Pacemakerで学ぶ HAクラスタ in Osaka

OSC2019 Osaka
2019/1/26

Linux-HA Japan
竹田　健二

はじめに：本日の話の流れ

1.可用性とクラスタ
- 可用性とは何か、可用性を向上させるためにはどうすればよいか
- 「クラスタ」の必要性と分類について説明

2.Pacemakerとは
- Pacemakerの主な機能について説明

3.PG-REXとは
- PG-REXの主な機能について説明

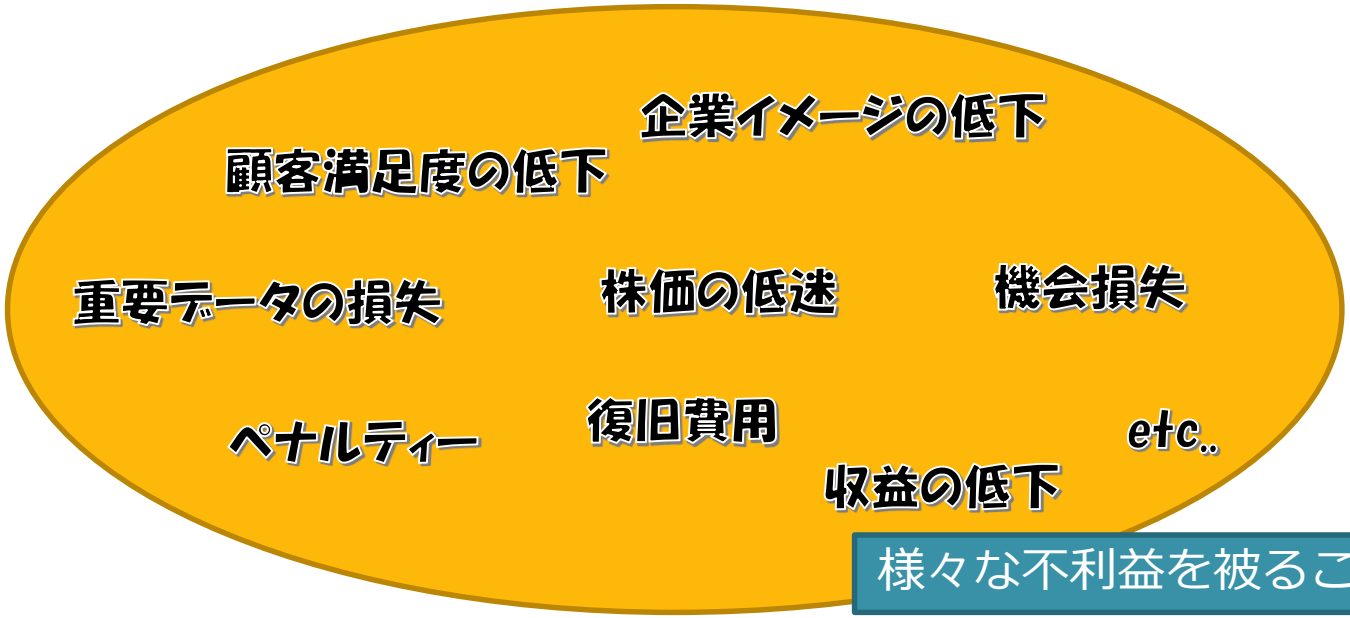
4.可用性、Pacemaker、PG-REXのまとめ

可用性とクラスタ

可用性とは

可用性(Availability)＝サービス継続性

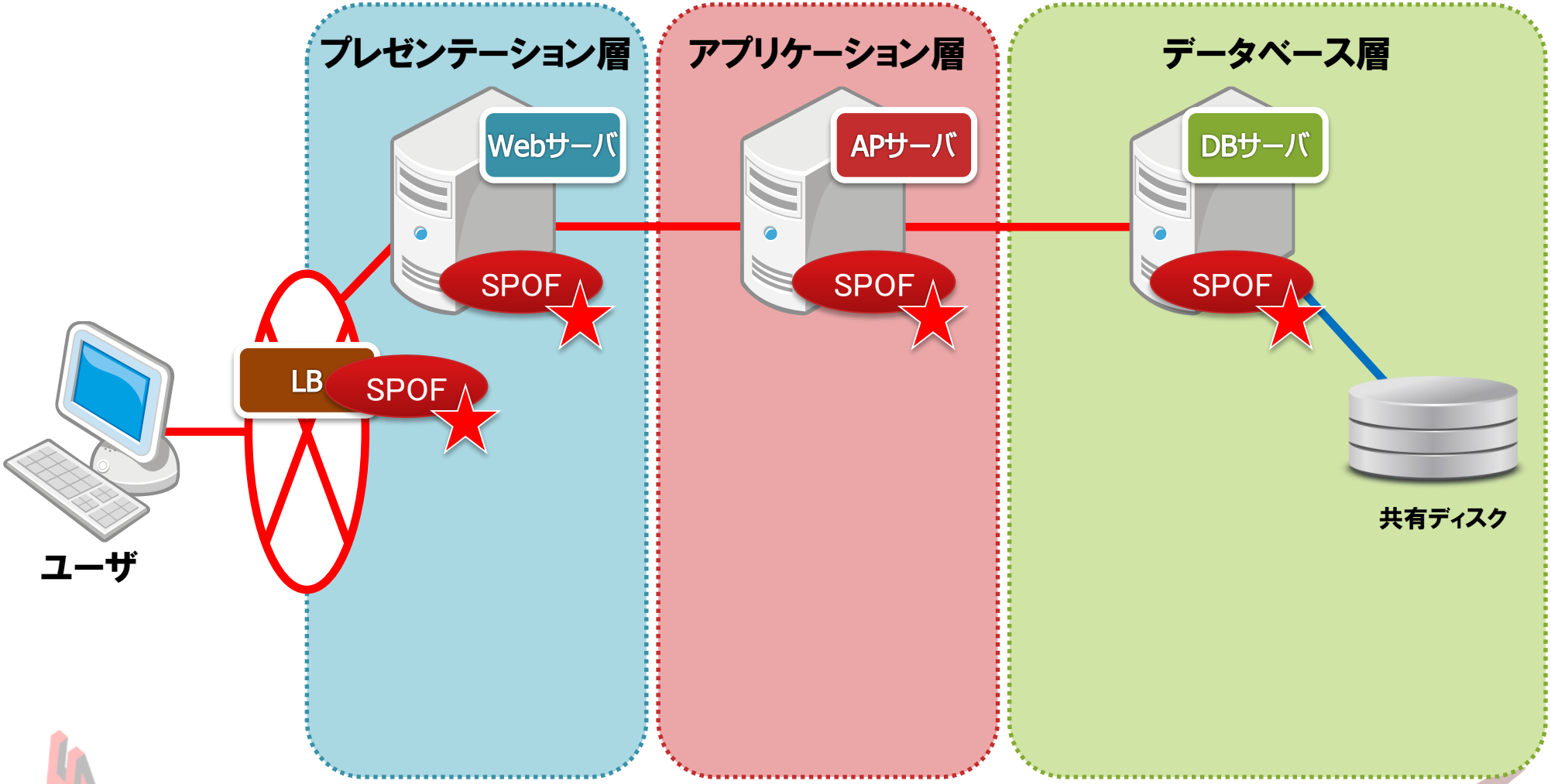
もしサービスが止まってしまったら？？



可用性を高め「止まらないサービス」の実現を目指す

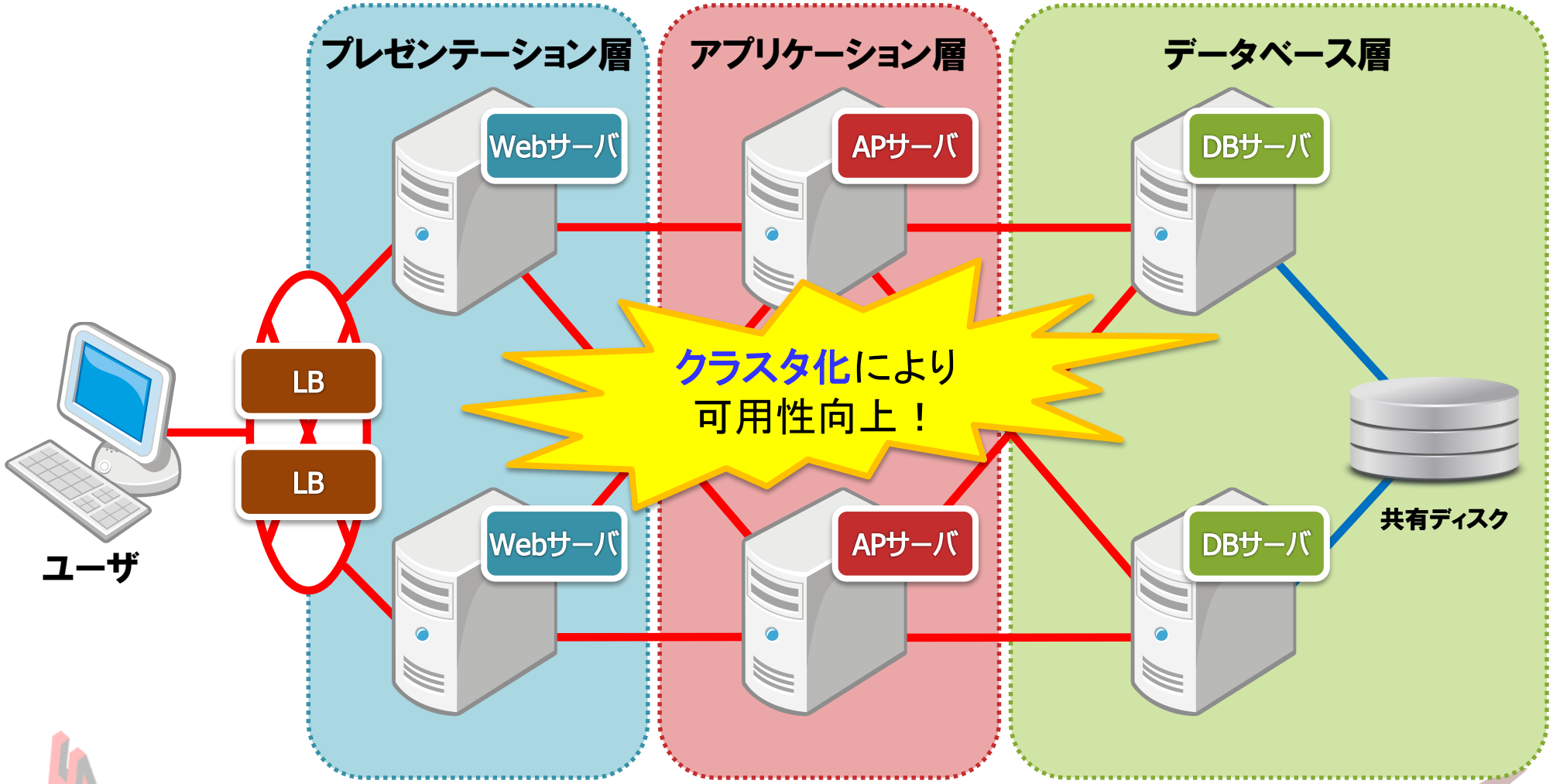
可用性の向上

可用性を高めるためには単一障害点(SPOF：Single Point of Failure ある一点が故障した場合にサービスが停止してしまう部位)の除去が重要



可用性の向上

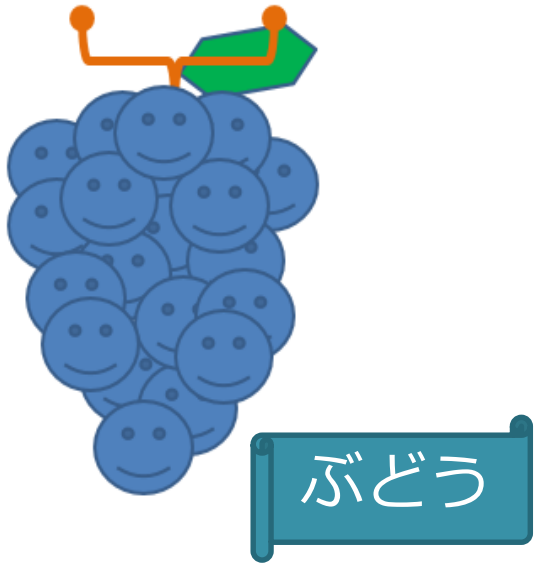
可用性を高めるためには単一障害点(SPOF：Single Point of Failure ある一点が故障した場合にサービスが停止してしまう部位)の除去が重要

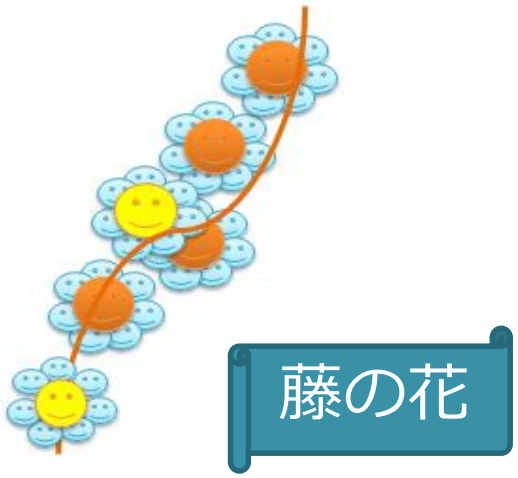


クラスタとは

ところで、クラスタとは？

- 語源は果実や花の房
- 同じようにまとまっているもののことを指す





- IT業界でいうクラスタとは、複数のコンピュータを連携させ、
  全体で一つのコンピュータのように振る舞わせる仕組みを指す

クラスタの分類

HA(High Availability：高可用)クラスタ

- ✓ 負荷分散クラスタ
  同一のサービスを提供する複数台のサーバに対して処理を分配する
  そのため、一台が停止しても残りのサーバでサービスを継続することができ、
  かつ一台のサーバでは得られなかった処理性能を確保することが可能
  主な適用対象：Webサーバ、APサーバ

- ✓ フェイルオーバクラスタ
  メインでサービスを提供するサーバ(稼働系,現用系,Activeなどと呼ばれる)と、
  それをバックアップするサーバ(待機系,予備系,Standbyなどと呼ばれる)で
  構成され、現用系に障害が発生した際に予備系でサービスを引き継ぐことで、
  サービスの停止時間を短くする
  主な適用対象：ロードバランサ(LB)、DBサーバ

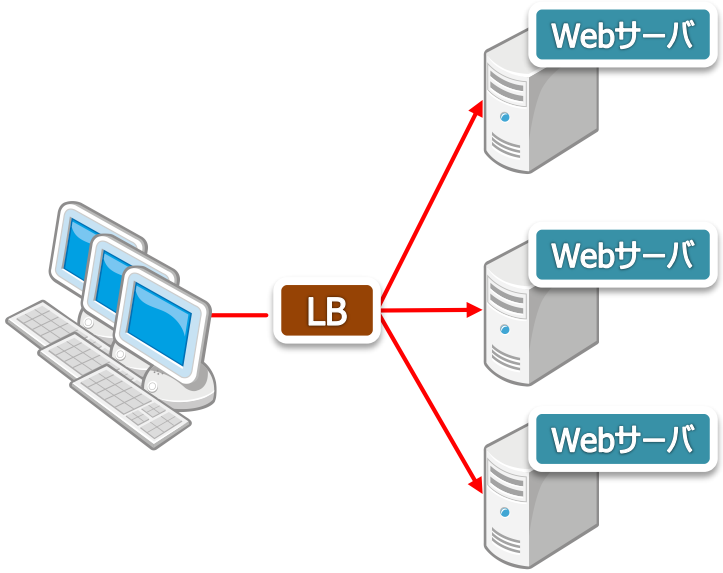
HPC(High Performance Computing Cluster)クラスタ

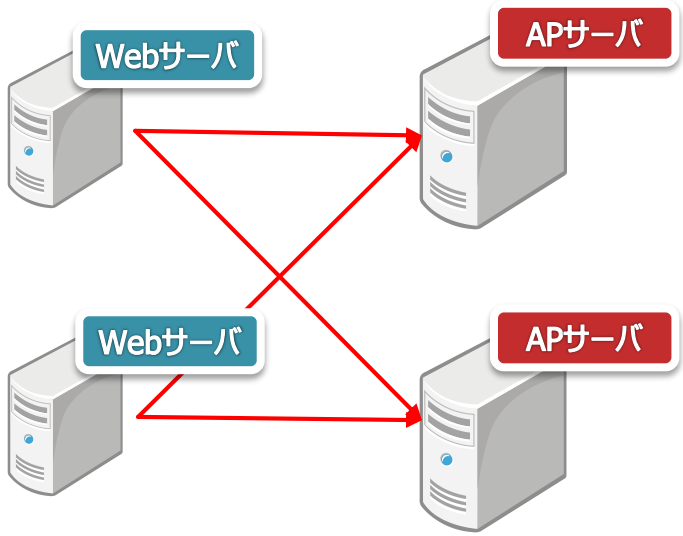
複数台のコンピュータを結合させて演算処理を分担し、
一台のコンピュータでは得られない高性能を確保することを目的とする

負荷分散クラスタ概要

□ 基本構成

- 同じサービスを提供する複数のサーバ群に対して処理を分散させ、1台あたりのサーバ負荷を低減させる
- LB(ロードバランサ)によるWebサーバへの負荷分散や、WebサーバによるAPサーバへの負荷分散処理等、複数のパターンがある
- 以降の説明ではLBによる負荷分散構成について説明する





