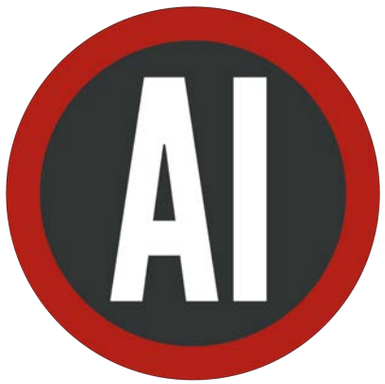


Building a Highly Scalable File Processing Platform with NServiceBus

Sam Martindale

Introduction

Sam Martindale

Architecting Innovation

14+ Years Development Experience

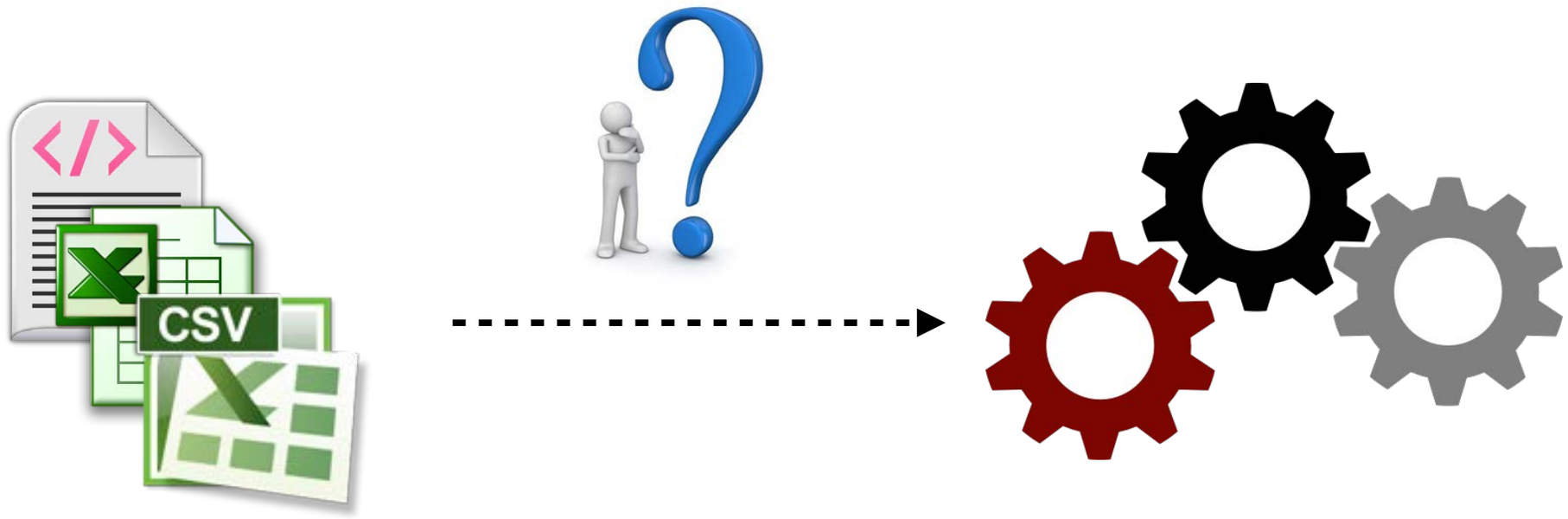
Co-Founder & Managing Partner, AI

NServiceBus adopter since v1.x

The Problem

The Problem

How do we move this data into our systems?

