2nd half

運用を見据えた障害監視

～障害監視フレームワーク～

波田野 裕一

(日本UNIXユーザ会)

2011-02-26 Shizuoka ITPro 6th techtalk

本セッションの概要

- 開発方法論やプロジェクト管理技法は巷にあふれていますが、本当に肝心なその**成果物を日々運用するための方法論はほぼ皆無**というのが現状です。
- 今回は、障害監視のためのフレームワークを軸に**運用への熱い想い**をさらりと語ります。

障害監視とツール

- いろいろな監視・運用ツールがあった
  - Net-Saint, MRTG, OpenView, アプライアンスの管理ツール, etc
- それぞれの**強みやフォーカスがばらばら**
  - 死活監視、リソース監視、ハードウェア、GUI/レポート機能
- 特に**ツール連携が難しい、面倒**。
  - 「統合監視」は、画面を並べただけ、みたいなことにも。

障害監視 自作ツール

補完するために自作ツールを作ってみる、のは定番

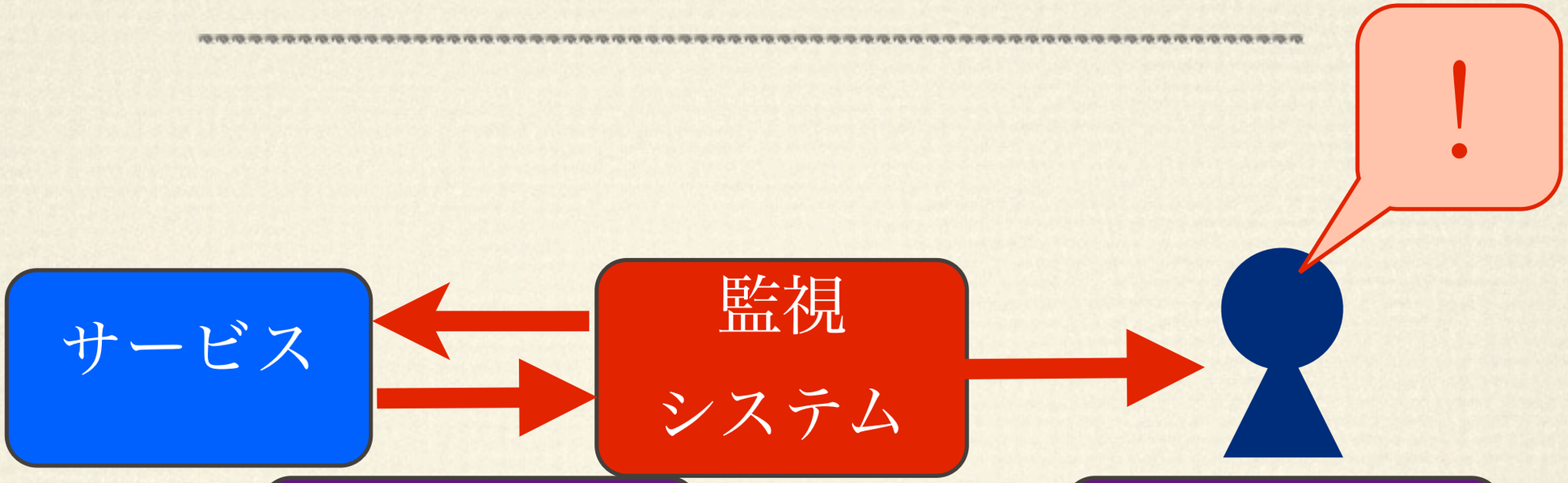
- だんだん機能や役割があちこちで重複
- じゃ、モジュール化して使いまわしをしてみよう。
- なんか、モジュール間連携が上手く動きません(泣
- 更に、仕様変更や新規案件に意外と上手く対応できません(泣

はたっと気付く

- そもそも、障害監視に**必要な機能**ってなんだっけ?
- そもそも、障害監視って**どんなプロセス**があるんだっけ?

障害監視の常識(?)

障害監視の全体像



たくさん
あがってくる

受信側の都合など知らない

何言っているか
わからない

プロダクトが勝手に叫ぶ

どんどん
多彩になる

受信側の都合など知らない

何やっていいか
わからない

初動手順がわからない!

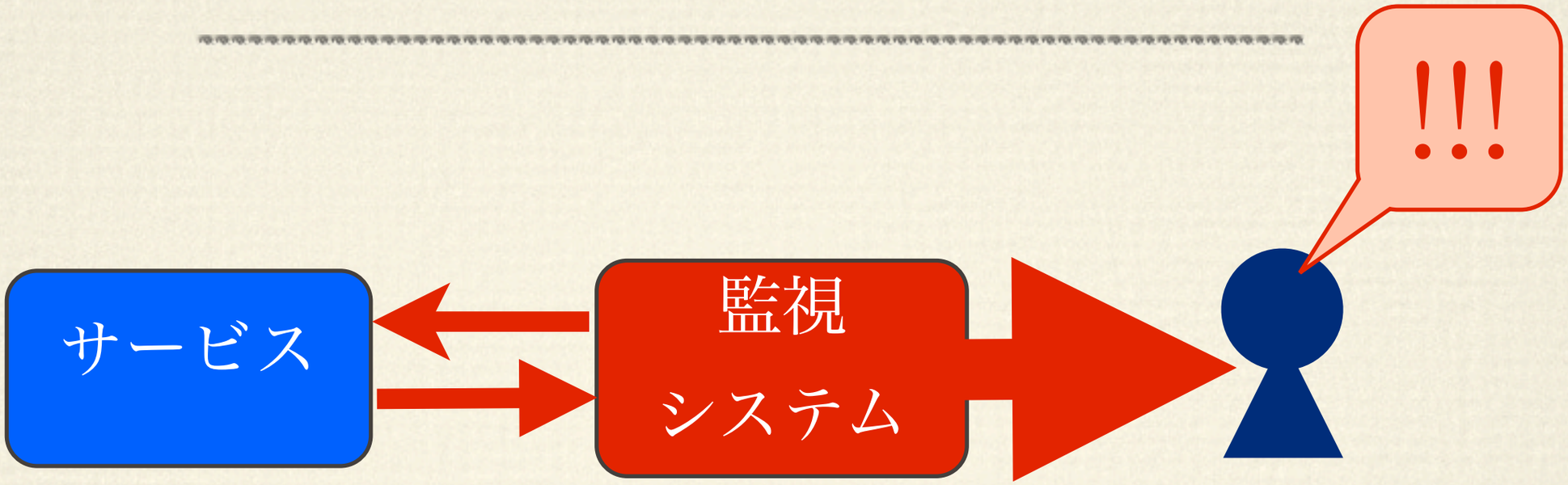
知らぬ間に
廃止されている

受信側の保守負荷など知らない

何やったか
わからない

前回どう復旧させた?

障害監視の全体像



水道の蛇口 vs. **水桶**

蛇口がどんどん開かれても、桶のサイズは簡単には変わらない

水に砂利が入っていたら、取り除くのは桶の仕事

でも、桶の本当の仕事は砂利除去や水あふれ対応じゃない、水の活用

障害監視の全体像

- そもそも、障害監視に**必要な機能**ってなんだっけ?
- そもそも、障害監視って**どんなプロセス**があるんだっけ?

