

Klik [hier](#) voor pdf...

TESTEN

Les 8 Webdevelopment

WAAR STAAN WE?

1. Frontend: HTML, CSS, Javascript
2. Frontend: Bootstrap
3. Backend: C#
4. MVC
5. Backend: C# LINQ
6. ORM
7. Testen
8. Layout
9. Zoeken/filteren, Sorteren, Pagineren
10. Web API, JSON, Ajax, Azure
11. Security
12. Architectuur

VANDAAG OP HET PROGRAMMA

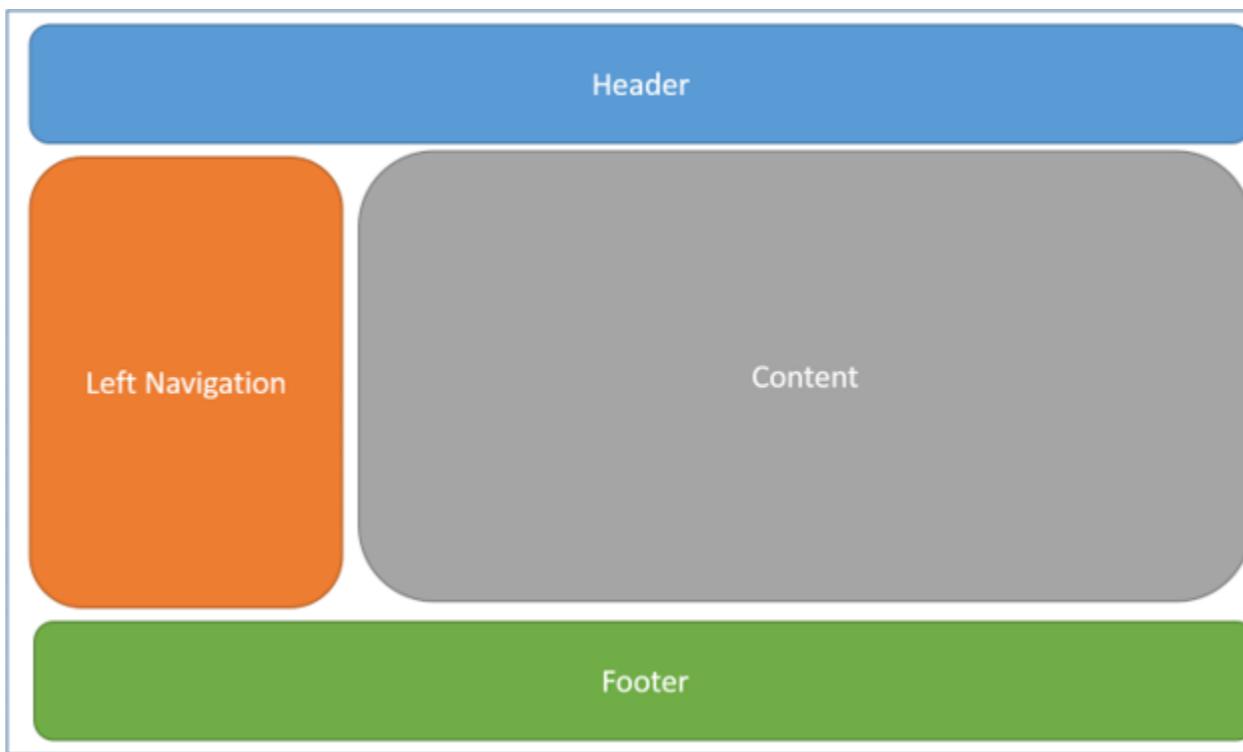
- Code hergebruik in de frontend
- ViewModel
- Client-side validatie
- Scaffolden
- Async
- Opdracht
- Overig: ViewData versus ViewBag

CODE HERGEBRUIK IN DE FRONTEND

SHARED LAYOUT

DESIGN VAN DE LAYOUT

Een pagina kan er bijvoorbeeld zo uitzien



De header, footer en navigation zijn heel vaak gelijk voor verschillende pagina's

OPLOSSING: SHARED LAYOUT FILE

Oplossing: Shared Layout file ([MSDN](#))

- Footer en Header komen op elke pagina voor (het gemeenschappelijke gedeelte)
- Het contentgedeelte is voor elke URL verschillend
- Anders gezegd: voor elke *Action* van iedere *Controller* wordt een andere pagina getoond
- In de *View* van de *Action* methode geef je aan welke *Shared Layout* je wilt gebruiken:

```
@{  
    Layout = "_LayoutSimple";  
    ViewData["Title"] = "Index";  
}
```

OPLOSSING: SHARED LAYOUT FILE

Oplossing: Shared Layout file ([MSDN](#))

- **Footer en Header** komen op elke pagina voor (het gemeenschappelijke gedeelte)
- Het contentgedeelte is voor elke URL verschillend
- Anders gezegd: voor elke *Action* van iedere *Controller* wordt een andere pagina getoond
- In de *View* van de *Action* methode geef je aan welke *Shared Layout* je wilt gebruiken:

```
@{  
    Layout = "_LayoutSimple";  
    ViewData["Title"] = "Index";  
}
```

OPLOSSING: SHARED LAYOUT FILE

Oplossing: Shared Layout file ([MSDN](#))

- **Footer en Header** komen op elke pagina voor (het gemeenschappelijke gedeelte)
- Het contentgedeelte is voor elke URL verschillend
- Anders gezegd: voor elke *Action* van iedere *Controller* wordt een andere pagina getoond
- In de *View* van de *Action* methode geef je aan welke *Shared Layout* je wilt gebruiken:

```
@{  
    Layout = "_LayoutSimple";  
    ViewData["Title"] = "Index";  
}
```

OPLOSSING: SHARED LAYOUT FILE

Oplossing: Shared Layout file ([MSDN](#))

- **Footer en Header** komen op elke pagina voor (het gemeenschappelijke gedeelte)
- Het contentgedeelte is voor elke URL verschillend
- Anders gezegd: voor elke *Action* van iedere *Controller* wordt een andere pagina getoond
- In de *View* van de *Action* methode geef je aan welke *Shared Layout* je wilt gebruiken:

```
@{  
    Layout = "_LayoutSimple";  
    ViewData["Title"] = "Index";  
}
```

OPLOSSING: SHARED LAYOUT FILE

Oplossing: Shared Layout file ([MSDN](#))

- **Footer en Header** komen op elke pagina voor (het gemeenschappelijke gedeelte)
- Het contentgedeelte is voor elke URL verschillend
- Anders gezegd: voor elke *Action* van iedere *Controller* wordt een andere pagina getoond
- In de *View* van de *Action* methode geef je aan welke *Shared Layout* je wilt gebruiken:

```
@{  
    Layout = "_LayoutSimple";  
    ViewData["Title"] = "Index";  
}
```

DE SHARED LAYOUT FILE

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>@ViewData["Title"] - WebApplicationCoreEFVoorbeeld</title>
</head>
<body>
<div style="background-color:#0000ff">
<p>Header: hello Shared Layout</p>
</div>
<div>
@RenderBody()
</div>
<hr />
<div style="background-color:#FF0000">
<p>footer © 2018 - WebApplicationCoreEFVoorbeeld</p>
</div>
</body>
</html>
```

UITLEG SHARED LAYOUT

- Een *Shared Layout* moet starten met de regel `<!DOCTYPE html>`
 - Dit is bij een normale HTML-file altijd de eerste regels (volgens de HTML-syntax)
- In de head kun je links naar stylesheets en javascript-bestanden toevoegen
- De content (die wordt aangemaakt in je *View*) wordt toegevoegd op de plek van `@RenderBody()`
- De *Views* die je zelf maakt, zijn dus geen zelfstandig HTML document
 - en beginnen dus ook niet met `<!DOCTYPE html>`
- Check met F12 de DOM van het uiteindelijke HTML-document
 - Daar zie je dat de HTML in je *View* is toegevoegd op de plek van `@RenderBody()`

UITLEG SHARED LAYOUT

- Een *Shared Layout* moet starten met de regel `<!DOCTYPE html>`
 - Dit is bij een normale HTML-file altijd de eerste regels (volgens de HTML-syntax)
- In de head kun je links naar stylesheets en javascript-bestanden toevoegen
- De content (die wordt aangemaakt in je View) wordt toegevoegd op de plek van `@RenderBody()`
- De Views die je zelf maakt, zijn dus geen zelfstandig HTML document
 - en beginnen dus ook niet met `<!DOCTYPE html>`
- Check met F12 de DOM van het uiteindelijke HTML-document
 - Daar zie je dat de HTML in je View is toegevoegd op de plek van `@RenderBody()`

UITLEG SHARED LAYOUT

- Een *Shared Layout* moet starten met de regel `<!DOCTYPE html>`
 - Dit is bij een normale HTML-file altijd de eerste regels (volgens de HTML-syntax)
- In de head kun je links naar stylesheets en javascript-bestanden toevoegen
- De content (die wordt aangemaakt in je *View*) wordt toegevoegd op de plek van `@RenderBody()`
- De *Views* die je zelf maakt, zijn dus geen zelfstandig HTML document
 - en beginnen dus ook niet met `<!DOCTYPE html>`
- Check met F12 de DOM van het uiteindelijke HTML-document
 - Daar zie je dat de HTML in je *View* is toegevoegd op de plek van `@RenderBody()`

UITLEG SHARED LAYOUT

- Een *Shared Layout* moet starten met de regel `<!DOCTYPE html>`
 - Dit is bij een normale HTML-file altijd de eerste regels (volgens de HTML-syntax)
- In de head kun je links naar stylesheets en javascript-bestanden toevoegen
- De content (die wordt aangemaakt in je *View*) wordt toegevoegd op de plek van `@RenderBody()`
- De *Views* die je zelf maakt, zijn dus geen zelfstandig HTML document
 - en beginnen dus ook niet met `<!DOCTYPE html>`
- Check met F12 de DOM van het uiteindelijke HTML-document
 - Daar zie je dat de HTML in je *View* is toegevoegd op de plek van `@RenderBody()`

UITLEG SHARED LAYOUT

- Een *Shared Layout* moet starten met de regel `<!DOCTYPE html>`
 - Dit is bij een normale HTML-file altijd de eerste regels (volgens de HTML-syntax)
- In de head kun je links naar stylesheets en javascript-bestanden toevoegen
- De content (die wordt aangemaakt in je *View*) wordt toegevoegd op de plek van `@RenderBody()`
- De *Views* die je zelf maakt, zijn dus geen zelfstandig HTML document
 - en beginnen dus ook niet met `<!DOCTYPE html>`
- Check met F12 de DOM van het uiteindelijke HTML-document
 - Daar zie je dat de HTML in je *View* is toegevoegd op de plek van `@RenderBody()`

UITLEG SHARED LAYOUT

- Een *Shared Layout* moet starten met de regel `<!DOCTYPE html>`
 - Dit is bij een normale HTML-file altijd de eerste regels (volgens de HTML-syntax)
- In de head kun je links naar stylesheets en javascript-bestanden toevoegen
- De content (die wordt aangemaakt in je *View*) wordt toegevoegd op de plek van `@RenderBody()`
- De *Views* die je zelf maakt, zijn dus geen zelfstandig HTML document
 - en beginnen dus ook niet met `<!DOCTYPE html>`
- Check met F12 de DOM van het uiteindelijke HTML-document
 - Daar zie je dat de HTML in je *View* is toegevoegd op de plek van `@RenderBody()`

UITLEG SHARED LAYOUT

- Een *Shared Layout* moet starten met de regel `<!DOCTYPE html>`
 - Dit is bij een normale HTML-file altijd de eerste regels (volgens de HTML-syntax)
- In de head kun je links naar stylesheets en javascript-bestanden toevoegen
- De content (die wordt aangemaakt in je *View*) wordt toegevoegd op de plek van `@RenderBody()`
- De *Views* die je zelf maakt, zijn dus geen zelfstandig HTML document
 - en beginnen dus ook niet met `<!DOCTYPE html>`
- Check met F12 de DOM van het uiteindelijke HTML-document
 - Daar zie je dat de HTML in je *View* is toegevoegd op de plek van `@RenderBody()`

UITLEG SHARED LAYOUT

- Een *Shared Layout* moet starten met de regel `<!DOCTYPE html>`
 - Dit is bij een normale HTML-file altijd de eerste regels (volgens de HTML-syntax)
- In de head kun je links naar stylesheets en javascript-bestanden toevoegen
- De content (die wordt aangemaakt in je *View*) wordt toegevoegd op de plek van `@RenderBody()`
- De *Views* die je zelf maakt, zijn dus geen zelfstandig HTML document
 - en beginnen dus ook niet met `<!DOCTYPE html>`
- Check met F12 de DOM van het uiteindelijke HTML-document
 - Daar zie je dat de HTML in je *View* is toegevoegd op de plek van `@RenderBody()`

Viewdata GEBRUIKEN IN SHARED LAYOUT

- In de *Shared Layout* kun je **ViewData** gebruiken:
 - Bijvoorbeeld om de titel van de pagina in de *header* te tonen
 - Of om een variabele in de *header* of in de *footer* te tonen
- De je *Action*-method van de *Controller* moet je deze **ViewData** dan natuurlijk wel vullen!
- Het hele idee van een *Shared Layout* is het voorkomen van dubbele HTML in de diverse *Views*
 - Dubbele HTML-code is namelijk slecht onderhoudbaar omdat elke wijziging op meerdere plekken moet worden doorgevoerd

Viewdata GEBRUIKEN IN SHARED LAYOUT

- In de *Shared Layout* kun je **ViewData** gebruiken:
 - Bijvoorbeeld om de titel van de pagina in de *header* te tonen
 - Of om een variabele in de *header* of in de *footer* te tonen
- De je *Action*-method van de *Controller* moet je deze **ViewData** dan natuurlijk wel vullen!
- Het hele idee van een *Shared Layout* is het voorkomen van dubbele HTML in de diverse *Views*
 - Dubbele HTML-code is namelijk slecht onderhoudbaar omdat elke wijziging op meerdere plekken moet worden doorgevoerd

Viewdata GEBRUIKEN IN SHARED LAYOUT

- In de *Shared Layout* kun je **ViewData** gebruiken:
 - Bijvoorbeeld om de titel van de pagina in de *header* te tonen
 - Of om een variabele in de *header* of in de *footer* te tonen
- De je *Action*-method van de *Controller* moet je deze **ViewData** dan natuurlijk wel vullen!
- Het hele idee van een *Shared Layout* is het voorkomen van dubbele HTML in de diverse *Views*
 - Dubbele HTML-code is namelijk slecht onderhoudbaar omdat elke wijziging op meerdere plekken moet worden doorgevoerd

Viewdata GEBRUIKEN IN SHARED LAYOUT

- In de *Shared Layout* kun je **ViewData** gebruiken:
 - Bijvoorbeeld om de titel van de pagina in de *header* te tonen
 - Of om een variabele in de *header* of in de *footer* te tonen
- De je *Action*-method van de *Controller* moet je deze **ViewData** dan natuurlijk wel vullen!
- Het hele idee van een *Shared Layout* is het voorkomen van dubbele HTML in de diverse *Views*
 - Dubbele HTML-code is namelijk slecht onderhoudbaar omdat elke wijziging op meerdere plekken moet worden doorgevoerd

Viewdata GEBRUIKEN IN SHARED LAYOUT

- In de *Shared Layout* kun je ViewData gebruiken:
 - Bijvoorbeeld om de titel van de pagina in de *header* te tonen
 - Of om een variabele in de *header* of in de *footer* te tonen
- De je Action-method van de Controller moet je deze ViewData dan natuurlijk wel vullen!
- Het hele idee van een *Shared Layout* is het voorkomen van dubbele HTML in de diverse Views
 - Dubbele HTML-code is namelijk slecht onderhoudbaar omdat elke wijziging op meerdere plekken moet worden doorgevoerd

Viewdata GEBRUIKEN IN SHARED LAYOUT

- In de *Shared Layout* kun je ViewData gebruiken:
 - Bijvoorbeeld om de titel van de pagina in de *header* te tonen
 - Of om een variabele in de *header* of in de *footer* te tonen
- De je Action-method van de Controller moet je deze ViewData dan natuurlijk wel vullen!
- Het hele idee van een *Shared Layout* is het voorkomen van dubbele HTML in de diverse Views
 - Dubbele HTML-code is namelijk slecht onderhoudbaar omdat elke wijziging op meerdere plekken moet worden doorgevoerd

PARTIAL VIEWS

PARTIAL VIEWS

Ook **Partial Views** ([MSDN](#)) zijn bedoeld om onderhoudbaarheid van de HTML in een View te verbeteren:

- Je neemt een deel van je View op in een *Partial View*
 - Dit is een onderdeel van je pagina dat ook op andere pagina's kan worden hergebruikt
- Op deze manier kun je een grotere View in kleinere stukken opdelen
- De *Partial View* is vooral handig voor HTML die in meerdere Views voorkomt maar niet in alle Views:
 - Deze HTML-code kun je dus niet in een *Shared Layout* plaatsen
- Als je bijvoorbeeld een inlogformulier wilt plaatsen op meerdere pagina's, dan doe je dat als volgt:

