

---

# 試して覚えるPacemaker入門

## 排他制御編



---

オープンソースカンファレンス2015 Tokyo/Fall  
LinuxHA-Japan 飯田 雄介

---

# 自己紹介

---

## ■ 名前

- 飯田 雄介(いいだ ゆうすけ)

## ■ 所属

- Linux-HA Japanプロジェクト

## ■ 主な活動

- Pacemaker本体の機能改善や、外部ツールの開発を行っています。
- Linux-HA Japanからpm\_logconvやpm\_crmgenといったツールを提供しています。

# 本日のおはなし

---

- 高可用(HA)クラスタにとって重要な機能の一つである排他制御機能について、その重要性やデモを通して実際どのように動くのかを解説していきます。
- 話題
  - Pacemakerとは？
  - なぜクラスタには排他制御が必要なのか？
  - Pacemakerの排他制御機能を紹介
  - デモを使ったSTONITHの実演

---

**そもそもPacemakerを  
知らない人のために**

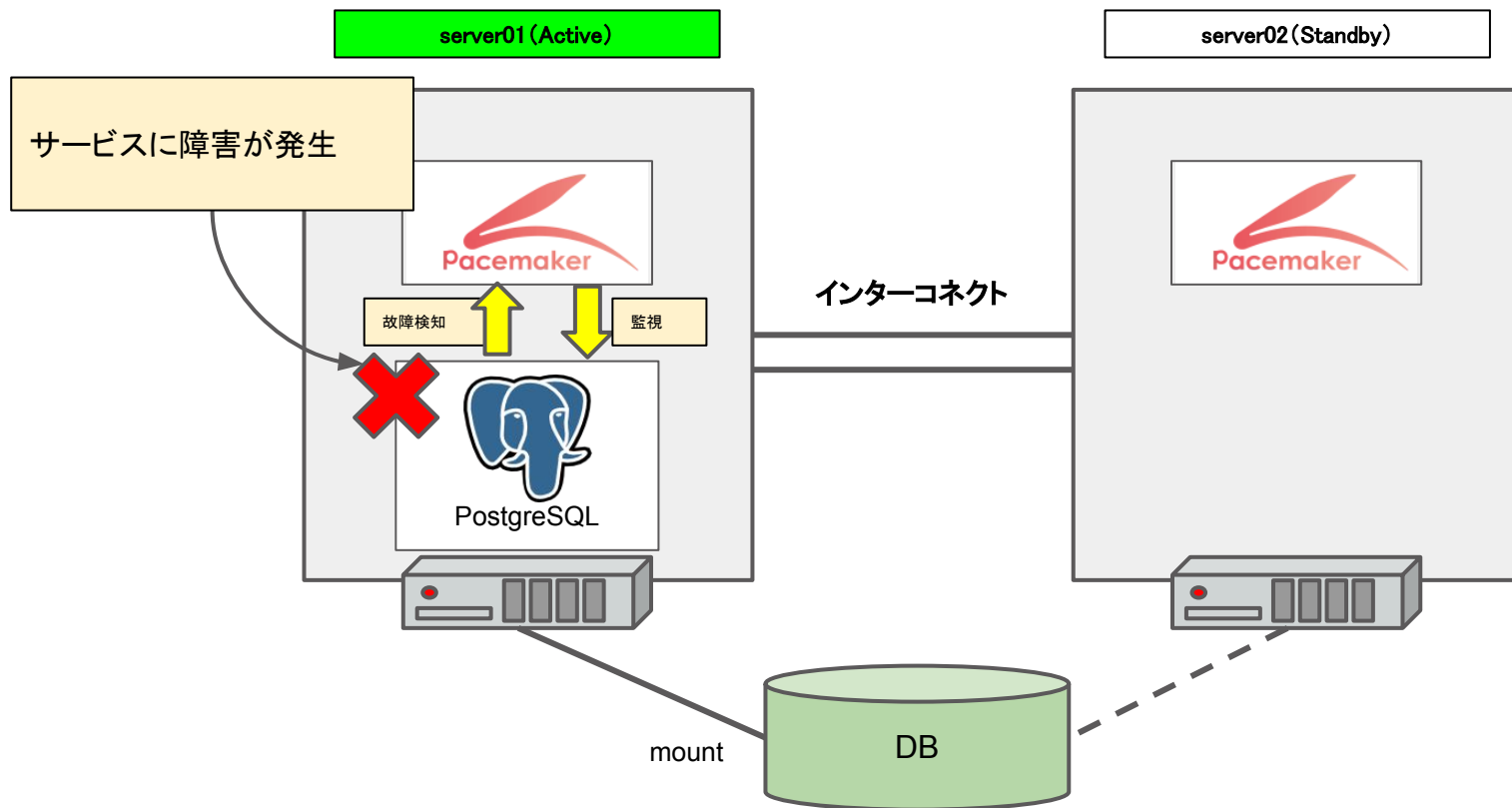
# Pacemakerとは？

---

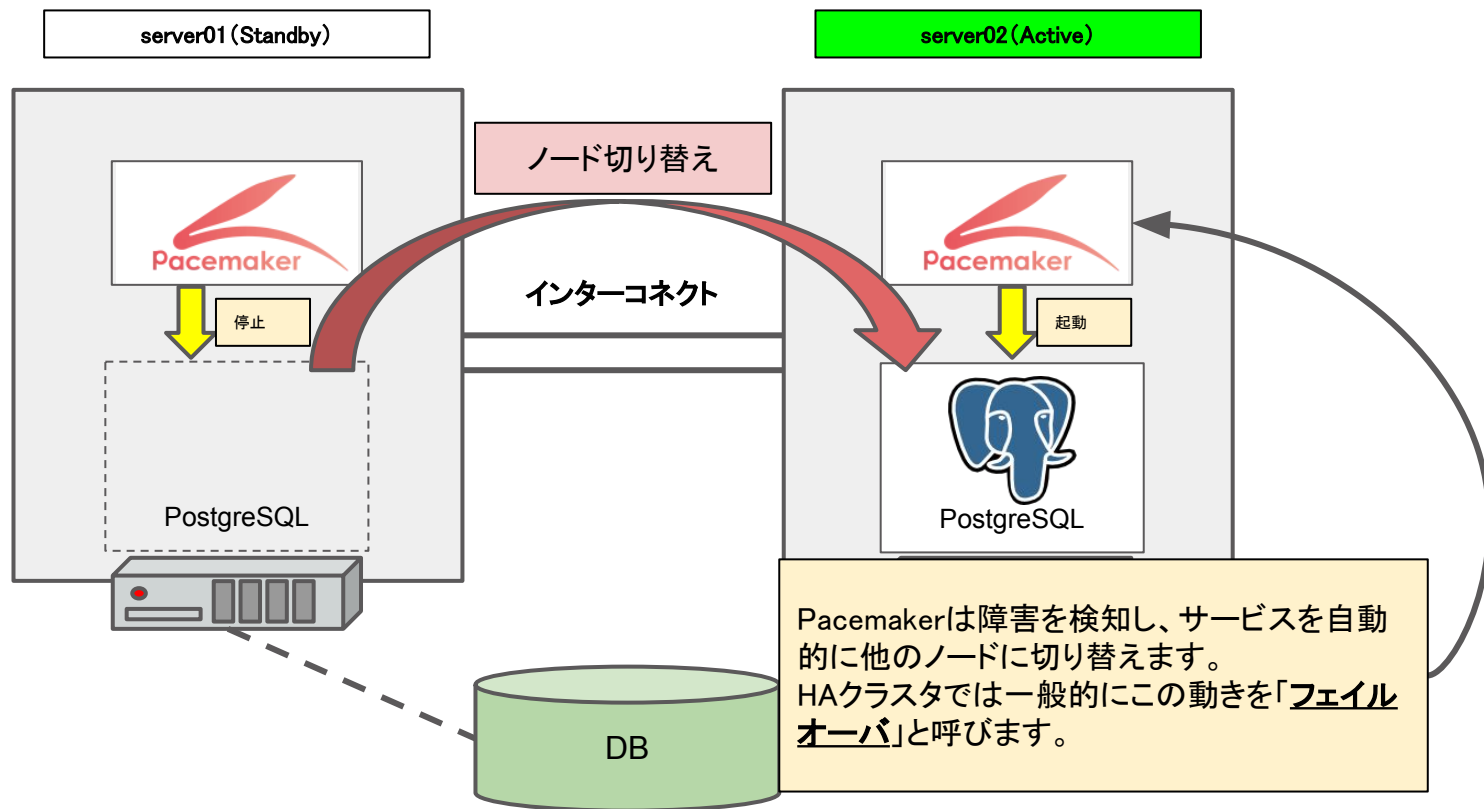
- Pacemaker(ペーすめーかー)とはオープンソースで開発されている高可用クラスタソフトウェアのことです。

**Pacemakerは障害が発生するとそれを検知し、自動的にサービスをフェイルオーバーすることでシステムの可用性を高めてくれます。**

# Pacemakerによるフェイルオーバーのイメージ (故障発生時)



# Pacemakerによるフェイルオーバーのイメージ (フェイルオーバー後)



# サービスが継続できなくなる2つの障害

---

- HAクラスタには発生するとサービス継続不能に陥ってしまう致命的な2つの障害が存在します。

1. スプリットブレイン
2. リソースの停止故障

これら2つの障害からクラスタを救うためには  
「排他制御機能」が必要となります！



# スプリットブレインとは？

---

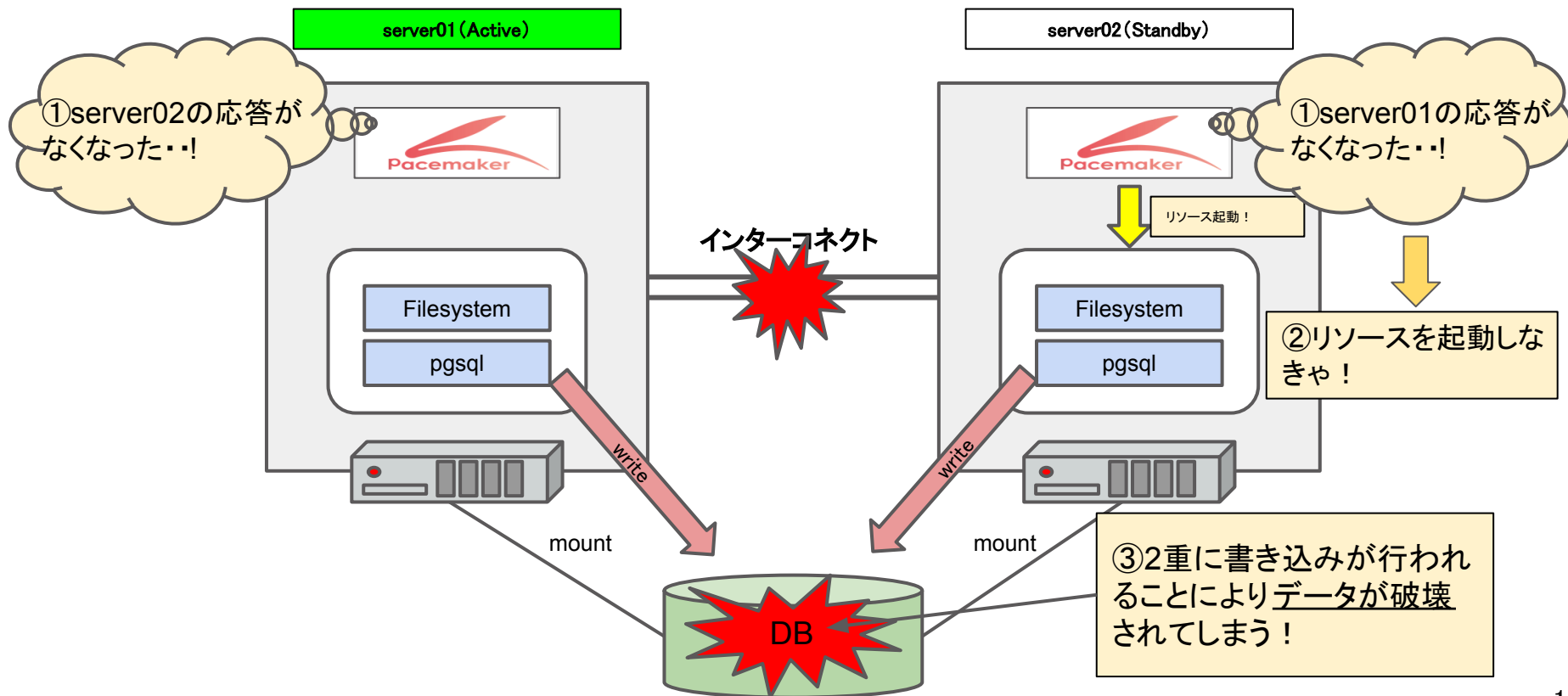
## ■ どんな障害なのか？

- クラスタを組んでいるノード間をつなぐインターコネクト通信が全て切断されてお互いの状態が把握できなくなったPacemakerが、それぞれのノードでリソースを起動し始めてしまう状態のことです。

