ネットワーク監視の自動化はどこまでできるのか？
-Apache Airflowによる アラート対応基盤-

LINEヤフー株式会社
上岡 輔乃 / Honai Ueoka

LINEヤフー

自己紹介

上岡 輔乃 / Honai Ueoka

LINEヤフー株式会社

愛媛県喜多郡内子町出身

- 松山市内の高校に通っていました！

経歴/JANOG歴

- 2023年4月 ヤフー株式会社 新卒入社
- 2025年1月 JANOG55 京都 現地参加

業務

- データセンターネットワーク構築・運用
- ネットワーク運用基盤・ツール開発

ネットワーク監視の自動化はどこまでできるのか?

-Apache Airflowによるアラート対応基盤-

1. 背景と課題
2. Apache Airflow
    1. Apache Airflowの紹介
    2. NW運用基盤としてのApache Airflow
    3. ワークフロー内でのAI活用
3. NWアラート対応基盤 “oyakata”
4. oyakata導入の結果・今後の課題・まとめ

ネットワーク監視の自動化はどこまでできるのか?

-Apache Airflowによるアラート対応基盤-

1. 背景と課題
2. Apache Airflow
    1. Apache Airflowの紹介
    2. NW運用基盤としてのApache Airflow
    3. ワークフロー内でのAI活用
3. NWアラート対応基盤 “oyakata”
4. oyakata導入の結果・今後の課題・まとめ

