



# HAクラスタで PostgreSQLを高可用化 (前編) ～Pacemaker入門編～

2012年5月26日 第23回しくみ+アプリケーション勉強会

Linux-HA Japan プロジェクト  
田中 崇幸



# はじめに...

DB屋さんではないので、DBに関する詳しい話はできません...

PostgreSQLをHAクラスタ化するお話をしますが、PostgreSQL9.1 の Streaming Replication機能を用いた話はしません...。(後編でお話する予定です)

インストール方法含めて、HAクラスタの超基本からお話します。

擬人化の話はしません。

# 自己紹介

- 名前: 田中崇幸 (Takayuki Tanaka)
  - Twitter: @tanakacchi21
- 所属: Linux-HA Japanプロジェクト
  - コミュニティ旗揚時のメンバー
  - Pacemaker普及促進のため、オープンソースカンファレンスでの講演等で全国行脚中
- 趣味: マラソン
  - フルマラソン(42.195km)で昨年念願のサブ3を達成した市民マラソンランナー
  - 1週間前に野辺山ウルトラマラソン(100km)に参加し、11時間52分で完走



# 本日のお話

- ① HAクラスタって何？ Pacemakerって何？
- ② Pacemakerのコンポーネント構成
- ③ 本日のPacemakerデモ環境
- ④ インストール・設定デモします！
- ⑤ いろいろ故障デモします！
- ⑥ Linux-HA Japanについて



①

HAクラスタって何？  
Pacemakerって何？



そもそもクラスタって何？

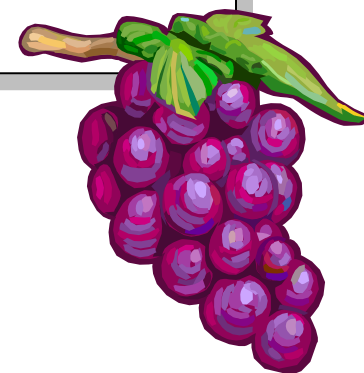


コンピュータの世界で

# クラスタということ

複数のコンピュータを結合し、  
果実・花などの房のように  
ひとまとまりとしたシステムのこと

(Wikipediaより)



HAクラスタ

HPC並列クラスタ

負荷分散クラスタ

....

さらに**HA**クラスタっていうと・・・





# High Availability = 高可用性

つまり 壊れにくさ

一台のコンピュータでは得られない  
高い信頼性を狙うために、  
複数のコンピュータを結合し、  
ひとまとまりとしたシステムのこと

# 今回ご紹介する



は

このHAクラスタ  
という部類のソフトウェアで、  
実績のある**「Heartbeat」**と  
呼ばれていたHAクラスタの  
後継ソフトウェアです。



ここで本日の客層を知るために  
皆さんに質問させてください。





# Pacemaker は すでにご存知ですか？



同じくHAクラスタである  
Heartbeat(バージョン1 or 2)は  
知っていますか？

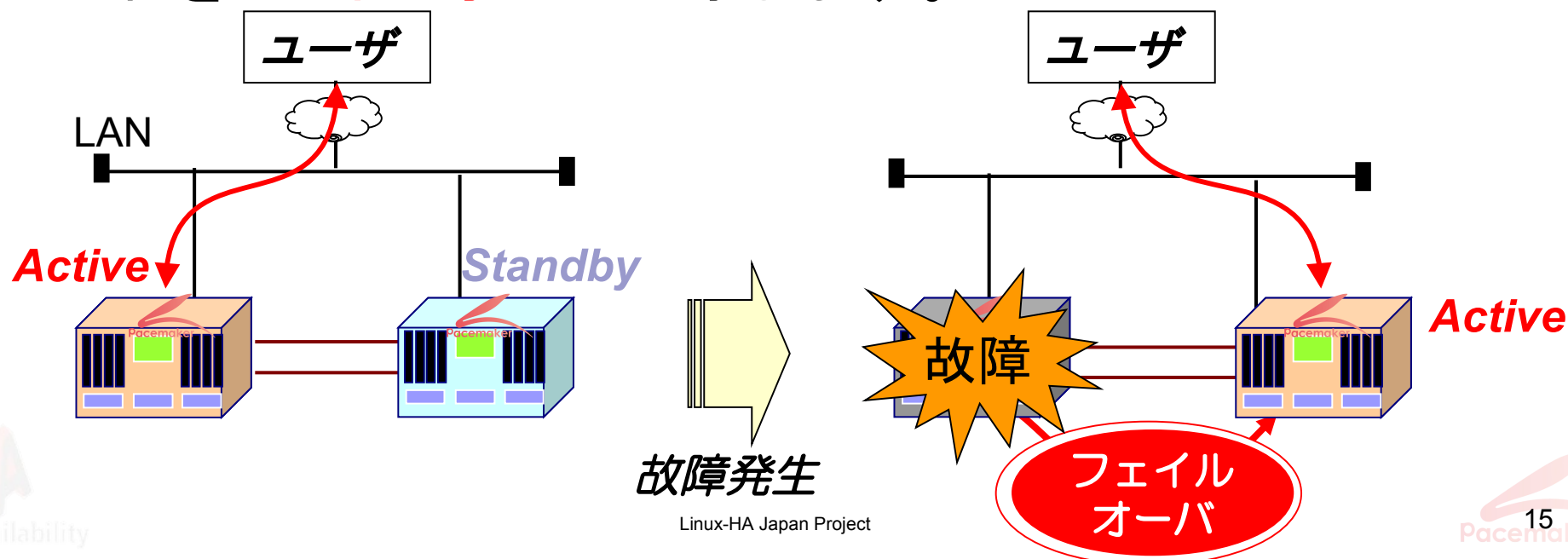


「Pacemaker」は「Heartbeat」の  
後継というだけあって、  
密接な関係があります。  
詳細は後ほどお話しします。

# 基本構成

## Active/Standby (1+1) 構成

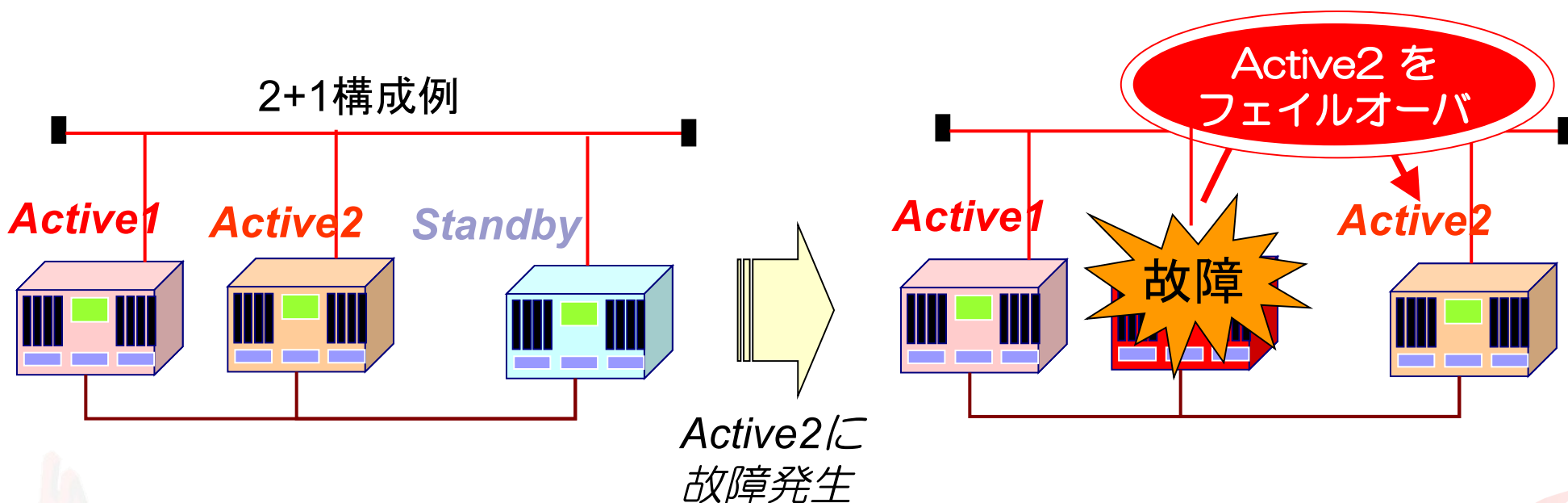
- 通常はActiveノードと呼ばれるサーバでサービスを提供します。
- Activeノードが故障した場合は、StandbyノードがActiveになりサービスを引き継ぎます。  
これを**フェイルオーバー**と呼びます。



# Pacemakerでは複数台構成も可能

※ Heartbeatバージョン1では実現できませんでした

- 複数台のActiveノードや、複数台のStandbyノードを設定可能です。  
(N+M構成)