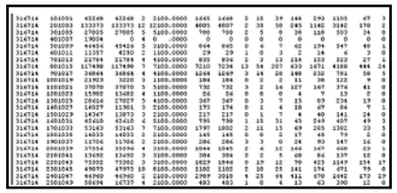


External Data Files

Internal Systems

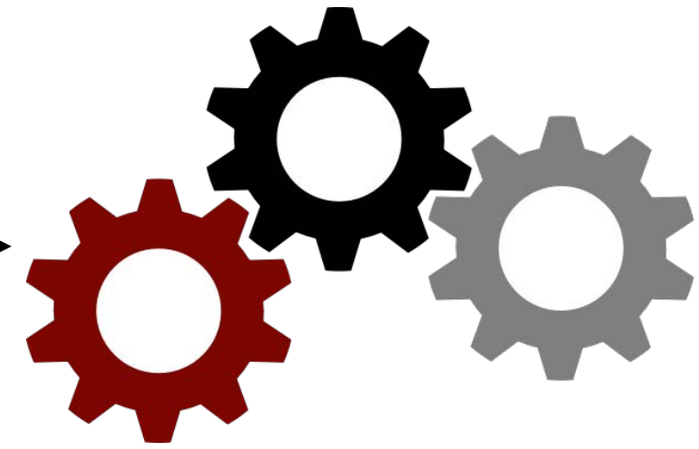
The Problem

Automated pickup of many file types



Fixed Width

Record Level Logic and Workflows



New Account

Account Payment

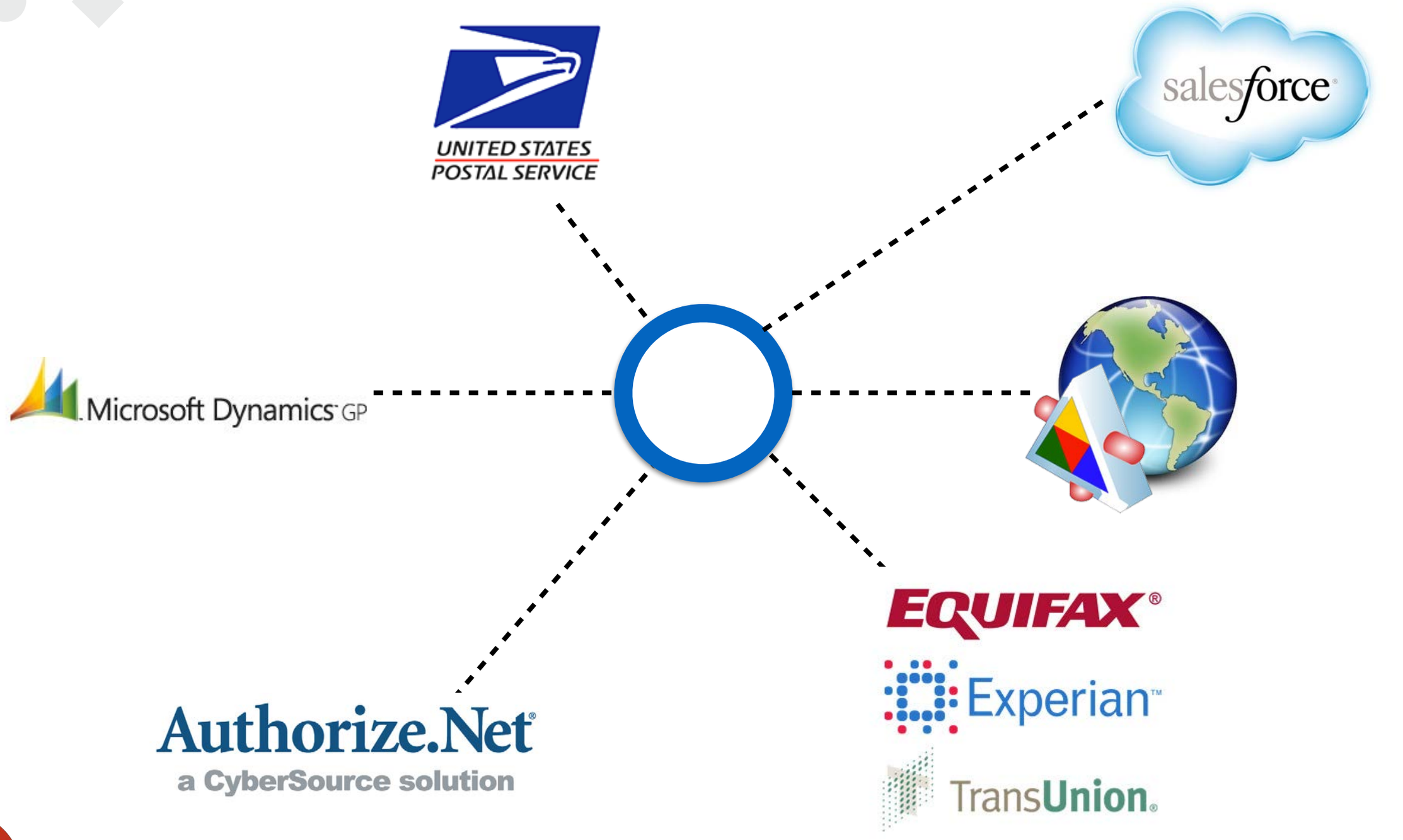
Loan Application

Account Payment

Overdue Credit Account

The Problem

Many Integration Points



The Problem

Guaranteed Delivery

© Leslie Jones/Boston Public Library

The Problem

Scalability

Why NServiceBus?

Evaluated Technologies

Sql Server Integration Services (SSIS)

- Workflow Integration Difficulty
- Systems Integration Difficulty
- Extensibility
- Not just ETL

Microsoft® SQL Server® Integration Services

Evaluated Technologies

BizTalk/Sterling Commerce/Axway/Others

- Licensing Costs
- Scalability Costs
- Difficult to “extend” functionality easily
- Requires development “specialists”

Evaluated Technologies

NServiceBus

- Easy & inexpensive to scale
- Easy to integrate with existing systems
- Provides Message Durability
- Great API for experienced .NET developers

System Design

The System

Api Endpoint

SignalR

Scheduler Endpoint

Alerts Endpoint

FileRetrieval Endpoint

Processing Endpoint

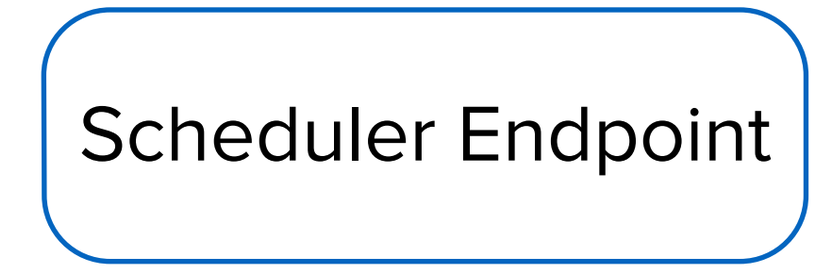
Parsing Endpoint

File Management Endpoint

Components

Scheduler

Manages scheduled “jobs” for picking up client files from FTP, SFTP, FileShare, etc



Why we needed it:

1. Needed the ability to schedule file pickups dynamically

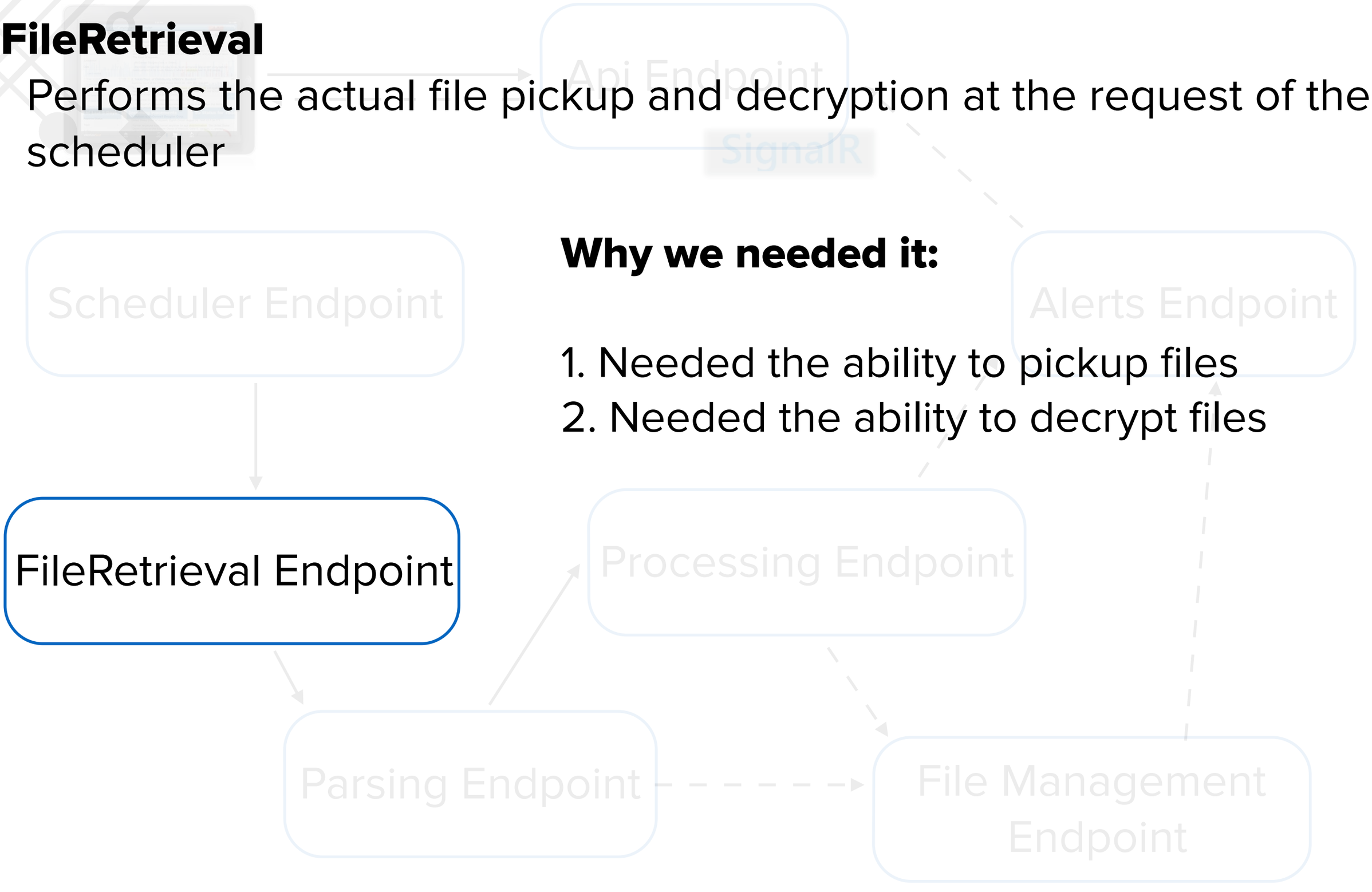
Components

FileRetrieval

Performs the actual file pickup and decryption at the request of the scheduler

Why we needed it:

1. Needed the ability to pickup files
2. Needed the ability to decrypt files



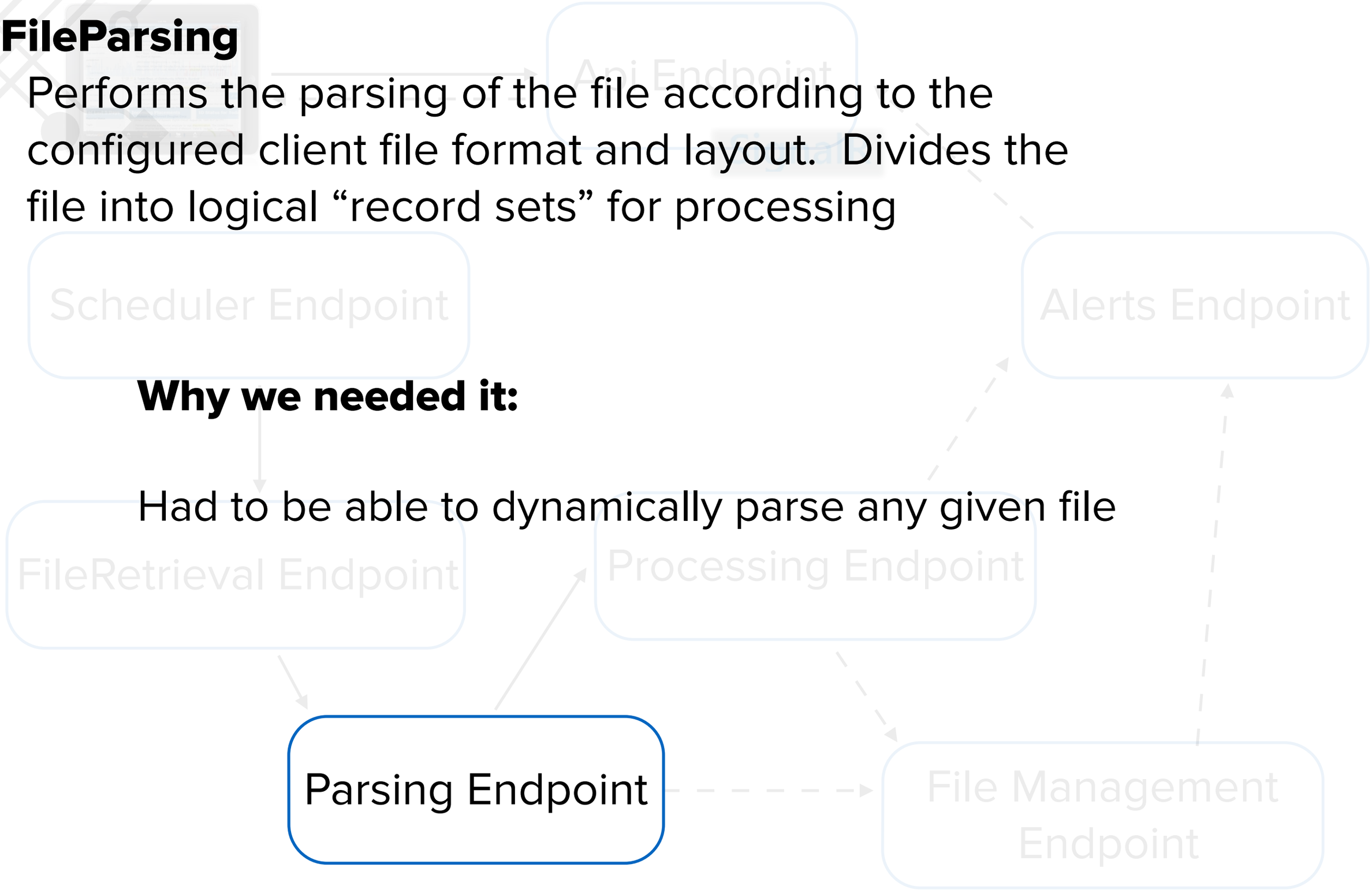
Components

FileParsing

Performs the parsing of the file according to the configured client file format and layout. Divides the file into logical “record sets” for processing

Why we needed it:

Had to be able to dynamically parse any given file



Components

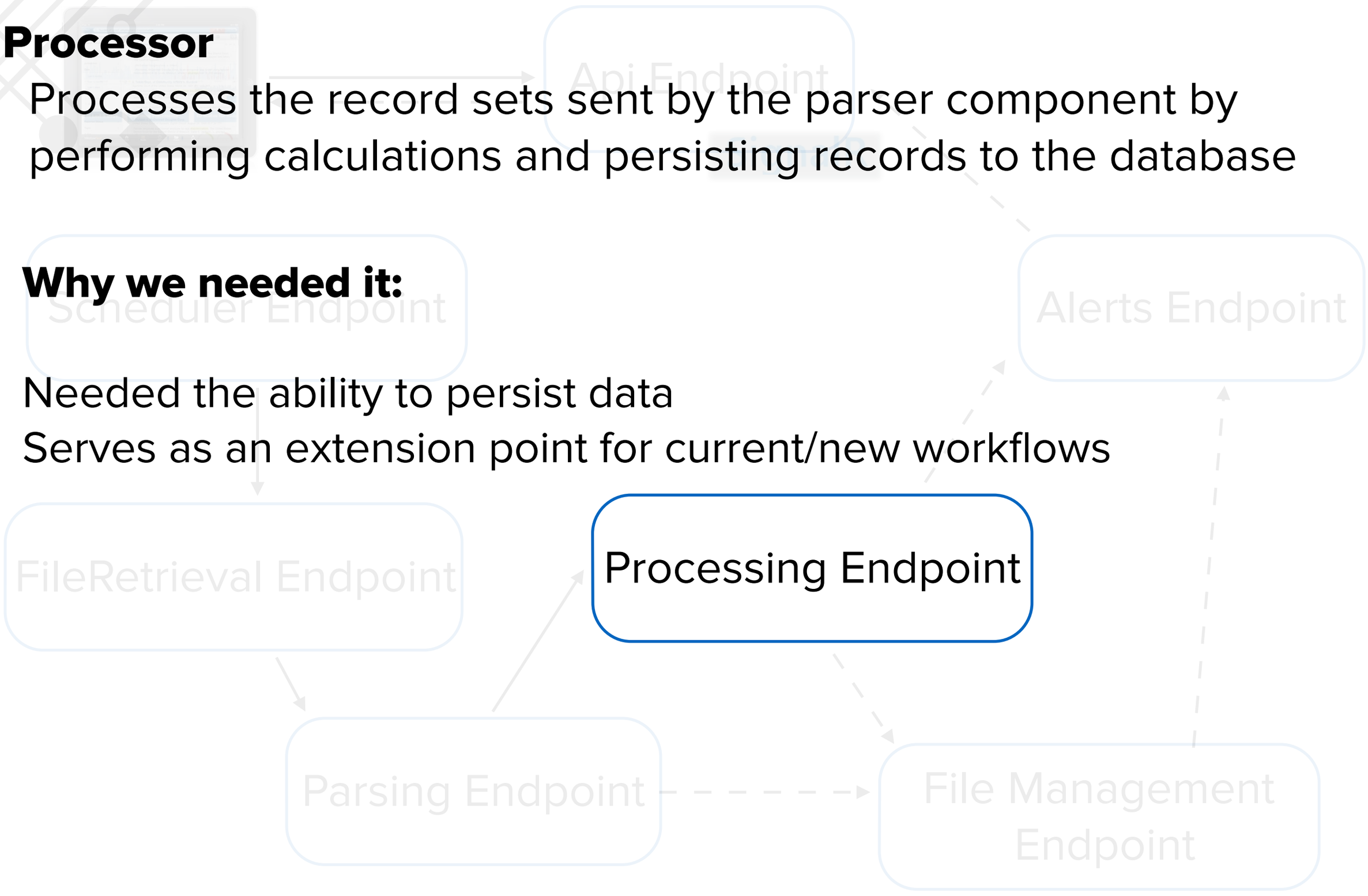
Processor

Processes the record sets sent by the parser component by performing calculations and persisting records to the database

Why we needed it:

Needed the ability to persist data

Serves as an extension point for current/new workflows



Components

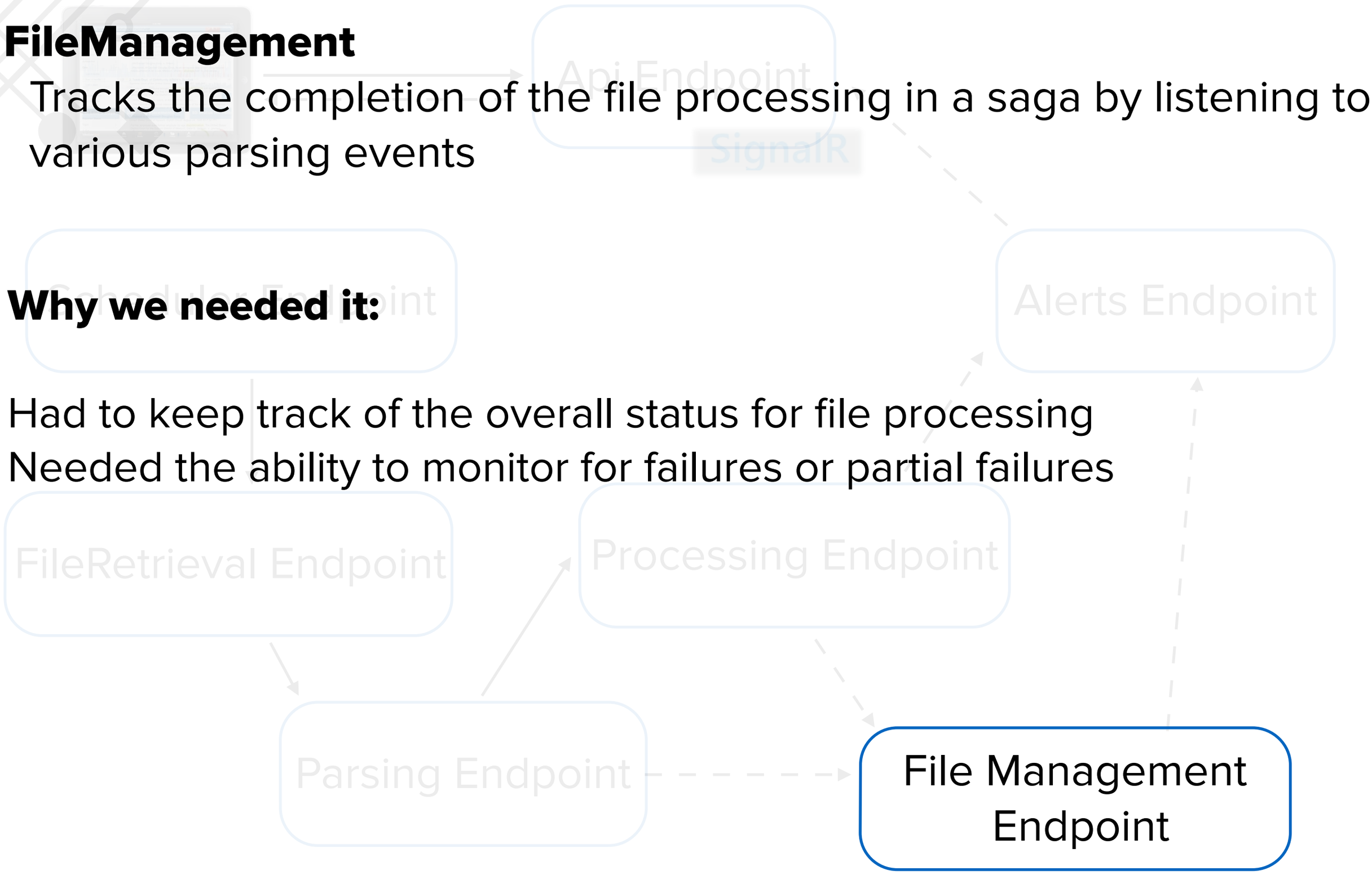
FileManagement

Tracks the completion of the file processing in a saga by listening to various parsing events

