



Managing File Uploads w/ **NgRx**

A Reactive Case Study

Wes Grimes
Senior Engineer @ *Nrwl*
NgRx Core Team

 *@wesgrimes*

 *wesleygrimes.com*



*Let's **build** a file upload component*



What *Problem* Are We Solving?

Building a file upload component that supports...

- Concurrent file uploads
- Supports canceling in-progress
- Reports accurate progress for each file





Managing File Uploads w/ NgRx

Browse Files

Upload Files

Clear Files

Cancel Upload

No files chosen

*How might we **solve** this?*



*Let's **talk** event-based architecture*



What are the unique events
from the *Client* and *Server*?



Client Events

User requests to...

- *Add* files to upload queue
- *Process* upload queue
- *Cancel* file in-progress upload queue
- *Retry* failed upload



Server Events

Server notifies file upload has...

- *Started*
- *Progressed*
- *Completed*
- *Failed*



Capturing *Client* Events

- Using HTML Button *click* events
- Using HTML File Input *change* events



Capturing Button click Events

```
// some.component.html
<button type="button" (click)="doSomething()">

// some.component.ts
@Component({...})
export class SomeComponent {
  doSomething() {
    // do something here
  }
}
```



Capturing File Input change Event

```
// some.component.html
<input type="file" multiple (change)="onFileChange($event)" />

// some.component.ts
@Component({...})
export class SomeComponent {
  onFileChange(event: Event) {
    const files = event.target.files; // do something with files
  }
}
```



Capturing *Server* Events

- Use the Angular *HttpClient* w/ *reportProgress*
- Returns Observable of *HttpEvent* for each *HttpEventType*



HttpClient **reportProgress**

```
const httpOptions = {  
  reportProgress: true  
};  
  
const req = new HttpRequest(  
  'POST',  
  'api/upload',  
  formData,  
  httpOptions  
);
```



*HttpClient w/ **reportProgress***

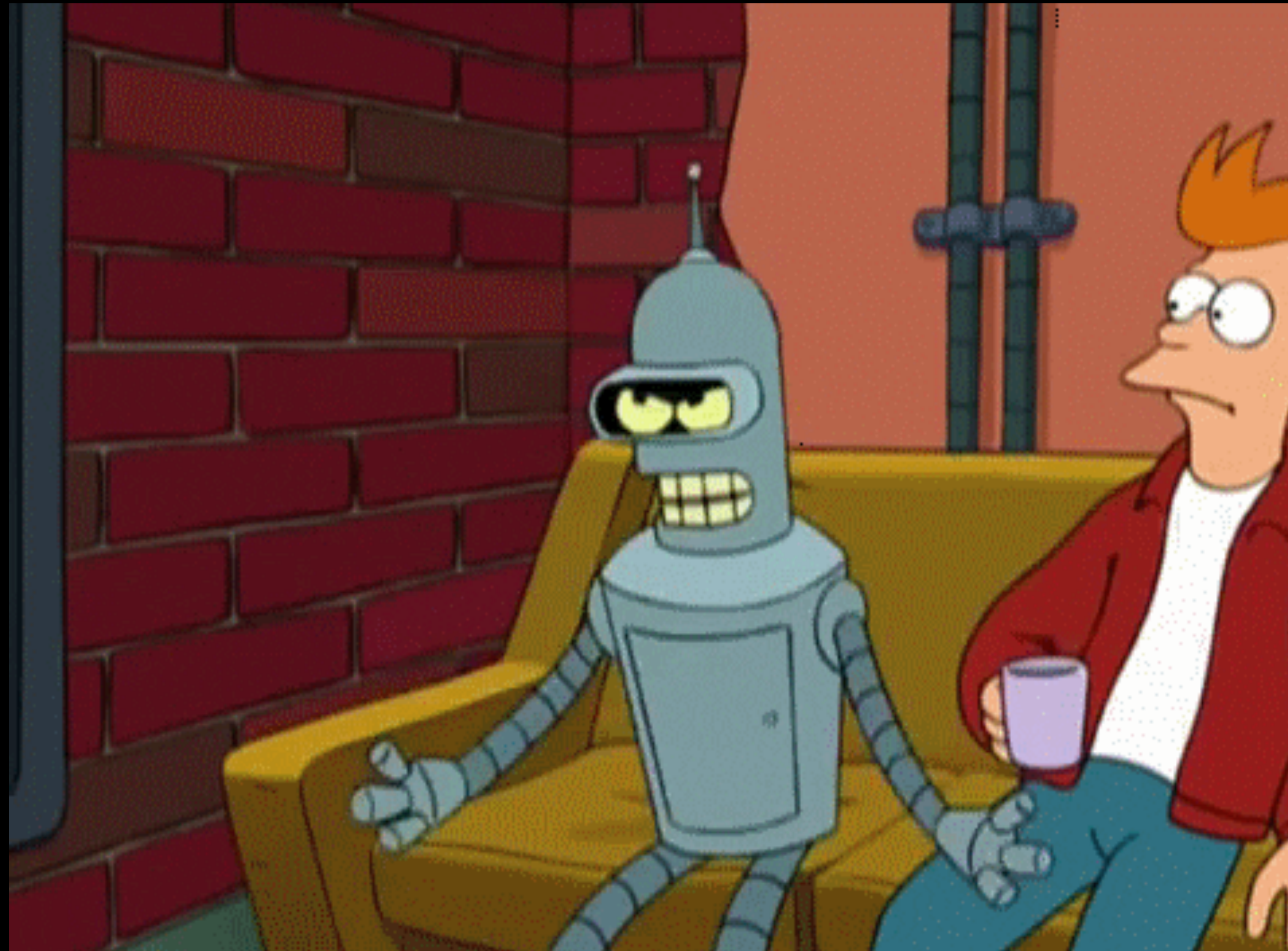
```
this.http  
  .request(req)  
  .pipe(  
    map(httpEvent  $\Rightarrow$  doSomethingWith(httpEvent))  
  );
```