## ■ クラスタはどうなってしまう？

**それぞれのノードで起動したリソースが2重で書き込みを行うことで、データが破壊されてしまいサービスが復旧不可能な状態になってしまいます。**

# スプリットブレインによるサービス停止のイメージ



# リソースの停止故障とは？

---

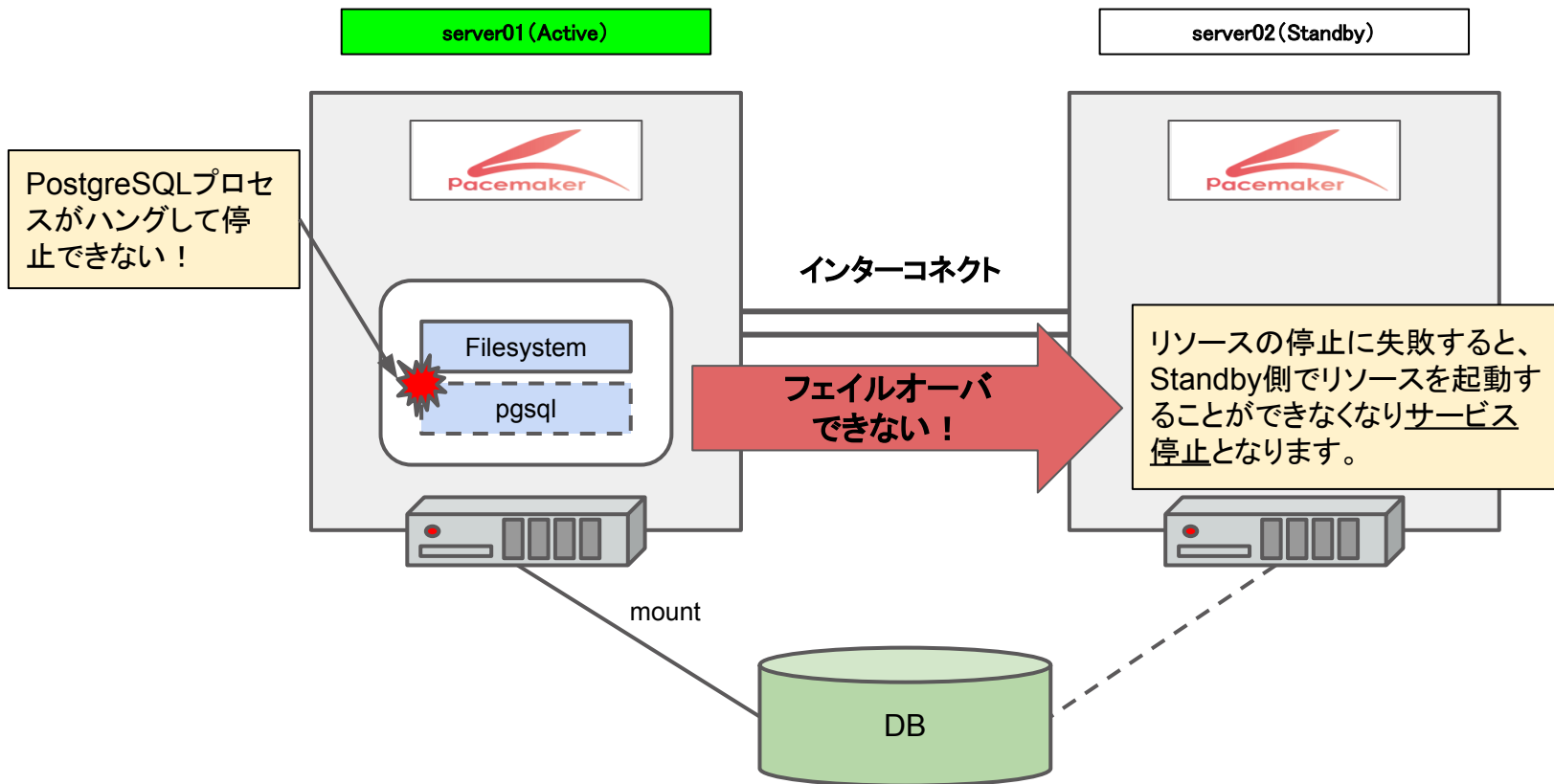
## ■ どんな障害なのか？

- Pacemakerが管理しているリソースが停止できなくなってしまった状態のことです。
- 例えば・・・プロセスの書き込みが継続して行われているため共有ディスクをアンマウントすることができない状態など

## ■ クラスタはどうなってしまう？

**停止に失敗したリソースはどのノードにもフェイルオーバーすることができないため、サービスが継続不可能な状態となってしまいます。**

# リソース停止故障によるサービス停止のイメージ



# もう一度念押し

---

**これらの障害が発生した時にリソースの重複起動やサービス停止を防ぐために排他制御機能が必要です！**

---

# **Pacemakerの備える 排他制御機能を紹介**

# Pacemakerの備える排他制御機能

---

- Pacemakerでは次に挙げる3つの排他制御機能を使用することができます。

1. STONITH(読み:すとにす)
  - ノードの電源操作を行う排他制御機能
2. SFEX(読み:えすえふいーえつくす)
  - 共有ディスクを使った排他制御機能
3. VIPcheck(読み:びっぷちえつく)
  - 仮想IPを使った排他制御機能

# 排他制御機能の対応範囲

- 下記は各排他制御機能と障害発生時のサービス継続性を表す対応表です。

- : サービス継続可能
- ×: サービス継続不可能

STONITHは障害に対する、  
対応範囲が最も広い！

障害パターン	排他制御なし	STONITH	SFEX	VIPcheck
スプリットブレイン	×	○	○	○
リソースの停止故障	×	○	×	×



---

# STONITHについて紹介

# STONITHとは？

---

- STONITHとは制御が利かなくなったノードの電源を強制的に停止してクラスタから「強制的に離脱(Fencing)」させる機能のことです。
- 下記の英文の頭文字を取って作られています。

「Shoot-**T**he-**O**ther-**N**ode-**I**n-**T**he-**H**ead」

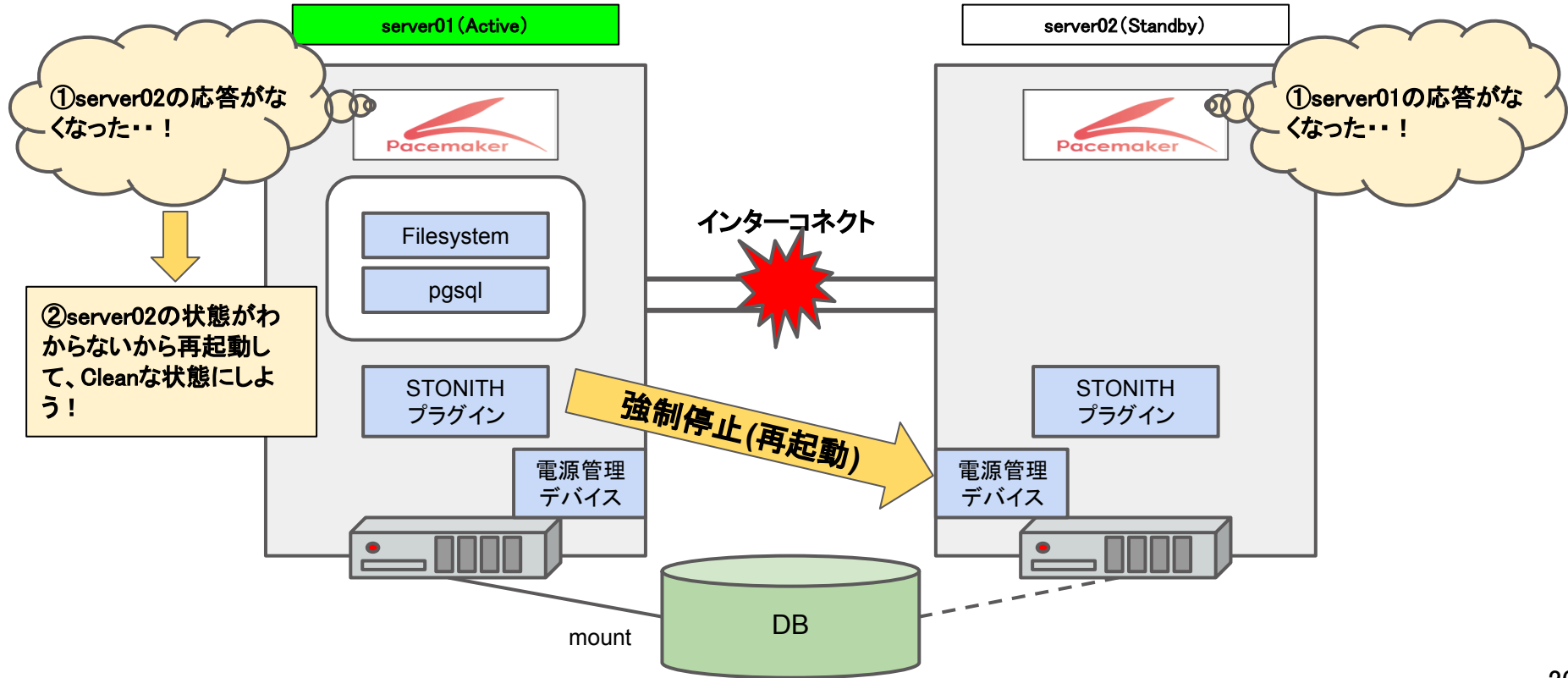
# STONITHプラグイン

- Pacemakerには、環境に合わせて使える様々なSTONITHプラグインが標準装備されています。

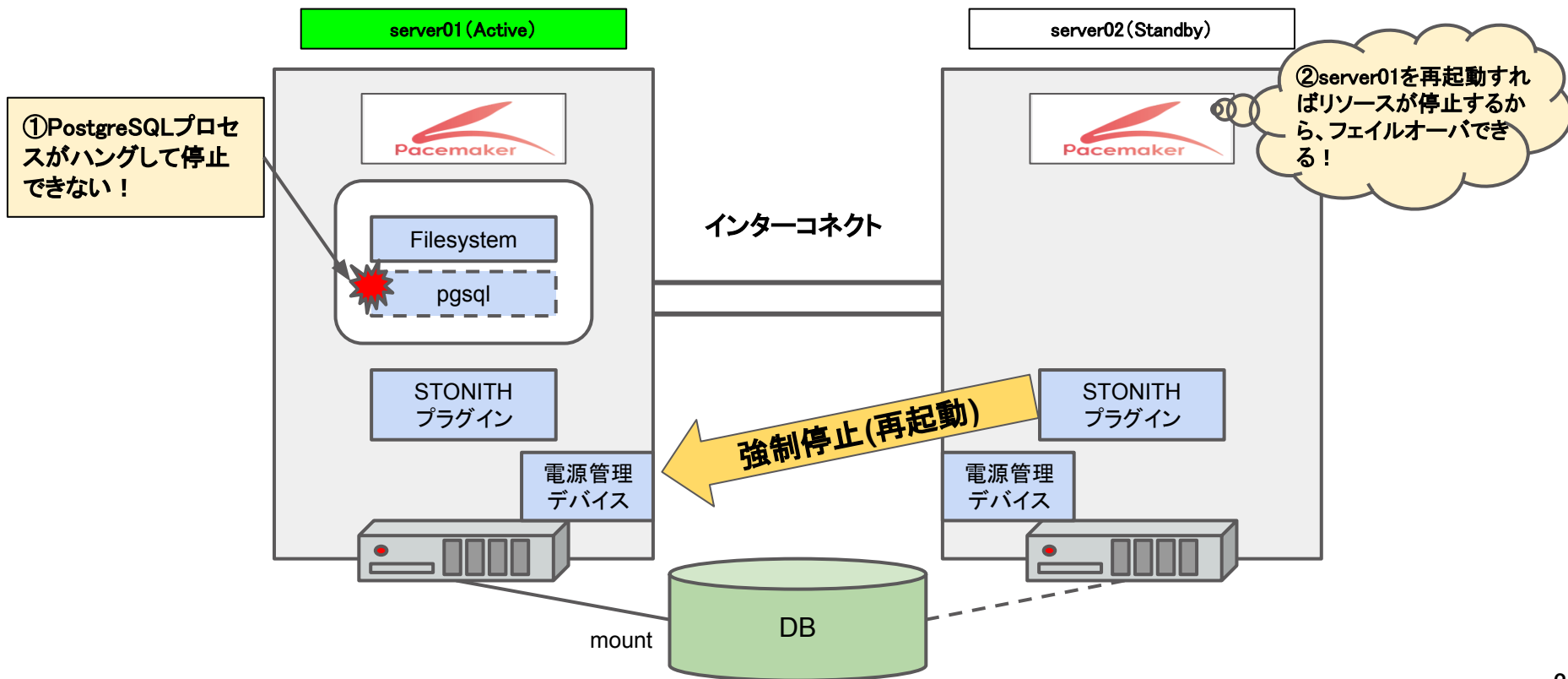
サーバに搭載されている電源管理デバイスに合わせて使い分けてください。

目的	プラグイン名
フェンシング(電源断)制御	ipmi (IPMIデバイス用) libvirt (KVM,Xen等 仮想マシン制御用) ec2 (AmazonEC2用) などなど
サーバ生死確認、相撃ち防止	stonith-helper