Why we needed it:

Had to keep track of the overall status for file processing

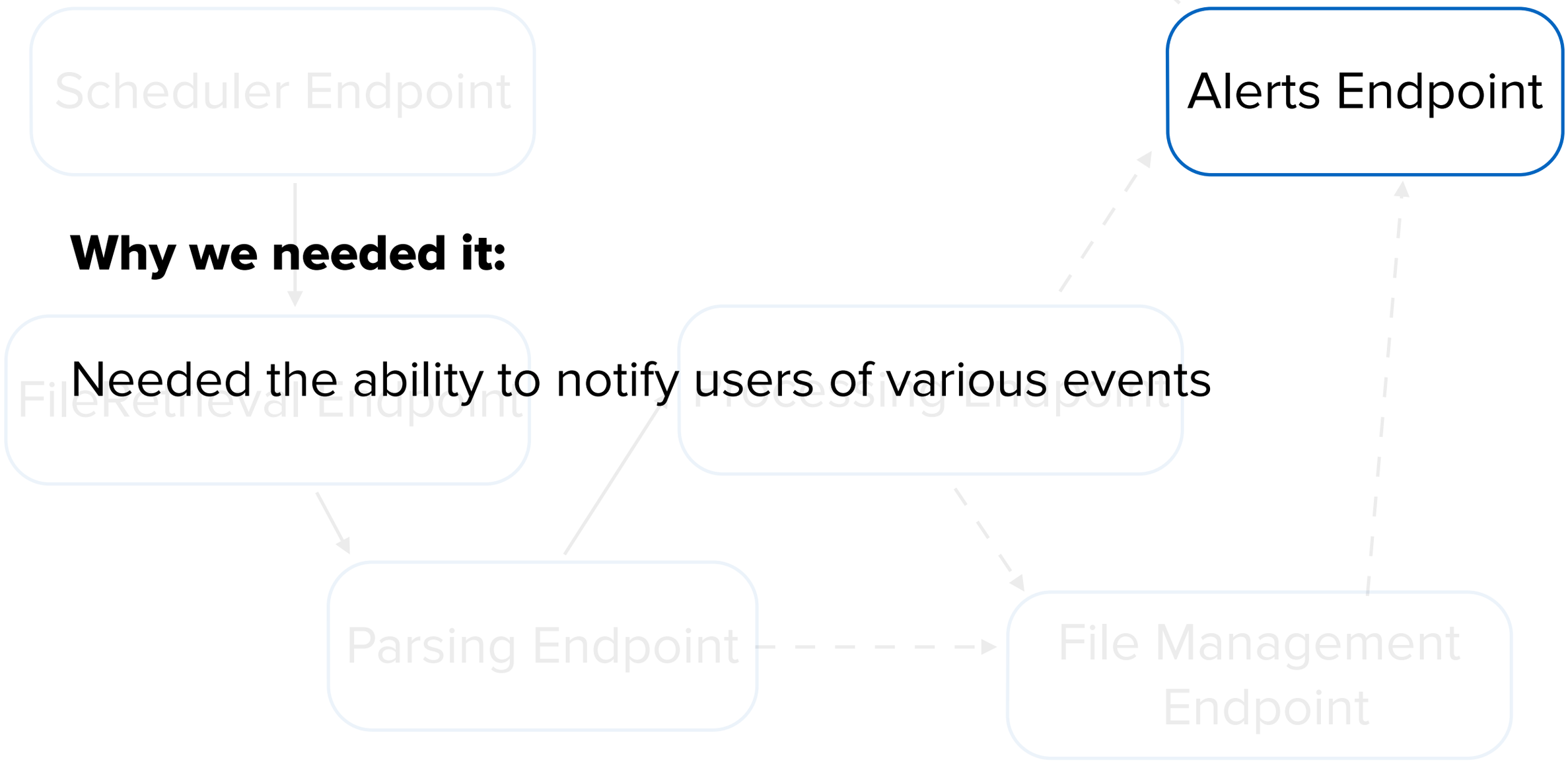
Needed the ability to monitor for failures or partial failures



Components

Alerts

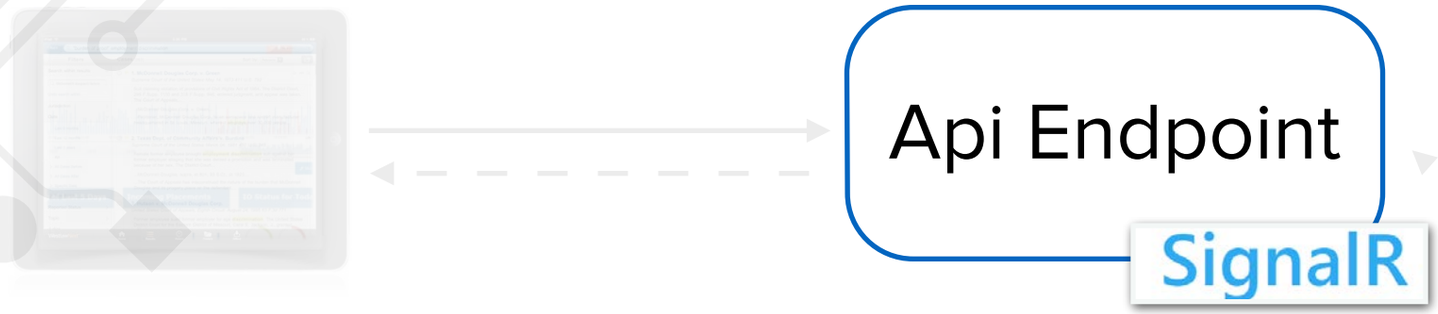
Receives events from other components, sending notifications and emails to the end users for system monitoring and alerting



