Creating Interceptors

Brice Wilson

@brice_wilson www.BriceWilson.net

What Are Interceptors?

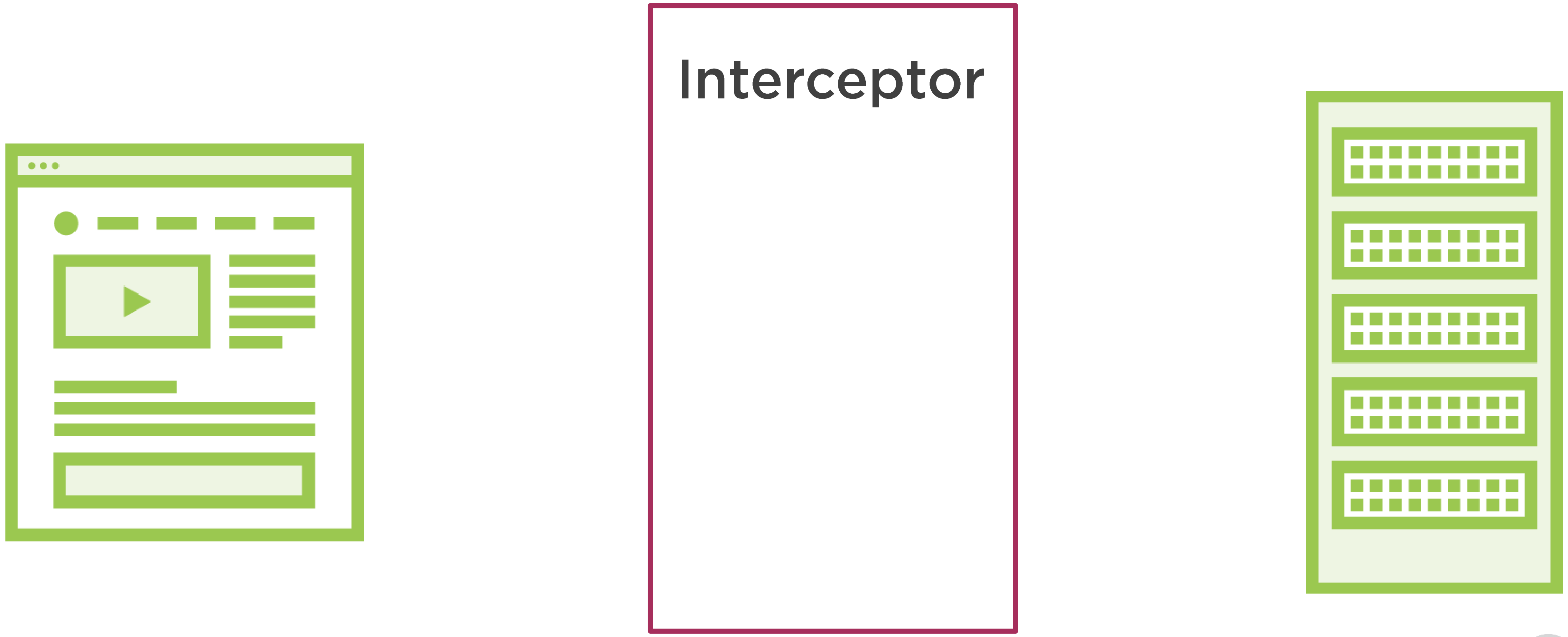
Services

Implement the HttpInterceptor interface

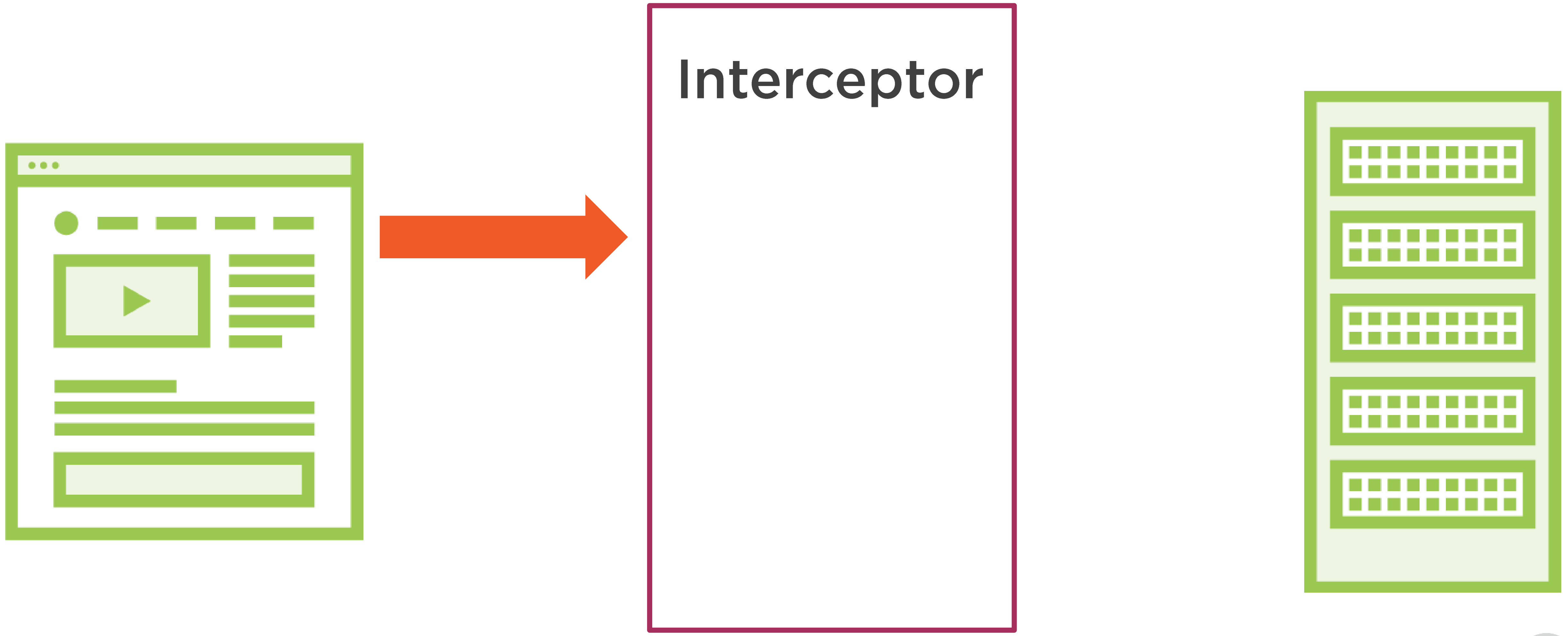
Manipulate HTTP requests before they’re sent to the server