負荷分散クラスタ概要

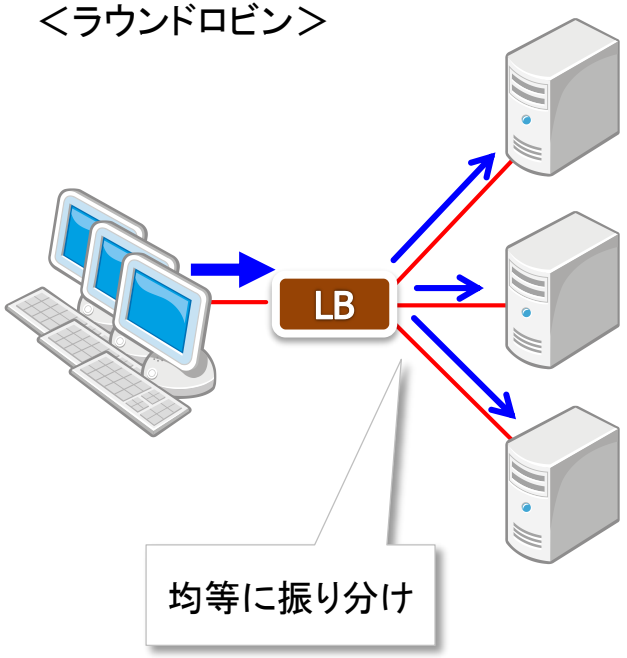
□ 主な機能

✓ 振り分け機能

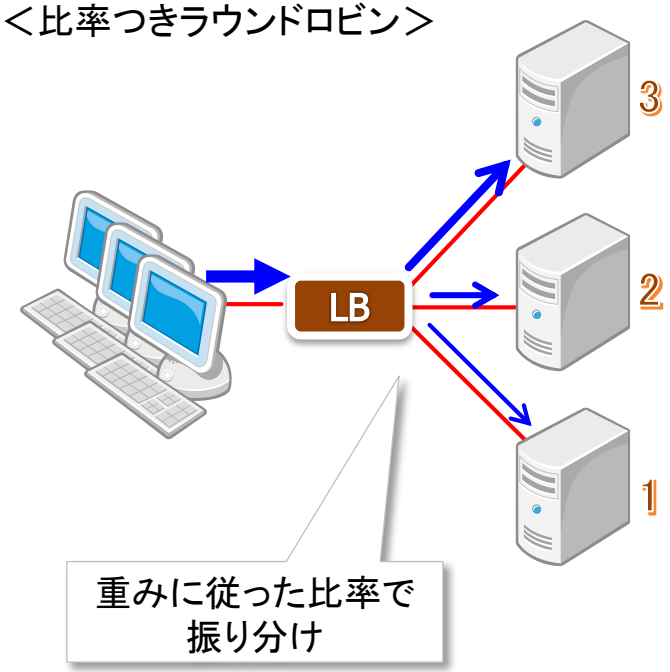
・クライアント(ユーザ)からの接続をバックエンドのサーバに振り分ける機能

・振り分けの方式にはラウンドロビン、比率つきラウンドロビン、リーストコネクションなどがある

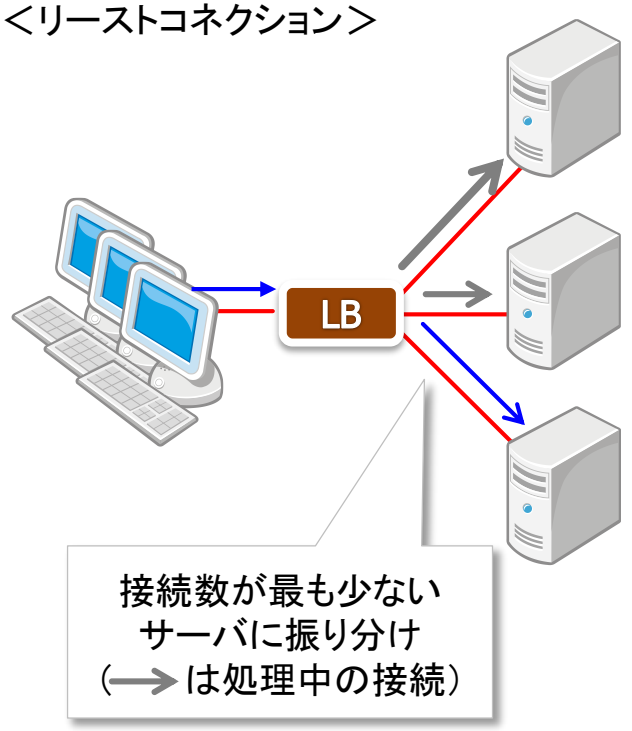
＜ラウンドロビン＞



＜比率つきラウンドロビン＞



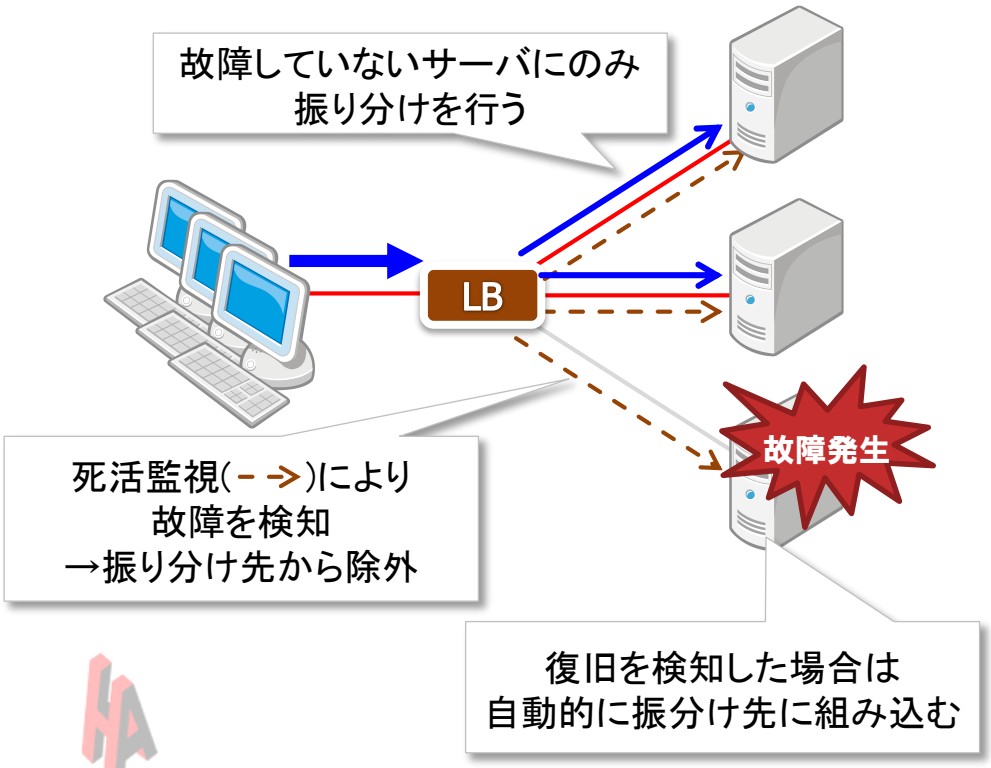
＜リーストコネクション＞



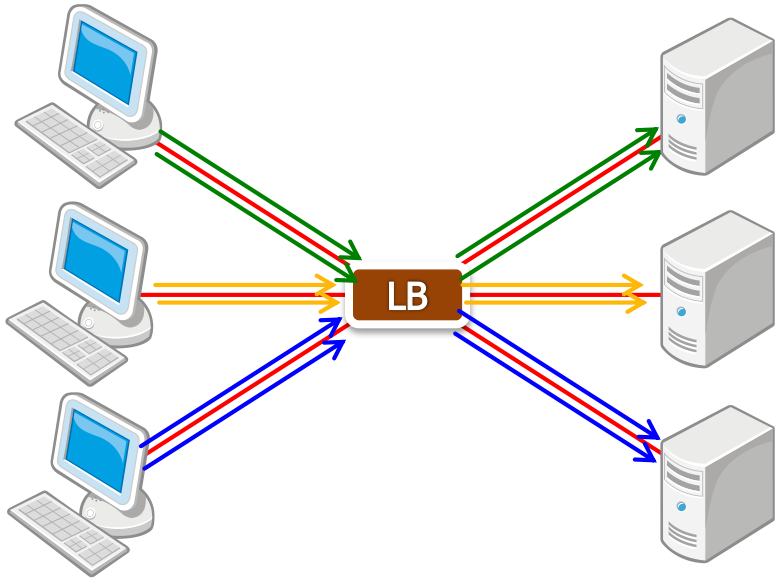
負荷分散クラスタ概要

- ✓ ヘルスモニタ
  - ・振分け先サーバの死活監視を行い、故障したサーバを振分け先から除外する
- ✓ セッション維持(スティッキー・セッション、パーシステンス)
  - ・同じクライアントからの接続は同じバックエンドサーバに接続するように制御する

＜ヘルスモニタイメージ＞



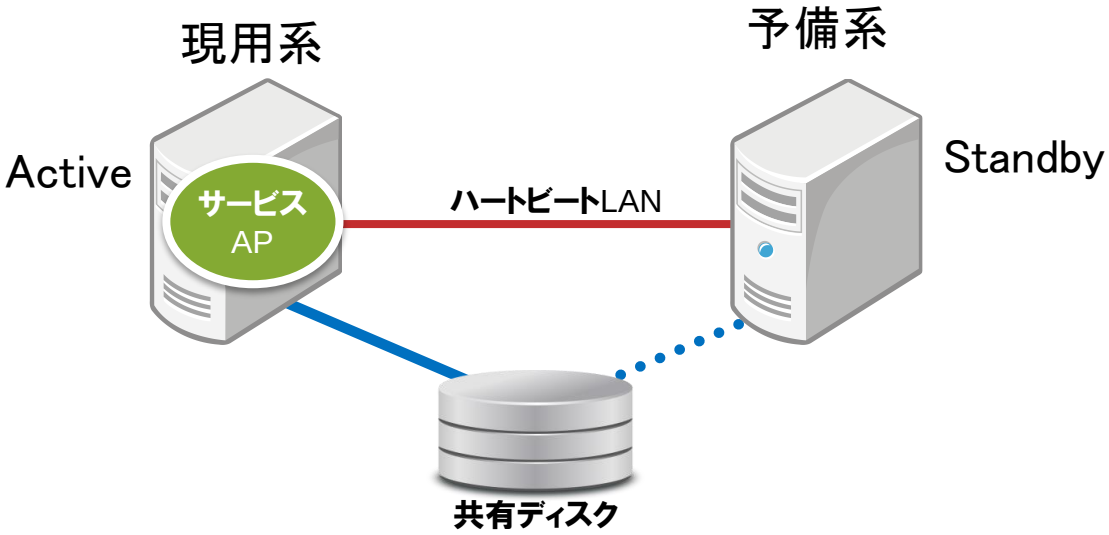
＜セッション維持イメージ＞



フェイルオーバクラスタ概要

基本構成

- サービス中のサーバに故障が発生した場合、他のサーバに処理を引き継ぐ
- サービスを提供する現用系サーバと待機状態にしておく予備系サーバから成るActive-Standby構成(Act-Sbyと書くことも)が一般的
- 同等の性能を有するサーバを2台配置し、サーバ間でデータの共有が必要な場合は共有ディスク※を使用



※ 共有ディスクを用いる代わりに、ソフトウェアの機能でデータを他のサーバへレプリケーションすることでデータを共有する構成もある(PostgreSQLを利用した構成を“PG-REX”と呼ぶ。後述)

フェイルオーバクラスタ概要

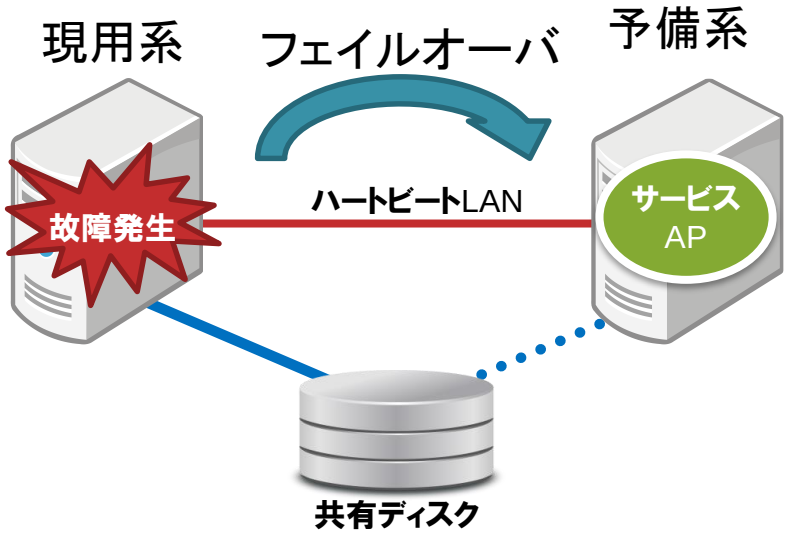
◘ 主な機能

✓ 故障検知とサービス引継ぎ

・常時ハードウェア及びサービスアプリケーションの稼動状態を監視
・現用系サーバ上で故障を検知した場合、
予備系でアプリケーションを起動してサービスを継続→「フェイルオーバ」という
・監視対象や故障検知の仕組みについては、クラスタソフトで異なる

✓ ノード監視

・クラスタ同士は定期的にハートビートと呼ばれる通信を行いお互いを監視
・ハートビートの応答が無くなった場合には、相手のサーバが故障したと判断し
フェイルオーバを実施

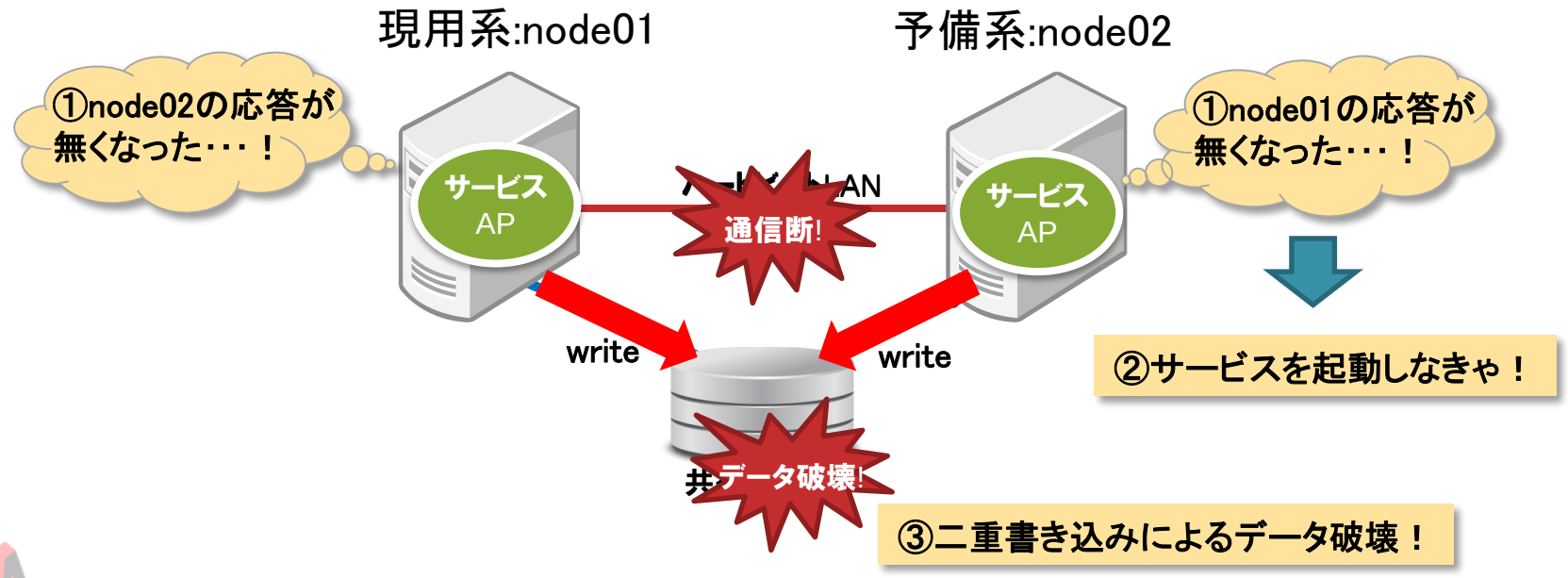


フェイルオーバクラスタ概要

- ✓ スプリットブレイン対策
  ハートビートの通信路が故障した場合には、両サーバでサービスが提供される
  →これを「スプリットブレイン」と呼ぶ

  スプリット・ブレインが発生すると、IPアドレスの重複や共有ディスクに対し両サーバから書き込みが行われデータが破壊される等の致命的な問題が発生
  →対策として排他制御機能が実装されている(詳細は次章)

＜データ破壊のイメージ＞



クラスタの分類

HA(High Availability：高可用)クラスタ

- ✓ 負荷分散クラスタ
  同一のサービスを提供する複数台のサーバに対して処理を分配する
  そのため、一台が停止しても残りのサーバでサービスを継続することができ、
  かつ一台のサーバでは得られなかった処理性能を確保することが可能
  主な適用対象：Webサーバ、APサーバ

- ✓ フェイルオーバクラスタ
  メインでサービスを提供するサーバ(稼働系,現用系,Activeなどと呼ばれる)と、
  それをバックアップするサーバ(待機系,予備系,Standbyなどと呼ばれる)で
  構成され、現用系に障害が発生した際に予備系でサービスを引き継ぐことで、
  サービスの停止時間を短くする
  主な適用対象：ロードバランサ、DBサ