Why we needed it:

Needed the ability to notify users of various events

Components



WebApi

WebApi component with SignalR. Delivers real time events and updates to the web front end

Why we needed it:

Needed to notify users of created alerts in real-time
Needed to allow users to quickly onboard clients

The System

Api Endpoint

SignalR

Scheduler Endpoint

Alerts Endpoint

FileRetrieval Endpoint

Processing Endpoint

Parsing Endpoint

File Management Endpoint

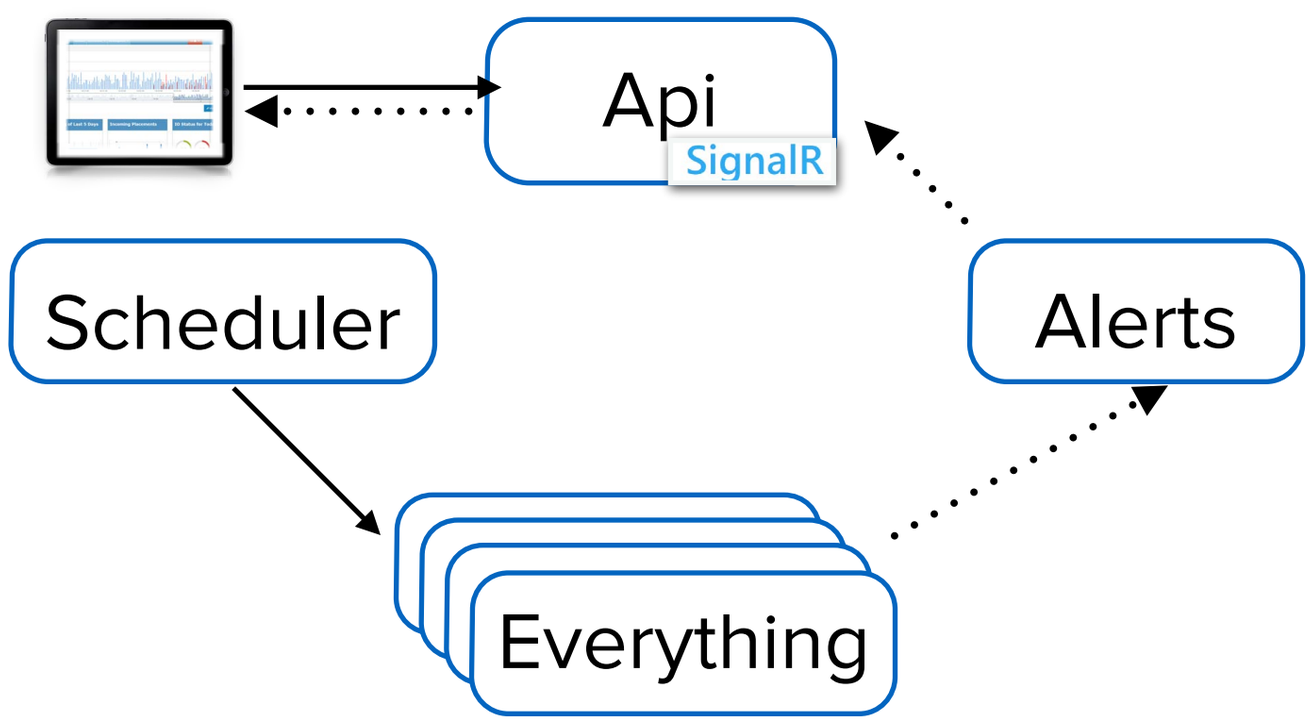
What did we learn along the way?

Division of Labor

Problem

Monolithic Endpoints

A naive implementation of our service placed far too many handlers within the same endpoint, making it impossible to prioritize messages and scale portions of the system



Division of Labor

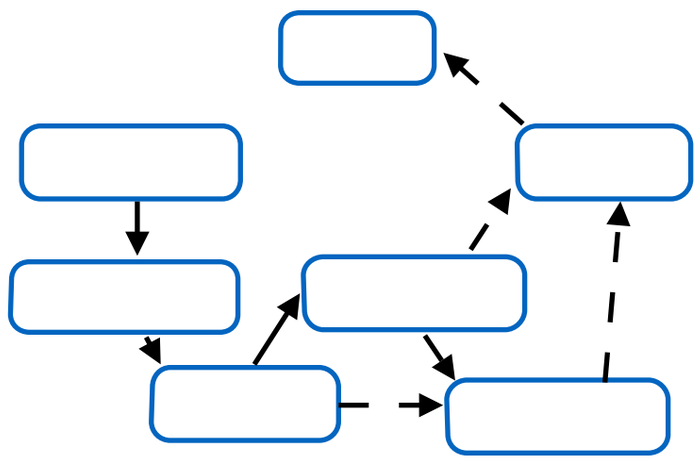
Resolution

A Service is NOT an endpoint

In order to ensure our success, we had to split the system into many components, each with its own distinct responsibilities (and queues!)

Granular Scaling

By splitting into several disparate components, we were able to meet our scalability needs more appropriately. (e.g. scaling only the parsing endpoint)