# STONITHによる排他制御のイメージ (スプリットブレイン時)



# STONITHによる排他制御のイメージ (リソース停止故障)



---

# STONITHの相撃ち問題

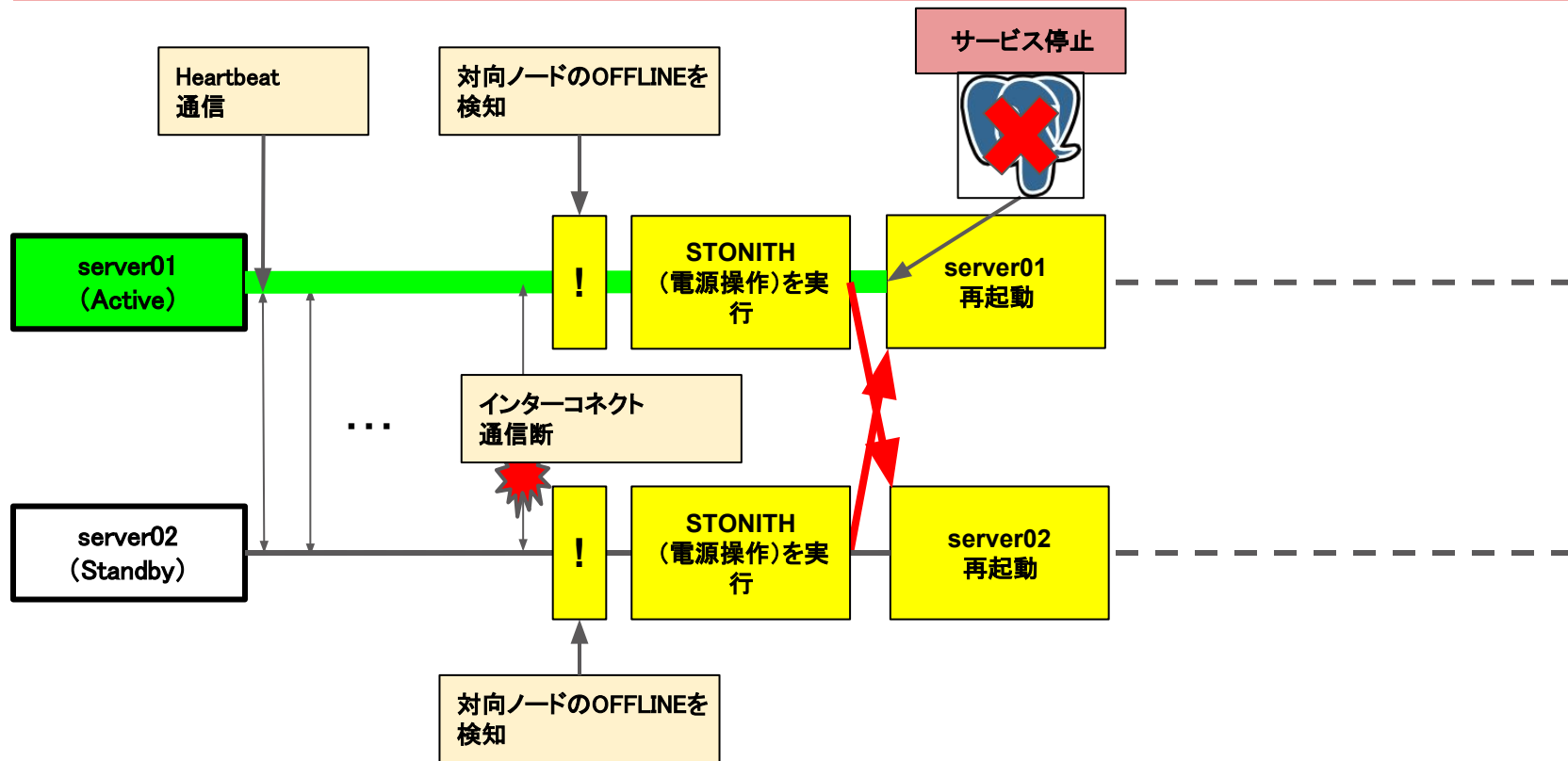
# STONITHの相撃ちという問題

---

- ノードの電源を強制的に停止させることでリソースを安全にフェイルオーバさせるSTONITHですが、STONITHには一つ弱点があります。
- それはスプリットブレインになった時、お互いのノードがSTONITHを実行し合い「相撃ち」が起こってしまうということです。

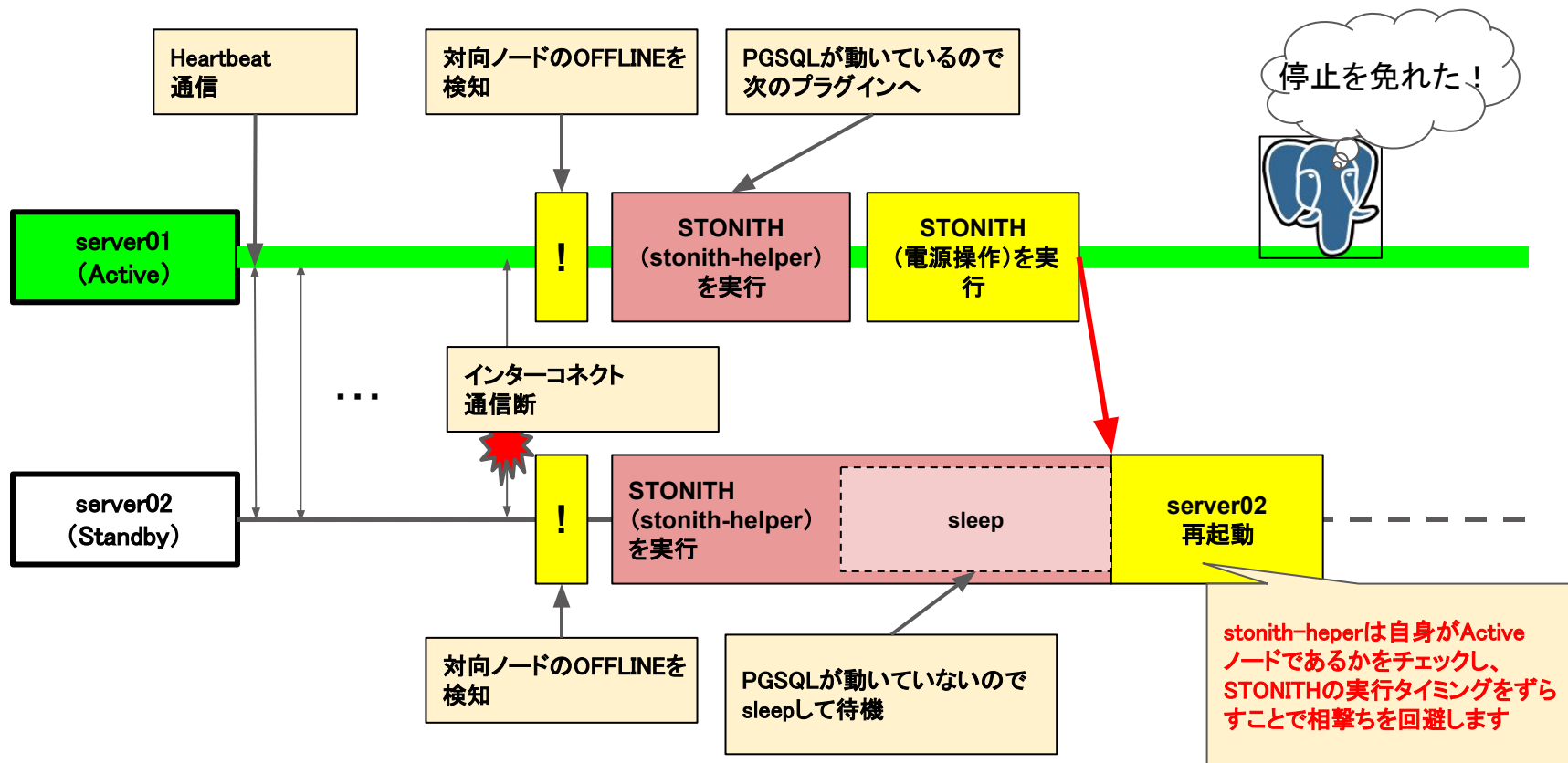
この問題は「stonith-helper」というSTONITHの補助プラグインを使うことで対処することができます。