Manipulate HTTP responses before they’re returned to your app

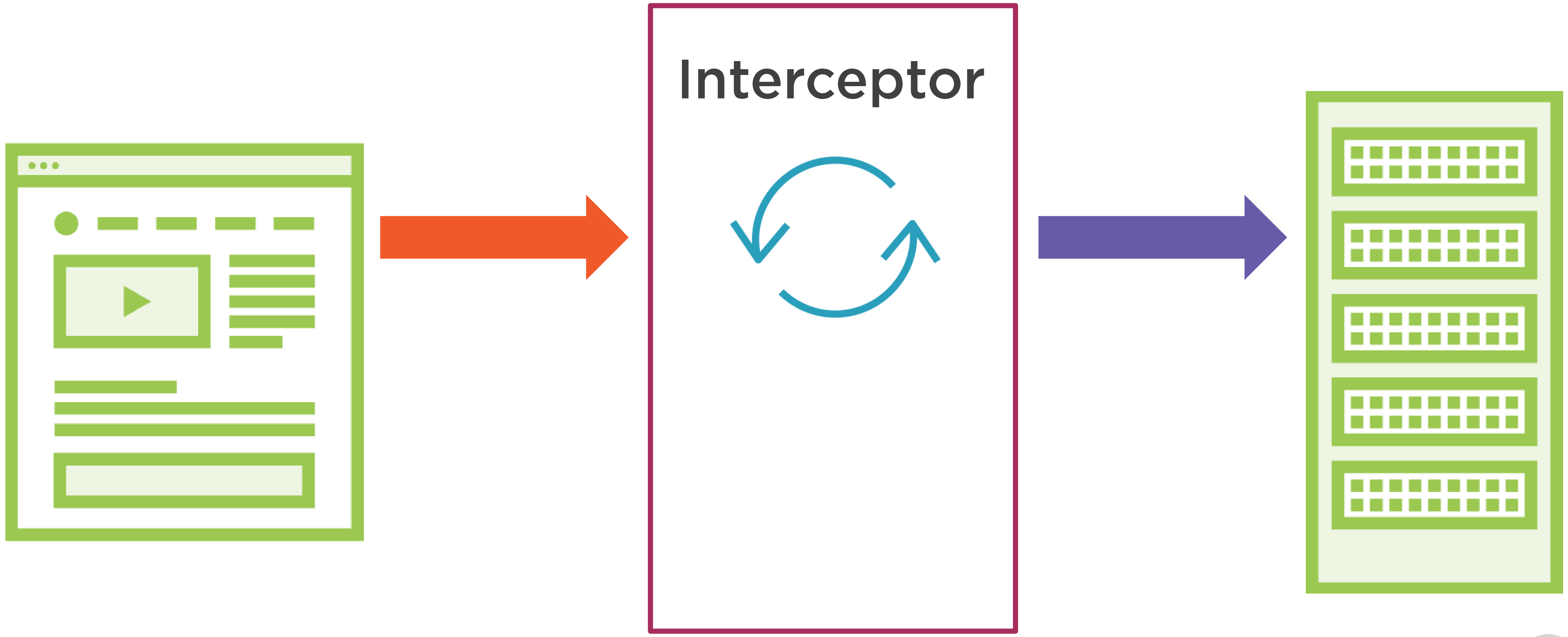
Intercepting Requests and Responses



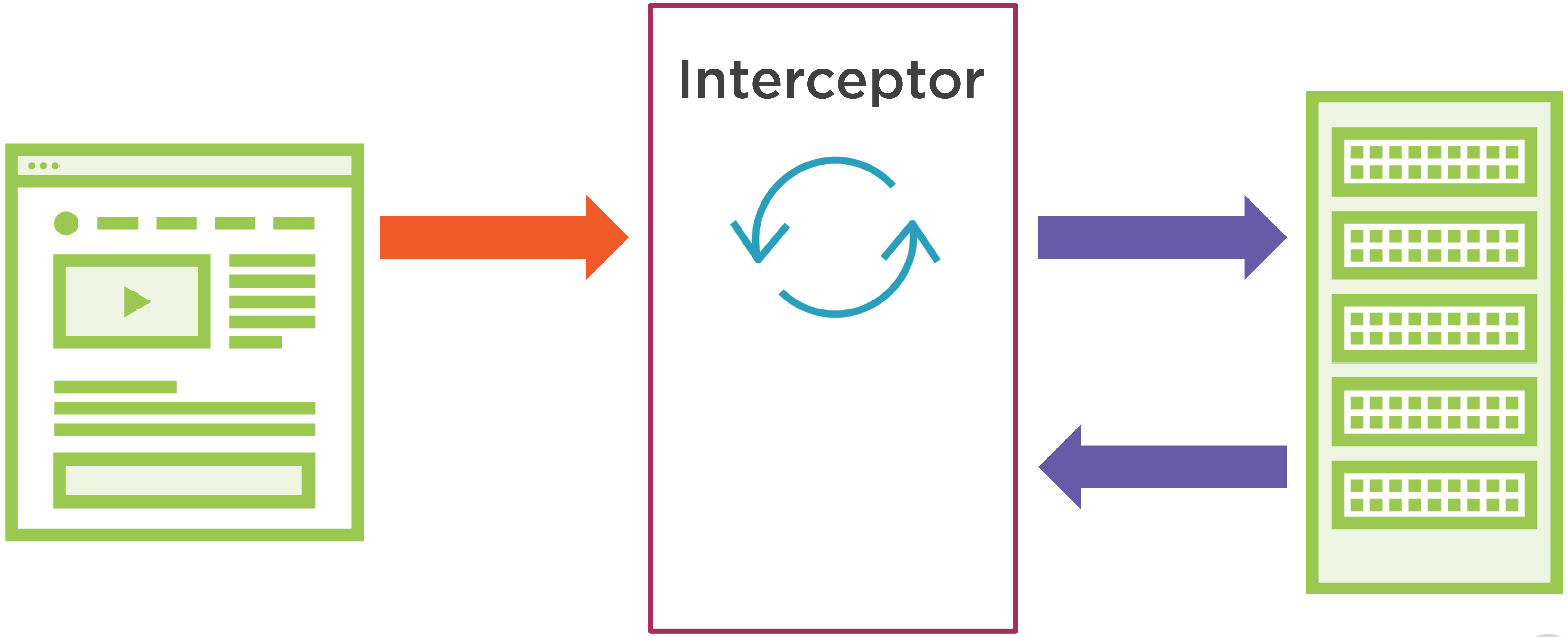
Intercepting Requests and Responses



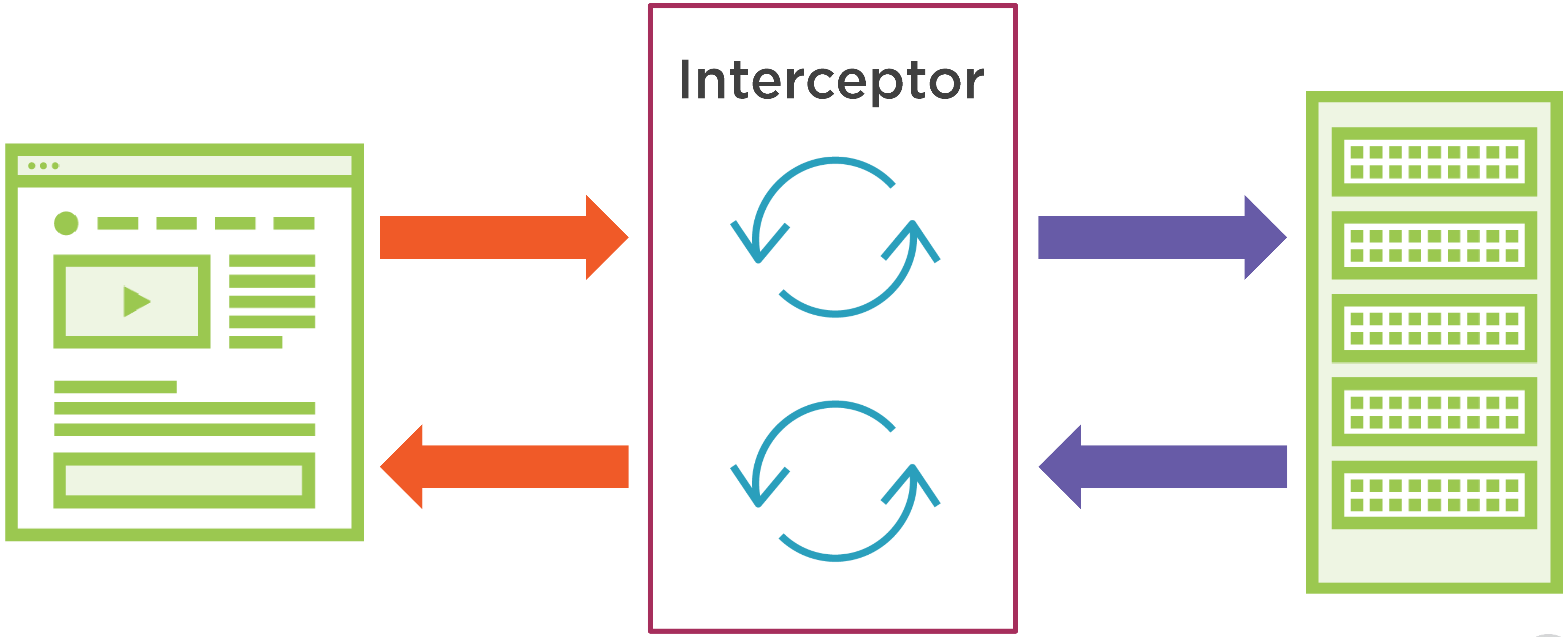
Intercepting Requests and Responses



Intercepting Requests and Responses



Intercepting Requests and Responses



Uses for Interceptors

Adding headers to all requests

Logging

Reporting progress events

Client-side caching

Defining an Interceptor

```
export class FirstInterceptor implements HttpInterceptor {
```

Defining an Interceptor

```
export class FirstInterceptor implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
```

Defining an Interceptor

```
export class FirstInterceptor implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    const modifiedRequest = req.clone();
```

Defining an Interceptor

```
export class FirstInterceptor implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    const modifiedRequest = req.clone();
```

Defining an Interceptor

```
export class FirstInterceptor implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    const modifiedRequest = req.clone();
    // change modifiedRequest here
```

