```ts
type User = {
  id: string;
  name: string
}

const api = {
  getUser(id: string, name: string, callback: (result: User) => void) {
    //
  },
  uploadFile(file: File, callback: (result: boolean) => void) {
    //
  }
}
```

```ts
type Promisify<T> = {
  [K in keyof T]: T[K] extends (...args: any) => any
     ? Parameters<T[K]> extends [...args: infer Args, callback: (result: infer Result) => void]
      ? (...args: Args) => Promise<Result>
      : never
     : never;
}

type LegacyApi = typeof api;
type ModernApi = Promisify<LegacyApi>
```

```
type LegacyApi = {
    getUser(id: string, name: string, callback: (result: User) ⇒ void): void;
    uploadFile(file: File, callback: (result: boolean) ⇒ void): void;
}
```

```ts
type ModernApi = {
    getUser: (id: string, name: string) => Promise<User>;
    uploadFile: (file: File) => Promise<boolean>;
}
```

Mapped Types

```ts
type User = {
  id: string;
  name: string
}

const api = {
  getUser(id: string, name: string, callback: (result: User) => void) {
    //
  },
  uploadFile(file: File, callback: (result: boolean) => void) {
    //
  }
}
```

Mapped Types

```ts
type User = {
  id: string;
  name: string
}
```

```ts
type LegacyApi = {
    getUser(id: string, name: string, callback: (result: User) => void): void;
    uploadFile(file: File, callback: (result: boolean) => void): void;
}
```

```ts
type ModernApi = {
    getUser: (id: string, name: string) => Promise<User>;
    uploadFile: (file: File) => Promise<boolean>;
}
```

er Result) => void]

```ts
type LegacyApi = typeof api;
type ModernApi = Promisify<LegacyApi>
```