複数台 (N+M) 構成が  
可能ということは、  
こんな構成も可能か？？

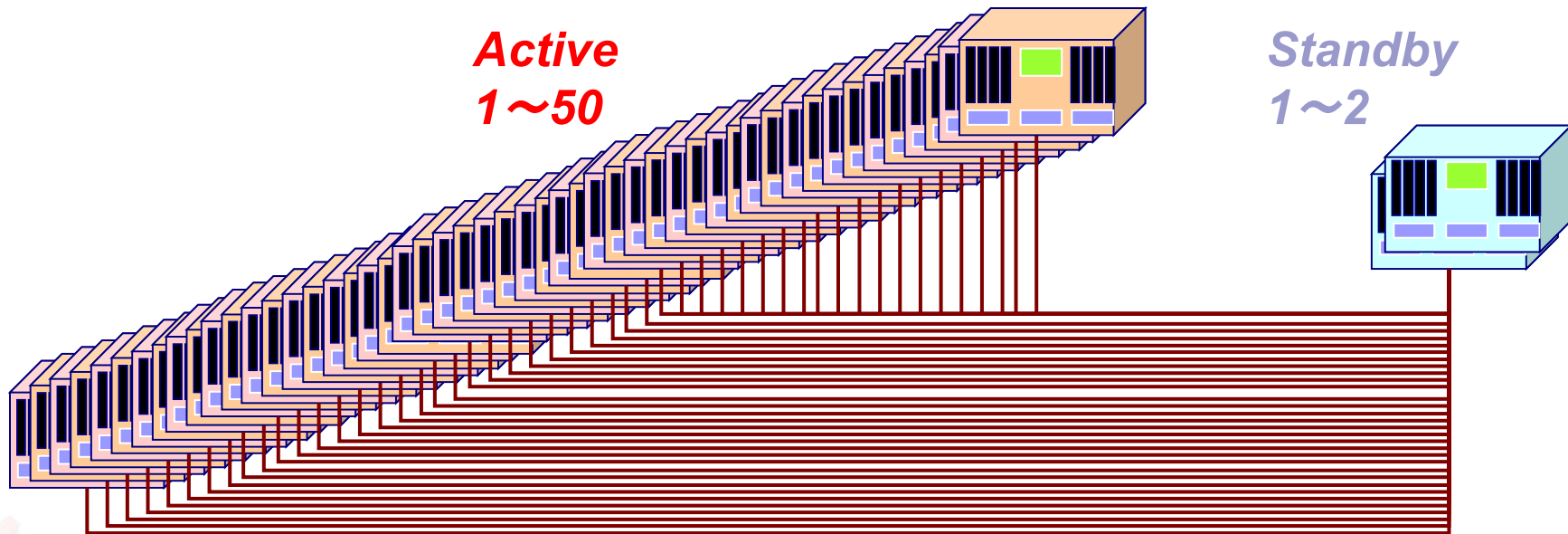


適切なHAクラスタ設計は  
十分検討しましょう！！

# 50+2構成は…

## ■ これはムチャ構成です..(泣)

実現しても、50台いっぺんにフェイルオーバーしてきたら  
大変なことになります。



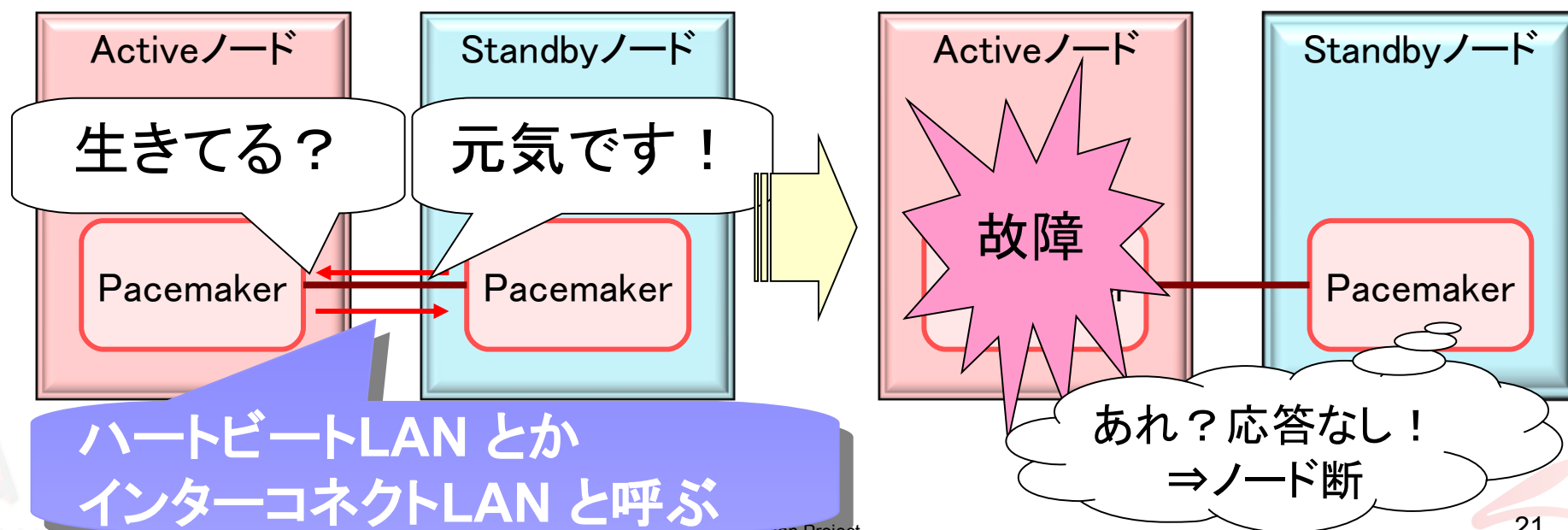
今回は、  
話を単純にするために、  
**Active / Standby (1+1構成)**  
の構成を例に話を進めます。

# Pacemaker 基本機能は主にこの2つ

1. ノード監視
2. リソース制御

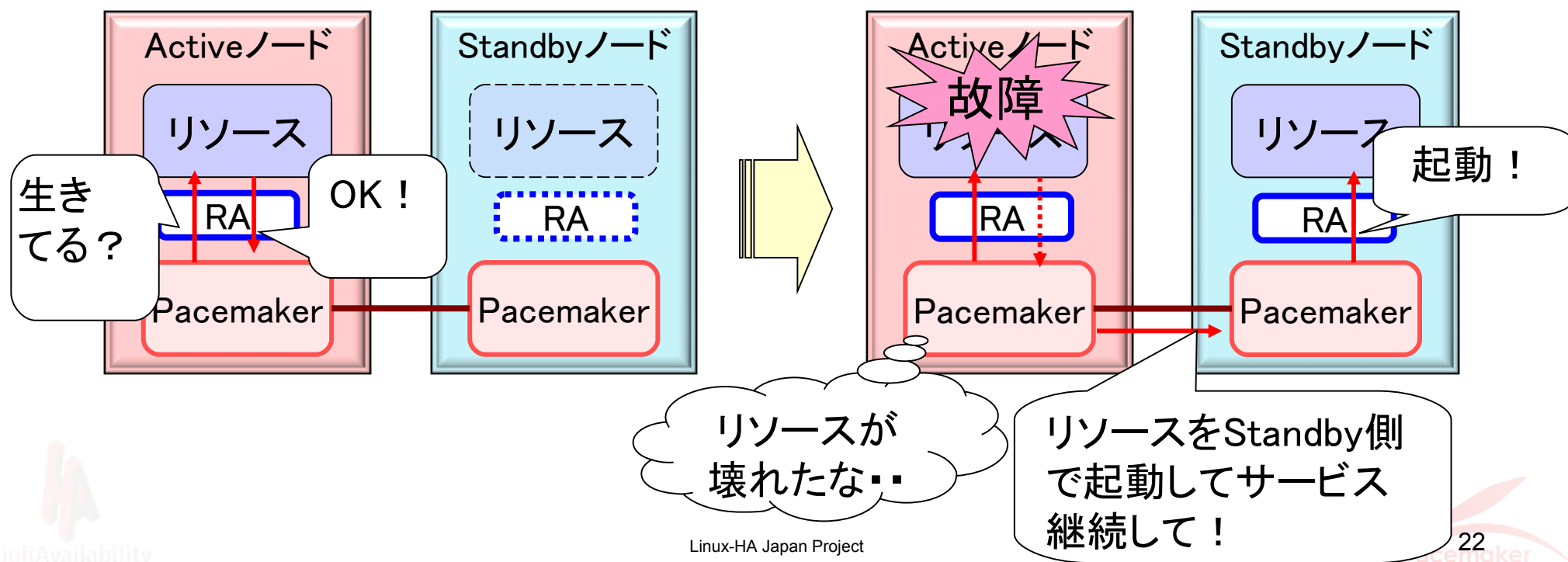
# 基本動作1：ノード監視

- 相手ノードの生死を確認するために、一定間隔で相手ノードを監視します。(ハートビート通信と呼ぶ)
- 相手ノードと通信できなくなった場合に、相手はダウンしたと判断し、フェイルオーバーさせるなどの**クラスタ制御**を行います。



# 基本動作2:リソース制御

- リソースと呼ばれる物をリソースエージェント(RA)を介して起動(start)、停止(stop)、監視(monitor)します。
- リソースが故障した場合にはフェイルオーバーといった**リソース制御**を行います。

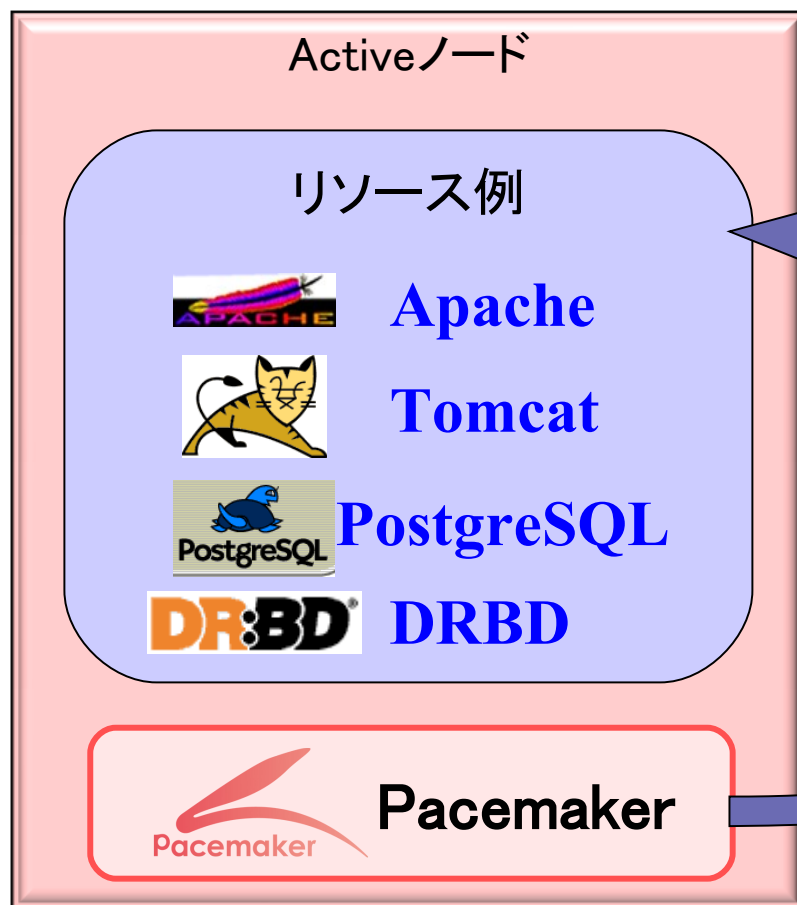


# 「リソース」って何？

Pacemakerではよく出てくる言葉なのでおぼえてください！

- ・ ノード間でサービスを引き継ぐために制御が必要なもの
- ・ サービスの故障を検知するのに監視が必要なもの

簡単に言うと、Pacemakerが起動、停止、監視するものがリソースになります。



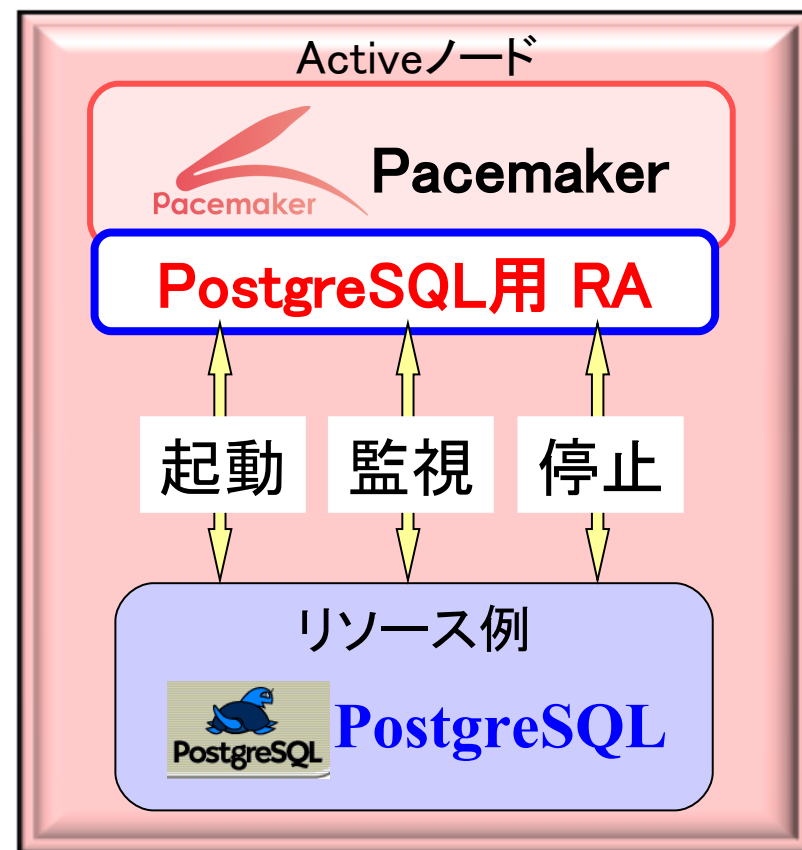
Pacemaker から見ると、PostgreSQL などのアプリケーションは、「リソース」となります。



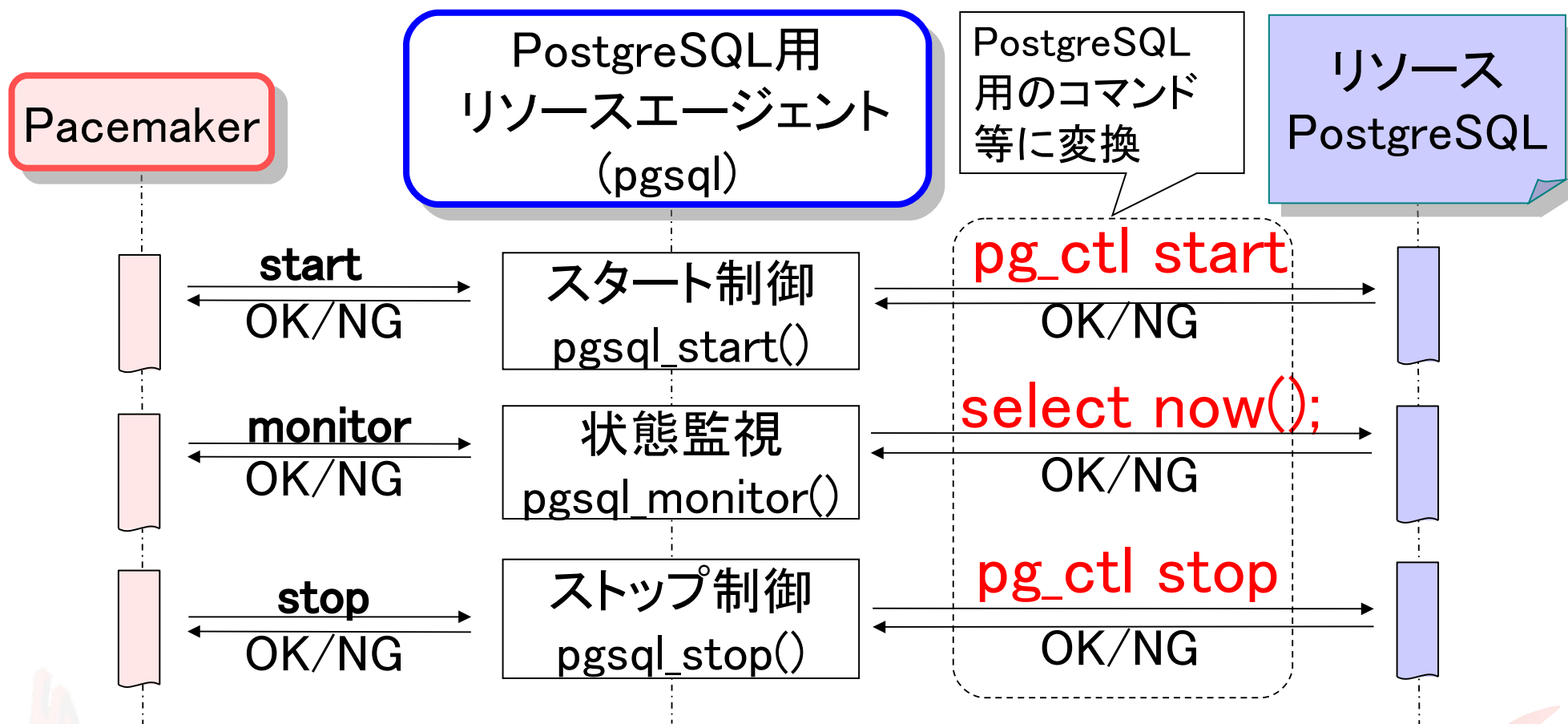
# 「リソースエージェント(RA)」とは？

リソースとPacemakerを仲介するプログラム  
主にシェルスクリプトで記述

Pacemakerは、  
リソースエージェントに対し  
リソースの  
起動(start)  
停止(stop)  
監視(monitor)  
を指示します。