```
<partial name="_LoginPartial" />
```
- Razor zoekt nu naar "_LoginPartial.cshtml" in Views/Shared.

PARTIAL VIEWS

Ook **Partial Views** ([MSDN](#)) zijn bedoeld om onderhoudbaarheid van de HTML in een View te verbeteren:

- Je neemt een deel van je *View* op in een *Partial View*
 - Dit is een onderdeel van je pagina dat ook op andere pagina's kan worden hergebruikt
- Op deze manier kun je een grotere View in kleinere stukken opdelen
- De *Partial View* is vooral handig voor HTML die in meerdere *Views* voorkomt maar niet in alle *Views*:
 - Deze HTML-code kun je dus niet in een *Shared Layout* plaatsen
- Als je bijvoorbeeld een inlogformulier wilt plaatsen op meerdere pagina's, dan doe je dat als volgt:

```
<partial name="_LoginPartial" />
```
- Razor zoekt nu naar `"_LoginPartial.cshtml"` in `Views/Shared`.

PARTIAL VIEWS

Ook **Partial Views** ([MSDN](#)) zijn bedoeld om onderhoudbaarheid van de HTML in een View te verbeteren:

- Je neemt een deel van je *View* op in een *Partial View*
 - Dit is een onderdeel van je pagina dat ook op andere pagina's kan worden hergebruikt
- Op deze manier kun je een grotere View in kleinere stukken opdelen
- De *Partial View* is vooral handig voor HTML die in meerdere *Views* voorkomt maar niet in alle *Views*:
 - Deze HTML-code kun je dus niet in een *Shared Layout* plaatsen
- Als je bijvoorbeeld een inlogformulier wilt plaatsen op meerdere pagina's, dan doe je dat als volgt:

```
<partial name="_LoginPartial" />
```

- Razor zoekt nu naar `"_LoginPartial.cshtml"` in `Views/Shared`.

PARTIAL VIEWS

Ook **Partial Views** ([MSDN](#)) zijn bedoeld om onderhoudbaarheid van de HTML in een View te verbeteren:

- Je neemt een deel van je *View* op in een *Partial View*
 - Dit is een onderdeel van je pagina dat ook op andere pagina's kan worden hergebruikt
- Op deze manier kun je een grotere View in kleinere stukken opdelen
- De *Partial View* is vooral handig voor HTML die in meerdere *Views* voorkomt maar niet in alle *Views*:
 - Deze HTML-code kun je dus niet in een *Shared Layout* plaatsen
- Als je bijvoorbeeld een inlogformulier wilt plaatsen op meerdere pagina's, dan doe je dat als volgt:

```
<partial name="_LoginPartial" />
```

- Razor zoekt nu naar `"_LoginPartial.cshtml"` in `Views/Shared`.

PARTIAL VIEWS

Ook **Partial Views** ([MSDN](#)) zijn bedoeld om onderhoudbaarheid van de HTML in een View te verbeteren:

- Je neemt een deel van je *View* op in een *Partial View*
 - Dit is een onderdeel van je pagina dat ook op andere pagina's kan worden hergebruikt
- Op deze manier kun je een grotere View in kleinere stukken opdelen
- De *Partial View* is vooral handig voor HTML die in meerdere *Views* voorkomt maar niet in alle *Views*:
 - Deze HTML-code kun je dus niet in een *Shared Layout* plaatsen
- Als je bijvoorbeeld een inlogformulier wilt plaatsen op meerdere pagina's, dan doe je dat als volgt:

```
<partial name="_LoginPartial" />
```

- Razor zoekt nu naar `"_LoginPartial.cshtml"` in `Views/Shared`.

PARTIAL VIEWS

Ook **Partial Views** ([MSDN](#)) zijn bedoeld om onderhoudbaarheid van de HTML in een View te verbeteren:

- Je neemt een deel van je *View* op in een *Partial View*
 - Dit is een onderdeel van je pagina dat ook op andere pagina's kan worden hergebruikt
- Op deze manier kun je een grotere View in kleinere stukken opdelen
- De *Partial View* is vooral handig voor HTML die in meerdere *Views* voorkomt maar niet in alle *Views*:
 - Deze HTML-code kun je dus niet in een *Shared Layout* plaatsen
- Als je bijvoorbeeld een inlogformulier wilt plaatsen op meerdere pagina's, dan doe je dat als volgt:

```
<partial name="_LoginPartial" />
```

- Razor zoekt nu naar `"_LoginPartial.cshtml"` in *Views/Shared*.

PARTIAL VIEWS

Ook **Partial Views** ([MSDN](#)) zijn bedoeld om onderhoudbaarheid van de HTML in een View te verbeteren:

- Je neemt een deel van je *View* op in een *Partial View*
 - Dit is een onderdeel van je pagina dat ook op andere pagina's kan worden hergebruikt
- Op deze manier kun je een grotere View in kleinere stukken opdelen
- De *Partial View* is vooral handig voor HTML die in meerdere *Views* voorkomt maar niet in alle *Views*:
 - Deze HTML-code kun je dus niet in een *Shared Layout* plaatsen
- Als je bijvoorbeeld een inlogformulier wilt plaatsen op meerdere pagina's, dan doe je dat als volgt:

```
<partial name="_LoginPartial" />
```

- Razor zoekt nu naar `"_LoginPartial.cshtml"` in `Views/Shared`.

PARTIAL VIEWS

Ook **Partial Views** ([MSDN](#)) zijn bedoeld om onderhoudbaarheid van de HTML in een View te verbeteren:

- Je neemt een deel van je *View* op in een *Partial View*
 - Dit is een onderdeel van je pagina dat ook op andere pagina's kan worden hergebruikt
- Op deze manier kun je een grotere View in kleinere stukken opdelen
- De *Partial View* is vooral handig voor HTML die in meerdere *Views* voorkomt maar niet in alle *Views*:
 - Deze HTML-code kun je dus niet in een *Shared Layout* plaatsen
- Als je bijvoorbeeld een inlogformulier wilt plaatsen op meerdere pagina's, dan doe je dat als volgt:

```
<partial name="_LoginPartial" />
```

- Razor zoekt nu naar `"_LoginPartial.cshtml"` in *Views/Shared*.

PARTIAL VIEW MET DATA

- Je kunt ook data meegeven aan de *Partial View*
- Het is sterk aan te bevelen om een *Model* mee te geven (zo'n *Partial View* wordt een **Strongly Typed Partial View** genoemd).
 - In de View wordt dan compiletime al gecheckt of de Properties van het Model bestaan
 - Je kunt dit bijvoorbeeld gebruiken voor een lijst met nieuwsberichten (bijv.: NOS.nl)

```
@foreach (var artikel in @Model) { // @Model is een List<Artikel>
    <partial name="_Artikel" model="artikel" />
} // alternatief: <partial name="_Artikel" for="@artikel" />
```
- Naast een (strongly typed) Model kun je ook gebruik maken van ViewData
- Ook hiervoor geldt natuurlijk dat de Action het Model of de ViewData moet vullen
 - Want: een Partial View heeft geen eigen controller!

PARTIAL VIEW MET DATA

- Je kunt ook data meegeven aan de *Partial View*
- Het is sterk aan te bevelen om een *Model* mee te geven (zo'n *Partial View* wordt een **Strongly Typed Partial View** genoemd).
 - In de View wordt dan compiletime al gecheckt of de Properties van het Model bestaan
 - Je kunt dit bijvoorbeeld gebruiken voor een lijst met nieuwsberichten (bijv.: NOS.nl)

```
@foreach (var artikel in @Model) { // @Model is een List<Artikel>
    <partial name="_Artikel" model="artikel" />
} // alternatief: <partial name="_Artikel" for="@artikel" />
```
- Naast een (strongly typed) Model kun je ook gebruik maken van ViewData
- Ook hiervoor geldt natuurlijk dat de Action het Model of de ViewData moet vullen
 - Want: een Partial View heeft geen eigen controller!

PARTIAL VIEW MET DATA

- Je kunt ook data meegeven aan de *Partial View*
- Het is sterk aan te bevelen om een *Model* mee te geven (zo'n *Partial View* wordt een **Strongly Typed Partial View** genoemd).
 - In de View wordt dan compiletime al gecheckt of de Properties van het Model bestaan
 - Je kunt dit bijvoorbeeld gebruiken voor een lijst met nieuwsberichten (bijv.: NOS.nl)

```
@foreach (var artikel in @Model) { // @Model is een List<Artikel>
    <partial name="_Artikel" model="artikel" />
} // alternatief: <partial name="_Artikel" for="@artikel" />
```
- Naast een (strongly typed) Model kun je ook gebruik maken van ViewData
- Ook hiervoor geldt natuurlijk dat de Action het Model of de ViewData moet vullen
 - Want: een Partial View heeft geen eigen controller!

PARTIAL VIEW MET DATA

- Je kunt ook data meegeven aan de *Partial View*
- Het is sterk aan te bevelen om een *Model* mee te geven (zo'n *Partial View* wordt een **Strongly Typed Partial View** genoemd).

- In de View wordt dan compiletime al gecheckt of de Properties van het Model bestaan
- Je kunt dit bijvoorbeeld gebruiken voor een lijst met nieuwsberichten (bijv.: NOS.nl)

```
@foreach (var artikel in @Model) { // @Model is een List<Artikel>
    <partial name="_Artikel" model="artikel" />
} // alternatief: <partial name="_Artikel" for="@artikel" />
```

- Naast een (strongly typed) Model kun je ook gebruik maken van ViewData
- Ook hiervoor geldt natuurlijk dat de Action het Model of de ViewData moet vullen
 - Want: een Partial View heeft geen eigen controller!

PARTIAL VIEW MET DATA

- Je kunt ook data meegeven aan de *Partial View*
- Het is sterk aan te bevelen om een *Model* mee te geven (zo'n *Partial View* wordt een **Strongly Typed Partial View** genoemd).

- In de View wordt dan compiletime al gecheckt of de Properties van het Model bestaan
- Je kunt dit bijvoorbeeld gebruiken voor een lijst met nieuwsberichten (bijv.: NOS.nl)

```
@foreach (var artikel in @Model) { // @Model is een List<Artikel>
    <partial name="_Artikel" model="artikel" />
} // alternatief: <partial name="_Artikel" for="@artikel" />
```

- Naast een (strongly typed) Model kun je ook gebruik maken van ViewData
- Ook hiervoor geldt natuurlijk dat de Action het Model of de ViewData moet vullen
 - Want: een Partial View heeft geen eigen controller!

PARTIAL VIEW MET DATA

- Je kunt ook data meegeven aan de *Partial View*
- Het is sterk aan te bevelen om een *Model* mee te geven (zo'n *Partial View* wordt een **Strongly Typed Partial View** genoemd).

- In de View wordt dan compiletime al gecheckt of de Properties van het Model bestaan
- Je kunt dit bijvoorbeeld gebruiken voor een lijst met nieuwsberichten (bijv.: NOS.nl)

```
@foreach (var artikel in @Model) { // @Model is een List<Artikel>
    <partial name="_Artikel" model="artikel" />
} // alternatief: <partial name="_Artikel" for="@artikel" />
```

- Naast een (strongly typed) Model kun je ook gebruik maken van ViewData
- Ook hiervoor geldt natuurlijk dat de Action het Model of de ViewData moet vullen
 - Want: een Partial View heeft geen eigen controller!

PARTIAL VIEW MET DATA

- Je kunt ook data meegeven aan de *Partial View*
- Het is sterk aan te bevelen om een *Model* mee te geven (zo'n *Partial View* wordt een **Strongly Typed Partial View** genoemd).

- In de View wordt dan compiletime al gecheckt of de Properties van het Model bestaan
- Je kunt dit bijvoorbeeld gebruiken voor een lijst met nieuwsberichten (bijv.: NOS.nl)