LINEヤフーのデータセンターネットワーク

LINEヤフー環境
ネットワーク

旧LINE環境
ネットワーク

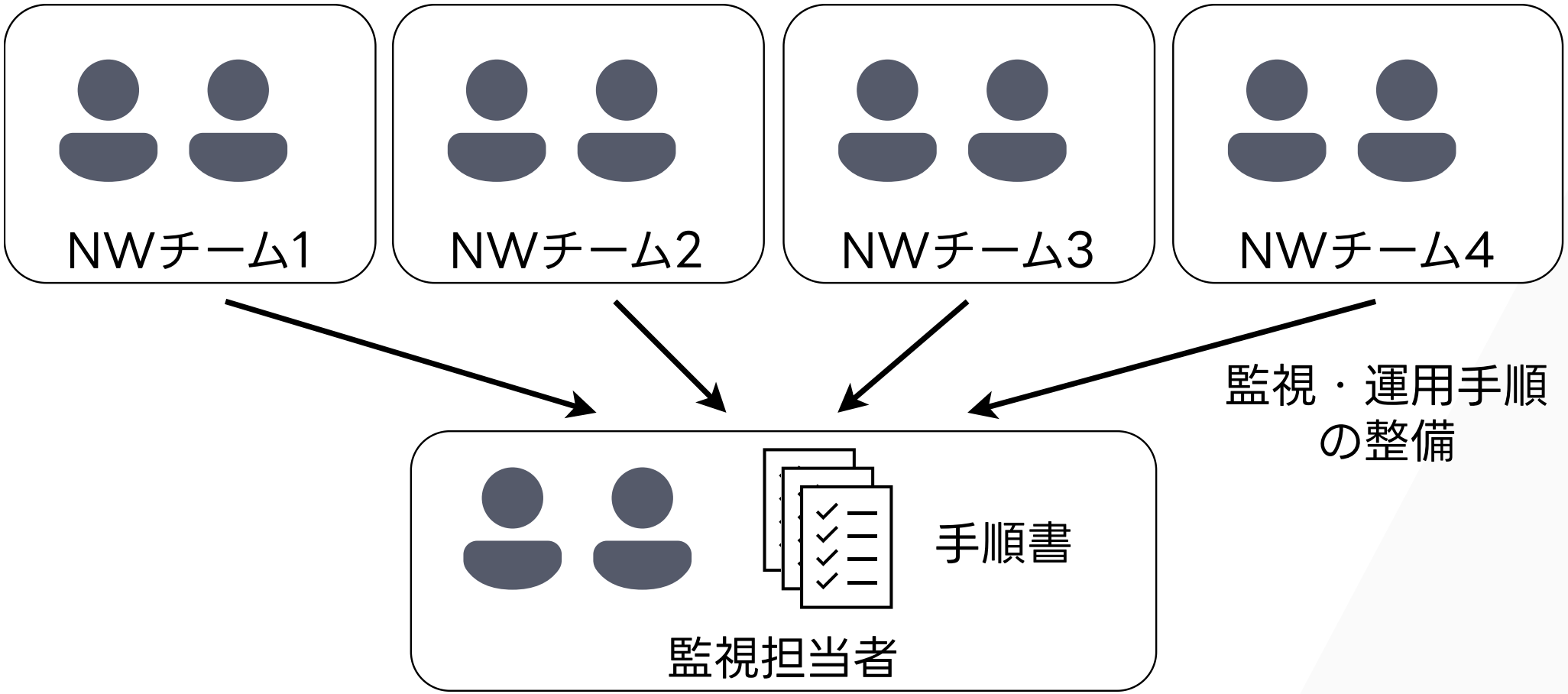
旧ヤフー環境
ネットワーク

多様なネットワークを運用
大規模：計22,000台以上のNW機器

LINEヤフーでは数多くのサービスをオンプレのネットワークで提供

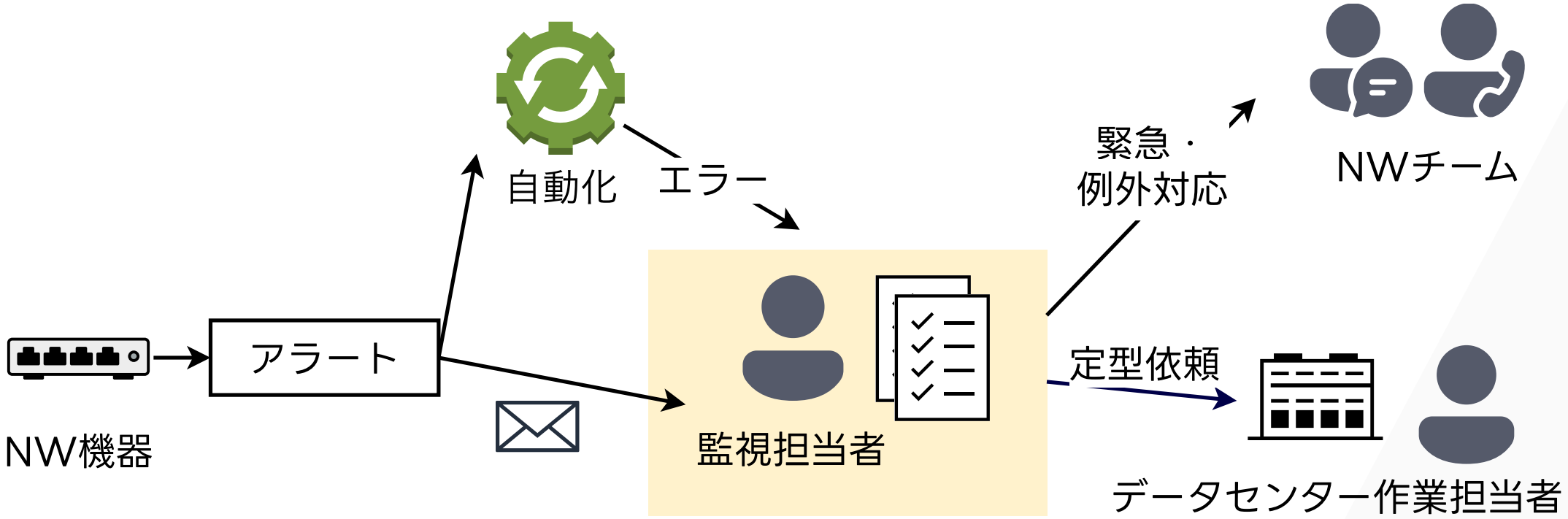
設計やポリシーが異なる多様な大規模ネットワークを運用

従来のネットワーク運用体制



扱うネットワークごとに複数のNWチーム が編成 NW構築や運用を分担

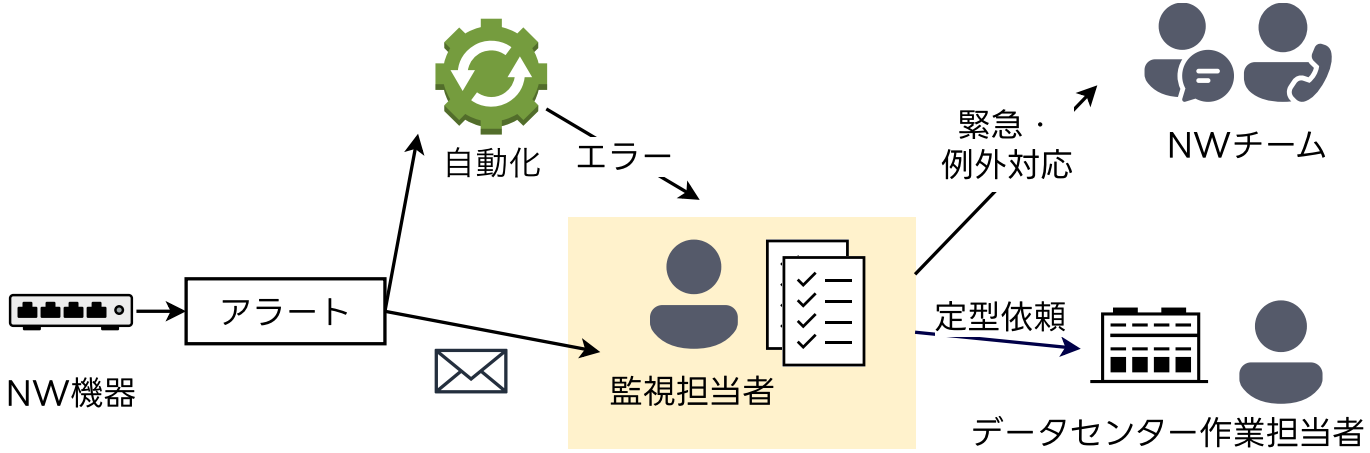
従来のネットワーク運用体制



24/365常駐の監視担当者による一次対応を前提 としたNW運用

課題1: 人手のボトルネック

再掲



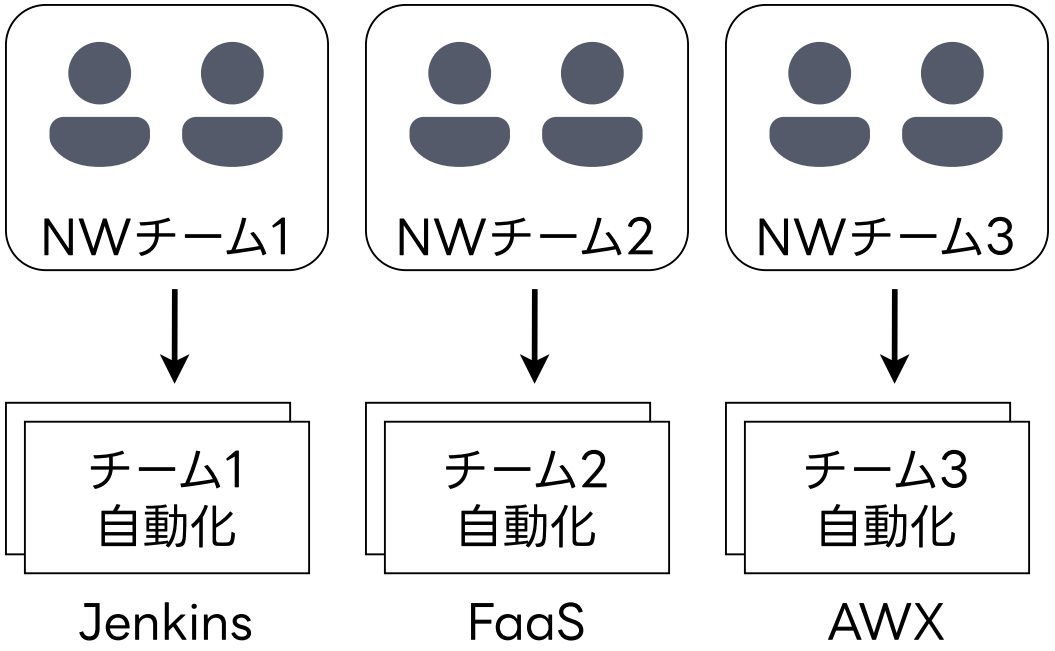
人手を前提にしたアラート対応で運用工数が取られている

従来の運用は「成熟」していても

改善の速度や運用工数・負荷、育成などの「継承」に限界

例：監視チームの運用負荷、手順の複雑化、
担当者による精度のブレ、改善速度（担当者間の調整）

課題2: 自動化の乱立



アラート対応の**自動化を実施してきた**

しかし…

NWチームごとに**独自の自動化が乱立**

- 基盤もバラバラ
- 重複した実装
- 自動化のブラックボックス化/属人化

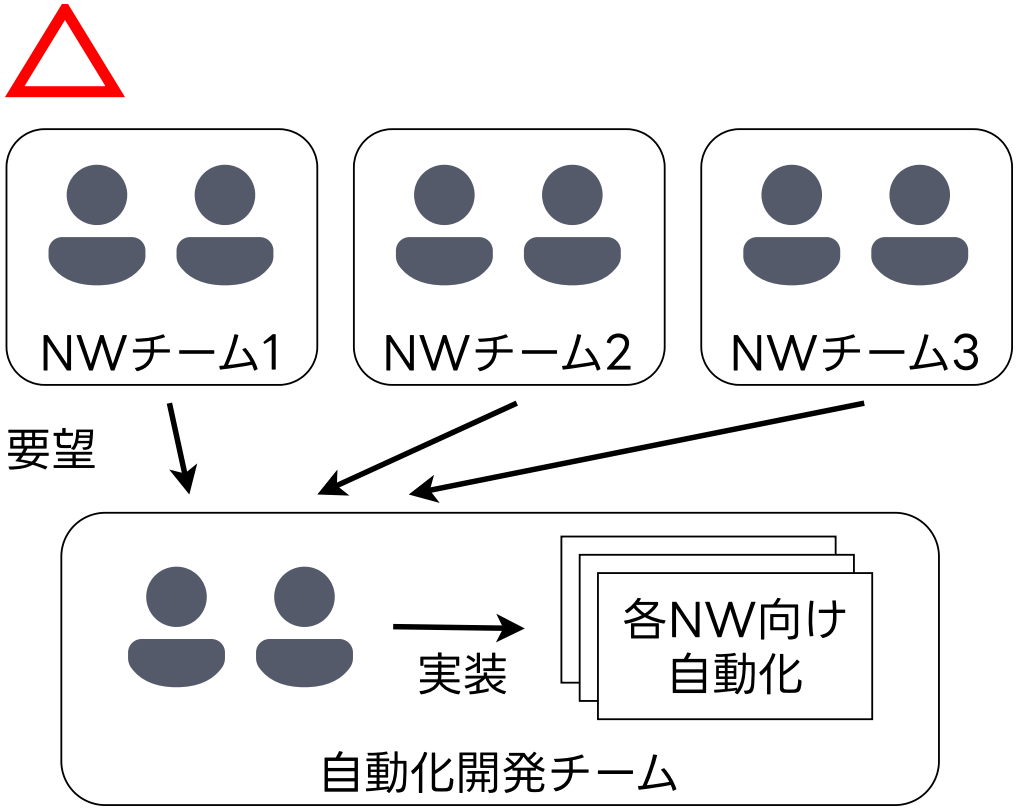
NWチームが集中したいのは

⭕ 自動化によって運用工数を減らす

❌ ~~基盤の構築やメンテナンス~~

課題2: 自動化の乱立

中央集権的な自動化を行えば良いわけでもない



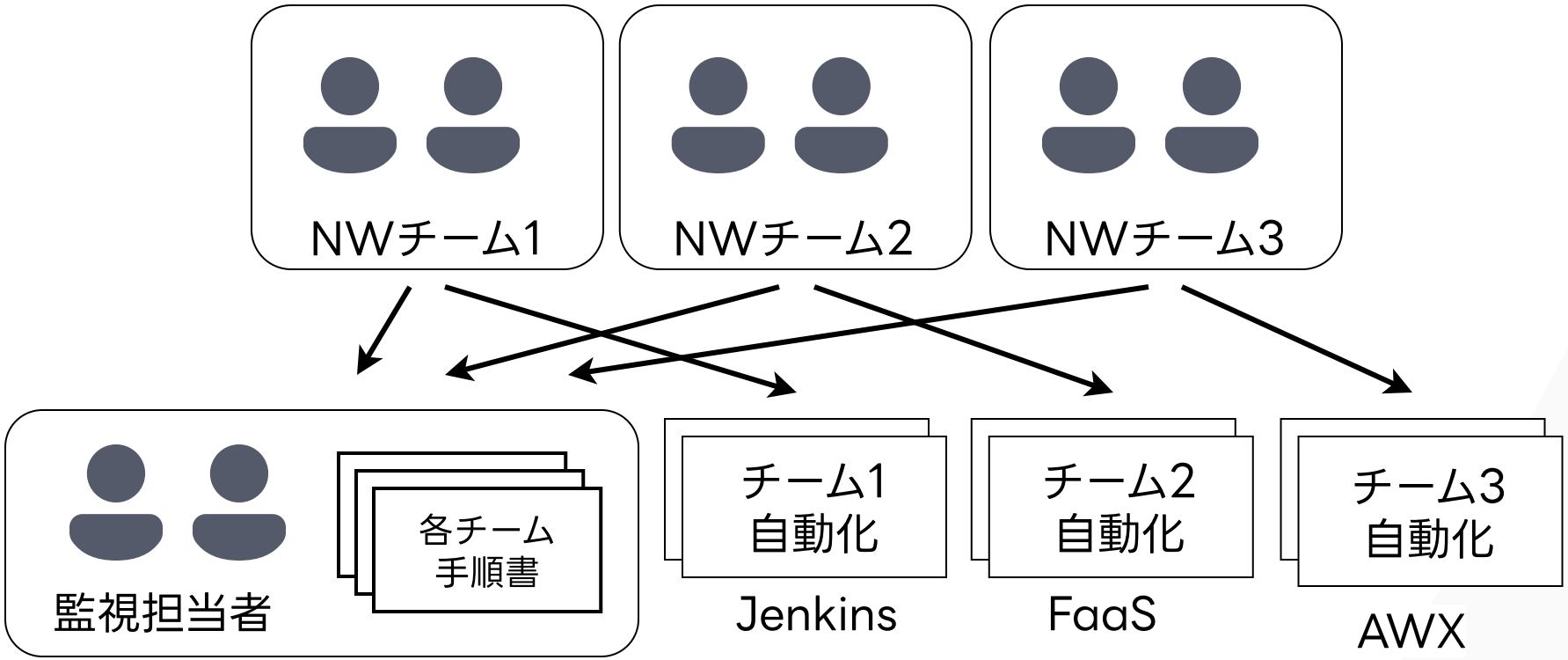
開発専門の組織が自動化を全て開発?

- 開発組織は各NWのドメイン知識を有していない
- NW・自動化チーム間の調整が発生

NWの運用知識を持つ各チームが
直接自動化を整備できるのが望ましい

自動化基盤に必要なこと

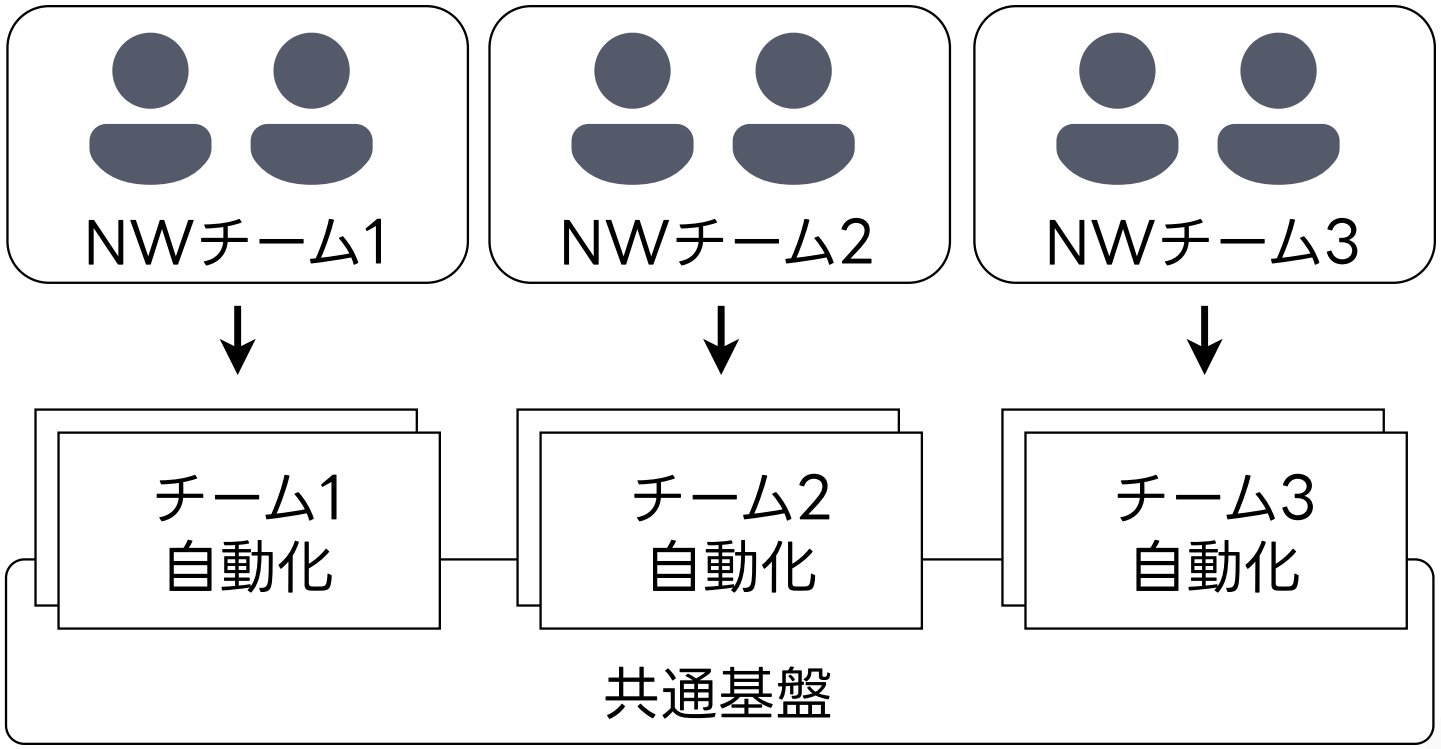
これまで



- 人手による手順書ベースの一次対応が前提
- チームごとに乱立した自動化

自動化基盤に必要なこと

あるべき姿



- 人手の一次対応を自動化に置き換え
- 自動化の乱立を避け各チームが直接自動化を実装できる**共通基盤**が必要

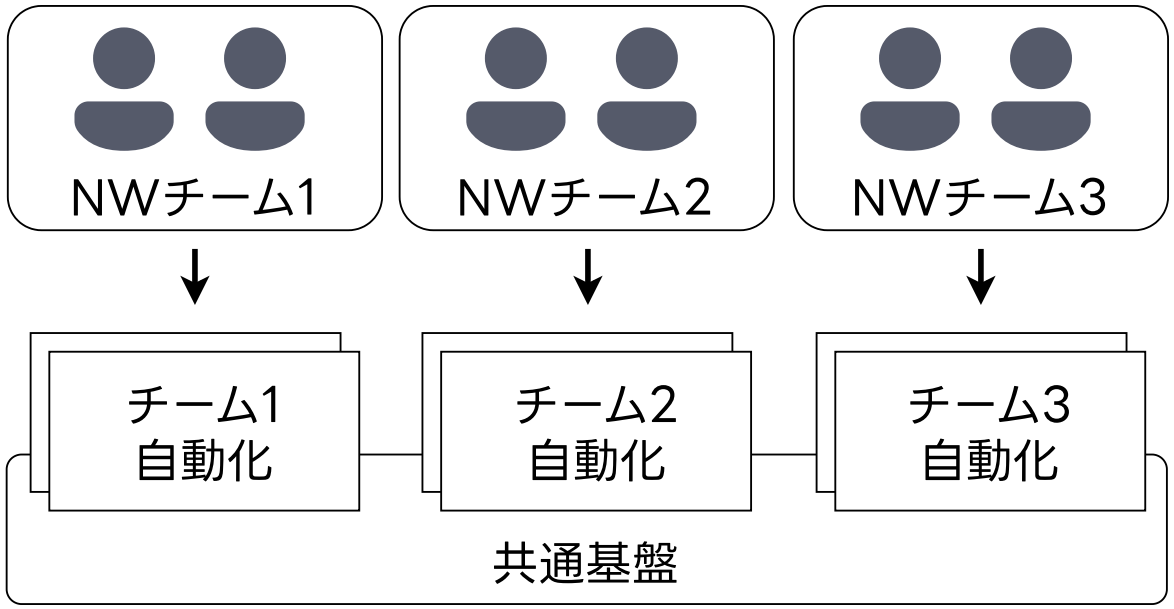
課題を解決するための取り組み

「NWアラート対応を人手をかけずに全て自動実行できる」を実現したい

- 課題1: 人手によるボトルネック
  → アラートの一次対応を、人手によるものから
  基盤上で自動実行し必要な時だけ人に渡す運用へ切り替える
- 課題2: 自動化の乱立
  → ネットワーク**各チームで使える共通基盤**を構築
  運用者が本質的な自動化の実装に注力できるようにする

共通基盤に求める条件

再掲



- 導入の容易性
- 共通化がしやすいこと
- 実行環境の自由度が高いこと
- 自動化の管理がしやすいこと

基盤となるワークフローランナーの候補

フルスクラッチ

❌ 導入容易性
0から開発は工数大

社内FaaS

❌ 実行環境自由度
Workerが固定

AWX

❌ 実行環境自由度
Ansible Playbook前提

Jenkins

🔺 共通化
Java/Groovy前提

Apache Airflow

⭕ 導入容易性
OSSのためすぐ始められる

⭕ 実行環境自由度
Python/任意のコンテナ

⭕ 共通化
Operatorによる共通化

⭕ 自動化の管理
ワークフローの可視化

ネットワーク監視の自動化はどこまでできるのか?