# STONITH相撃ちのイメージ





# stonith-helperによる相撃ち阻止のイメージ

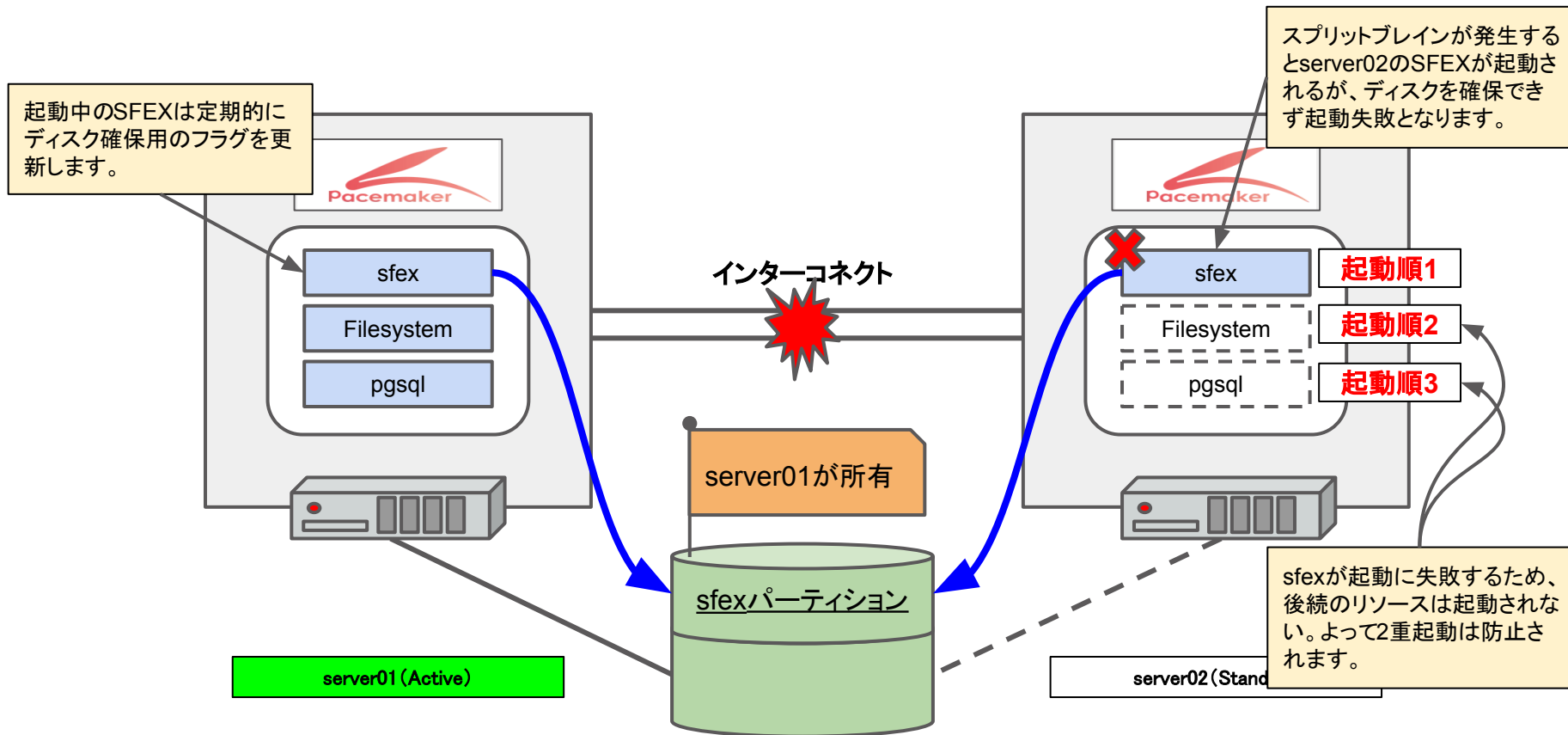


# STONITHが使えない環境での排他制御方法

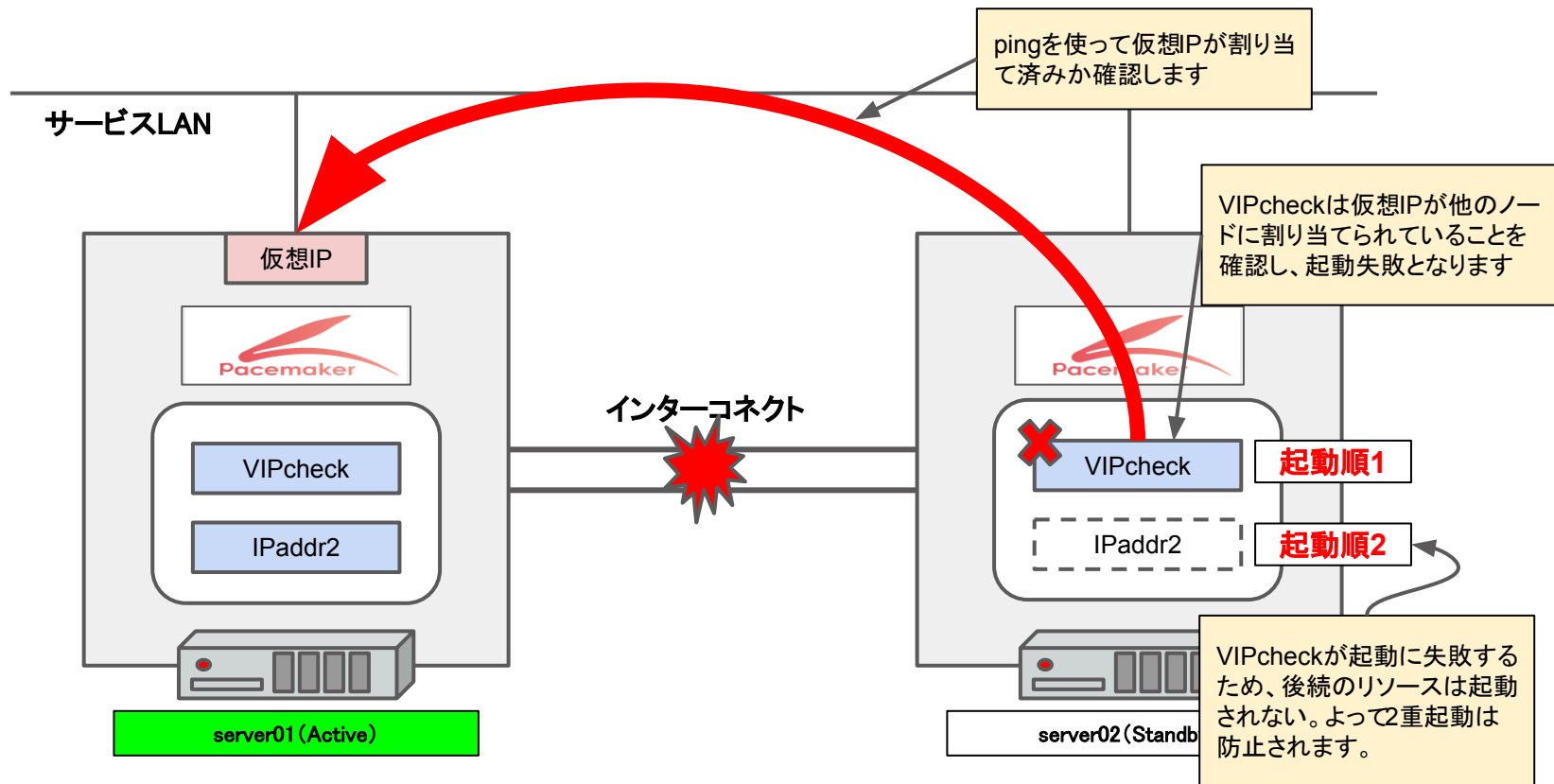
- サーバが電源管理デバイスを備えていないなど、STONITHを使うことができない環境の場合、リソースの停止故障には対応できませんが、環境に合わせて次の排他制御機能を使うことができます。

機能名	機能	使用要件
SFEX	共有ディスクを使った排他制御	共有ディスクを使っている構成で使用可能
VIPcheck	仮想IPを使った排他制御	仮想IPを使っている構成で使用可能

# SFEXによる排他制御のイメージ



# VIPcheckによる排他制御のイメージ



---

# STONITHの使い方

# リソース定義ファイルの設定

---

- STONITHを使用するためにはリソース定義ファイル※に次に挙げる設定を行います。
  - STONITH機能の有効化設定
  - STONITHプラグインの設定
  - STONITHプラグインの実行順序設定
  - リソース停止故障時にSTONITHを発動させる設定

※リソース定義ファイルとは、Pacemakerに管理させるサービスの情報やクラスタ全体の設定を行うファイルのことです。

# STONITH機能の有効化設定

- STONITH機能を有効化するためリソース定義ファイルに下記の設定を行います。

```
property \  
  no-quorum-policy="ignore" \  
  stonith-enabled="true" \  
  startup-fencing="false"
```

STONITHを有効にする場合、“**true**”を設定します。

※STONITH機能はデフォルトで有効になっています。

※無効にしたい場合は“false”を設定します。

# STONITHプラグインの設定

- STONITHプラグインに関する設定もリソース定義ファイルに設定します。
- 下記に物理マシン環境で一般的に使用されるipmiプラグインの設定例を提示します。

```
primitive prmlpmi stonith:external/ipmi \
```

```
params \
```

```
hostname=server01 \
```

```
ipaddr=192.168.28.50 \
```

```
userid=USERID \
```

```
passwd=PASSWORD \
```

```
interface=lanplus \
```

```
op start interval="0s" timeout="60s" on-fail=restart \
```

```
op monitor interval="3600s" timeout="60s" on-fail=restart \
```

```
op stop interval="0s" timeout="60s" on-fail=ignore
```

使用するプラグインの種類を設定します。  
プラグインの実体は下記のディレクトリに  
存在します。

`/usr/lib64/stonith/plugins/external/`

プラグインのパラメータを設定します。

プラグインのオペレーション  
を設定します。



# プラグインのパラメータ解説(ipmi)

---

■ 下記はipmiプラグインに設定するパラメータの解説です。

パラメータ名	意味
hostname	このプラグインを使って STONITHする対象のホスト名を指定します。
ipaddr	STONITH対象のノードに搭載されている IPMIデバイスのIPを指定します。
userid	IPMIデバイスにログインするためのユーザ IDを指定します。
passwd	IPMIデバイスにログインするためのパスワードを指定します。
interface	使用するIPMIインタフェースを指定します。

# プラグインのパラメータ解説(stonith-helper)

- 今回のデモで使うstonith-helperプラグインのパラメータです。

パラメータ名	意味
pcmk_reboot_retries	プラグインがreboot処理に失敗した時に、やり直す回数を指定します。
hostlist	このプラグインを使って STONITHする対象のホスト名を指定します。
dead_check_target	対象ノードの電源が完全に落ちている事を確認するための設定 ノードに割り当てられている IPを全て指定します。
standby_check_command	相撃ち回避用の設定 スプリットブレインになった時、自ノードが Activeノードであると判定するためのコマンドを設定します。 コマンドのリターンコードが "0"なら自ノードを Activeと判定します。 Standbyだと判定されたノードは一定時間 sleepを実行してSTONITHされるのを待ちます。
run_online_check	スプリットブレイン時以外では相撃ち回避処理を実行する必要はありません。 "yes"を設定することでスプリットブレイン時以外では相撃ち回避用の sleepが実行されなくなります。

# プラグインのパラメータ解説(libvirt)

---

- 同じくデモで使用しているlibvirtプラグインのパラメータです。

パラメータ名	意味
hostlist	このプラグインを使って STONITH する対象のホスト名を指定します。
hypervisor_uri	仮想マシンが起動しているホストマシンのアドレスを指定します。

# STONITHプラグインの実行順序設定

- STONITHプラグインを複数設定した場合、下記の設定を行うことでプラグインを実行する順番を制御することができます。
  - プラグインの実行に失敗した場合、次に指定したプラグインが実行されるという動きになります。

```
fencing_topology \  
server01: prmHelper1-1 prmLibvirt1-2 \  
server02: prmHelper2-1 prmLibvirt2-2
```

STONITH対象のノード名を指定します。

STONITHプラグインのリソースIDをスペース区切りで指定します。  
指定したプラグインは左から順番に実行されます。  
ここでは”stonith-helper → libvirt”の順番で実行されるように設定しています。

# リソース停止故障時に STONITHを発動させる設定

リソース停止故障時にSTONITHを発動させるには下記のようにリソースのopにon-failの設定を行います。

```
primitive prmDB ocf:heartbeat:pgsql \  
  params \  
    pgctl="/usr/pgsql-9.4/bin/pg_ctl" \  
    psql="/usr/pgsql-9.4/bin/psql" \  
    pgdata="/pgsqlldb/data" \  
    start_opt="-p 5432" \  
    pgdba="postgres" \  
    pgport="5432" \  
    pgdb="template1" \  
  op start interval="0s" timeout="120s" on-fail="restart" \  
  op monitor interval="30s" timeout="30s" on-fail="restart" \  
  op stop interval="0s" timeout="120s" on-fail="fence"
```

オペレーション失敗時にSTONITHを実行させるには**fence**を設定します。

STONITHを無効にしている環境では**block**を設定しましょう。

---

# デモ環境の説明と準備

# デモ環境について

---

- デモ環境には仮想マシンを2台使います。
  - OSはCentOS7.1を使用します。
  - Pacemaker-1.1.13がインストールしてあります。
  - TracサービスがActive/Standbyで動作する環境を構築してあります。

# デモ環境で使うリソース

## ■ デモ環境では次に挙げるものをリソース化して管理しています。

### □ サービスリソース

- apache
- postgresql
- IPAddr2 (仮想IPの管理)
- Filesystem (マウントの管理)

### □ 監視リソース

- ping (ネットワークを監視するリソース)
- diskd (ディスクを監視するリソース)