# Pacemaker と PostgreSQLリソース エージェントの関係



# pgsql リソースエージェント実装例

## 起動(start)処理の抜粋

```
#!/bin/sh
```

```
pgsql_start() {
```

←PostgreSQLの起動のメイン関数

```
    if pgsql_status; then
    then
```

←PostgreSQLプロセスの存在を確認

←PostgreSQLプロセスがいれば

```
        ocf_log info "PostgreSQL is already running. PID=`cat $PIDFILE`"
```

```
        return $OCF_SUCCESS
```

←PostgreSQLは動作していると判断

```
    fi
```

\$OCF\_SUCCESSはPacemakerで定義済みの変数

```
    runasowner "$OCF_RESKEY_pgctl $pgctl_options start"
```

↑

\$OCF\_RESKEY\_pgctl、\$pgctl\_options は外部パラメータで  
設定することが可能

RAでのpg\_ctlのデフォルトは、*/usr/bin/pg\_ctl*

```
}
```

## 停止 (stop) 処理の抜粋

```
#!/bin/sh
```

```
pgsql_stop() {
```

```
    if ! pgsql_status
```

```
    then
```

```
        return $OCF_SUCCESS
```

```
    fi
```

```
    runasowner "$OCF_RESKEY_pgctl -D $OCF_RESKEY_pgdata stop -m fast"
```

```
    if pgsql_status
```

```
    then
```

```
        runasowner "$OCF_RESKEY_pgctl -D $OCF_RESKEY_pgdata stop -m immediate"
```

```
    fi
```

```
}
```

←PostgreSQLの停止のメイン関数

←PostgreSQLプロセスの存在を確認

←PostgreSQLプロセスがいなければ

←PostgreSQLは停止していると判断

↑  
fastモードで停止を実行

←再度PostgreSQLプロセスの存在を確認

←まだPostgreSQLプロセスがいれば

↑  
immediateモードで停止を実行

## 監視(monitor)処理の抜粋

```
#!/bin/sh
```

```
pgsql_monitor() {
```

←PostgreSQLの監視のメイン関数

```
    if ! pgsql_status
```

←PostgreSQLプロセスの存在を確認

```
    then
```

←PostgreSQLプロセスがいなければ

```
        ocf_log info "PostgreSQL is down"
```

```
        return $OCF_NOT_RUNNING
```

←PostgreSQLは停止していると判断

```
    fi
```

\$OCF\_NOT\_RUNNINGはPacemakerで定義済みの変数

```
    runasowner -q $loglevel "$OCF_RESKEY_psql $psql_options -c  
' $OCF_RESKEY_monitor_sql' "
```



実際にSQLを実行してPostgreSQLの正常性を確認  
RAのデフォルトは、***select now();*** を実行

```
}
```

## ● 各パラメータ:

```
pgctl="/usr/pgsql-9.1/bin/pg_ctl"  
pgdata="/var/lib/pgsql/9.1/data"  
pgport="5432"
```

```
psql="/usr/pgsql-9.1/bin/psql"  
pgdba="postgres"  
pgdb="template1"
```

## ● 起動:

```
su postgres -c 'cd /var/lib/pgsql/9.1/data; /usr/pgsql-9.1/bin/pg_ctl  
-D /var/lib/pgsql/9.1/data -l /dev/null start '
```

## ● 監視:

```
su postgres -c 'cd /var/lib/pgsql/9.1/data; kill -s 0 `head -n 1  
/var/lib/pgsql/9.1/data/postmaster.pid` >/dev/null 2>&1 '
```

```
su postgres -c 'cd /var/lib/pgsql/9.1/data; /usr/pgsql-9.1/bin/psql -p  
5432 -U postgres template1 -c '¥' 'select now();'¥' ' '
```

## ● 停止:

```
su postgres -c 'cd /var/lib/pgsql/9.1/data; /usr/pgsql-9.1/bin/pg_ctl  
-D /var/lib/pgsql/9.1/data stop -m fast '
```

```
su postgres -c 'cd /var/lib/pgsql/9.1/data; /usr/pgsql-9.1/bin/pg_ctl  
-D /var/lib/pgsql/9.1/data stop -m immediate'
```

Pacemakerでは、様々なリソースエージェントが用意されています。

### 標準リソースエージェントの一例

目的	リソース	リソースエージェント名 ( <code>/usr/lib/ocf/resource.d/</code> に存在)
アプリケーション動作 の引継ぎ	PostgreSQL Apache, Tomcat	pgsql apache, tomcat,
データ引継ぎ	ファイルシステム のマウント	Filesystem (複数のファイルシステムに対応)
IPの引継ぎ	仮想IPアドレス付 与	IPaddr, IPaddr2
異常監視	ディスク監視 ネットワーク監視	diskd (Linux-HA Japan提供) pingd

# リソースエージェントは自分でも作れます！

```
#!/bin/sh
: ${OCF_FUNCTIONS_DIR:=${OCF_ROOT}/lib/heartbeat}
. ${OCF_FUNCTIONS_DIR}/ocf-shellfuncs
```

```
start処理() {
}
stop処理() {
}
monitor処理 {
}
meta-data処理() {
}
validate-all処理() {
}
```


```
case $1 in
  start)    start処理();;
  stop)     stop処理();;
  monitor)  monitor処理();;
esac
```

通常のシェルスクリプトの記述方法ですが、いくつか必須のパラメータ呼び出しに対する処理と、定義済みの戻り値を返すように実装する必要があります。

リソース開始・監視・停止の処理

シェルに渡されるパラメータを元にRA処理を振り分け





HAクラスタで  
怖い物って  
何か知ってますか？



それは  
スプリットブレイン。



これによって  
起こる恐ろしいこと。



# こんな状態。

あいつ  
死んだな..

あいつ  
死んだな..

切断

サービス起動しまーす

サービス起動中

mount

mount

共有ディスク

ダブルマウント

データ破壊

が！これを防ぐのが



Pacemakerの

**STONITH** と **sfex** 。



# STONITHって？




# Shoot-The-Other-Node-In-The-Head

制御が利かないサーバをHAクラスタから「強制的に離脱」させる機能です。

フェンシング

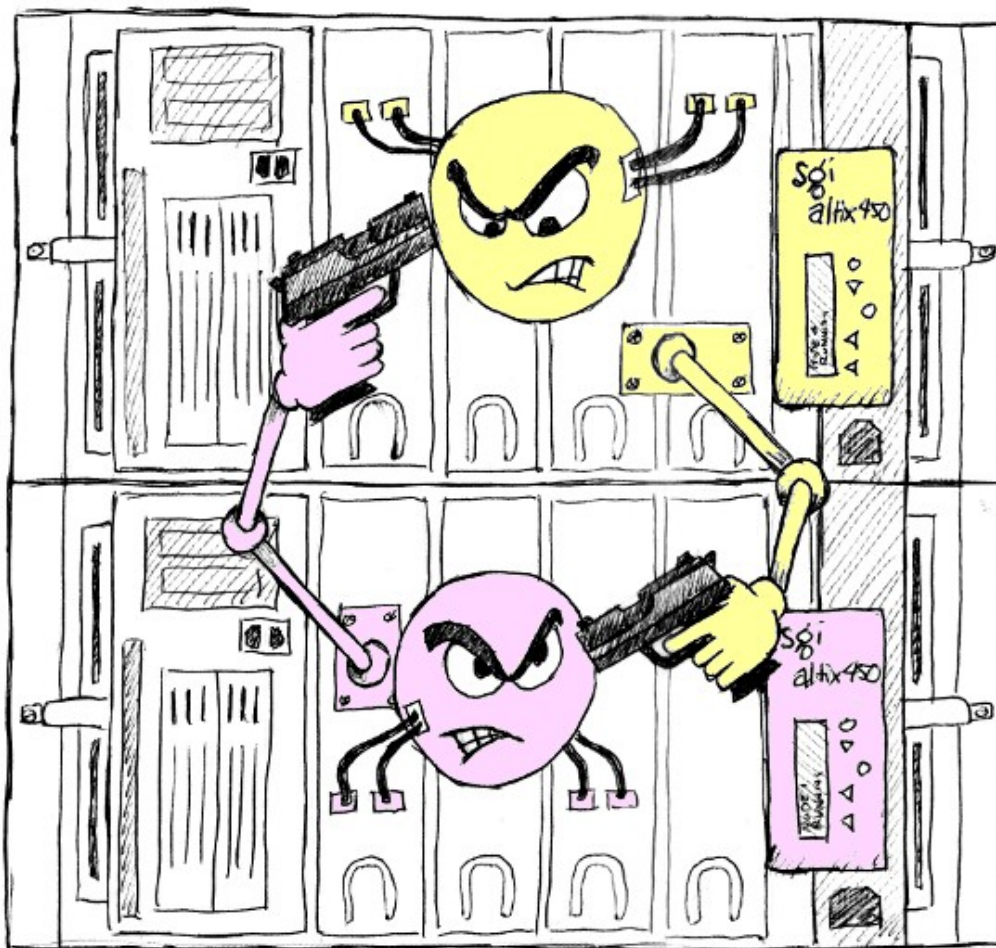




絵で書くとこんなイメージ..



# 自殺機能ではなく、他殺機能です。



DON'T ANYBODY MOVE ...