障害監視をオブジェクト指向で考える

試行錯誤してみました。

オブジェクト指向分析/設計(OOA/OOD)って

のがあるらしい

- アクター、ユースケース、MVCフレームワーク。。
- **重複の排除**、部品の**独立性向上**、依存関係の非循環。。

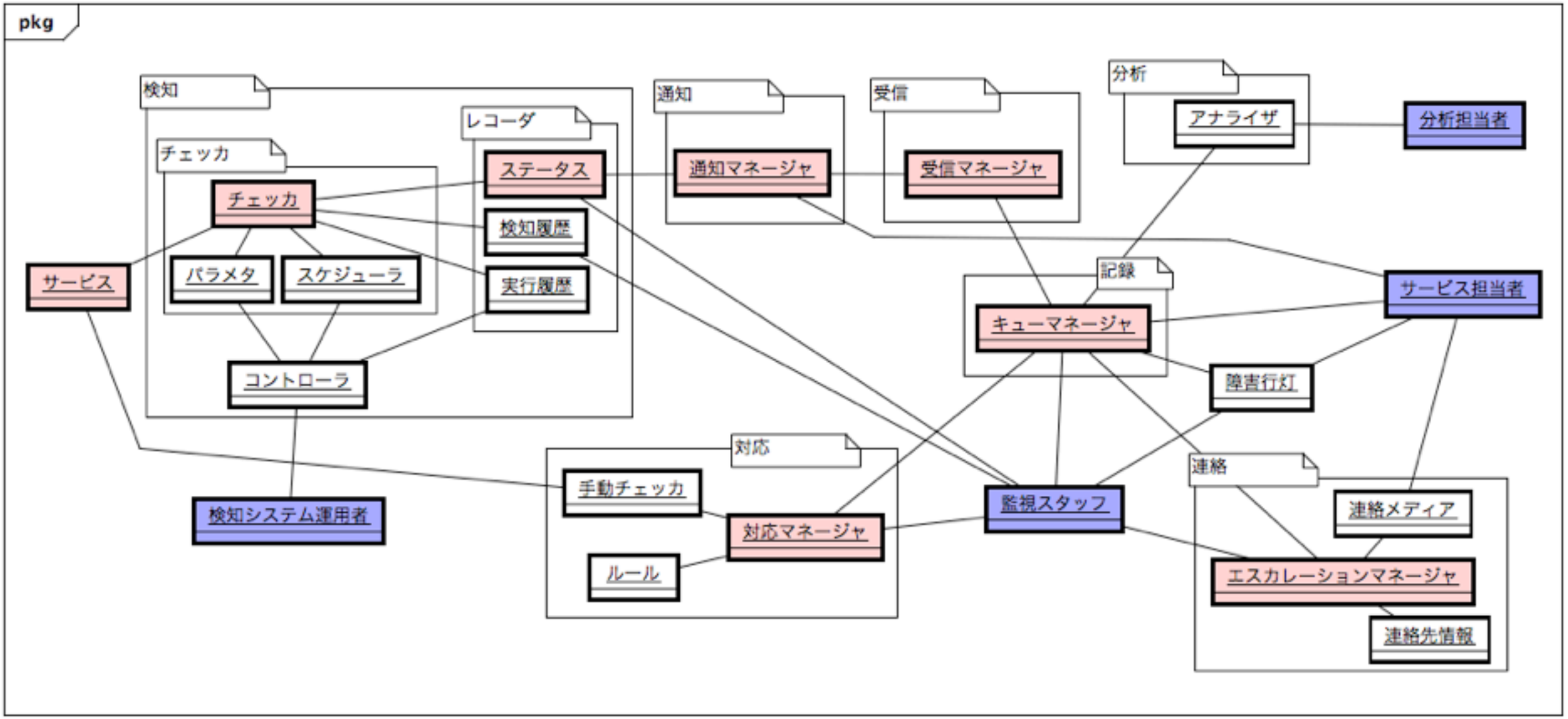
ぶっちゃけオブジェクト指向

- **ヒト/モノ**(オブジェクト)が、
- 相互に**メッセージ**で会話する、
- **一連の相互作用**を把握/表現するもの。

※個人的な超訳です:-)

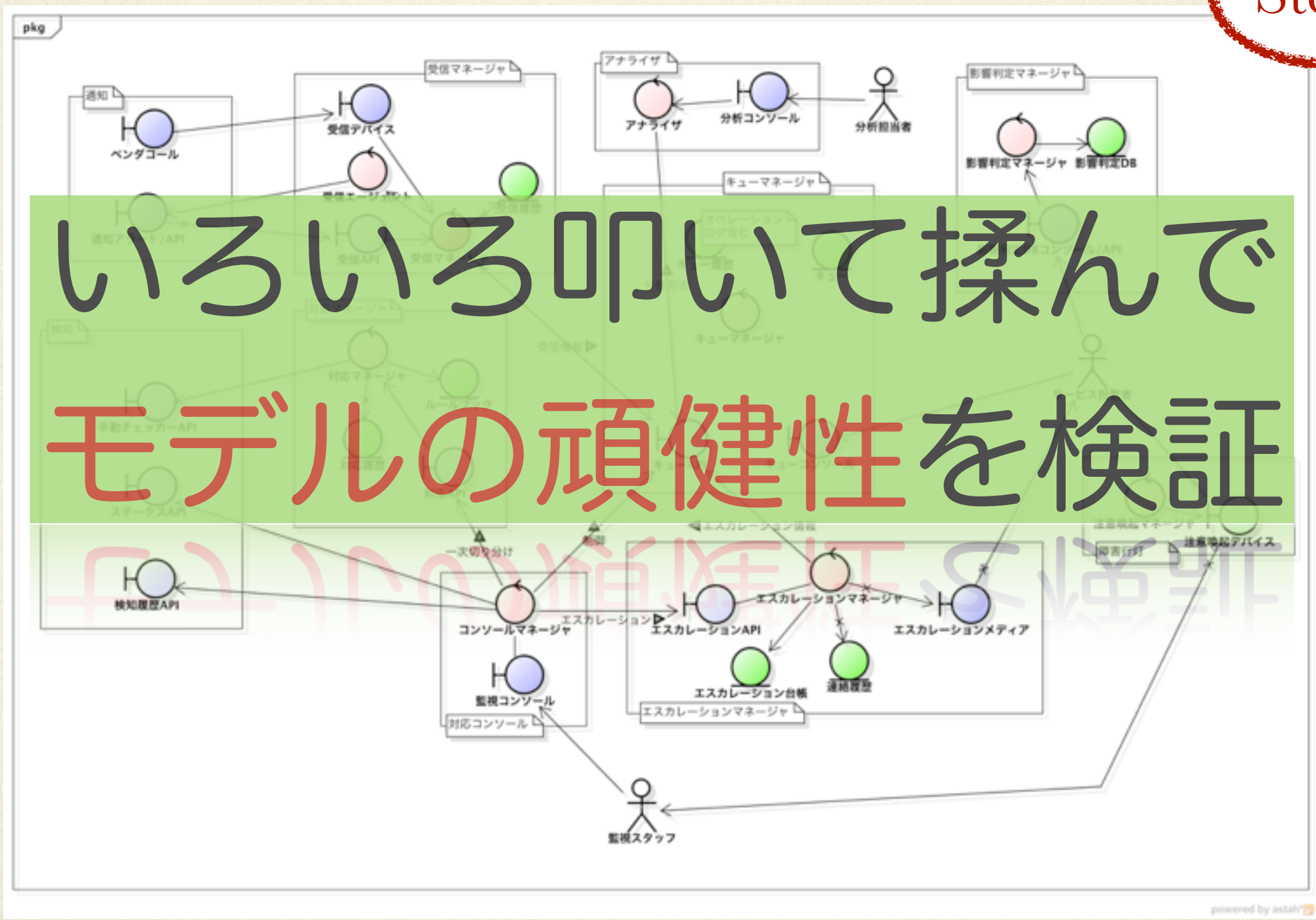
障害監視 (オブジェクト図)