### □ STONITHプラグイン

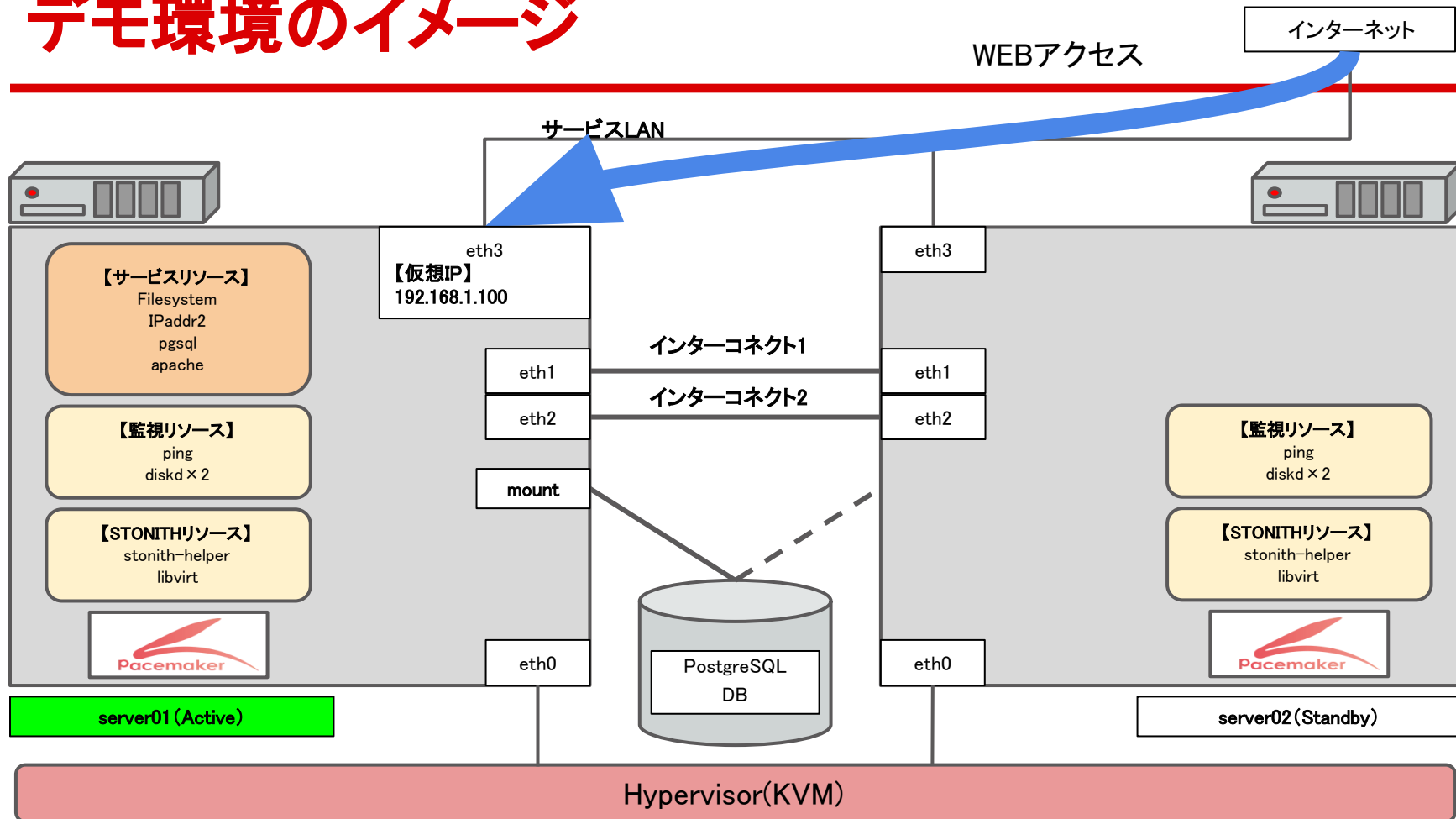
- stonith-helper (STONITHの動作を補助するプラグイン)
- libvirt (KVMなどの仮想マシンを再起動させるプラグイン)

STONITHによる相撃ちを回避するためstonith-helperを使用します。

今回のデモは仮想環境で行うため、電源操作プラグインにlibvirtを使用します。



# デモ環境のイメージ



---

**では、クラスタを起動していきましょう**

# まずサービスが動いていないことを確認

---

- 下記のアドレスにアクセスしてまだサービスが動いていないことを確認します。

- <http://192.168.1.100/osc2015>

# クラスタを起動する

---

- 以下のコマンドを実行してクラスタを起動します。

★CentOS6向け

```
# initctl start pacemaker.combined
```

★CentOS7向け

```
# systemctl start pacemaker
```

- クラスタを停止するコマンドはこちら

★CentOS6向け

```
# initctl stop pacemaker.combined
```

★CentOS7向け

```
# systemctl stop pacemaker
```

# リソース定義をクラスタに読み込ませる

---

- crmコマンド※を実行してクラスタにリソース定義ファイルを読み込ませます。
  - ※crmコマンドはPacemakerクラスタを操作する運用管理コマンドです。

```
# crm configure load update osc2015.crm
```

リソース定義ファイル

# クラスタの状態を確認する

- クラスタの状態は「crm\_mon」コマンドで確認します。

```
# ssh server01 "crm_mon -fDA1"
Online: [ server01 server02 ]

Resource Group: grpStonith1
  prmHelper1-1 (stonith:external/stonith-helper): Started server02
  prmLibvirt1-2 (stonith:external/libvirt): Started server02
Resource Group: grpStonith2
  prmHelper2-1 (stonith:external/stonith-helper): Started server01
  prmLibvirt2-2 (stonith:external/libvirt): Started server01
Resource Group: grpTrac
  prmFS (ocf::heartbeat:Filesystem): Started server01
  prmVIP (ocf::heartbeat:IPaddr2): Started server01
  prmDB (ocf::heartbeat:pgsql): Started server01
  prmWEB (ocf::heartbeat:apache): Started server01
Clone Set: clnDisk1 [prmDisk1]
  Started: [ server01 server02 ]
Clone Set: clnDisk2 [prmDisk2]
  Started: [ server01 server02 ]
Clone Set: clnPing [prmPing]
  Started: [ server01 server02 ]
```

(省略)

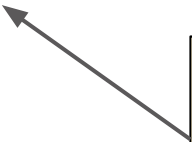
サービスがserver01上で  
「Started」状態になっていることを  
確認します。

# サービスが起動できたことを確認してみよう

---

- WEBブラウザを起動して、下記アドレスにアクセスします。  
Tracに接続できたら無事起動完了です。

- <http://192.168.1.100/osc2015>



このIPはリソース定義  
のIPaddr2リソースに  
設定した仮想IPです。

---

**デモ環境が整ったところで  
STONITHの動きを体験してみましょう！**



# 今回のデモ内容

---

今回のデモでは次に挙げる障害が発生した時に、STONITHによって排他制御が行われてサービスが継続できることを確認します。

1. スプリットブレイン
2. リソース停止故障

---

# スプリットブレイン

# スプリットブレインによるSTONITH実行デモ

---

- server01とserver02をつなぐインターコネクトが切れてスプリットブレインとなる障害のデモを行います。
  - デモ環境は仮想マシンを使用しているため、ホスト側でインターコネクトを通してBridgeインターフェイスを停止させ擬似的に故障を発生させます。

```
(ホスト)# ifdown br2
```

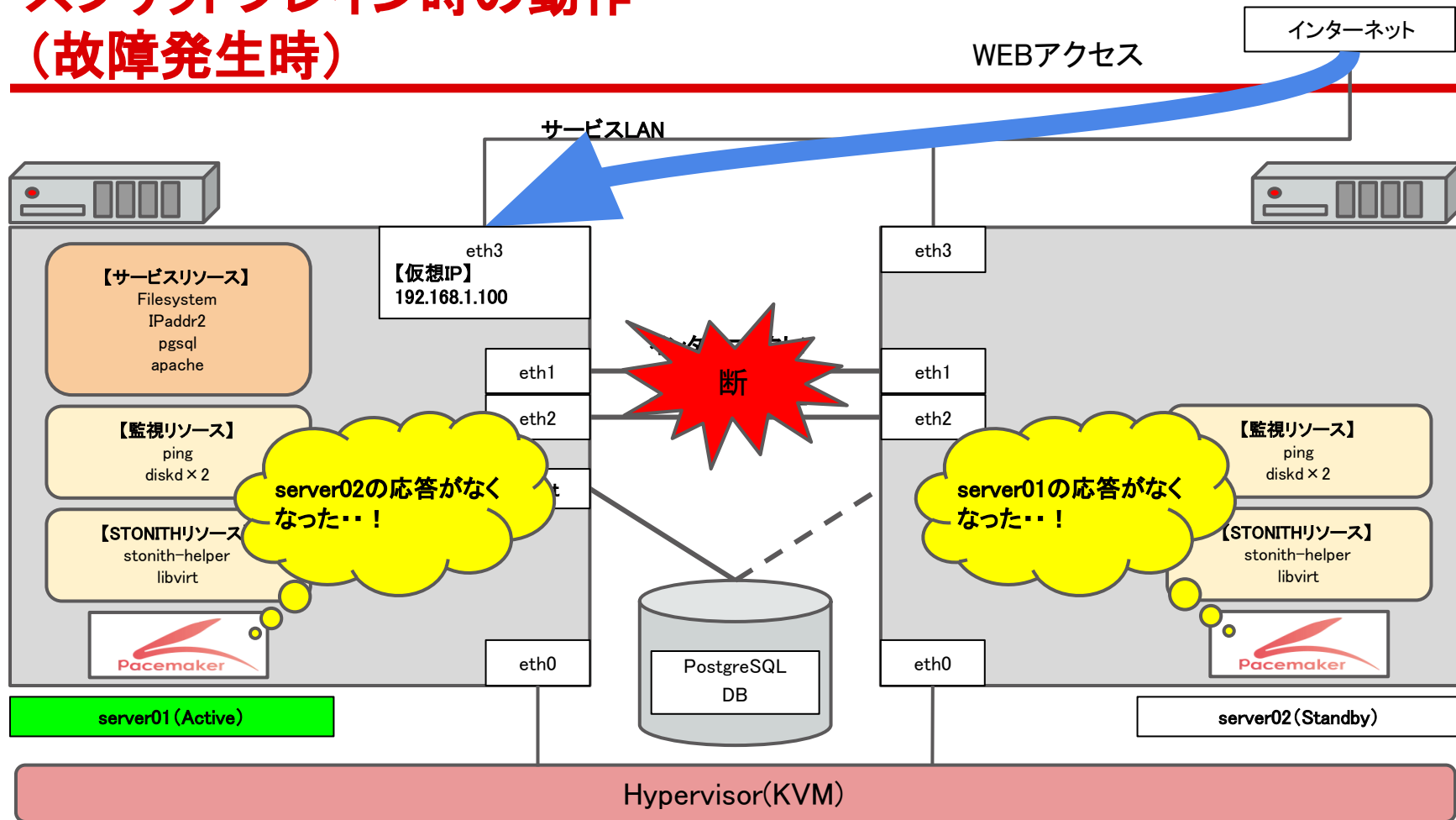
```
(ホスト)# ifdown br3
```

# デモの流れと注目ポイント

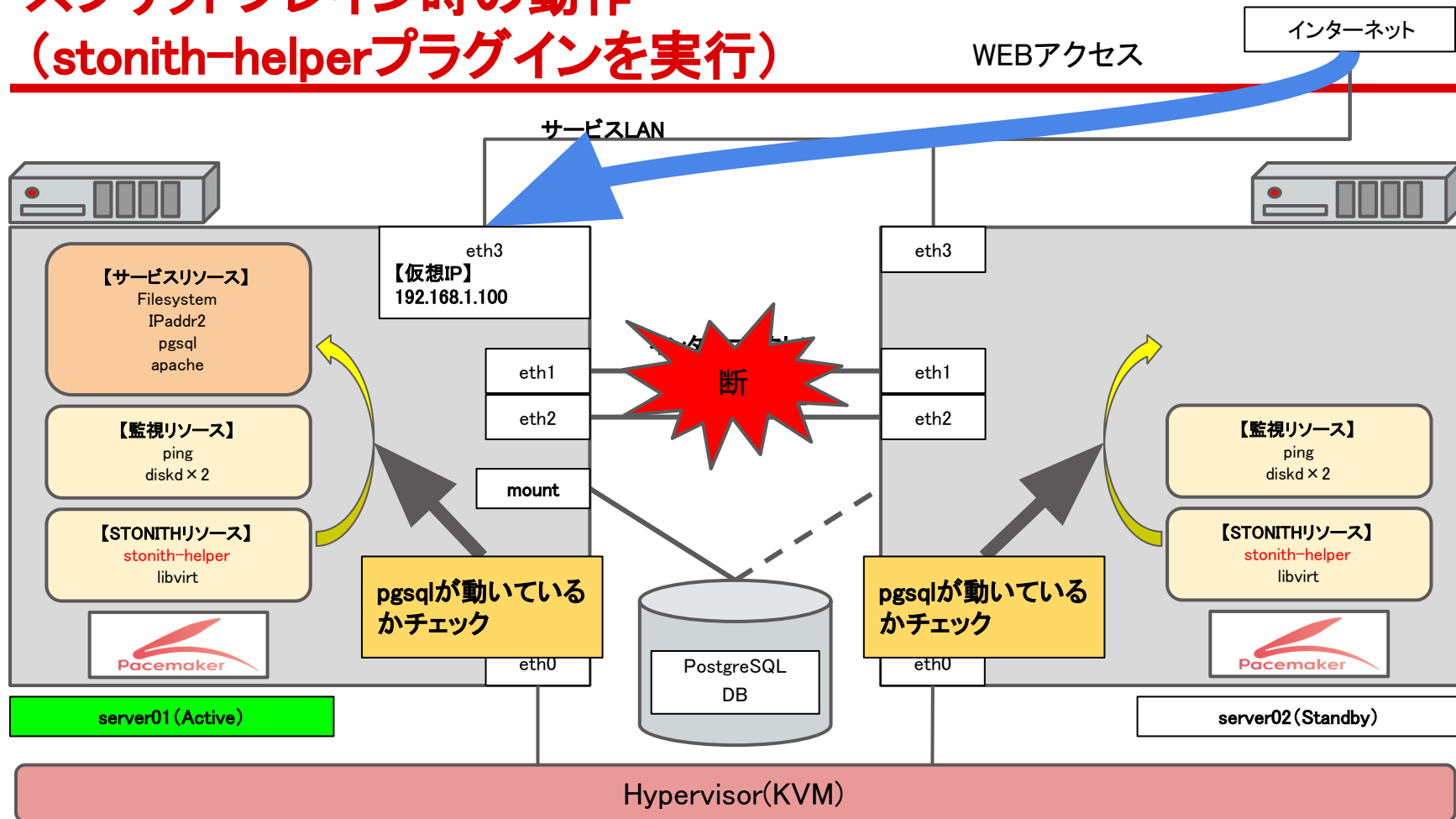
---

- このデモで注目してもらいたいポイントはココ！
  - スプリットブレイン発生後、STONITHによりserver02が再起動される場所
- デモの流れ
  1. ホストマシンのブリッジを停止（擬似故障）
  2. server01とserver02でお互いのノードの状態がわからなくなる
  3. server01とserver02でstonith-helperが実行される
    - 3-1. server01はpgsqlが動いているので、すぐ次のlibvirtプラグインを実行
    - 3-2. server02はpgsqlが動いていないので、sleepを実行しSTONITHされるのを待つ
  4. server01からlibvirtプラグインが実行されserver02が再起動
  5. サービスはserver01で継続される