<http://ourobengr.com/ha>

Linux-HA Japan Project

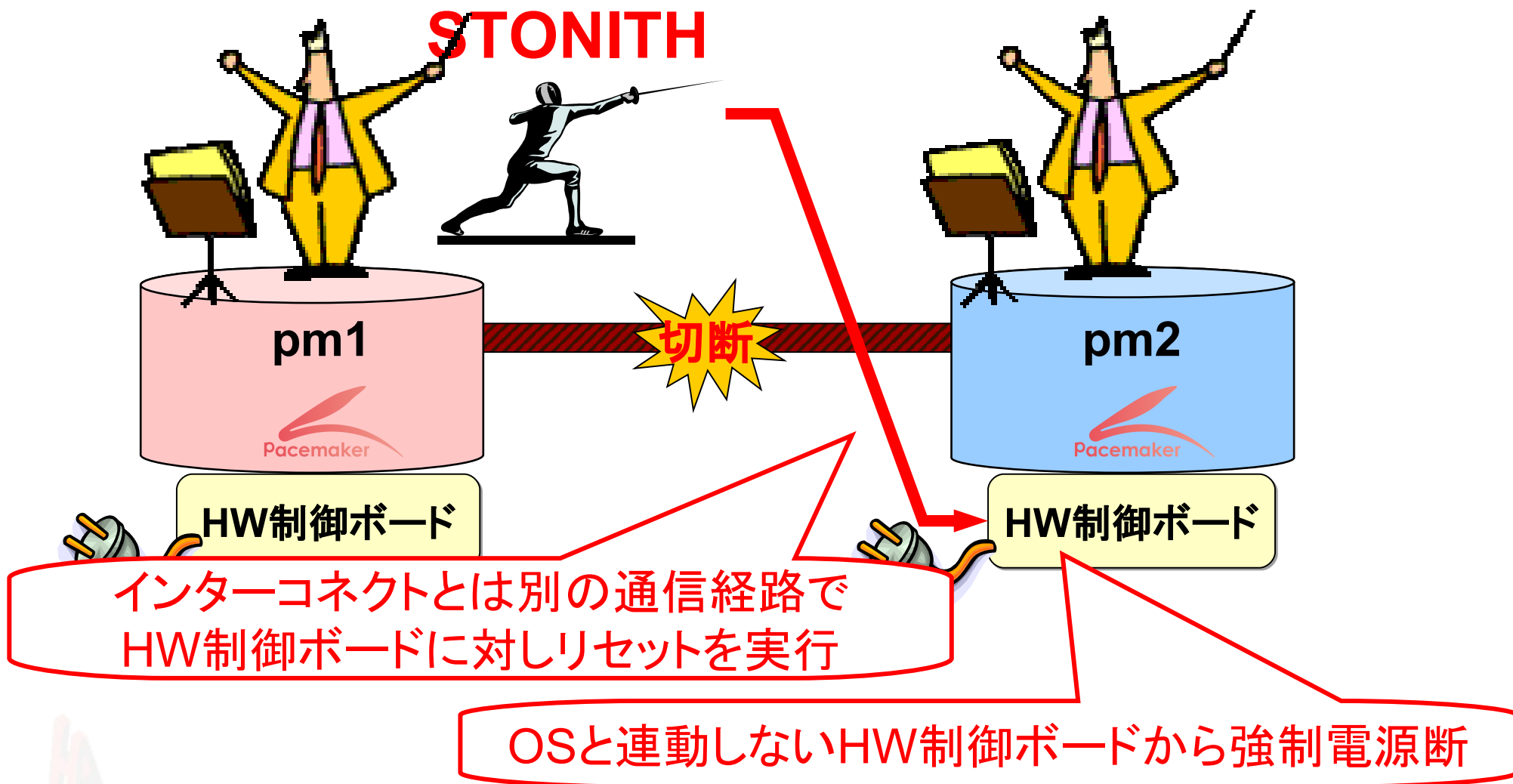


HighAvailability

では、  
具体的にどうやって  
強制離脱させるの？



# STONITH実行例(スプリットブレイン時)



# HW制御ボードは例えばこんなの。

高価なサーバ  
のみ搭載！？

## ■ DELL社 【iDRAC6】

### □ Integrated Dell Remote Access Controller 6

- PowerEdge R610 等に標準搭載
- IPMI 2.0 対応

## ■ HP社 【iLO3】

### □ Integrated Lights-Out 3

- ProLiant DL360 G7 等に標準搭載
- IPMI 2.0 対応



▲ iLO3

# 安価なサーバでも搭載されていますので、 STONITHは敷居は高くありません！

## ■ 例) HP MicroServer

- MicroServerにリモート管理オプションのリモートアクセスカードキットを搭載すればSTONITH機能は使用可能です。



▲ HP MicroServer



▲ リモートアクセスカード  
(IPMI 2.0 対応)

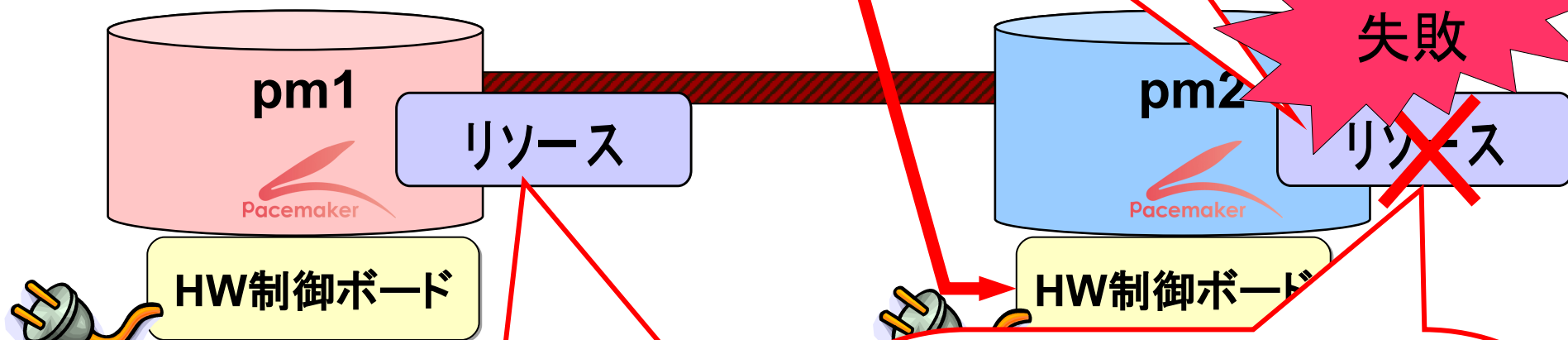


ちなみに  
STONITHには  
こんな機能もあります。



# STONITH実行例(リソース停止失敗時)

## STONITH



リソース故障

停止処理  
失敗

STONITH成功後、  
リソースがフェイルオーバ

リソース故障時、フェイル  
オーバーしようとして、  
リソース停止失敗または  
停止タイムアウト

# sfexって？

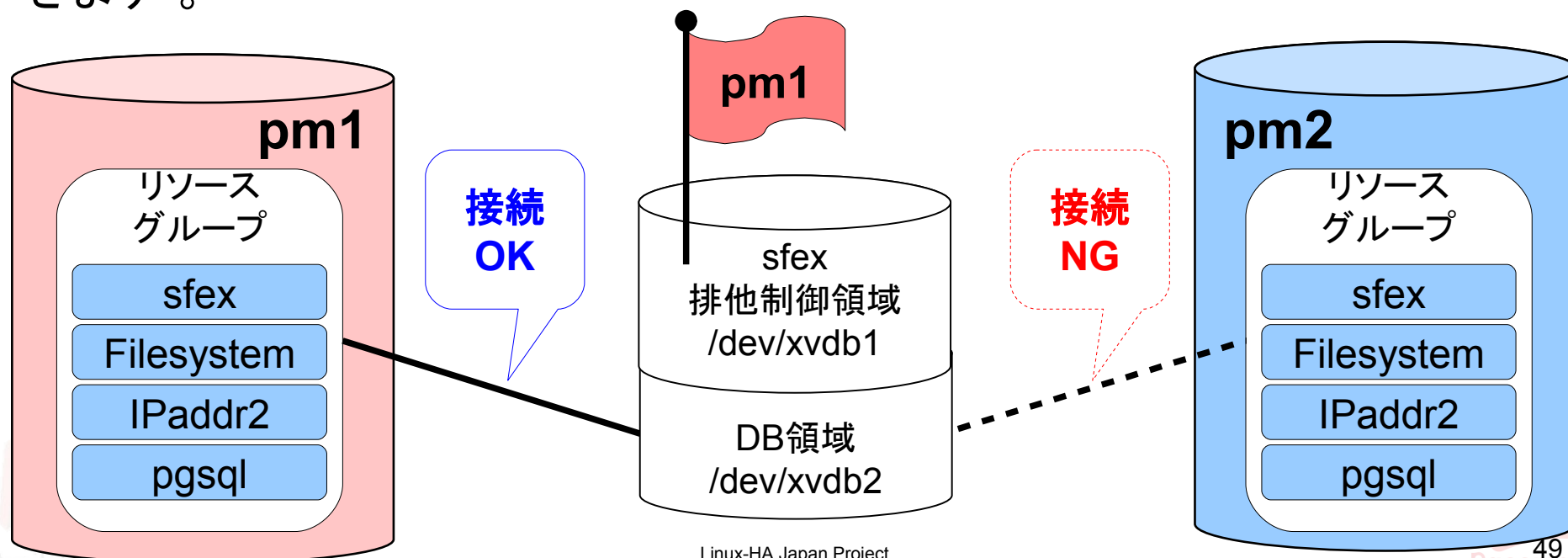




# 共有ディスク排他制御機能

## Shared Disk File Exclusiveness Control Program

sfexは共有ディスクの所有権を制御するリソースです。  
共有ディスク上のデータパーティションを使うリソースと一緒にリソースグループを作ります。  
所有権を持ったノードのリソースのみがデータパーティションにアクセスできます。





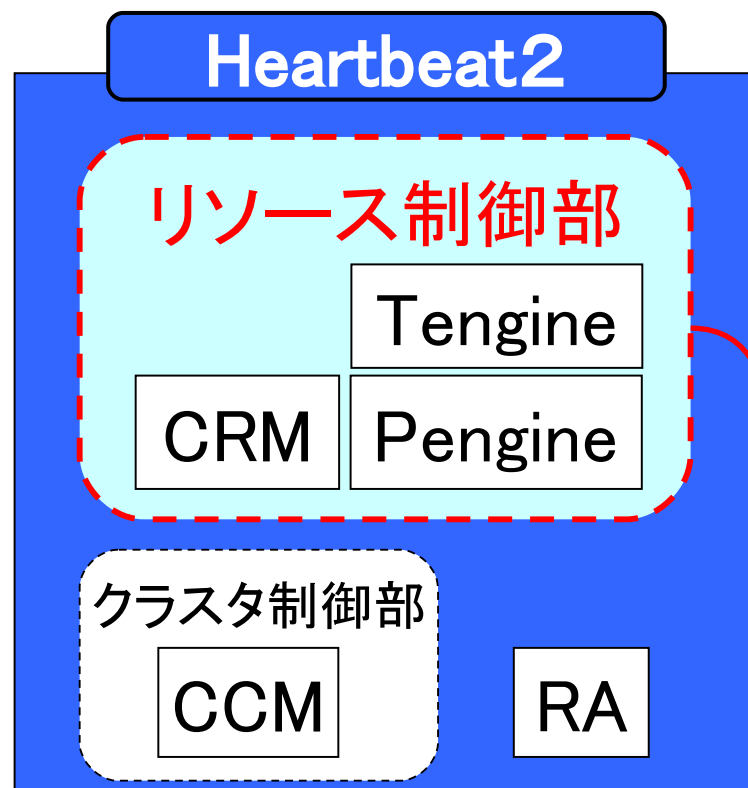
②

# Pacemakerの コンポーネント構成

# Pacemaker のコンポーネント構成は 少々複雑なのです...



# リソース制御部 : Pacemaker



他のクラスタソフトウェア間とのコンポーネントの共通化のために、Heartbeat<sup>2</sup>のリソース制御部がPacemakerとして切り出されました

**Pacemaker**

CRM:	Cluster Resource Manager
Tengine:	Transition Engine
Pengine:	Policy engine
CCM:	Cluster Consensus Membership
RA:	Resource Agent

※リソース制御機能は主にこのコンポーネントに含まれる

切り出されたということは・・・  
Pacemaker 単独では  
HAクラスタソフトとして  
動作しない？



そのとおりです..

Pacemaker は  
クラスタ制御部の  
アプリケーションと  
組み合わせて  
使用しなければなりません..

