Droonga 移行後の世界

須藤功平　株式会社クリアコード

Droonga Meetup 1
2014/07/30

目指すこと

Droonga移行後の
システムを
想像できること

話題

- 運用方法（重点的に説明）
- 特有の機能（軽く説明）
- 今後の開発（軽く説明）

質疑応答時間アリ

- 気になったら…
  - メモ→最後に質問
  - その場で質問
- 気になることは聞いて欲しい
  - 何が気になるか知りたい

話題1

- 運用方法
- 特有の機能
- 今後の開発

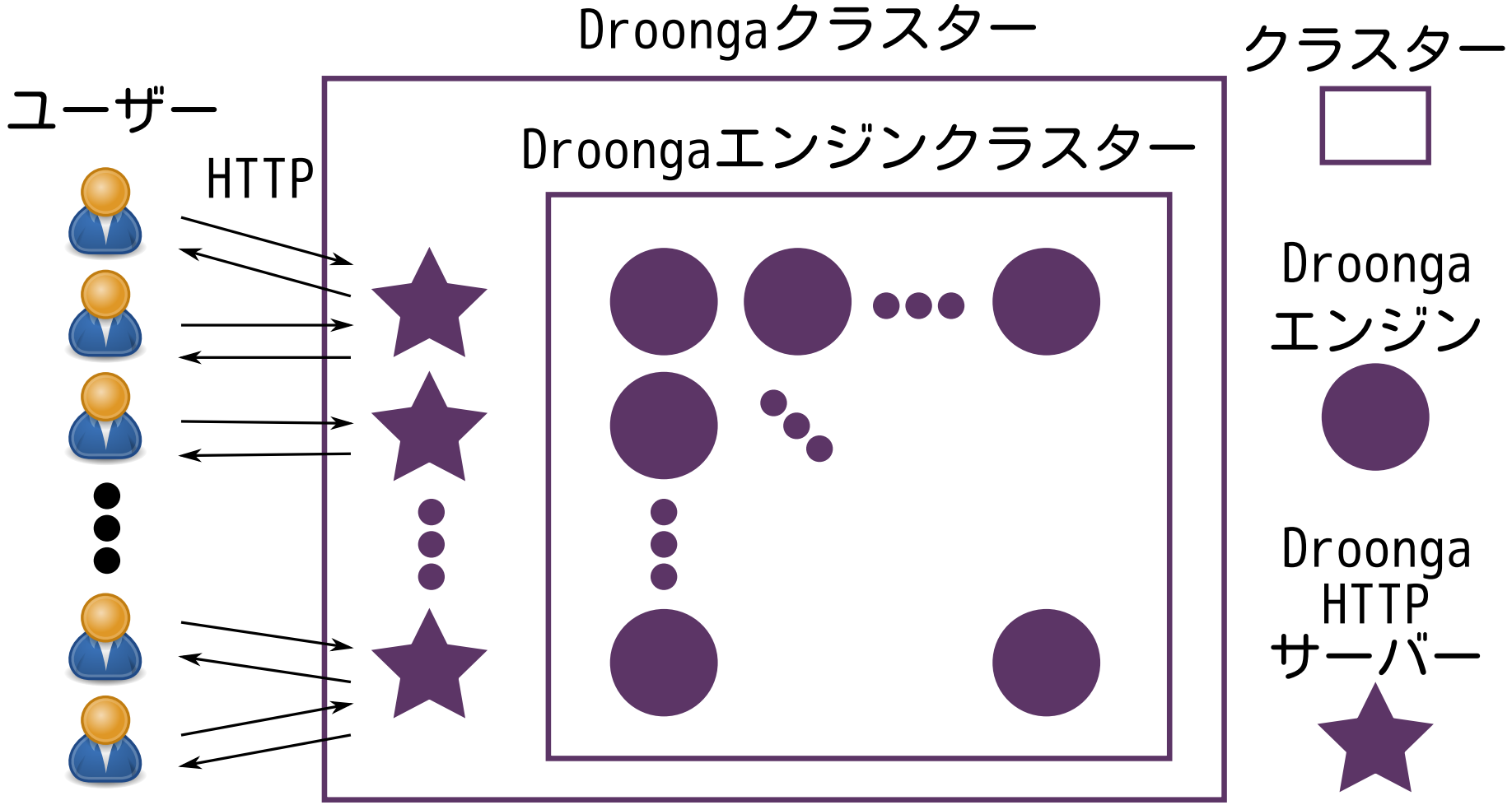
運用方法

- 死活監視
- パフォーマンス監視
- ノード構成の変更方法

死活監視

- システムが正常動作しているか継続的に確認
  - →システムの構成要素は？
  - →正常動作とは？
- Droongaの場合
  - クラスターの構成要素は？
  - それぞれの提供サービスは？