一般に「HAクラスタ」というのはこちらを指す
本資料では以降こちらをHAクラスタと呼ぶ

HPC(High Performance Computing Cluster)クラスタ

複数台のコンピュータを結合させて演算処理を分担し、
一台のコンピュータでは得られない高性能を確保することを目的とする

次章からは「Pacemaker」を例に、
先に説明したHAクラスタ(フェイルオーバクラスタ)について
学習していきましょう！

Pacemakerとは

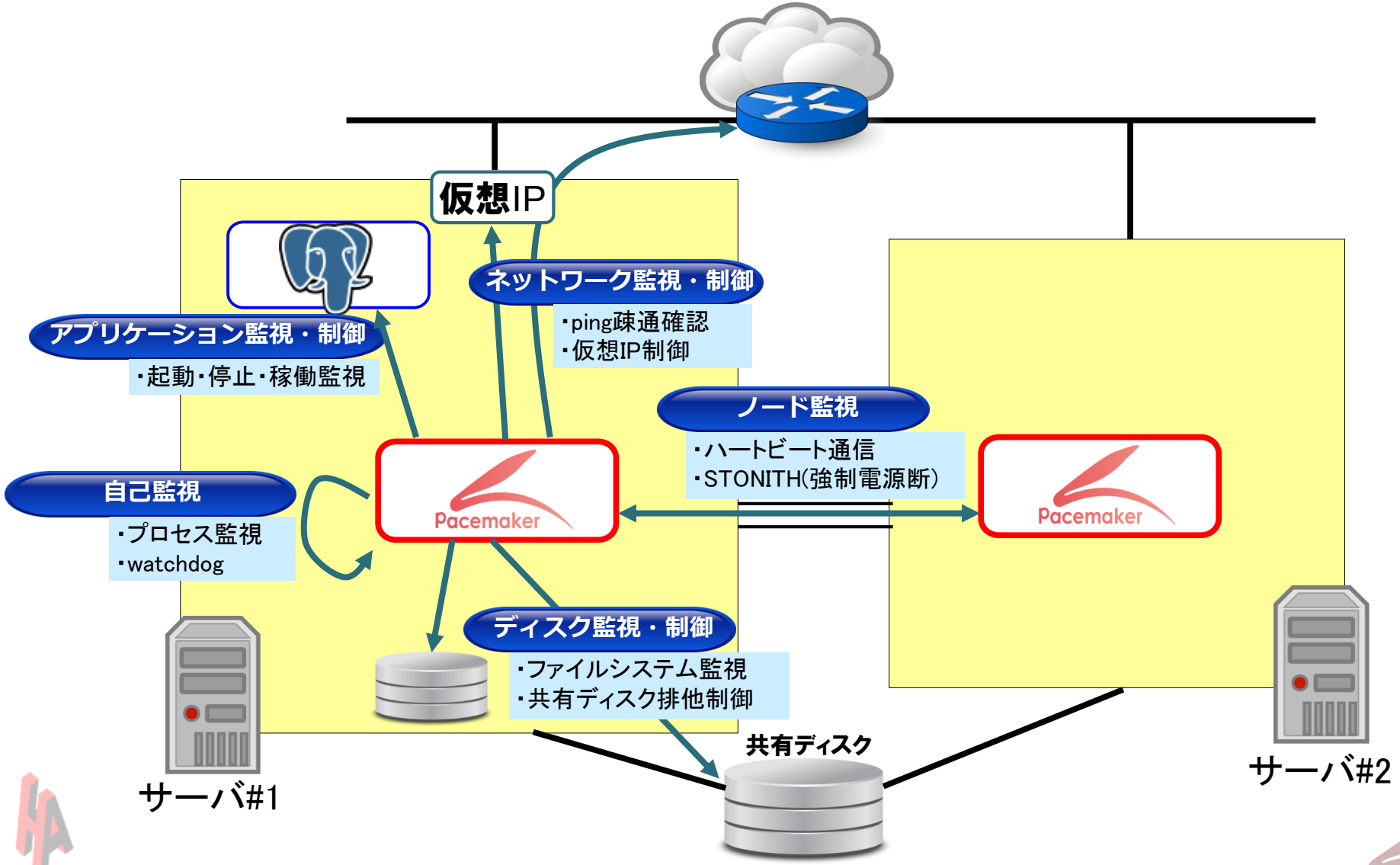
Pacemakerとは

Pacemaker(ぺーすめーかー)とはオープンソースで
開発されている高可用(High Availability)ソフトウェア

- Heartbeatの後継として開発されたソフトウェアであり、ソースはGitHubで管理
  - ClusterLabs：https://github.com/ClusterLabs
- バイナリファイルは以下より入手可能
  - Linux-HA Japanプロジェクト：http://linux-ha.osdn.jp/wp/

Pacemakerにできること

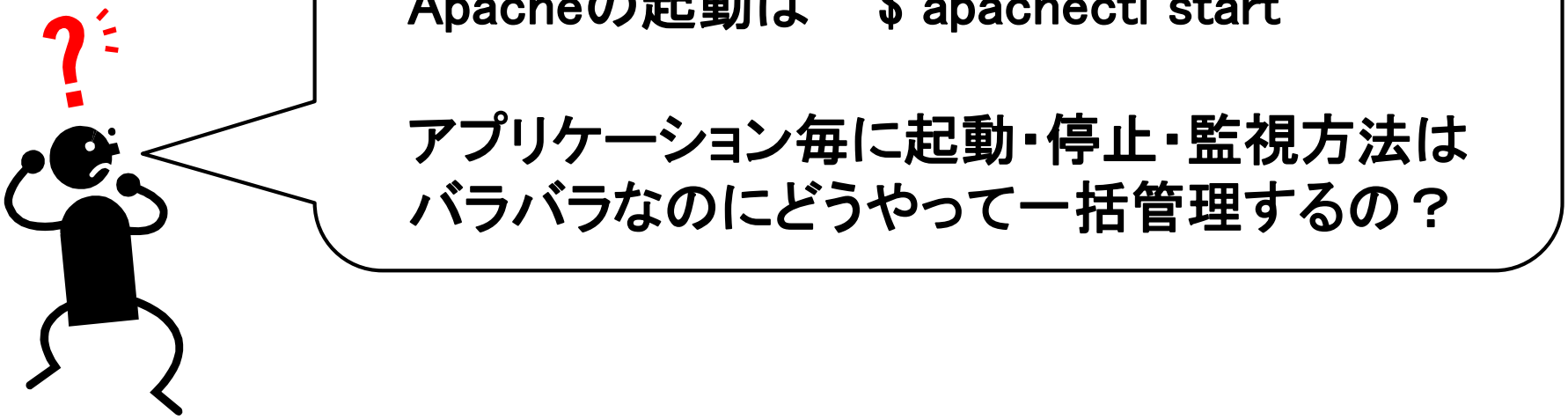
Pacemakerは様々な対象を起動/停止/監視することが可能



Pacemakerにできること

Pacemakerは様々な対象を起動/停止/監視することが可能　　・・・だけど

どうやって様々な種類のアプリケーション等を
監視・制御しているの？

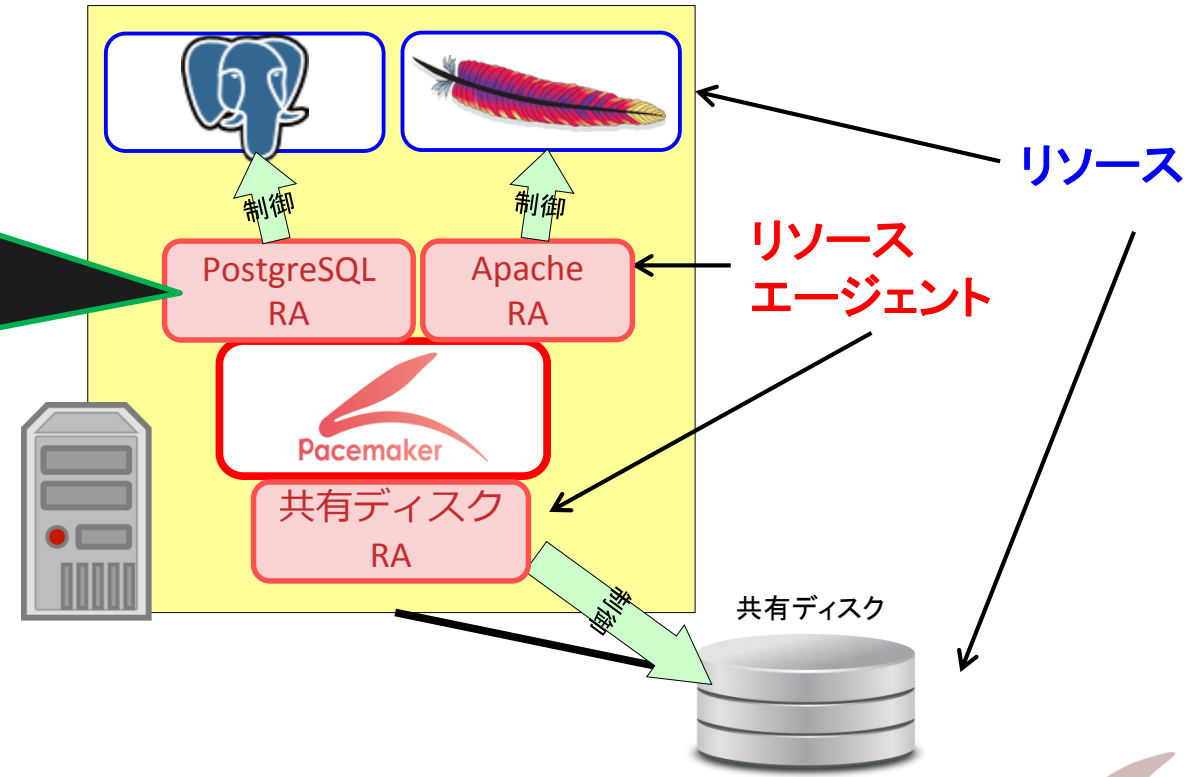


Pacemakerの「リソース」管理

- Pacemakerが起動/停止/監視する対象をリソースと呼ぶ
  例)Apache, PostgreSQL, 共有ディスク, 仮想IPアドレス　etc..
- リソース毎に操作方法などは異なるためPacemakerが様々なリソースを管理するための仲介者が必要→この役目をこなすのが「リソースエージェント(RA)」
  RAは各リソースの操作をラップしてPacemakerが制御できるようにしている
  標準で様々なRAが用意されており、自作することも可能(多くはシェルスクリプトで実装)

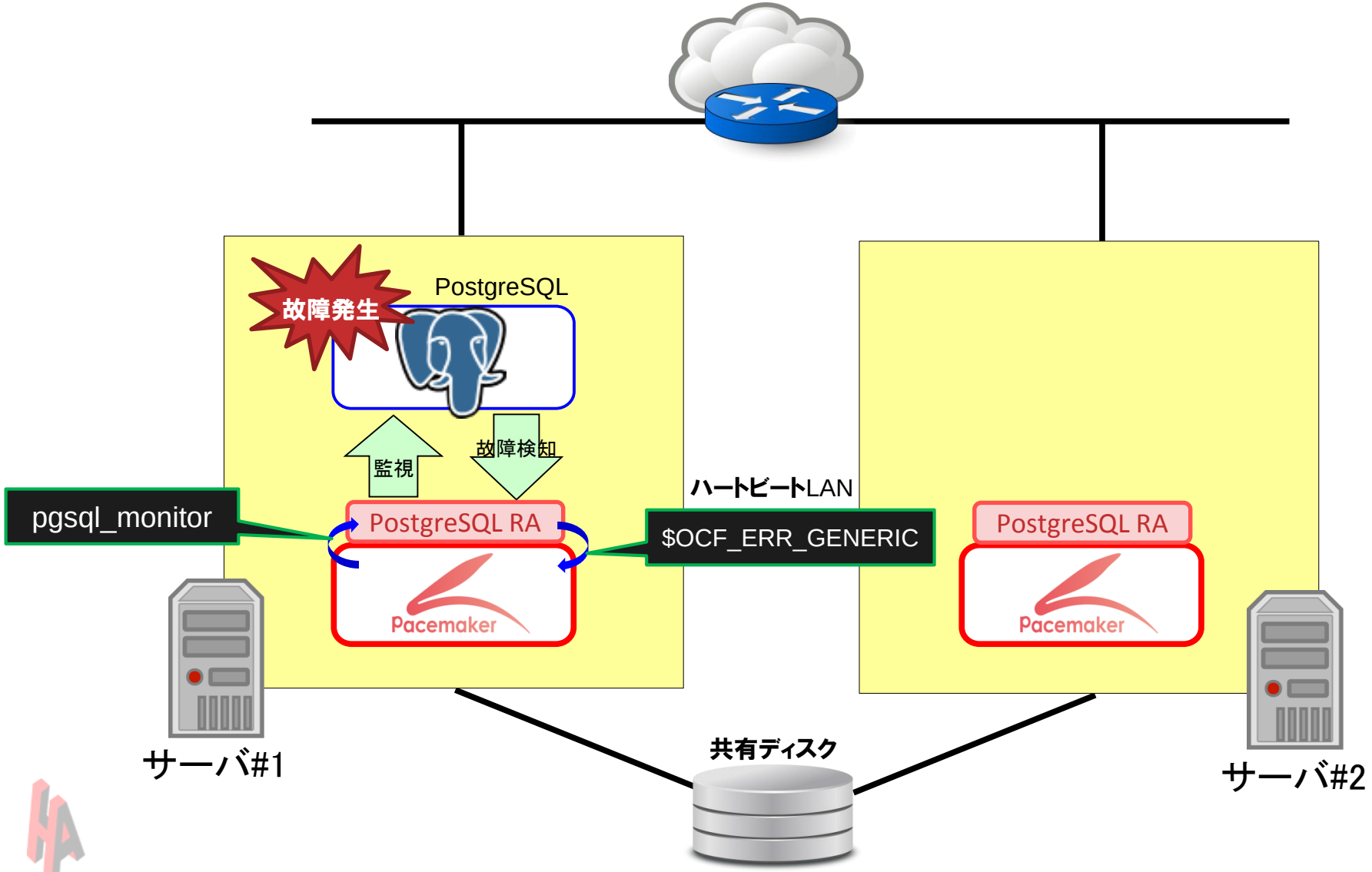
PostgreSQLの監視における
RA実行の具体的な流れ

PacemakerからRAの監視用関数
「pgsql_monitor」実行
↓
関数内部でDBに対して
”select now()”コマンド実行
成功なら”$OCF_SUCCESS”返却
↓
Pacemaker がRAの返り値を受領
”$OCF_SUCCESS”が返却された場合は
監視成功、それ以外は失敗と判定



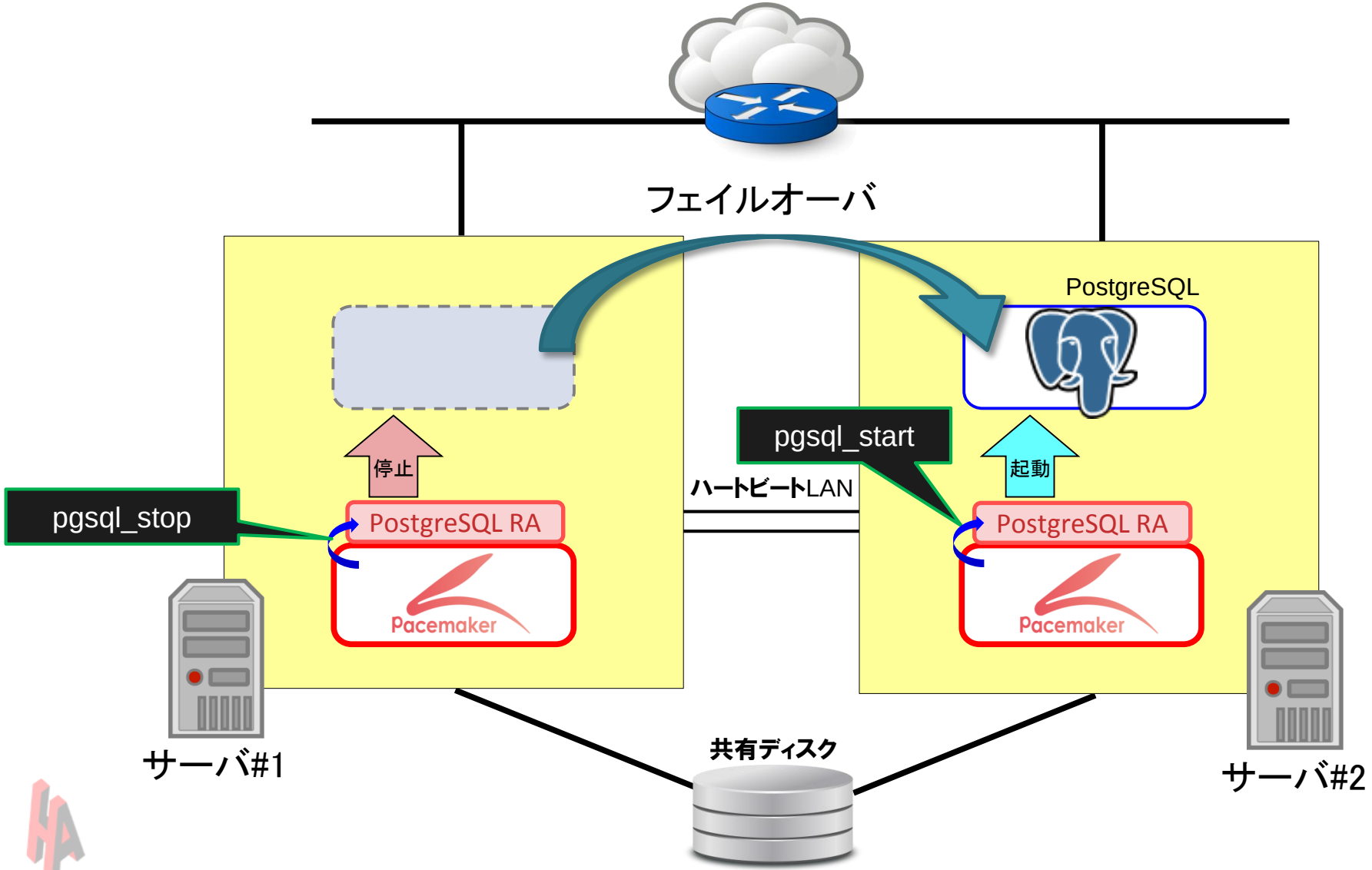
故障検知とサービス引継ぎ(フェイルオーバ)