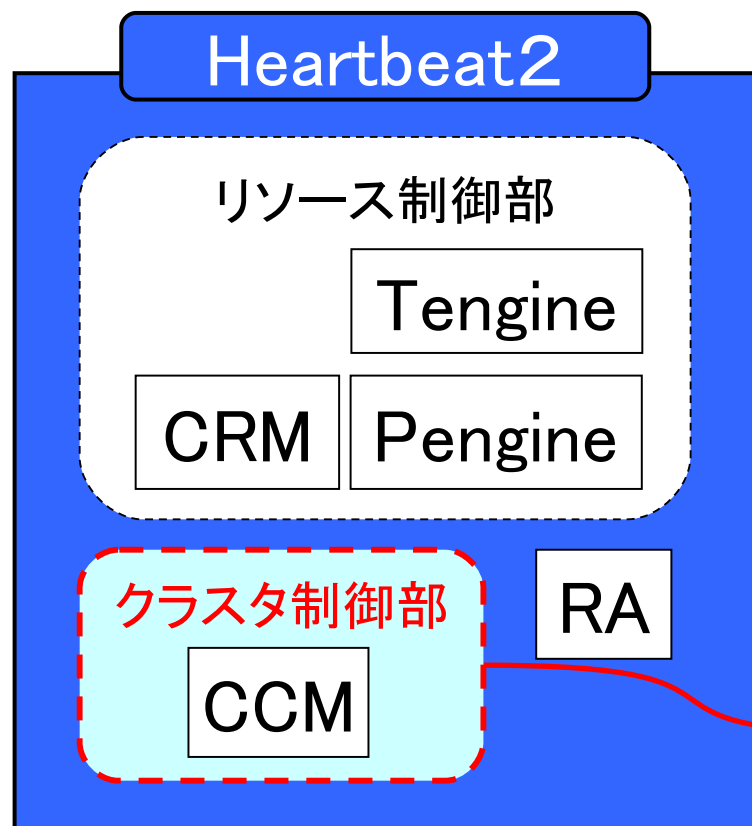


ですが、  
クラスタ制御部の  
候補がいくつもあると  
前向きにとらえてください！





# クラスタ制御部 候補1 : Heartbeat3



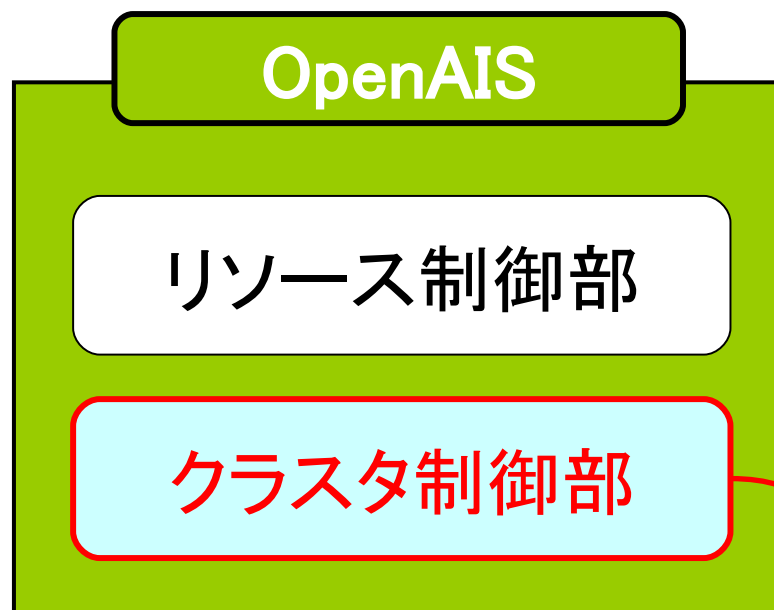
Heartbeat2の  
クラスタ制御部が、  
**Heartbeat3** として切り  
出されました

**Heartbeat3**

切り出されたので“2”から“3”と数字が  
上がったのに、機能的にはデグレ！？

※ノード監視は主にこちらの  
コンポーネントに含まれる

# クラスタ制御部 候補2 : Corosync



“OpenAIS”というオープンソースのHAクラスタがあり、このクラスタ制御部がCorosyncとして切り出されました

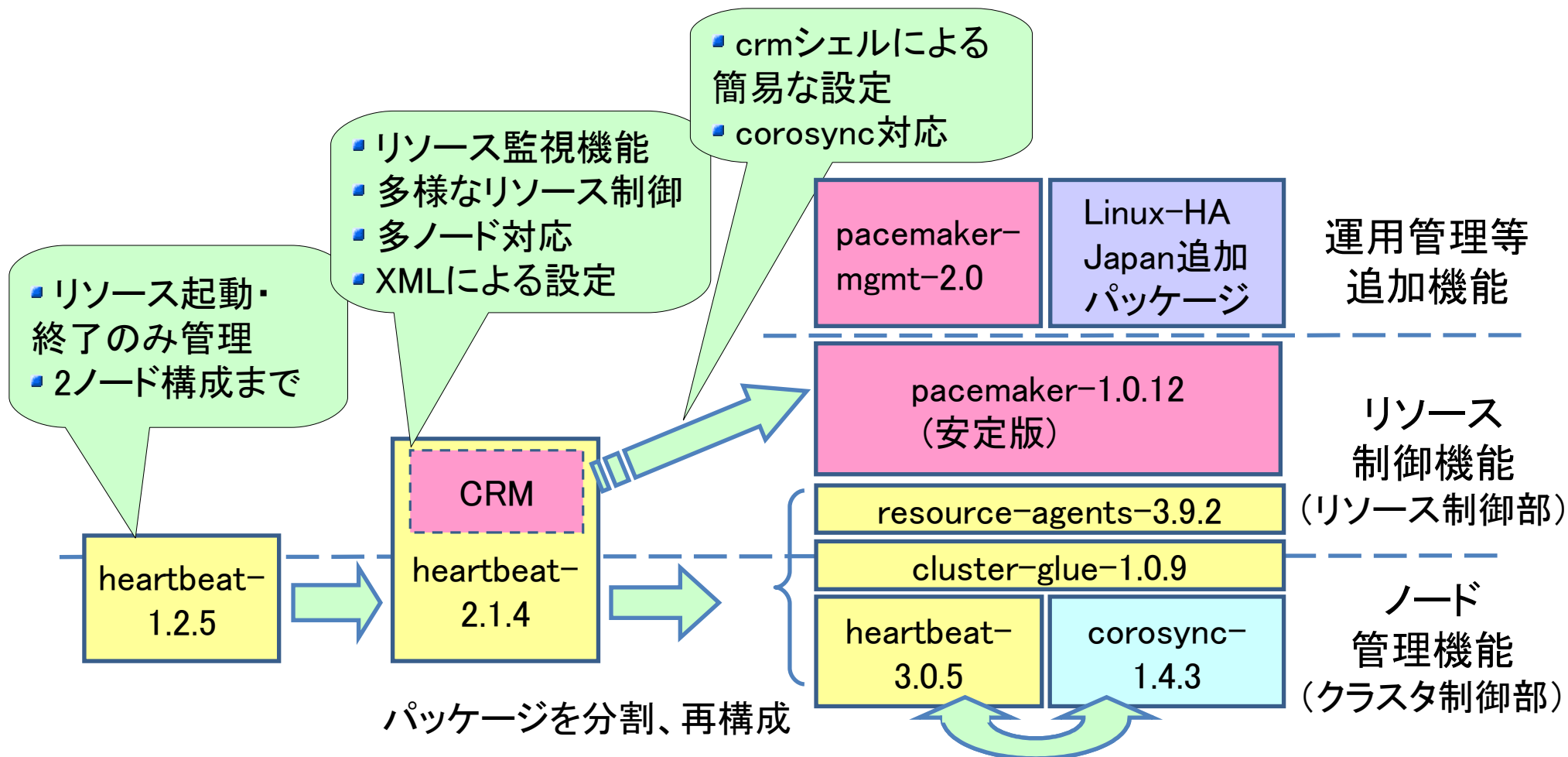
**Corosync**



# Pacemaker は 「Heartbeat3」 「Corosync」 2つのクラスタ制御部が 選択可能



# Pacemakerの生い立ち



CRM: Cluster Resource Manager

バージョン番号は2012年5月時点での最新版

ノード管理機能を選択可能！

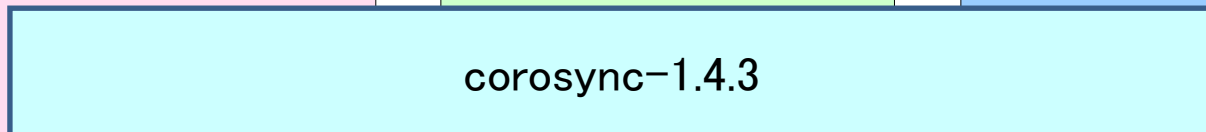
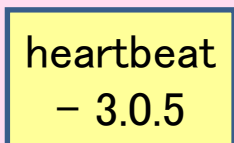
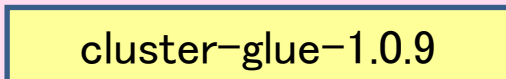
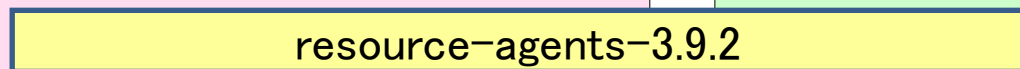
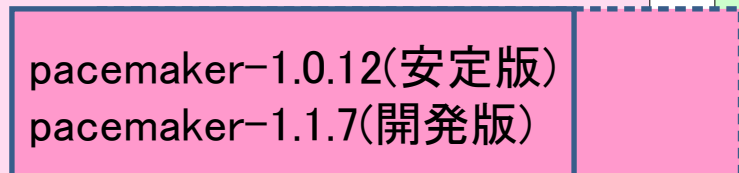
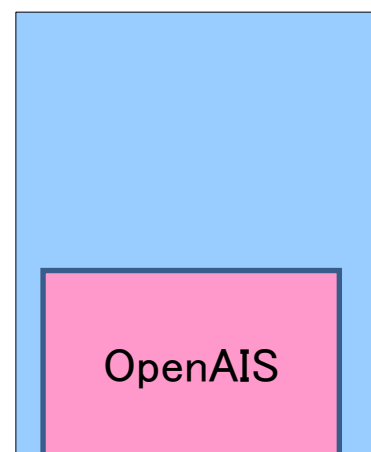
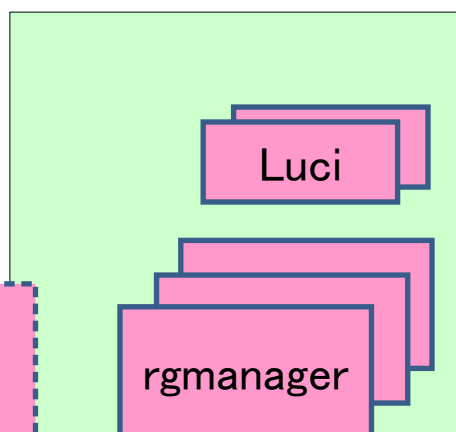
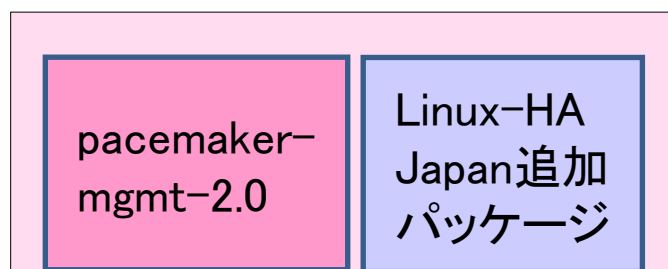
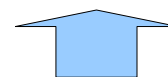
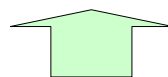
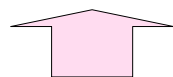
# 他HAプロダクトとコンポーネントの共通化が進められています

Pacemaker

Red Hat

High Availability Add-on

OpenAIS



リソース  
制御機能  
(リソース制御部)

ノード  
管理機能  
(クラスタ制御部)

# Heartbeat3 と Corosync どちらのクラスタ制御部が 優れているの？



# Heartbeat3 のメリット・デメリット

## ■ メリット

### □ 安定

- Heartbeat2系のノード管理機能を引き継いでいるため、これまでの使用方法ならば実績と安定性がある

## ■ デメリット

### □ 多ノード構成に向いていない

- リソース数にもよるが、7ノードくらいが限界

### □ ハートビートLAN切断時(スプリットブレイン) に弱い

- スプリットブレイン時の復旧手順がやや複雑
- クォーラム制御(3ノード以上時に使用)が不安定

### □ オンラインによるノード追加・削除時の動作が不安定

# Corosync のメリット・デメリット

## ■ メリット

- 多ノード構成に向いている
  - 11ノードで動いた！
  - 次期バージョンでは16ノード以上でも動くという情報も…
- ノード故障検出時間が短い
- スプリットブレイン回復時の動作が安定
- オンラインによるノード追加・削除時の動作が安定

## ■ デメリット

- 開発途上で不安定
  - 頻繁にバグフィックス版がリリース





③

# 本日の Pacemakerデモ環境



# 本日のPacemakerデモ環境

## ■ ハードウェア

- ノートPC (Core2Duo 2.26MHz、メモリ 4G)

## ■ OS

- CentOS 5.8 x86\_64

## ■ HAクラスタ

- Pacemaker-1.0.12
- アクティブ/スタンバイの2台構成

## ■ クラスタ化するアプリケーション

- PostgreSQL 9.1.3 (Streaming Replicationは使用しません)