```
@foreach (var artikel in @Model) { // @Model is een List<Artikel>
    <partial name="_Artikel" model="artikel" />
} // alternatief: <partial name="_Artikel" for="@artikel" />
```

- Naast een (strongly typed) Model kun je ook gebruik maken van ViewData
- Ook hiervoor geldt natuurlijk dat de Action het Model of de ViewData moet vullen
 - Want: een Partial View heeft geen eigen controller!

VIEWMODEL

MODEL VERSUS VIEWMODEL

- Een *Model* komt door de *ORM* in de database terecht. Een *ViewModel* niet.
- Een *Model* hoeft geen *View* te hebben. Een *ViewModel* wel.
 - Een *ViewModel* heeft geen bestaansrecht zonder de bijbehorende *View*.
- Een *Model* representeert een entiteit in het domeinmodel, dus staat centraal in de webapplicatie. Een *ViewModel* allebei niet.
 - Een *Model* representeert de toestand van de applicatie met bijbehorende business logica, en kan op veel plekken in de webapplicatie worden gebruikt.
 - Een *ViewModel* wordt alleen gebruikt om data mee te geven van een *Controller* naar *View* (en vanaf daar mogelijk weer verder naar een *Partial View*).
- *ViewModels* staan in een andere map dan de *Models*: voorkomt verwarring

Dit is een architectuur pattern: **Model-View-ViewModel (MVVM)**.

MODEL VERSUS VIEWMODEL

- Een *Model* komt door de *ORM* in de database terecht. Een *ViewModel* niet.
- Een *Model* hoeft geen *View* te hebben. Een *ViewModel* wel.
 - Een *ViewModel* heeft geen bestaansrecht zonder de bijbehorende *View*.
- Een *Model* representeert een entiteit in het domeinmodel, dus staat centraal in de webapplicatie. Een *ViewModel* allebei niet.
 - Een *Model* representeert de toestand van de applicatie met bijbehorende business logica, en kan op veel plekken in de webapplicatie worden gebruikt.
 - Een *ViewModel* wordt alleen gebruikt om data mee te geven van een *Controller* naar *View* (en vanaf daar mogelijk weer verder naar een *Partial View*).
- *ViewModels* staan in een andere map dan de *Models*: voorkomt verwarring

Dit is een architectuur pattern: **Model-View-ViewModel (MVVM)**.

MODEL VERSUS VIEWMODEL

- Een *Model* komt door de *ORM* in de database terecht. Een *ViewModel* niet.
- Een *Model* hoeft geen *View* te hebben. Een *ViewModel* wel.
 - Een *ViewModel* heeft geen bestaansrecht zonder de bijbehorende *View*.
- Een *Model* representeert een entiteit in het domeinmodel, dus staat centraal in de webapplicatie. Een *ViewModel* allebei niet.
 - Een *Model* representeert de toestand van de applicatie met bijbehorende business logica, en kan op veel plekken in de webapplicatie worden gebruikt.
 - Een *ViewModel* wordt alleen gebruikt om data mee te geven van een *Controller* naar *View* (en vanaf daar mogelijk weer verder naar een *Partial View*).
- *ViewModels* staan in een andere map dan de *Models*: voorkomt verwarring

Dit is een architectuur pattern: **Model-View-ViewModel (MVVM)**.

MODEL VERSUS VIEWMODEL

- Een *Model* komt door de *ORM* in de database terecht. Een *ViewModel* niet.
- Een *Model* hoeft geen *View* te hebben. Een *ViewModel* wel.
 - Een *ViewModel* heeft geen bestaansrecht zonder de bijbehorende *View*.
- Een *Model* representeert een entiteit in het domeinmodel, dus staat centraal in de webapplicatie. Een *ViewModel* allebei niet.
 - Een *Model* representeert de toestand van de applicatie met bijbehorende business logica, en kan op veel plekken in de webapplicatie worden gebruikt.
 - Een *ViewModel* wordt alleen gebruikt om data mee te geven van een *Controller* naar *View* (en vanaf daar mogelijk weer verder naar een *Partial View*).
- *ViewModels* staan in een andere map dan de *Models*: voorkomt verwarring

Dit is een architectuur pattern: **Model-View-ViewModel (MVVM)**.

MODEL VERSUS VIEWMODEL

- Een *Model* komt door de *ORM* in de database terecht. Een *ViewModel* niet.
- Een *Model* hoeft geen *View* te hebben. Een *ViewModel* wel.
 - Een *ViewModel* heeft geen bestaansrecht zonder de bijbehorende *View*.
- Een *Model* representeert een entiteit in het domeinmodel, dus staat centraal in de webapplicatie. Een *ViewModel* allebei niet.
 - Een *Model* representeert de toestand van de applicatie met bijbehorende business logica, en kan op veel plekken in de webapplicatie worden gebruikt.
 - Een *ViewModel* wordt alleen gebruikt om data mee te geven van een *Controller* naar *View* (en vanaf daar mogelijk weer verder naar een *Partial View*).
- *ViewModels* staan in een andere map dan de *Models*: voorkomt verwarring

Dit is een architectuur pattern: **Model-View-ViewModel (MVVM)**.

MODEL VERSUS VIEWMODEL

- Een *Model* komt door de *ORM* in de database terecht. Een *ViewModel* niet.
- Een *Model* hoeft geen *View* te hebben. Een *ViewModel* wel.
 - Een *ViewModel* heeft geen bestaansrecht zonder de bijbehorende *View*.
- Een *Model* representeert een entiteit in het domeinmodel, dus staat centraal in de webapplicatie. Een *ViewModel* allebei niet.
 - Een *Model* representeert de toestand van de applicatie met bijbehorende business logica, en kan op veel plekken in de webapplicatie worden gebruikt.
 - Een *ViewModel* wordt alleen gebruikt om data mee te geven van een *Controller* naar *View* (en vanaf daar mogelijk weer verder naar een *Partial View*).
- *ViewModels* staan in een andere map dan de *Models*: voorkomt verwarring

Dit is een architectuur pattern: **Model-View-ViewModel (MVVM)**.

MODEL VERSUS VIEWMODEL

- Een *Model* komt door de *ORM* in de database terecht. Een *ViewModel* niet.
- Een *Model* hoeft geen *View* te hebben. Een *ViewModel* wel.
 - Een *ViewModel* heeft geen bestaansrecht zonder de bijbehorende *View*.
- Een *Model* representeert een entiteit in het domeinmodel, dus staat centraal in de webapplicatie. Een *ViewModel* allebei niet.
 - Een *Model* representeert de toestand van de applicatie met bijbehorende business logica, en kan op veel plekken in de webapplicatie worden gebruikt.
 - Een *ViewModel* wordt alleen gebruikt om data mee te geven van een *Controller* naar *View* (en vanaf daar mogelijk weer verder naar een *Partial View*).
- *ViewModels* staan in een andere map dan de *Models*: voorkomt verwarring

Dit is een architectuur pattern: **Model-View-ViewModel (MVVM)**.

MODEL VERSUS VIEWMODEL

- Een *Model* komt door de *ORM* in de database terecht. Een *ViewModel* niet.
- Een *Model* hoeft geen *View* te hebben. Een *ViewModel* wel.
 - Een *ViewModel* heeft geen bestaansrecht zonder de bijbehorende *View*.
- Een *Model* representeert een entiteit in het domeinmodel, dus staat centraal in de webapplicatie. Een *ViewModel* allebei niet.
 - Een *Model* representeert de toestand van de applicatie met bijbehorende business logica, en kan op veel plekken in de webapplicatie worden gebruikt.
 - Een *ViewModel* wordt alleen gebruikt om data mee te geven van een *Controller* naar *View* (en vanaf daar mogelijk weer verder naar een *Partial View*).
- *ViewModels* staan in een andere map dan de *Models*: voorkomt verwarring

Dit is een architectuur pattern: **Model-View-ViewModel (MVVM)**.

VOORBEELDEN VIEWMODEL

Voorbeeld 1: Company en Project zijn beide een *Model*.

```
public class ProjectDashboardViewModel
{
    public Company Company { get; set; }
    public IEnumerable<Project> AllProjects { get; set; }
    public int NumberOfFinishedProjects { get; set; }
    public IEnumerable<Project> OverdueProjects { get; set; }
}
```

Voorbeeld 2: Student is een *Model* zonder FullName, met CreatedAt.

```
public class StudentViewModel
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string FullName { get; }
}
```

VOORBEELDEN VIEWMODEL

Voorbeeld 1: Company en Project zijn beide een *Model*.

```
public class ProjectDashboardViewModel
{
    public Company Company { get; set; }
    public IEnumerable<Project> AllProjects { get; set; }
    public int NumberOfFinishedProjects { get; set; }
    public IEnumerable<Project> OverdueProjects { get; set; }
}
```

Voorbeeld 2: Student is een *Model* zonder FullName, met CreatedAt.

```
public class StudentViewModel
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string FullName { get; }
}
```

CLIENT-SIDE VALIDATIE

HERINNER: SERVER-SIDE VALIDATIE

```
public class Movie
{
    public int Id { get; set; }

    [StringLength(60, MinimumLength = 3)]
    [Required]
    public string Title { get; set; }

    [Display(Name = "Release Date")]
    [DataType(DataType.Date)]
    public DateTime ReleaseDate { get; set; }

    [Range(1, 100)]
    [DataType(DataType.Currency)]
    [Column(TypeName = "decimal(18, 2)")]
    public decimal Price { get; set; }

    [RegularExpression(@"^ [A-Z]+[a-zA-Z]*$")]
    [Required]
    [StringLength(30)]
    public string Genre { get; set; }

    [RegularExpression(@"^ [A-Z]+[a-zA-Z0-9""]"\s-]*$")]
    [StringLength(5)]
    [Required]
    public string Rating { get; set; }
}
```

```
// GET: Movies/Create
public IActionResult Create()
{
    return View();
}

// POST: Movies/Create
[HttpPost]
public void IActionResult Create(Movie movie)
{
    if (ModelState.IsValid)
    {
        _context.Add(movie);
        _context.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(movie);
}
```

Niet alle data-annotaties zijn constraints in de database!

HERINNERT: SERVER-SIDE VALIDATIE

```
public class Movie
{
    public int Id { get; set; }

    [StringLength(60, MinimumLength = 3)]
    [Required]
    public string Title { get; set; }

    [Display(Name = "Release Date")]
    [DataType(DataType.Date)]
    public DateTime ReleaseDate { get; set; }

    [Range(1, 100)]
    [DataType(DataType.Currency)]
    [Column(TypeName = "decimal(18, 2)")]
    public decimal Price { get; set; }

    [RegularExpression(@"^[A-Z][a-zA-Z]*$")]
    [Required]
    [StringLength(30)]
    public string Genre { get; set; }

    [RegularExpression(@"^[A-Z][a-zA-Z0-9]*$")]
    [StringLength(5)]
    [Required]
    public string Rating { get; set; }
}
```

```
// GET: Movies/Create
public IActionResult Create()
{
    return View();
}

// POST: Movies/Create
[HttpPost]
public void IActionResult Create(Movie movie)
{
    if (ModelState.IsValid)
    {
        _context.Add(movie);
        _context.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(movie);
}
```

Niet alle data-annotaties zijn constraints in de database!

TWEE SOORTEN DATA-ANNOTATIES

- Bijv. `[Regex(...)]` en `[StringLength(10)]` zijn er alleen voor validatie
- Bijv. het type van de variabele, `[Key]` en `[MaxLength(10)]` bepalen het database schema [en validatie]
- ((je kunt ook check constraints toevoegen:
`modelBuilder.Entity<Student>(s=>s.HasCheckConstraint("constr1", "[Leeftijd] >= 18"));))`

TWEE SOORTEN DATA-ANNOTATIES

- Bijv. `[Regex(...)]` en `[StringLength(10)]` zijn er alleen voor validatie
- Bijv. het type van de variabele, `[Key]` en `[MaxLength(10)]` bepalen het database schema [en validatie]
- ((je kunt ook check constraints toevoegen:
`modelBuilder.Entity<Student>(s=>s.HasCheckConstraint("constr1", "[Leeftijd] >= 18"));))`

TWEE SOORTEN DATA-ANNOTATIES

- Bijv. `[Regex(...)]` en `[StringLength(10)]` zijn er alleen voor validatie
- Bijv. het type van de variabele, `[Key]` en `[MaxLength(10)]` bepalen het database schema [en validatie]
- ((je kunt ook check constraints toevoegen:

```
modelBuilder.Entity<Student>(s=>s.HasCheckConstraint("constr1", "[Leeftijd] >= 18"));))
```

HET VOORBEELD MODEL

```
public class Student
{
    public int Id { get; set; }
    [StringLength(10)]
    public string Name { get; set; }
    [Required]
    [Display(Name = "Geboortedatum")]
    public DateTime Geboren { get; set; }
    [Range(0, 10)]
    public double GemiddeldeCijfer { get; set; }
}
```

FORMULIER ZONDER VALIDATIE

(zie les 10b)

```
<form action="/Students/Create" method="post">
    <label for="Name">Name</label>
    <input type="text" id="Name" name="Name" value="">
    <label for="Geboren">Geboortedatum</label>
    <input type="datetime-local" id="Geboren" name="Geboren" value="">
    <label for="GemiddeldeCijfer">GemiddeldeCijfer</label>
    <input type="text" id="GemiddeldeCijfer" name="GemiddeldeCijfer" value="">
    <input type="submit" value="Create">
</form>
```

FORMULIER MET VALIDATIE

Importeer jQuery Unobtrusive Validation (standaard in ASP.NET Core MVC applicatie)

```
<form action="/Students/Create" method="post">
    <label for="Name">Name</label>
    <input type="text" data-val="true"
        data-val-length="The field Name must be a string with a maximum length of 10."
        data-val-length-max="10"
        id="Name" maxlength="10" name="Name" value="">
    <span data-valmsg-for="Name"></span>

    <label for="Geboren">Geboortedatum</label>
    <input type="datetime-local" data-val="true"
        data-val-required="The Geboortedatum field is required."
        id="Geboren" name="Geboren" value="">
    <span data-valmsg-for="Geboren"></span>

    <label for="GemiddeldeCijfer">GemiddeldeCijfer</label>
    <input class="form-control valid" type="text" data-val="true"
        data-val-number="The field GemiddeldeCijfer must be a number."
        data-val-range="The field GemiddeldeCijfer must be between 0 and 10." data-val-range-max="10" data-val-range-min="0"
        data-val-required="The GemiddeldeCijfer field is required."
        id="GemiddeldeCijfer" name="GemiddeldeCijfer" value="" aria-describedby="GemiddeldeCijfer-error" aria-invalid="false">
    <span data-valmsg-for="GemiddeldeCijfer"></span>
    <input type="submit" value="Create">
</form>
```

FORMULIER MET VALIDATIE MET TAG HELPERS

```
<form asp-action="Create">
<div asp-validation-summary="ModelOnly"></div>
<label asp-for="Name"></label>
<input asp-for="Name" />
<span asp-validation-for="Name"></span>
<label asp-for="Geboren"></label>
<input asp-for="Geboren" />
<span asp-validation-for="Geboren"></span>
<label asp-for="GemiddeldeCijfer"></label>
<input asp-for="GemiddeldeCijfer" />
<span asp-validation-for="GemiddeldeCijfer"></span>
<input type="submit" value="Create"/>
</form>
```

FORMULIER MET VALIDATIE MET TAG HELPERS EN OPMAAK

```
<form asp-action="Create">
    <div asp-validation-summary="ModelOnly" class="text-danger"></div>
    <div class="form-group">
        <label asp-for="Name" class="control-label"></label>
        <input asp-for="Name" class="form-control" />
        <span asp-validation-for="Name" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Geboren" class="control-label"></label>
        <input asp-for="Geboren" class="form-control" />
        <span asp-validation-for="Geboren" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="GemiddeldeCijfer" class="control-label"></label>
        <input asp-for="GemiddeldeCijfer" class="form-control" />
        <span asp-validation-for="GemiddeldeCijfer" class="text-danger"></span>
    </div>
    <div class="form-group">
        <input type="submit" value="Create" class="btn btn-primary" />
    </div>
```