-Apache Airflowによるアラート対応基盤-

1. 背景と課題
2. Apache Airflow
    1. Apache Airflowの紹介
    2. NW運用基盤としてのApache Airflow
    3. ワークフロー内でのAI活用
3. NWアラート対応基盤 “oyakata”
4. oyakata導入の結果・今後の課題・まとめ

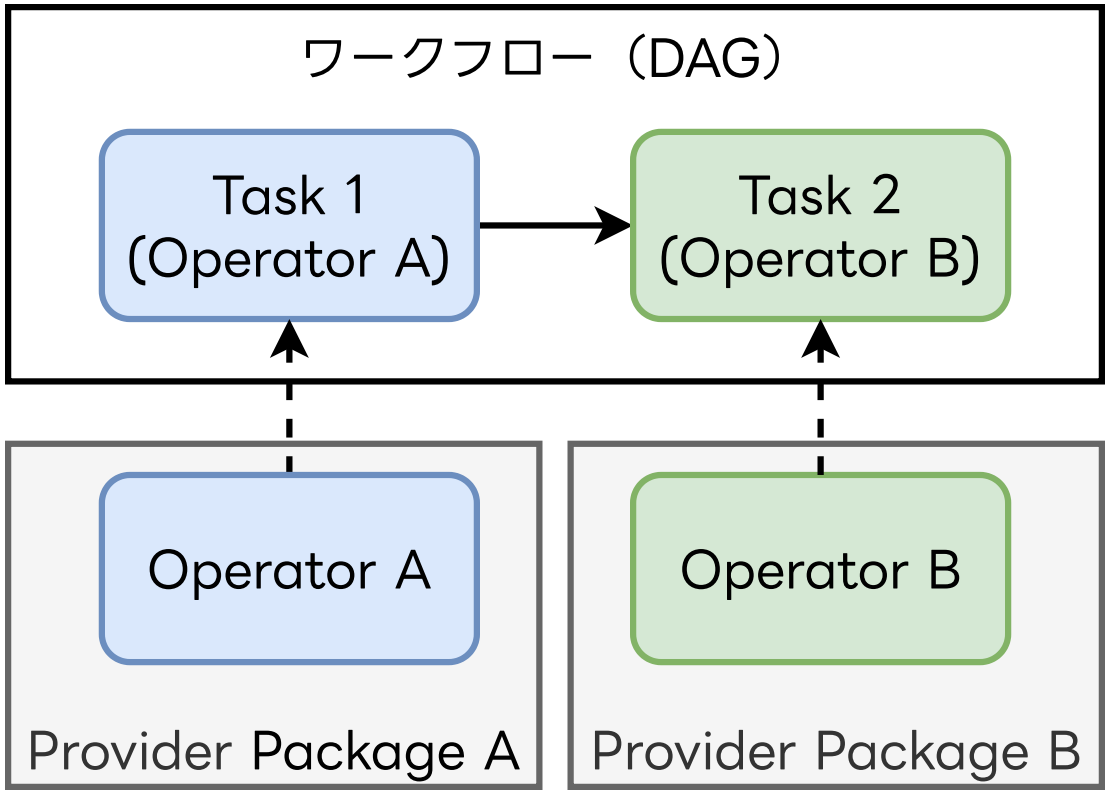
Apache Airflowとは?

- Python製のワークフローランナー
- OSSでアクティブに開発
  データ処理基盤などを中心に利用
- ワークフローをDAG（Directed Acyclic Graph、有向非巡回グラフ）と呼ばれるPythonコードとして記述
- ワークフローを可視化して管理し、スケジュール・モニタリング可能

タスクの依存関係と実行履歴の可視化

タスクの依存関係と実行履歴の可視化

OperatorとTask

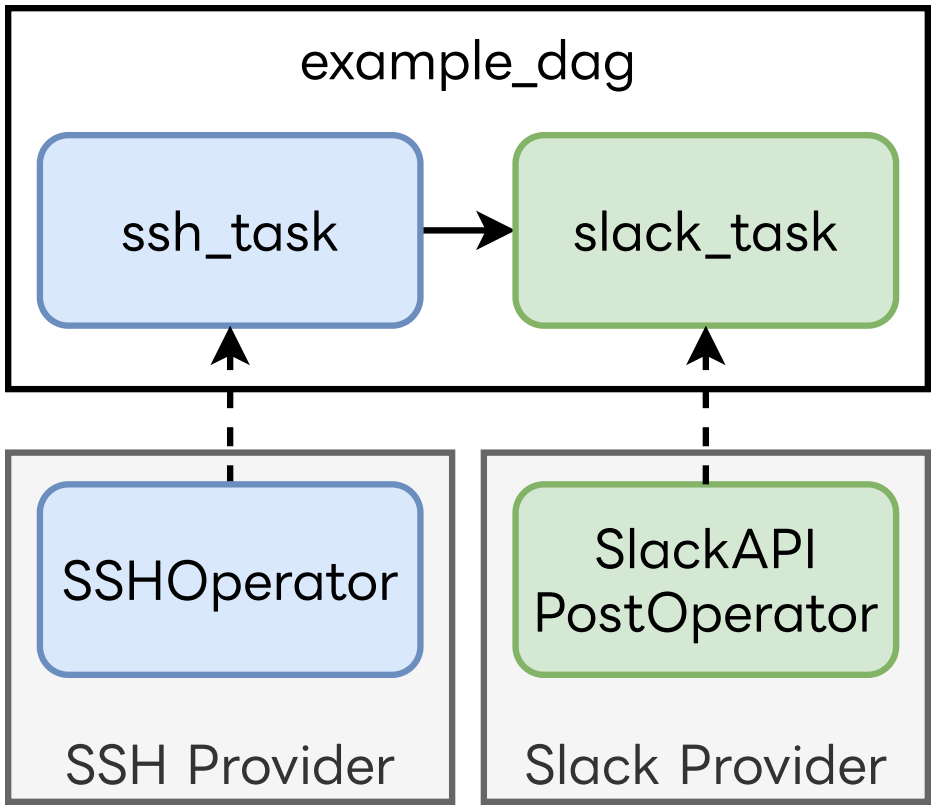


ワークフロー（DAG）は
複数のTask（処理）からなる

- Task：Operatorを利用して作成
- Operator：処理をテンプレート化して再利用できるパーツのようなもの
- Provider：Operator等を提供するパッケージ

OperatorとTask（具体例）

1. `show version` 実行
2. 結果をSlack通知する




```python
@dag(schedule=None)
def example_dag():
  ssh_task = SSHOperator(
    task_id="ssh",
    remote_host="host1",
    command="show version",
  )

  slack_task= SlackAPIPostOperator(
    task_id="slack_task",
    text="{{ ti.xcom_pull('ssh') }}",
    channel="#notification"
  )

  ssh_task >> slack_task
```

Operator等を提供するProvider

Active providers

- Airbyte
- Alibaba
- Amazon
- Apache Beam
- Apache Cassandra
- Apache Drill
- Apache Druid
- Apache Flink
- Apache HDFS
- Apache Hive
- Apache Iceberg
- Apache Impala
- Apache Kafka
- Apache Kylin
- Dingding
- Discord
- Docker
- Edge3
- Elasticsearch
- Exasol
- FAB (Flask-AppBuilder)
- Facebook
- File Transfer Protocol (FTP)
- Git
- GitHub
- Google
- gRPC
- Hashicorp
- OpenAI
- OpenFaaS
- OpenLineage
- Open Search
- Opsgenie
- Oracle
- Pagerduty
- Papermill
- PgVector
- Pinecone
- PostgreSQL
- Presto
- Qdrant
- Redis

https://airflow.apache.org/docs/

Operatorを提供するパッケージ（Provider）が多数利用可能

公式ページに掲載されているものだけで100種類

独自のOperator・Providerを開発することもできる

Sensor: 待機処理

```
# チケットクローズを待機するTask
wait_ticket_closed = JiraTicketSensor(
  task_id="wait_issue_closed",
  ticket_id="PROJ-1234",
  field="status",
  expected_value="Closed"
)
```

条件を満たすまでワークフローの実行を一時停止する特殊なOperator

「チケットのステータスが変化すれば処理続行」

「人が承認するまで待機」

現地作業や承認を含む数日間に渡るようなワークフローにも対応

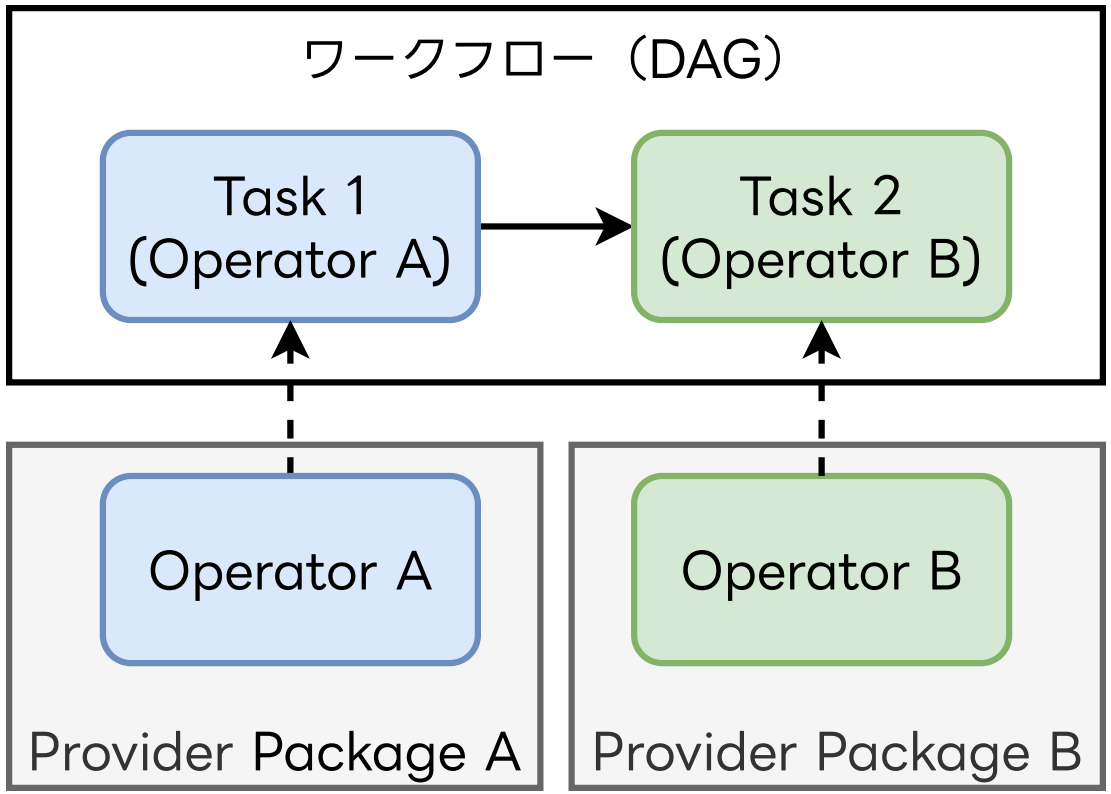
ネットワーク監視の自動化はどこまでできるのか?