Step1



ロバストネス分析

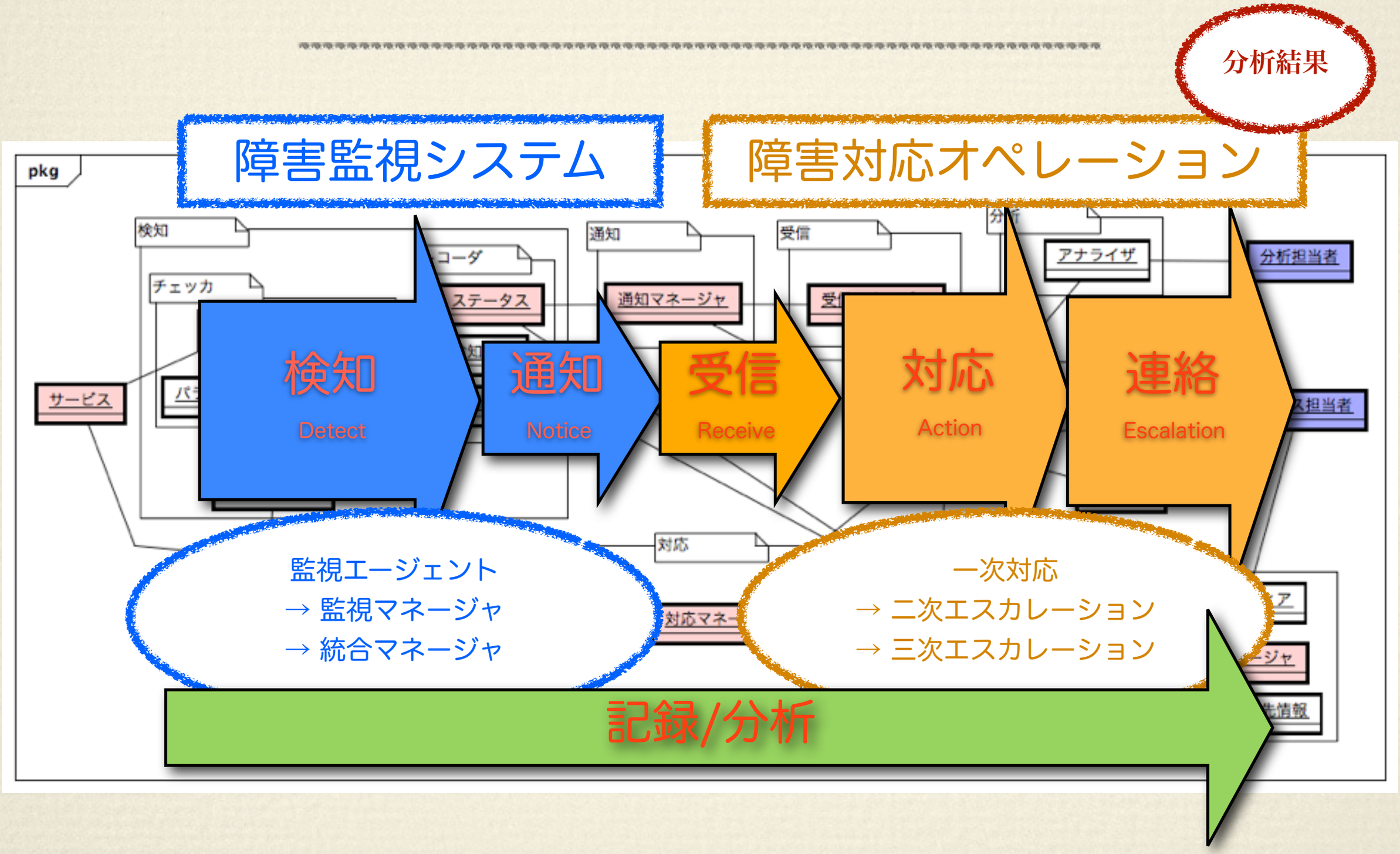
Step2



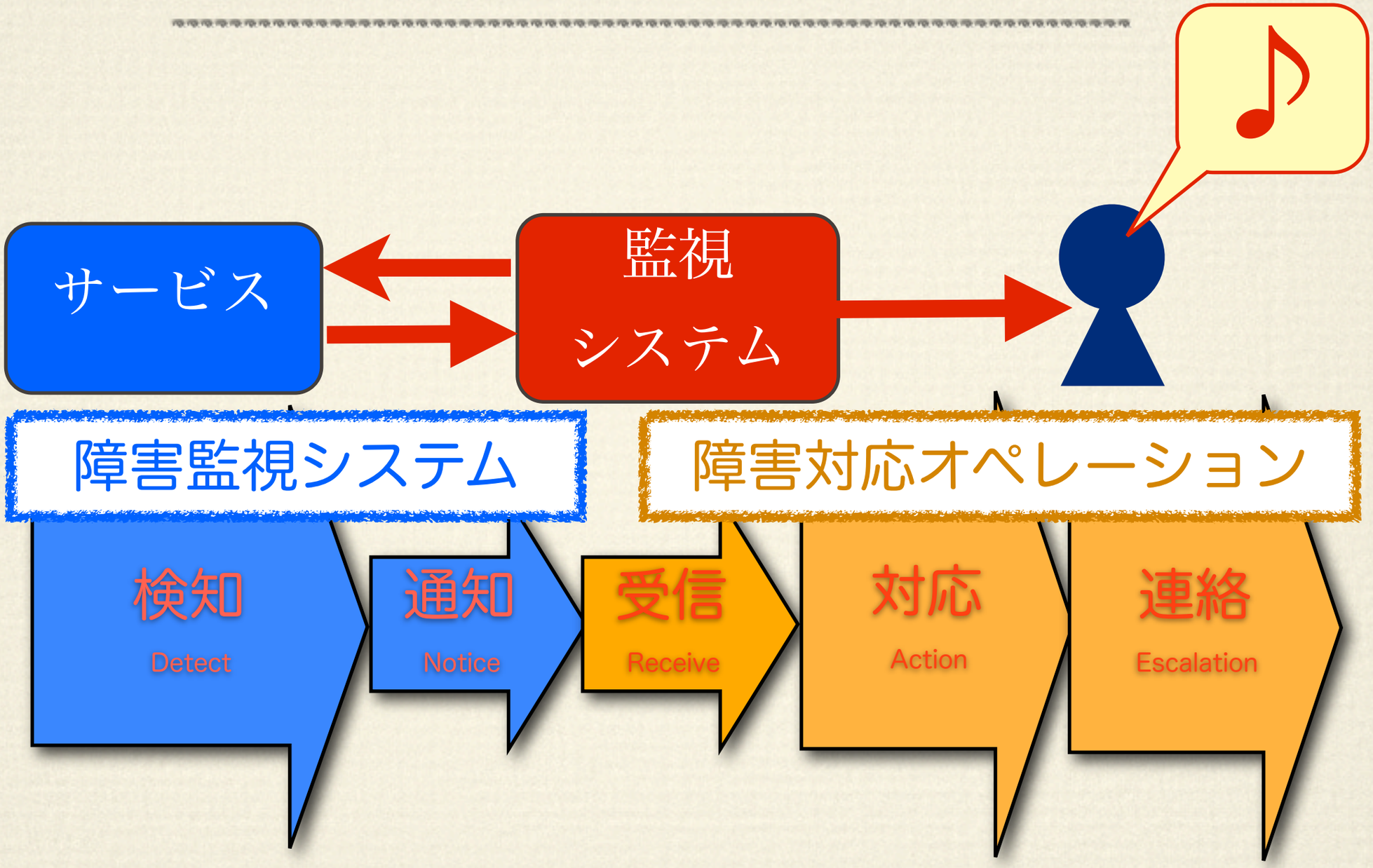
いろいろ叩いて揉んで

モデルの頑健性を検証

障害監視の6分野

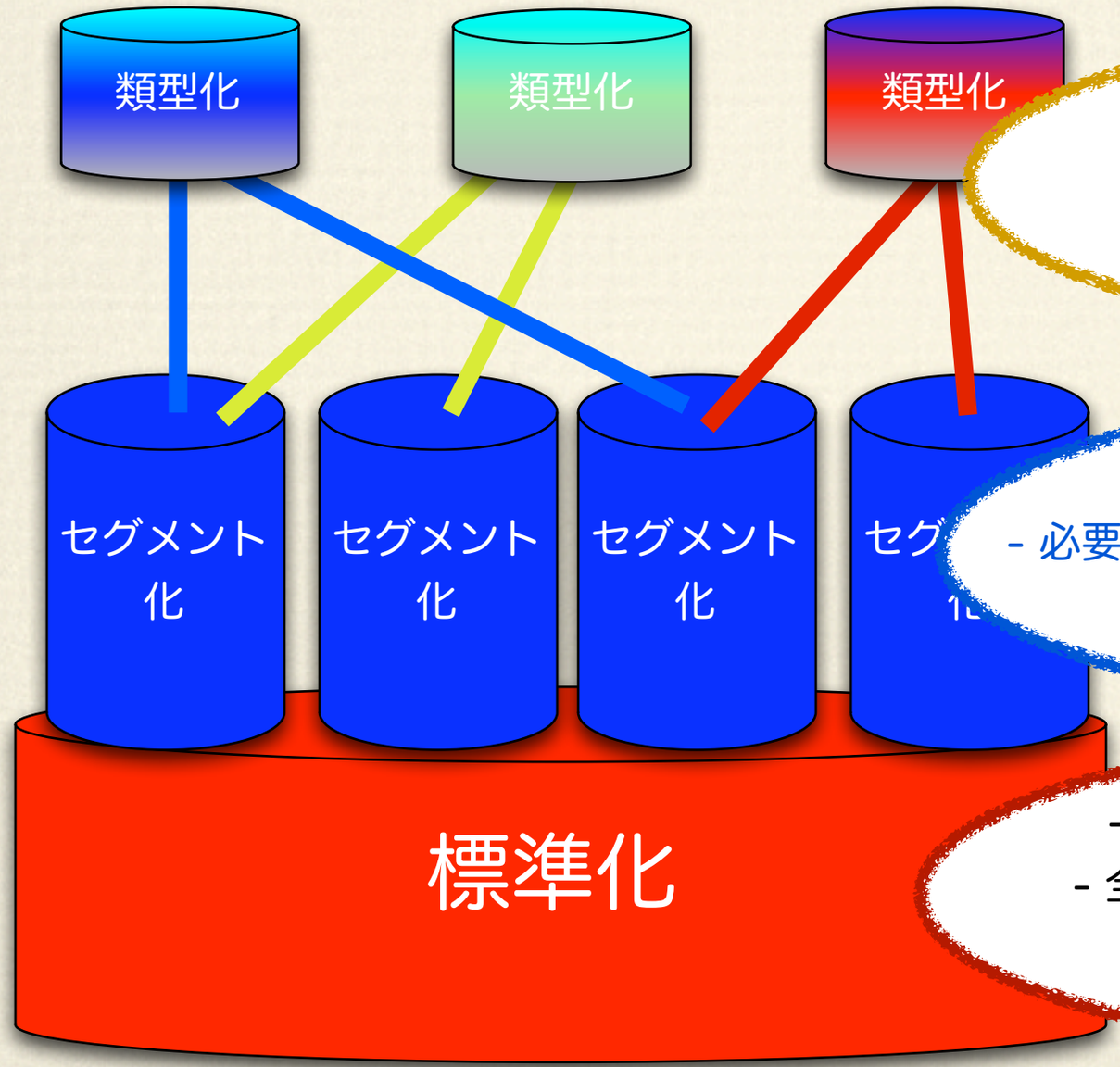

分析結果
障害監視システム
障害対応オペレーション
pkg
検知
チェッカ
コーダ
ステータス
通知
通知マネージャ
受信
アナライザ
分析担当者
サービス
対応
対応マネー
ア
ージャ
先情報
検知
Detect
通知
Notice
受信
Receive
対応
Action
連絡
Escalation
監視エージェント
→ 監視マネージャ
→ 統合マネージャ
一次対応
→ 二次エスカレーション
→ 三次エスカレーション
記録/分析

障害監視の全体像


サービス
監視
システム
障害監視システム
障害対応オペレーション
検知
Detect
通知
Notice
受信
Receive
対応
Action
連絡
Escalation

3つの方向性

標準化 / セグメント化 / 類型化



- 類似問題点の共通化、ナレッジ化
- 類型に基く個別モデル化
- セミオートメーション化

- S/N比の改善
- 必要な情報を必要あるとき必要あるところへ。
