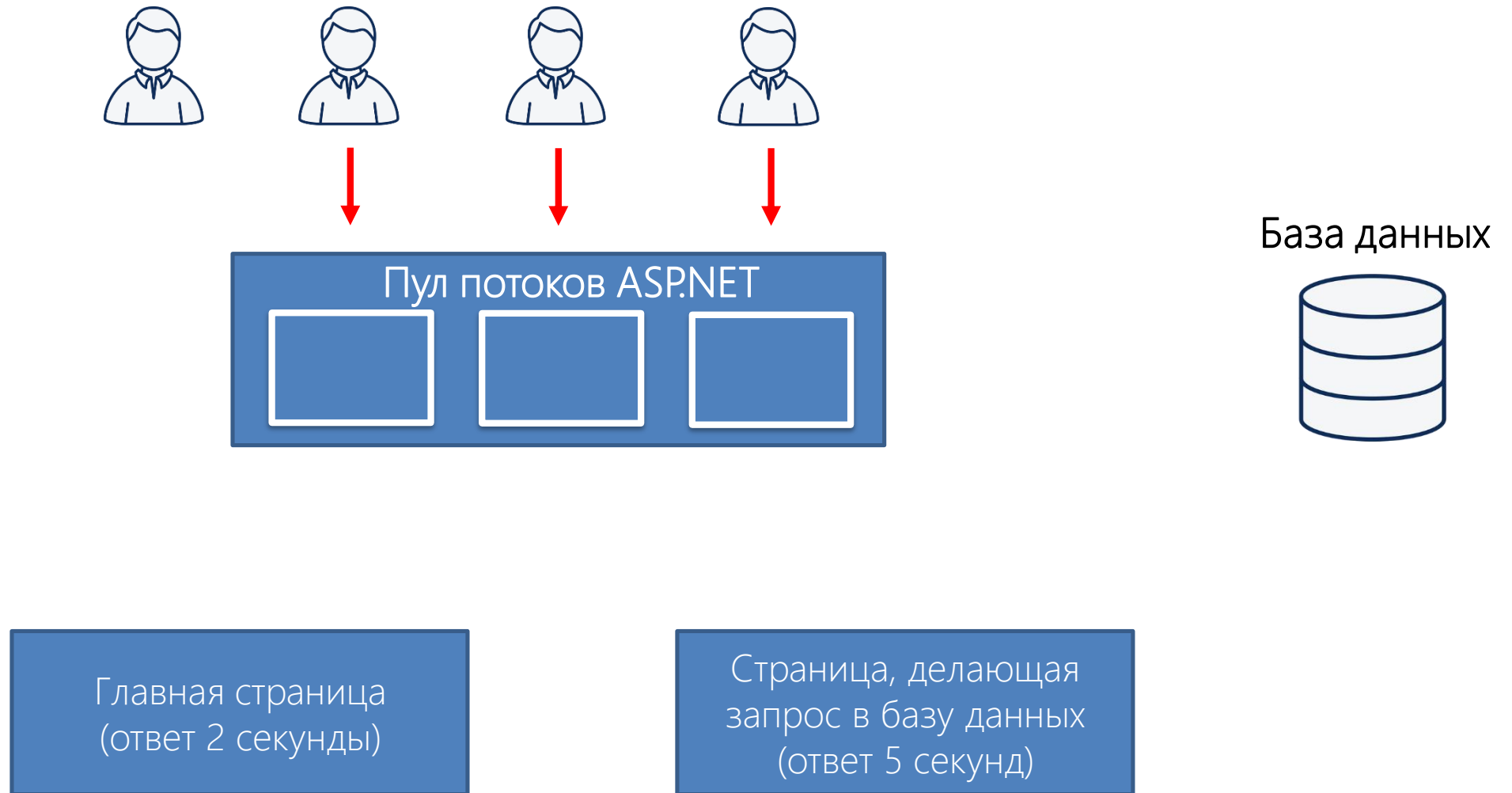
C# Асинхронное программирование

Асинхронная обработка запросов в ASP.NET



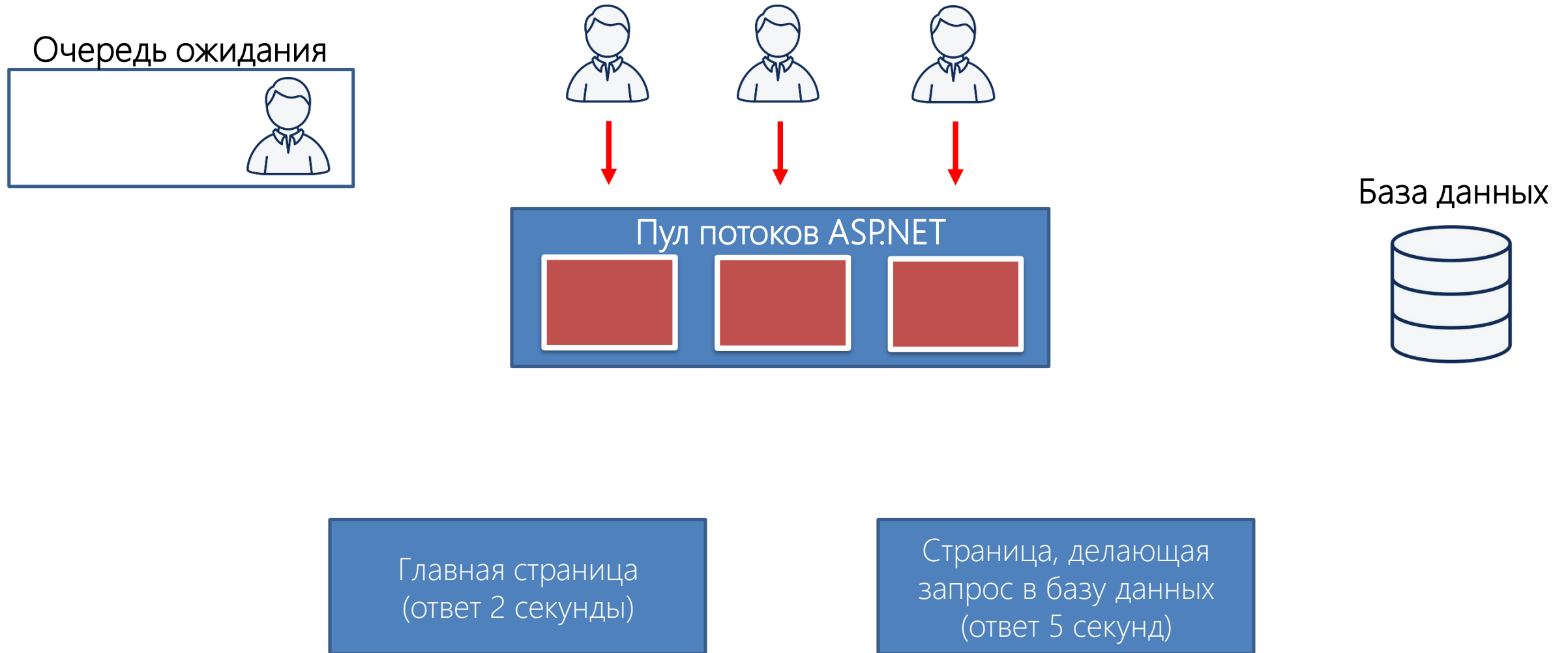
C# Асинхронное программирование

Асинхронная обработка запросов в ASP.NET



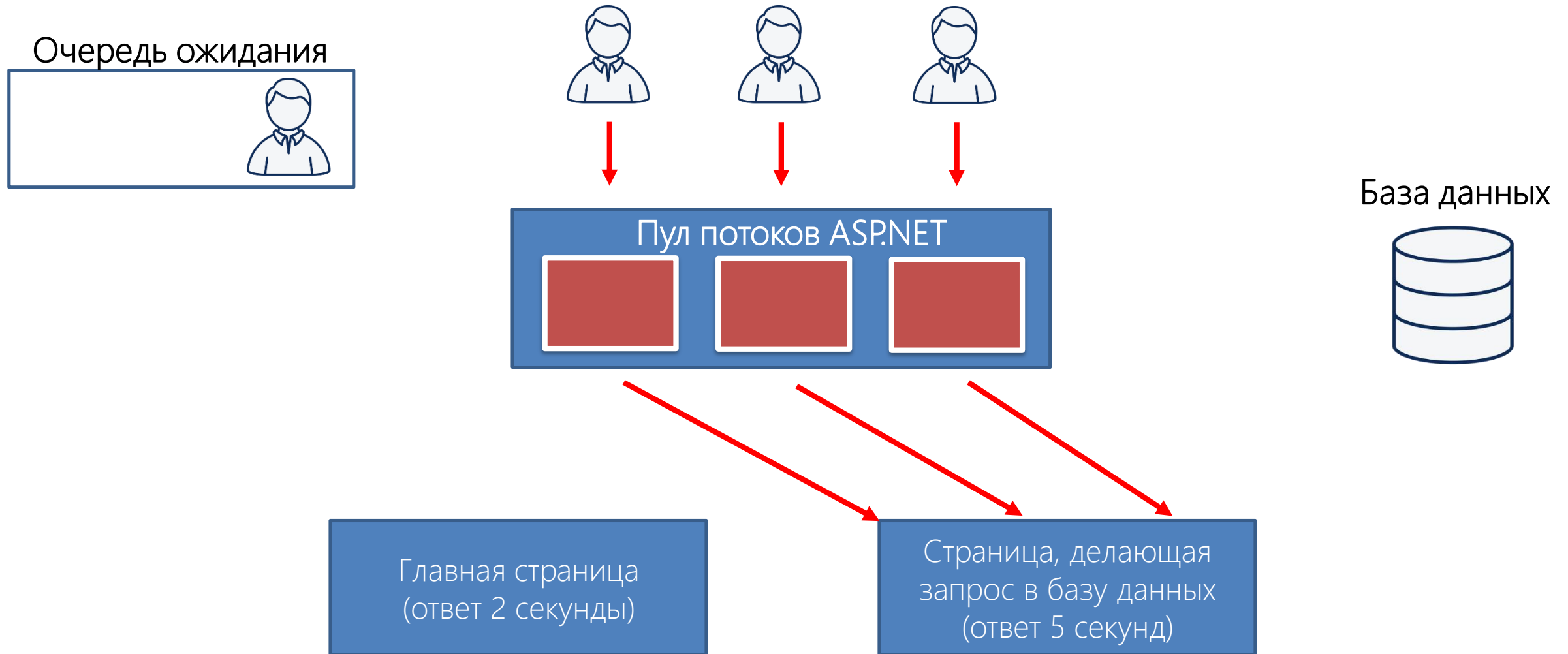