```
</div>
```

```
</form>
```

WAT ZIJN TAG HELPERS?

MSDN

- **Tag Helpers** genereren een stuk HTML-code met allerlei attributen voor je.
- Het is natuurlijk niet verplicht om *Tag Helpers* te gebruiken. Je mag de HTML ook uitschrijven
- Echter: ze besparen je typewerk in je *View*:
 - Het is altijd zinvol om te weten wat ze voor jou doen
- De *Tag Helpers* kom je o.a. veel tegen in *Views* die je via *Scaffolding* kan maken.
 - Sommige *Tag Helpers* doen veel voor jou, o.a. voor validatie in *formulieren*
 - Andere *Tag Helpers* doen eigenlijk vrij weinig
- N.B.: *HTML-Helpers* zijn een oudere manier om hetzelfde te doen
 - Het advies is om de modernere *Tag Helpers* te gebruiken als je kunt kiezen voor een *Tag Helper* i.p.v. voor een *HTML-Helper*

WAT ZIJN TAG HELPERS?

MSDN

- **Tag Helpers** genereren een stuk HTML-code met allerlei attributen voor je.
- Het is natuurlijk niet verplicht om *Tag Helpers* te gebruiken. Je mag de HTML ook uitschrijven
- Echter: ze besparen je typewerk in je *View*:
 - Het is altijd zinvol om te weten wat ze voor jou doen
- De *Tag Helpers* kom je o.a. veel tegen in *Views* die je via *Scaffolding* kan maken.
 - Sommige *Tag Helpers* doen veel voor jou, o.a. voor validatie in *formulieren*
 - Andere *Tag Helpers* doen eigenlijk vrij weinig
- N.B.: *HTML-Helpers* zijn een oudere manier om hetzelfde te doen
 - Het advies is om de modernere *Tag Helpers* te gebruiken als je kunt kiezen voor een *Tag Helper* i.p.v. voor een *HTML-Helper*

WAT ZIJN TAG HELPERS?

MSDN

- **Tag Helpers** genereren een stuk HTML-code met allerlei attributen voor je.
- Het is natuurlijk niet verplicht om *Tag Helpers* te gebruiken. Je mag de HTML ook uitschrijven
- Echter: ze besparen je typewerk in je *View*:
 - Het is altijd zinvol om te weten wat ze voor jou doen
- De *Tag Helpers* kom je o.a. veel tegen in *Views* die je via *Scaffolding* kan maken.
 - Sommige *Tag Helpers* doen veel voor jou, o.a. voor validatie in *formulieren*
 - Andere *Tag Helpers* doen eigenlijk vrij weinig
- N.B.: *HTML-Helpers* zijn een oudere manier om hetzelfde te doen
 - Het advies is om de modernere *Tag Helpers* te gebruiken als je kunt kiezen voor een *Tag Helper* i.p.v. voor een *HTML-Helper*

WAT ZIJN TAG HELPERS?

MSDN

- **Tag Helpers** genereren een stuk HTML-code met allerlei attributen voor je.
- Het is natuurlijk niet verplicht om *Tag Helpers* te gebruiken. Je mag de HTML ook uitschrijven
- Echter: ze besparen je typewerk in je *View*:
 - Het is altijd zinvol om te weten wat ze voor jou doen
- De *Tag Helpers* kom je o.a. veel tegen in *Views* die je via *Scaffolding* kan maken.
 - Sommige *Tag Helpers* doen veel voor jou, o.a. voor validatie in *formulieren*
 - Andere *Tag Helpers* doen eigenlijk vrij weinig
- N.B.: *HTML-Helpers* zijn een oudere manier om hetzelfde te doen
 - Het advies is om de modernere *Tag Helpers* te gebruiken als je kunt kiezen voor een *Tag Helper* i.p.v. voor een *HTML-Helper*

WAT ZIJN TAG HELPERS?

MSDN

- **Tag Helpers** genereren een stuk HTML-code met allerlei attributen voor je.
- Het is natuurlijk niet verplicht om *Tag Helpers* te gebruiken. Je mag de HTML ook uitschrijven
- Echter: ze besparen je typewerk in je *View*:
 - Het is altijd zinvol om te weten wat ze voor jou doen
- De *Tag Helpers* kom je o.a. veel tegen in *Views* die je via *Scaffolding* kan maken.
 - Sommige *Tag Helpers* doen veel voor jou, o.a. voor validatie in *formulieren*
 - Andere *Tag Helpers* doen eigenlijk vrij weinig
- N.B.: *HTML-Helpers* zijn een oudere manier om hetzelfde te doen
 - Het advies is om de modernere *Tag Helpers* te gebruiken als je kunt kiezen voor een *Tag Helper* i.p.v. voor een *HTML-Helper*

WAT ZIJN TAG HELPERS?

MSDN

- **Tag Helpers** genereren een stuk HTML-code met allerlei attributen voor je.
- Het is natuurlijk niet verplicht om *Tag Helpers* te gebruiken. Je mag de HTML ook uitschrijven
- Echter: ze besparen je typewerk in je *View*:
 - Het is altijd zinvol om te weten wat ze voor jou doen
- De *Tag Helpers* kom je o.a. veel tegen in *Views* die je via *Scaffolding* kan maken.
 - Sommige *Tag Helpers* doen veel voor jou, o.a. voor validatie in *formulieren*
 - Andere *Tag Helpers* doen eigenlijk vrij weinig
- N.B.: *HTML-Helpers* zijn een oudere manier om hetzelfde te doen
 - Het advies is om de modernere *Tag Helpers* te gebruiken als je kunt kiezen voor een *Tag Helper* i.p.v. voor een *HTML-Helper*

WAT ZIJN TAG HELPERS?

MSDN

- **Tag Helpers** genereren een stuk HTML-code met allerlei attributen voor je.
- Het is natuurlijk niet verplicht om *Tag Helpers* te gebruiken. Je mag de HTML ook uitschrijven
- Echter: ze besparen je typewerk in je *View*:
 - Het is altijd zinvol om te weten wat ze voor jou doen
- De *Tag Helpers* kom je o.a. veel tegen in *Views* die je via *Scaffolding* kan maken.
 - Sommige *Tag Helpers* doen veel voor jou, o.a. voor validatie in *formulieren*
 - Andere *Tag Helpers* doen eigenlijk vrij weinig
- N.B.: *HTML-Helpers* zijn een oudere manier om hetzelfde te doen
 - Het advies is om de modernere *Tag Helpers* te gebruiken als je kunt kiezen voor een *Tag Helper* i.p.v. voor een *HTML-Helper*

WAT ZIJN TAG HELPERS?

MSDN

- **Tag Helpers** genereren een stuk HTML-code met allerlei attributen voor je.
- Het is natuurlijk niet verplicht om *Tag Helpers* te gebruiken. Je mag de HTML ook uitschrijven
- Echter: ze besparen je typewerk in je *View*:
 - Het is altijd zinvol om te weten wat ze voor jou doen
- De *Tag Helpers* kom je o.a. veel tegen in *Views* die je via *Scaffolding* kan maken.
 - Sommige *Tag Helpers* doen veel voor jou, o.a. voor validatie in *formulieren*
 - Andere *Tag Helpers* doen eigenlijk vrij weinig
- N.B.: *HTML-Helpers* zijn een oudere manier om hetzelfde te doen
 - Het advies is om de modernere *Tag Helpers* te gebruiken als je kunt kiezen voor een *Tag Helper* i.p.v. voor een *HTML-Helper*

WAT ZIJN TAG HELPERS?

MSDN

- **Tag Helpers** genereren een stuk HTML-code met allerlei attributen voor je.
- Het is natuurlijk niet verplicht om *Tag Helpers* te gebruiken. Je mag de HTML ook uitschrijven
- Echter: ze besparen je typewerk in je *View*:
 - Het is altijd zinvol om te weten wat ze voor jou doen
- De *Tag Helpers* kom je o.a. veel tegen in *Views* die je via *Scaffolding* kan maken.
 - Sommige *Tag Helpers* doen veel voor jou, o.a. voor validatie in *formulieren*
 - Andere *Tag Helpers* doen eigenlijk vrij weinig
- N.B.: *HTML-Helpers* zijn een oudere manier om hetzelfde te doen
 - Het advies is om de modernere *Tag Helpers* te gebruiken als je kunt kiezen voor een *Tag Helper* i.p.v. voor een *HTML-Helper*

CLIENT-SIDE VALIDATIE

- Het doel van **validatie** is dat je alleen iets kunt opslaan als alle velden voldoen aan de regels:
 - Er moet een geldige datum worden ingevoerd volgens het template dd/mm/jjjj
 - Er mogen alleen letters (en geen cijfers) worden ingevoerd (of andersom)
- De *Tag Helpers* maken HTML5-attributen aan voor de *validatie*
- Zolang je geen goede invoer geeft, wordt na Submit geen *HTTP-Post Request* uitgevoerd
 - Check maar via de F12 in de netwerk tab
 - Deze *validatie* gebeurt in de browser (*client-side validatie*)
- De *Tag Helper* gebruikt *Data Annotaties* in de *Model* klassen voor de validatieregels
 - Ook de standaard foutmelding kun je aanpassen m.b.v. deze *Data Annotaties*
- Waarom is er *client-side validatie* EN *server-side validatie*?

CLIENT-SIDE VALIDATIE

- Het doel van **validatie** is dat je alleen iets kunt opslaan als alle velden voldoen aan de regels:
 - Er moet een geldige datum worden ingevoerd volgens het template dd/mm/jjjj
 - Er mogen alleen letters (en geen cijfers) worden ingevoerd (of andersom)
- De *Tag Helpers* maken HTML5-attributen aan voor de *validatie*
- Zolang je geen goede invoer geeft, wordt na Submit geen *HTTP-Post Request* uitgevoerd
 - Check maar via de F12 in de netwerk tab
 - Deze *validatie* gebeurt in de browser (*client-side validatie*)
- De *Tag Helper* gebruikt *Data Annotaties* in de *Model* klassen voor de validatieregels
 - Ook de standaard foutmelding kun je aanpassen m.b.v. deze *Data Annotaties*
- Waarom is er *client-side validatie* EN *server-side validatie*?

CLIENT-SIDE VALIDATIE

- Het doel van **validatie** is dat je alleen iets kunt opslaan als alle velden voldoen aan de regels:
 - Er moet een geldige datum worden ingevoerd volgens het template dd/mm/jjjj
 - Er mogen alleen letters (en geen cijfers) worden ingevoerd (of andersom)
- De *Tag Helpers* maken HTML5-attributen aan voor de *validatie*
- Zolang je geen goede invoer geeft, wordt na Submit geen *HTTP-Post Request* uitgevoerd
 - Check maar via de F12 in de netwerk tab
 - Deze *validatie* gebeurt in de browser (*client-side validatie*)
- De *Tag Helper* gebruikt *Data Annotaties* in de *Model* klassen voor de validatieregels
 - Ook de standaard foutmelding kun je aanpassen m.b.v. deze *Data Annotaties*
- Waarom is er *client-side validatie* EN *server-side validatie*?

CLIENT-SIDE VALIDATIE

- Het doel van **validatie** is dat je alleen iets kunt opslaan als alle velden voldoen aan de regels:
 - Er moet een geldige datum worden ingevoerd volgens het template dd/mm/jjjj
 - Er mogen alleen letters (en geen cijfers) worden ingevoerd (of andersom)
- De *Tag Helpers* maken HTML5-attributen aan voor de *validatie*
- Zolang je geen goede invoer geeft, wordt na Submit geen *HTTP-Post Request* uitgevoerd
 - Check maar via de F12 in de netwerk tab
 - Deze *validatie* gebeurt in de browser (*client-side validatie*)
- De *Tag Helper* gebruikt *Data Annotaties* in de *Model* klassen voor de validatieregels
 - Ook de standaard foutmelding kun je aanpassen m.b.v. deze *Data Annotaties*
- Waarom is er *client-side validatie* EN *server-side validatie*?

CLIENT-SIDE VALIDATIE

- Het doel van **validatie** is dat je alleen iets kunt opslaan als alle velden voldoen aan de regels:
 - Er moet een geldige datum worden ingevoerd volgens het template dd/mm/jjjj
 - Er mogen alleen letters (en geen cijfers) worden ingevoerd (of andersom)
- De *Tag Helpers* maken HTML5-attributen aan voor de *validatie*
- Zolang je geen goede invoer geeft, wordt na Submit geen *HTTP-Post Request* uitgevoerd
 - Check maar via de F12 in de netwerk tab
 - Deze *validatie* gebeurt in de browser (*client-side validatie*)
- De *Tag Helper* gebruikt *Data Annotaties* in de *Model* klassen voor de validatieregels
 - Ook de standaard foutmelding kun je aanpassen m.b.v. deze *Data Annotaties*
- Waarom is er *client-side validatie* EN *server-side validatie*?

CLIENT-SIDE VALIDATIE

- Het doel van **validatie** is dat je alleen iets kunt opslaan als alle velden voldoen aan de regels:
 - Er moet een geldige datum worden ingevoerd volgens het template dd/mm/jjjj
 - Er mogen alleen letters (en geen cijfers) worden ingevoerd (of andersom)
- De *Tag Helpers* maken HTML5-attributen aan voor de *validatie*
- Zolang je geen goede invoer geeft, wordt na Submit geen *HTTP-Post Request* uitgevoerd
 - Check maar via de F12 in de netwerk tab
 - Deze *validatie* gebeurt in de browser (*client-side validatie*)
- De *Tag Helper* gebruikt *Data Annotaties* in de *Model* klassen voor de validatieregels
 - Ook de standaard foutmelding kun je aanpassen m.b.v. deze *Data Annotaties*
- Waarom is er *client-side validatie* EN *server-side validatie*?

CLIENT-SIDE VALIDATIE

- Het doel van **validatie** is dat je alleen iets kunt opslaan als alle velden voldoen aan de regels:
 - Er moet een geldige datum worden ingevoerd volgens het template dd/mm/jjjj
 - Er mogen alleen letters (en geen cijfers) worden ingevoerd (of andersom)
- De *Tag Helpers* maken HTML5-attributen aan voor de *validatie*
- Zolang je geen goede invoer geeft, wordt na Submit geen *HTTP-Post Request* uitgevoerd
 - Check maar via de F12 in de netwerk tab
 - Deze *validatie* gebeurt in de browser (*client-side validatie*)
- De *Tag Helper* gebruikt *Data Annotaties* in de *Model* klassen voor de validatieregels
 - Ook de standaard foutmelding kun je aanpassen m.b.v. deze *Data Annotaties*
- Waarom is er *client-side validatie* EN *server-side validatie*?

CLIENT-SIDE VALIDATIE

- Het doel van **validatie** is dat je alleen iets kunt opslaan als alle velden voldoen aan de regels:
 - Er moet een geldige datum worden ingevoerd volgens het template dd/mm/jjjj
 - Er mogen alleen letters (en geen cijfers) worden ingevoerd (of andersom)
- De *Tag Helpers* maken HTML5-attributen aan voor de *validatie*
- Zolang je geen goede invoer geeft, wordt na Submit geen *HTTP-Post Request* uitgevoerd
 - Check maar via de F12 in de netwerk tab
 - Deze *validatie* gebeurt in de browser (*client-side validatie*)
- De *Tag Helper* gebruikt *Data Annotaties* in de *Model* klassen voor de validatieregels
 - Ook de standaard foutmelding kun je aanpassen m.b.v. deze *Data Annotaties*
- Waarom is er *client-side validatie* EN *server-side validatie*?

CLIENT-SIDE VALIDATIE

- Het doel van **validatie** is dat je alleen iets kunt opslaan als alle velden voldoen aan de regels:
 - Er moet een geldige datum worden ingevoerd volgens het template dd/mm/jjjj
 - Er mogen alleen letters (en geen cijfers) worden ingevoerd (of andersom)
- De *Tag Helpers* maken HTML5-attributen aan voor de *validatie*
- Zolang je geen goede invoer geeft, wordt na Submit geen *HTTP-Post Request* uitgevoerd
 - Check maar via de F12 in de netwerk tab
 - Deze *validatie* gebeurt in de browser (*client-side validatie*)
- De *Tag Helper* gebruikt *Data Annotaties* in de *Model* klassen voor de validatieregels
 - Ook de standaard foutmelding kun je aanpassen m.b.v. deze *Data Annotaties*
- Waarom is er *client-side validatie* EN *server-side validatie*?

CLIENT-SIDE VALIDATIE

- Het doel van **validatie** is dat je alleen iets kunt opslaan als alle velden voldoen aan de regels:
 - Er moet een geldige datum worden ingevoerd volgens het template dd/mm/jjjj
 - Er mogen alleen letters (en geen cijfers) worden ingevoerd (of andersom)
- De *Tag Helpers* maken HTML5-attributen aan voor de *validatie*
- Zolang je geen goede invoer geeft, wordt na Submit geen *HTTP-Post Request* uitgevoerd
 - Check maar via de F12 in de netwerk tab
 - Deze *validatie* gebeurt in de browser (*client-side validatie*)
- De *Tag Helper* gebruikt *Data Annotaties* in de *Model* klassen voor de validatieregels
 - Ook de standaard foutmelding kun je aanpassen m.b.v. deze *Data Annotaties*
- Waarom is er *client-side validatie* EN *server-side validatie*?

SCAFFOLDEN

WAT IS CODE SCAFFOLDEN?

- **Scaffolding** gaan toepassen, dit is:
 - *Formulieren* met CRUD functionaliteit laten genereren
 - De *Controller* met database acties laten genereren
 - De bijbehorende *Views* worden aangemaakt
 - En dit alles op basis van het *Model* en je *DbContext*
- Dit kun je doen voor een tabel maar ook relaties tussen tabellen kun je scaffolden
- De gegenereerde code moet je kunnen begrijpen
- Soms wordt er enge code gegenereerd... Wat is **async** en **await**?

WAT IS CODE SCAFFOLDEN?

- **Scaffolding** gaan toepassen, dit is:
 - *Formulieren* met CRUD functionaliteit laten genereren
 - De *Controller* met database acties laten genereren
 - De bijbehorende *Views* worden aangemaakt
 - En dit alles op basis van het *Model* en je *DbContext*
- Dit kun je doen voor een tabel maar ook relaties tussen tabellen kun je scaffolden
- De gegenereerde code moet je kunnen begrijpen
- Soms wordt er enge code gegenereerd... Wat is **async** en **await**?

WAT IS CODE SCAFFOLDEN?

- **Scaffolding** gaan toepassen, dit is:
 - *Formulieren* met CRUD functionaliteit laten genereren
 - De *Controller* met database acties laten genereren
 - De bijbehorende *Views* worden aangemaakt
 - En dit alles op basis van het *Model* en je *DbContext*
- Dit kun je doen voor een tabel maar ook relaties tussen tabellen kun je scaffolden
- De gegenereerde code moet je kunnen begrijpen
- Soms wordt er enge code gegenereerd... Wat is **async** en **await**?

WAT IS CODE SCAFFOLDEN?

- **Scaffolding** gaan toepassen, dit is:
 - *Formulieren* met CRUD functionaliteit laten genereren
 - De *Controller* met database acties laten genereren
 - De bijbehorende *Views* worden aangemaakt
 - En dit alles op basis van het *Model* en je *DbContext*
- Dit kun je doen voor een tabel maar ook relaties tussen tabellen kun je scaffolden
- De gegenereerde code moet je kunnen begrijpen
- Soms wordt er enge code gegenereerd... Wat is **async** en **await**?

WAT IS CODE SCAFFOLDEN?

- **Scaffolding** gaan toepassen, dit is:
 - *Formulieren* met CRUD functionaliteit laten genereren
 - De *Controller* met database acties laten genereren
 - De bijbehorende *Views* worden aangemaakt
 - En dit alles op basis van het *Model* en je *DbContext*
- Dit kun je doen voor een tabel maar ook relaties tussen tabellen kun je scaffolden
- De gegenereerde code moet je kunnen begrijpen
- Soms wordt er enge code gegenereerd... Wat is **async** en **await**?

WAT IS CODE SCAFFOLDEN?

- **Scaffolding** gaan toepassen, dit is:
 - *Formulieren* met CRUD functionaliteit laten genereren
 - De *Controller* met database acties laten genereren
 - De bijbehorende *Views* worden aangemaakt
 - En dit alles op basis van het *Model* en je *DbContext*
- Dit kun je doen voor een tabel maar ook relaties tussen tabellen kun je scaffolden
- De gegenereerde code moet je kunnen begrijpen
- Soms wordt er enge code gegenereerd... Wat is **async** en **await**?

WAT IS CODE SCAFFOLDEN?

- **Scaffolding** gaan toepassen, dit is:
 - *Formulieren* met CRUD functionaliteit laten genereren
 - De *Controller* met database acties laten genereren
 - De bijbehorende *Views* worden aangemaakt
 - En dit alles op basis van het *Model* en je *DbContext*
- Dit kun je doen voor een tabel maar ook relaties tussen tabellen kun je scaffolden
- De gegenereerde code moet je kunnen begrijpen
- Soms wordt er enge code gegenereerd... Wat is *async* en *await*?

WAT IS CODE SCAFFOLDEN?

- **Scaffolding** gaan toepassen, dit is:
 - *Formulieren* met CRUD functionaliteit laten genereren
 - De *Controller* met database acties laten genereren
 - De bijbehorende *Views* worden aangemaakt
 - En dit alles op basis van het *Model* en je *DbContext*
- Dit kun je doen voor een tabel maar ook relaties tussen tabellen kun je scaffolden
- De gegenereerde code moet je kunnen begrijpen
- Soms wordt er enge code gegenereerd... Wat is **async** en **await**?

HOE SCAFFOLDEN WE?

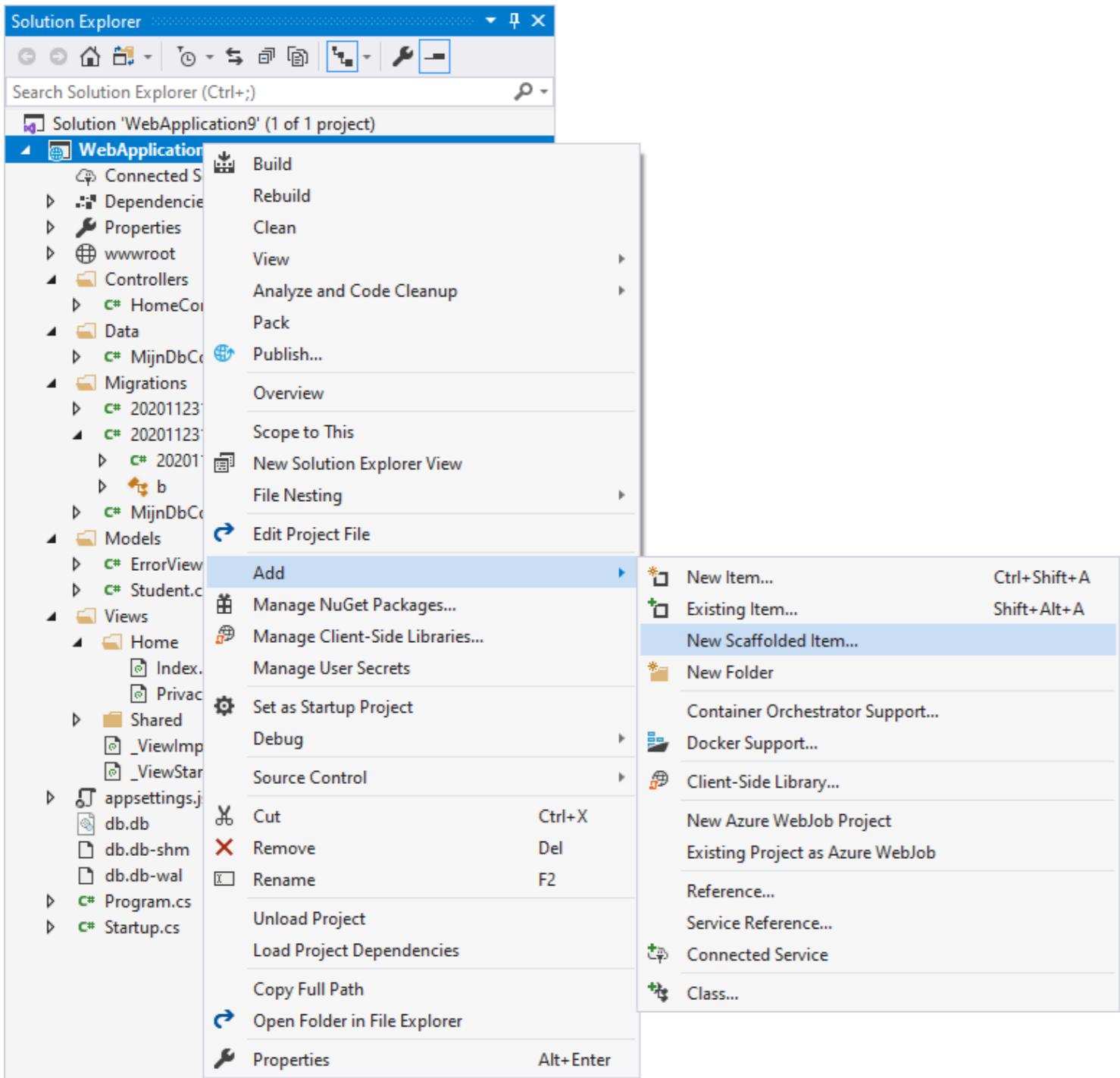
In  Visual Studio:

In  VS Code