-Apache Airflowによるアラート対応基盤-

1. 背景と課題
2. Apache Airflow
    1. Apache Airflowの紹介
    2. NW運用基盤としてのApache Airflow
    3. ワークフロー内でのAI活用
3. NWアラート対応基盤 “oyakata”
4. oyakata導入の結果・今後の課題・まとめ

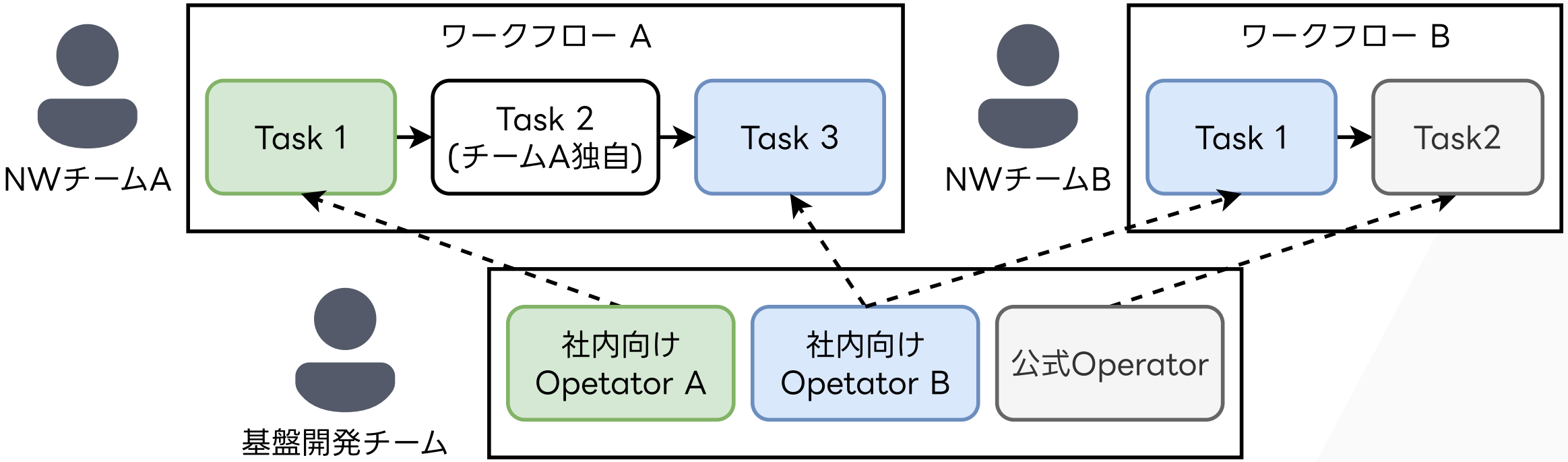
NW運用基盤としてのApache Airflow

再掲



AirflowのOperatorというコンセプトをNW運用自動化の共通基盤に適用

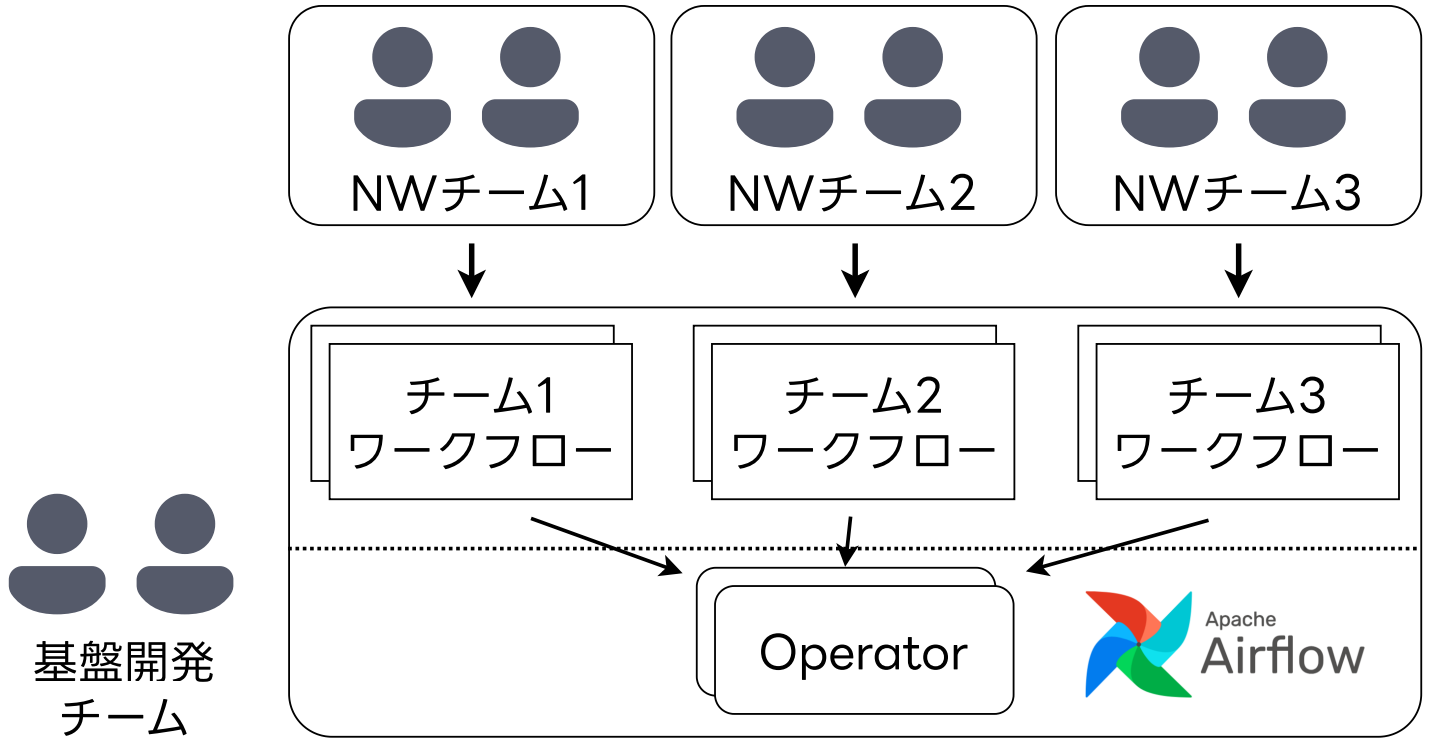
NW運用基盤としてのApache Airflow



- 基盤開発チーム：共通のOperatorを開発
- NWチーム：Operatorを利用してワークフローを実装

実装の重複を避け各NWチームは **本質的な自動化の実装に注力**

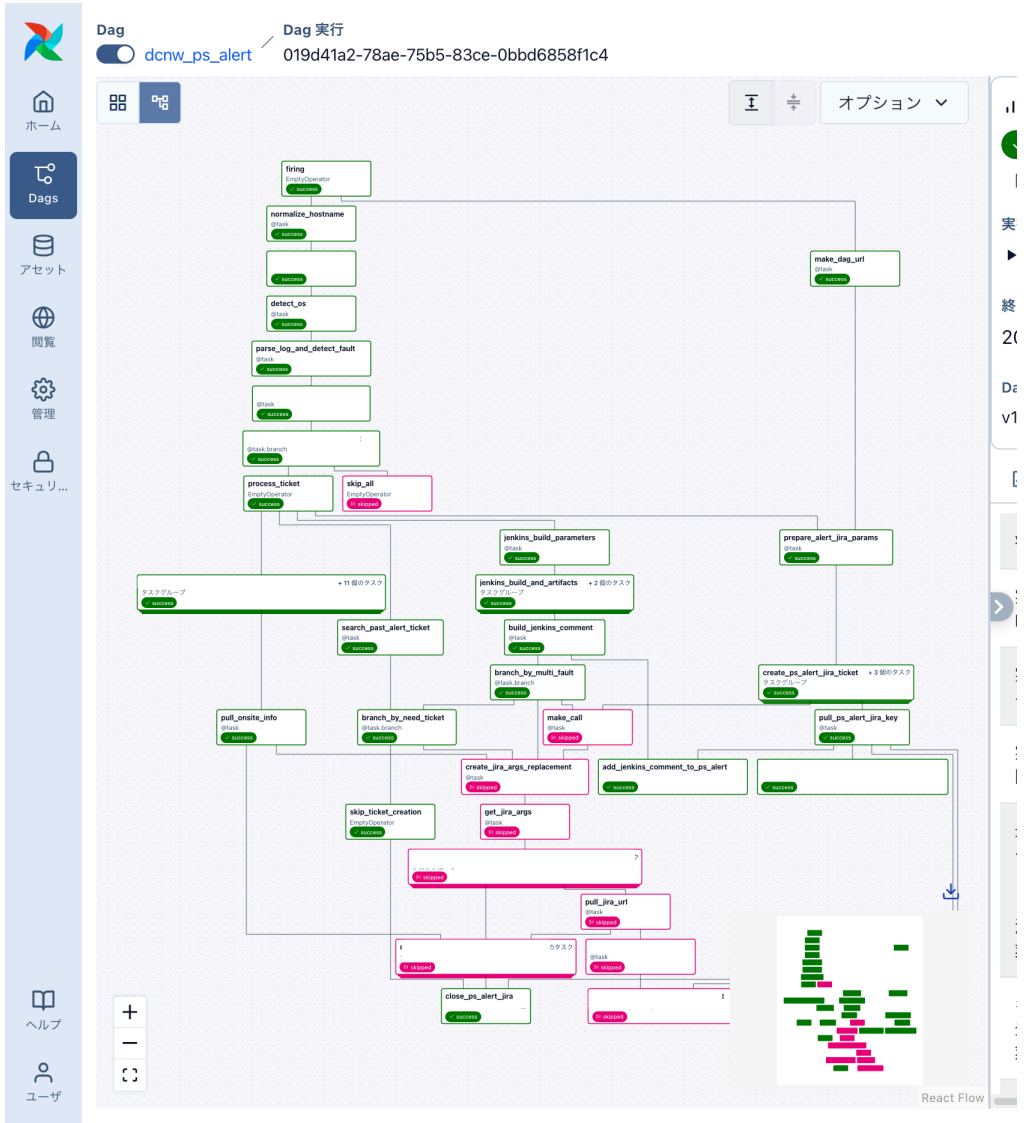
NW運用基盤としてのApache Airflow



Airflowで人手の一次対応をワークフローで置き換え

- 基盤チームはAirflowとOperatorを整備
- NWチームは運用知識の実装に注力

ワークフロー実装の例



NWチームが実装したPSUアラート対応

- アラート管理チケットを起票
- 構成管理DBから機器情報を取得
- ログや実機でPSUステータス取得
- 過去の発生状況確認
- データセンター側で対応中でないか確認
  （作業影響によるアラートは静観）
- データセンターにPSU交換依頼を起票
- 複数故障等の場合はエスカレーション

従来：人手で1件5～10分

自動化：1-2分で自動実行、工数0

ネットワーク監視の自動化はどこまでできるのか?

-Apache Airflowによるアラート対応基盤-

1. 背景と課題
2. Apache Airflow
    1. Apache Airflowの紹介
    2. NW運用基盤としてのApache Airflow
    3. ワークフロー内でのAI活用
3. NWアラート対応基盤 “oyakata”
4. oyakata導入の結果・今後の課題・まとめ