Defining an Interceptor

```
export class FirstInterceptor implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    const modifiedRequest = req.clone();
    // change modifiedRequest here
    return next.handle(modifiedRequest)
```

Defining an Interceptor

```
export class FirstInterceptor implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    const modifiedRequest = req.clone();
    // change modifiedRequest here
    return next.handle(modifiedRequest)
```

Defining an Interceptor

```
export class FirstInterceptor implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    const modifiedRequest = req.clone();
    // change modifiedRequest here
    return next.handle(modifiedRequest)
```

Defining an Interceptor

```
export class FirstInterceptor implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    const modifiedRequest = req.clone();
    // change modifiedRequest here
    return next.handle(modifiedRequest)
```

Defining an Interceptor

```
export class FirstInterceptor implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    const modifiedRequest = req.clone();
    // change modifiedRequest here
    return next.handle(modifiedRequest)
```

Defining an Interceptor

```
export class FirstInterceptor implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    const modifiedRequest = req.clone();
    // change modifiedRequest here
    return next.handle(modifiedRequest)
      .pipe(
        .tap(event => {
          if (event instanceof HttpResponse) {
            // modify the HttpResponse here
          }
        })
      );
  }
}
```

Providing an Interceptor

```
@NgModule({
  imports: [],
  declarations: [],
  providers: [



  ]
})
export class AppModule { }
```

Providing an Interceptor

```
@NgModule({
  imports: [],
  declarations: [],
  providers: [
    { provide: HTTP_INTERCEPTORS, useClass: FirstInterceptor, multi: true },


  ]
})
export class AppModule { }
```

Providing an Interceptor

```
@NgModule({
  imports: [],
  declarations: [],
  providers: [
    { provide: HTTP_INTERCEPTORS, useClass: FirstInterceptor, multi: true },
    { provide: HTTP_INTERCEPTORS, useClass: SecondInterceptor, multi: true },
  ]
})
export class AppModule { }
```

Providing an Interceptor

```
@NgModule({
  imports: [],
  declarations: [],
  providers: [
    { provide: HTTP_INTERCEPTORS, useClass: FirstInterceptor, multi: true },
    { provide: HTTP_INTERCEPTORS, useClass: SecondInterceptor, multi: true },
  ]
})
export class AppModule { }
```

Providing an Interceptor

```
@NgModule({
  imports: [],
  declarations: [],
  providers: [
    { provide: HTTP_INTERCEPTORS, useClass: FirstInterceptor, multi: true },
    { provide: HTTP_INTERCEPTORS, useClass: SecondInterceptor, multi: true },
  ]
})
export class AppModule { }
```

Providing an Interceptor

```
@NgModule({
  imports: [],
  declarations: [],
  providers: [
    { provide: HTTP_INTERCEPTORS, useClass: FirstInterceptor, multi: true },
    { provide: HTTP_INTERCEPTORS, useClass: SecondInterceptor, multi: true },
  ]
})
export class AppModule { }
```

Providing an Interceptor

```
@NgModule({
  imports: [],
  declarations: [],
  providers: [
    { provide: HTTP_INTERCEPTORS, useClass: FirstInterceptor, multi: true },
    { provide: HTTP_INTERCEPTORS, useClass: SecondInterceptor, multi: true },
  ]
})
export class AppModule { }
```