HttpEventType's

```
export enum HttpEventType {  
  Sent,  
  UploadProgress,  
  ResponseHeader,  
  DownloadProgress,  
  Response,  
  User,  
}
```





**SHUT UP AND GET TO
THE POINT!**

An Event-based *Solution* Using *NgRx*

- Defines *Actions* as Events
- Uses *Reducers* and *Effects* to handle those events
- Uses *Selectors* to provide feedback to the User on those events



Client Actions in *NgRx*

```
export const added = createAction('[Upload Form] Added', props<{ file: File }>());  
  
export const processRequested = createAction('[Upload Form] Process Requested');  
  
export const cancelRequested = createAction('[Upload Form] Cancel Requested');  
  
export const retryRequested = createAction(  
  '[Upload Form] Retry Requested',  
  props<{ id: string }>()  
);
```



Bridge Action in *NgRx*

```
export const uploadRequested = createAction(  
  '[Upload Effect] Upload Requested',  
  props<{ fileToUpload: FileUploadModel }>()  
);
```



Server Actions in *NgRx*

```
export const uploadStarted = createAction(
  '[Upload API] Upload Started',
  props<{ id: string }>()
);

export const uploadProgressed = createAction(
  '[Upload API] Upload Progressed',
  props<{ id: string; progress: number }>()
);

export const uploadCompleted = createAction(
  '[Upload API] Upload Completed',
  props<{ id: string }>()
);

export const uploadFailed = createAction(
  '[Upload API] Upload Failed',
  props<{ id: string; error: string }>()
);
```



State in *NgRx*

```
export interface FileUploadState {  
  files: FileUploadModel[];  
}
```



State in *NgRx*

```
export interface FileUploadModel {  
  id: string;  
  fileName: string;  
  fileSize: number;  
  rawFile: File;  
  progress: number;  
  status: FileUploadStatus;  
  error: string;  
}
```



Reducers in *NgRx*

Reducers in *NgRx* are responsible for handling transitions from one state to the next state in your application.



File Upload *Reducer*

- *added* — Add the file to the state
- *started* — status = *Started*
- *progressed* — status = *InProgress* & progress = *event.progress*
- *completed* — status = *Completed* & progress = *100*
- *failed* — status = *Failed* & progress = *0*
- *retryRequested* — status = *Ready* & progress = *0*



Selectors in *NgRx*

Selectors are *pure functions* used for *deriving* slices of store state.



File Upload *Selectors* in *NgRx*

- ***selectFilesReadyForUpload*** - Select files ready for upload using files in list filtered to list of *Ready* status
- ***selectFileUploadQueue*** - Select files currently in the queue, regardless of status, with status, progress, and other info. This is our “view model” selector.





Effects in NgRx

- *processQueueEffect*
 - **Triggered When:** processRequested or retryRequested actions are dispatched
 - **Maps To:** uploadRequested action for every ready file in the queue
- *uploadEffect*
 - **Triggered When:** uploadRequested action is dispatched
 - **Maps To:** Calls FileUploadService.uploadFile for every file in the queue that is set to status of Requested
 - **Maps To:** Specific Action for every HTTP Status Event



Process Queue *Effect* in *NgRx*

```
processQueueEffect$ = createEffect(() =>
  this.actions$.pipe(
    ofType(
      FileUploadUIActions.processRequested,
      FileUploadUIActions.retryRequested
    ),
    withLatestFrom(
      this.store.select(FileUploadSelectors.selectFilesReadyForUpload)
    ),
    switchMap(([_, filesToUpload]) =>
      filesToUpload.map(fileToUpload =>
        FileUploadAPIActions.uploadRequested({ fileToUpload })
      )
    )
  )
);
```



Process Queue *Effect* in *NgRx*

```
processQueueEffect$ = createEffect(() =>
  this.actions$.pipe(
    ofType(
      FileUploadUIActions.processRequested,
      FileUploadUIActions.retryRequested
    ),
    withLatestFrom(
      this.store.select(FileUploadSelectors.selectFilesReadyForUpload)
    ),
    switchMap(([_, filesToUpload]) =>
      filesToUpload.map(fileToUpload =>
        FileUploadAPIActions.uploadRequested({ fileToUpload })
      )
    )
  )
);
```



Process Queue *Effect* in *NgRx*

```
processQueueEffect$ = createEffect(() =>
  this.actions$.pipe(
    ofType(
      FileUploadUIActions.processRequested,
      FileUploadUIActions.retryRequested
    ),
    withLatestFrom(
      this.store.select(FileUploadSelectors.selectFilesReadyForUpload)
    ),
    switchMap(([_, filesToUpload]) =>
      filesToUpload.map(fileToUpload =>
        FileUploadAPIActions.uploadRequested({ fileToUpload })
      )
    )
  )
);
```



Process Queue *Effect* in *NgRx*

```
processQueueEffect$ = createEffect(() =>
  this.actions$.pipe(
    ofType(
      FileUploadUIActions.processRequested,
      FileUploadUIActions.retryRequested
    ),
    withLatestFrom(
      this.store.select(FileUploadSelectors.selectFilesReadyForUpload)
    ),
    switchMap(([_, filesToUpload]) =>
      filesToUpload.map(fileToUpload =>
        FileUploadAPIActions.uploadRequested({ fileToUpload })
      )
    )
  )
);
```