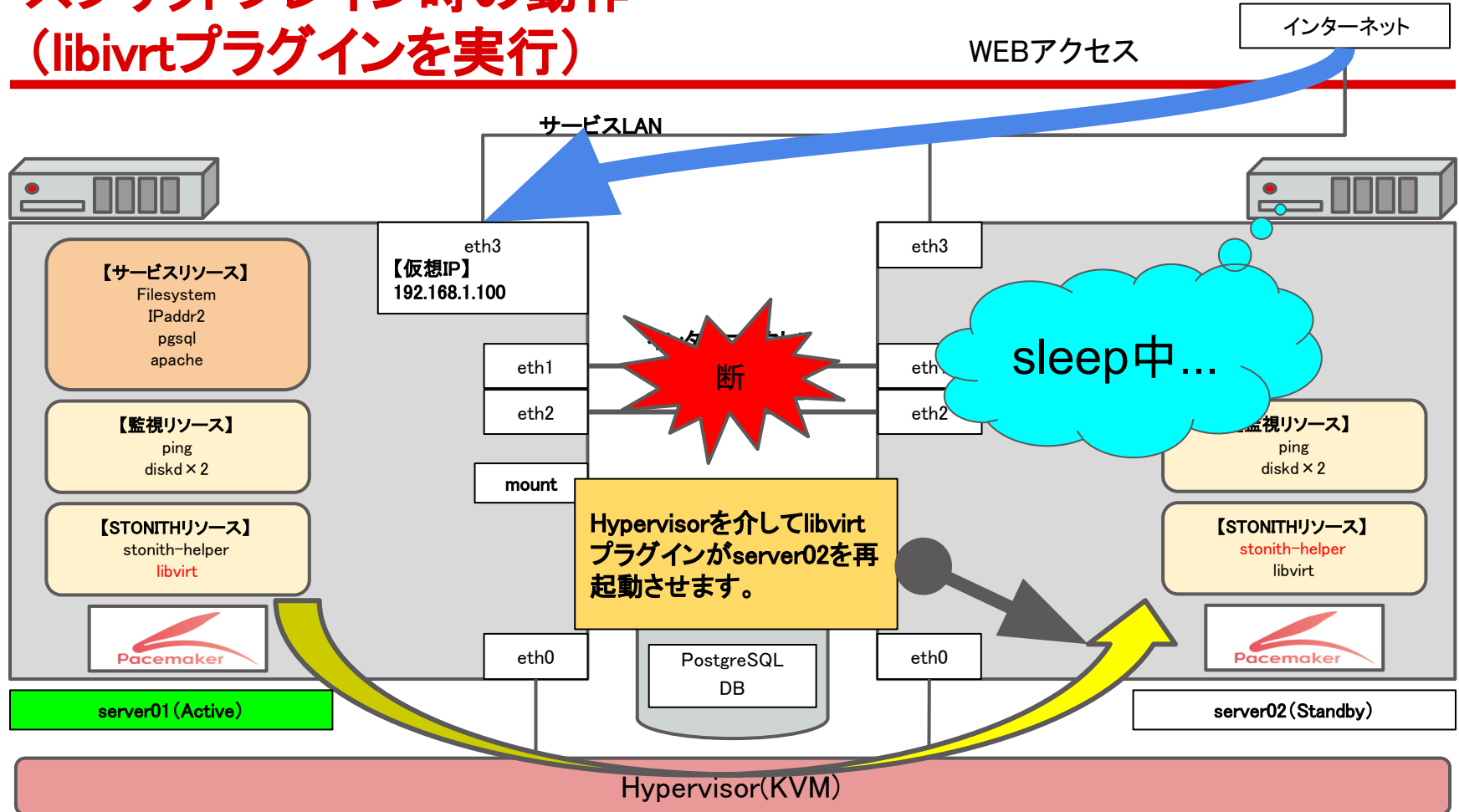
# スプリットブレイン時の動作 (故障発生時)



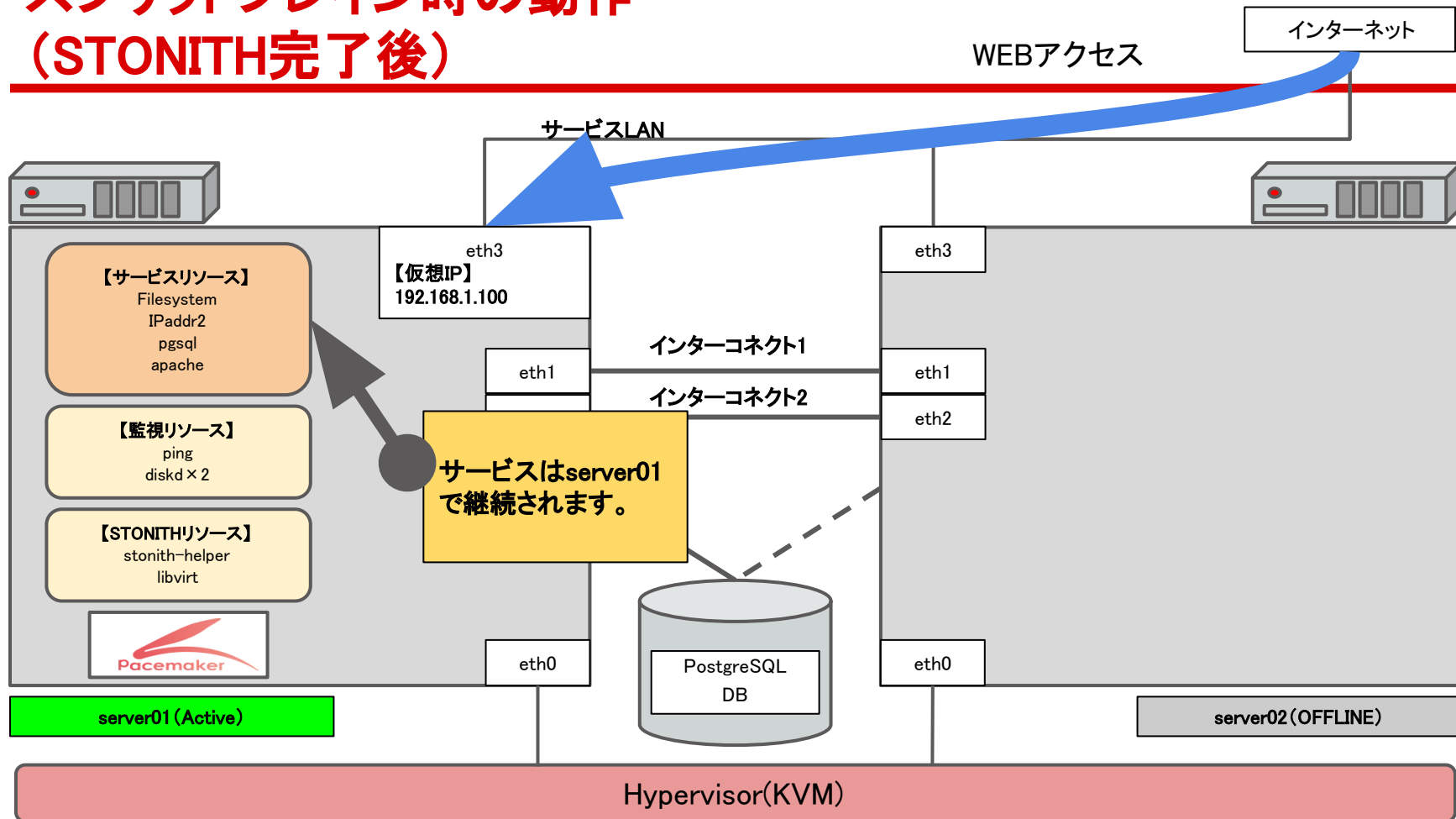
# スプリットブレイン時の動作 (stonith-helperプラグインを実行)



# スプリットブレイン時の動作 (libvirtプラグインを実行)



# スプリットブレイン時の動作 (STONITH完了後)





# スプリットブレインによる STONITH完了後のクラスタ状態

```
# ssh server01 "crm_mon -fDA1"
```

```
Online: [ server01 ]
```

```
OFFLINE: [ server02 ]
```

```
Resource Group: grpStonith2
```

```
  prmHelper2-1 (stonith:external/stonith-helper): Started server01
```

```
  prmLibvirt2-2 (stonith:external/libvirt): Started server01
```

```
Resource Group: grpTrac
```

```
  prmFS (ocf::heartbeat:Filesystem):
```

```
Started server01
```

```
  prmVIP (ocf::heartbeat:IPAddr2):
```

```
Started server01
```

```
  prmDB (ocf::heartbeat:pgsql):
```

```
Started server01
```

```
  prmWEB (ocf::heartbeat:apache):
```

```
Started server01
```

```
Clone Set: clnDisk1 [prmDisk1]
```

```
  Started: [ server01 ]
```

```
  Stopped: [ server02 ]
```

```
Clone Set: clnDisk2 [prmDisk2]
```

```
  Started: [ server01 ]
```

```
  Stopped: [ server02 ]
```

```
Clone Set: clnPing [prmPing]
```

```
  Started: [ server01 ]
```

```
  Stopped: [ server02 ]
```

```
(省略)
```

STONITH完了後、STONITHが  
実行されたノードは、Pacemaker  
が止まるため  
「OFFLINE」となります。

サービスはserver01上で動き続  
けます。

---

# リソース停止故障

# リソース停止故障によるSTONITH実行デモ

---

- 今回のデモではserver01でPostgreSQLプロセスを擬似的に停止させてリソースの停止故障を発生させます。
  - 講演時間の関係上、デモを円滑にすすめるためpgsqlのstopタイムアウト値を短め(20s)に設定しています。

```
(server01)# pkill -STOP postgres
```