```
dotnet tool install -g dotnet-aspnet-codegenerator  
dotnet aspnet-codegenerator controller  
  -name StudentsController  
  -m Student  
  -dc MijnDbContext  
  --relativeFolderPath Controllers  
  --useDefaultLayout
```

HOE SCAFFOLDEN WE?

In  Visual Studio:

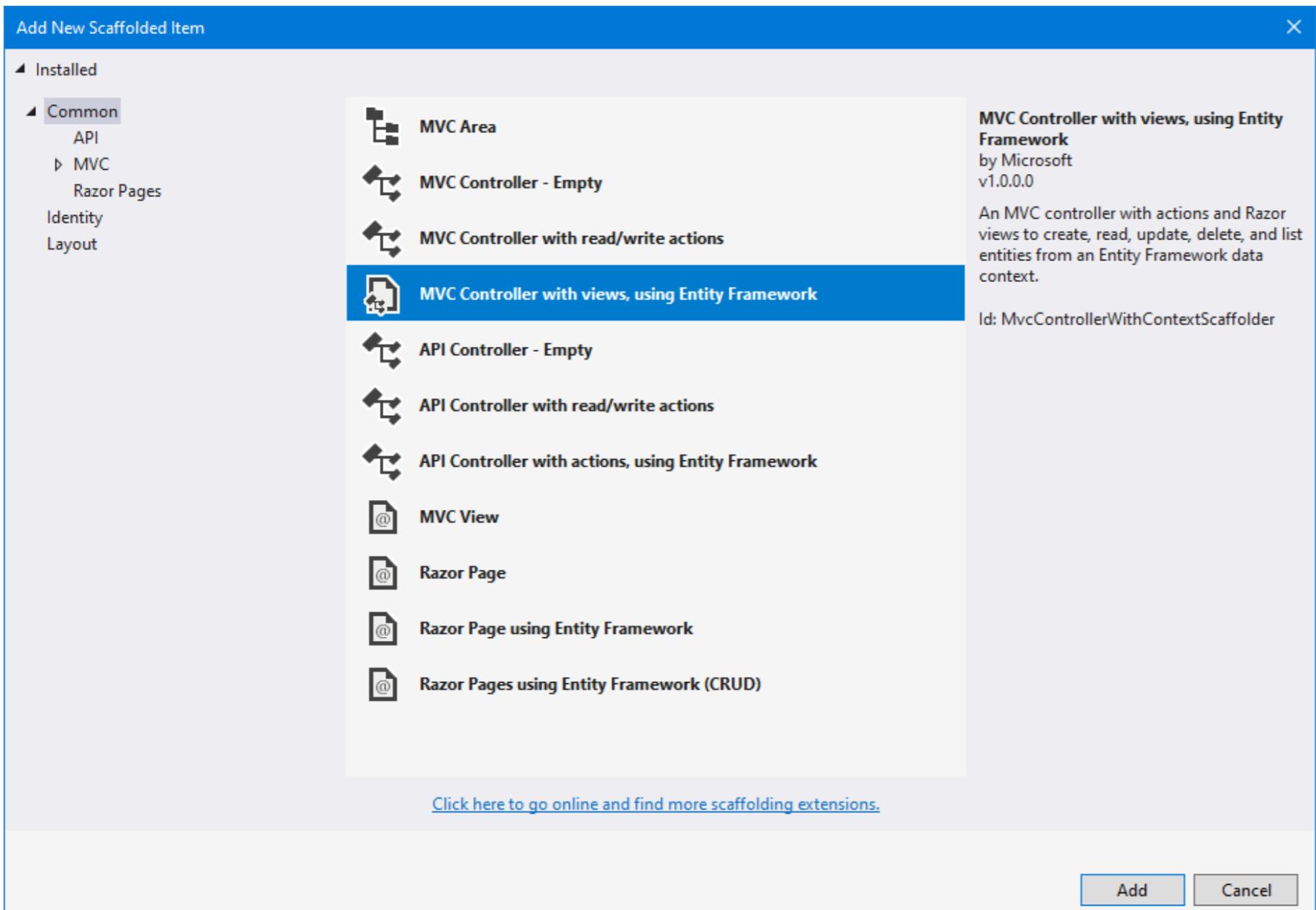


In  VS Code

```
dotnet tool install -g dotnet-aspnet-codegenerator  
dotnet aspnet-codegenerator controller  
-name StudentsController  
-m Student  
-dc MijnDbContext  
--relativeFolderPath Controllers  
--useDefaultLayout
```

HOE SCAFFOLDEN WE?

In  Visual Studio:

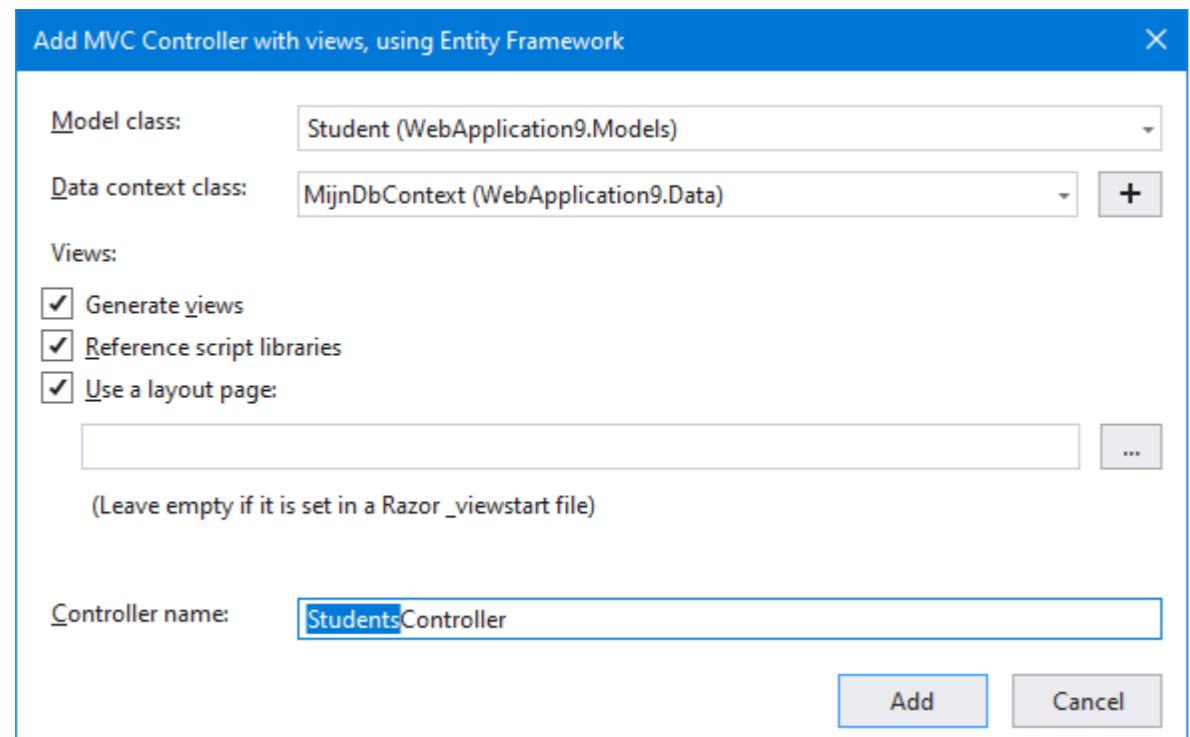


In  VS Code