## ■ 仮想環境

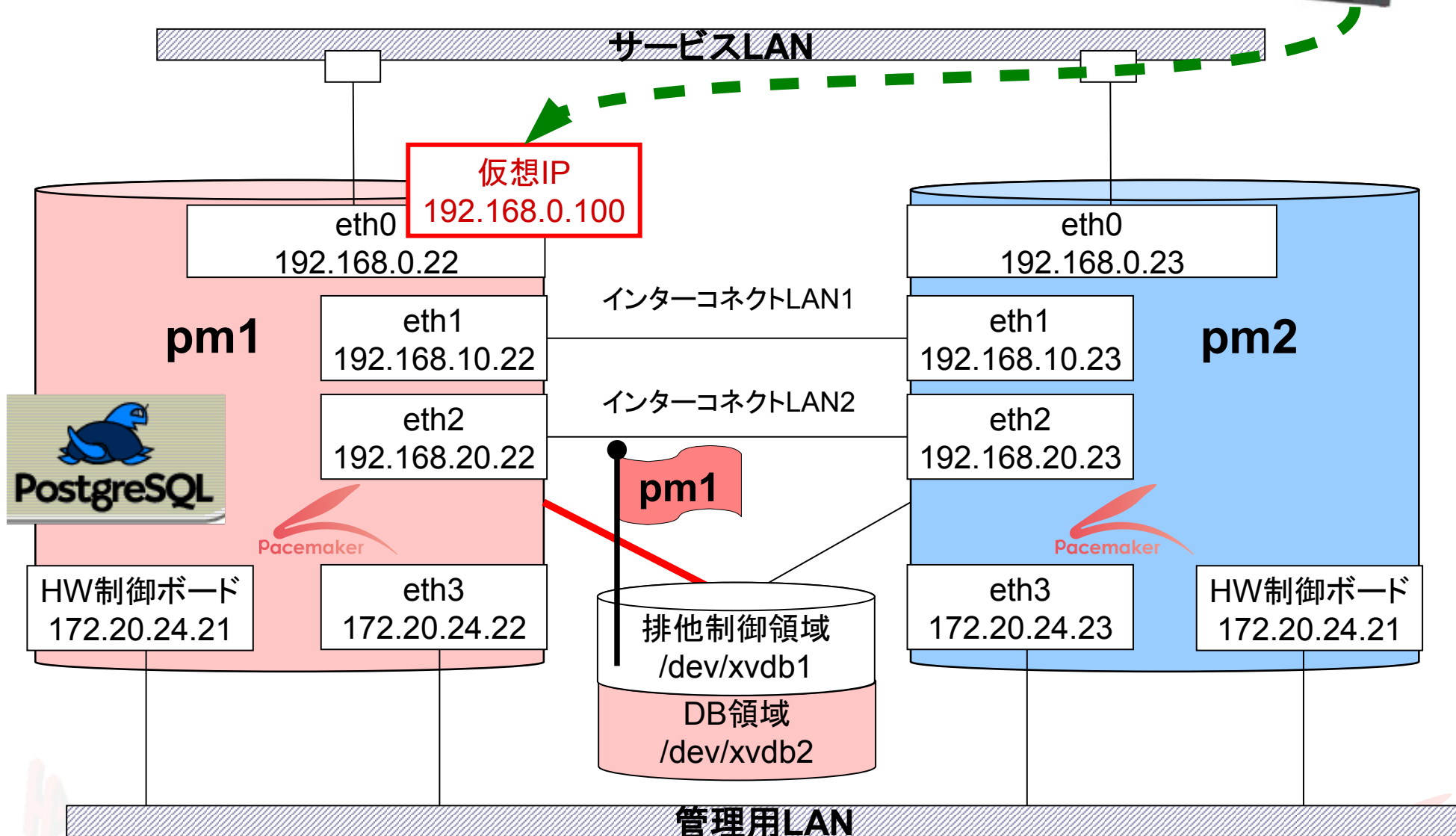
- Xen (CentOS 5.8同梱版)
- Domain-Uは2ドメインで構成
- 各ドメインには、CPU×1・メモリ1024M を割り当て

# Pacemakerデモ構成

pm1: アクティブ

pm2: スタンバイ

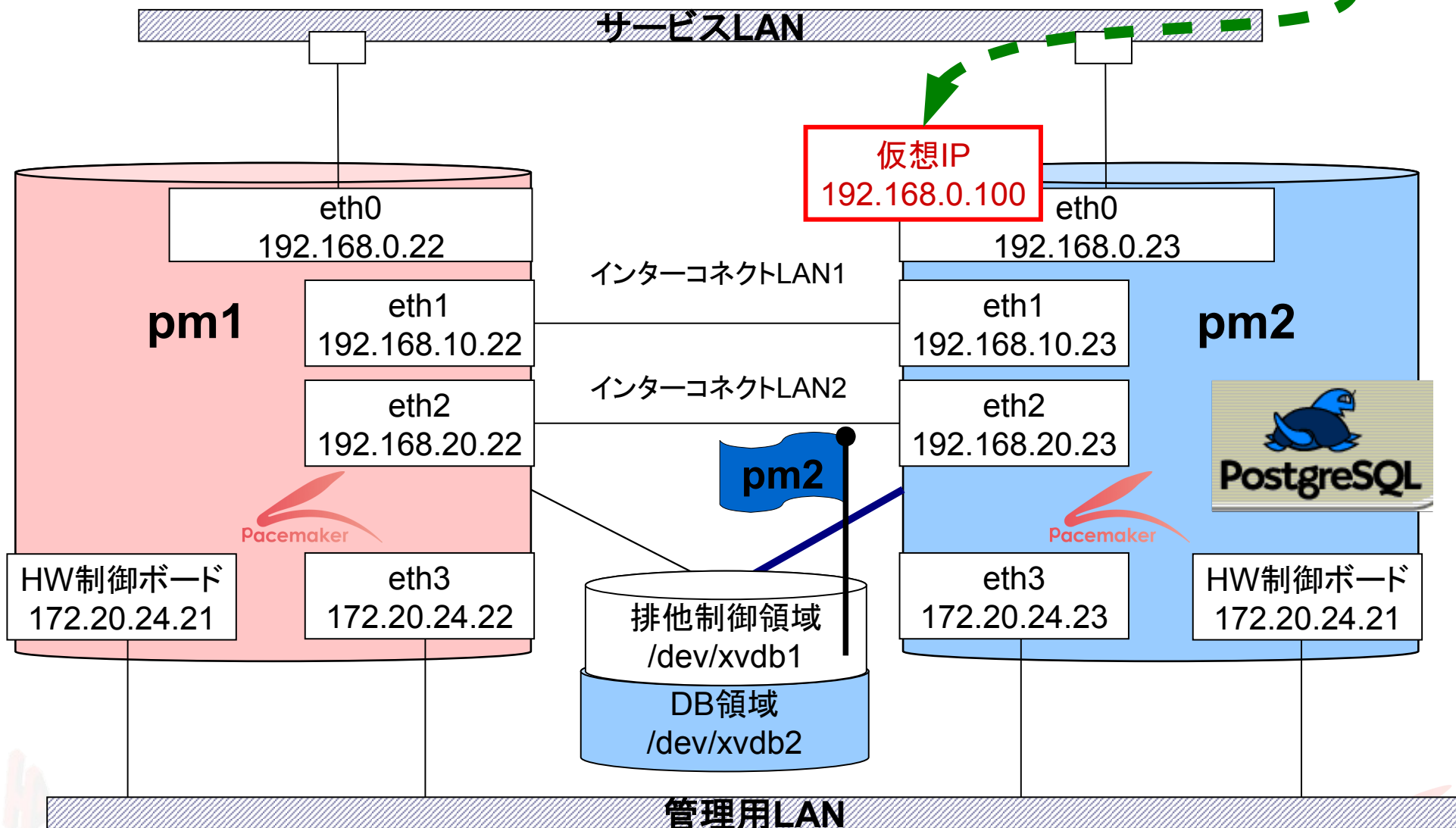
demo  
(Domain-0)