NW自動化とAI

コードによる自動化

メリット

- 外部APIのSLAに依存しない
- 確実・決定的に動作する

デメリット

- 柔軟性に限界

AI（LLM）による自動化

メリット

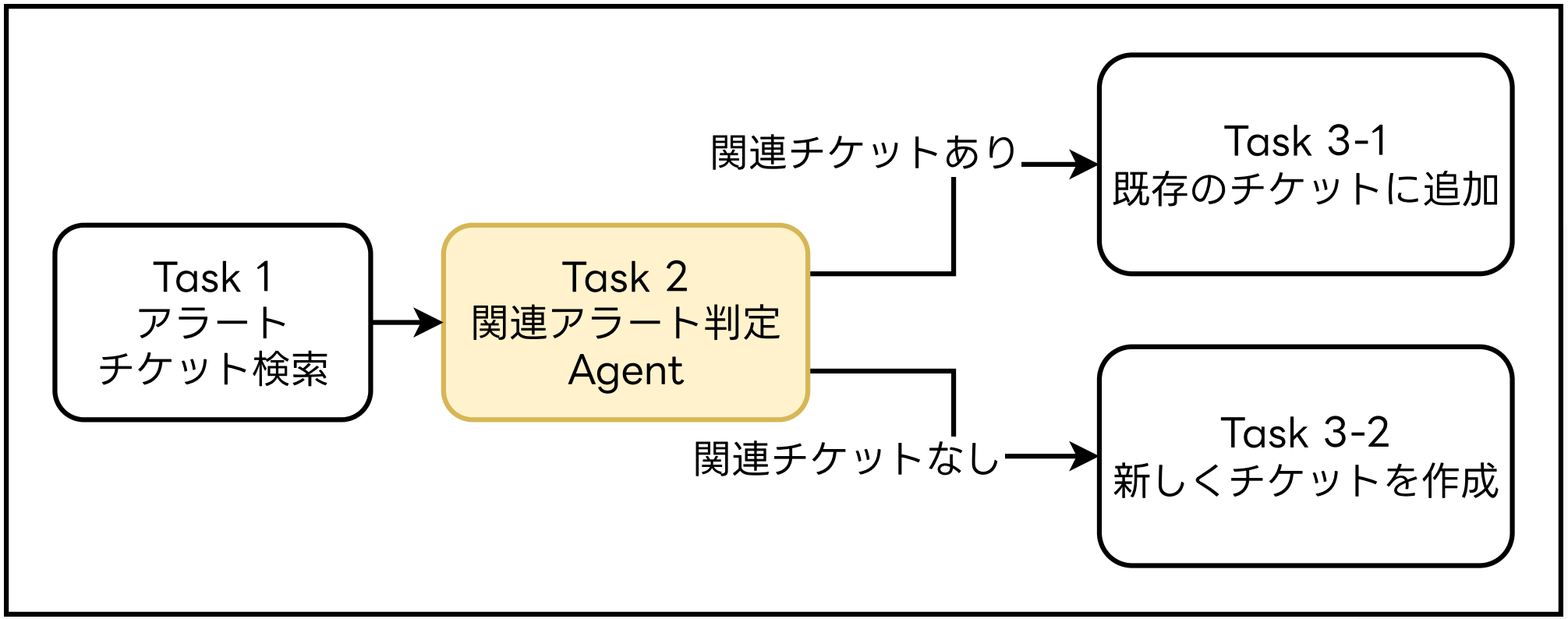
- ルールベースではカバーできない柔軟な対応
- 自然言語による指示

デメリット

- APIのSLAに依存
- 精度のばらつき

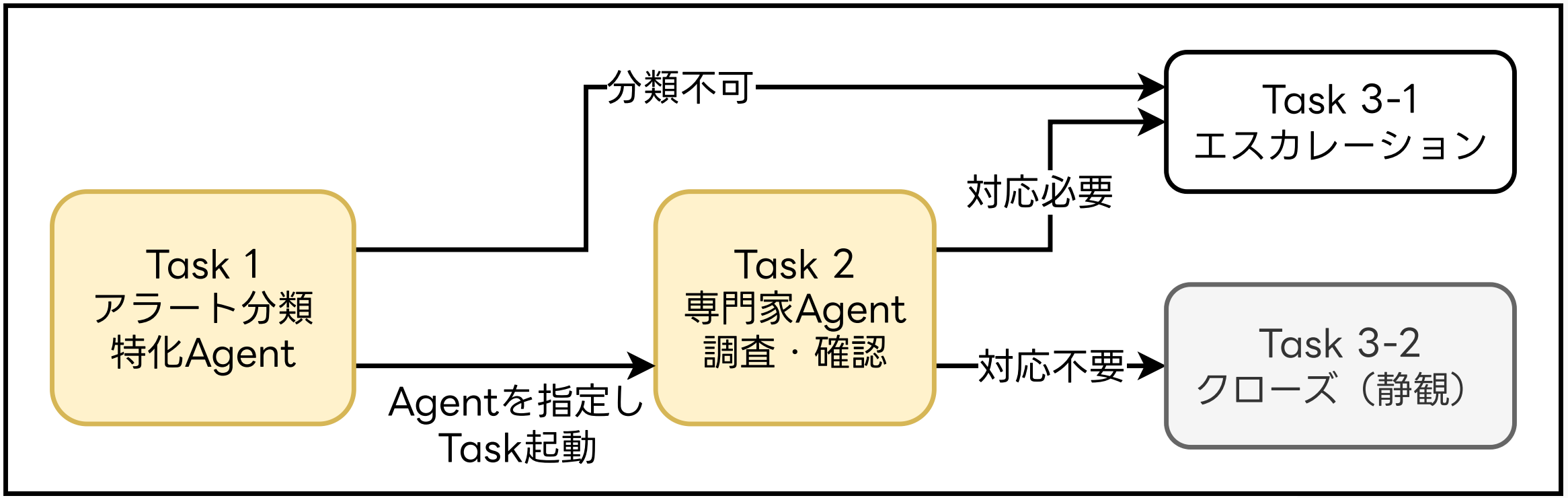
Airflowではコードベースの自動化と
AIによる処理をTask単位で組み合わせられる

LLM Agentを組み込んだワークフロー 1



従来人手で行っていた関連アラートの集約をLLMで実施

LLM Agentを組み込んだワークフロー 2



アラート分類に特化したAgent・各アラートの専門家Agentの二段処理

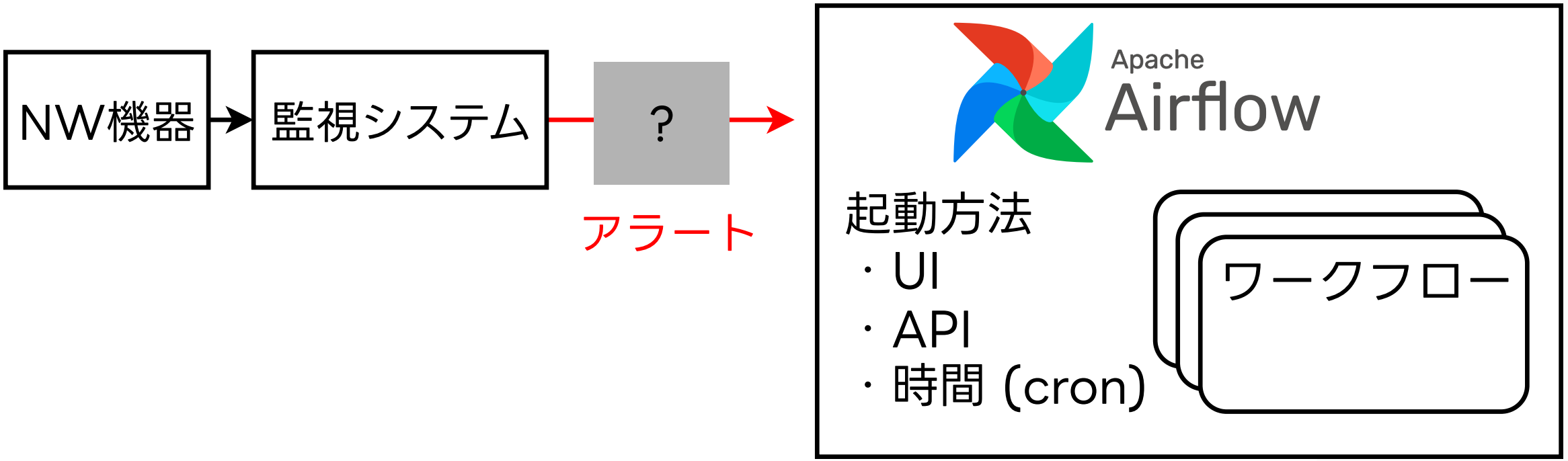
Agentの更新のみで対応できる範囲を増やせる

Apache Airflowまとめ

アラート対応の基盤としてApache Airflowを選定

- NWチーム：運用手順をワークフローとして直接実装
  基盤開発チーム：共通部品（Operator）と環境を提供
- ワークフローと実行履歴の可視化
- フロー内で待機や承認も可能
- コードベースの処理とAI処理の組み合わせも可能