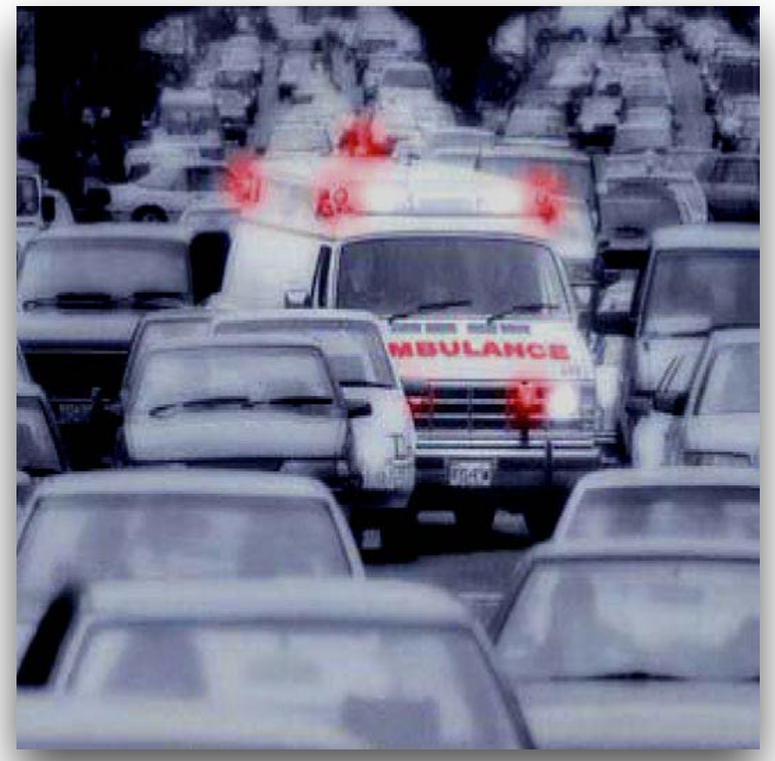


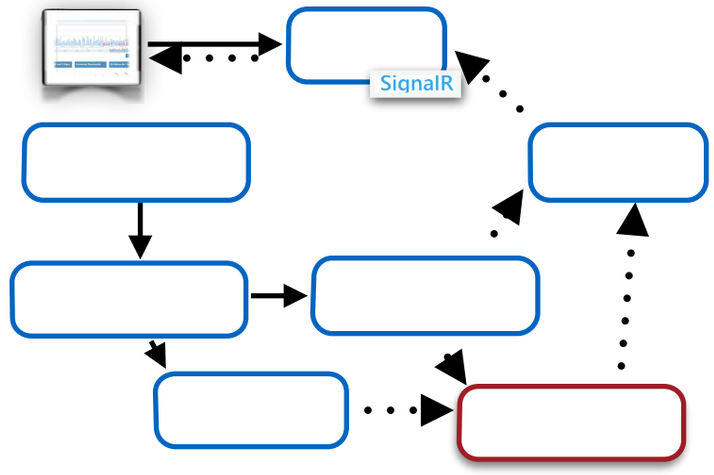
Saga Responsibilities

Problem

Scaling the File Processing Saga

Our initial (read: naive) implementation was responsible for far too much business logic, making it difficult to scale





Saga Responsibilities

Resolution

Keep it Simple!

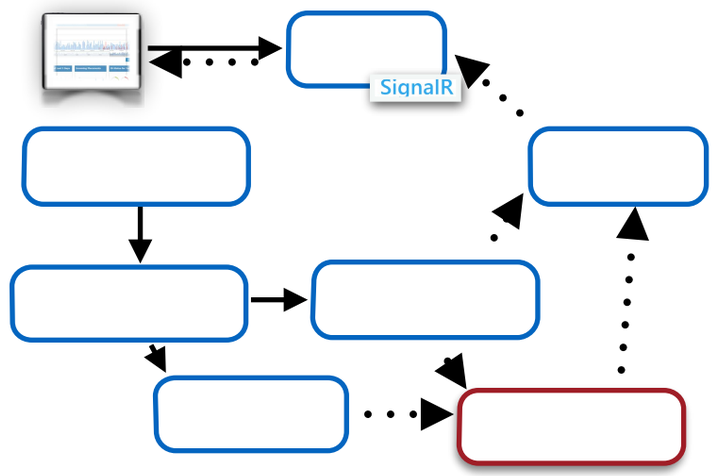
Sagas should be as clean and tight as possible

Scaling Issues

The more complicated the saga, the harder they are to scale

“Observer" Implementation

In our case, we found that a “passive” saga responding to events and simply keeping track of state was far easier to maintain and scale

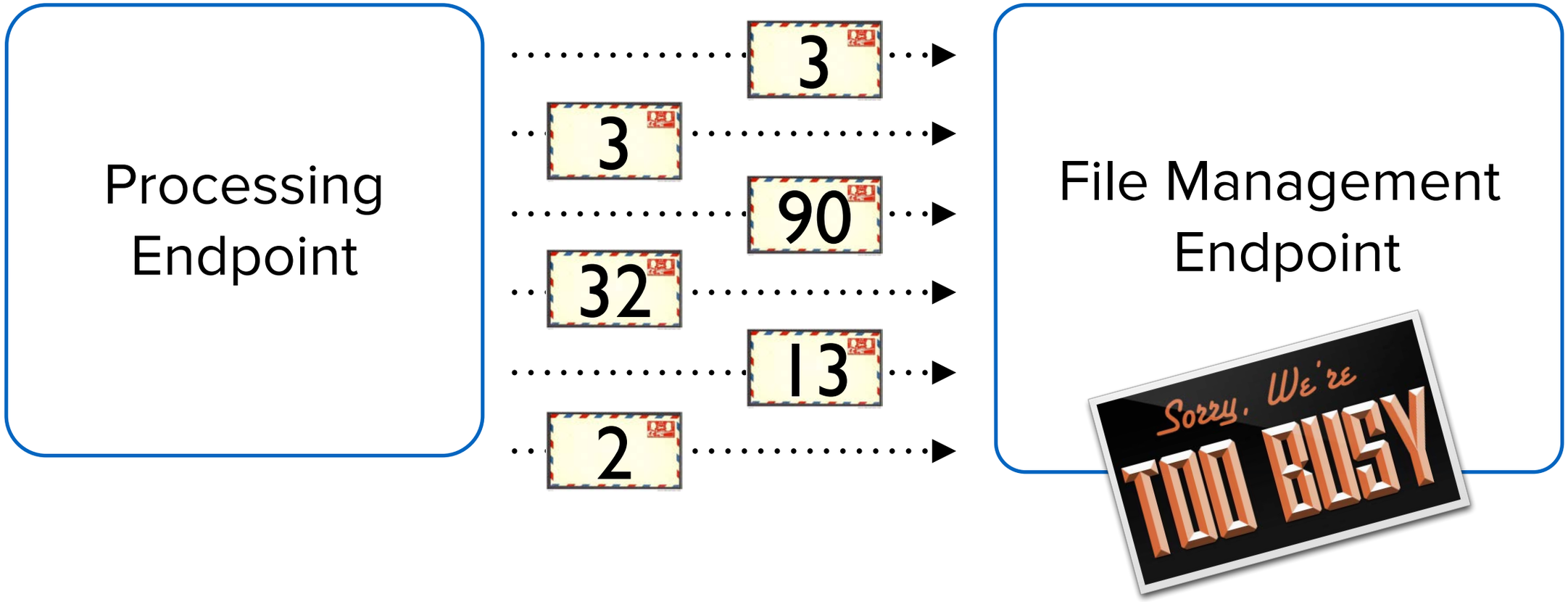


Saga Methodology

Problem

Saga Contention

Our initial (read: naive) implementation responded to too many events, creating a large degree of contention and a high number of rollbacks