C# Асинхронное программирование

Асинхронная обработка запросов в ASP.NET



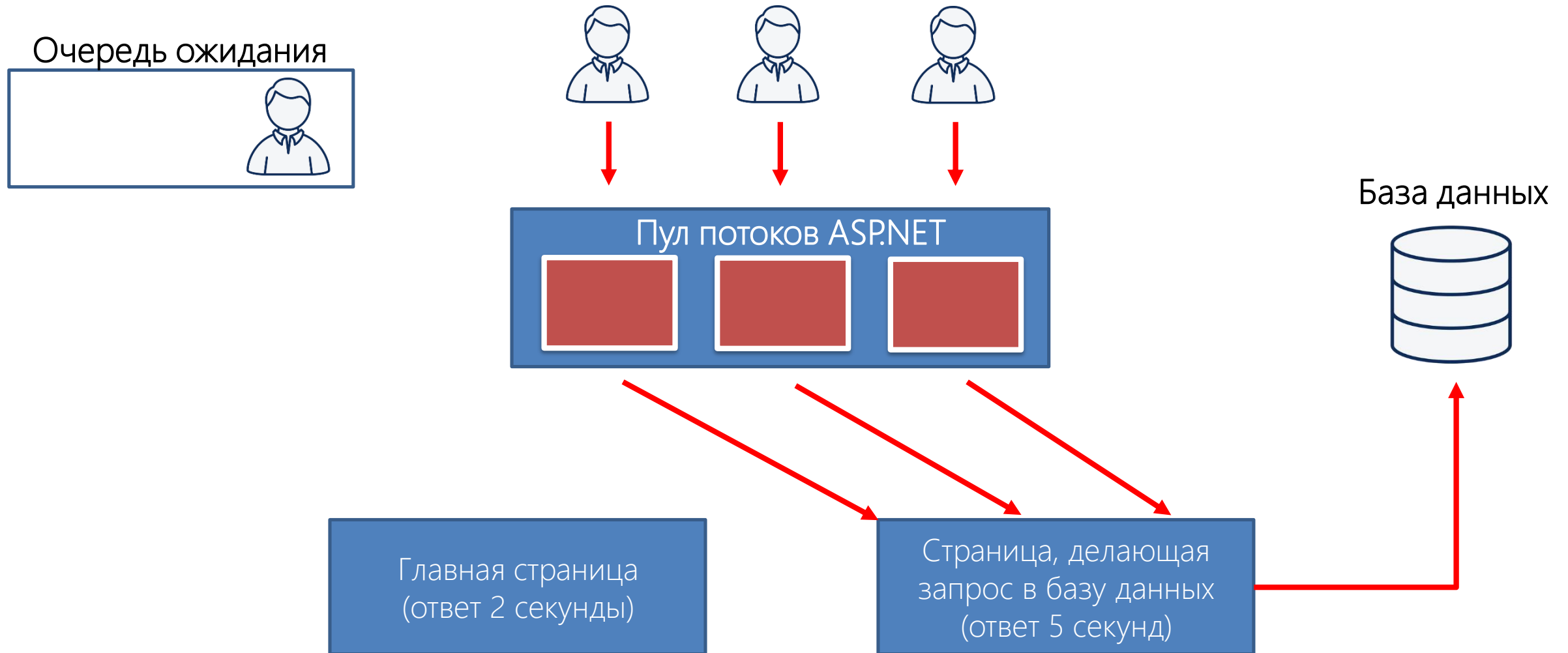
C# Асинхронное программирование

Асинхронная обработка запросов в ASP.NET



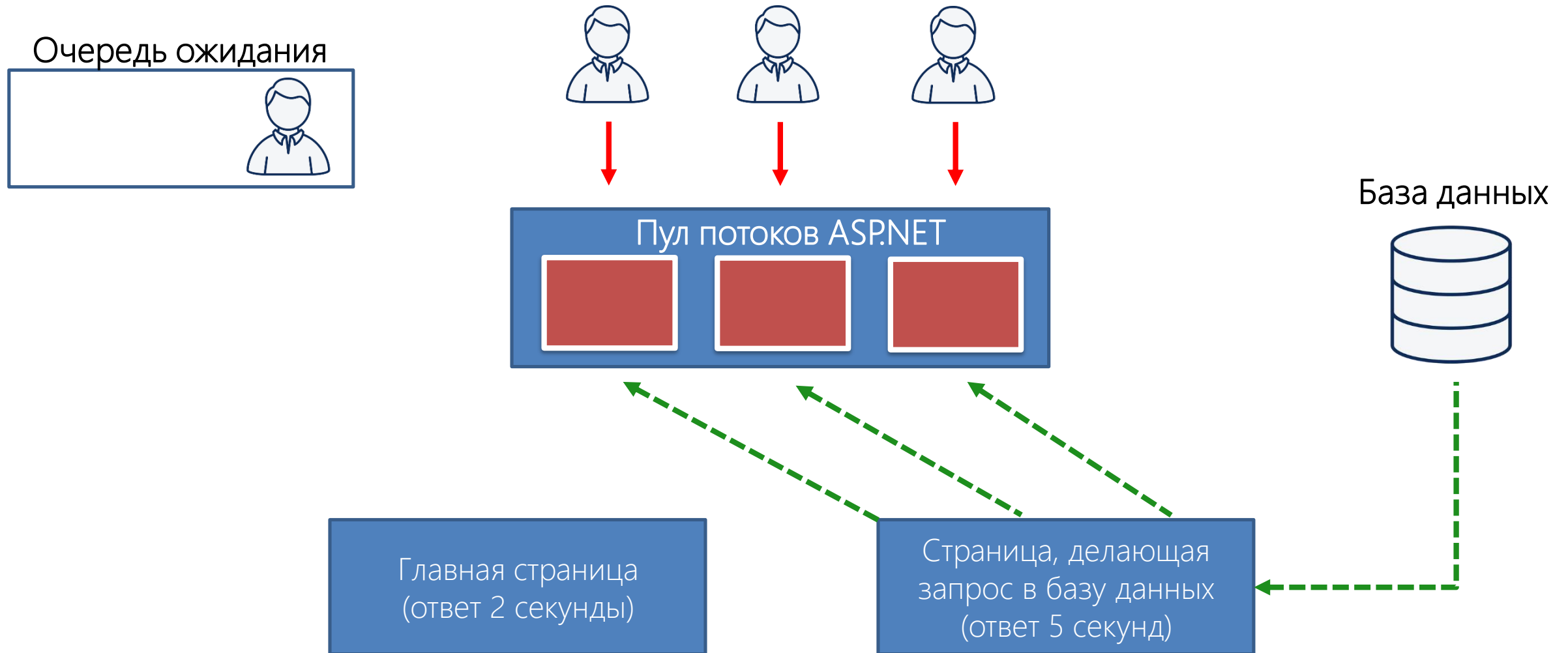
C# Асинхронное программирование