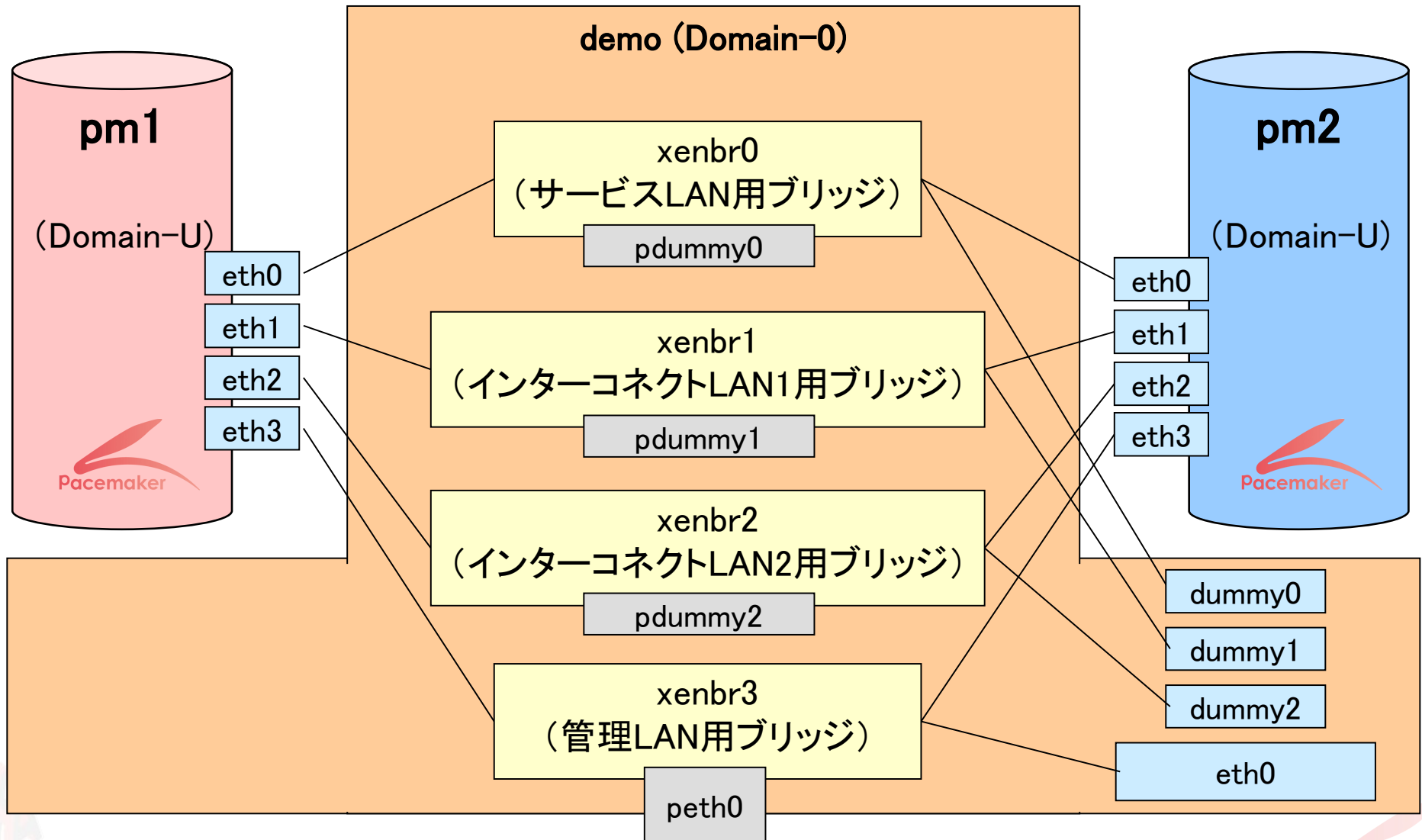
# Pacemakerデモ構成

pm1: スタンバイ  
**pm2: アクティブ**

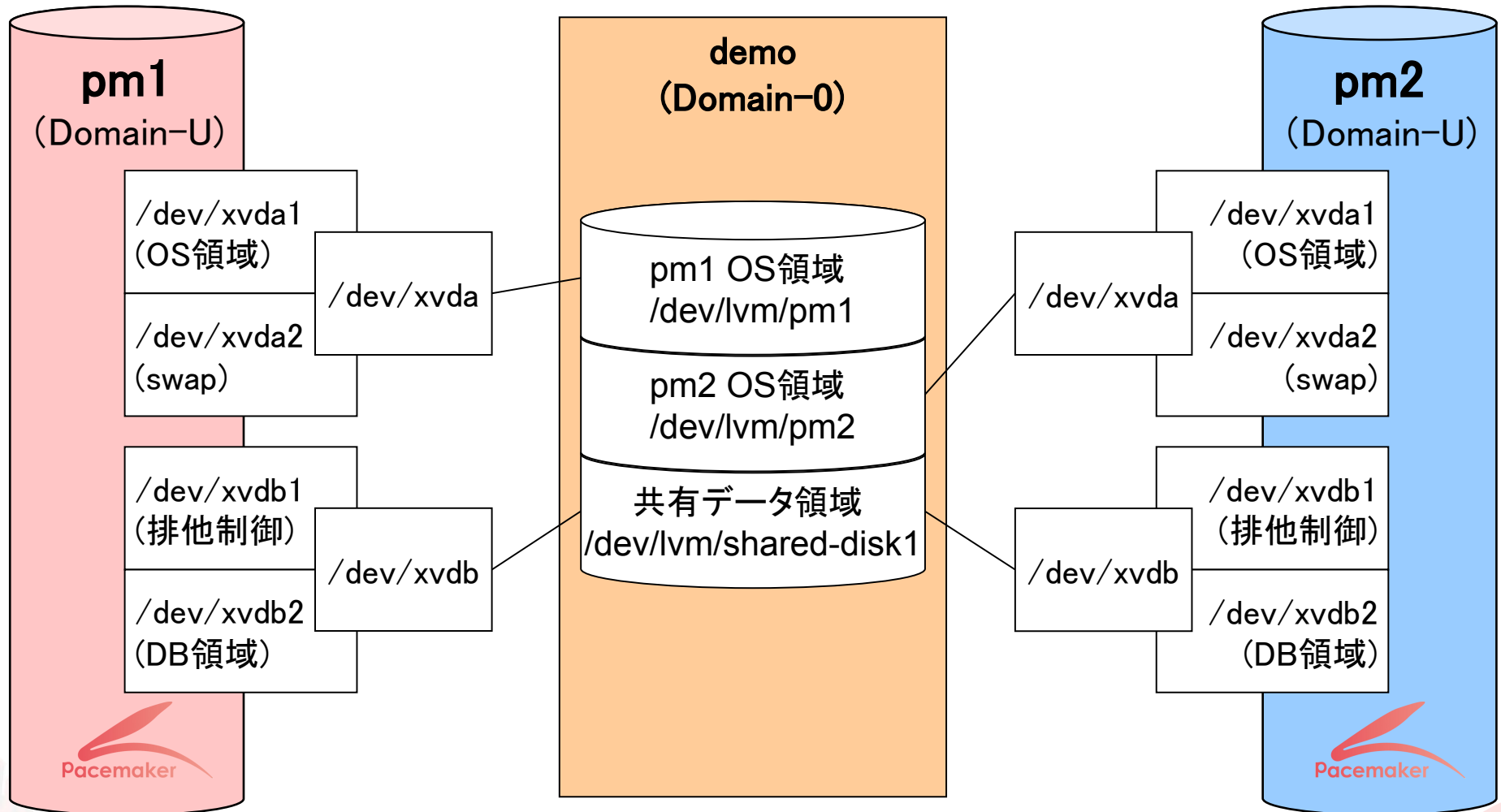
demo  
(Domain-0)



# Pacemakerデモ機構成(Xen仮想NW)



# Pacemakerデモ機構成(Xen仮想ディスク)



# Pacemakerデモ

## リソース構成

これら4つの  
リソースは  
グループ設定します

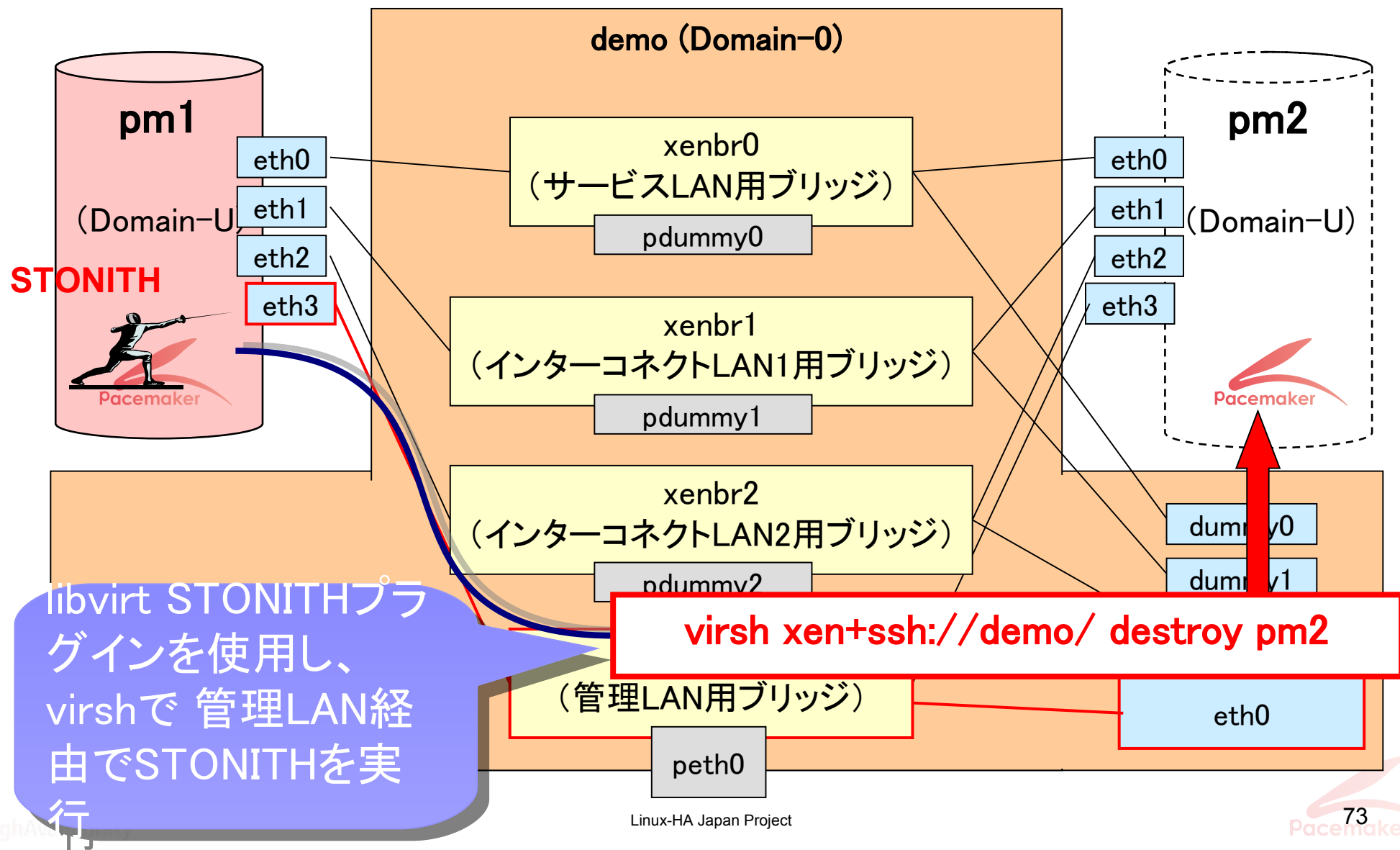
- ディスク排他制御 (sfex)
  - 共有ディスクの排他制御を行います
- DBデータ領域マウント (Filesystem)
  - 共有ディスクにあるDBデータ領域のマウント制御を行います
- PostgreSQL制御 (pgsql)
  - PostgreSQL 9.1.3 の制御を行います
- 仮想IP割り当て (IPaddr2)
  - サービス提供用の仮想IPを割り当てます

今日はSTONITH  
のデモも行います

- STONITH (stonith-helper, libvirt, meatclient)
  - 監視対象サーバの異常を検出したときに、強制的にそのサーバをダウンさせるノードフェンシングを行います。
- ネットワーク監視 (pingd)
  - 指定したIPアドレスに ping送信し、ネットワーク疎通があるかどうかの監視を行います。
- ディスク監視 (diskd)
  - 指定したディスクデバイスにアクセスし、ディスクの正常性確認を行います。



# Pacemakerデモ機フェンシング (STONITH) 構成





④

インストール・設定を  
デモします！



# インストール方法の種類

1. yum を使ってネットワークインストール
  - Pacemaker本家(clusterlabs) の yumのリポジトリを使用
  - サーバにインターネット接続必須
2. ローカルリポジトリ + yum を使ってインストール
  - Linux-HA Japan 提供のリポジトリパッケージを使用
  - Linux-HA Japan オリジナルパッケージも含まれる
3. rpm を手動でインストール
  - 沢山のrpmを個別にダウンロードする必要あり
4. ソースからインストール
  - 最新の機能をいち早く試せる
  - コンポーネントが多いので、コンパイルは面倒

本日は「2」の  
構築デモを行  
います

# Pacemaker sourceファイル

- Heartbeat

- <http://hg.linux-ha.org/dev/>

- Corosync

- <https://github.com/corosync/corosync/>

- Pacemaker

- <https://github.com/ClusterLabs/pacemaker/>

- Cluster Glue

- <http://hg.linux-ha.org/glue/>

- Resource Agents

- <https://github.com/ClusterLabs/resource-agents/>

- Pacemaker mgmt

- <http://hg.clusterlabs.org/pacemaker/pygui/>

- Linux-HA Japan追加パッケージ

- <http://hg.sourceforge.jp/view/linux-ha/>

こーんなに沢山のサイトから  
ダウンロード&インストール  
するのは大変・・・



CentOS系(RHEL系)ならば  
Linux-HA Japan提供の  
リポジトリパッケージを使えば  
インストールは簡単！  
(EL5、EL6系対応)



# ～ ローカルリポジトリ + yum を使ってインストール ～

(サーバにインターネット接続環境がなくてもOK！)

## ■ 1. Pacemakerリポジトリパッケージをダウンロード

Linux-HA Japan 提供の Pacemakerリポジトリパッケージを sourceforge.jp からダウンロードしておきます。

pacemaker-1.0.12-1.1.el5.x86\_64.repo.tar.gz  
をダウンロード

SourceForge.jp > ソフトウェアを探す > Linux-HA Japan > 概要

Linux-HA Japan

本ページはLinux-HA Japan 開発者向けサイトです。プロジェクトのメインサイトはこちらです <http://linux-ha.sourceforge.jp/>  
Linux-HA Japanプロジェクトは、Linux上で高可用クラスシステムを構築するための部品として、オープンソースの、クラスタリソースマネージャ、クラスタ通信レイヤ、ブロックデバイス複製、その他、さまざまなアプリケーションに対応するための数多くのリソースエージェント、などを、日本国内向けに維持管理、支援等を行っているプロジェクトです。

主な製品として、Pacemaker、Heartbeat、Corosync、DRBD等を取り扱っています。

[Linux-HA Japanの詳細情報へ](#)

[Linux-HA Japanのインストール方法](#)

[Linux-HA Japanの使い方](#)

最終更新日: 2010-08-23 12:44

開発メンバー: ksk, t-matsuo, takayukitanaka, b-oka, bellche, hideoyamauchi, iidayuus, ikeda, inoue-kazu, jsuglura, kmii, kitateish, 他6名 [一覧]

[その他の情報](#)

No Image  
Available

Pacemaker-1.0.12-1.1 版は  
4/27リリース

ダウンロード

最終更新日: 2010-06-18 07:21

## ■ 2. yumでインストール！

/tmp で展開し、yumコマンドでインストールします。

```
# cd /tmp
# tar zxvf pacemaker-1.0.12-1.1.el5.x86_64.repo.tar.gz
# cd /tmp/pacemaker-1.0.12-1.1.el5.x86_64.repo/
# yum -c pacemaker.repo install pacemaker pm_crmgen pm_diskd
pm_logconv-hb pm_extras
```

- pm\_crmgen-1.1-1.el5.noarch.rpm ... crm用ファイル編集ツール
- pm\_diskd-1.1-1.el5.x86\_64.rpm ... ディスク監視アプリとRA
- pm\_logconv-hb-1.1-1.el5.noarch.rpm ... ログ変換ツール
- pm\_extras-1.2-1.el5.x86\_64.rpm ... その他オリジナルRA 等

ぜひぜひ使ってみてください！



# ここで Pacemakerのインストールを デモします！

pm1のみインストール  
デモします



## クラスタ制御部基本設定

### /etc/ha.d/ha.cf

- クラスタ制御部の基本設定ファイル
- クラスタ内の全サーバに同じ内容のファイルを設置

```
pacemaker on  
  
debug 0  
udpport 694  
keepalive 2  
warntime 7  
deadtime 10  
initdead 48  
logfacility local1  
  
bcast eth1  
bcast eth2  
  
node pm1  
node pm2  
  
watchdog /dev/watchdog  
respawn root /usr/lib64/heartbeat/ifcheckd
```

pm\_extrasをインストールし、  
この ifcheckd の設定を追加  
すればインターコネクトLAN  
の接続状況も確認可能です

### /etc/ha.d/authkeys

- サーバ間の「認証キー」を設定するファイル
- クラスタ内の全サーバに、同じ内容のファイルを配置
- 所有ユーザ/グループ・パーミッションは root/root ・ rw---- に設定

```
auth 1  
1 sha1 hogehoge
```

これも基本的に  
Heartbeat2 と  
設定は同じです

認証キー: 任意の文字列

認証キーの計算方法: sha1, md5, crcを指定可

## /etc/syslog.conf

- 必須の設定ではないが、多くのログが/var/log/messagesに出力されるため出力先を個別のファイルに変更するのがお勧め

local1.info を使用し、/var/log/ha-log へ出力する場合の例

```
*.info;mail.none;authpriv.none;cron.none;local1.none
```

```
/var/log/messages
```

```
:
```

```
(省略)
```

```
:
```

local1.info

/var/log/ha-log

ha.cf で設定したlogfacility 名

ここまでいけば、  
Pacemakerが起動できます！

```
# /etc/init.d/heartbeat start
```

← 2サーバで実行

```
Starting High-Availability services:
```

[ OK ]

ということで、  
Pacemakerを  
起動してみます！

基本設定は  
省略します..