```
dotnet tool install -g dotnet-aspnet-codegenerator  
dotnet aspnet-codegenerator controller  
-name StudentsController  
-m Student  
-dc MijnDbContext  
--relativeFolderPath Controllers  
--useDefaultLayout
```

HOE SCAFFOLDEN WE?

In  Visual Studio:



In  VS Code

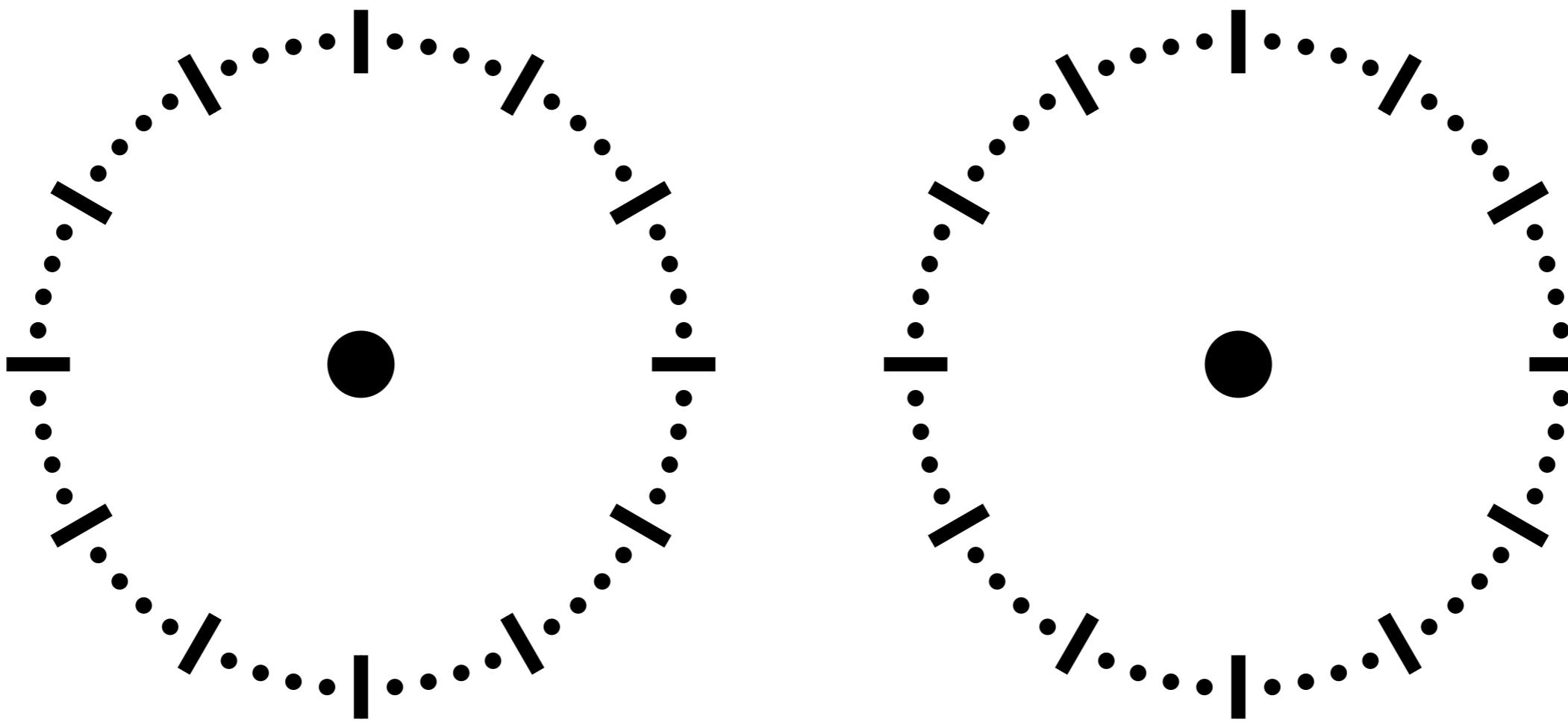
```
dotnet tool install -g dotnet-aspnet-codegenerator  
dotnet aspnet-codegenerator controller  
-name StudentsController  
-m Student  
-dc MijnDbContext  
--relativeFolderPath Controllers  
--useDefaultLayout
```

DEMO

Probeer iedere letter code begrijpen.

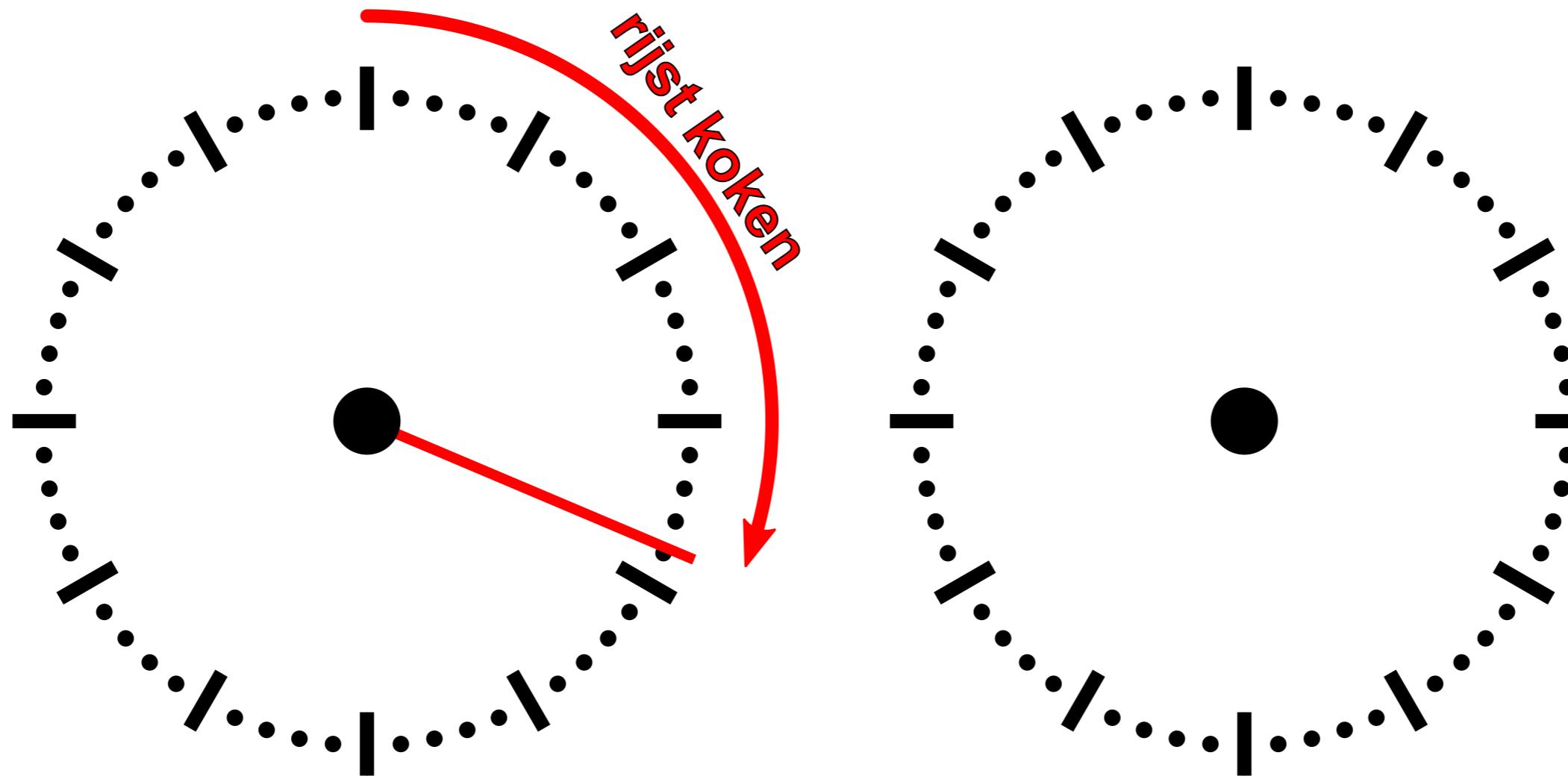
ASYNC

ASYNCHROON



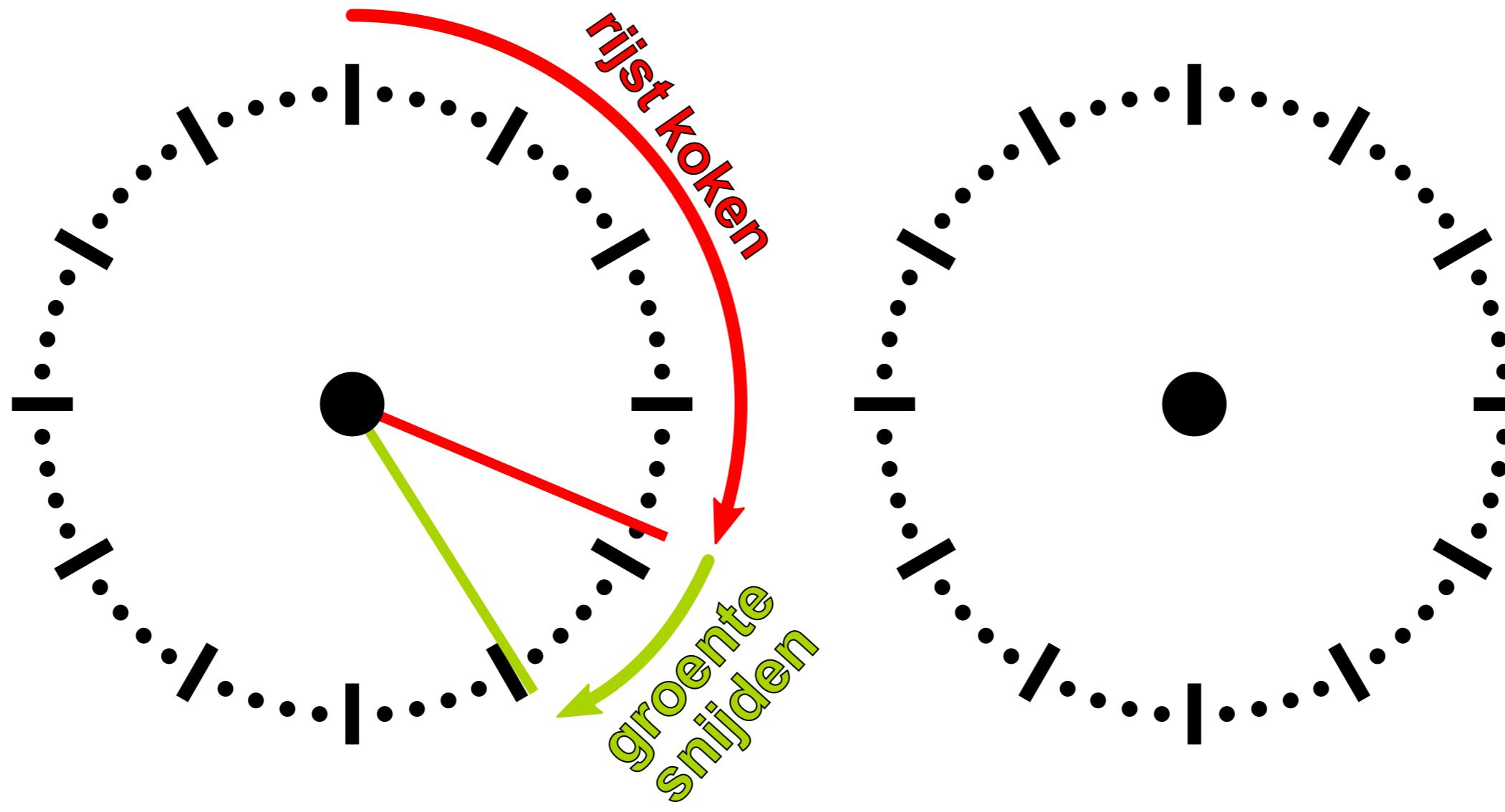
- de term *asynchroon* is misschien tegenintuitief ([dit helpt](#))
- meerdere CPU kernen / meerdere threads (hyperthreading)
- Bijvoorbeeld: de spellingchecker runt parallel in Word

ASYNCHROON



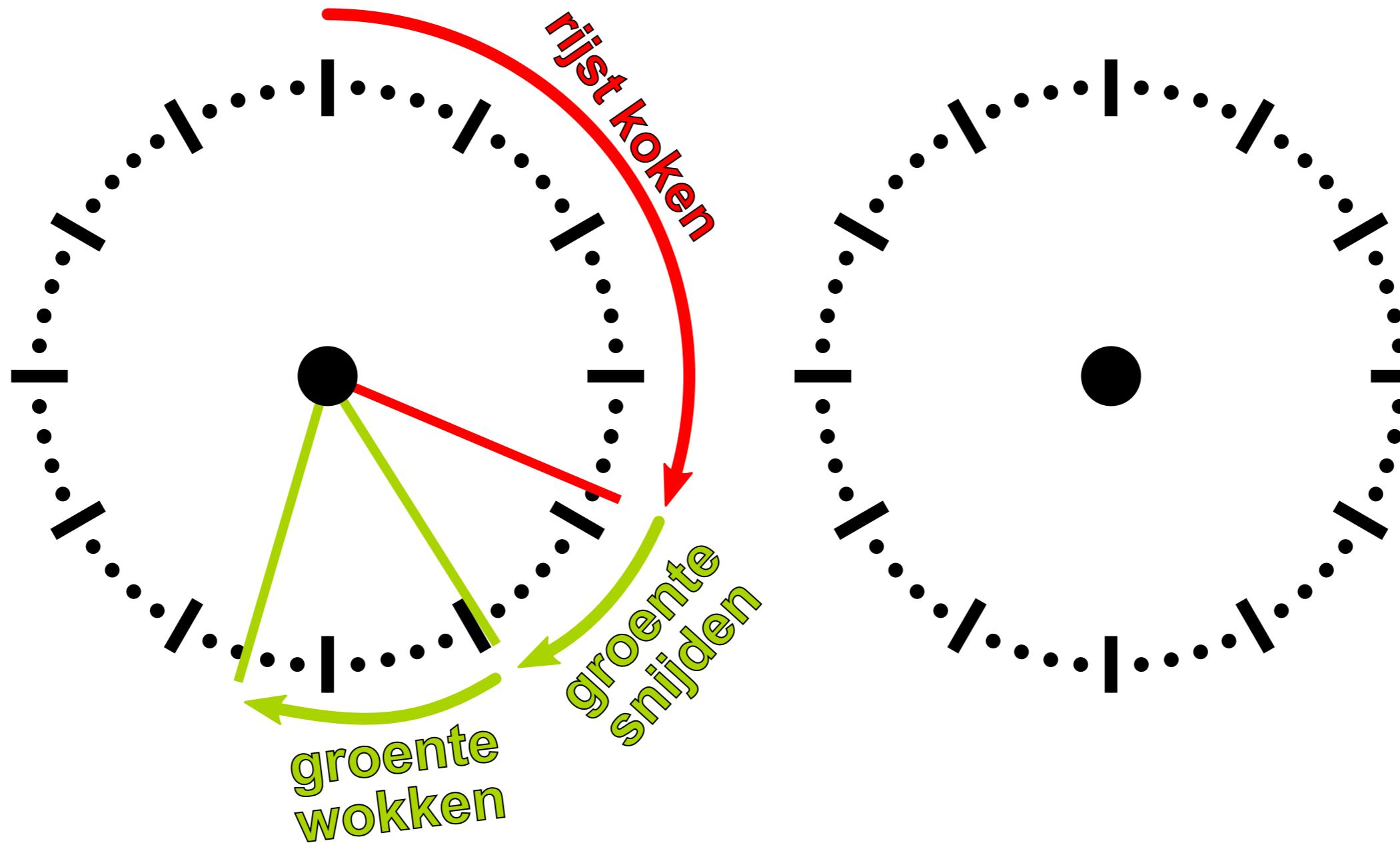
- de term *asynchroon* is misschien tegenintuitief ([dit helpt](#))
- meerdere CPU kernen / meerdere threads (hyperthreading)
- Bijvoorbeeld: de spellingchecker runt parallel in Word

ASYNCHROON



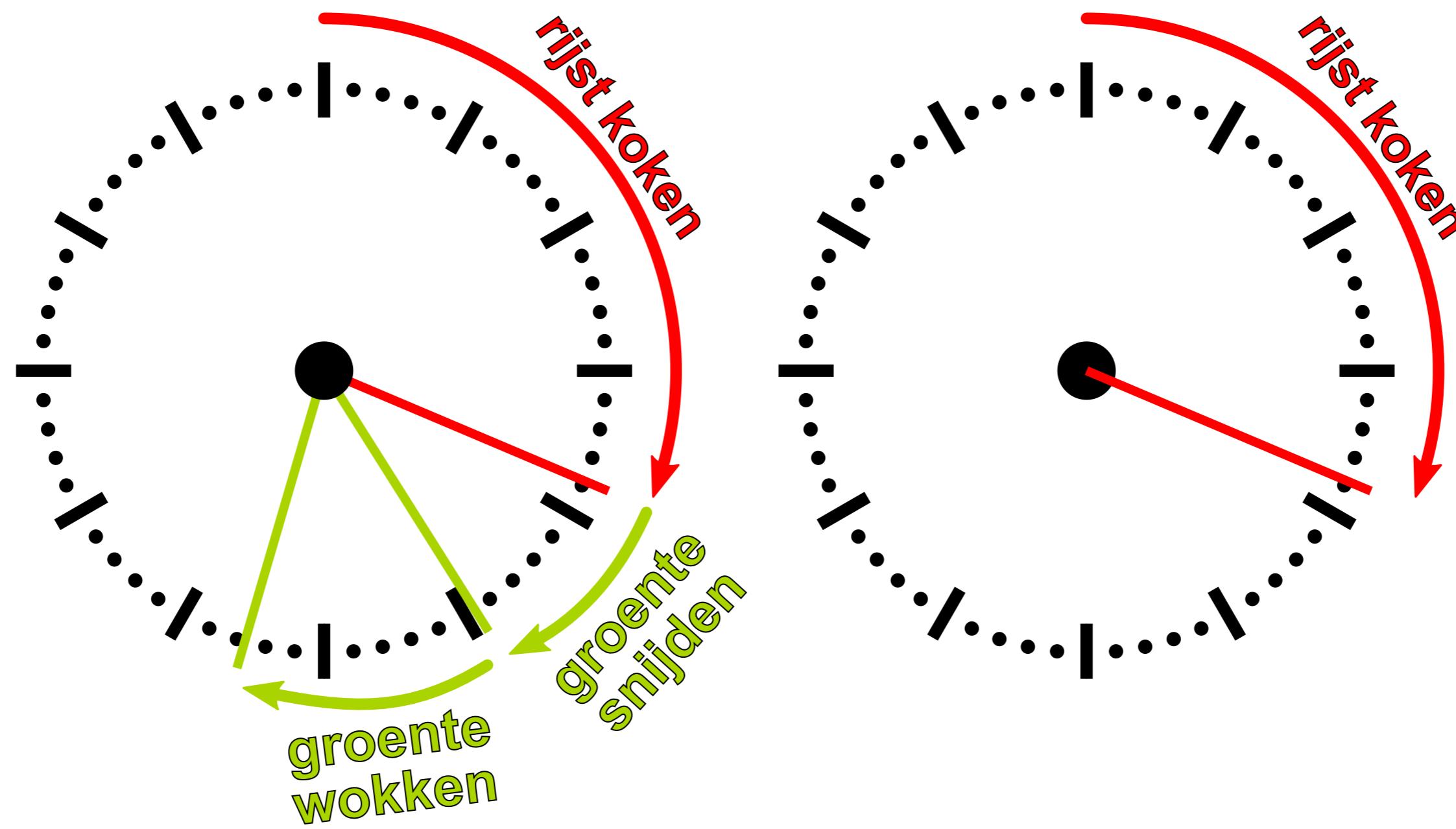
- de term *asynchroon* is misschien tegenintuitief ([dit helpt](#))
- meerdere CPU kernen / meerdere threads (hyperthreading)
- Bijvoorbeeld: de spellingchecker runt parallel in Word

ASYNCHROON



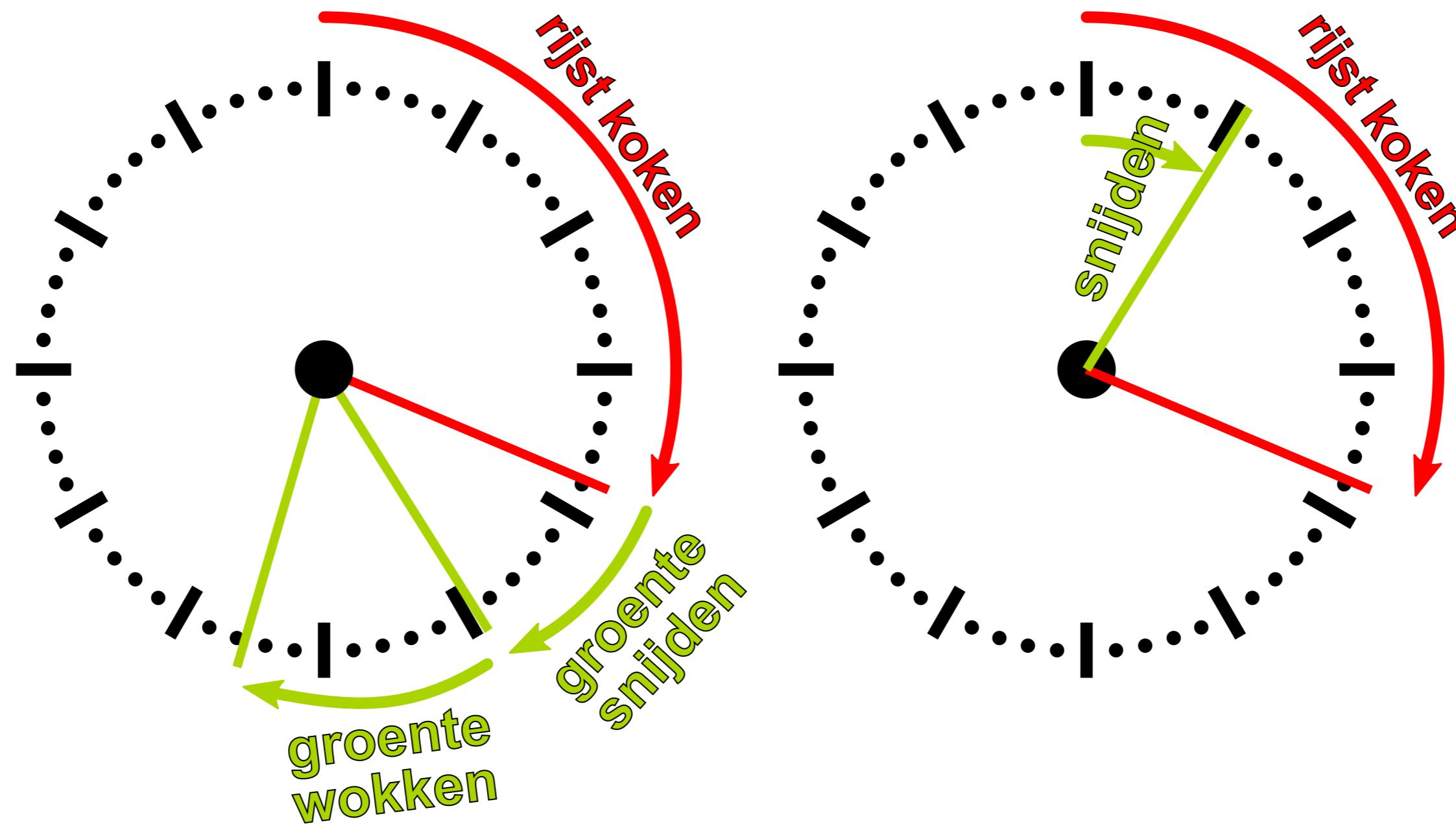
- de term *asynchroon* is misschien tegenintuitief ([dit helpt](#))
- meerdere CPU kernen / meerdere threads (hyperthreading)