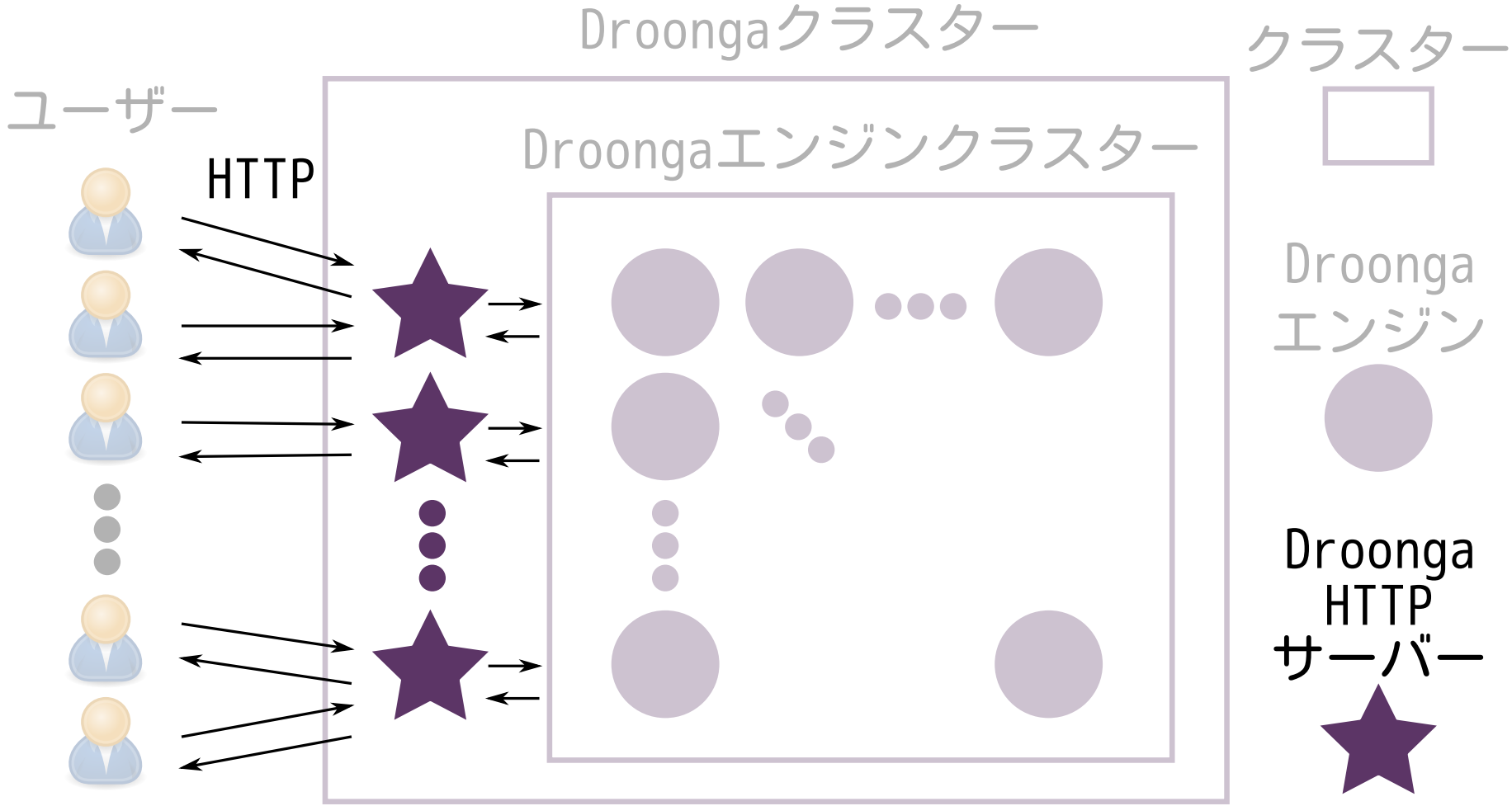
Droongaクラスター



ポイント

- SPOFなし
  - 管理ノードなし
  - すべてのエンジンは同列
  - 管理情報はすべてのエンジンが持つ
- ユーザーはHTTPでアクセス

HTTPサーバー図


Droongaクラスター
クラスター
ユーザー
Droongaエンジンクラスター
HTTP
Droonga
エンジン
Droonga
HTTP
サーバー

HTTPサーバー

- 1番ユーザーに近い
  - ここで正常動作を確認
  - →Droongaクラスターは正常動作
- n台構成推奨
  - 冗長化・負荷分散
  - ロードバランス対象

HTTPサーバー：役割

- HTTPのAPIを提供
- 実体はプロキシ
  - バックエンドは Droongaエンジンクラスター
  - プロトコルを変換
  - データを持っていない

HTTPサーバー：監視図

HTTPサーバー：監視項目

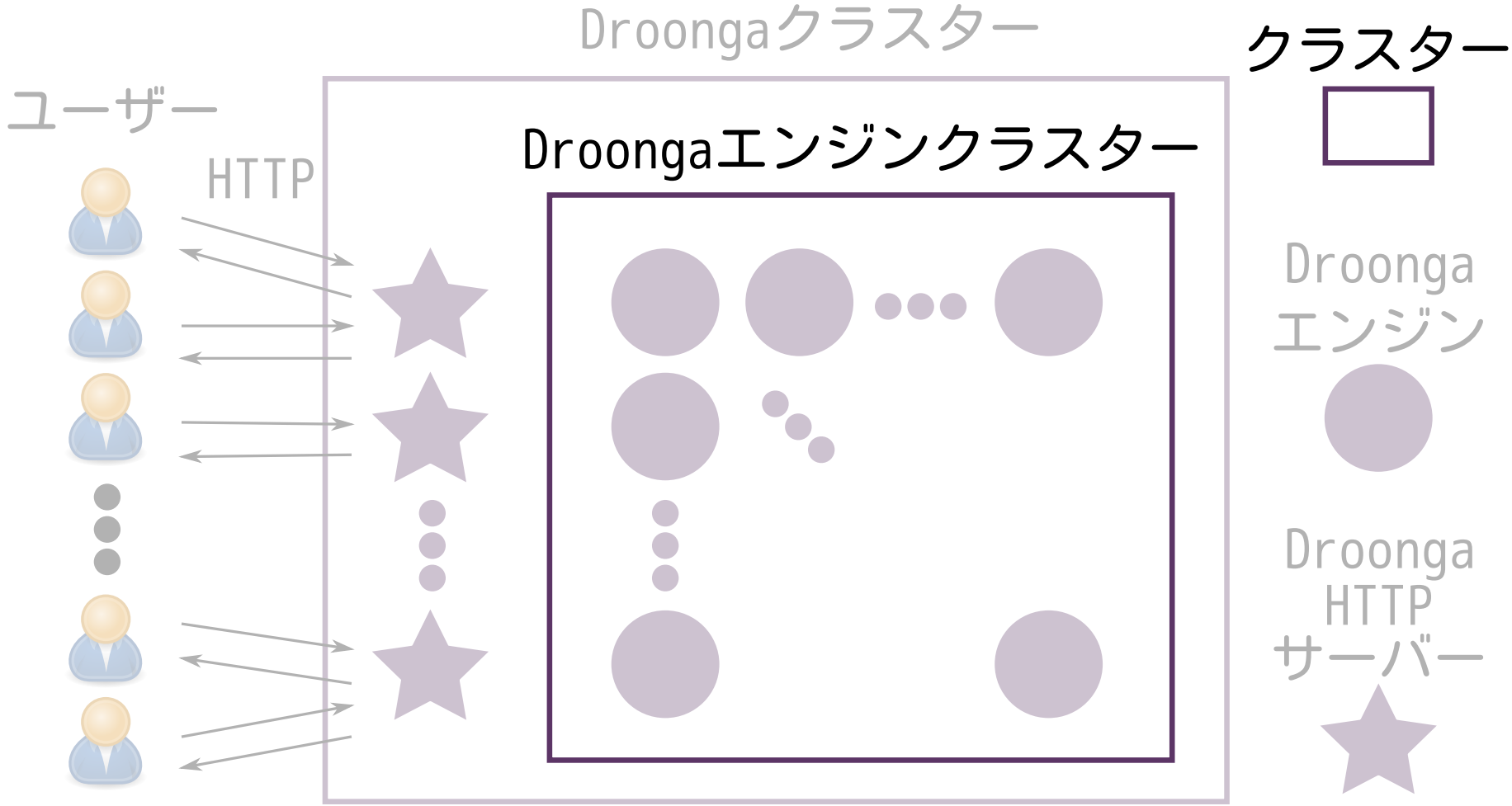
- HTTPを話せるか？
  - 基本的な監視
  - "/"をGETして200か監視
- プロキシできているか？
  - 詳細な監視
  - 検索してヒットするか監視