<リソースの故障検知とフェイルオーバイメージ>



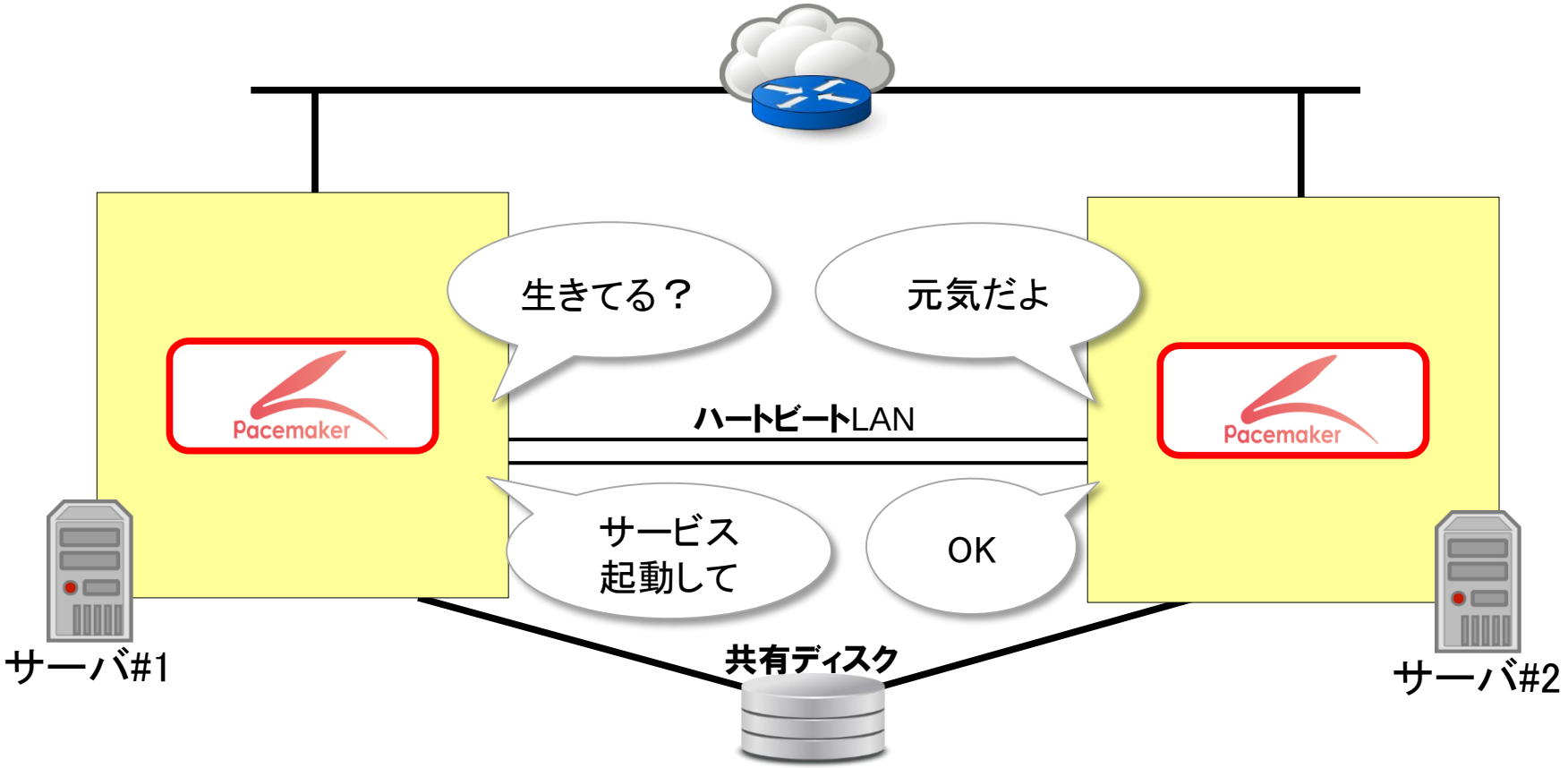
故障検知とサービス引継ぎ(フェイルオーバ)

＜リソースの故障検知とフェイルオーバイメージ＞



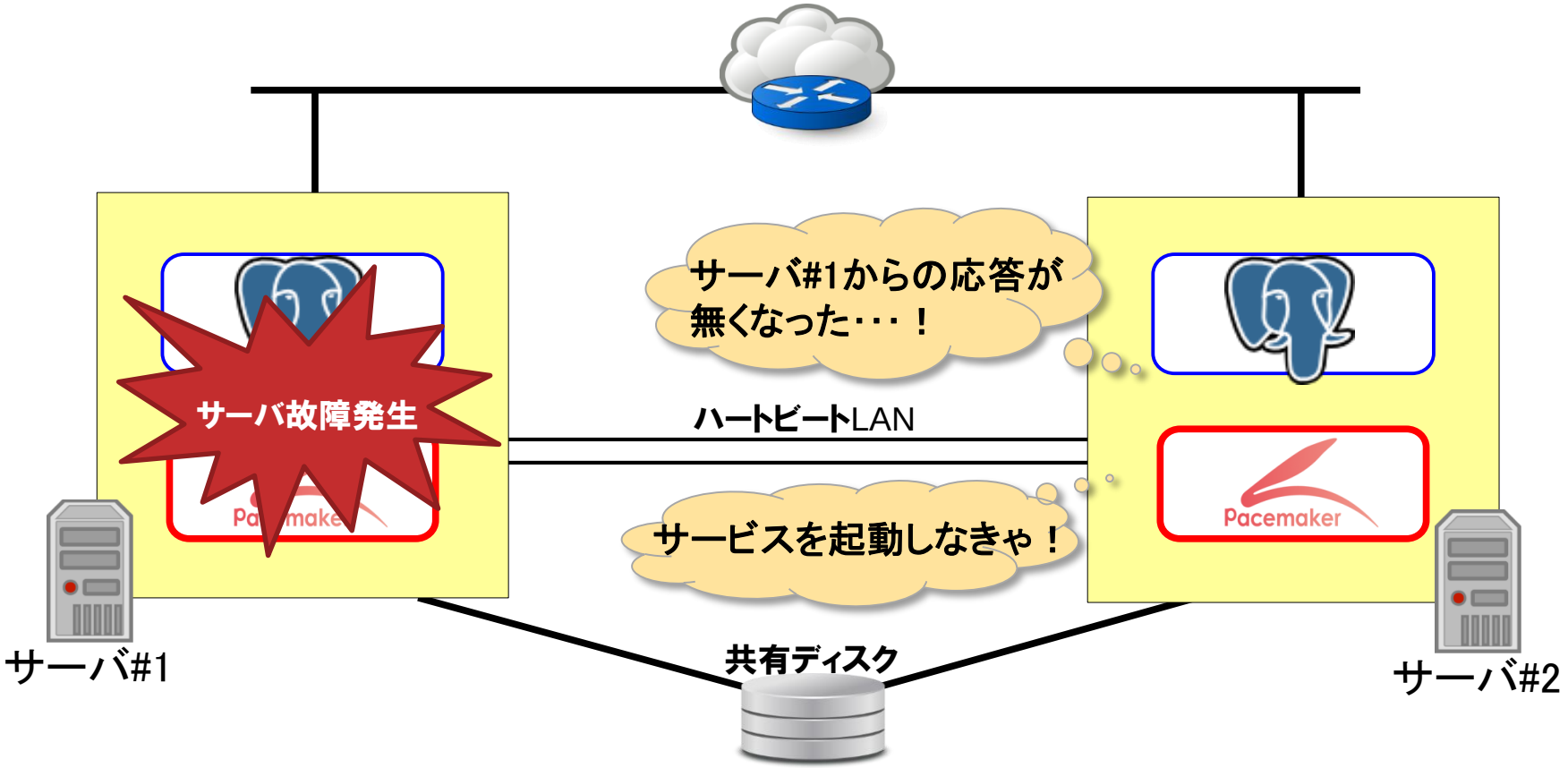
ノード監視

- ✓ ハートビート通信
  PacemakerはハートビートLAN(インターコネクトLANなどとも呼ばれる)を通して
  お互いの死活確認や情報交換を実施



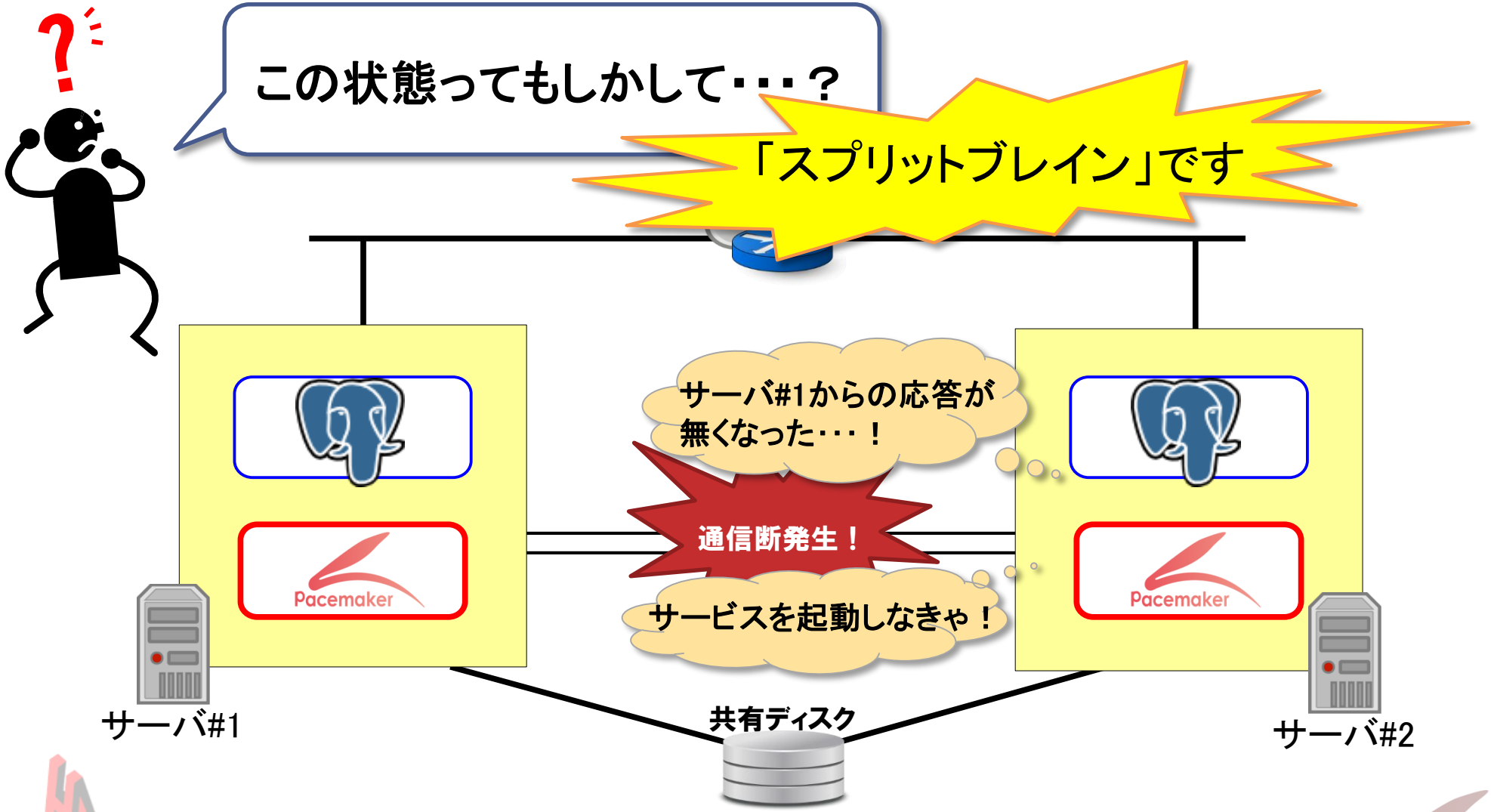
ノード監視

＜ノード故障におけるフェイルオーバイメージ＞



ノード監視

＜ハートビート通信故障の場合＞



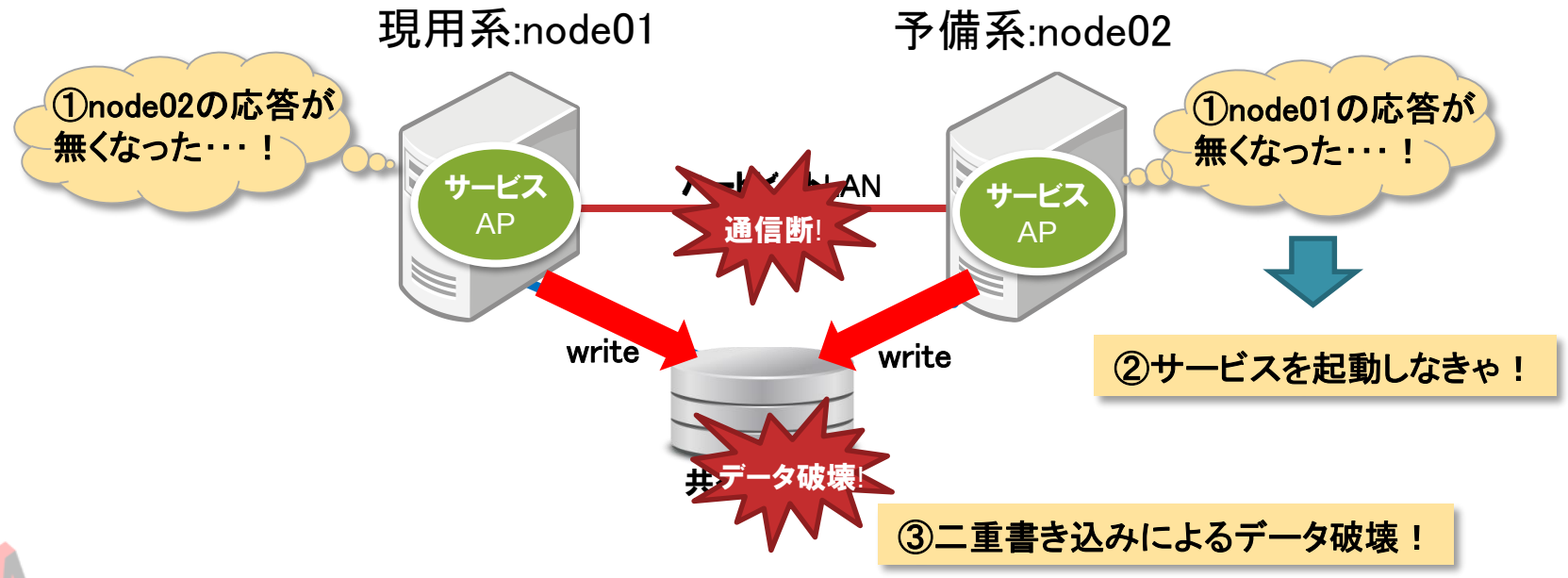
フェイHAクラスタラスタ概要

前ページの再掲

- ✓ スプリットブレイン対策
  ハートビートの通信路が故障した場合には、両サーバでサービスが提供される
  →これを「スプリットブレイン」と呼ぶ

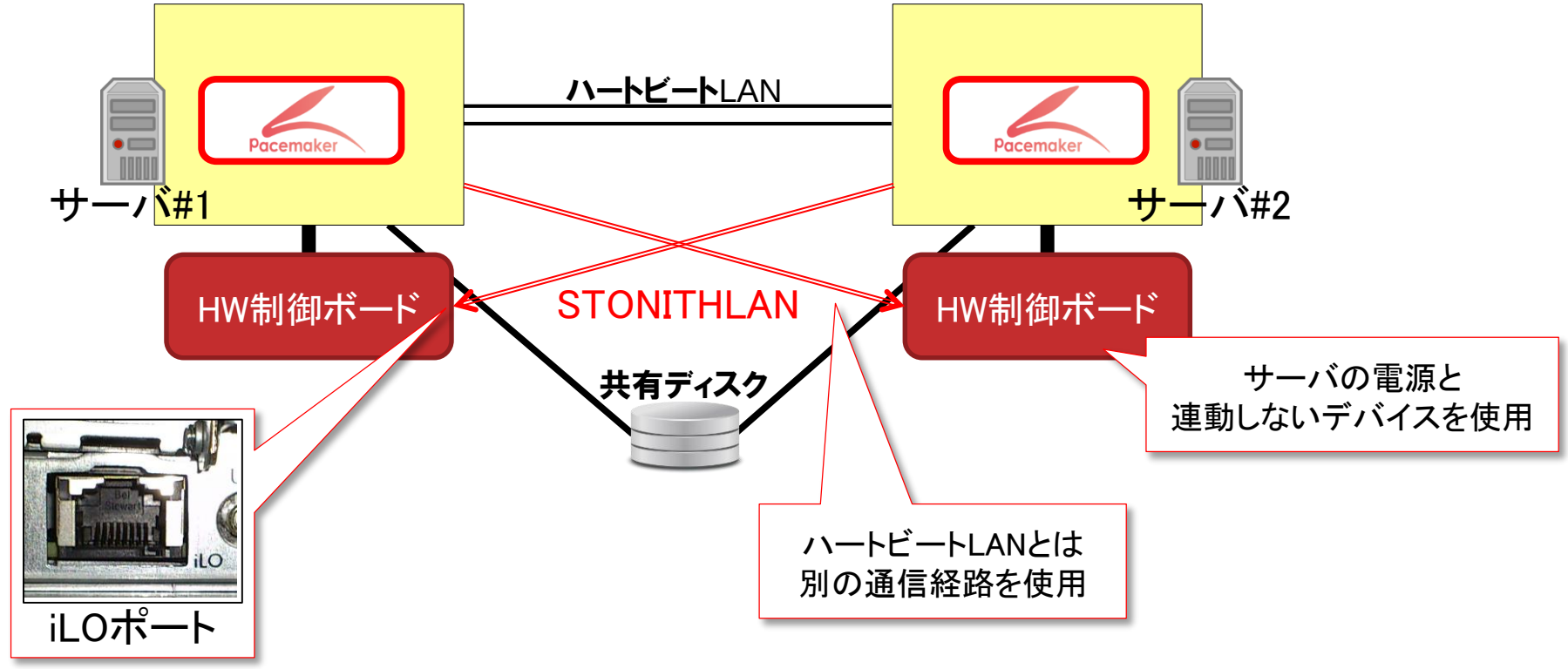
  スプリット・ブレインが発生すると、IPアドレスの重複や共有ディスクに対し両サーバから書き込みが行われデータが破壊される等の致命的な問題が発生
  →対策として排他制御機能が実装されている(詳細は次章)