Асинхронная обработка запросов в ASP.NET



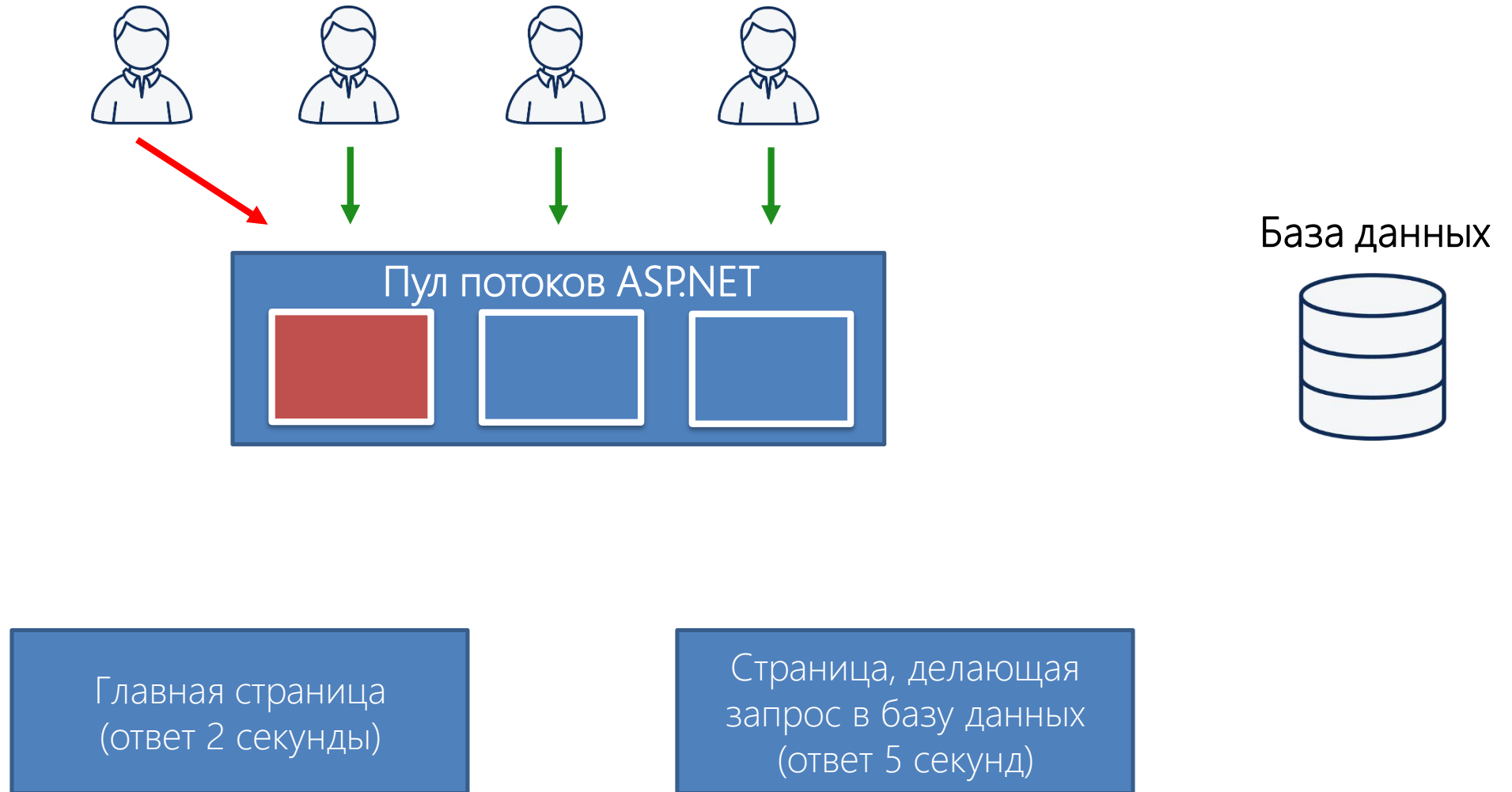
C# Асинхронное программирование

Асинхронная обработка запросов в ASP.NET



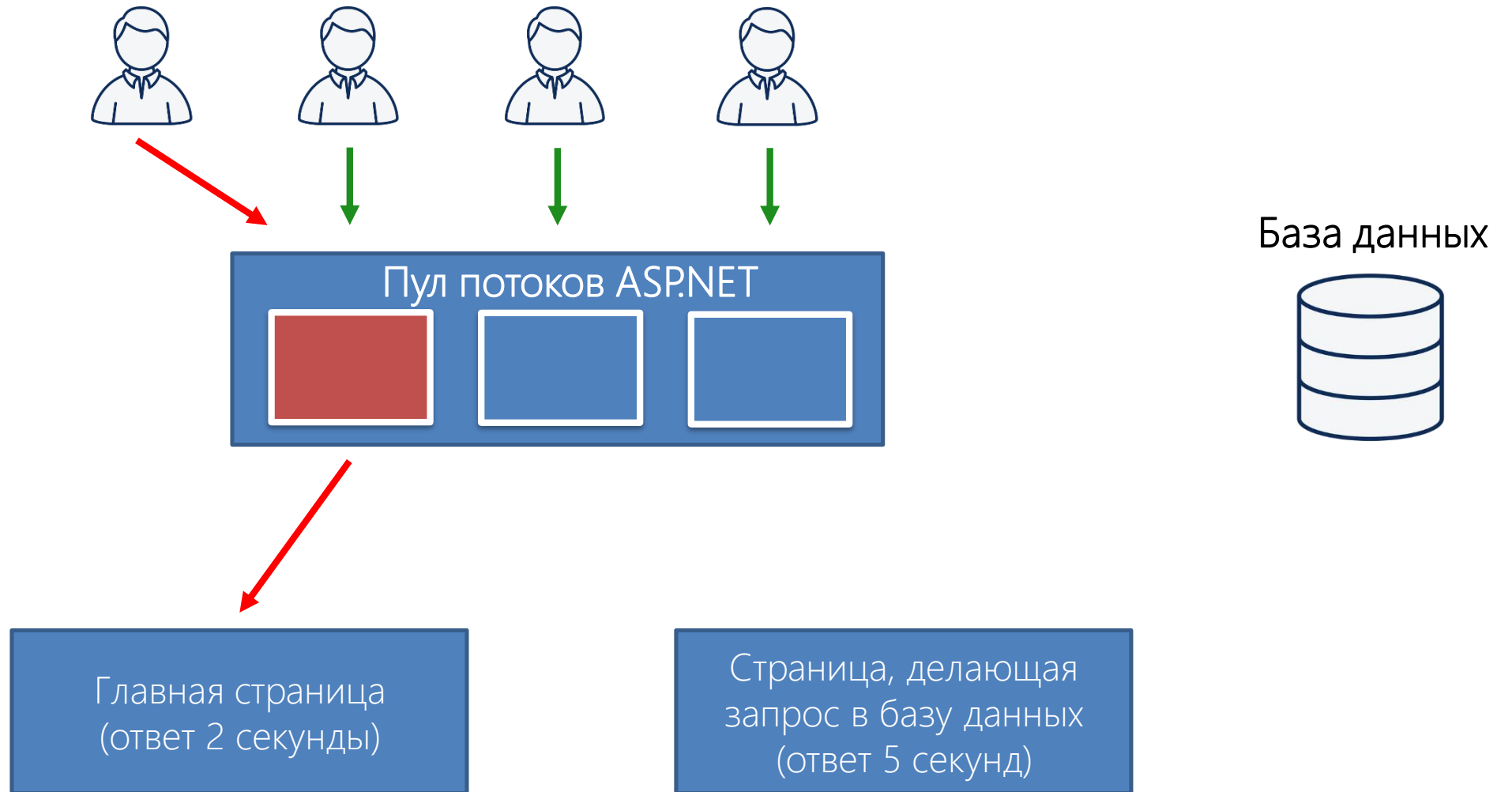
С# Асинхронное программирование

Асинхронная обработка запросов в ASP.NET