Process Queue *Effect* in *NgRx*

```
processQueueEffect$ = createEffect(() =>
  this.actions$.pipe(
    ofType(
      FileUploadUIActions.processRequested,
      FileUploadUIActions.retryRequested
    ),
    withLatestFrom(
      this.store.select(FileUploadSelectors.selectFilesReadyForUpload)
    ),
    switchMap(([_, filesToUpload]) =>
      filesToUpload.map(fileToUpload =>
        FileUploadAPIActions.uploadRequested({ fileToUpload })
      )
    )
  )
);
```



```

uploadEffect$ = createEffect(() =>
  this.actions$.pipe(
    ofType(FileUploadAPIActions.uploadRequested),
    mergeMap(({ fileToUpload }) =>
      this.fileUploadService.uploadFile(fileToUpload.rawFile).pipe(
        takeUntil(
          this.actions$.pipe(ofType(FileUploadUIActions.cancelRequested))
        ),
        map(event => this.getActionFromHttpEvent(event, fileToUpload.id)),
        catchError(error =>
          of(
            FileUploadAPIActions.uploadFailed({
              error: error.message,
              id: fileToUpload.id
            })
          )
        )
      )
    )
  )
);

```

Upload *Effect* in *NgRx*



```

uploadEffect$ = createEffect(() =>
  this.actions$.pipe(
    ofType(FileUploadAPIActions.uploadRequested),
    mergeMap(({ fileToUpload }) =>
      this.fileUploadService.uploadFile(fileToUpload.rawFile).pipe(
        takeUntil(
          this.actions$.pipe(ofType(FileUploadUIActions.cancelRequested))
        ),
        map(event => this.getActionFromHttpEvent(event, fileToUpload.id)),
        catchError(error =>
          of(
            FileUploadAPIActions.uploadFailed({
              error: error.message,
              id: fileToUpload.id
            })
          )
        )
      )
    )
  )
);

```

Upload *Effect* in *NgRx*



```

uploadEffect$ = createEffect(() =>
  this.actions$.pipe(
    ofType(FileUploadAPIActions.uploadRequested),
    mergeMap(({ fileToUpload }) =>
      this.fileUploadService.uploadFile(fileToUpload.rawFile).pipe(
        takeUntil(
          this.actions$.pipe(ofType(FileUploadUIActions.cancelRequested))
        ),
        map(event => this.getActionFromHttpEvent(event, fileToUpload.id)),
        catchError(error =>
          of(
            FileUploadAPIActions.uploadFailed({
              error: error.message,
              id: fileToUpload.id
            })
          )
        )
      )
    )
  )
);

```

Upload *Effect* in *NgRx*



```

uploadEffect$ = createEffect(() =>
  this.actions$.pipe(
    ofType(FileUploadAPIActions.uploadRequested),
    mergeMap(({ fileToUpload }) =>
      this.fileUploadService.uploadFile(fileToUpload.rawFile).pipe(
        takeUntil(
          this.actions$.pipe(ofType(FileUploadUIActions.cancelRequested))
        ),
        map(event => this.getActionFromHttpEvent(event, fileToUpload.id)),
        catchError(error =>
          of(
            FileUploadAPIActions.uploadFailed({
              error: error.message,
              id: fileToUpload.id
            })
          )
        )
      )
    )
  )
);

```

Upload *Effect* in *NgRx*




```

uploadEffect$ = createEffect(() =>
  this.actions$.pipe(
    ofType(FileUploadAPIActions.uploadRequested),
    mergeMap(({ fileToUpload }) =>
      this.fileUploadService.uploadFile(fileToUpload.rawFile).pipe(
        takeUntil(
          this.actions$.pipe(ofType(FileUploadUIActions.cancelRequested))
        ),
        map(event => this.getActionFromHttpEvent(event, fileToUpload.id)),
        catchError(error =>
          of(
            FileUploadAPIActions.uploadFailed({
              error: error.message,
              id: fileToUpload.id
            })
          )
        )
      )
    )
  )
);

```

Upload *Effect* in *NgRx*




```

uploadEffect$ = createEffect(() =>
  this.actions$.pipe(
    ofType(FileUploadAPIActions.uploadRequested),
    mergeMap(({ fileToUpload }) =>
      this.fileUploadService.uploadFile(fileToUpload.rawFile).pipe(
        takeUntil(
          this.actions$.pipe(ofType(FileUploadUIActions.cancelRequested))
        ),
        map(event => this.getActionFromHttpEvent(event, fileToUpload.id)),
        catchError(error =>
          of(
            FileUploadAPIActions.uploadFailed({
              error: error.message,
              id: fileToUpload.id
            })
          )
        )
      )
    )
  )
);

```

Upload *Effect* in *NgRx*



Mapping *HttpEvent*'s to *NgRx* Actions

```
private getActionFromHttpEvent(id: string, event: HttpEvent<any>) {  
  switch (event.type) {  
    case HttpEventType.Sent: {  
      return FileUploadAPIActions.uploadStarted({ id });  
    }  
    case HttpEventType.DownloadProgress:  
    case HttpEventType.UploadProgress: {  
      return FileUploadAPIActions.uploadProgressed({  
        id,  
        progress: Math.round((100 * event.loaded) / event.total)  
      });  
    }  
    case HttpEventType.ResponseHeader:  
    case HttpEventType.Response: {  
      if (event.status === 200) {  
        return FileUploadAPIActions.uploadCompleted({ id });  
      } else {  
        return FileUploadAPIActions.uploadFailed({  
          id,  
          error: event.statusText  
        });  
      }  
    }  
    default: {  
      return FileUploadAPIActions.uploadFailed({  
        id,  
        error: `Unknown Event: ${JSON.stringify(event)}`  
      });  
    }  
  }  
}
```



HttpEventType.Sent
uploadStarted



```
private getActionFromHttpEvent(id: string, event: HttpEvent<any>) {  
  switch (event.type) {  
    case HttpEventType.Sent: {  
      return FileUploadAPIActions.uploadStarted({ id });  
    }  
    case HttpEventType.DownloadProgress:  
    case HttpEventType.UploadProgress: {  
      return FileUploadAPIActions.uploadProgressed({  
        id,  
        progress: Math.round((100 * event.loaded) / event.total)  
      });  
    }  
    case HttpEventType.ResponseHeader:  
    case HttpEventType.Response: {  
      if (event.status === 200) {  
        return FileUploadAPIActions.uploadCompleted({ id });  
      } else {  
        return FileUploadAPIActions.uploadFailed({  
          id,  
          error: event.statusText  
        });  
      }  
    }  
    default: {  
      return FileUploadAPIActions.uploadFailed({  
        id,  
        error: `Unknown Event: ${JSON.stringify(event)}`  
      });  
    }  
  }  
}
```