HTTPサーバー：監視効果

- HTTPを話せるか？
  - サービスは生きている
  - 役割を果たしているとは限らない
- プロキシできているか？
  - 役割を果たしている
  - 実現方法はデータに依存

HTTPサーバー：監視効果図

エンジンクラスター図


Droongaクラスター
クラスター
ユーザー
HTTP
Droongaエンジンクラスター
Droonga
エンジン
Droonga
HTTP
サーバー

エンジンクラスター

- エンジンが連携して動作
- クラスター内にnレプリカ存在
- 障害度合いとサービスレベル
  - 低: 100%サービス利用可
  - 中: 構成変更中の書き込み不可
  - 高: サービス不可（参照も不可）

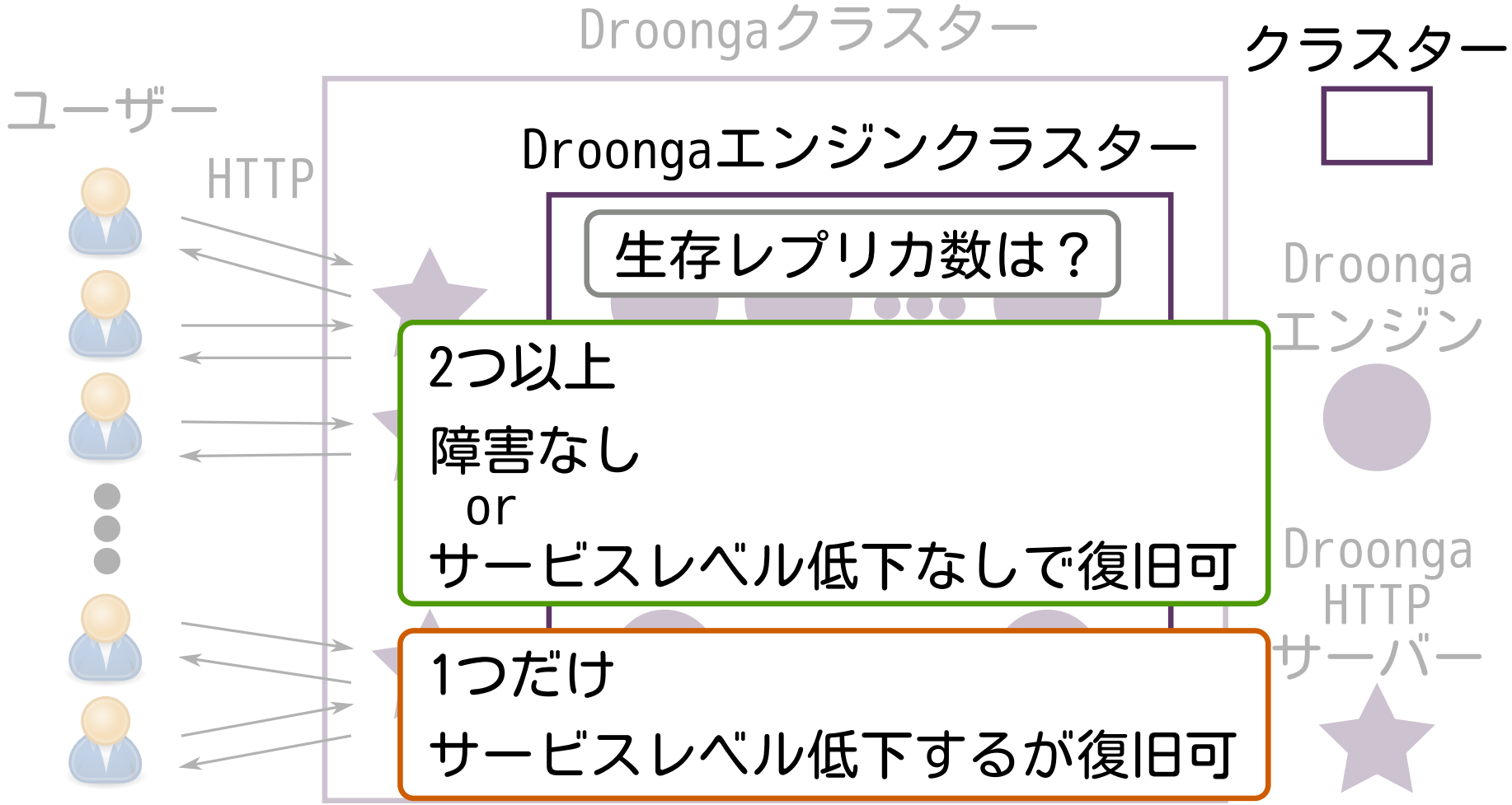
監視項目

- 2つ以上のレプリカが生存？
  - 障害度合い：低（100%サービス利用可）
  - 構成を考慮した死活チェック
- 1つ以上のレプリカが生存？
  - 障害度合い：中（構成変更中の書き込み不可）
  - 検索してヒットするか監視