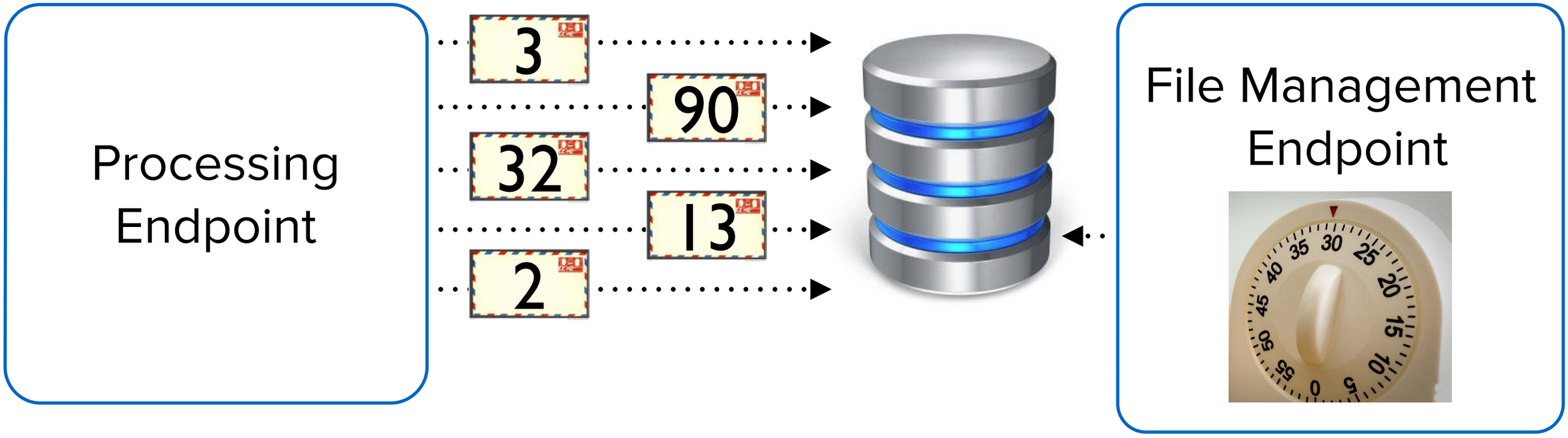


Saga Methodology

Resolution

“Check” Pattern

In order to reduce contention, we implemented a “check” pattern using timeouts within the saga

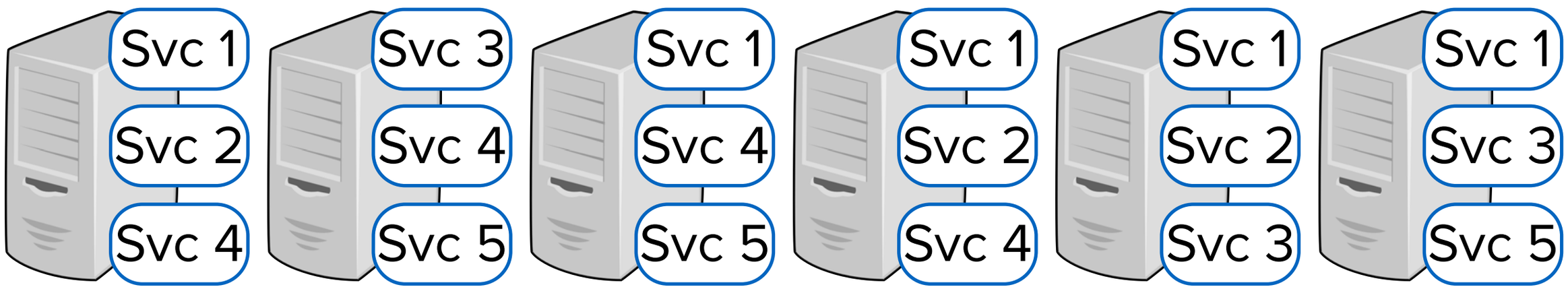


Deployment Automation

Problem

Deployment Challenges Hindering Proper Design

The team (and client) were initially skeptical of breaking the system into many disparate deployable components, due to the increased difficulty of deployment



Deployment Automation

Resolution

Simple Automation

By leveraging the *NServiceBus.Powershell* bits along with our own extended *Powershell* scripts, we were able to automate the deployment of **all** of our NServiceBus projects

- Configuration Transformations
- Distributed Deployments
- Backup Strategy

Firewalls, Servers & Failure, Oh My!

Problem

Deploying Into New Environments is Hard!

Misconfiguration, network issues, security issues… All these make moving into a new environment cumbersome and error prone!

Firewalls, Servers & Failure, Oh My!

Resolution

ServicePulse, ServiceControl & Custom Checks

- *MSDTC* Firewall Checks
- *RavenDB* Firewall Checks
- FTP Accessibility Checks
- Databus File Share Accessibility Checks

The Result?

The Result

Business Impact

Faster Processing Times

> 10K records/sec parsed and processed, per worker machine

Happier Users

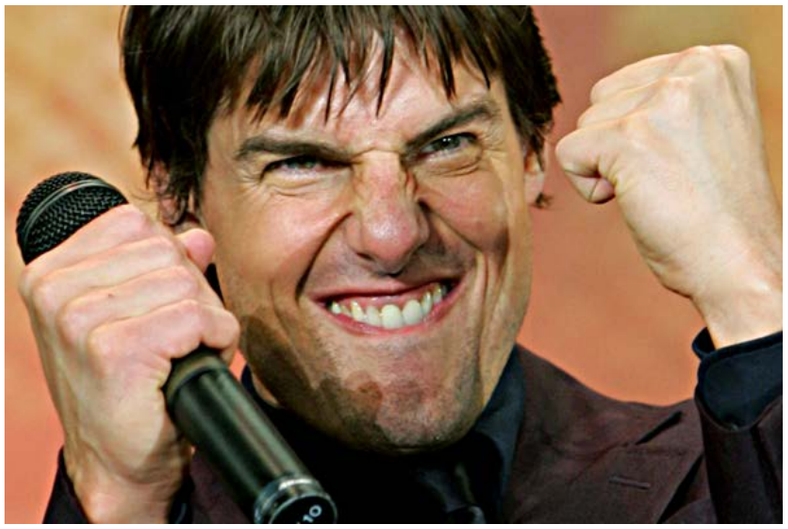
Full system visibility and no lost data

Happier Developers

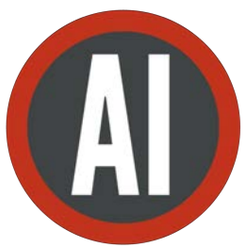
Clean, maintainable system

Happier Executives

Faster onboarding times for new clients



Thank You



Particular Software