- 個別事情への柔軟な対応

- 不統一によるオーバヘッドの削減
- 全体問題点の明確化、改善サイクル化
- 導入/展開の迅速化

6分野 × 3つの方向性 = 障害監視「論点表」

分野 / 領域	D.検知	N.通知	R.受信	A.対応	E.連絡	P.記録/分析
標準化	• D00 検知ログ	• N00 通知ログ	• R00 受信ログ	• A00 対応ログ	• E00 エスカレーションログ	
	• D01 検知手法 • D02 発動と復旧 (検知ID) • D03 アクティブ監視 サイジング (監視対象の選別基準) • D04 パッシブ監視 サイジング (バースト防止基準)	• N01 フォーマット • N02 通知メディア • N03 通知サイジング (流量)		• A01 対応フロー • A02 インシデント ID • A03 復旧通知による発動の消し込み • A04 対応状況の集中管理/進捗共有 • A05 短期特別対応フロー	• E01 エスカレーションフロー • E02 連絡先管理	• P01 インシデント管理 • P02 問題管理 • P03 分析/KPI
セグメント化	• D05 現象重要度 • D06 ノード重要度 • D07 影響範囲 (単体) • D08 影響サービス群	• N04 通知先	--	--	• E05 連絡先 • E06 伝達レベル	