HttpEventType.DownloadProgress
HttpEventType.UploadProgress
uploadProgress

```
private getActionFromHttpEvent(id: string, event: HttpEvent<any>) {  
  switch (event.type) {  
    case HttpEventType.Sent: {  
      return FileUploadAPIActions.uploadStarted({ id });  
    }  
    case HttpEventType.DownloadProgress:  
    case HttpEventType.UploadProgress: {  
      return FileUploadAPIActions.uploadProgressed({  
        id,  
        progress: Math.round((100 * event.loaded) / event.total)  
      });  
    }  
    case HttpEventType.ResponseHeader:  
    case HttpEventType.Response: {  
      if (event.status === 200) {  
        return FileUploadAPIActions.uploadCompleted({ id });  
      } else {  
        return FileUploadAPIActions.uploadFailed({  
          id,  
          error: event.statusText  
        });  
      }  
    }  
    default: {  
      return FileUploadAPIActions.uploadFailed({  
        id,  
        error: `Unknown Event: ${JSON.stringify(event)}`  
      });  
    }  
  }  
}
```



HttpEventType.ResponseHeader
HttpEventType.Response

200 ... uploadCompleted
!200 ... uploadFailure



```
private getActionFromHttpEvent(id: string, event: HttpEvent<any>) {  
  switch (event.type) {  
    case HttpEventType.Sent: {  
      return FileUploadAPIActions.uploadStarted({ id });  
    }  
    case HttpEventType.DownloadProgress:  
    case HttpEventType.UploadProgress: {  
      return FileUploadAPIActions.uploadProgressed({  
        id,  
        progress: Math.round((100 * event.loaded) / event.total)  
      });  
    }  
    case HttpEventType.ResponseHeader:  
    case HttpEventType.Response: {  
      if (event.status === 200) {  
        return FileUploadAPIActions.uploadCompleted({ id });  
      } else {  
        return FileUploadAPIActions.uploadFailed({  
          id,  
          error: event.statusText  
        });  
      }  
    }  
    default: {  
      return FileUploadAPIActions.uploadFailed({  
        id,  
        error: `Unknown Event: ${JSON.stringify(event)}`  
      });  
    }  
  }  
}
```



```

private getActionFromHttpEvent(id: string, event: HttpEvent<any>) {
  switch (event.type) {
    case HttpEventType.Sent: {
      return FileUploadAPIActions.uploadStarted({ id });
    }
    case HttpEventType.DownloadProgress:
    case HttpEventType.UploadProgress: {
      return FileUploadAPIActions.uploadProgressed({
        id,
        progress: Math.round((100 * event.loaded) / event.total)
      });
    }
    case HttpEventType.ResponseHeader:
    case HttpEventType.Response: {
      if (event.status === 200) {
        return FileUploadAPIActions.uploadCompleted({ id });
      } else {
        return FileUploadAPIActions.uploadFailed({
          id,
          error: event.statusText
        });
      }
    }
  }
  default: {
    return FileUploadAPIActions.uploadFailed({
      id,
      error: `Unknown Event: ${JSON.stringify(event)}`
    });
  }
}

```

All Other HttpEventType
uploadFailure




```

uploadEffect$ = createEffect(() =>
  this.actions$.pipe(
    ofType(FileUploadAPIActions.uploadRequested),
    mergeMap(({ fileToUpload }) =>
      this.fileUploadService.uploadFile(fileToUpload.rawFile).pipe(
        takeUntil(
          this.actions$.pipe(ofType(FileUploadUIActions.cancelRequested))
        ),
        map(event => this.getActionFromHttpEvent(event, fileToUpload.id)),
        catchError(error =>
          of(
            FileUploadAPIActions.uploadFailed({
              error: error.message,
              id: fileToUpload.id
            })
          )
        )
      )
    )
  )
);

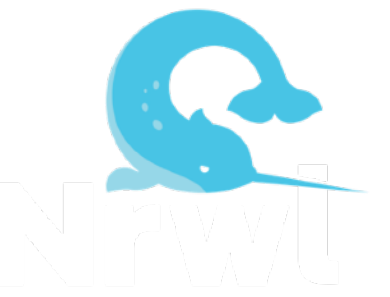
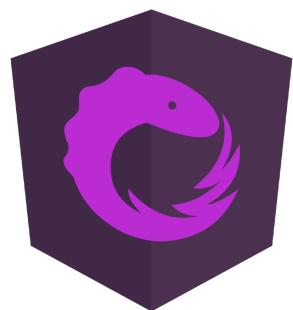
```

Upload *Effect* in *NgRx*





Wire



```
@Component({...})
export class FileUploadComponent {
  fileUploadQueue$ = this.store.select(
    FileUploadSelectors.selectFileUploadQueue
  );

  constructor(private store: Store<{}>) {}

  addFiles(event) {
    const files: File[] = event.target.files;

    files.forEach(file =>
      this.store.dispatch(FileUploadUIActions.added({ file })))
    );
  }

  requestRetry(id: string) {
    this.store.dispatch(FileUploadUIActions.retryRequested({ id }));
  }

  requestCancel() {
    this.store.dispatch(FileUploadUIActions.cancelRequested());
  }

  requestProcess() {
    this.store.dispatch(FileUploadUIActions.processRequested());
  }
}
```