＜データ破壊のイメージ＞



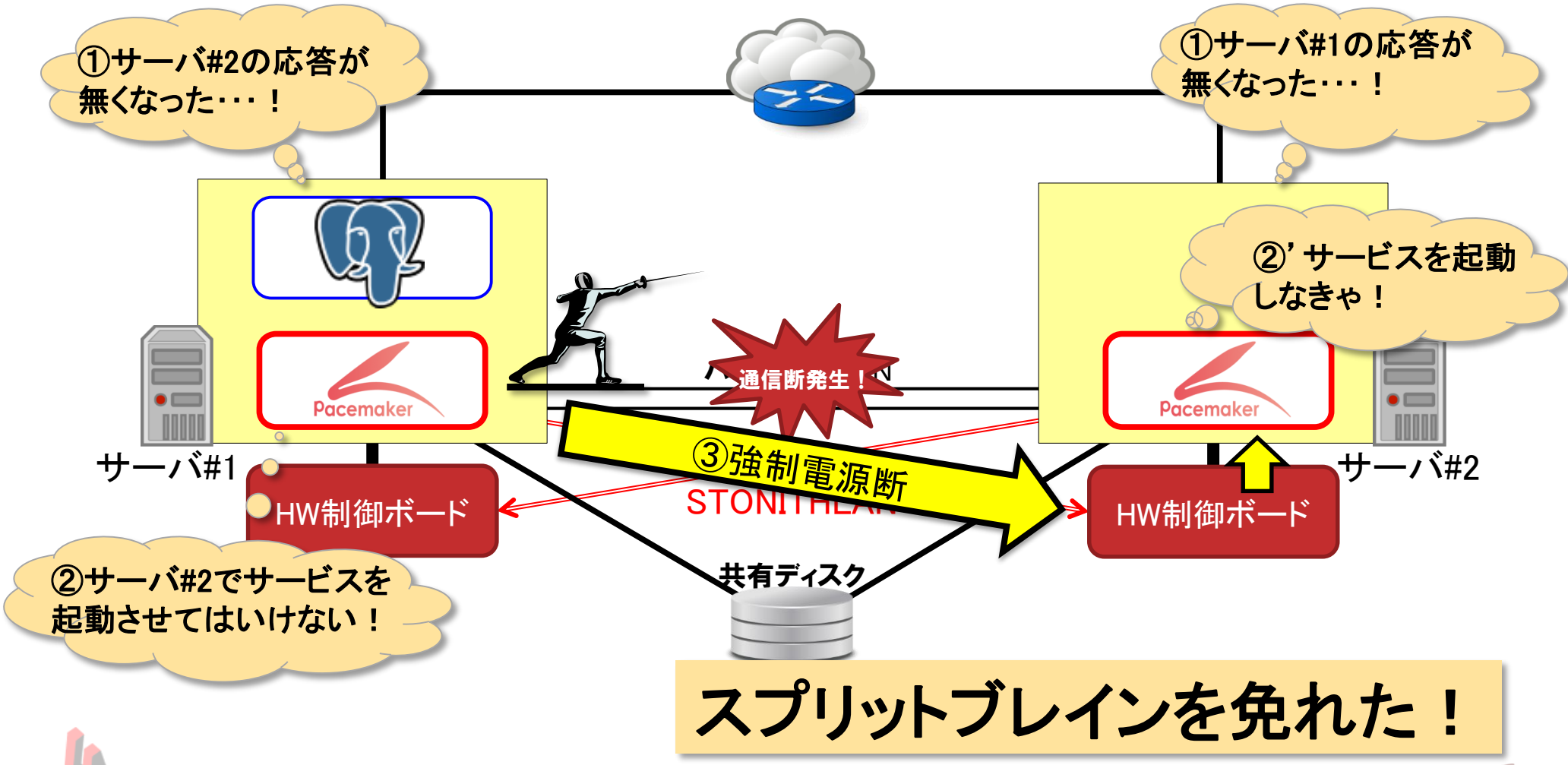
スプリットブレイン対策

- ✓ STONITH(Shoot The Other Node In The Head)
  ・スプリットブレイン(両系がActive状態)になる前に
  対向ノードの強制電源断を実行する機能(排他制御機能)
  ・サーバ付属のリモートHW制御ボード(iLOなど)を操作



スプリットブレイン対策

＜STONITHによる排他制御イメージ＞



スプリットブレインを免れた！

【補足】STONITHが使えない環境の場合

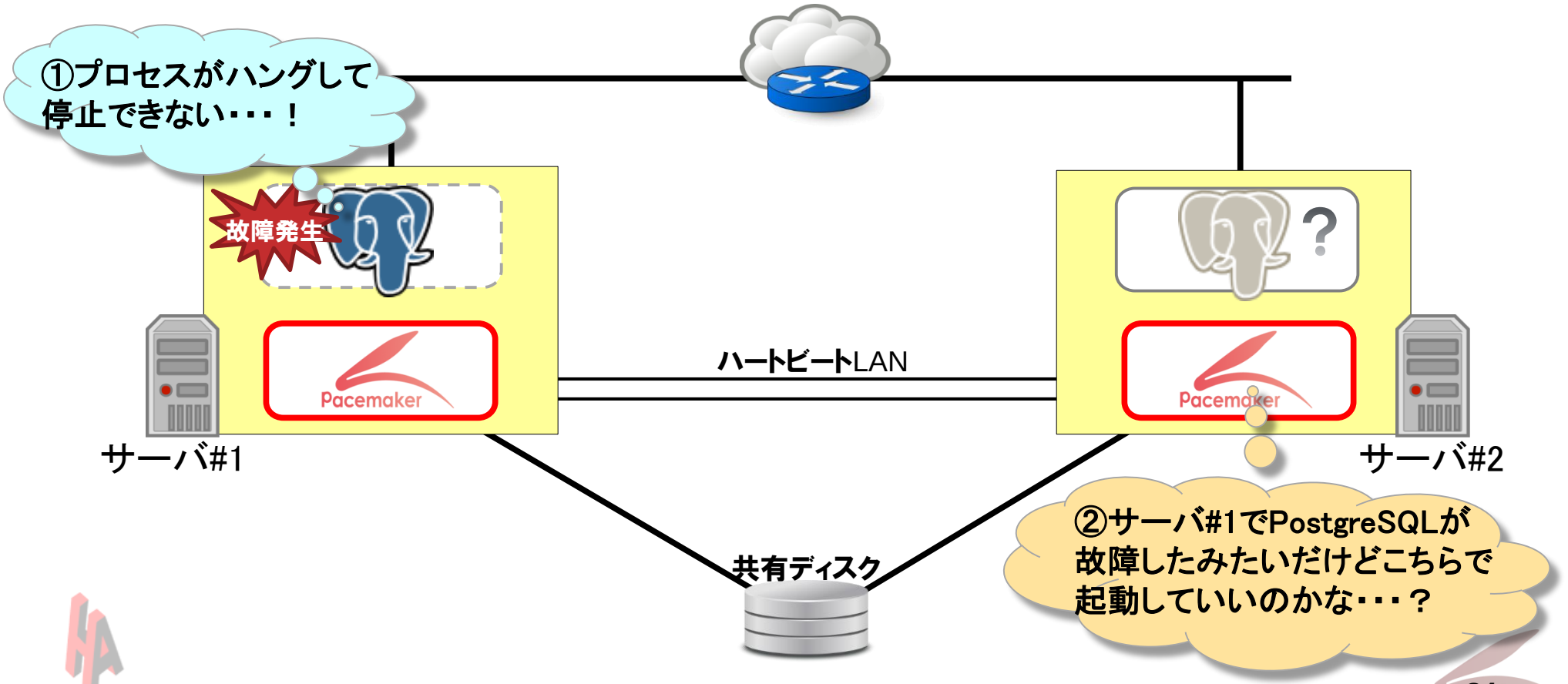
HW制御ボードなどが存在せずSTONITHが使用できない場合は
以下のような機能によりスプリットブレイン対策を行うことが可能

- ✓ sfex
  - ・共有ディスクのsfex専用パーティションに、ディスクのロック情報を定期的に書き込む
  - ・Active系によりロック情報が更新されていれば、Active系が生存していると判断し、Standby系でのリソース起動を抑止
  - ・STONITHが使用できる環境においても、信頼性を高めるために本機能を併用することがある

- ✓ VIPcheck
  - ・Standby系からActive系の仮想IP(VIP)に対してpingを送信
  - ・ping応答があれば、Active系が生存していると判断し、Standby系でのリソース起動を抑止

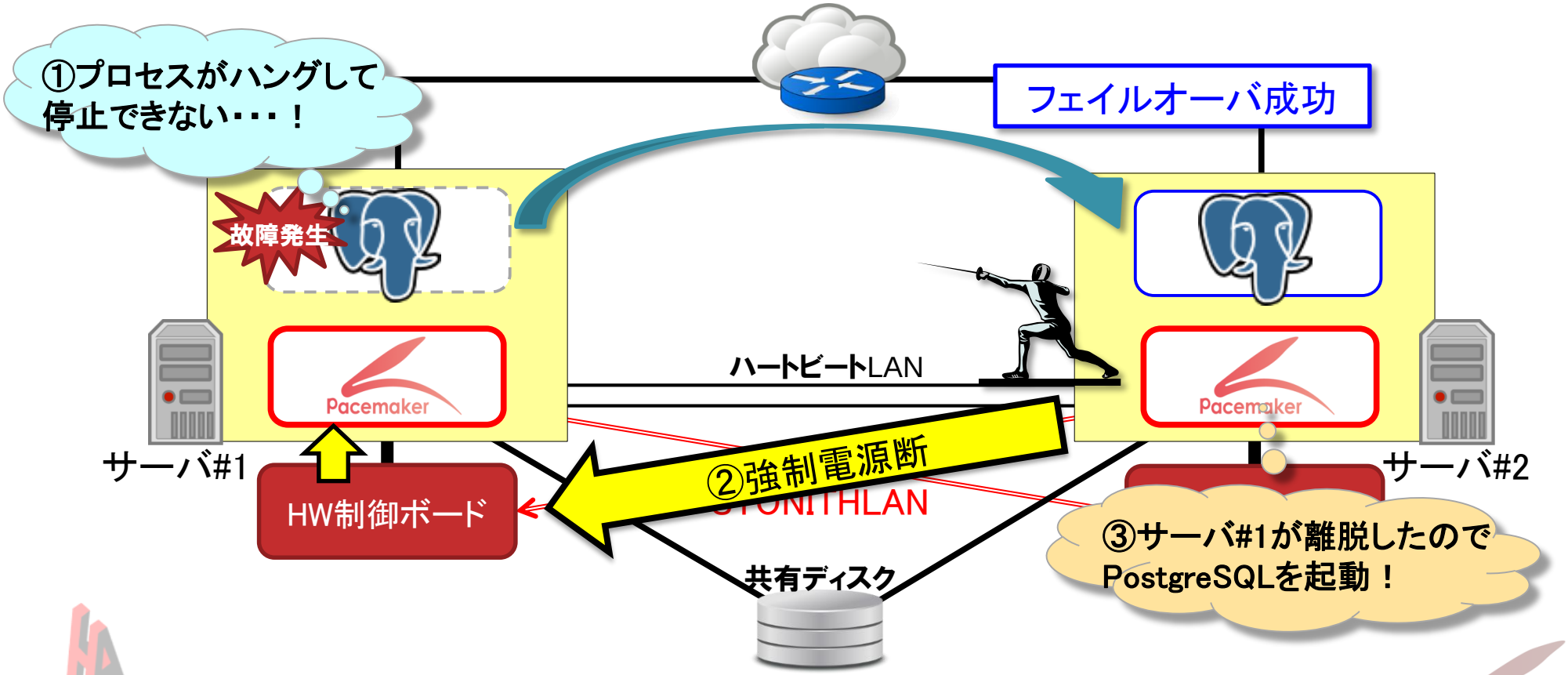
【補足】リソース停止失敗

- HAクラスタにおける致命的な障害として、スプリットブレイン以外に「リソース停止失敗」がある
- 停止に失敗したリソースの状態は不定(本当に止まっているのかどうかわからない)となるため、最悪プロセスが両系で起動してしまう



【補足】リソース停止失敗

STONITHはリソースが停止失敗したノードを強制電源断することで
クラスタから離脱させ、フェイルオーバを継続させることが可能



【補足】障害パターンとサービス継続性

✓ 各排他制御機能と障害発生時のサービス継続可否の対応
○:サービス継続可能
×:サービス継続不可能

障害パターン	排他制御なし	STONITH	sfex	VIPcheck
スプリットブレイン	×	○	○	○
リソース停止失敗	×	○	×	×

STONITHは致命的な障害に対するカバー範囲が最も広いため
環境が許せばSTONITHの利用を推奨

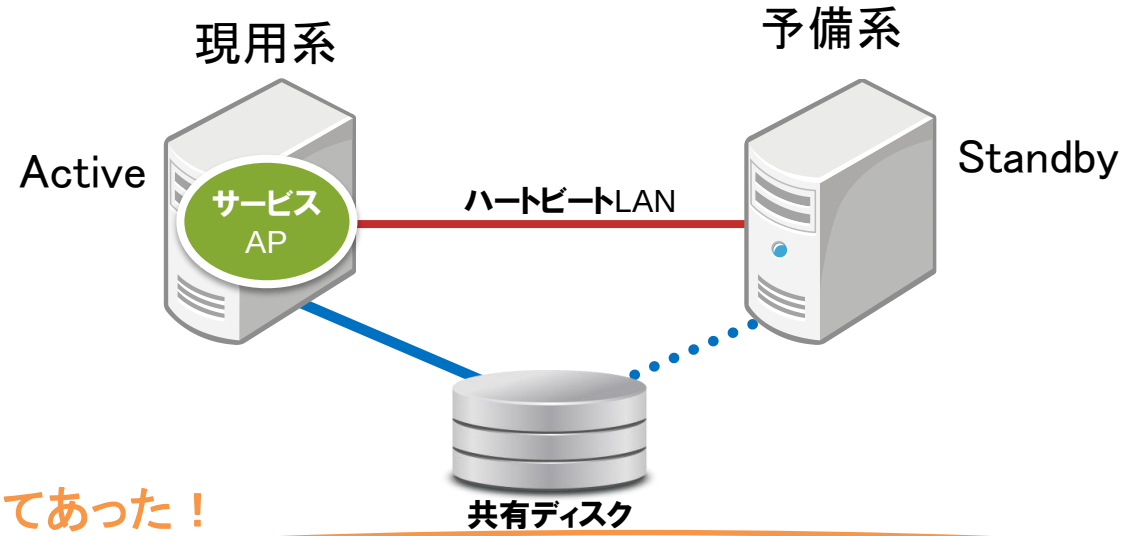
PG-REXとは

フェイHAクラスタラスタ概要

前ページの再掲

□ 基本構成

- サービス中のサーバに故障が発生した場合、他のサーバに処理を引き継ぐ
- サービスを提供する現用系サーバと待機状態にしておく予備系サーバから成るActive-Standby構成(Act-Sbyと書くことも)が一般的
- 同等の性能を有するサーバを2台配置し、サーバ間でデータの共有が必要な場合は共有ディスク※を使用