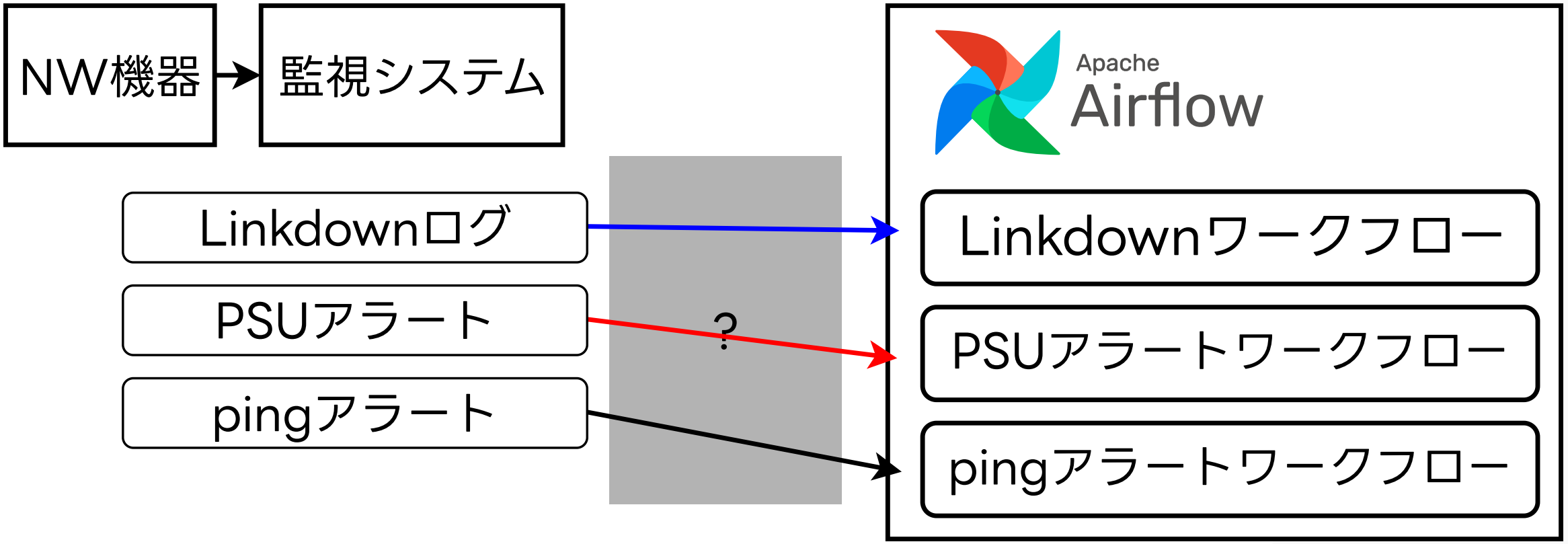
Airflow単体では実現できないこと 1



Airflow標準のワークフロー起動は3種類のみ

NWアラートを受けてワークフローを起動する仕組みが必要

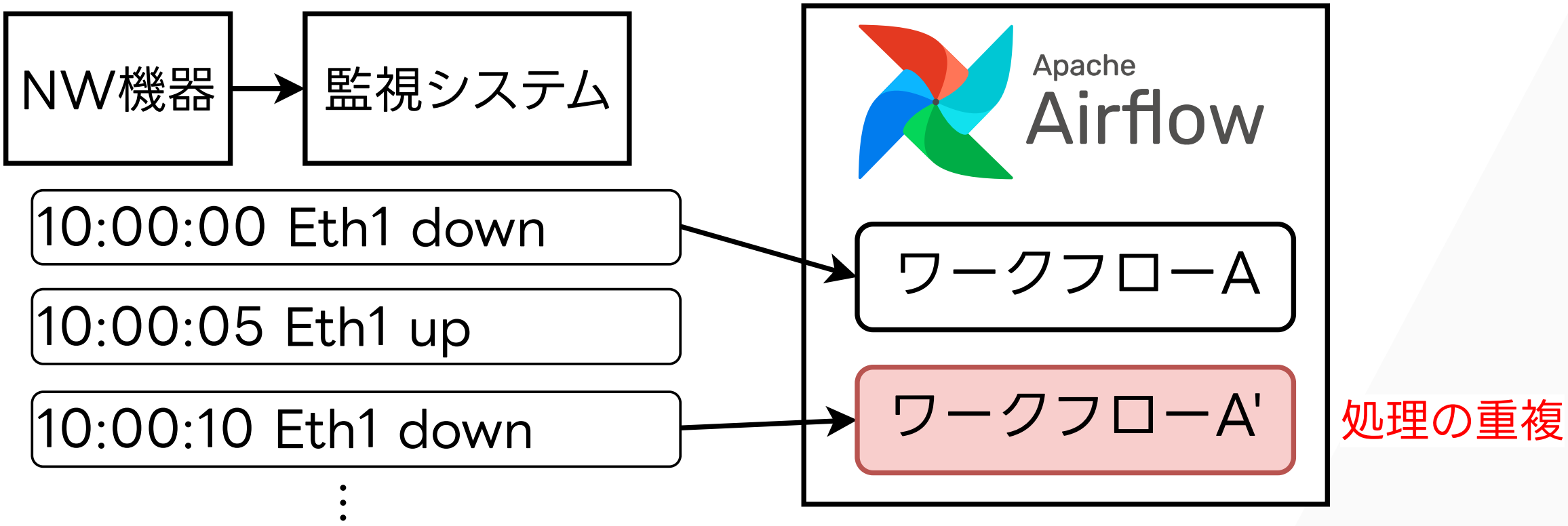
Airflow単体では実現できないこと 2



各アラートから どのワークフローを どんなパラメータで 起動するか

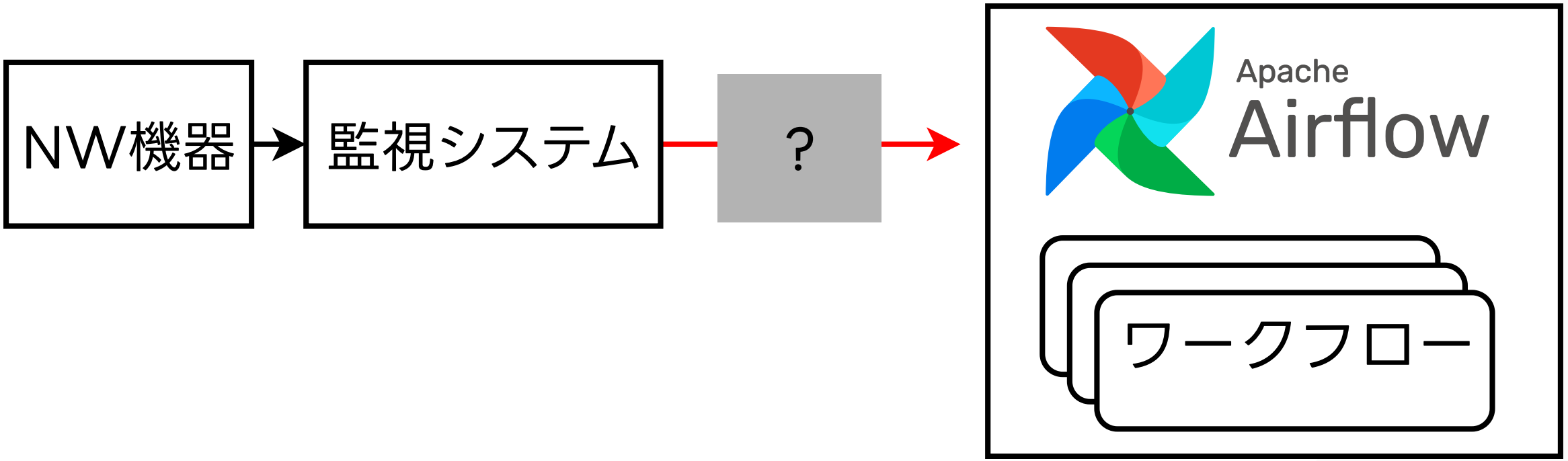
これもNWチーム主体で管理したい

Airflow単体では実現できないこと 3



単一の事象で複数発生するアラートで重複して自動処理を起動したくない

Airflow単体では実現できないこと



必要な機能

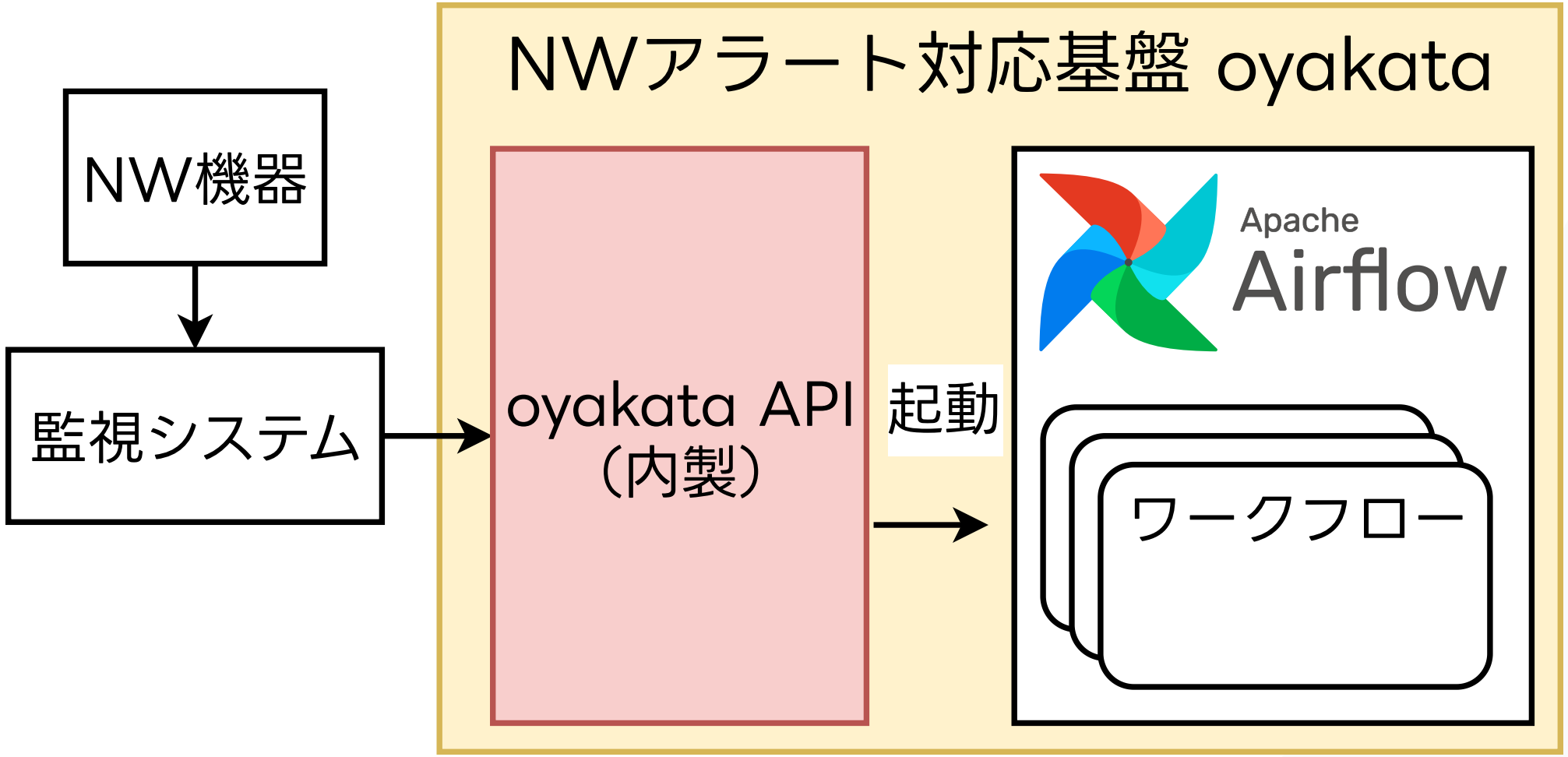
1. 監視システムからアラートを受信しワークフローを起動
2. アラートの種類とワークフローを紐付け
3. アラートの重複排除

ネットワーク監視の自動化はどこまでできるのか?

-Apache Airflowによるアラート対応基盤-

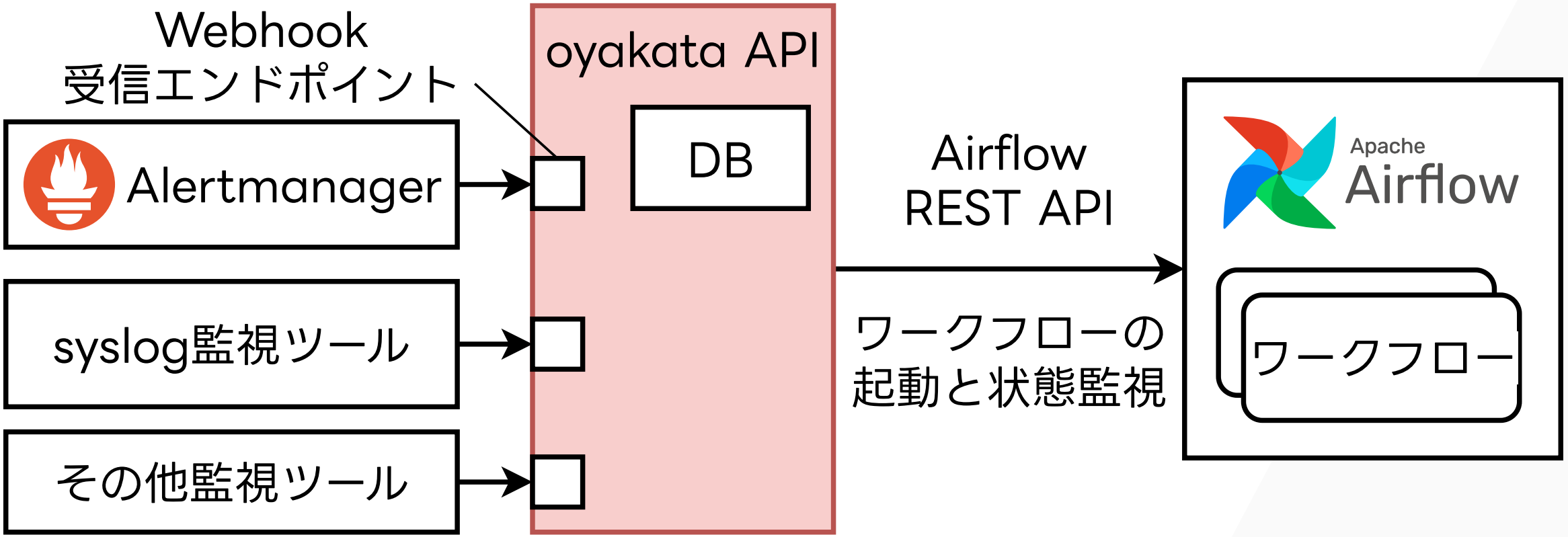
1. 背景と課題
2. Apache Airflow
    1. Apache Airflowの紹介
    2. NW運用基盤としてのApache Airflow
    3. ワークフロー内でのAI活用
3. NWアラート対応基盤 “oyakata”
4. oyakata導入の結果・今後の課題・まとめ