Inject **Store** & **NOTHING ELSE**

File Upload **Component**

```
@Component({...})
export class FileUploadComponent {
  fileUploadQueue$ = this.store.select(
    FileUploadSelectors.selectFileUploadQueue
  );

  constructor(private store: Store<{}>) {}

  addFiles(event) {
    const files: File[] = event.target.files;

    files.forEach(file =>
      this.store.dispatch(FileUploadUIActions.added({ file })))
    );
  }

  requestRetry(id: string) {
    this.store.dispatch(FileUploadUIActions.retryRequested({ id }));
  }

  requestCancel() {
    this.store.dispatch(FileUploadUIActions.cancelRequested());
  }

  requestProcess() {
    this.store.dispatch(FileUploadUIActions.processRequested());
  }
}
```

Wire up the view-model *selector*

File Upload *Component*

```
@Component({...})
export class FileUploadComponent {
  fileUploadQueue$ = this.store.select(
    FileUploadSelectors.selectFileUploadQueue
  );

  constructor(private store: Store<{}>) {}

  addFiles(event) {
    const files: File[] = event.target.files;

    files.forEach(file =>
      this.store.dispatch(FileUploadUIActions.added({ file }));
    );
  }

  requestRetry(id: string) {
    this.store.dispatch(FileUploadUIActions.retryRequested({ id }));
  }

  requestCancel() {
    this.store.dispatch(FileUploadUIActions.cancelRequested());
  }

  requestProcess() {
    this.store.dispatch(FileUploadUIActions.processRequested());
  }
}
```



Wire up the **Add** Event

File Upload **Component**

```
@Component({...})
export class FileUploadComponent {
  fileUploadQueue$ = this.store.select(
    FileUploadSelectors.selectFileUploadQueue
  );

  constructor(private store: Store<{}>) {}

  addFiles(event) {
    const files: File[] = event.target.files;

    files.forEach(file =>
      this.store.dispatch(FileUploadUIActions.added({ file })))
    );
  }

  requestRetry(id: string) {
    this.store.dispatch(FileUploadUIActions.retryRequested({ id }));
  }

  requestCancel() {
    this.store.dispatch(FileUploadUIActions.cancelRequested());
  }

  requestProcess() {
    this.store.dispatch(FileUploadUIActions.processRequested());
  }
}
```

← Wire up the **Retry** Event

File Upload **Component**


```
@Component({...})
export class FileUploadComponent {
  fileUploadQueue$ = this.store.select(
    FileUploadSelectors.selectFileUploadQueue
  );

  constructor(private store: Store<{}>) {}

  addFiles(event) {
    const files: File[] = event.target.files;

    files.forEach(file =>
      this.store.dispatch(FileUploadUIActions.added({ file })))
    );
  }

  requestRetry(id: string) {
    this.store.dispatch(FileUploadUIActions.retryRequested({ id }));
  }

  requestCancel() {
    this.store.dispatch(FileUploadUIActions.cancelRequested());
  }

  requestProcess() {
    this.store.dispatch(FileUploadUIActions.processRequested());
  }
}
```

Wire up the **Cancel** Event

File Upload **Component**

File Upload *Component*

```
@Component({...})
export class FileUploadComponent {
  fileUploadQueue$ = this.store.select(
    FileUploadSelectors.selectFileUploadQueue
  );

  constructor(private store: Store<{}>) {}

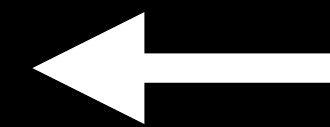
  addFiles(event) {
    const files: File[] = event.target.files;

    files.forEach(file =>
      this.store.dispatch(FileUploadUIActions.added({ file }))
    );
  }

  requestRetry(id: string) {
    this.store.dispatch(FileUploadUIActions.retryRequested({ id }));
  }

  requestCancel() {
    this.store.dispatch(FileUploadUIActions.cancelRequested());
  }

  requestProcess() {
    this.store.dispatch(FileUploadUIActions.processRequested());
  }
}
```