類型化	• D09 潜在リスク要素のカタログ化
	• D10 潜在リスク アセスメント (ツール)
	• D11 現象パターン化
	• D12 ベンダコール受信
		• N05 通知リスク アセスメント (ツール)		• A06 一次対応パターン
				• A07 イレギュラルール
				• A08 リカバリパターン
				  • 手順化
				  • 自動化
				• A09 影響範囲 (複合)
					• E03 伝達項目
					• E04 連絡シナリオ	

障害監視の将来像

アラート情報の構造化

対応情報の構造化

現象モデル

対応(初動)モデル

障害監視のデザインパターン

障害検知のデザインパターン

現象モデル

対応(初動)モデル

障害監視のデザインパターン

仮説

障害の**現象パターン**と**対応パターン**、および**検知手法**は関連付けることが可能である。

実際には、不可能なものもあると考えられるが、対応付けが可能な部分については、**セミオートメーション化**できる可能性が見えてくる(かも。

検知(監視)対象オブジェクト 2つの属性

- **定性モデル**
  - 対象の動作や構成的状態から障害の発生を検知するモデル
- **定量モデル**
  - 対象の量的な状態により、障害の発生を検知するモデル

各モデルについて、共通の「**状態パターン**」を持っていると仮定。

障害監視 9モデル

定性モデル

Service: サービス提供モデル

Process: プロセスモデル

Module: プログラムモジュールモデル

Varidate: データ検証モデル

Route: 経路モデル

Node: ノードモデル

定量モデル

Relative Value: 相対量モデル

Absolute Value: 絶対量モデル

Unknown/Complex: 不明/複合要因

定性6モデル

Service: サービス提供モデル

Process: プロセスモデル

Module: プログラムモジュールモデル

Varidate: データ検証モデル

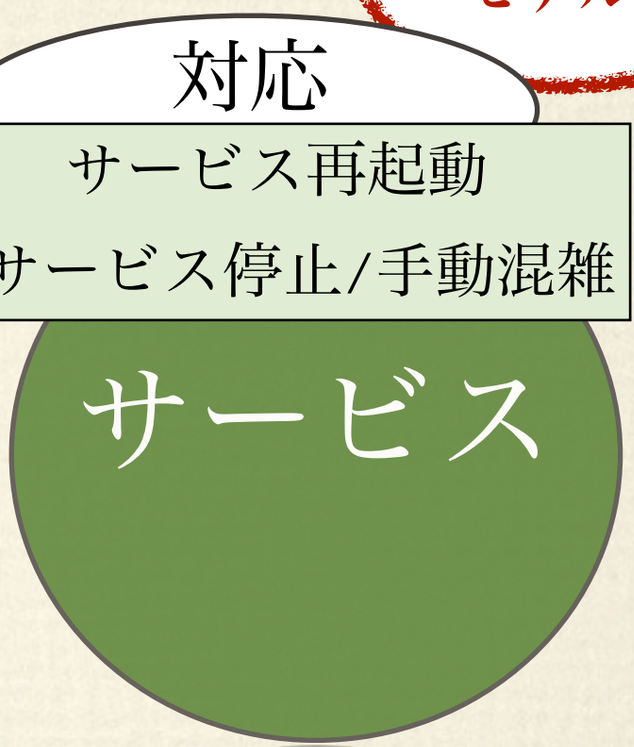
Route: 経路モデル

Node: ノードモデル

Service: サービス提供モデル

定性
モデル1

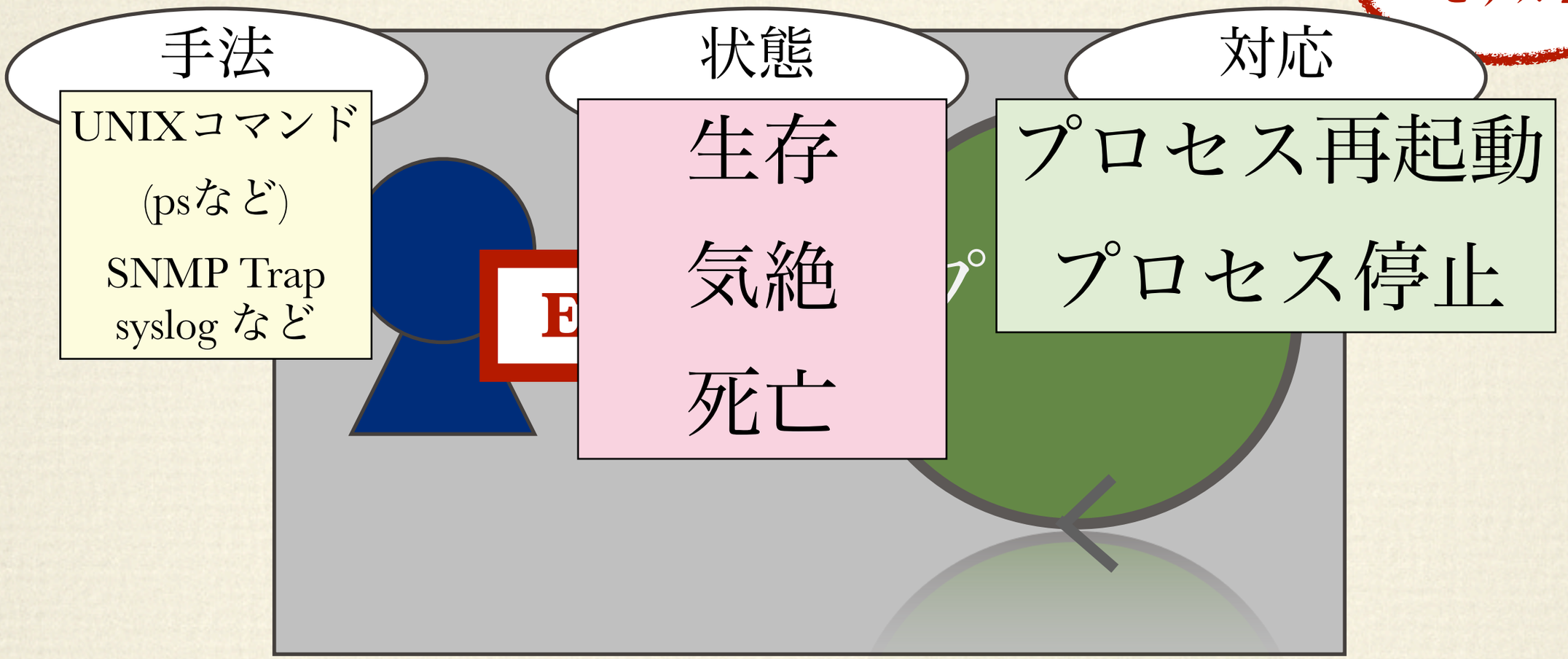
手法	状態	対応
プロトコルセッション監視 - ICMP/TCPポート/各種プロトコル ノード間サービス監視 (request/API) サービス遅延監視 混雑/sorry監視 サービスステップ監視 モジュール間サービス監視 (リターンコードなど)	正常 遅延(非混雑) 混雑 異常処理 停止	サービス再起動 サービス停止/手動混雑