# デモの流れと注目ポイント

---

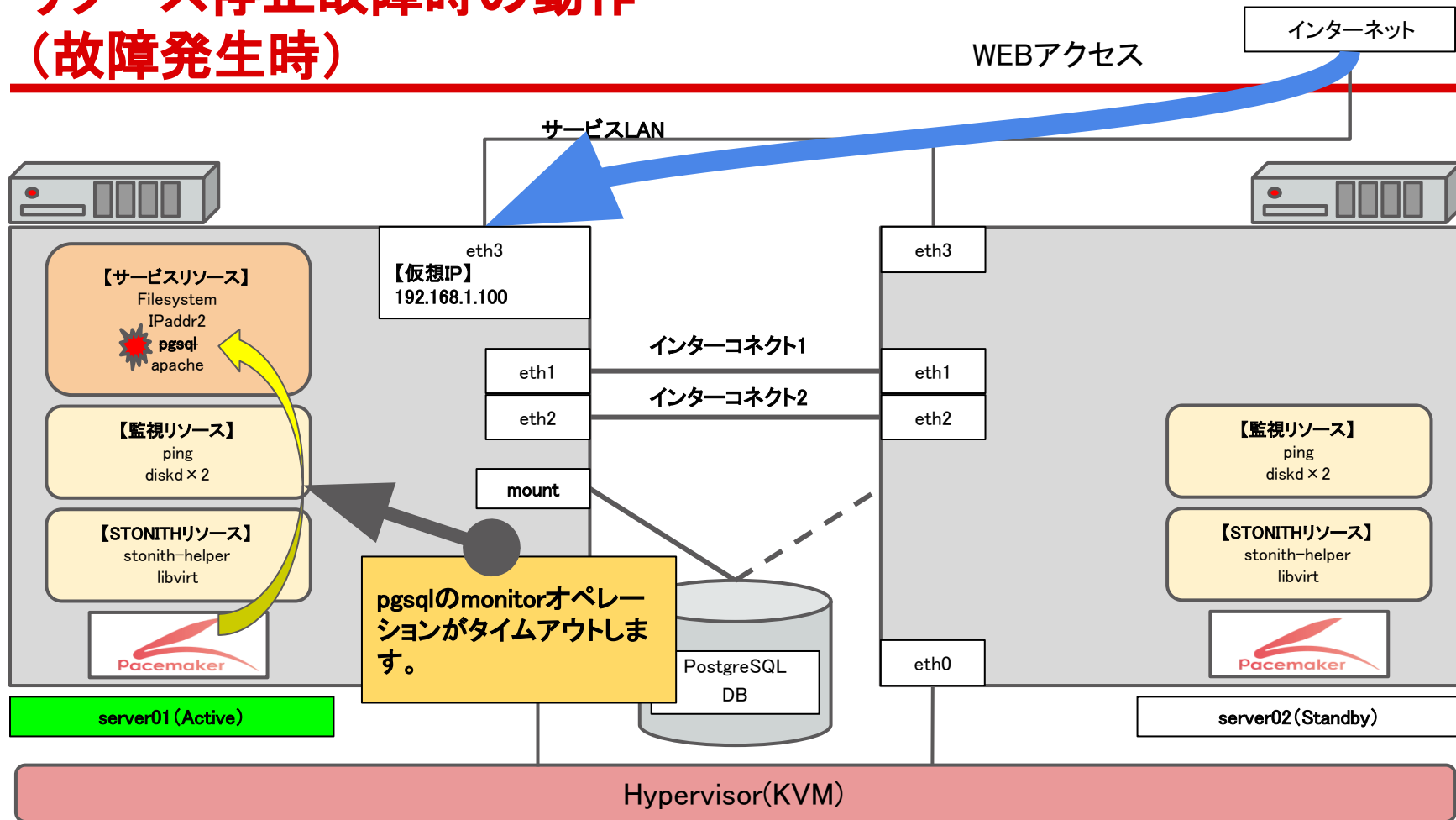
## ■ このデモで注目してもらいたいポイントはココ！

- リソース停止故障を起こしたserver01がSTONITHにより再起動される場所

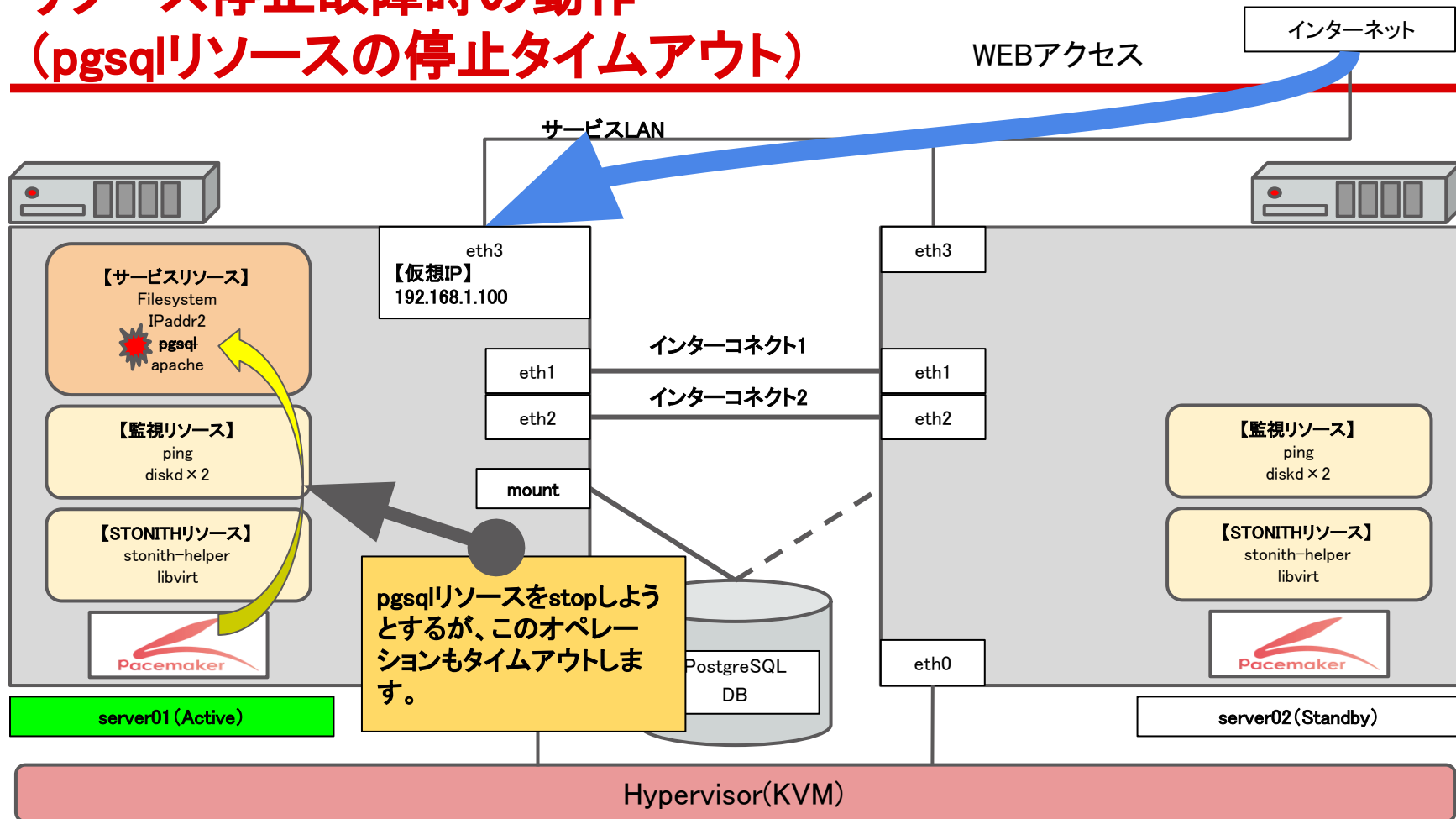
## ■ デモの流れ

1. postgresqlプロセスをSIGSTOP(擬似故障)
2. postgresqlRAでmonitorタイムアウト発生
3. postgresqlをstop開始
4. postgresqlRAでstopタイムアウト発生
5. stopオペレーションのon-fail=fence設定に従い、server01をSTONITH(再起動)
6. server01のSTONITH完了後、サービスはserver02にフェイルオーバー

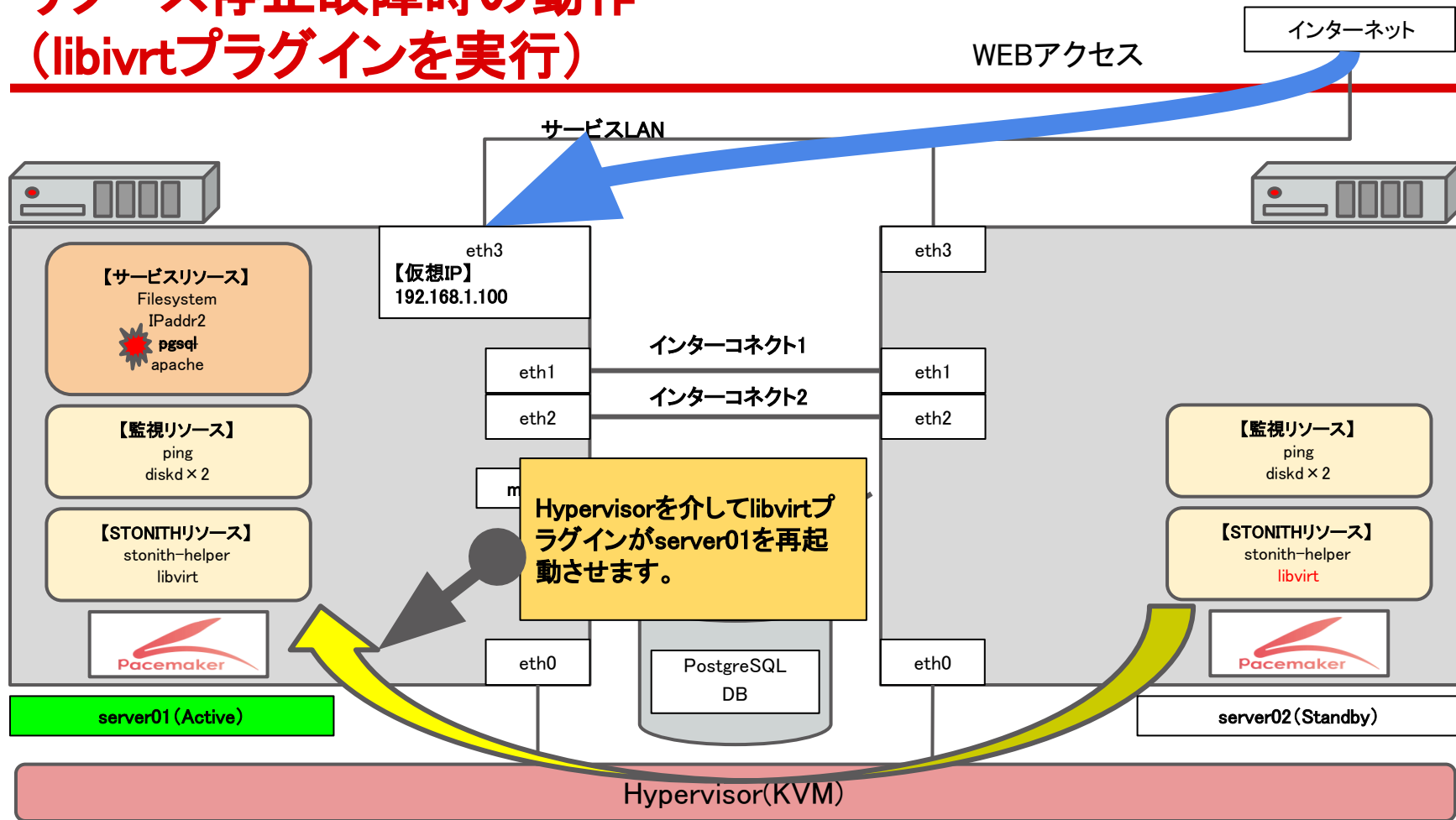
# リソース停止故障時の動作 (故障発生時)