Wire up the *Process* Event



~~Push it real good!~~



File Upload Component

onFileChange

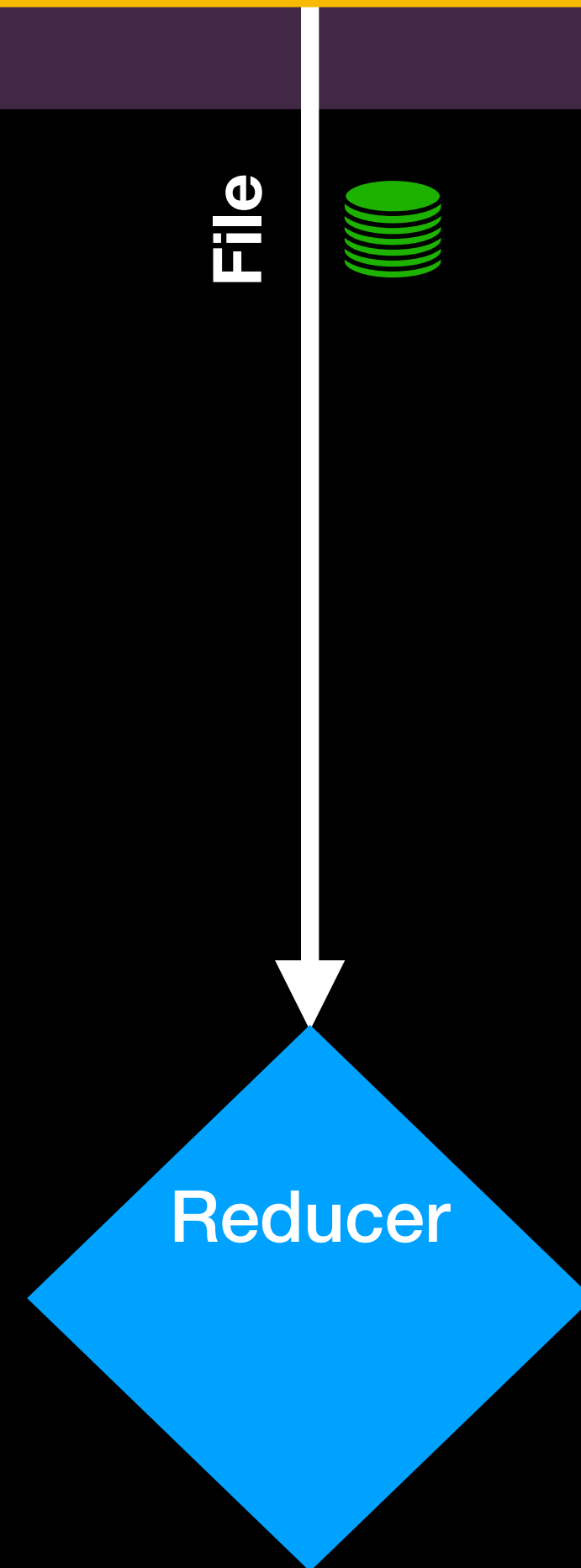
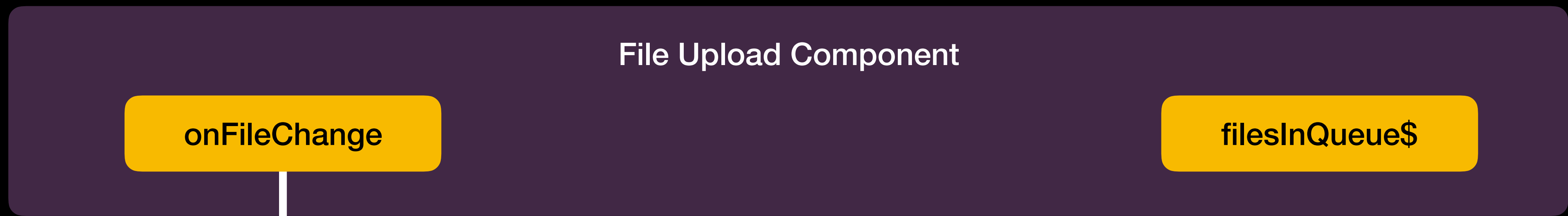
filesInQueue\$

Adding a File to Queue

Reducer

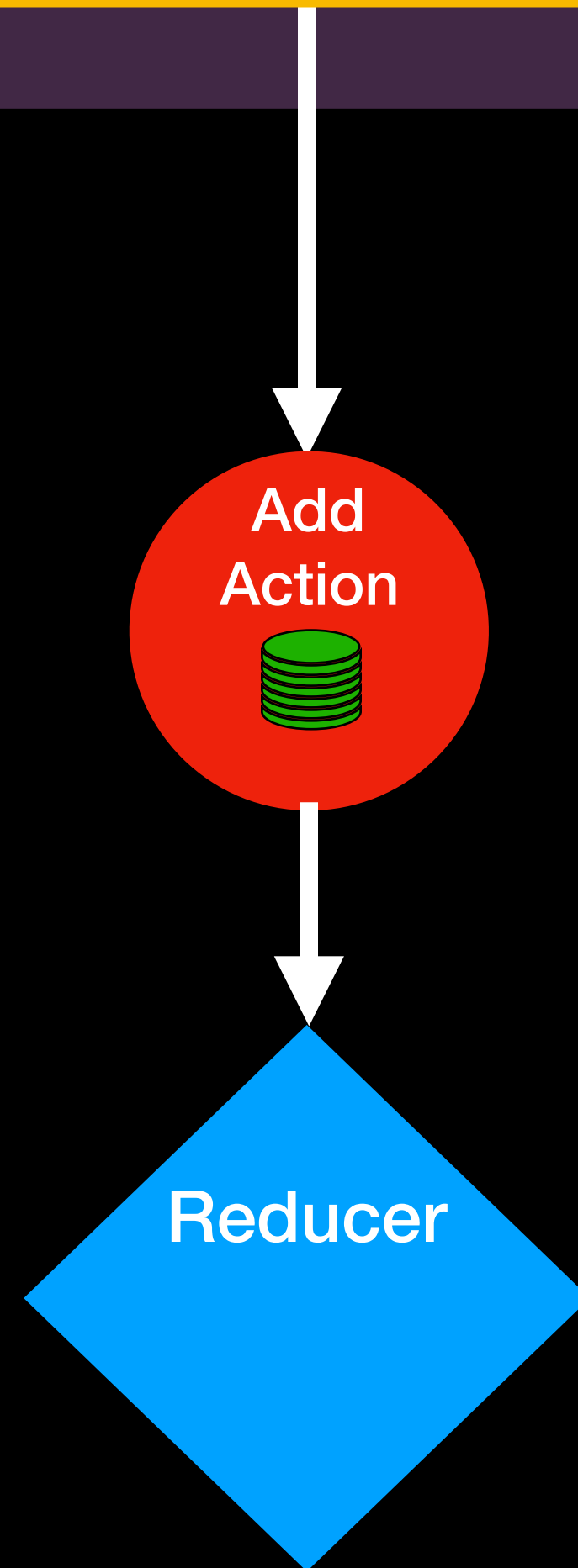
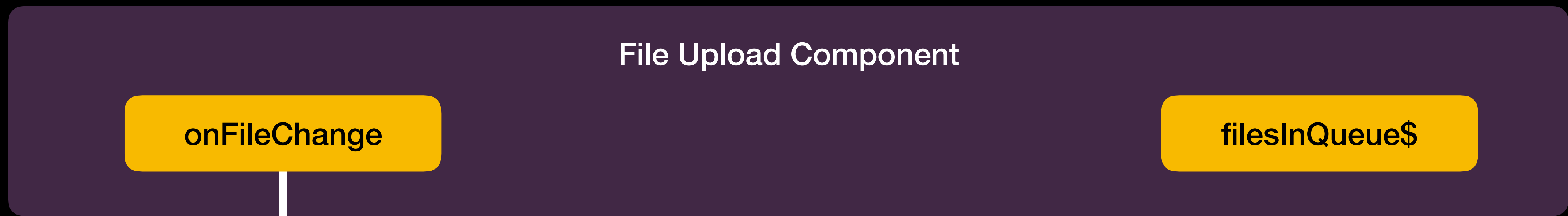
Store