外部から**サービス提供**をチェック

Process: プロセスモデル

定性
モデル2



外部から**プロセスの存在**をチェック

Module: プログラムモジュールモデル

定性
モデル3

手法
(エラーコード/ログ)
モジュール内監視
ミドルウェア監視

状態
正常
遅延
エラー
例外
停止

対応
(ジョブ型)
後続プロセス保留
プロセス再実行
手動再実行
(デーモン型)
プロセス再起動
プロセス停止

プロ
モジ

Error Log

内部処理から検出

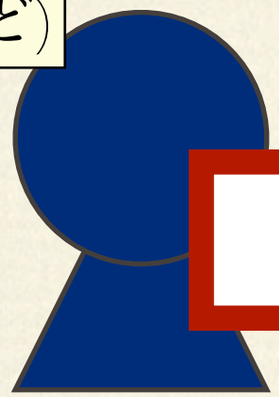
Validate: データ検証モデル

定性
モデル4

手法
バリデタ
(lint系ツールなど)

状態
正常
異常
不存在

対応
データ登録
データ修正
データ提供停止



存在するデータが**正常な形式**かどうか検証

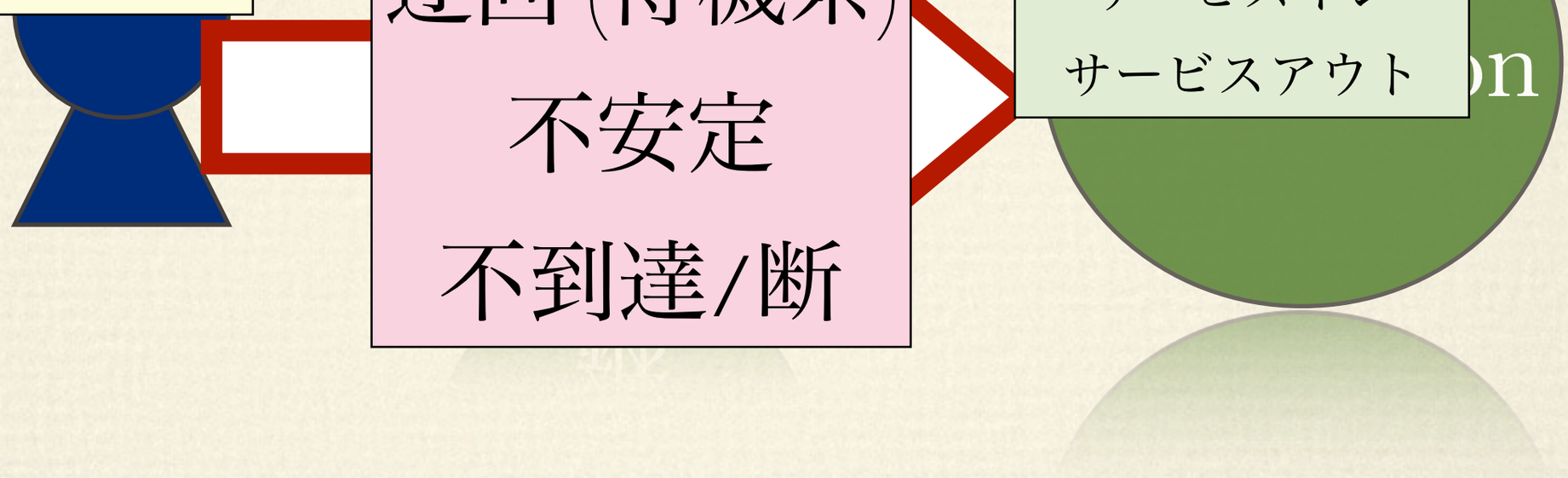
Route: 経路モデル

定性
モデル5

手法
ハートビート
経路探索

状態
正常 (稼動系)
迂回 (待機系)
不安定
不到達/断

対応
フェイルオーバ
フェイルバック
サービスイン
サービスアウト



現在の**経路**が正常かどうかを検知

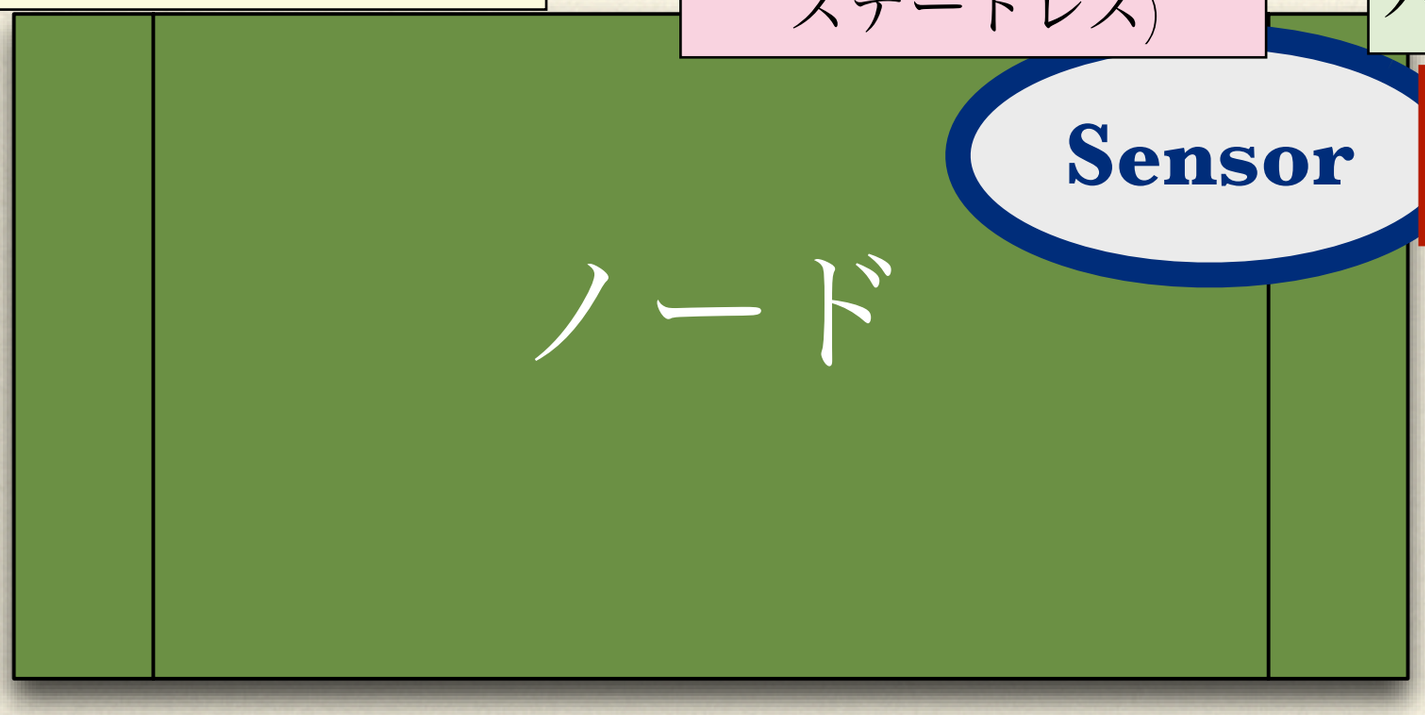
Node: ノードモデル

定性
モデル6

手法
OSログイン/改ざん監視
ハードウェアエラー監視 など

状態
(復旧情報がないため
ステートレス)