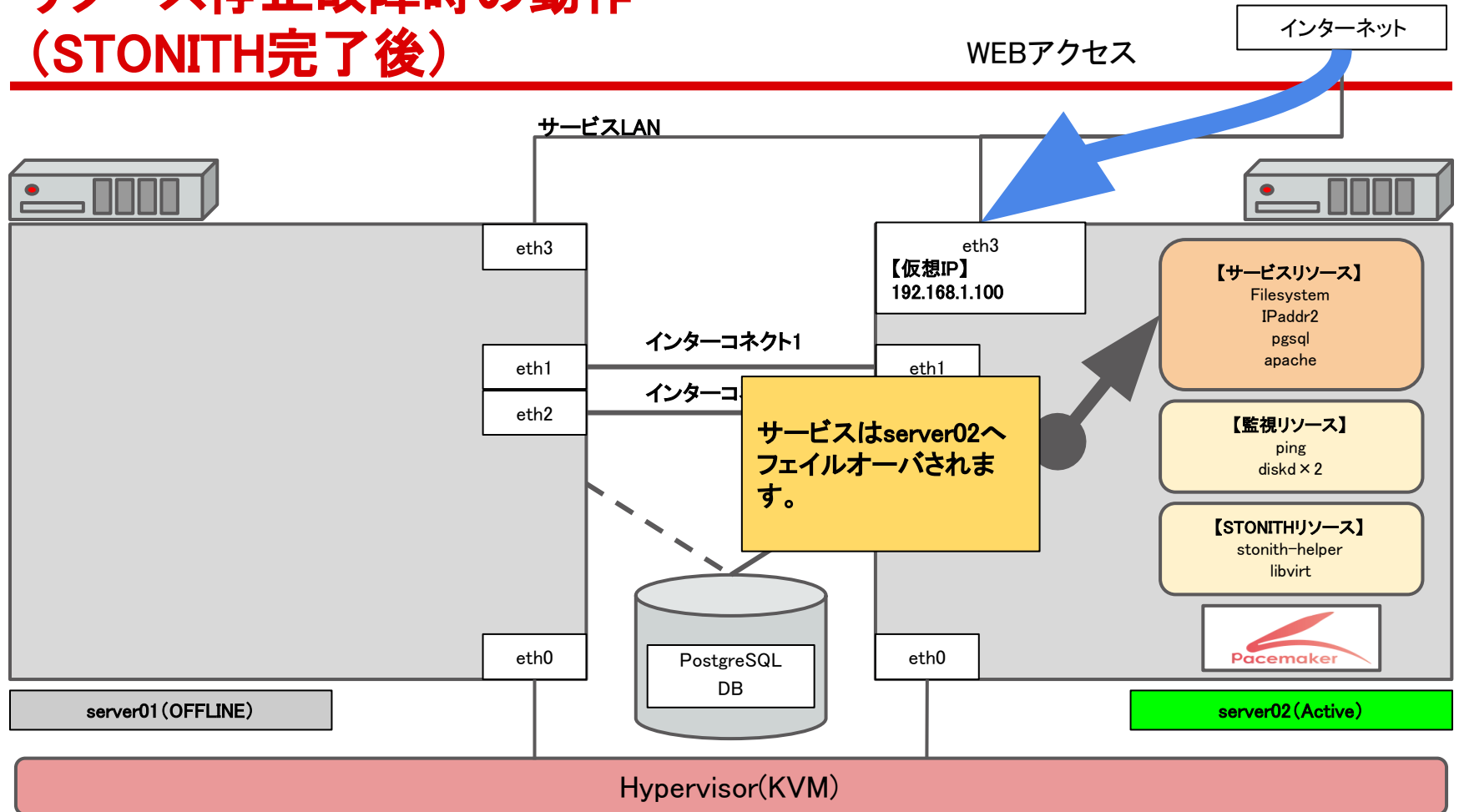
# リソース停止故障時の動作 (pgsqlリソースの停止タイムアウト)



# リソース停止故障時の動作 (libvirtプラグインを実行)



# リソース停止故障時の動作 (STONITH完了後)





# リソース停止故障による STONITH完了後のクラスタ状態

```
# ssh server02 "crm_mon -fDA1"
```

```
Online: [ server02 ]
```

```
OFFLINE: [ server01 ]
```

```
Resource Group: grpStonith1
```

```
  prmHelper1-1  (stonith:external/stonith-helper):  Started server02
```

```
  prmLibvirt1-2 (stonith:external/libvirt):  Started server02
```

```
Resource Group: grpTrac
```

```
  prmFS  (ocf::heartbeat:Filesystem):
```

```
  prmVIP  (ocf::heartbeat:IPAddr2):
```

```
  prmDB  (ocf::heartbeat:pgsql):
```

```
  prmWEB  (ocf::heartbeat:apache):
```

```
Started server02
```

```
Started server02
```

```
Started server02
```

```
Started server02
```

```
Clone Set: clnDisk1 [prmDisk1]
```

```
  Started: [ server02 ]
```

```
  Stopped: [ server01 ]
```

```
Clone Set: clnDisk2 [prmDisk2]
```

```
  Started: [ server02 ]
```

```
  Stopped: [ server01 ]
```

```
Clone Set: clnPing [prmPing]
```

```
  Started: [ server02 ]
```

```
  Stopped: [ server01 ]
```

```
(省略)
```

STONITH完了後、STONITHが  
実行されたノードは、Pacemaker  
が止まるため  
「OFFLINE」となります。

リソースはフェイルオーバーされて  
server02上で起動します。

# さいごに

---

## ■ 本セッションのおさらい

- リソースの停止故障に対応できるのはSTONITHだけ！
- STONITHはスプリットブレインになってもサービスの2重起動を完全に防ぐ！
- Pacemakerは環境に合わせて使える多様なSTONITHプラグインを備えている！

## ■ Pacemakerを使う際にはクラスタをサービス停止から救う「STONITH」の導入を、ぜひご検討ください！

- Pacemakerの日本公式コミュニティとして「Linux-HA Japan」を運営しています。
- 下記サイトにて、Pacemaker関連の最新情報を日本語で発信しています。
  - <http://linux-ha.osdn.jp/wp/>
- Linux-HA JapanではPacemakerのrpmパッケージ※<sup>1</sup>の配布も行っています。
  - ※<sup>1</sup>: Pacemaker関連パッケージをまとめてインストールを簡単にしたりリポジトリパッケージ
  - rpmパッケージダウンロードはこちらから
    - <http://osdn.jp/projects/linux-ha/>

# Linux-HA Japanメーリングリスト

- 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、Corosync、DRBDなど、HAクラスタに関連する話題を歓迎しています！

- MLへの登録はこちらから！

- <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](mailto: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



# 最新情報

---

- 10/16に「Pacemaker-repo-1.1.13-1.1」をリリースしました！
  - 最新のPacemakerを使いたい方は以下からダウンロードしてください。
    - <http://linux-ha.osdn.jp/wp/archives/4154>

# ご清聴ありがとうございました。

---

## 【展示情報】

- 206教室にて「Linux-HA Japan Project」ブース展示中です！



# 付録: デモ環境のリソース定義ファイル(1)

```
### Cluster Option ###
property \
  no-quorum-policy="ignore" \
  stonith-enabled="true" \
  startup-fencing="false"

### Fencing Topology ###
fencing_topology \
  server01: prmHelper1-1 prmLibvirt1-2 \
  server02: prmHelper2-1 prmLibvirt2-2

#### Group Configuration ###
group grpStonith1 \
  prmHelper1-1 \
  prmLibvirt1-2

group grpStonith2 \
  prmHelper2-1 \
  prmLibvirt2-2

primitive prmHelper1-1 stonith:external/stonith-helper \
  params \
    pcmk_reboot_retries=1 \
    hostlist="server01" \
    dead_check_target="192.168.1.10 192.168.11.10 192.168.12.10 192.168.122.40" \
    standby_check_command="/usr/sbin/crm_resource -r prmDB -W | grep -qi 'hostname'" \
    run_online_check="yes" \
    op start interval="0s" timeout="20s" on-fail="restart" \
    op monitor interval="3600s" timeout="20s" on-fail="restart" \
    op stop interval="0s" timeout="15" on-fail="ignore"

primitive prmLibvirt1-2 stonith:external/libvirt \
  params \
    hostlist="server01" \
    hypervisor_uri="qemu+ssh://192.168.122.1/system" \
    op start interval="0s" timeout="60s" on-fail="restart" \
    op monitor interval="3600s" timeout="60s" on-fail="restart" \
    op stop interval="0s" timeout="60s" on-fail="ignore"
```

```
primitive prmHelper2-1 stonith:external/stonith-helper \
  params \
    pcmk_reboot_retries=1 \
    hostlist="server02" \
    dead_check_target="192.168.1.20 192.168.11.20 192.168.12.20 192.168.122.190" \
    standby_check_command="/usr/sbin/crm_resource -r prmDB -W | grep -qi 'hostname'" \
    run_online_check="yes" \
    op start interval="0s" timeout="20s" on-fail="restart" \
    op monitor interval="3600s" timeout="20s" on-fail="restart" \
    op stop interval="0s" timeout="15" on-fail="ignore"

primitive prmLibvirt2-2 stonith:external/libvirt \
  params \
    hostlist="server02" \
    hypervisor_uri="qemu+ssh://192.168.122.1/system" \
    op start interval="0s" timeout="60s" on-fail="restart" \
    op monitor interval="3600s" timeout="60s" on-fail="restart" \
    op stop interval="0s" timeout="60s" on-fail="ignore"

### Group Configuration ###
group grpTrac \
  prmFS \
  prmVIP \
  prmDB \
  prmWEB

### Clone Configuration ###
clone clnPing \
  prmPing

clone clnDiskd1 \
  prmDiskd1

clone clnDiskd2 \
  prmDiskd2
```

# 付録: デモ環境のリソース定義ファイル(2)

```
### Primitive Configuration ###
primitive prmFS ocf:heartbeat:Filesystem \
  params \
    fstype="ext4" \
    run_fsck="force" \
    device="/dev/vdb1" \
    directory="/pgsqldb" \
    op start interval="0s" timeout="60s" on-fail="restart" \
    op monitor interval="20s" timeout="40s" on-fail="restart" \
    op stop interval="0s" timeout="60s" on-fail="fence"

primitive prmVIP ocf:heartbeat:IPAddr2 \
  params \
    ip="192.168.1.100" \
    nic="eth3" \
    cidr_netmask="24" \
    op start interval="0s" timeout="20s" on-fail="restart" \
    op monitor interval="10s" timeout="20s" on-fail="restart" \
    op stop interval="0s" timeout="20s" on-fail="fence"

primitive prmDB ocf:heartbeat:pgsql \
  params \
    pgctl="/usr/pgsql-9.4/bin/pg_ctl" \
    psqldata="/usr/pgsql-9.4/bin/psql" \
    pgdata="/pgsqldb/data" \
    start_opt="-p 5432" \
    pgdba="postgres" \
    pgport="5432" \
    pgdb="template1" \
    op start interval="0s" timeout="120s" on-fail="restart" \
    op monitor interval="10s" timeout="10s" on-fail="restart" \
    op stop interval="0s" timeout="20s" on-fail="fence"

primitive prmWEB ocf:heartbeat:apache \
  op start interval="0s" timeout="40s" on-fail="restart" \
  op monitor interval="10s" timeout="20s" on-fail="restart" \
  op stop interval="0s" timeout="60s" on-fail="fence"
```

```
primitive prmPing ocf:pacemaker:ping \
  params \
    name="default_ping_set" \
    host_list="192.168.1.1" \
    multiplier="100" \
    attempts="2" \
    timeout="2" \
    debug="true" \
    op start interval="0s" timeout="60s" on-fail="restart" \
    op monitor interval="10s" timeout="60s" on-fail="restart" \
    op stop interval="0s" timeout="60s" on-fail="ignore"

primitive prmDisk1 ocf:pacemaker:diskd \
  params \
    name="diskcheck_status" \
    device="/dev/vdb" \
    options="-e -t 70" \
    interval="10" \
    dampen="2" \
    op start interval="0s" timeout="60s" on-fail="restart" \
    op monitor interval="10s" timeout="60s" on-fail="restart" \
    op stop interval="0s" timeout="60s" on-fail="ignore"

primitive prmDisk2 ocf:pacemaker:diskd \
  params \
    name="diskcheck_status_internal" \
    device="/dev/vda" \
    options="-e" \
    interval="10" \
    dampen="2" \
    op start interval="0s" timeout="60s" on-fail="restart" \
    op monitor interval="10s" timeout="60s" on-fail="restart" \
    op stop interval="0s" timeout="60s" on-fail="ignore"
```



# 付録: デモ環境のリソース定義ファイル(3)

```
### Resource Location ###
location rsc_location-grpTrac-1 grpTrac \
    rule 200: #uname eq server01 \
    rule 100: #uname eq server02 \
    rule -INFINITY: not_defined default_ping_set or default_ping_set lt 100 \
    rule -INFINITY: not_defined diskcheck_status or diskcheck_status eq ERROR \
    rule -INFINITY: not_defined diskcheck_status_internal or diskcheck_status_internal eq ERROR

### Resource Colocation ###
colocation rsc_colocation-grpTrac-clnPing-1 INFINITY: grpTrac clnPing
colocation rsc_colocation-grpTrac-clnDiskd1-2 INFINITY: grpTrac clnDiskd1
colocation rsc_colocation-grpTrac-clnDiskd2-3 INFINITY: grpTrac clnDiskd2

### Resource Order ###
order rsc_order-clnPing-grpTrac-1 0: clnPing grpTrac symmetrical=false
order rsc_order-clnDiskd1-grpTrac-2 0: clnDiskd1 grpTrac symmetrical=false
order rsc_order-clnDiskd2-grpTrac-3 0: clnDiskd2 grpTrac symmetrical=false

### Resource Defaults ###
rsc_defaults resource-stickiness="INFINITY" \
    migration-threshold="1"
```