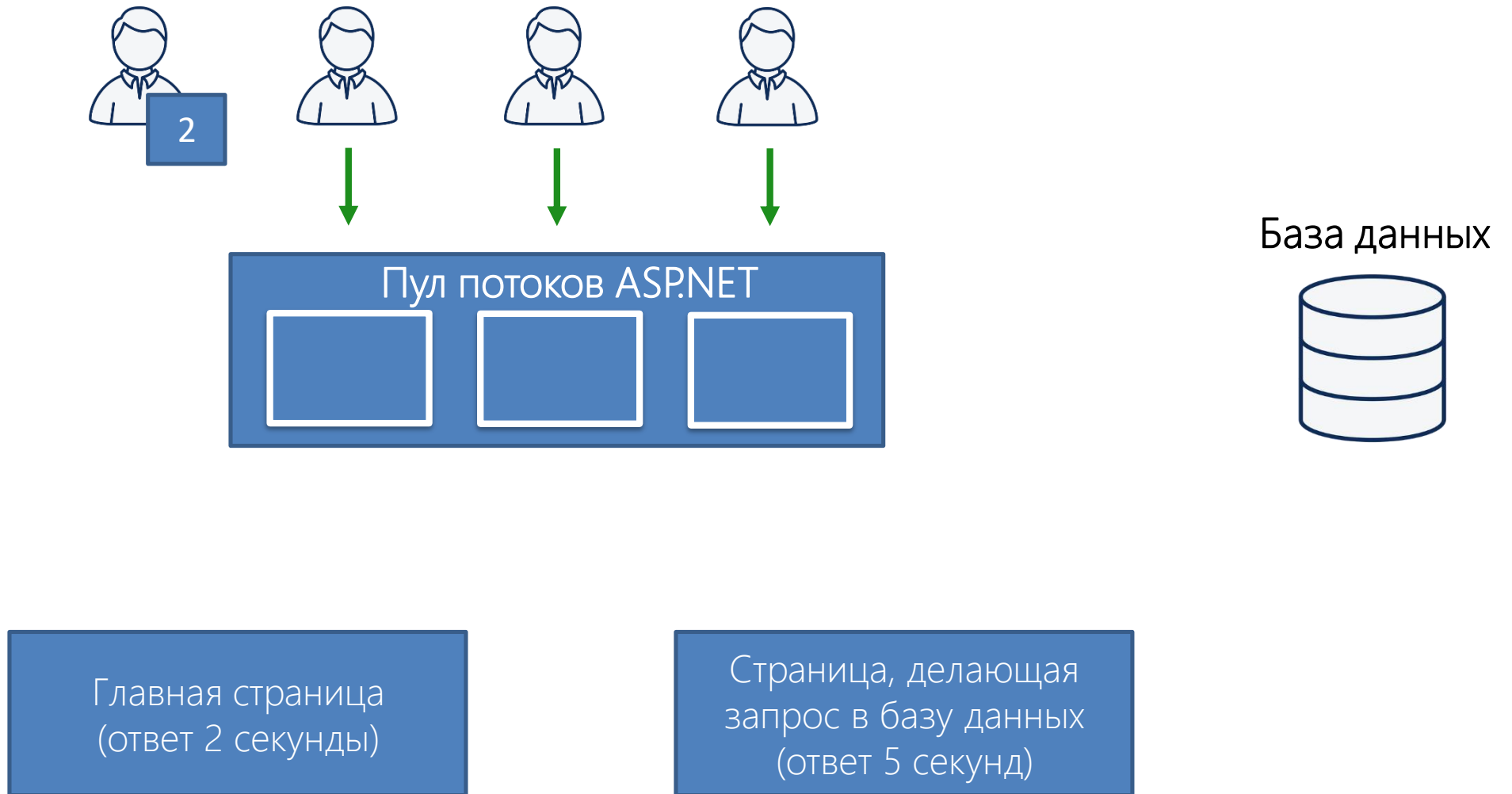
С# Асинхронное программирование

Асинхронная обработка запросов в ASP.NET



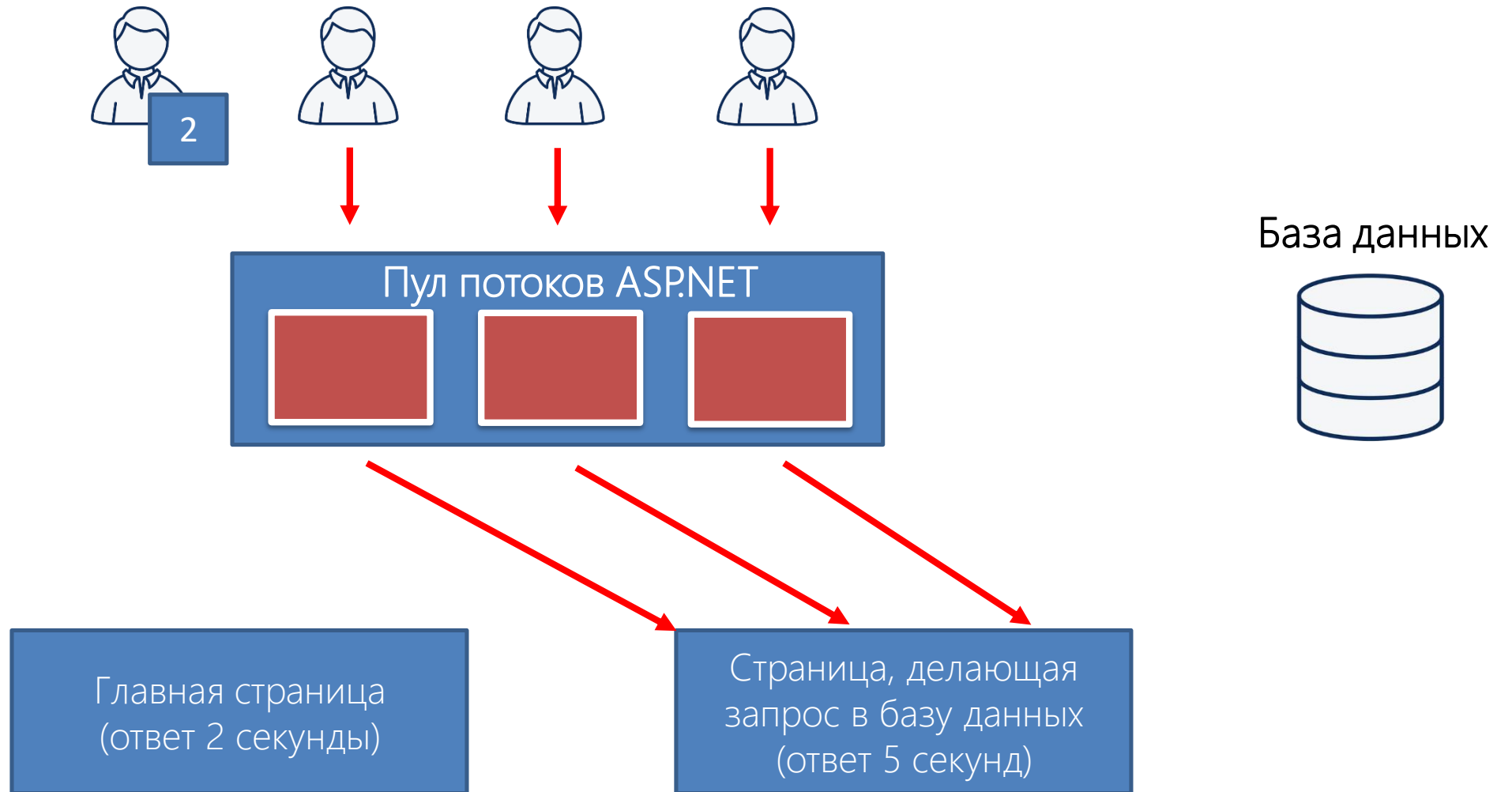
C# Асинхронное программирование

Асинхронная обработка запросов в ASP.NET



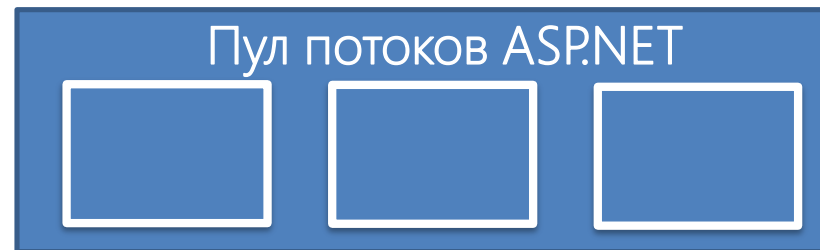
С# Асинхронное программирование

Асинхронная обработка запросов в ASP.NET



C# Асинхронное программирование

Асинхронная обработка запросов в ASP.NET



База данных



Главная страница
(ответ 2 секунды)

Страница, делающая
запрос в базу данных
(ответ 5 секунд)

C# Асинхронное программирование

Асинхронность в ASP.NET

Использование синхронных методов:

- Простые операции
- Операции работающие с ЦП

Использование асинхронных методов:

- Работа с файловой системой
- Работа с сетевыми запросами
- Работа с удаленной базой данных
- Работа с удаленными веб-сервисами



C# Асинхронное программирование

Асинхронные методы действия

Синхронный метод действия:

```
public ActionResult ShowCars()
{
    IEnumerable<Car> cars = db.Cars.ToList();
    return View(cars);
}
```

Асинхронный метод действия:

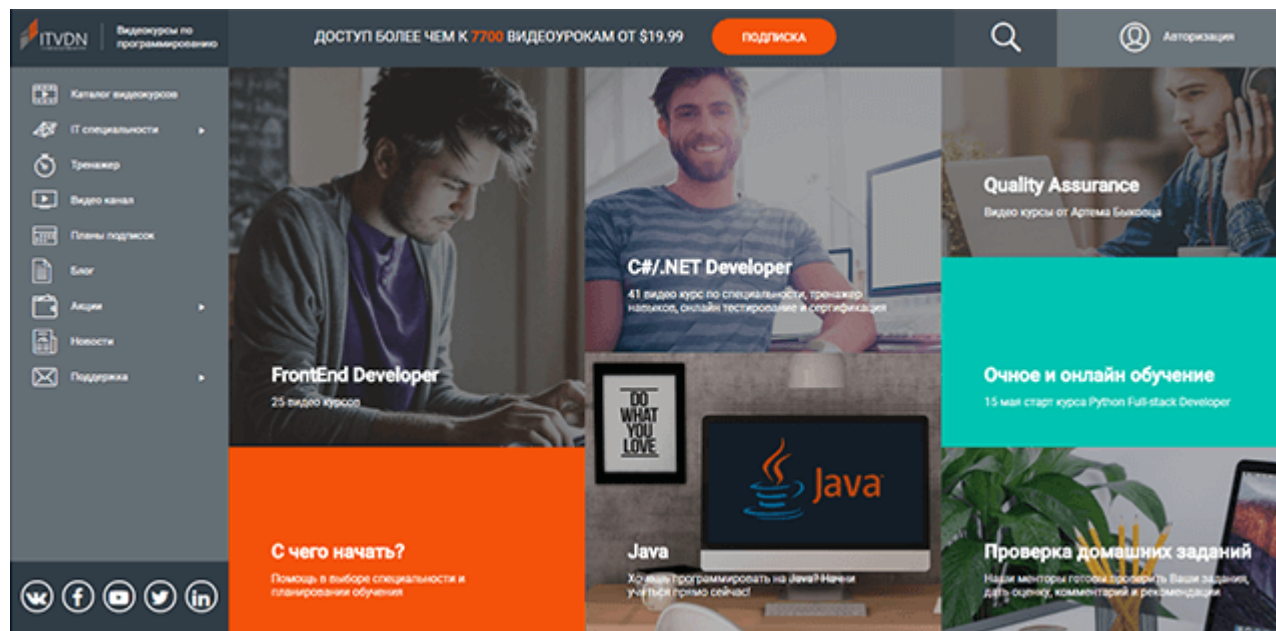
```
public async Task<ActionResult> ShowCarsAsync()
{
    IEnumerable<Car> cars = await db.Cars.ToListAsync();
    return View(cars);
}
```

C# Асинхронное программирование

Q&A

Смотрите наши уроки в видео формате

ITVDN.com



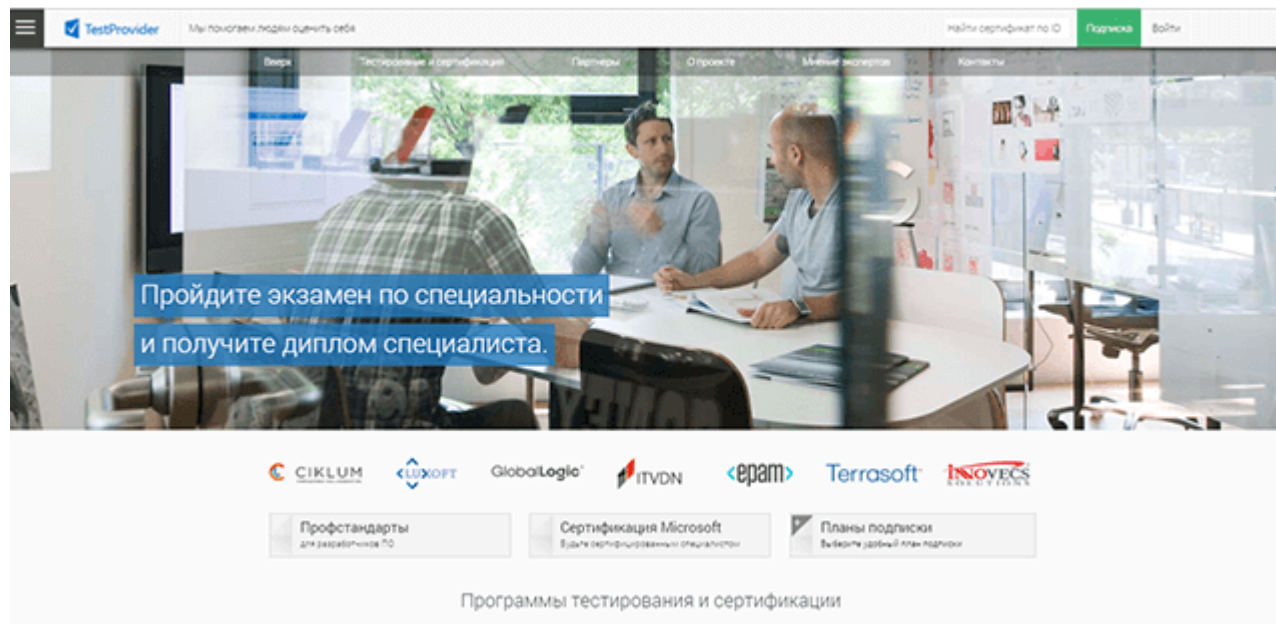
Посмотрите этот урок в видео формате на образовательном портале ITVDN.com для закрепления пройденного материала.

Курсы записаны сертифицированными тренерами, которые работают в учебном центре CyberBionic Systematics и другими высококвалифицированными разработчиками.



Проверка знаний

TestProvider.com



TestProvider – это online сервис проверки знаний по информационным технологиям. С его помощью Вы можете оценить Ваш уровень и выявить слабые места. Он будет полезен как в процессе изучения технологии, так и для общей оценки знаний IT специалиста.

После каждого урока проходите тестирование для проверки знаний на [TestProvider.com](https://testprovider.com)

Успешное прохождение финального тестирования позволит Вам получить соответствующий Сертификат.



Асинхронное программирование

После урока обязательно



Повторите этот урок в видео формате на [ITVDN.com](http://itvdn.com)



Проверьте как Вы усвоили данный материал на [TestProvider.com](http://testprovider.com)

Информационный видеосервис для разработчиков программного обеспечения