対応
サービスアウト
ハードウェア保守対応



ノードが一方的に**Trapを発呼**する。

定量2モデル

Relative Value: 相対量モデル

Absolute Value: 絶対量モデル

Relative Value: 相対量モデル

定量
モデル1

手法	状態	対応
UNIXコマンド(stat系) SNMP Trap / syslog リソースアプライアンス管理ツール MRTG など	good(潤沢/存在) 報告域 (例: 残25%未満) 注意域 (例: 残10%未満) 警告域 (例: 残5%未満) 枯渇/不存在 (残0%)	リソース追加 消費要因削除 リソース提供停止

%

or

Persentage

監視対象リソースを**相対量**で監視する

Absolute Value: 絶対量モデル

定量
モデル2

手法

UNIXコマンド(stat系)

SNMP Trap / syslog
リソースアプライアンス管理ツール

MRTG
など

状態

絶対値

Sensor

対応

リソース追加

消費要因削除

リソース提供停止

Amount

∞

監視対象リソースを**絶対量**で監視する

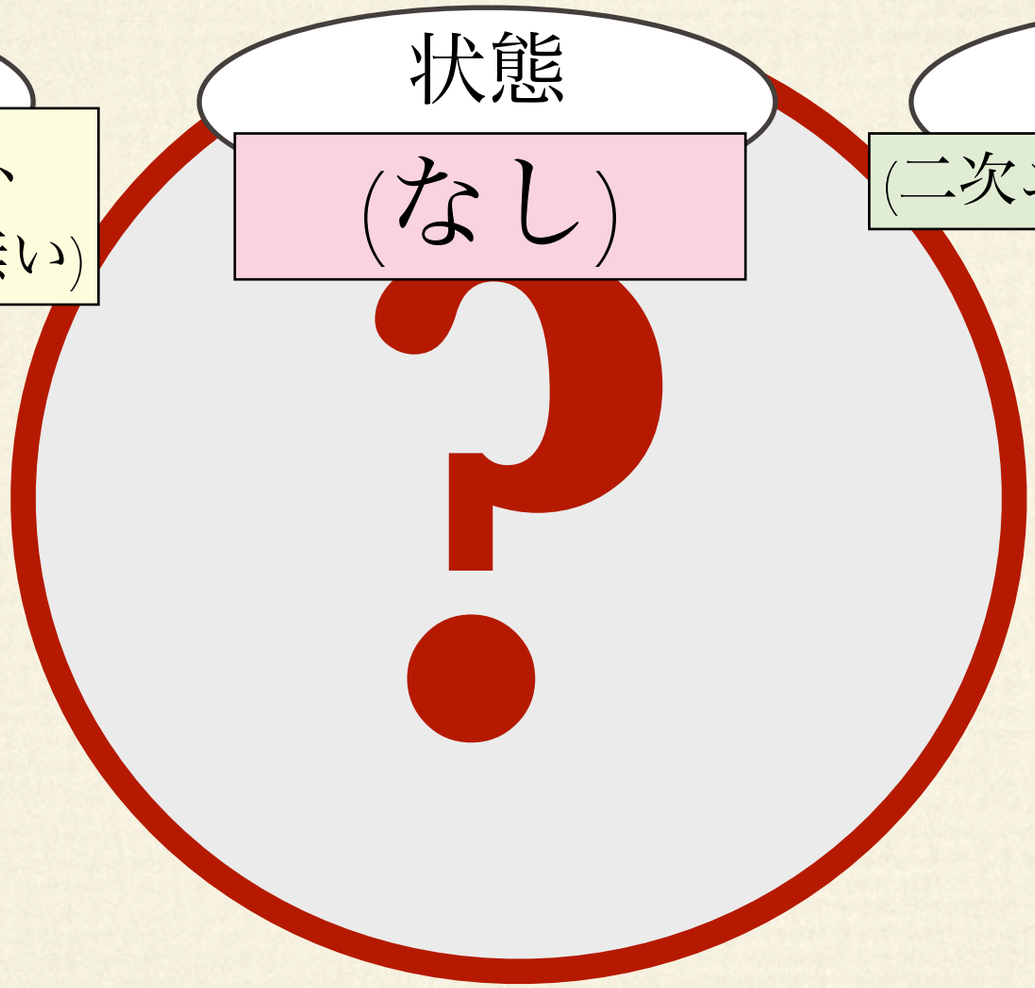
定性/定量でない **Unknown/Complex**

手法

(要因を分解しないと、
実質的な検知方法は無い)

状態

(なし)



対応

(二次エスカレーション)

不明/複合要因

障害監視の将来像

アラート情報の構造化

対応情報の構造化

現象モデル

対応(初動)モデル

障害監視のデザインパターン

通知メッセージの類型化

対応情報の構造化

受信メッセージの定型化

- エラーを受信しても、担当じゃない人にはなんだかよくわからない。
- エラー現象を標準化しエラーコードで表現
- HTTP/SMTPエラーコードを参考

現象コード体系

- 100番台: Informational (コマンドの確認待ち)
- 200番台: Successful (成功)
- 300番台: Redirection (後続データ待ち)

x00〜x49: 共通利用のため予約

x50〜x99: 個別自由利用

- **400番台: 要求側エラー (リクエストエラー)**
- **500番台: 処理側エラー (レスポンスエラー)**
- **600番台: リモート側エラー (コネクションエラー)**

400番台: 要求側エラー (リクエストエラー)

- クライアント側の理由によりリクエストがエラーとなった。
- (リクエスト前の内部エラー、存在する相手への不適切なリクエストなど)

500番台: 処理側エラー (レスポンスエラー)

- クライアント側は正常にリクエストしたが、サーバ側の理由によりエラーが帰ってきた。
  - (サーバの内部エラーなど)

600番台: リモート側エラー (コネクションエラー)

- クライアント側のリクエストが、リモート側と正常にやりとりできなかった。
- (存在せず/タイムアウト/遅延)

障害監視の将来像

アラート情報の構造化

対応情報の構造化

現象モデル

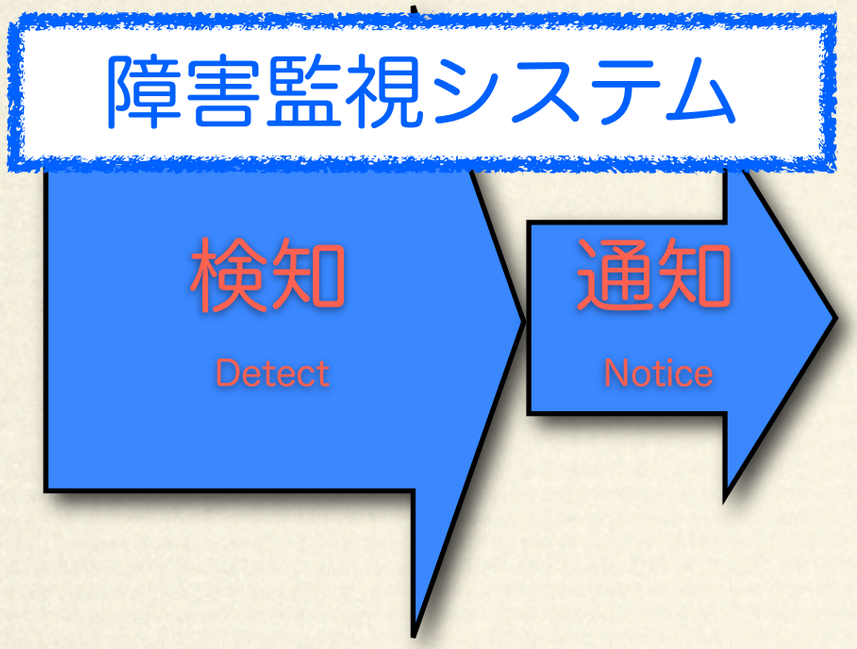
対応(初動)モデル

障害監視のデザインパターン

通知情報の構造化

アラート情報の構造化

アラート情報の構造化



検知情報の構造化: syslog

通知情報の構造化: YAML/JSON

検知情報の構造化

syslog

前振り : 実はsyslog好き

- 2002年頃にSoftwareDesignでsyslogの記事を書いた(記憶がなんとなくある)
- 2002年頃に jus 勉強会で syslogネタの講師をやった(記憶がなんとなくある)
- そして2010年 (キターッ