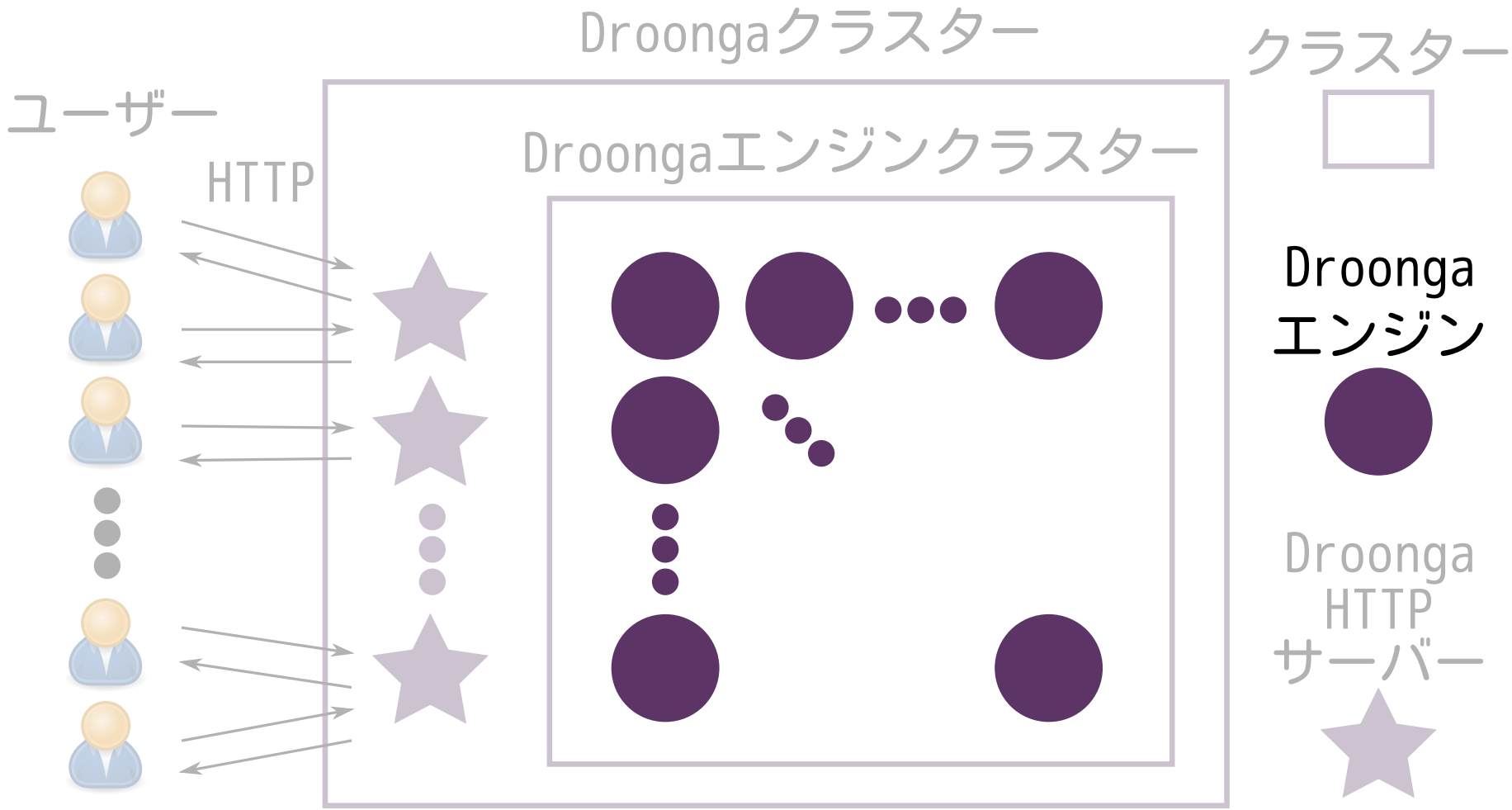
監視効果

- 2つ以上のレプリカが生存？
  - 障害なし or
  - サービスレベル低下なしで復旧可能
- 1つ以上のレプリカが生存？
  - サービスは提供できている
  - サービスレベル低下するが復旧可能

監視効果図


Droongaクラスター
クラスター
ユーザー
HTTP
Droongaエンジンクラスター
生存レプリカ数は？
Droonga
エンジン
2つ以上
障害なし
or
サービスレベル低下なしで復旧可
Droonga
HTTP
サーバー
1つだけ
サービスレベル低下するが復旧可

エンジン図


Droongaクラスター
クラスター
ユーザー
Droongaエンジンクラスター
HTTP
Droonga
エンジン
Droonga
HTTP
サーバー

エンジン

- データを持ち、処理する
- 1エンジンnワーカー
  - マルチプロセス
- 自律的に構成変更を検知
  - 管理ノードなし
  - あるエンジンがダウンしたら
    自分が持つクラスター情報を更新