- Bijvoorbeeld: de spellingchecker runt parallel in Word

ASYNCHROON



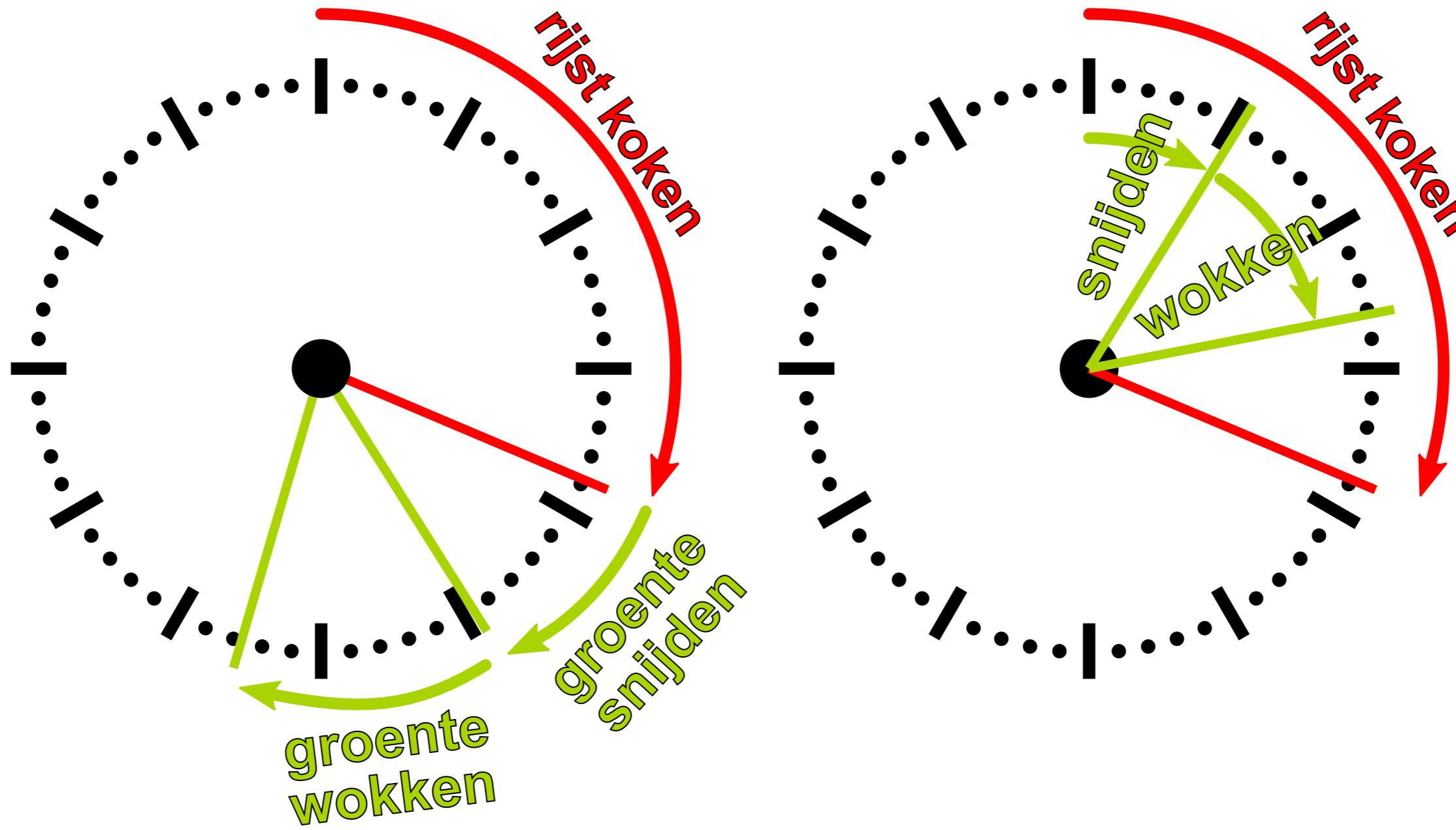
- de term *asynchroon* is misschien tegenintuitief ([dit helpt](#))
- meerdere CPU kernen / meerdere threads (hyperthreading)
- Bijvoorbeeld: de spellingchecker runt parallel in Word

ASYNCHROON



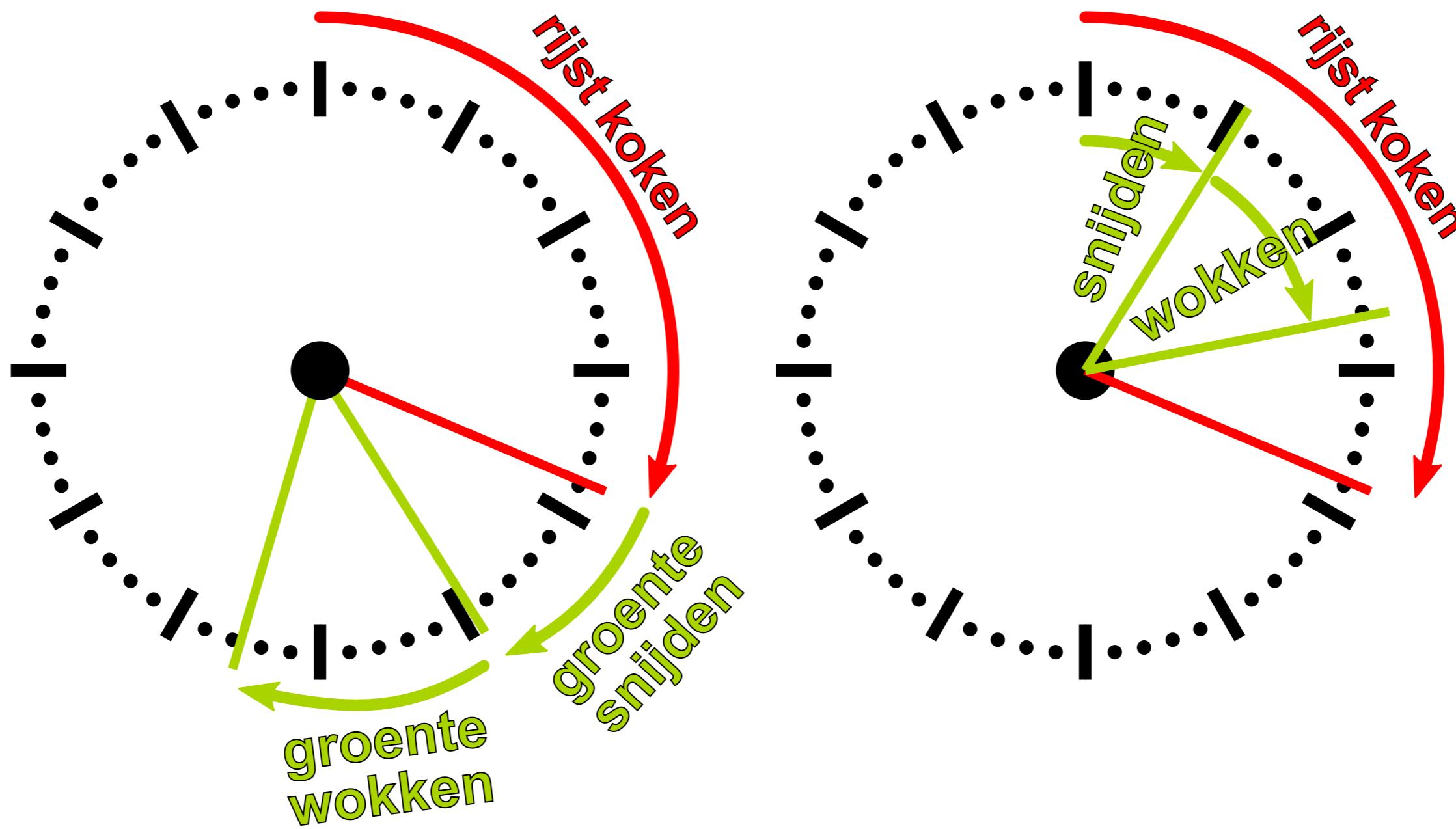
- de term *asynchroon* is misschien tegenintuitief ([dit helpt](#))
- meerdere CPU kernen / meerdere threads (hyperthreading)
- Bijvoorbeeld: de spellingchecker runt parallel in Word

ASYNCHROON



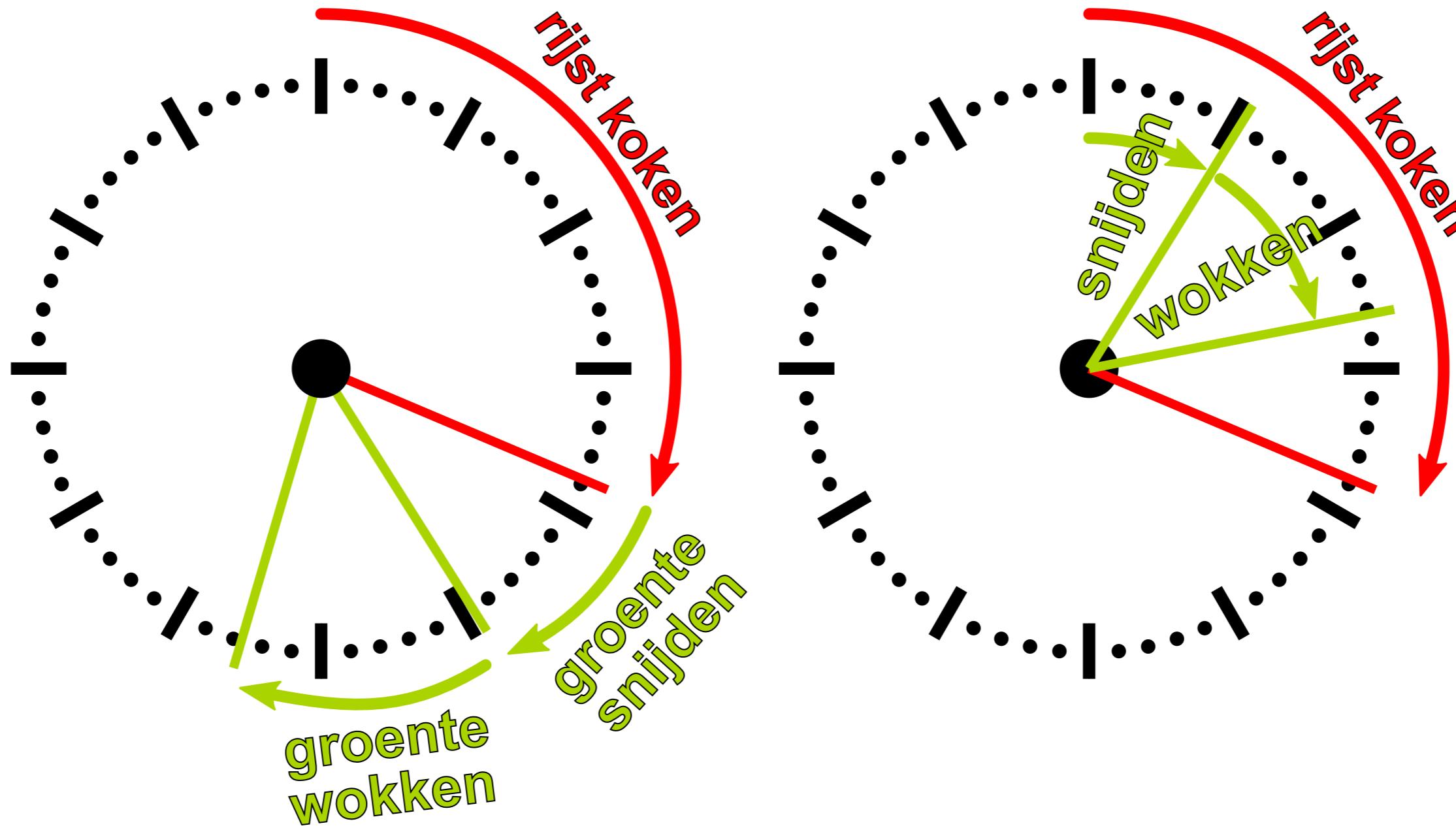
- de term *asynchroon* is misschien tegenintuitief ([dit helpt](#))
- meerdere CPU kernen / meerdere threads (hyperthreading)
- Bijvoorbeeld: de spellingchecker runt parallel in Word

ASYNCHROON



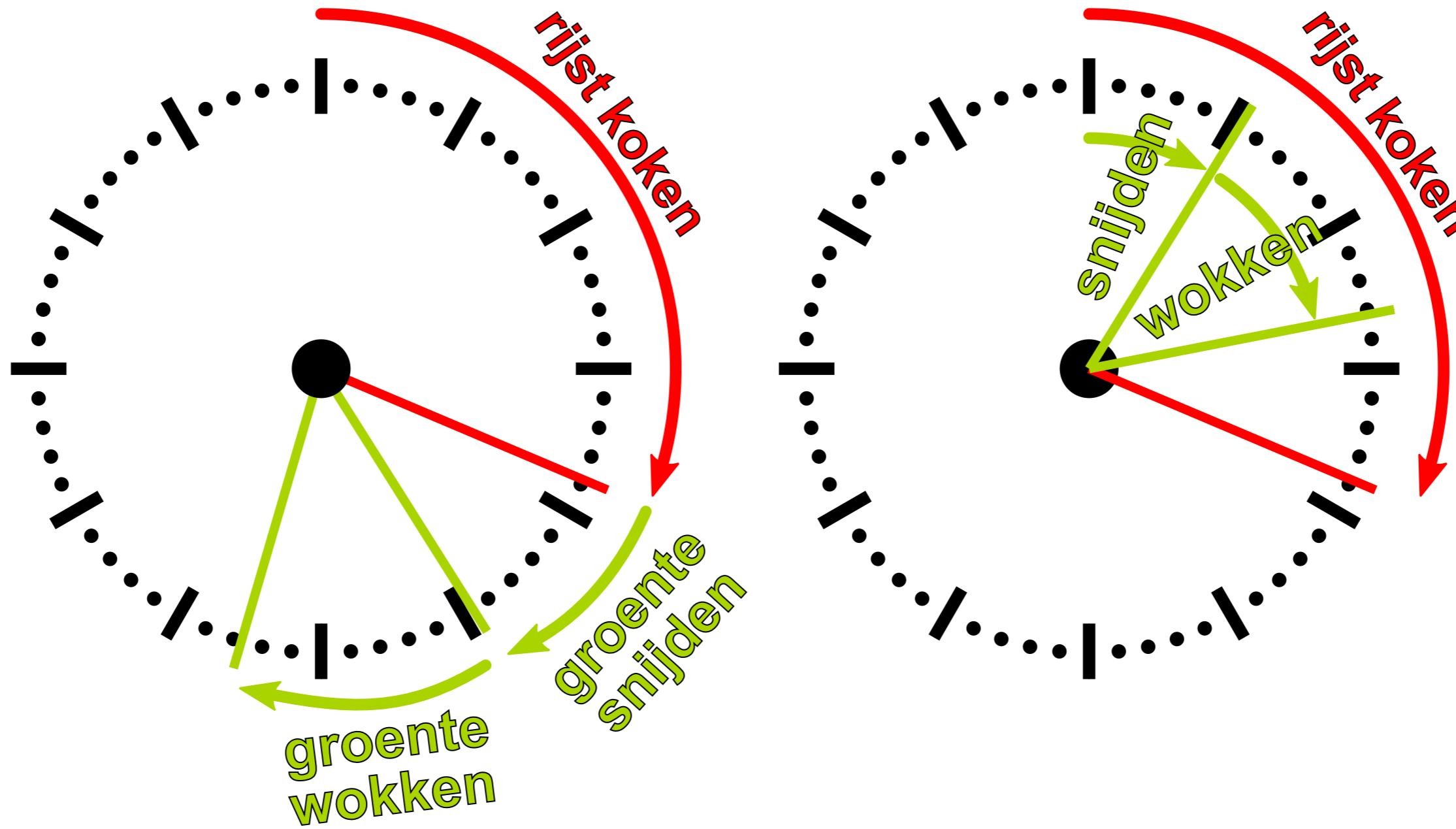
- de term *asynchroon* is misschien tegenintuitief (**dit** helpt)
- meerdere CPU kernen / meerdere threads (hyperthreading)
- Bijvoorbeeld: de spellingchecker runt parallel in Word

ASYNCHROON



- de term *asynchroon* is misschien tegenintuitief ([dit](#) helpt)
- meerdere CPU kernen / meerdere threads (hyperthreading)
- Bijvoorbeeld: de spellingchecker runt parallel in Word

ASYNCHROON



- de term *asynchroon* is misschien tegenintuitief ([dit](#) helpt)
- meerdere CPU kernen / meerdere threads (hyperthreading)
- Bijvoorbeeld: de spellingchecker runt parallel in Word

WAAR KOM JE ASYNC TEGEN?

- SaveChangesAsync
- ToListAsync
- async Task<IActionResult>

VOORBEELD 1