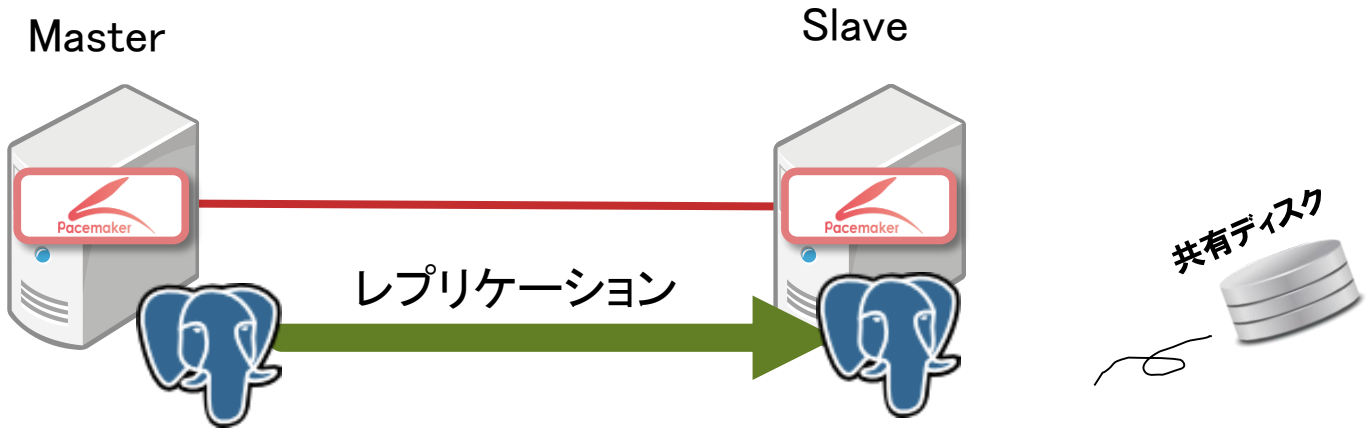


ココに書いてあった！

※ 共有ディスクを用いる代わりに、ソフトウェアの機能でデータを他のサーバへレプリケーションすることでデータを共有する構成もある(PostgreSQLを利用した構成を“PG-REX”と呼ぶ。後述)

PG-REXとは

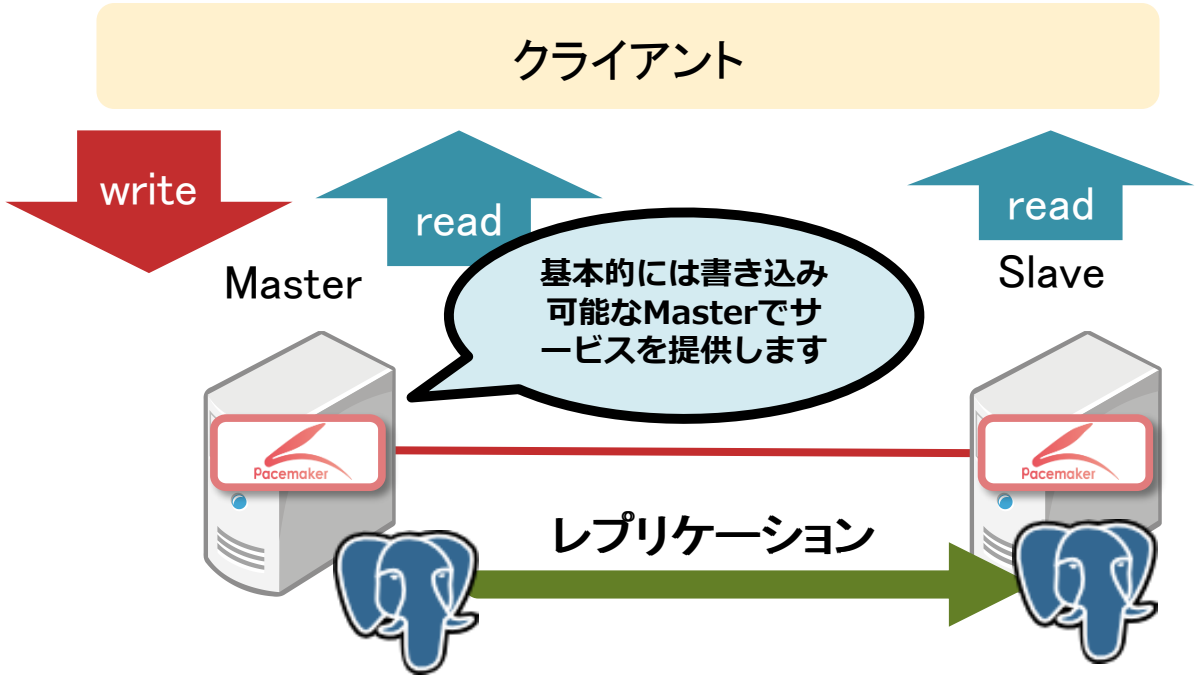
PG-REX(ぴーじーれっくす)とは
PostgreSQLのレプリケーション機能とPacemakerを
組み合わせた高可用性ソリューション



- PostgreSQLのデータは各サーバのローカルディスク上に配置し、PostgreSQLのレプリケーション機能を用いて共有する。そのため共有ディスクが不要。

PG-REXにできること

- ✓ ここまでに説明した”Pacemakerにできること”と同等のことが可能
- ✓ データ共有に共有ディスクを使用する共有ディスク型構成では予備系でPostgreSQLが起動していないのに対し、PG-REXは予備系でもレプリケーションのためにPostgreSQLが起動している。前者はAct-Sby構成と呼称され、後者はMaster-Slave構成と呼ばれる
- ✓ PacemakerはAct-Sbyの状態遷移と同様にMaster-Slaveの状態遷移を制御する
- ✓ データの一貫性を保つため、Masterは書き込み-読み込みを提供するのに対し、Slaveは読み込みだけを提供する



PG-REXと共有ディスク型の比較

	PG-REX	共有ディスク型	理由
コスト	win	lose	共有ディスクは高価
サービス継続性	win	lose	PG-REXの方がフェイルオーバが早い
DB性能	lose	win	レプリケーションのオーバヘッドが影響
実績	lose	win	PG-REXの方が後発

参考資料

もっとPG-REXについて知りたい方へ

✓オープンソースカンファレンス2017 Kyoto
「試して覚えるPacemaker入門 PG-REX(Pacemaker + PostgreSQLによるシェアードナッシングHA構成)構築」
http://linux-ha.osdn.jp/wp/archives/4627

✓オープンソースカンファレンス2018 Osaka
「試して覚えるPacemaker入門 PG-REX(Pacemaker + PostgreSQLによるシェアードナッシングHA構成)運用」
http://linux-ha.osdn.jp/wp/archives/4664

クラスタ、Pacemaker、PG-REXのまとめ

まとめ

- ✓ 可用性とクラスタ
  - ・可用性を向上させるためにクラスタを導入しSPOFを除去
  - ・「負荷分散クラスタ」の主な機能として振り分け、監視、セッション維持がある
  - ・「HAクラスタ(フェイルオーバクラスタ)」の主な機能として故障検知とサービス引継ぎ、ノード監視、スプリットブレイン対策がある

- ✓ PacemakerによるHAクラスタ
  - ・HAクラスタソフトであるPacemakerでは様々なリソースを起動/停止/監視することが可能
  - ・リソースはリソースエージェント(RA)で管理
  - ・スプリットブレイン対策としてSTONITHが使用可能

- ✓ PG-REXについて
  - ・PG-REXはPostgreSQLのレプリケーション機能とPacemakerを組み合わせた高可用性ソリューション
  - ・データの共有に共有ディスクを必要としないため、共有ディスク型に比較し低コストだが、DB性能はレプリケーションの分低下する

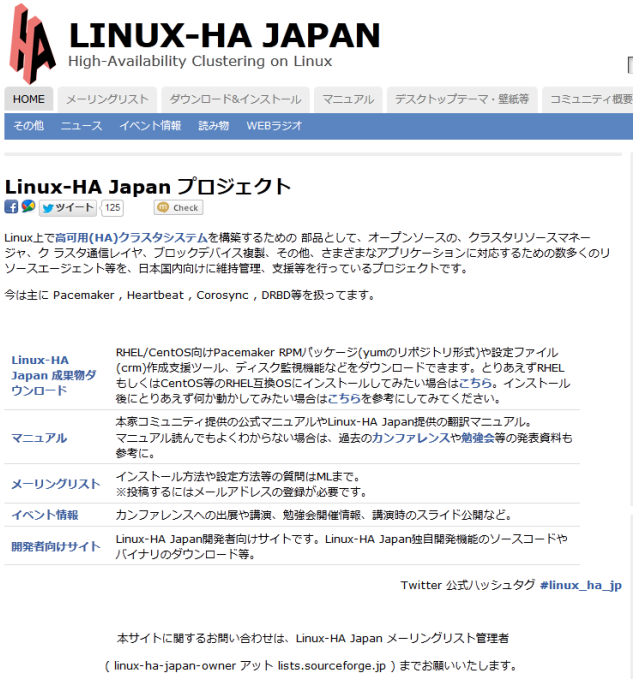
コミュニティ紹介

コミュニティ紹介

Linux-HA Japan URL

http://linux-ha.osdn.jp/

https://ja.osdn.net/projects/linux-ha/



Pacemaker関連の最新情報を日本語で発信

Pacemakerのダウンロードもこちらからどうぞ
(インストールが楽なリポジトリパッケージを公開しています)

コミュニティ紹介

日本におけるHAクラスタについての活発な意見交換の場として
「Linux-HA Japan日本語メーリングリスト」も開設しています

Linux-HA-Japan MLでは、Pacemaker、Heartbeat3、Corosync
DRBDなど、HAクラスタに関連する話題は歓迎！

- ML登録用URL
  http://linux-ha.osdn.jp/
  の「メーリングリスト」をクリック

- MLアドレス

linux-ha-japan@lists.osdn.me

※スパム防止のために、登録者以外の投稿は許可制です

コミュニティ紹介

PG-REX URL

https://ja.osdn.net/projects/pg-rex/

PG-REXの構築手順書やpm_crmgen_env.xls、PG-REX運用補助ツールを提供しています

本セミナーでは説明できなかった詳細内容も上記手順書に詳しく書いてあるので是非読んでください！

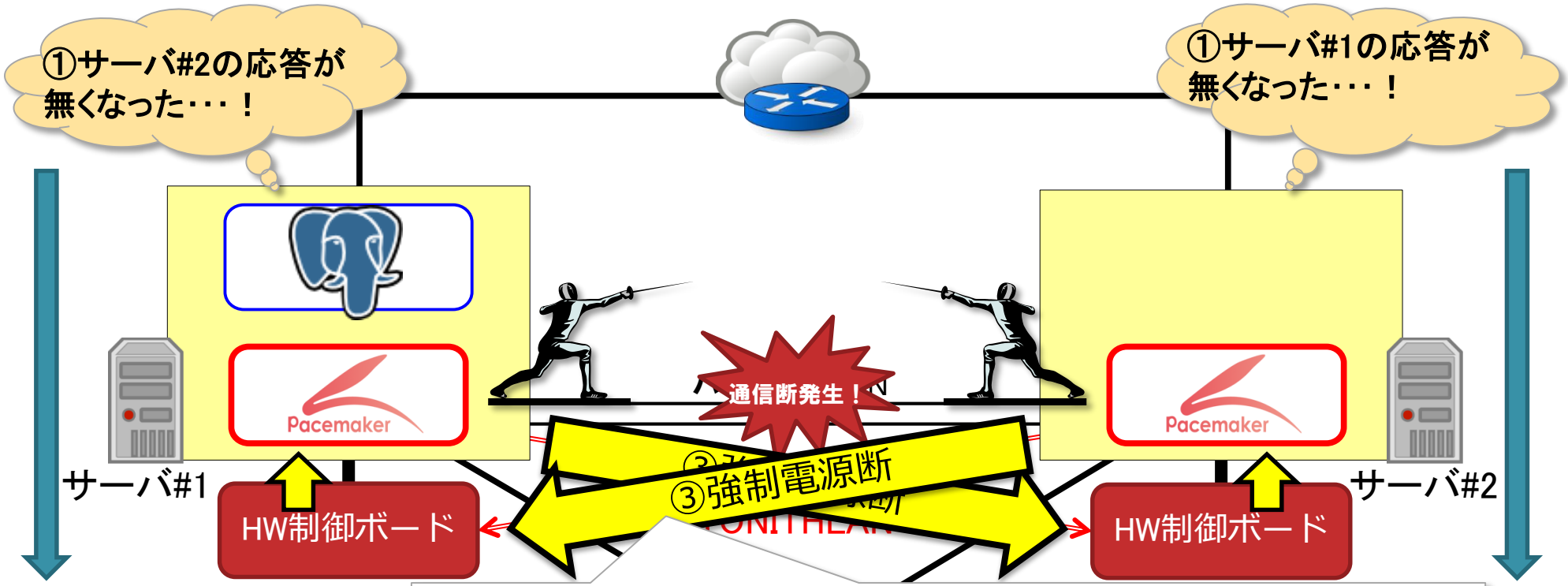
ご清聴ありがとうございました

付録

stonith-helper

STONITHの相撃ち

＜STONITHの相撃ちイメージ＞



②サーバ#2の状態がわから
強制的に電源を落とそ
（サーバ#2でのサービス起動

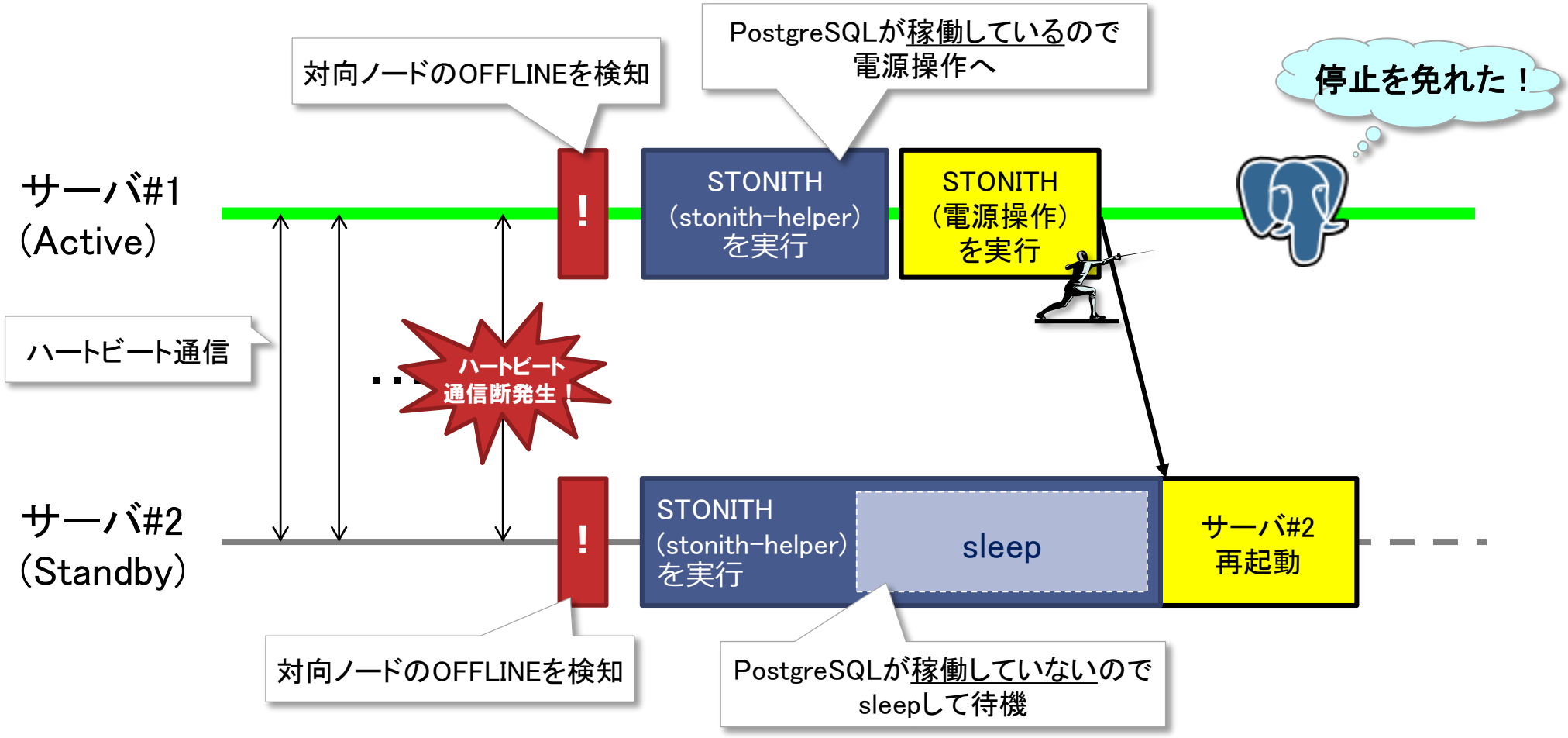
から

止）

実際にはサーバ#2も対向ノードを強制電源断しようとするため、
STONITHの”相撃ち”が発生する可能性がある
→ 「stonith-helper」というプラグインを利用することで
相撃ちを阻止することができる