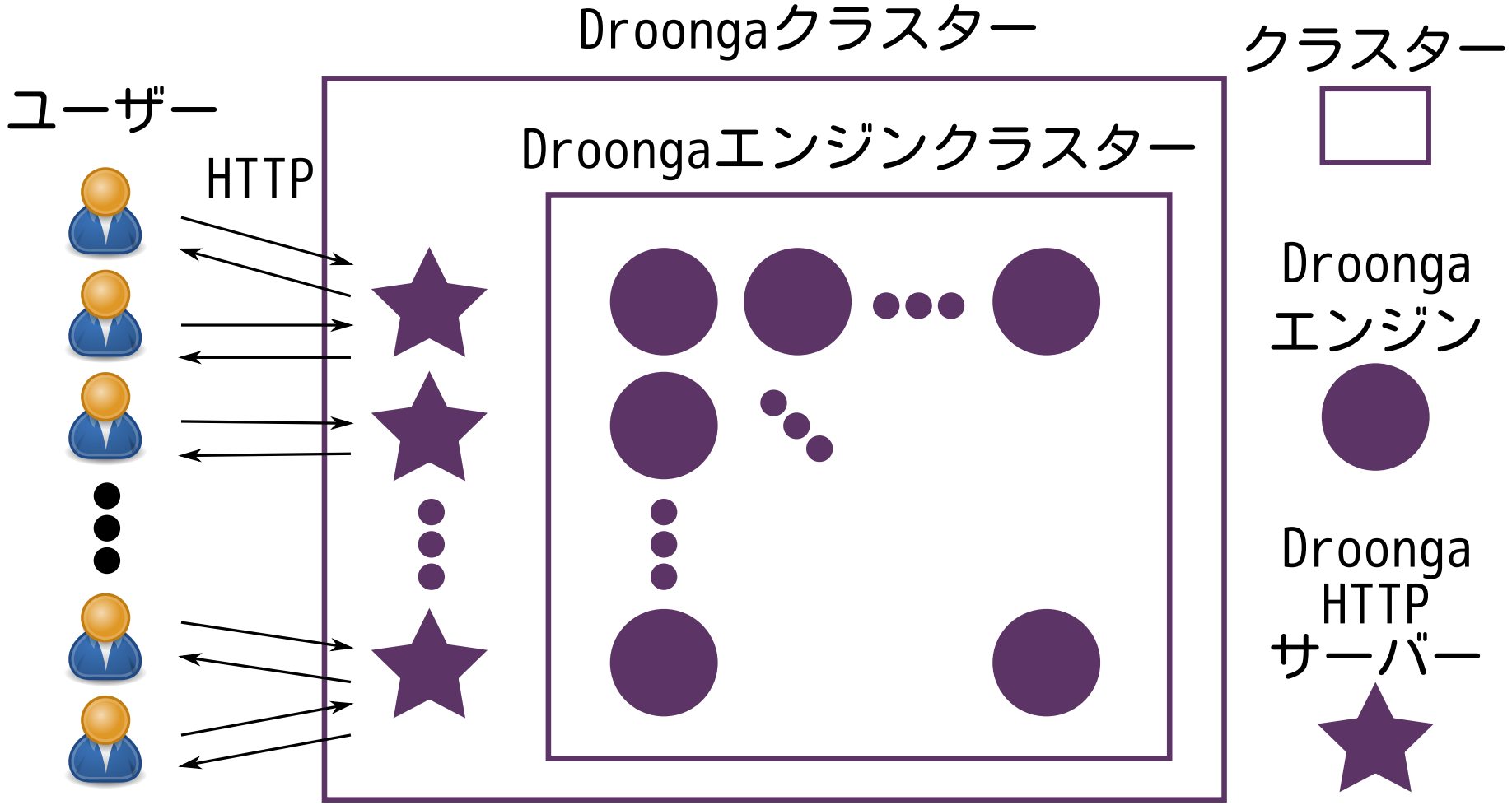
監視項目

- 監視しなくてよい
- エンジンクラスター監視で十分

死活監視のおさらい

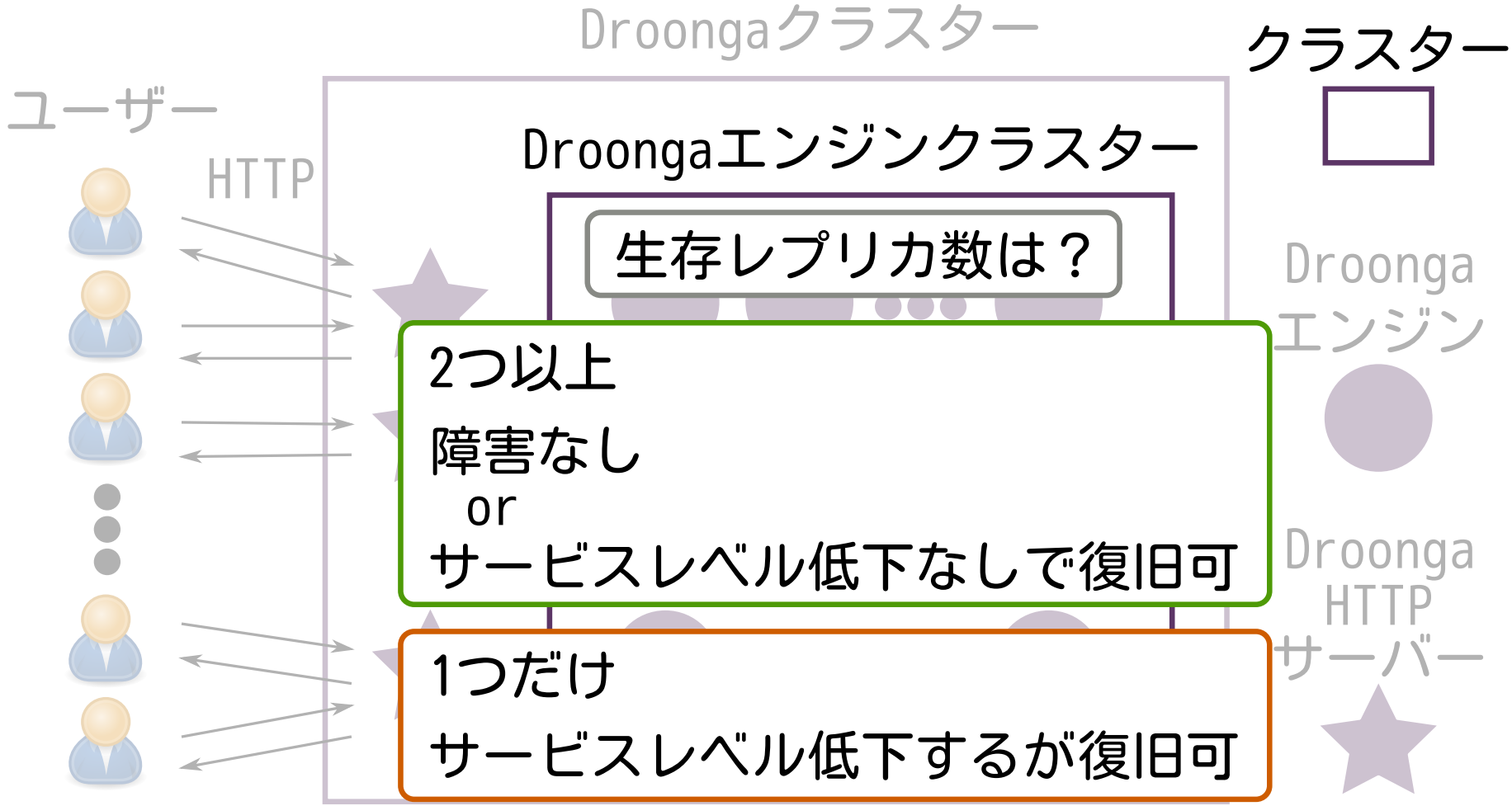
- 目的
  - システムの正常動作を確認
- システムの構成要素
  - HTTPサーバー（監視対象）
  - エンジンクラスター（監視対象）
  - エンジン

Droongaクラスター



HTTPサーバーの監視

エンジンクラスターの監視



死活監視

- HTTPサーバー
  - HTTPの機能をチェック
  - プロキシの機能をチェック
- エンジンクラスター
  - 生存レプリカ数をチェック
- エンジン
  - 死活監視しなくてよい

運用方法：死活監視

- **死活監視**
- パフォーマンス監視
- クラスター構成の変更方法

パフォーマンス監視

- 死活監視
- **パフォーマンス監視**
- クラスター構成の変更方法

パフォーマンス監視

- 監視項目