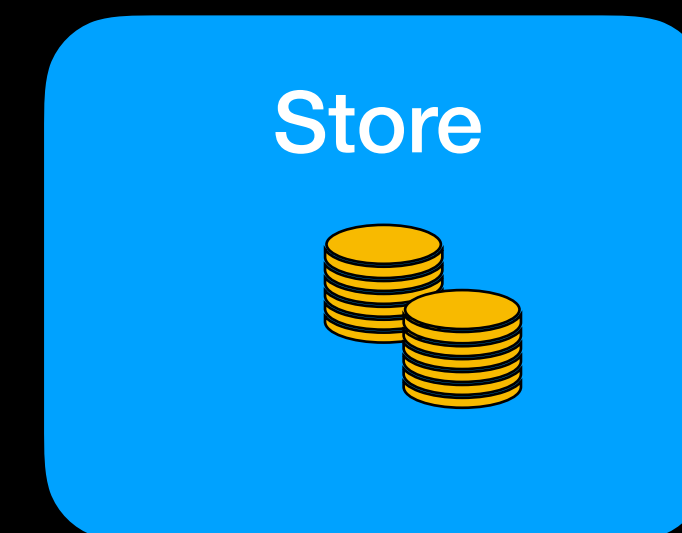


*Adding a File
to Queue*





*Adding a File
to Queue*



File Upload Component

onFileChange

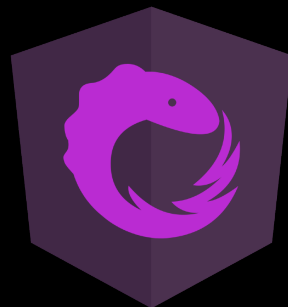
filesInQueue\$

*Adding a File
to Queue*

Reducer

Add
Action

Store



File Upload Component

onFileChange

filesInQueue\$

*Adding a File
to Queue*

Reducer

Store

Current State



File Upload Component

onFileChange

filesInQueue\$

*Adding a File
to Queue*

Reducer

Store

Current State



File Upload Component

onFileChange

filesInQueue\$

*Adding a File
to Queue*

Reducer

New State

Store

Current State



File Upload Component

onFileChange

filesInQueue\$

*Adding a File
to Queue*

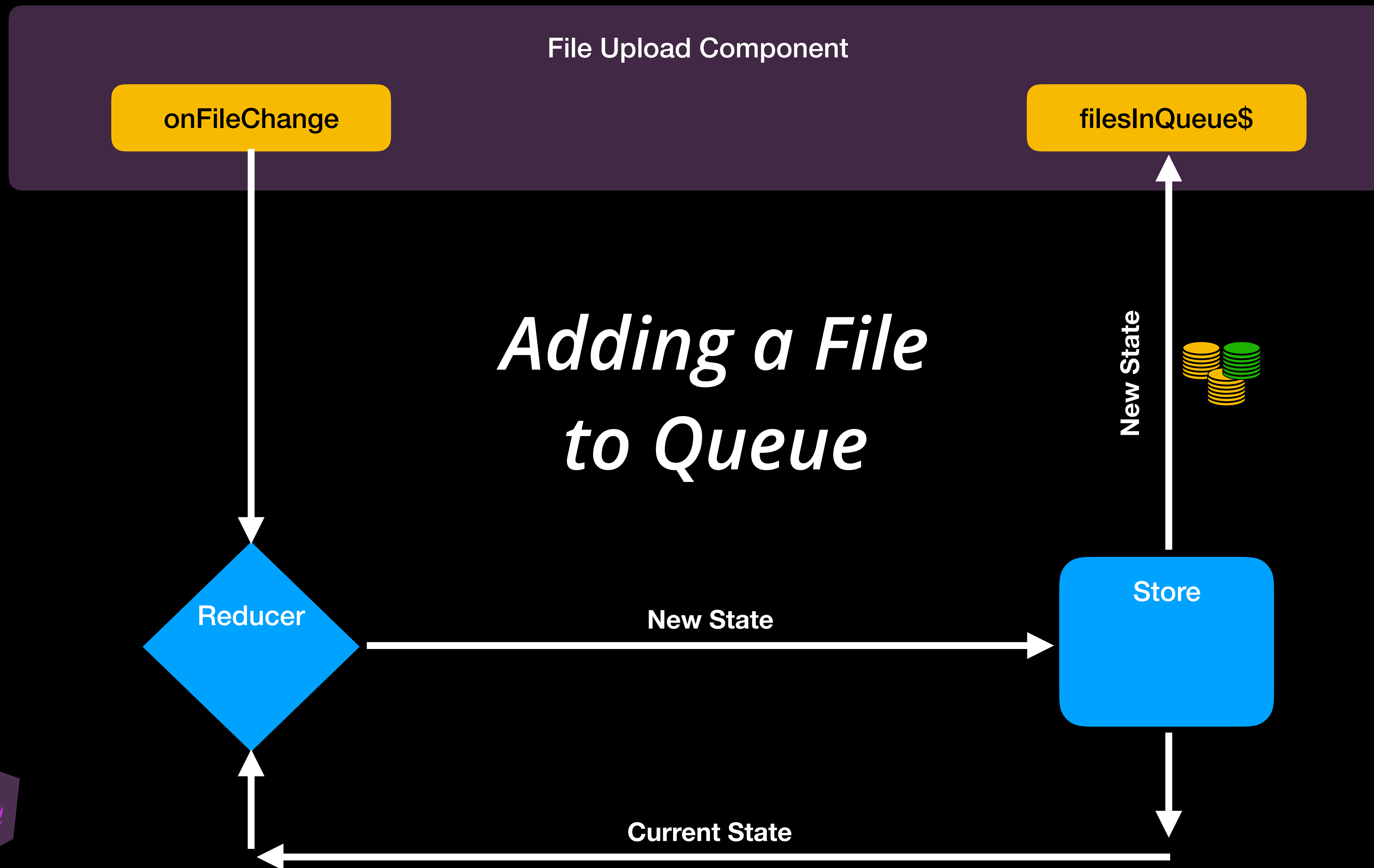
Reducer

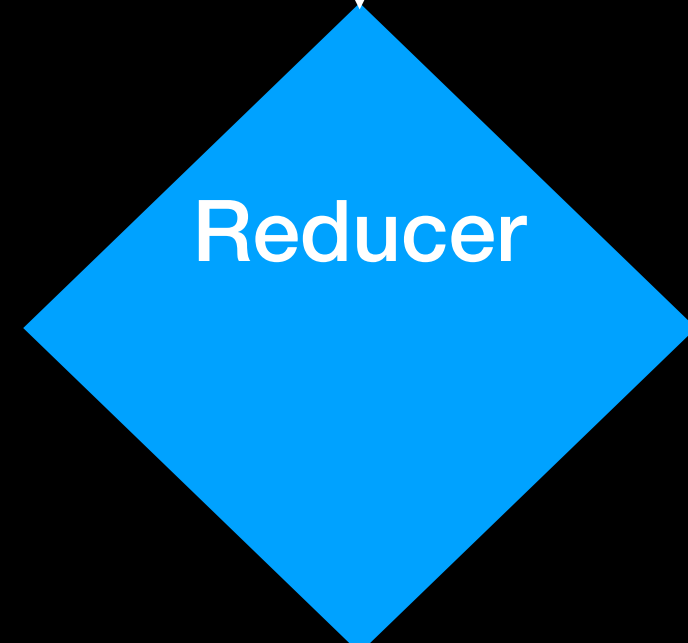
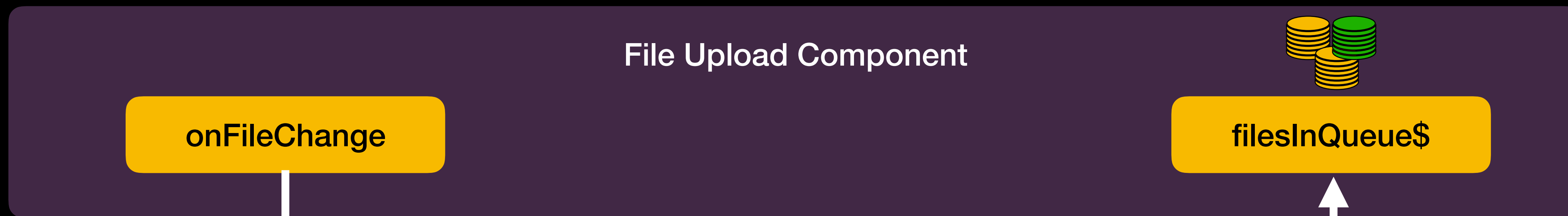
New State

Store

Current State



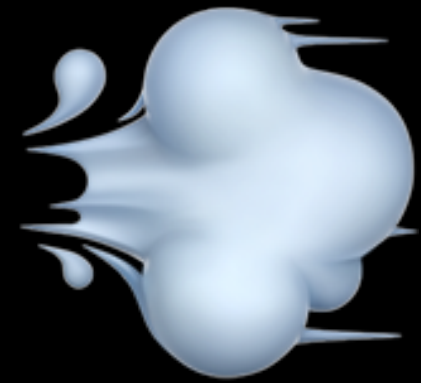




Current State

*Adding a File
to Queue*





Let's quickly look at another..



File Upload Effect

HTTP Request

File Upload Component

filesInQueue\$

*File progresses
in upload*

Reducer

Store



File Upload Effect

HTTP Request

File Upload Component

filesInQueue\$

Progress



*File progresses
in upload*

Reducer

Store



File Upload Effect

HTTP Request

File Upload Component

filesInQueue\$

Progress
Action



Reducer

*File progresses
in upload*

Store



File Upload Effect

HTTP Request

File Upload Component

filesInQueue\$

*File progresses
in upload*

Reducer

Progress

Store



File Upload Effect

HTTP Request

File Upload Component