STONITHの相撃ち

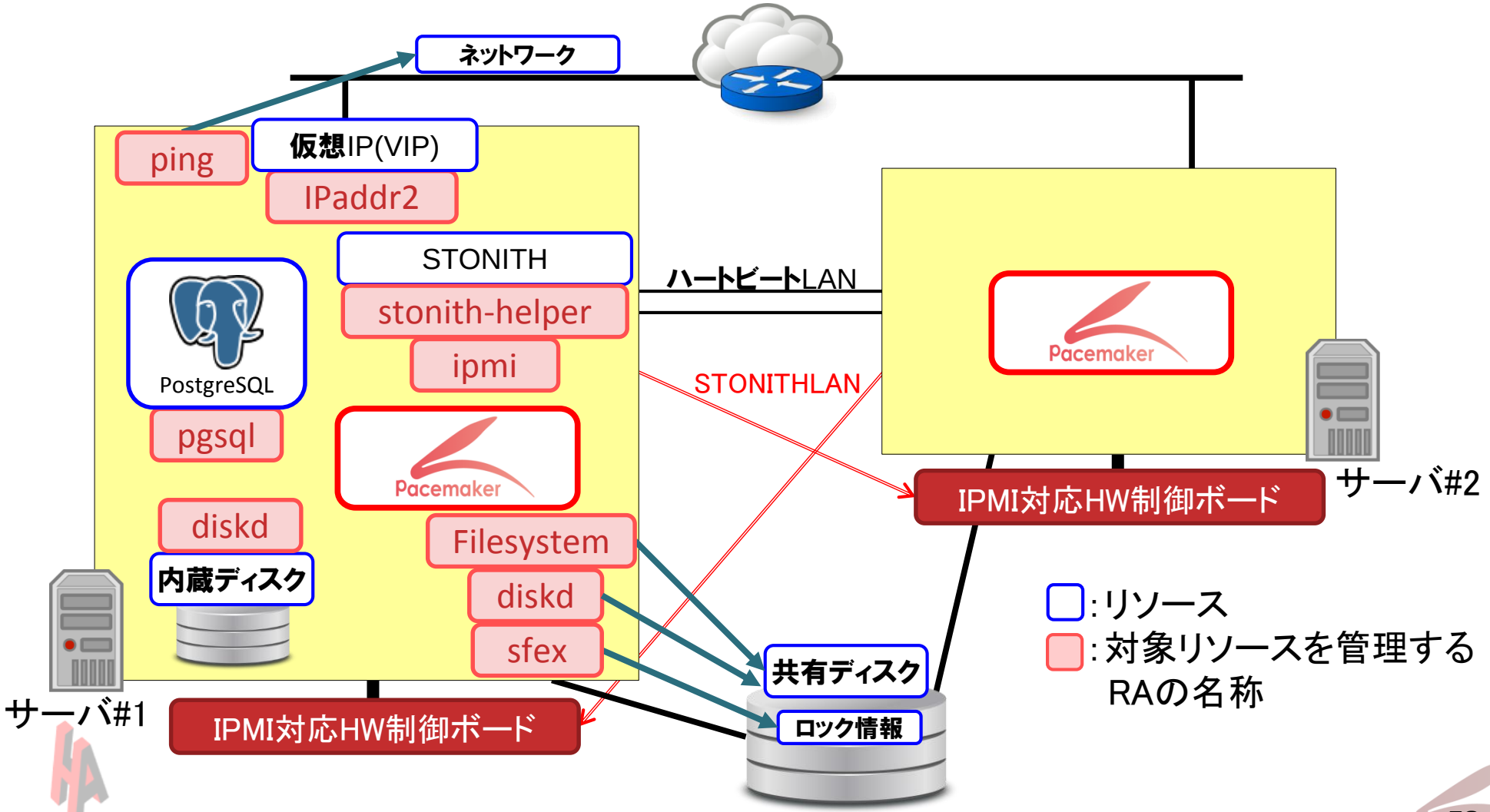
＜stonith-helperによる相撃ち阻止のイメージ＞



Pacemakerによる
リソース管理

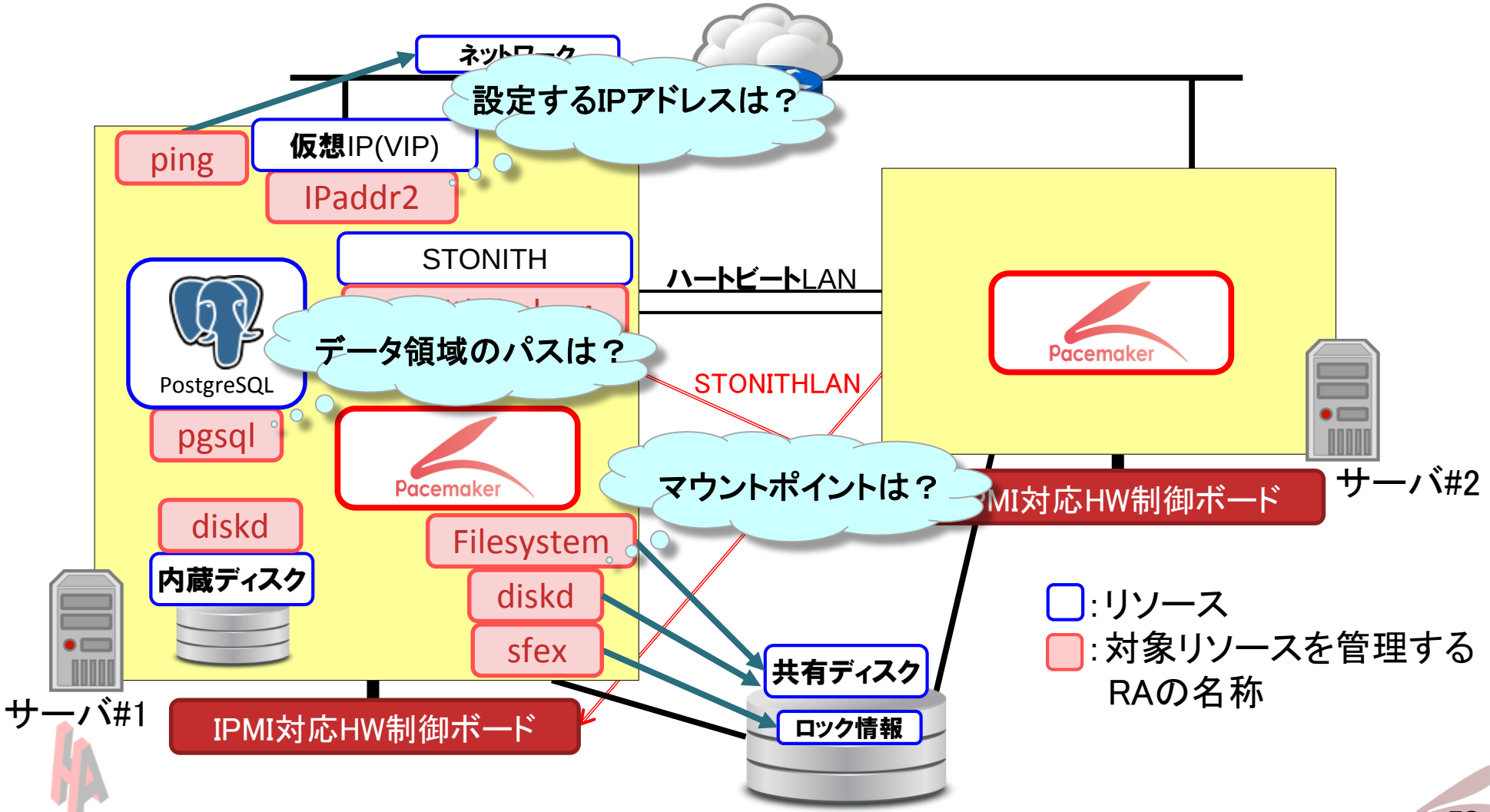
リソース定義

実際のHAクラスタ構成では様々なリソースを管理



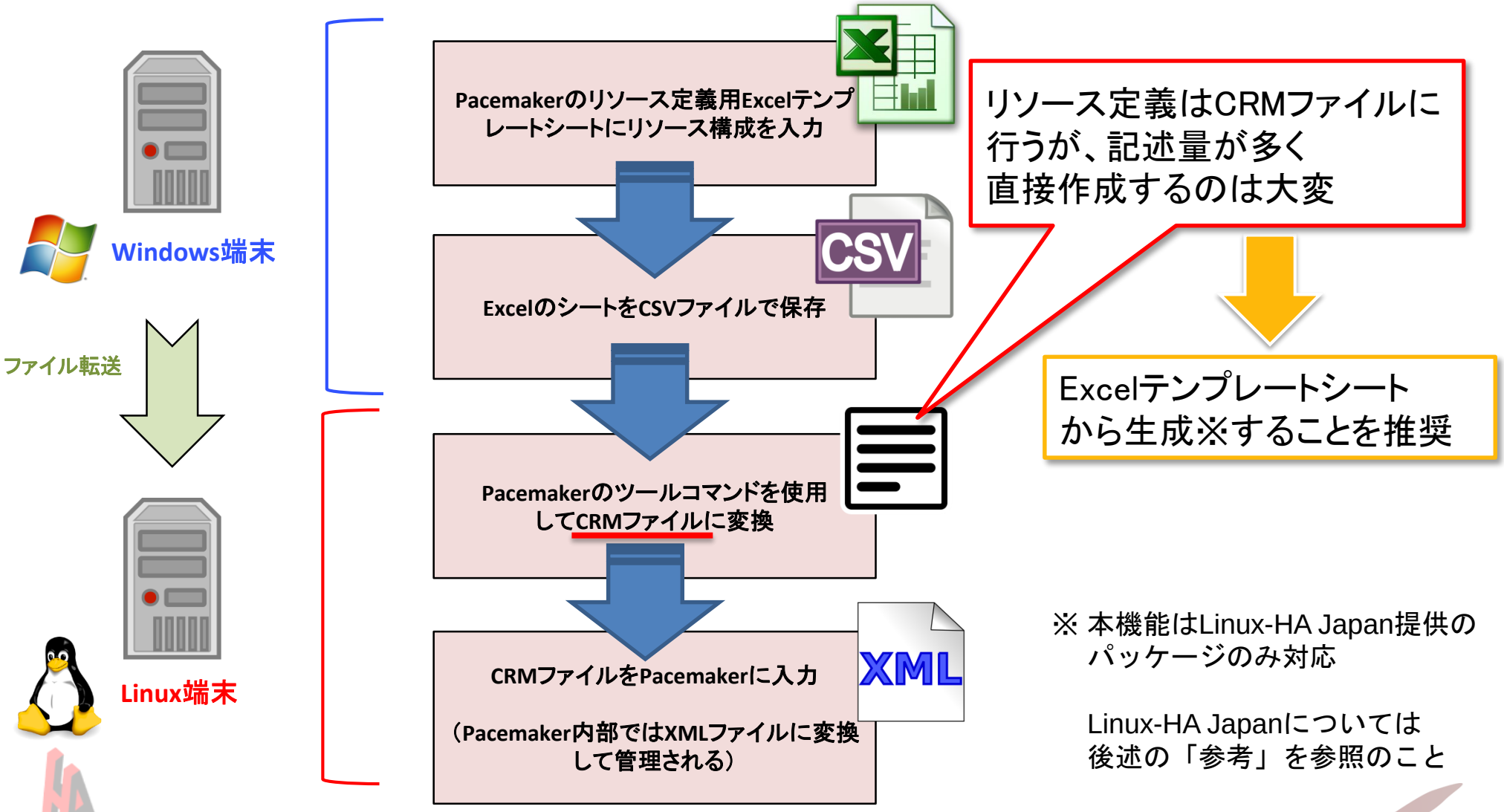
リソース定義

実際のHAクラスタ構成では様々なリソースを管理　→リソース毎に設定が必要



リソース定義

✓ Pacemakerにリソースを定義するまでの流れ



※ 本機能はLinux-HA Japan提供のパッケージのみ対応

Linux-HA Japanについては後述の「参考」を参照のこと

リソース定義

- ✓ 個別リソース(primitiveリソース)の定義例
  以下の要素を設定
  ①IDおよび使用するリソースエージェント(RA)
  ②リソースエージェントに渡すパラメータ
  ③リソースの起動/停止/監視に関する設定

例)PostgreSQLを管理するリソース(以下、pgsqlリソース)の場合

#表 7-1-6 クラスタ設定 … Primitiveリソース (id=prmApPostgreSQLDB)

PRIMITIVE

	P	id	class	provider	type	
#		リソースID	class	provider	type	概要
①		prmApPostgreSQLDB	ocf	heartbeat	pgsql	PostgreSQL制御*A

	A	type	name	value	
#		パラメータ種別	項目	設定内容	概要
②		params	pgctl	**/usr/pgsql-9.6/bin/pg_ctl**	pg_ctlコマンド・パス*A'
			psql	**/usr/pgsql-9.6/bin/psql**	psqlコマンド・パス*A'
			pgdata	**/dbfp/pgdata/data**	DBクラスタ領域ディレクトリ名*A'
			pgdba	**postgres**	DB管理ユーザ*A'
			pgport	**5432**	通信ポート番号*A'
			pgdb	**template1**	アクセス確認用DB名*A'

	O	type	timeout	interval	on-fail	start-delay	
#		オペレーション	タイムアウト値	監視間隔	障害時の動作	起動前待機時間	備考
③		start	300s	0s	restart		タイムアウト値は*C、それ以外は*A
		monitor	60s	10s	restart		タイムアウト値、監視間隔は*C、それ以外は*A
		stop	300s	0s	fence		タイムアウト値は*C、それ以外は*A

リソース定義

- ✓ 個別リソース(primitiveリソース)の定義例
  以下の要素を設定
  ①IDおよび使用するリソースエージェント(RA)
  ②リソースエージェントに渡すパラメータ
  ③リソースの起動/停止/監視に関する設定

例)PostgreSQLを管[illegible]ース)の場合

本リソースのIDは”prmApPostgreSQL”
→同じRAを使用して複数のリソースを
稼働させる場合もIDにより識別が可能

pgsql RAを使用

データ領域のパスは
“/dbfp/pgdata/data”

#表 7-1-6 クラスタ設定 … Primi[illegible]mApPostgreSQLDB)

PRIMITIVE

	P	id	class	provider	type	
#		リソースID	class	provider	type	概要
①		prmApPostgreSQLDB	ocf	heartbeat	pgsql	PostgreSQL制御*A

	A	type	name	value	
#		パラメータ種別	項目	設定内容	概要
②		params	pgctl	/usr/pgsql-9.6/bin/pg_ctl	pg_ctlコマンド・パス*A'
			psql	/usr/pgsql-9.6/bin/psql	
			pgdata	/dbfp/pgdata/data	トリ名*A'
			pgdba	postgres	
			pgport	5432	
			pgdb	template1	アクセス確認用DB名*A'

	O	type	timeout	interval	on-fail	start-delay	
#		オペレーション	タイムアウト値	監視間隔	障害時の動作	起動前待機時間	備考
③		start	300s	0s	restart		タイムアウト値は*C、それ以外は*A
		monitor	60s	10s	restart		タイムアウト値、監視間隔は*C、それ以外は*A
		stop	300s	0s	fence		タイムアウト値は*C、それ以外は*A

10秒間隔で監視、60秒応答がない場合にタイムアウト

リソース定義

- ✓ 個別リソース(primitiveリソース)の定義例
  以下の要素を設定
  ①IDおよび使用するリソースエージェント(RA)
  ②リソースエージェントに渡すパラメータ
  ③リソースの起動/停止/監視に関する設定

例)共有ディスクを管理するリソース(以下、Filesystemリソース)の場合

#表 7-1-2 クラスタ設定 … Primitiveリソース (id=prmFsPostgreSQLDB1)

PRIMITIVE

	P	id	class	provider	type	
#		リソースID	class	provider	type	概要
①		prmFsPostgreSQLDB1	ocf	heartbeat	Filesystem	DBクラスタ領域 *A

	A	type	name	value	
#		パラメータ種別	項目	設定内容	概要
②		params	fstype	xfs	
			device	/dev/mapper/mpathd1	
			directory	/dbfp/pgdata	
			force_unmount	safe	

	O	type	timeout	interval	on-fail	start-delay	
#		オペレーション	タイムアウト値	監視間隔	障害時の動作	起動前待機時間	備考
③		start	60s	0s			タイムアウト値は*C、それ以外は *A
		monitor	60s	10s			タイムアウト値、監視間隔は *C、それ以外は *A
		stop	60s	0s	fence		タイムアウト値は*C、それ以外は *A