- 監視方法

監視項目

- 検索時間
  - 目的：スロークエリを発見
- キャッシュヒット率
  - 目的：検索処理の削減
- CPU使用率
  - 目的：ボトルネックの解消

監視方法

- 検索時間
  - アプリケーション側で計測
- キャッシュヒット率
  - "/statistics/cache"をGET
- CPU使用率
  - 既存の統計情報可視化ツール
  - 例：Munin

検索時間：計測場所と粒度

- アプリケーション
  - 全体の時間
- HTTPサーバー
  - Droonga側の処理全体の時間
    （X-Response-Time HTTPヘッダー）
- エンジン
  - 各処理の時間
    （クエリーログ：未実装）

キャッシュヒット率

http://.../statistics/cache

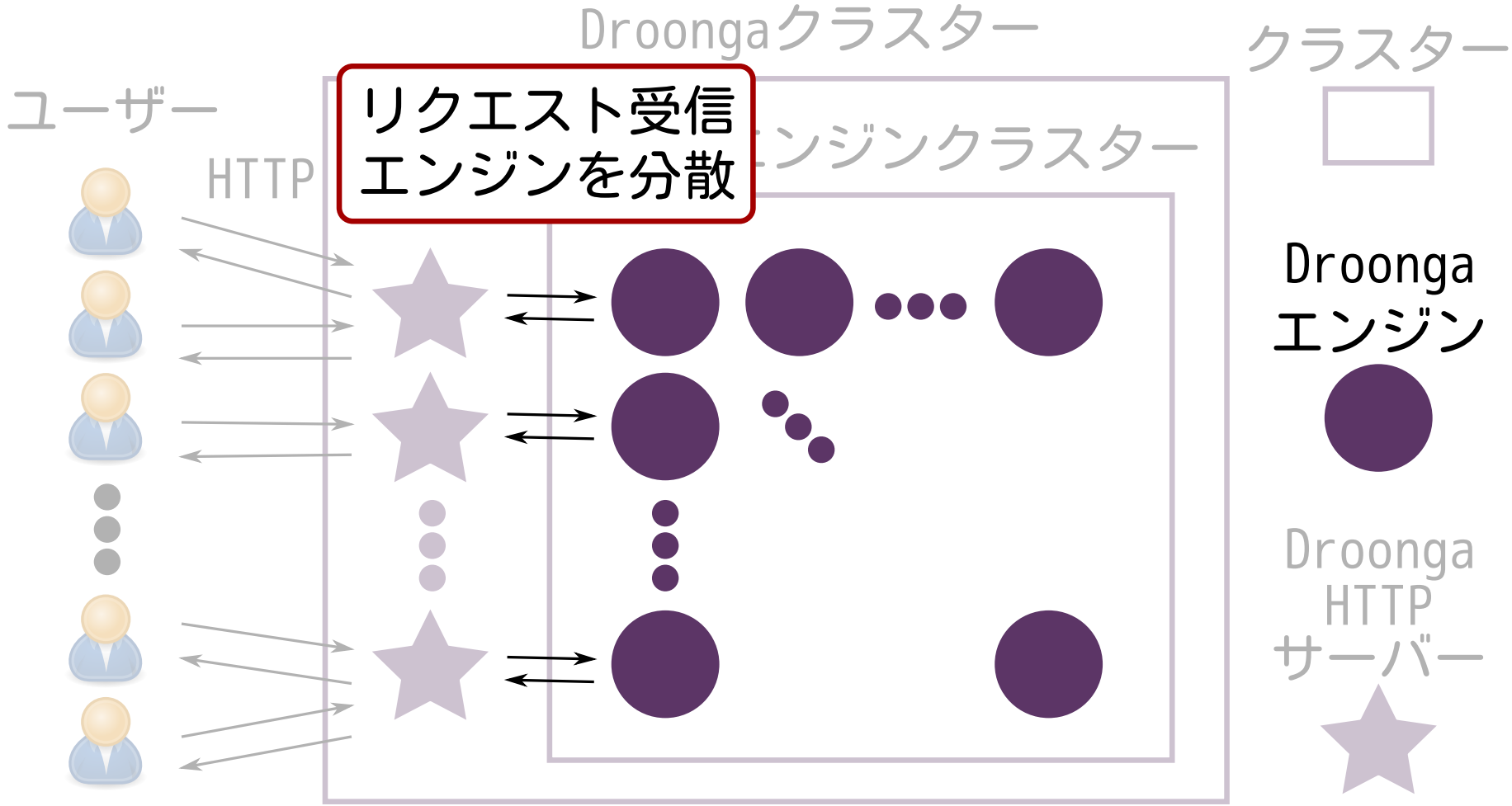
```
{
 "nGets": 8,
 "nHits": 6,
 "hitRatio": 0.75
}
```