検知情報の構造化

- 従来のログは、勝手放題で扱いが難しかった。
- そこで **RFC5424**
  - 2009年3月に **syslog の RFC** が大幅に改訂
  - 旧来の RFC3164は obsoluted に

新世代 syslog (RFC5424)

～障害監視フレームワーク(その3)～

波田野 裕一

(日本UNIXユーザ会)

Daemon Festival 2010-06-15

RFC5424の概要

- **syslog**メッセージは、**HEADER** / **STRUCTURED-DATA** / **MSG** の3部構成

- 人が読むだけでなく、システムに読ませることを意識したフォーマット

- 二次元ハッシュ構造を保持できる。

RFC5424 メッセージ構成

HEADER

syslogd

syslogd間の伝達情報

必須

STRUCTURED-DATA

system

構造化情報 (今回の目玉)

MSG

human

従来通りの自由奔放なメッセージ

話題の

UTF-8に対応!

RFC5424のいいとこ取り

1. syslogd 間の通信を意識しなければ、

   HEADERは不要

2. 人間が読むことを意識しなければ、

   MSGも不要

3. **STRUCTURED-DATA** 形式を利用する

   ことで、**セマンティックなログ**を実現

構造化ログのサンプル

```
[base@ version="1.0" date="2009-07-28" time="16:06:00" zone="+0900"
detectModel="service" ident="node001.example.co.jp:80:HTTP:/
index.html:serviceModel:node"][phenomenon@ version="1.0" id="404"]
[agent@ node="ismon411.example.co.jp" module0="monitor"
module1="HTTP"][serviceModel@ status="dead" dataType="node"]
[nodeType@ name="node101.example.co.jp" port="80" protocol="HTTP"
resource="/index.html"]
```

- **hoge@** が1次ハッシュのキー
- **hoge=** が2次ハッシュのキー

検知パターン

```
[base@ version="1.0" date="2009-07-28" time="16:06:00" zone="+0900"
detectModel="service" ident="node001.example.co.jp:80:HTTP:/
index.html:serviceModel:node"][phenomenon@ version="1.0" id="404"]
[agent@ node="ismon411.example.co.jp" module0="monitor"
module1="HTTP"][serviceModel@ status="dead" dataType="node"]
[nodeType@ name="node101.example.co.jp" port="80" protocol="HTTP"
resource="/index.html"]
```

検知パターンによって、

状態パターンが異なる。

データ構造パターン

```
[base@ version="1.0" date="2009-07-28" time="16:06:00" zone="+0900"
detectModel="service" ident="node001.example.co.jp:80:HTTP:/
index.html:serviceModel:node"][phenomenon@ version="1.0" id="404"]
[agent@ node="ismon411.example.co.jp" module0="monitor"
module1="HTTP"][serviceModel@ status="dead" dataType="node"]
[nodeType@ name="node101.example.co.jp" port="80" protocol="HTTP"
resource="/index.html"]
```

たとえば、サーバホストとDBインスタンスでは、アラート項目が違う

現象(エラー)パターン

```
[base@ version="1.0" date="2009-07-28" time="16:06:00" zone="+0900"
detectModel="service" ident="node001.example.co.jp:80:HTTP:/
index.html:serviceModel:node"][phenomenon@ version="1.0" id="404"]
[agent@ node="ismon411.example.co.jp" module0="monitor"
module1="HTTP"][serviceModel@ status="dead" dataType="node"]
[nodeType@ name="node101.example.co.jp" port="80" protocol="HTTP"
resource="/index.html"]
```

現象パターンに対応したコードを記載、

通知メッセージの類型化

通知情報の構造化

YAML/JSON

ツールが扱いやすいフォーマット

- **YAML**形式 (人が読まないならJSONでもいい)
- RubyやPerlでの変数取り込みが容易。
- 単純なインデント形式のため、echo文での実装も容易。
- 比較的人間も読みやすいフォーマット
- UTF8による記述であれば、非アスキー文字への対応も可能。

YAML形式による多次元ハッシュ構造

- **項目定義が柔軟**。(対象システム毎に異なる構造を持つことが可能)
- **拡張が容易**。必要ない項目が存在しても影響しない。(ハッシュの性質)
- 「影響サービス範囲」のみ、(タグ的に扱う関係上)配列形式を採用。

媒体への非依存

メールヘッダなどに依存しないため、ファイルでの受け渡し(NFS/SCP/HTTP/FTP)などが可能。

YAMLのサンプル

```
base:
        version: 1.0
        date: 2009-07-28
        time: 16:06:00
        zone: +0900
        detectModel: service
        ident: node001.example.co.jp:80:HTTP:/index.html:serviceModel:node
phenomenon:
        version: 1.0
        id: 404
agent:
        node: ismon411.example.co.jp
        module0: monitor
        module1: HTTP
```

アラート情報の構造化

サンプル

構造化ログのサンプル

```
[base@ version="1.0" date="2009-07-28" time="16:06:00" zone="+0900"
detectModel="service" ident="node001.example.co.jp:80:HTTP:/
index.html:serviceModel:node"][phenomenon@ version="1.0" id="404"]
[agent@ node="ismon411.example.co.jp" module0="monitor"
module1="HTTP"][serviceModel@ status="dead" dataType="node"]
[nodeType@ name="node101.example.co.jp" port="80" protocol="HTTP"
resource="/index.html"]
```

YAMLで通知してみる

```
base:
        version: 1.0
        date: 2009-07-28
        time: 16:06:00
        zone: +0900
        detectModel: service
        ident: node001.example.co.jp:80:HTTP:/index.html:serviceModel:node
phenomenon:
        version: 1.0
        id: 404
agent:
        node: ismon411.example.co.jp
        module0: monitor
        module1: HTTP
```

YAMLで通知してみる

```
serviceModel:
        status: dead
        dataType: node
nodeType:
        name: node001.example.co.jp
        port: 80
        protocol: HTTP
        resource: /index.html
```

障害監視の将来像(まとめ)

アラート情報の構造化

対応情報の構造化

現象モデル

対応(初動)モデル

障害監視のデザインパターン

まとめ

- **モデリングの重要性、オブジェクト指向分析**
  - SOAやBPMへの道 (現実的になりつつある)
- **障害監視のフレームワーク化**
  - 障害のモデル化、監視のデザインパターン化
- **障害を減らすにはまず運用を知ることが先**
  - という現場が多い...