filesInQueue\$

*File progresses
in upload*

Reducer

Progress

Store

Current State



File Upload Effect

HTTP Request

File Upload Component

filesInQueue\$

*File progresses
in upload*

Reducer



Store



Current State



File Upload Effect

HTTP Request

File Upload Component

filesInQueue\$

*File progresses
in upload*

Reducer

New State

Store

Current State



File Upload Effect

HTTP Request

File Upload Component

filesInQueue\$

*File progresses
in upload*

Reducer

New State

Store



Current State



File Upload Effect

HTTP Request

File Upload Component

filesInQueue\$

*File progresses
in upload*

Reducer

New State

Store

New State



Current State



File Upload Effect

HTTP Request

File Upload Component

filesInQueue\$



*File progresses
in upload*

Reducer

New State

Store

Current State





Let's watch it in action!





Managing File Uploads w/ NgRx

Browse Files

Upload Files

Clear Files

Cancel Upload

No files chosen

Built with





*That was a lot of
information, so..*





*..find me after and let's
continue the conversation!*





Develop like Google

We are an engineering consulting firm that helps enterprises build large apps at scale using monorepos and development practices pioneered at Google, Microsoft, and Facebook.

Creators of some fine open-source products:



+



+



Partner with us

- Consulting
- Training
- Engineering

Join our team

Sr. Angular Engineers & Architects
React Architects

nrwl.io/careers

Thank you!!

 *bit.ly/nwa-talk*

 *@wesgrimes*