CPU使用率

- ボトルネックはCPUになるはず
- HTTPサーバーの場合
  - 数を増やして負荷分散
- エンジンの場合
  - CPUのコア数までワーカーを増やす
  - リクエスト受信エンジンを分散

HTTPサーバーの負荷分散

エンジンの負荷分散


Droongaクラスター
クラスター
ユーザー
HTTP
リクエスト受信
エンジンを分散
エンジンクラスター
Droonga
エンジン
Droonga
HTTP
サーバー

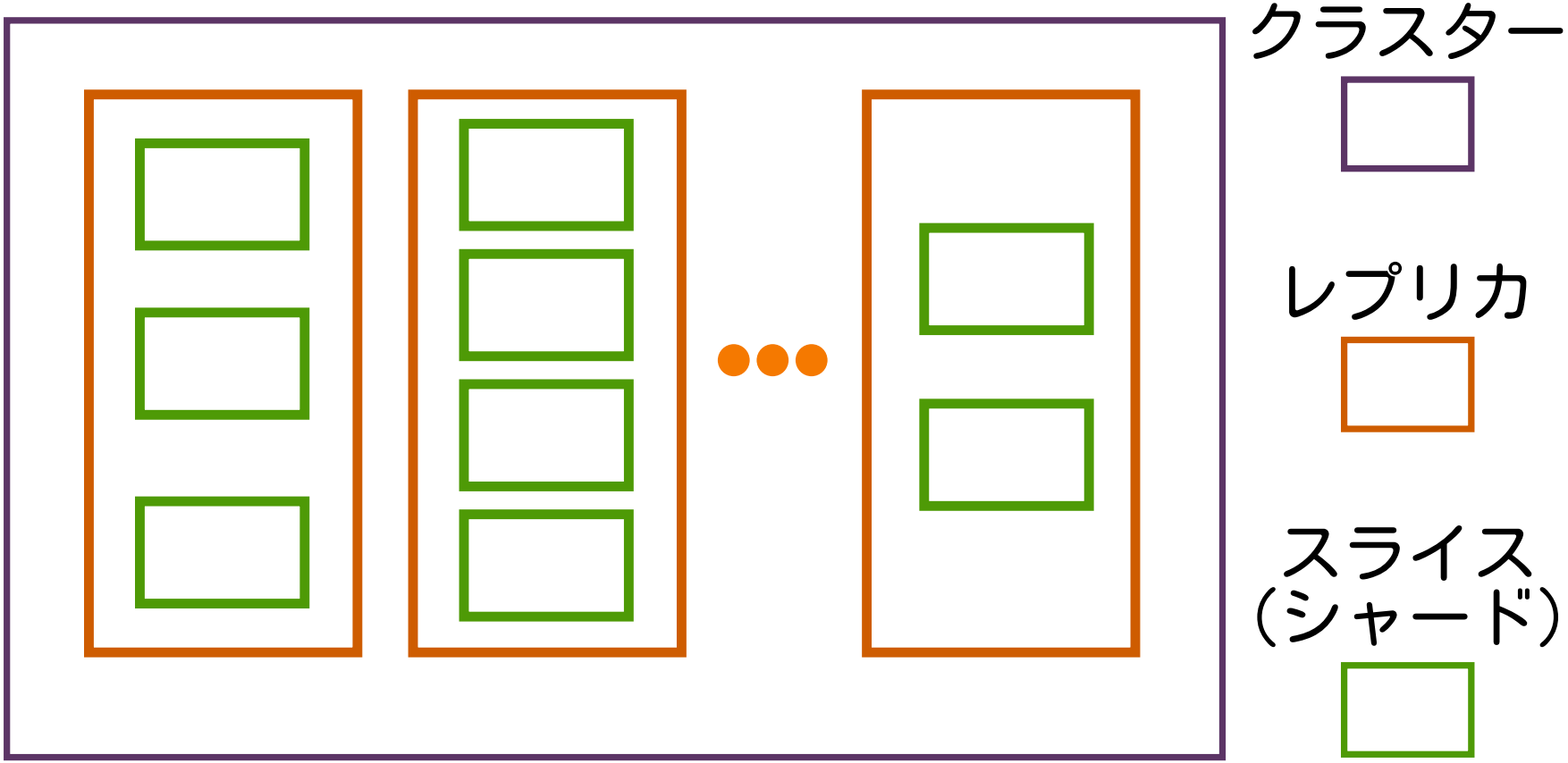
おさらい

- 目的
  - 高速化
  - ボトルネックの解消
- 監視項目
  - 検索時間（ボトルネックの解消）
  - キャッシュヒット率（高速化）
  - CPU使用率（ボトルネックの解消）