Filesystem RAを使用

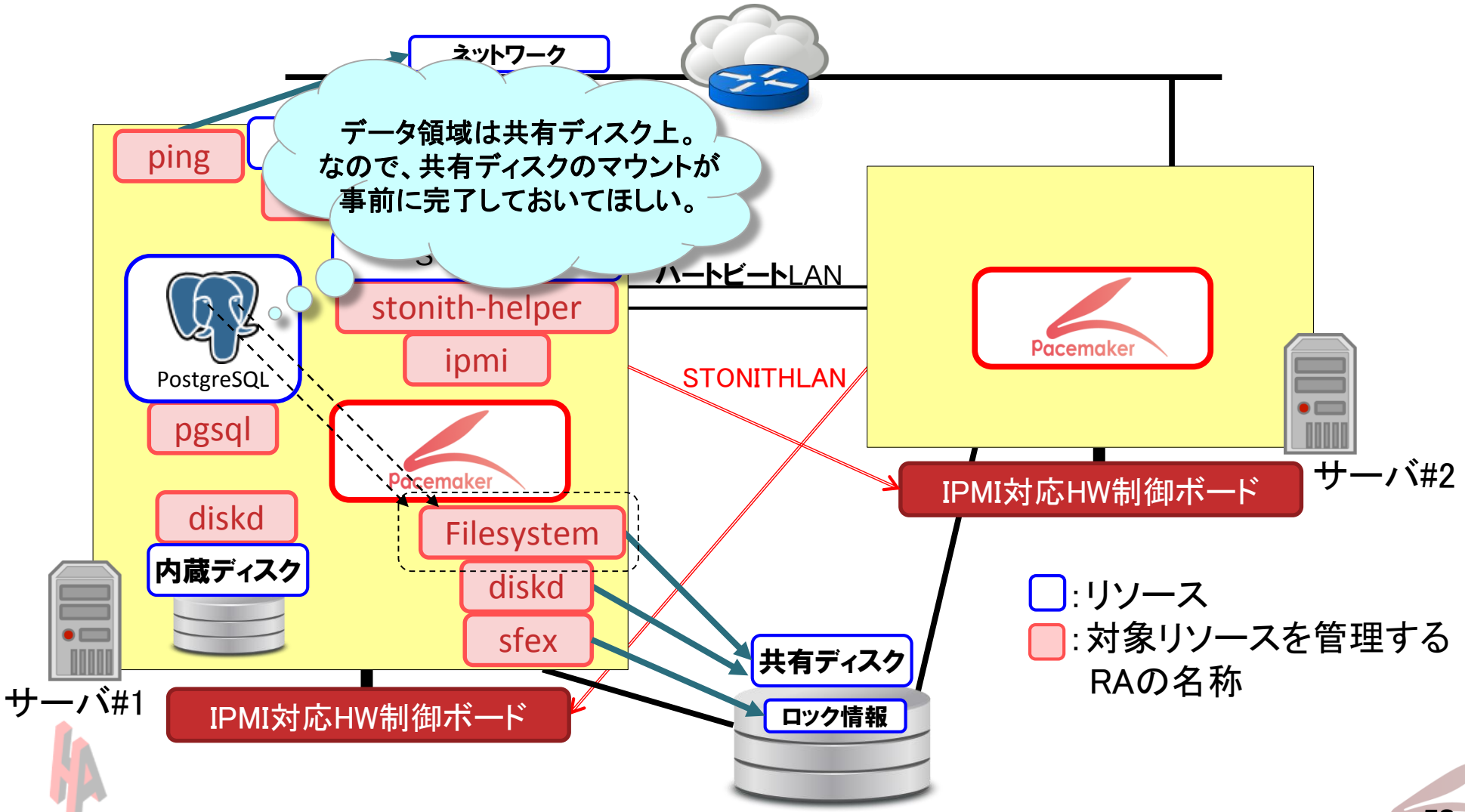
マウント対象のデバイス(本例では共有ディスク)は
“/dev/mapper/mpathd1”

マウントポイントは
“/dbfp/pgdata”

個別のリソース定義についてはこれでOK！　だけど…

リソース構成

リソース同士の関連を意識した制御が必要なケースがある(その1)



リソース構成

- ✓ リソース構成(Groupリソース)の定義
  複数のprimitiveリソースをひとまとめにして同じノード上で起動する、
  リソースの起動・停止順序を制御する、といったことが可能

例)pgsqlリソースを含むGroupリソース(以下、pgsqlリソースグループ)の定義

#表 4-1 クラスタ設定 … リソース構成

RESOURCES

#	resourceItem リソース構成要素	resourceItem	resourceItem	id リソースID	概要	備考
	Group			grpPostgreSQLDB	グループ定義	*A
		Primitive		prmExPostgreSQLDB	共有ディスク排他制御	*A
		Primitive		prmFsPostgreSQLDB1	DBクラスタ領域	*A
		Primitive		prmFsPostgreSQLDB2	WALログ領域	*A
		Primitive		prmFsPostgreSQLDB3	アーカイブログ領域	*A
		Primitive		prmIpPostgreSQLDB	DB仮想IP割当	*A
		Primitive		prmApPostgreSQLDB	PostgreSQL制御	*A

リソース構成

- ✓ リソース構成(Groupリソース)の定義
  複数のprimitiveリソースをひとまとめにして同じノード上で起動する、
  リソースの起動・停止順序を制御する、といったことが可能

例)pgsqlリソースを含むGroupリソース(以下、pgsqlリソースグループ)の定義

#表 4-1 クラスタ設定 … リソース構成

RESOURCES

	resourceItem	resourceItem	resourceItem	id		
#	リソース構成要素			リソースID	概要	備考
	Group			grpPostgreSQLDB	グループ定義	*A
		Primitive		prmExPostgreSQLDB	共有ディスク排他制御	*A
		Primitive		prmFsPostgreSQLDB1	DBクラスタ領域	*A
		Primitive		prmFsPostgreSQLDB2	WALログ領域	*A
		Primitive		prmFsPostgreSQLDB3	アーカイブログ領域	*A
		Primitive		prmIpPostgreSQLDB	DB仮想IP割当	*A
		Primitive		prmApPostgreSQLDB	PostgreSQL制御	*A

グループ内のリソースが一つでも故障した場合は
グループに所属する全てのリソースがフェールオーバする

リソース構成

- ✓ リソース構成(Groupリソース)の定義
  複数のprimitiveリソースをひとまとめにして同じノード上で起動する、
  リソースの起動・停止順序を制御する、といったことが可能

例)pgsqlリソースを含むGroupリソース(以下、pgsqlリソースグループ)の定義

#表 4-1 クラスタ設定 … リソース構成

RESOURCES

#	resourceItem リソース構成要素	resourceItem	resourceItem	id リソースID		
	Group			grpPostgreSQLDB		
		Primitive		prmExPostgreSQLDB	共有ディスク排他制御	*A
		Primitive		prmFsPostgreSQLDB1	DBクラスタ領域	*A
		Primitive		prmFsPostgreSQLDB2	WALログ領域	*A
		Primitive		prmFsPostgreSQLDB3	アーカイブログ領域	*A
		Primitive		prmIpPostgreSQLDB	DB仮想IP割当	*A
		Primitive		prmApPostgreSQLDB	PostgreSQL制御	*A

起動

Filesystemリソースにより
共有ディスクデバイスを
“/dbfp/pgdata”にマウント

記述した順にリソースが起動する

pgsqlリソースによりPostgreSQL起動時に
“/dbfp/pgdata”配下のディレクトリを
データ領域として設定

リソース構成

- ✓ リソース構成(Groupリソース)の定義
  複数のprimitiveリソースをひとまとめにして同じノード上で起動する、
  リソースの起動・停止順序を制御する、といったことが可能

例)pgsqlリソースを含むGroupリソース(以下、pgsqlリソースグループ)の定義

#表 4-1 クラスタ設定 … リソース構成

RESOURCES

#	resourceItem	resourceItem	resourceItem	id		
	リソース構成要素			リソースID		
	Group			grpPostgreSQLDB		
		Primitive		prmExPostgreSQLDB		*A
		Primitive		prmFsPostgreSQLDB1	DBクラスタ領域	*A
		Primitive		prmFsPostgreSQLDB2	WALログ領域	*A
		Primitive		prmFsPostgreSQLDB3	アーカイブログ領域	*A
		Primitive		prmIpPostgreSQLDB	DB仮想IP割当	*A
		Primitive		prmApPostgreSQLDB	PostgreSQL制御	*A

“/dbfp/pgdata”へのアクセスは発生しないため
Filesystemリソースが対象デバイスを
安全にアンマウントできる

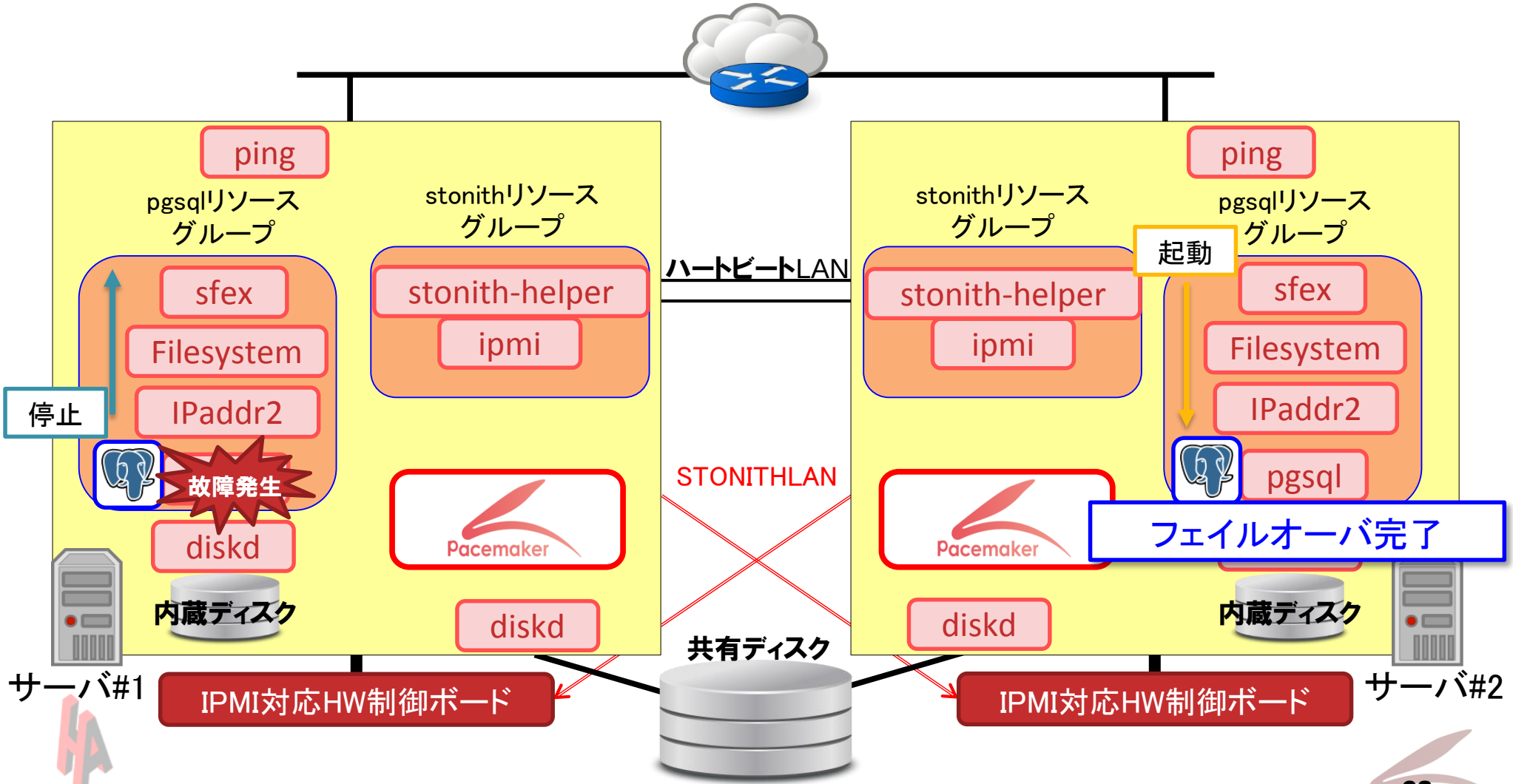
停止

記述した逆順にリソースが停止する

pgsqlリソースによりPostgreSQLを停止
→“/dbfp/pgdata”へのアクセスが停止

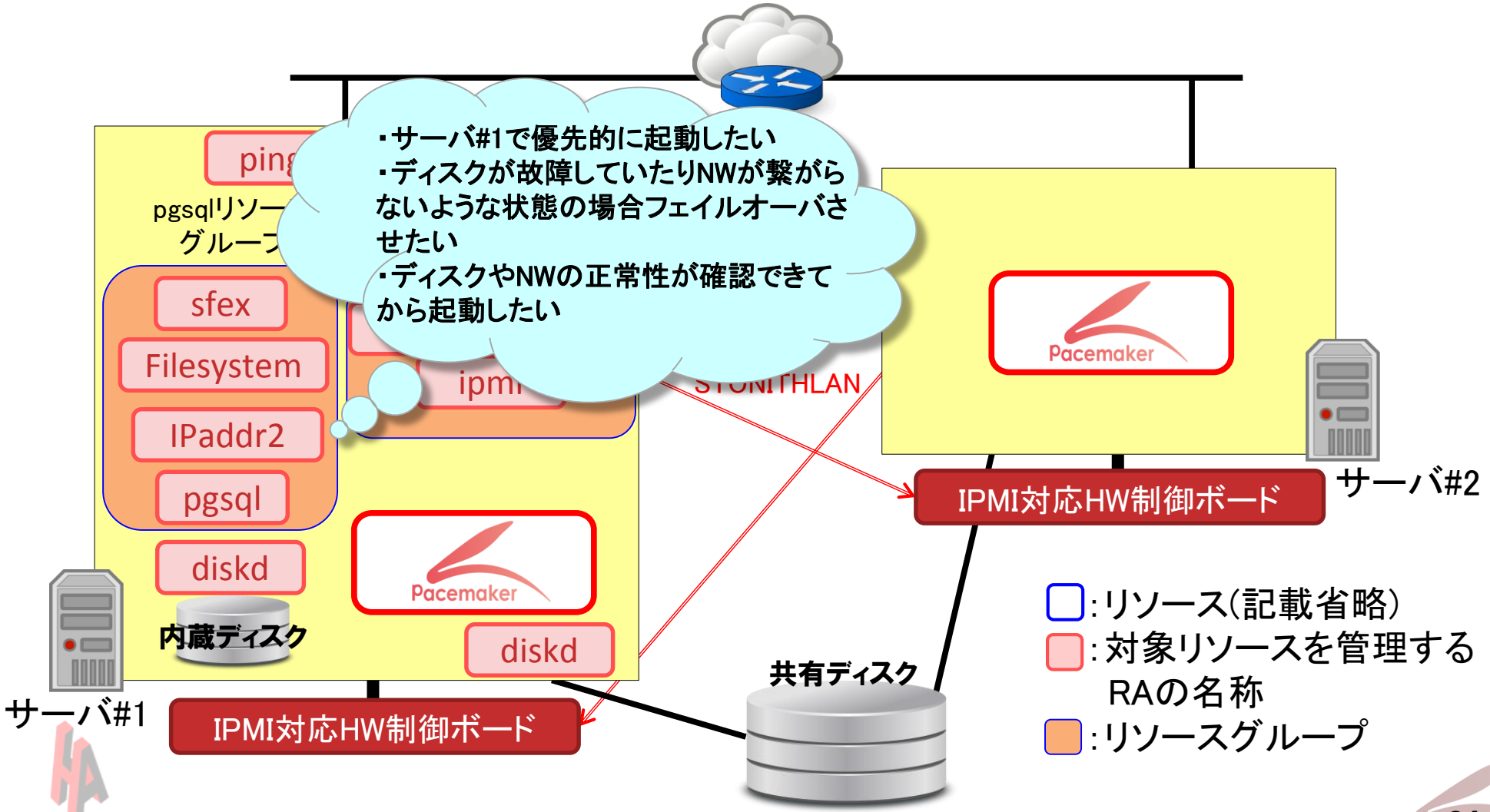
リソース構成

＜リソース構成を踏まえたPostgreSQL故障時のフェイルオーバイメージ＞



リソース制約

リソース同士の関連やサーバを意識した制御が必要なケースがある(その2)



リソース制約

どこで、どのように起動するかの制約(location,colocation,order)をかけることが可能

例)pgsqlリソースグループのlocation (リソースが稼動するノード)制約

#表 8-1 クラスタ設定 … リソース配置制約

LOCATION_EXPERT

#	rsc リソースID	score スコア	bool_op and/or	attribute 条件属性名	op 条件	value 条件値	role 役割	id_spec ルールID	備考
	grpPostgreSQLDB	200		#uname	eq	**server01**			ホスト hpdb0101を優先的にActiveにする *A'
		100		#uname	eq	**server02**			
		-inf	or	lt_ping_set	not_defined				ネットワーク故障が発生したノードでは起動しない*A
					lt	100			
					not_defined				共有ディスク故障が発生したノードでは起動しない*A
				diskcheck_status	eq	ERROR			
		-inf	or	diskcheck_status_internal	not_defined				内蔵ディスク故障が発生したノードでは起動しない*A
				diskcheck_status_internal	eq	ERROR			
	grpStonith1	-inf		#uname	eq	**hpdb0101**			*A'
	grpStonith2	-inf		#uname	eq	**hpdb0201**			*A'

“server01”というホスト名のノード上で優先的に起動する

6ページ

その他にも起動するノードに関する
様々な制約をかけることが可能

リソース制約

どこで、どのように起動するかの制約(location,colocation,order)をかけることが可能

例)pgsqlリソースグループのcolocation(リソースの配置の依存関係),order(起動順序)制約

#表 9-1 クラスタ設定 ... リソース同居制約

COLOCATION

#	rsc 制約関連リソースID *A	with-rsc 制約対象リソースID *A	score スコア(重み付け) *A			
	grpPostgreSQLDB	clnPing	inf			
	grpPostgreSQLDB	clnDiskd1	inf			
	grpPostgreSQLDB	clnDiskd2	inf			

#表 10-1 クラスタ設定 ... リソース起動順序制約

ORDER

#	first-rsc 先に起動するリソースID *A	then-rsc 後に起動するリソースID *A	score スコア(重み付け) *A				
	clnPing	grpPostgreSQLDB	0			n	
	clnDiskd1	grpPostgreSQLDB	0			n	
	clnDiskd2	grpPostgreSQLDB	0			n	

pgsqlリソースグループが起動するには
pingリソース，diskdリソースが起動していなければいけない
(NWやディスクが正常稼働していなければいけない)

pgsqlリソースグループはpingリソース，diskdリソースの
後に起動しなければいけない

リソース制約

＜リソース制約を踏まえたネットワーク故障時のフェイルオーバイメージ＞