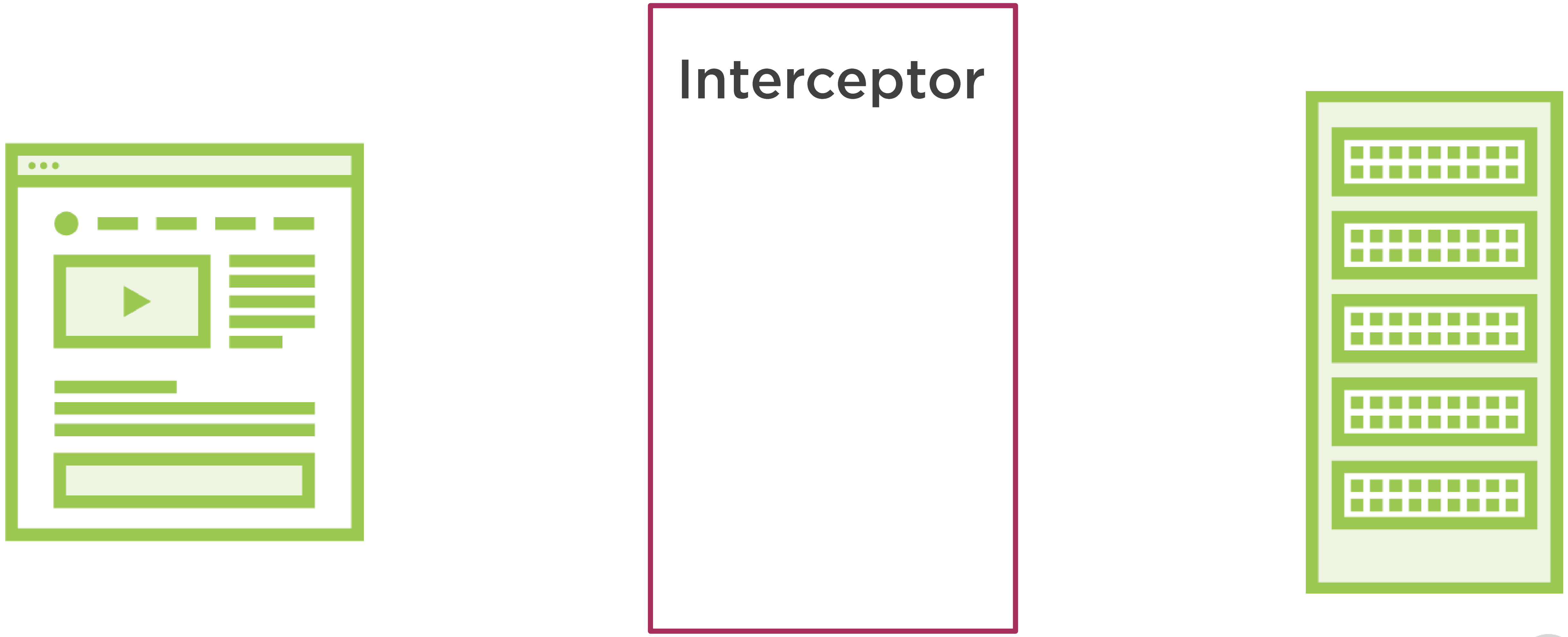
Demo

Creating an interceptor to add headers to all requests

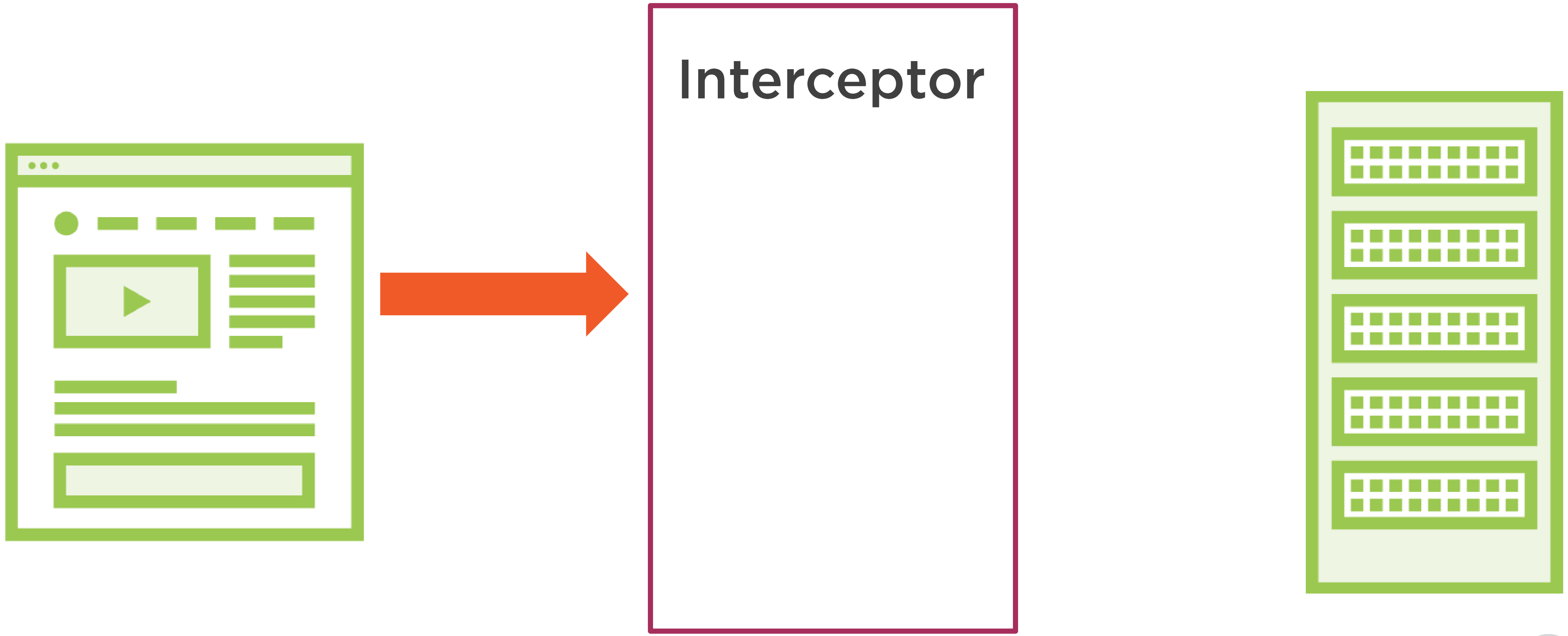
Demo

Intercepting responses and working with multiple interceptors

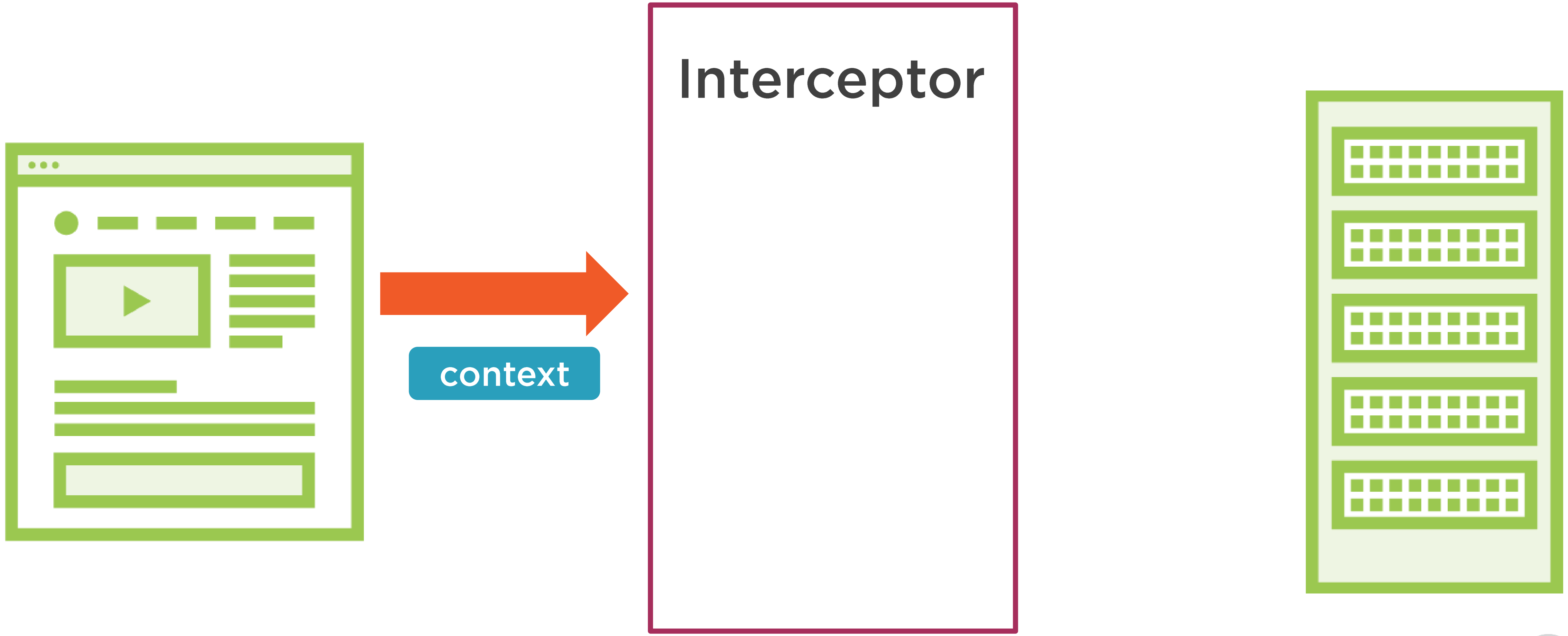
Configuring Interceptors



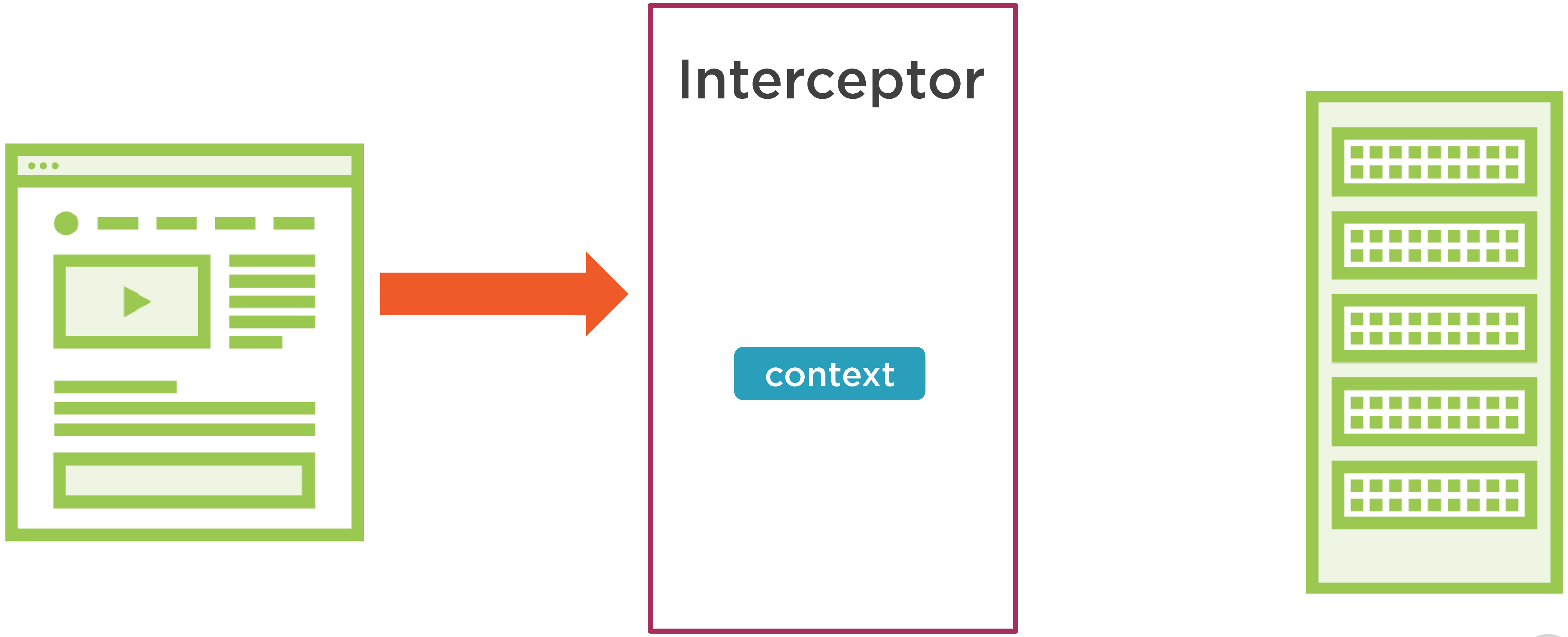
Configuring Interceptors



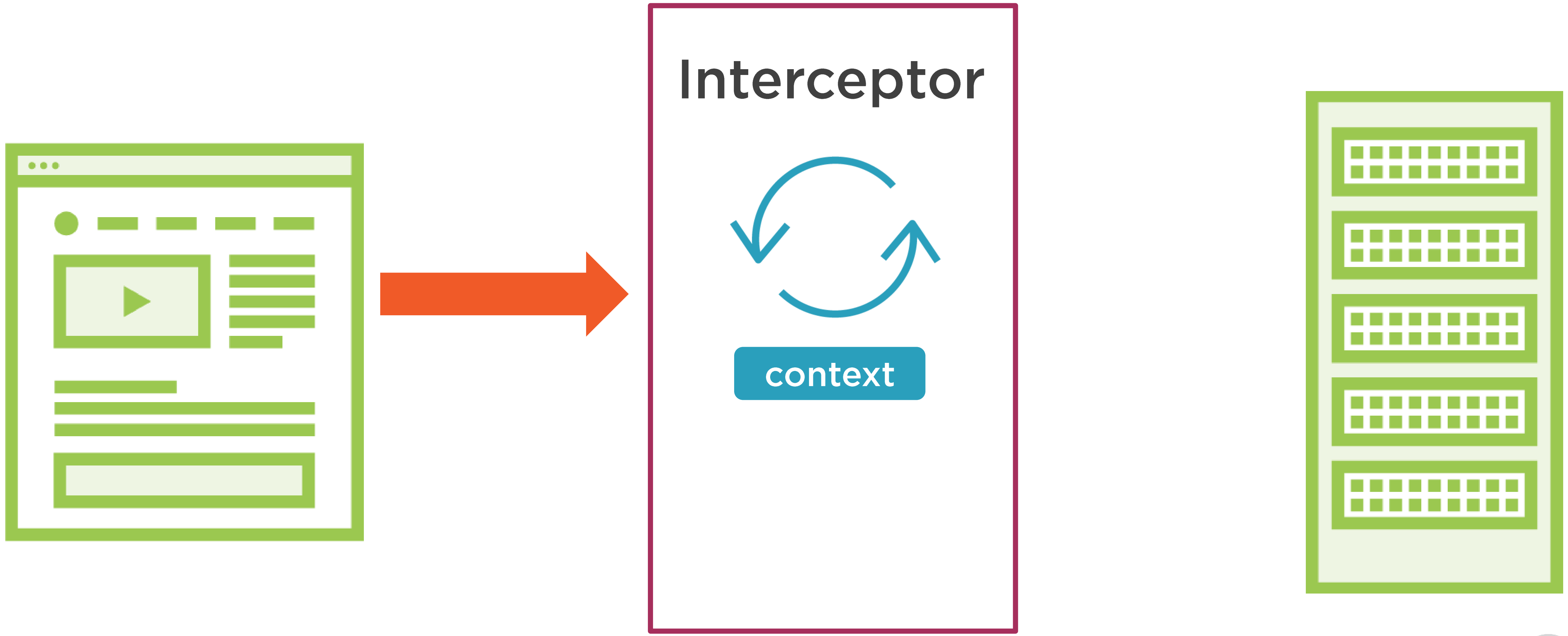
Configuring Interceptors



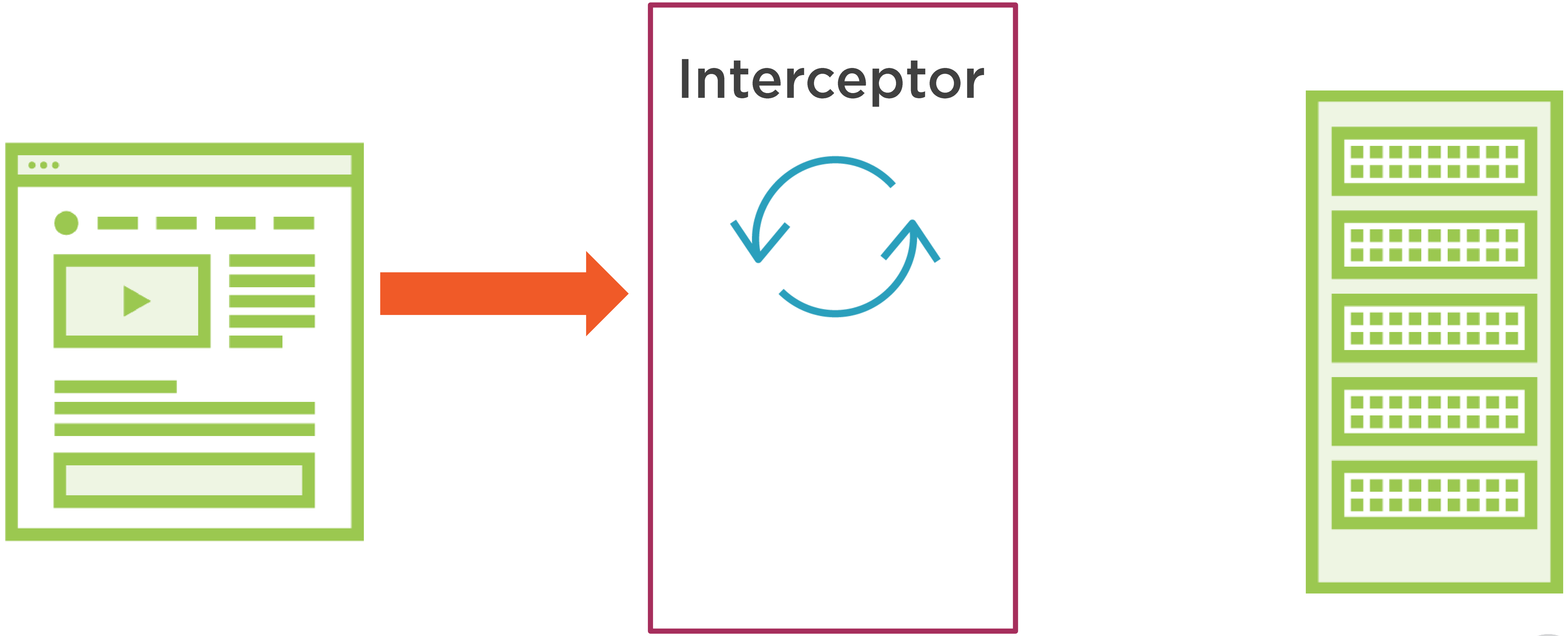
Configuring Interceptors



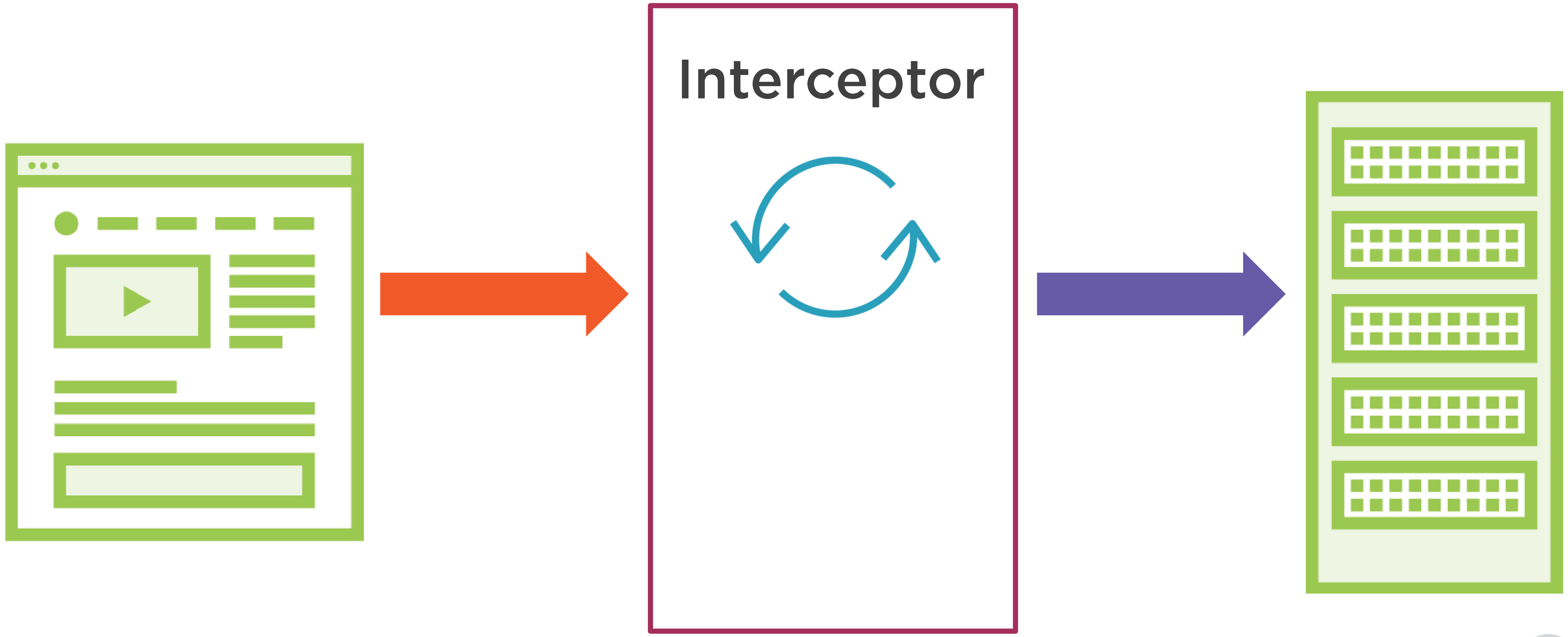
Configuring Interceptors



Configuring Interceptors



Configuring Interceptors



Creating and Processing Interceptor Metadata

```
// interceptor
export const OPTION_1 = new HttpContextToken<number>(() => 42);
```

Creating and Processing Interceptor Metadata