Airflow + API = oyakata



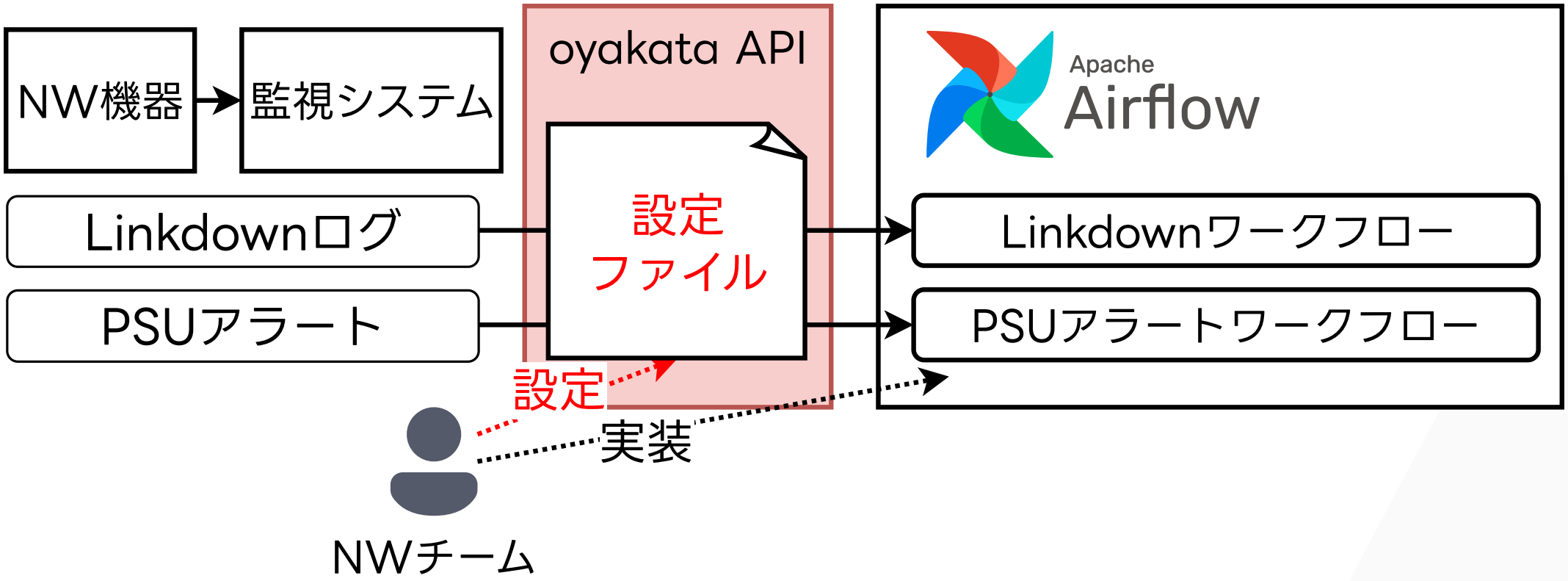
名前の由来：様々な自動化・業務を束ねる「親方」のような基盤を目指す

oyakata APIの機能
1. アラート受信とワークフロー起動



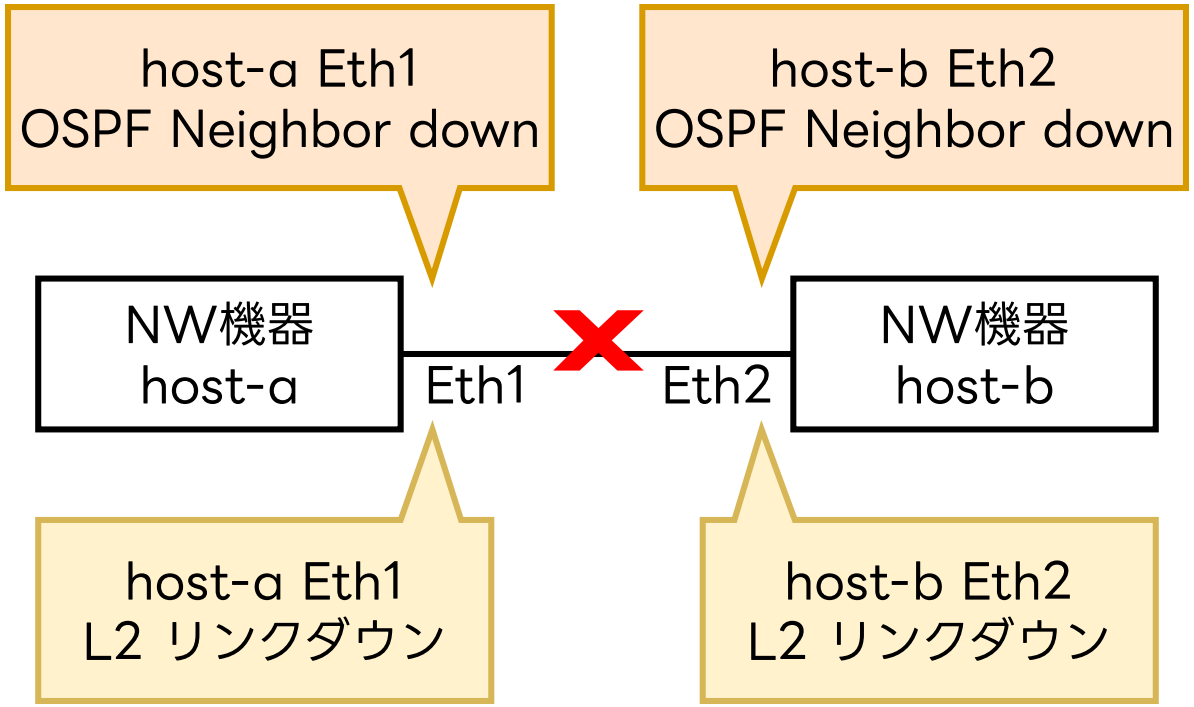
各アラートシステムのWebhookの受信エンドポイントを実装

oyakata APIの機能
2. アラートとワークフローの紐つけ



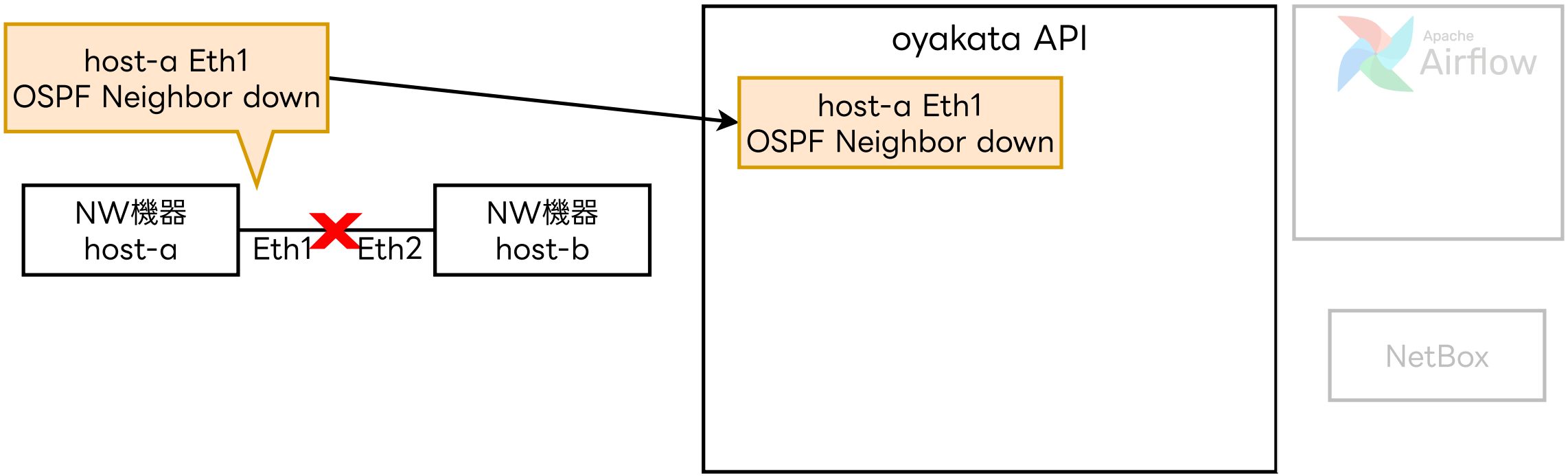
設定ファイルでどのアラートからどのワークフローを起動するか指定できる

oyakata APIの機能
3. アラートの重複排除

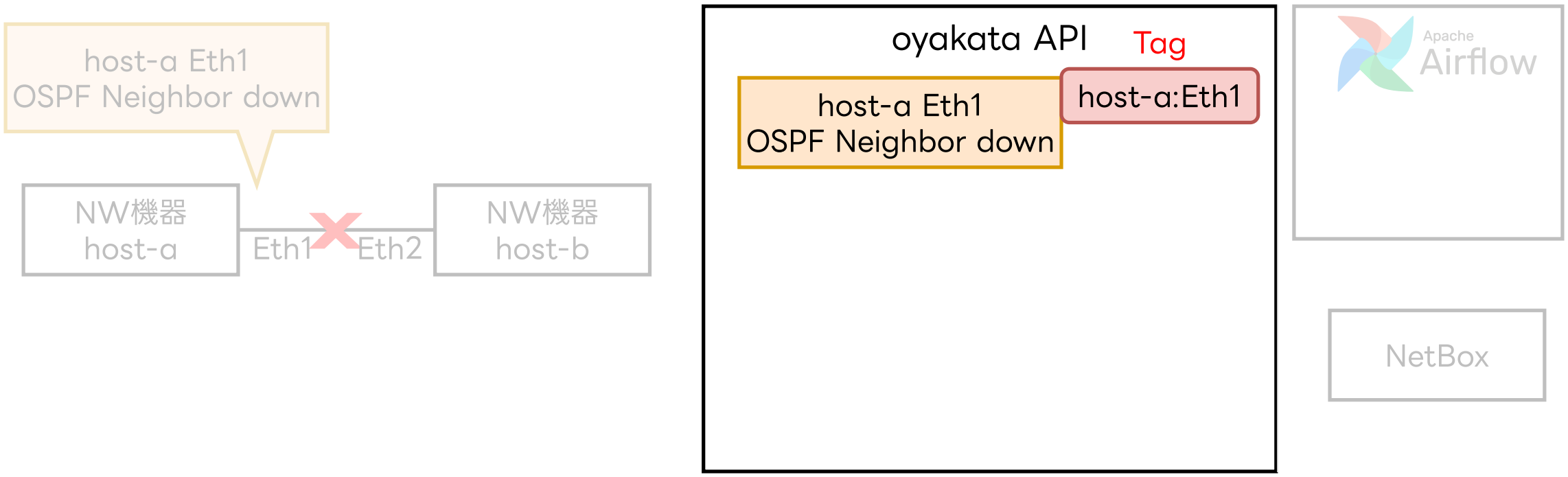


- 同じアラートが短期間に複数出る
- 単一事象でL2/L3や対向側などの複数アラートが出る

アラートの重複排除 1/3



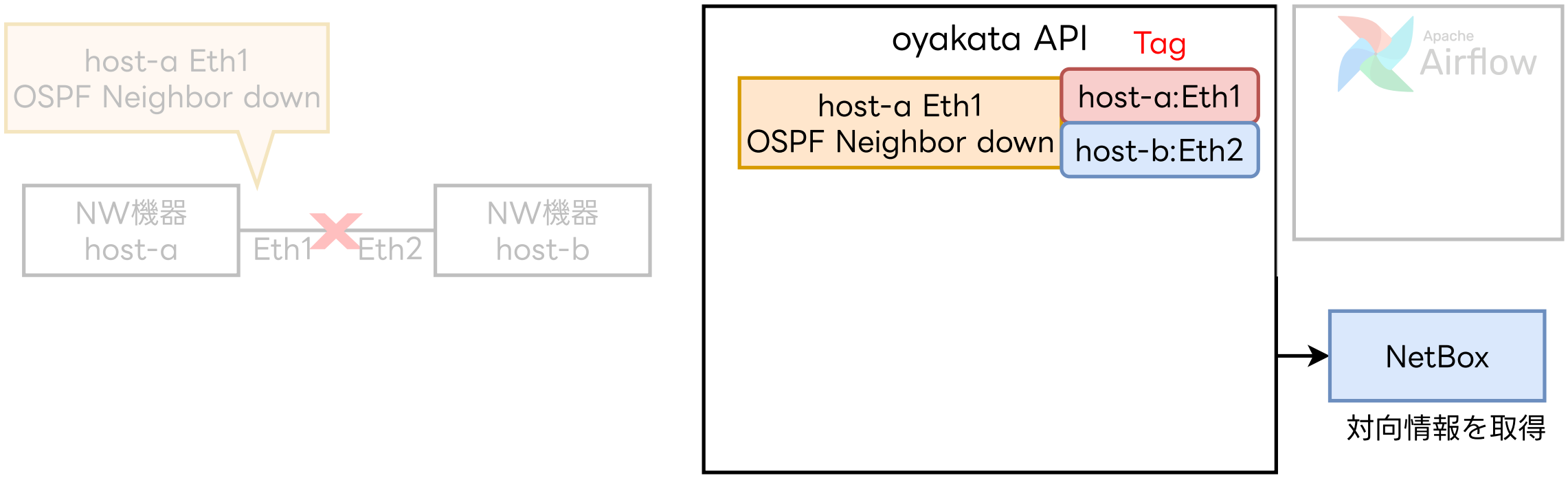
アラートの重複排除 1/3



oyakata APIで設定ファイルに基づき「Tag」を付与

例：`{hostname}:{interface}` Tagを付与

アラートの重複排除 1/3

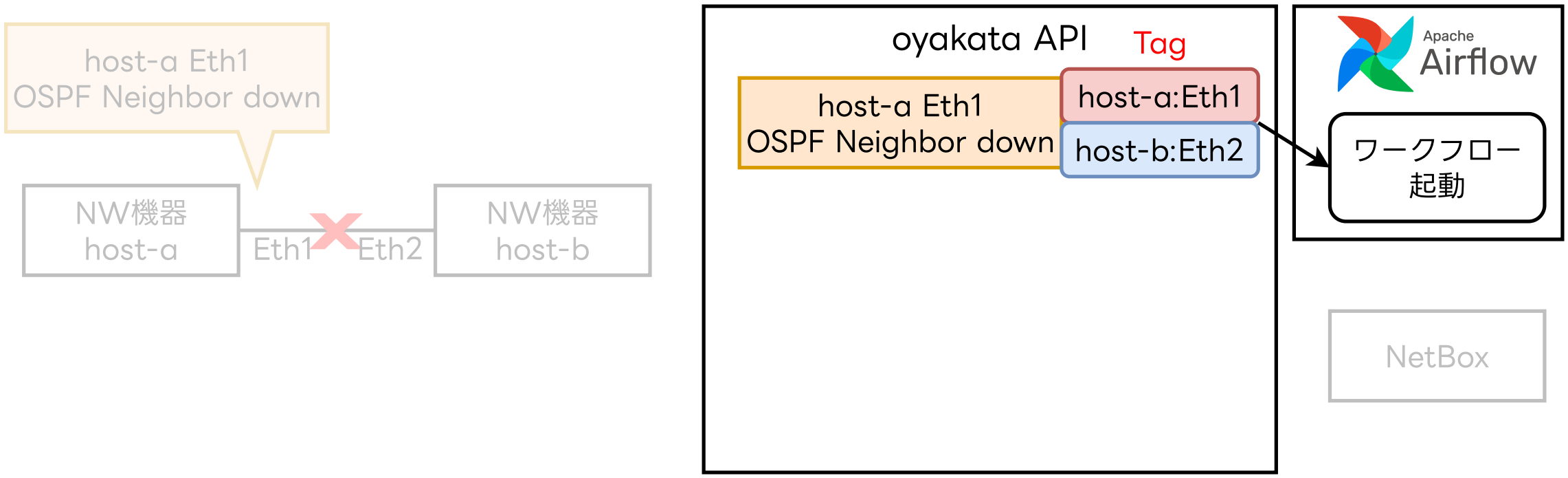


oyakata APIで設定ファイルに基づき「Tag」を付与

例：`{hostname}:{interface}` Tagを付与

NetBoxに登録されている配線情報を参照し対向側のタグも付与

アラートの重複排除 1/3

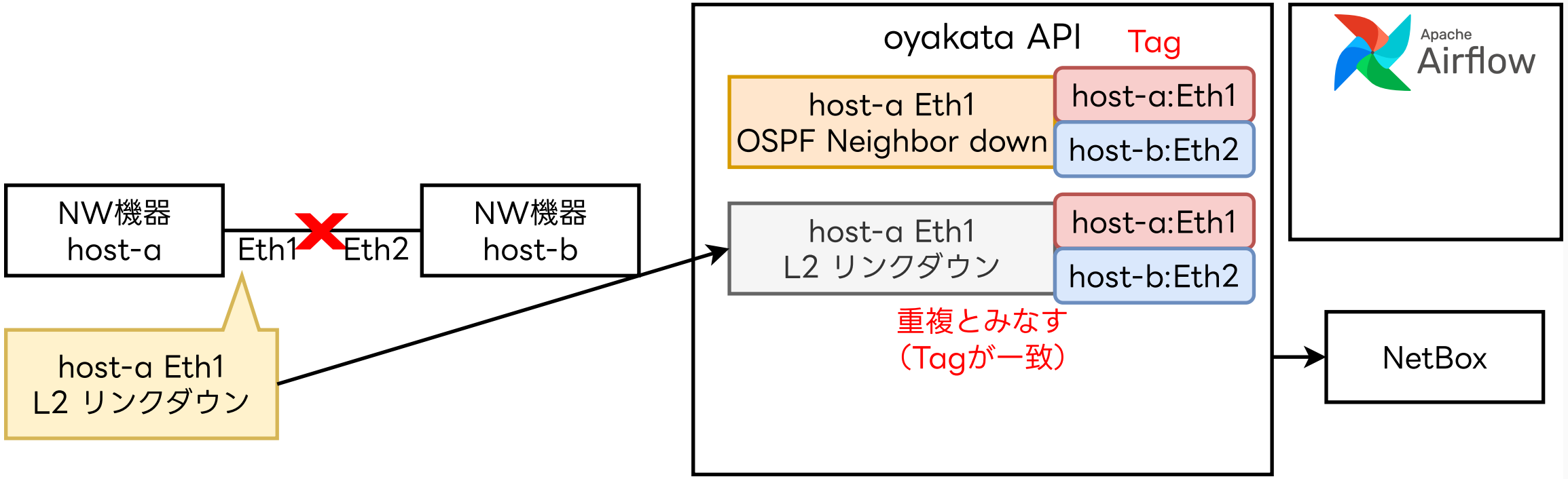


oyakata APIで設定ファイルに基づき「Tag」を付与

例：`{hostname}:{interface}` Tagを付与

NetBoxに登録されている配線情報を参照し対向側のタグも付与

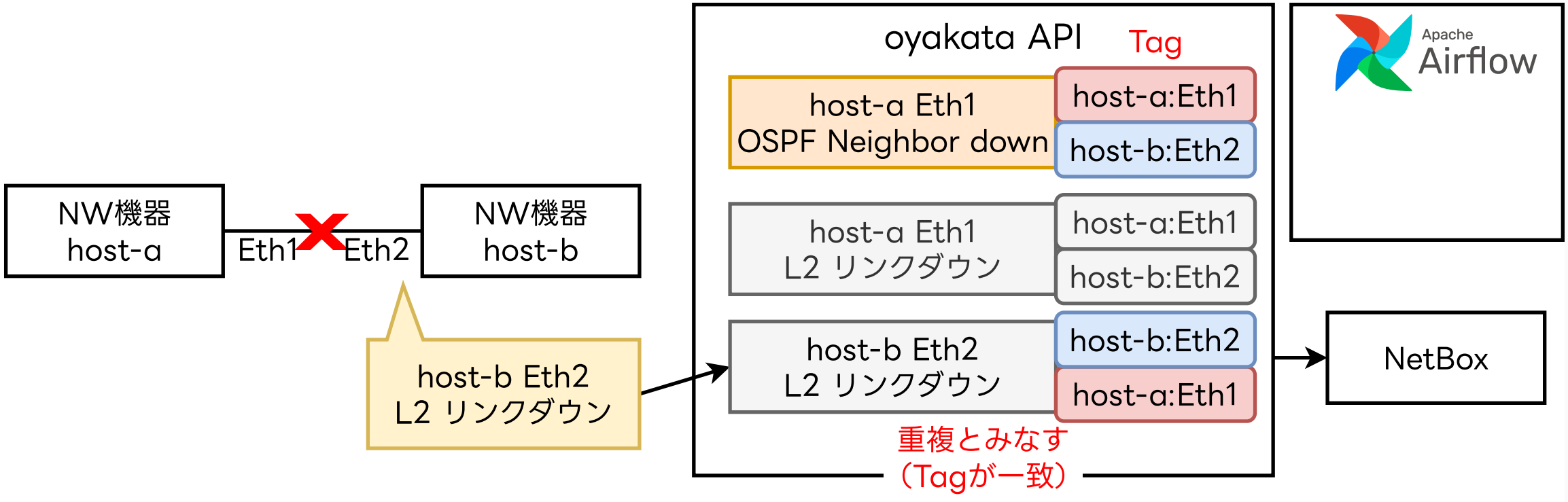
アラートの重複排除 2/3



同一箇所の別アラートにも同様のTagが付与

→ Tagが一致したら重複：ワークフロー起動をスキップ

アラートの重複排除 3/3



対向からのアラートも、同様にタグが付与（順不同）され処理がスキップ

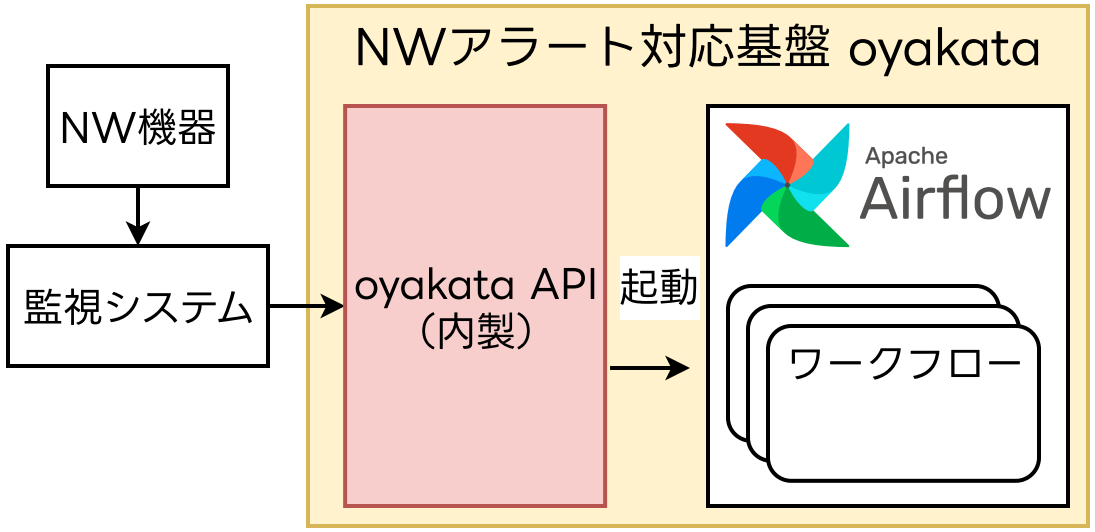
oyakata まとめ

Apache Airflowを中心としたNWアラート対応基盤oyakataを構築

- 複数のNWアラートシステムとの連携
- アラートとワークフローの紐つけ
- アラートの重複排除

を実現するために **oyakata API** を内製

再掲



ネットワーク監視の自動化はどこまでできるのか?

-Apache Airflowによるアラート対応基盤-

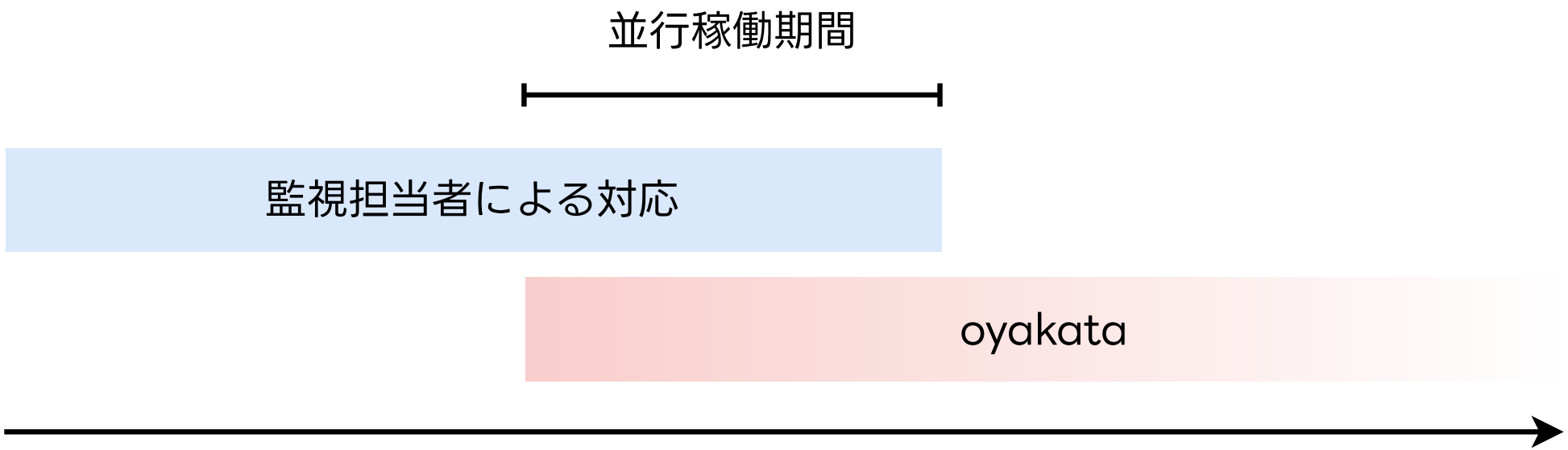
1. 背景と課題
2. Apache Airflow
    1. Apache Airflowの紹介
    2. NW運用基盤としてのApache Airflow
    3. ワークフロー内でのAI活用
3. NWアラート対応基盤 “oyakata”
4. oyakata導入の結果・今後の課題・まとめ

実運用へ移すにあたっての工夫

oyakata勉強会の実施	Agent Skill提供	共通ワークフローの提供