```
public static async Task methode(int arg) {
    Console.WriteLine("'methode' begint");
    var wacht = Task.Delay(1000);
    Console.WriteLine("'wacht' begonnen");
    await wacht;
    Console.WriteLine("'wacht' klaar");
    Console.WriteLine("'methode' klaar");
}

public static async Task Main(string[] args)
{
    Console.WriteLine("'main' begint");
    var taak = methode(2);
    Console.WriteLine("'main' verder");
    await taak;
    Console.WriteLine("'main' klaar");
    Console.ReadLine();
}
```

VOORBEELD 2

code

ASYNC IS INGEWIKKELD

- Hoe werken exceptions?
- Locks? Deadlocks?
- Moet alles async worden?

OPDRACHT

- Maak een model klasse Student met een Id, 6 properties en een extra property "List<Student> Vrienden".
- Voor elk van de 6 properties: voeg een data-attribute toe waarop gevalideerd kan worden client-side.
- Scaffold de controller en views voor de CRUD operaties. Sla de studenten op in een SQLite database (of eventueel SQL Server).
- Zorg dat er client-side en server-side validatie is.
- In de footer van alle webpagina's wil ik 3 studenten zien die voldoen aan een zelf gekozen criterium (bijv. studenten met het hoogste cijfer). Maak in je Shared View gebruik van een Partial View. Aan de Partial View geef je een IEnumerable van maximaal 3 items mee. Hint: gebruik de LINQ method Take (3).
 - Zorg dat je van alle gesccaffolde code begrijpt wat het betekent.
 - Bestudeer de gegeneerde HTML van het formulier met Tag Helpers.

OVERIG: VIEWDATA VERSUS VIEWBAG

- **ViewData** is een Dictionary Object (die je kunt lezen en schrijven)
- **ViewBag** is een Wrapper die verwijst naar **ViewData** (NB ViewBag zelf kun je dus geen waarde geven)
 - Het is een heel bijzonder object dat dynamisch eigen properties aanmaakt
 - Omdat ViewBag een wrapper is om ViewData heen, verwijzen ze naar dezelfde content:
 - `ViewData["Message"] = "text";` vs. `ViewBag.Message = "text";`
- Weet je nog? Als je objecten in **ViewData** stopt, moet je casten om de waarde te kunnen lezen
 - Dat is bij **ViewBag** niet meer nodig (**ViewBag** organiseert dit zelf)
 - `Student s = (Student) ViewData["BesteStudent"];`
 - `Student s = ViewBag.BesteStudent;`

OVERIG: VIEWDATA VERSUS VIEWBAG

- ViewData is een Dictionary Object (die je kunt lezen en schrijven)
- ViewBag is een Wrapper die verwijst naar ViewData (NB ViewBag zelf kunt je dus geen waarde geven)
 - Het is een heel bijzonder object dat dynamisch eigen properties aanmaakt
 - Omdat ViewBag een wrapper is om ViewData heen, verwijzen ze naar dezelfde content:
 - ViewData["Message"] = "text"; vs. ViewBag.Message = "text";
- Weet je nog? Als je objecten in ViewData stopt, moet je casten om de waarde te kunnen lezen
 - Dat is bij ViewBag niet meer nodig (ViewBag organiseert dit zelf)
 - Student s = (Student) ViewData["BesteStudent"];
 - Student s = ViewBag.BesteStudent;

OVERIG: VIEWDATA VERSUS VIEWBAG

- ViewData is een Dictionary Object (die je kunt lezen en schrijven)
- ViewBag is een Wrapper die verwijst naar ViewData (NB ViewBag zelf kun je dus geen waarde geven)
 - Het is een heel bijzonder object dat dynamisch eigen properties aanmaakt
 - Omdat ViewBag een wrapper is om ViewData heen, verwijzen ze naar dezelfde content:
 - ViewData["Message"] = "text"; vs. ViewBag.Message = "text";
- Weet je nog? Als je objecten in ViewData stopt, moet je casten om de waarde te kunnen lezen
 - Dat is bij ViewBag niet meer nodig (ViewBag organiseert dit zelf)
 - Student s = (Student) ViewData["BesteStudent"];
 - Student s = ViewBag.BesteStudent;

OVERIG: VIEWDATA VERSUS VIEWBAG

- ViewData is een Dictionary Object (die je kunt lezen en schrijven)
- ViewBag is een Wrapper die verwijst naar ViewData (NB ViewBag zelf kun je dus geen waarde geven)
 - Het is een heel bijzonder object dat dynamisch eigen properties aanmaakt
 - Omdat ViewBag een wrapper is om ViewData heen, verwijzen ze naar dezelfde content:
 - ViewData["Message"] = "text"; vs. ViewBag.Message = "text";
- Weet je nog? Als je objecten in ViewData stopt, moet je casten om de waarde te kunnen lezen
 - Dat is bij ViewBag niet meer nodig (ViewBag organiseert dit zelf)
 - Student s = (Student) ViewData["BesteStudent"];
 - Student s = ViewBag.BesteStudent;

OVERIG: VIEWDATA VERSUS VIEWBAG

- ViewData is een Dictionary Object (die je kunt lezen en schrijven)
- ViewBag is een Wrapper die verwijst naar ViewData (NB ViewBag zelf kun je dus geen waarde geven)
 - Het is een heel bijzonder object dat dynamisch eigen properties aanmaakt
 - Omdat ViewBag een wrapper is om ViewData heen, verwijzen ze naar dezelfde content:
 - `ViewData["Message"] = "text";` vs. `ViewBag.Message = "text";`
- Weet je nog? Als je objecten in ViewData stopt, moet je casten om de waarde te kunnen lezen
 - Dat is bij ViewBag niet meer nodig (ViewBag organiseert dit zelf)
 - `Student s = (Student) ViewData["BesteStudent"];`
 - `Student s = ViewBag.BesteStudent;`

OVERIG: VIEWDATA VERSUS VIEWBAG

- ViewData is een Dictionary Object (die je kunt lezen en schrijven)
- ViewBag is een Wrapper die verwijst naar ViewData (NB ViewBag zelf kun je dus geen waarde geven)
 - Het is een heel bijzonder object dat dynamisch eigen properties aanmaakt
 - Omdat ViewBag een wrapper is om ViewData heen, verwijzen ze naar dezelfde content:
 - `ViewData["Message"] = "text";` vs. `ViewBag.Message = "text";`
- Weet je nog? Als je objecten in ViewData stopt, moet je casten om de waarde te kunnen lezen
 - Dat is bij ViewBag niet meer nodig (ViewBag organiseert dit zelf)
 - `Student s = (Student) ViewData["BesteStudent"];`
 - `Student s = ViewBag.BesteStudent;`

OVERIG: VIEWDATA VERSUS VIEWBAG

- ViewData is een Dictionary Object (die je kunt lezen en schrijven)
- ViewBag is een Wrapper die verwijst naar ViewData (NB ViewBag zelf kun je dus geen waarde geven)
 - Het is een heel bijzonder object dat dynamisch eigen properties aanmaakt
 - Omdat ViewBag een wrapper is om ViewData heen, verwijzen ze naar dezelfde content:
 - `ViewData["Message"] = "text";` vs. `ViewBag.Message = "text";`
- Weet je nog? Als je objecten in ViewData stopt, moet je casten om de waarde te kunnen lezen
 - Dat is bij ViewBag niet meer nodig (ViewBag organiseert dit zelf)
 - `Student s = (Student) ViewData["BesteStudent"];`
 - `Student s = ViewBag.BesteStudent;`

OVERIG: VIEWDATA VERSUS VIEWBAG

- ViewData is een Dictionary Object (die je kunt lezen en schrijven)
- ViewBag is een Wrapper die verwijst naar ViewData (NB ViewBag zelf kun je dus geen waarde geven)
 - Het is een heel bijzonder object dat dynamisch eigen properties aanmaakt
 - Omdat ViewBag een wrapper is om ViewData heen, verwijzen ze naar dezelfde content:
 - `ViewData["Message"] = "text";` vs. `ViewBag.Message = "text";`
- Weet je nog? Als je objecten in ViewData stopt, moet je casten om de waarde te kunnen lezen
 - Dat is bij ViewBag niet meer nodig (ViewBag organiseert dit zelf)
 - `Student s = (Student) ViewData["BesteStudent"];`
 - `Student s = ViewBag.BesteStudent;`

OVERIG: VIEWDATA VERSUS VIEWBAG

- ViewData is een Dictionary Object (die je kunt lezen en schrijven)
- ViewBag is een Wrapper die verwijst naar ViewData (NB ViewBag zelf kun je dus geen waarde geven)
 - Het is een heel bijzonder object dat dynamisch eigen properties aanmaakt
 - Omdat ViewBag een wrapper is om ViewData heen, verwijzen ze naar dezelfde content:
 - `ViewData["Message"] = "text";` vs. `ViewBag.Message = "text";`
- Weet je nog? Als je objecten in ViewData stopt, moet je casten om de waarde te kunnen lezen
 - Dat is bij ViewBag niet meer nodig (ViewBag organiseert dit zelf)
 - `Student s = (Student) ViewData["BesteStudent"];`
 - `Student s = ViewBag.BesteStudent;`