```
// interceptor
export const OPTION_1 = new HttpContextToken<number>(() => 42);
```

Creating and Processing Interceptor Metadata

```
// interceptor
export const OPTION_1 = new HttpContextToken<number>(() => 42);
```

Creating and Processing Interceptor Metadata

```
// interceptor
export const OPTION_1 = new HttpContextToken<number>(() => 42);
```

Creating and Processing Interceptor Metadata

```
// interceptor
export const OPTION_1 = new HttpContextToken<number>(() => 42);
```

Creating and Processing Interceptor Metadata

```
// interceptor
export const OPTION_1 = new HttpContextToken<number>(() => 42);
```

Creating and Processing Interceptor Metadata

```
// interceptor
export const OPTION_1 = new HttpContextToken<number>(() => 42);

// service
let my_context: HttpContext = new HttpContext();
```

Creating and Processing Interceptor Metadata

```
// interceptor
export const OPTION_1 = new HttpContextToken<number>(() => 42);

// service
let my_context: HttpContext = new HttpContext();
my_context.set(OPTION_1, 13);
```

Creating and Processing Interceptor Metadata

```
// interceptor
export const OPTION_1 = new HttpContextToken<number>(() => 42);

// service
let my_context: HttpContext = new HttpContext();
my_context.set(OPTION_1, 13);

this.http.get('/api/books', {
  context: my_context
});
```

Creating and Processing Interceptor Metadata

```
// interceptor
export const OPTION_1 = new HttpContextToken<number>(() => 42);

// service
let my_context: HttpContext = new HttpContext();
my_context.set(OPTION_1, 13);

this.http.get('/api/books', {
  context: my_context
});
```

Creating and Processing Interceptor Metadata

```
// interceptor
export const OPTION_1 = new HttpContextToken<number>(() => 42);

// service
let my_context: HttpContext = new HttpContext();
my_context.set(OPTION_1, 13);

this.http.get('/api/books', {
  context: my_context
});

// interceptor
let first_option: number = req.context.get<number>(OPTION_1);
```

Creating and Processing Interceptor Metadata

```
// interceptor
export const OPTION_1 = new HttpContextToken<number>(() => 42);

// service
let my_context: HttpContext = new HttpContext();
my_context.set(OPTION_1, 13);

this.http.get('/api/books', {
  context: my_context
});

// interceptor
let first_option: number = req.context.get<number>(OPTION_1);
```

Demo

Using metadata with interceptors