構成変更

- 死活監視
- パフォーマンス監視
- **クラスター構成の変更方法**

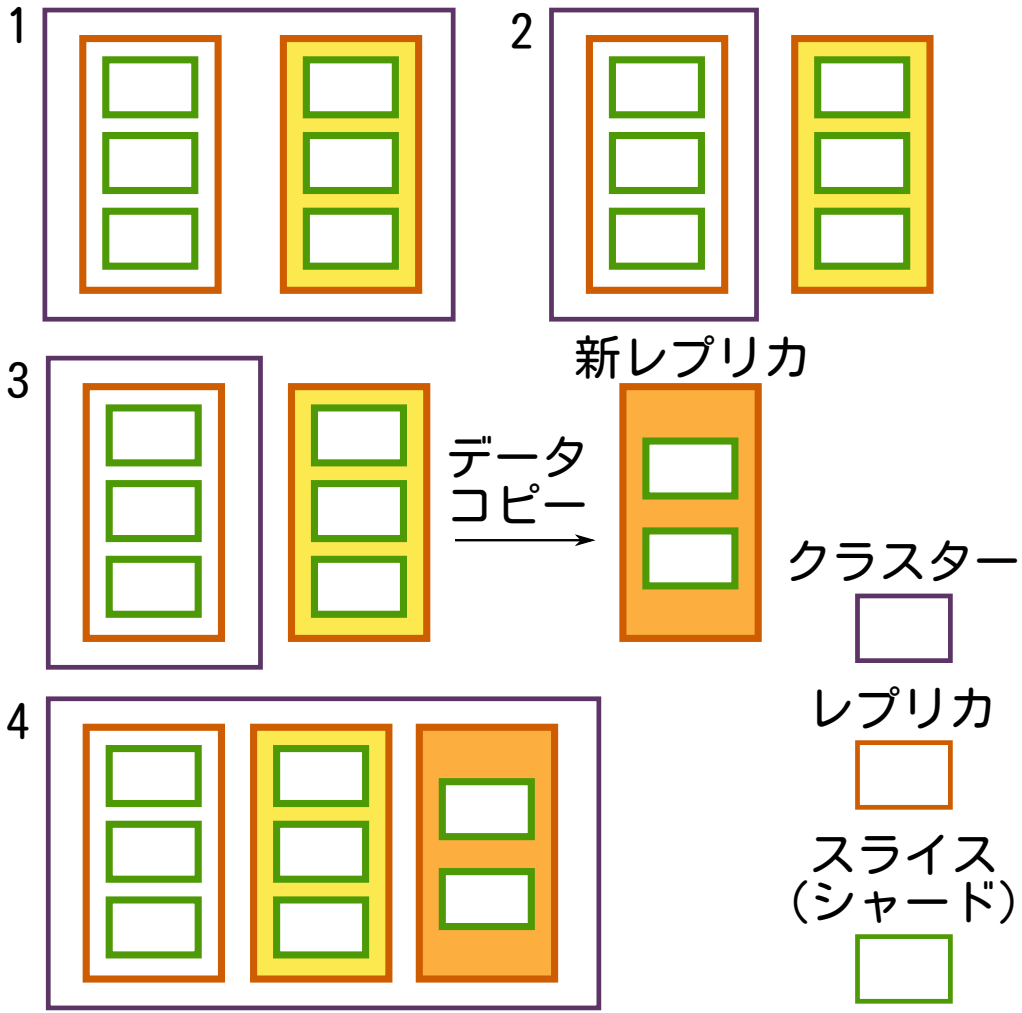
Droongaクラスターの構成図



Droongaクラスターの構成

- シャーディング構成
  - レプリカ毎に設定可
- シャーディング構成変更
  - 運用しながら変更可
- レプリカの増減
  - 運用しながら変更可

構成変更の基本作戦図



構成変更の基本作戦

- レプリカを1つ外す
- ↑からデータをダンプ
- ↑を新しいレプリカに投入
- 外したレプリカを戻す
- 新しいレプリカを追加

運用方法のおさらい

- 死活監視
  - 構成要素毎に紹介
- パフォーマンス監視
  - 項目と目的を紹介
- クラスター構成の変更方法
  - 特徴と流れを紹介

特有の機能

- 運用方法
- **特有の機能**
- 今後の開発

特有の機能

- 多段ファセット
  - Solrでいうピボットファセット
  - Elasticseachでいうaggregations
- ファセット内のレコードを取得
  - 利用例：スレッドでグループ化し、各スレッド内のコメントを数件表示

今後の開発

- 運用方法
- 特有の機能
- **今後の開発**

今後の開発1

- Groongaとの互換性強化
  - 未実装の機能を実装
  - よく使われている機能から順に
- 検索処理の高速化
  - 通信量を減らす
  - ルーティング周りの処理の軽量化

今後の開発2

- 導入の簡易化
  - パッケージの提供
  - Chefとの連携
- 運用支援機能の開発
  - ダッシュボード
  - リクエスト受信エンジンを自動で負荷分散

まとめ

- 運用方法
  - 監視・構成変更方法
- 特有の機能
  - Groongaのコアができることを解放
- 今後の開発
  - いろいろ
  - ユーザーの声を考慮して優先度決定