NWチーム向けに ・oyakata基盤の解説 ・ワークフロー作成チュートリアル	oyakata APIの設定を作成するAgent Skillを提供	各NWチームが共通で使えるワークフローは基盤開発チームで提供 NWチームの工数をさらに削減

実運用へ移すにあたっての工夫

- 新たな基盤にいきなり切り替えるのはリスク
- 監視担当者と連携し、並行稼働期間を設けた
- この期間にワークフローの挙動確認や改善を実施



課題（再掲）

「NWアラート対応を人手をかけずに全て自動実行できる」を実現したい

- 課題1: 人手によるボトルネック
  → アラートの一次対応を、人手によるものから
  基盤上で自動実行し必要な時だけ人に渡す運用へ切り替える
- 課題2: 自動化の乱立
  → ネットワーク**各チームで使える共通基盤**を構築
  運用者が本質的な自動化の実装に注力できるようにする

導入結果

oyakata は課題を解決した！

- 課題1: 人手によるボトルネック

  → **複数のNWチームの一次対応をoyakataに載せ自動化**

  一次対応については90%以上を自動対応に移行

- 課題2: 自動化の乱立

  → **NWチームが運用知識の実装に注力できる共通基盤oyakataを整備**

今後の課題（野望）

~~NWアラート対応~~を人手をかけずに全て自動実行できる

↓

NW運用全て を人手をかけずに自動実行できる

oyakata：アラートだけでなく**NW運用全般の自動化・効率化**ができる設計

→ NW運用に人手がかからない状態を目指す

【宣伝】

弊社ブースでもoyakataを中心としたNW運用全体の自動化について紹介！

今後の課題

新たなステージでの「成熟」と「継承」

- **成熟**
  - NW運用がワークフロー・LLMによって自動化されている
- **継承**
  - 自動化された運用を新メンバーが理解しその運用を改善していく
  - 自動化していても例外発生時に対応できる運用知見の継承

→ **自動化の成熟が進むことで、NW運用知見の継承が難しくなる** のでは?

oyakataはワークフロー・実行状態の可視化はできるが
可視化すれば知見を継承できるわけでもない

まとめ

課題

- 人手を前提とした運用のボトルネック
- 複数チームでの自動化の乱立

実施した取り組み

Apache Airflowを中心にNWアラート対応基盤 “oyakata” を構築

- 人手前提の対応 → ワークフローとして自動実行
- NWチームが運用知識の実装に注力できる

複数NWチームで本番導入し 一次対応の90%以上を自動対応へ移行

今後の課題

NW運用全体の自動化推進・自動化が進んだ環境でのNW運用知見継承

議論したいこと

- NWアラート対応はどれくらい自動化されていますか
  自動化するハードルはどこにありますか
- NW運用にワークフローランナー（Airflow）を利用することの
  メリット・デメリット
- 1件の障害で重複してアラートが上がることにどう対処していますか
- 共通の運用基盤を使ってもらうにはどのようなハードルがありますか
- NW運用の自動化が進むことで、
  NW運用の知識・知見が継承されにくくなるのをどう防ぎますか

注記

- Apache Airflow、Airflow、およびApache Airflowのロゴは、米国およびその他の国におけるThe Apache Software Foundationの登録商標または商標です。
- PrometheusおよびPrometheusロゴは、米国およびその他の国におけるThe Linux Foundationの登録商標または商標です。AlertmanagerはPrometheus projectのコンポーネントです。