# 起動確認

Pacemakerの状態表示コマンドである  
**crm\_mon**コマンドにより、Pacemakerが制御している  
サーバ状態やリソース状態、インターコネクトLAN、  
ネットワークの状態を確認できます。

```
# crm_mon
```

# crm\_mon実行例

```
=====
Last updated: Fri May 25 14:59:57 2012
Stack: Heartbeat
Current DC: pm2 (7f1b5dcb-e696-414d-8fca-da79274b0a74) - partition with quorum
Version: 1.0.12-066152e
2 Nodes configured, unknown expected votes
6 Resources configured.
=====
```

```
Online: [ pm1 pm2 ]
```

```
Resource Group: grpPg
```

```
  prmEx      (ocf::heartbeat:sfex):  Started pm1
  prmFs      (ocf::heartbeat:Filesystem):  Started pm1
  prmPg      (ocf::heartbeat:pgsql): Started pm1
  prmIp      (ocf::heartbeat:IPaddr2):  Started pm1
```

```
Resource Group: grpStonith1
```

```
  prmStonith1-1 (stonith:external/stonith-helper):  Started pm2
  prmStonith1-2 (stonith:external/libvirt):  Started pm2
  prmStonith1-3 (stonith:meatware):  Started pm2
```

```
Resource Group: grpStonith2
```

```
  prmStonith2-1 (stonith:external/stonith-helper):  Started pm1
  prmStonith2-2 (stonith:external/libvirt):  Started pm1
  prmStonith2-3 (stonith:meatware):  Started pm1
```

```
Clone Set: clnDiskd1
```

```
  Started: [ pm1 pm2 ]
```

```
Clone Set: clnDiskd2
```

```
  Started: [ pm1 pm2 ]
```

```
Clone Set: clnPingd
```

```
  Started: [ pm1 pm2 ]
```



# サーバ状態表示

Pacemakerが稼動しているサーバは「Online」と表示され、停止しているサーバは「OFFLINE」と表示されます。

=====

Last updated: Fri May 25 14:59:57 2012

Stack: Heartbeat

Current DC: pm2 (7f1b5dcb-e696-414d-8fca-da79274b0a74) -  
partition with quorum

Version: 1.0.12-066152e

2 Nodes configured, unknown expected votes

6 Resources configured.

=====

Online: [ pm1 pm2 ]

クラスタに組み込まれている  
サーバ名が表示されます

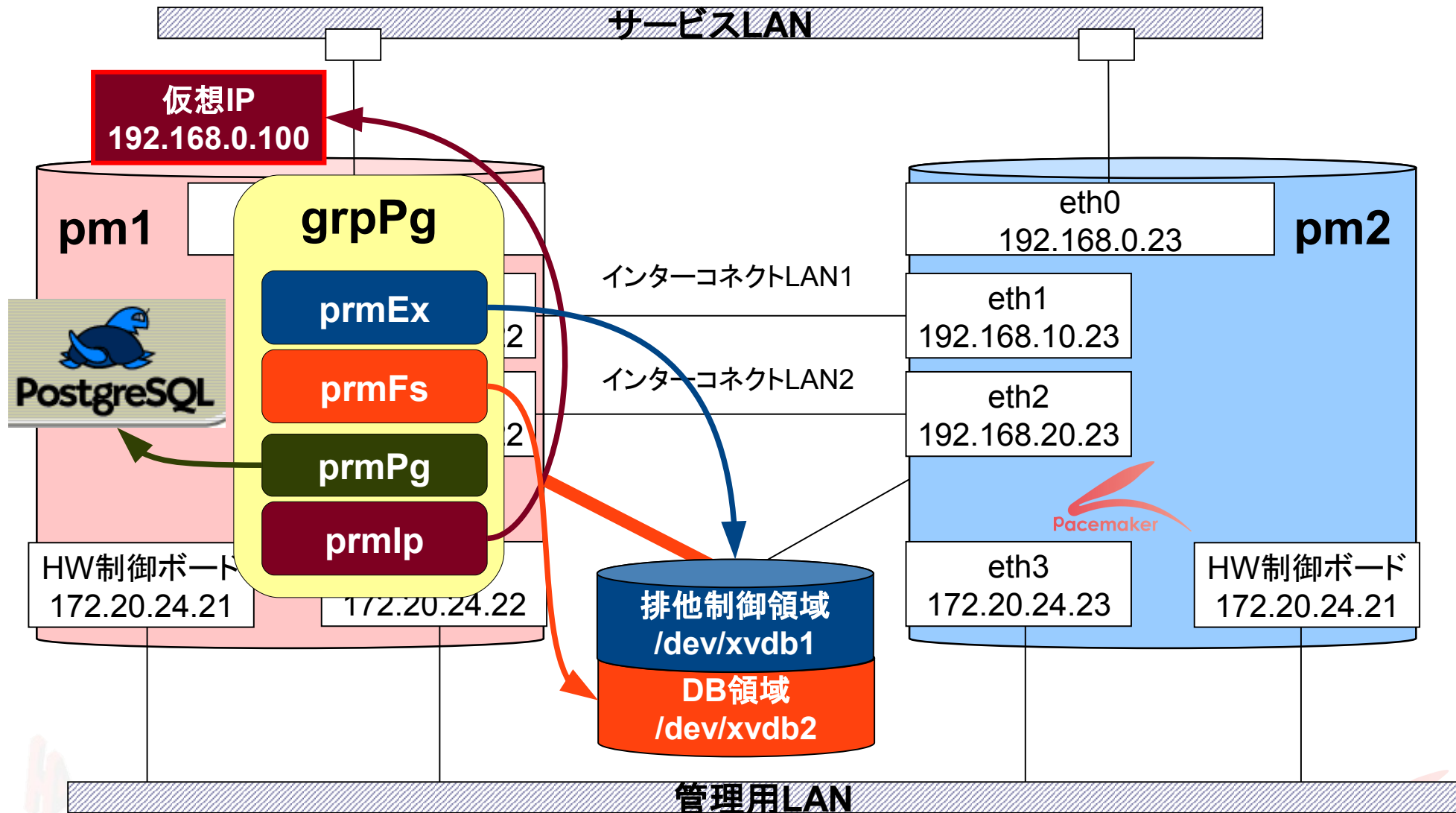
# リソース状態表示(サービス系リソース)

Pacemakerが制御しているリソースの状態が表示されます。  
リソース稼動状態と稼働中のサーバ名が「**Started サーバ名**」  
などと表示されます。

```
Resource Group: grpPg
  prmEx      (ocf::heartbeat:sfex):   Started pm1
  prmFs      (ocf::heartbeat:Filesystem): Started pm1
  prmPg      (ocf::heartbeat:pgsql):  Started pm1
  prmIp      (ocf::heartbeat:IPaddr2): Started pm1
```

- prmEx: ディスク排他制御 (sfex)
- prmFs: DBデータ領域マウント (Filesystem)
- prmPg: PostgreSQL制御 (pgsql)
- prmIp: 仮想IP割り当て (IPaddr2)

# サービス系リソース



# リソース状態表示 (STONITHリソース)

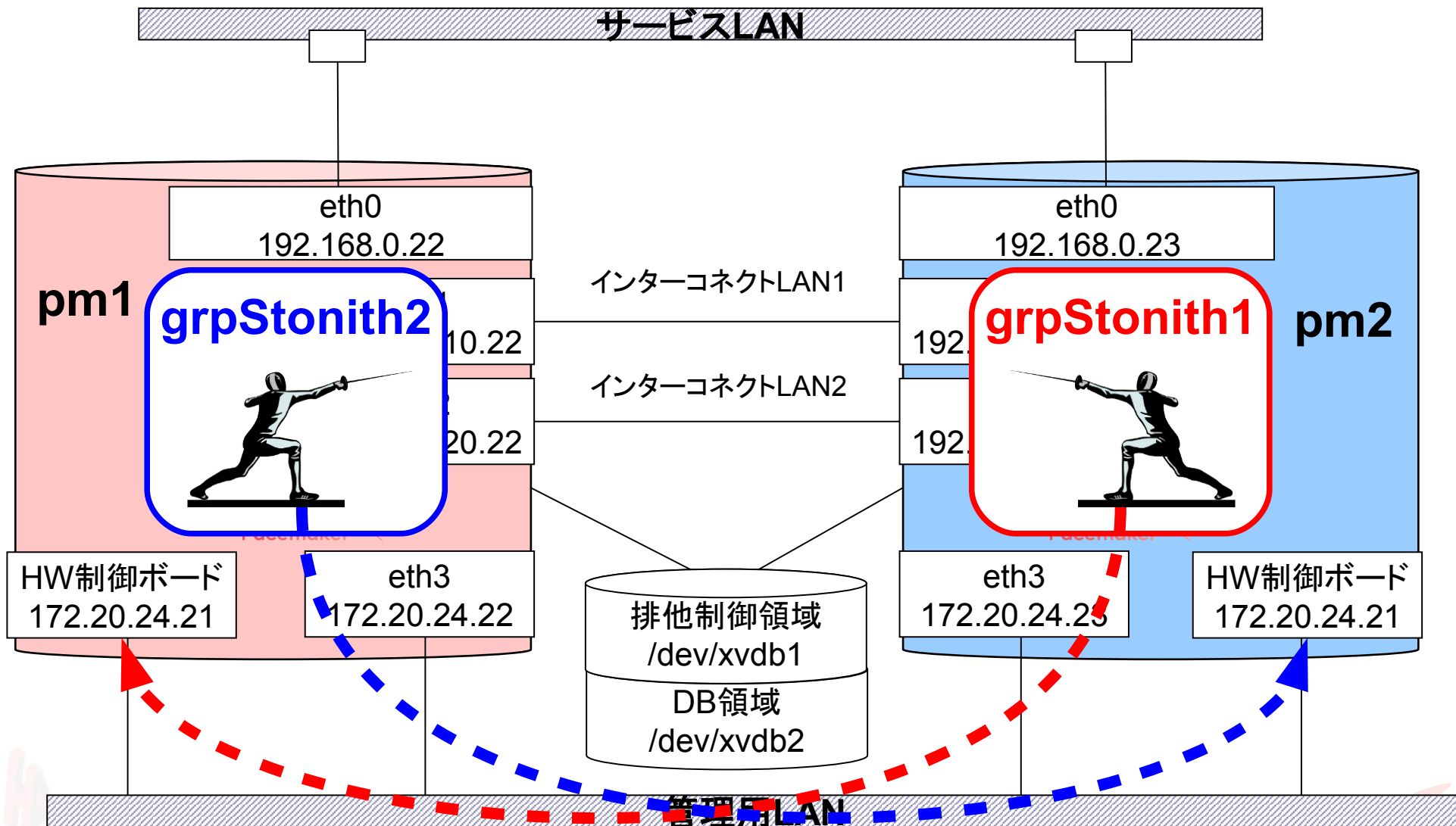
サービス系リソースと同様に、「Started サーバ名」と表示されます。

```
Resource Group: grpStonith1
  prmStonith1-1  (stonith:external/stonith-helper):  Started pm2
  prmStonith1-2  (stonith:external/libvirt):        Started pm2
  prmStonith1-3  (stonith:meatware):                 Started pm2
```

- prmStonith1-1: サーバ断確認、相打防止プラグイン (stonith-helper)
- prmStonith1-2: KVM/Xen用フェンシングプラグイン (libvirt)
- prmStonith1-3: 停止通知用プラグイン (meatware)

このデモでは、grpStonith1 は pm1をフェンシングするSTONITHリソースのため、pm2で稼動しているのが確認できます。

# STONITHリソース



# リソース状態表示(監視系リソース)

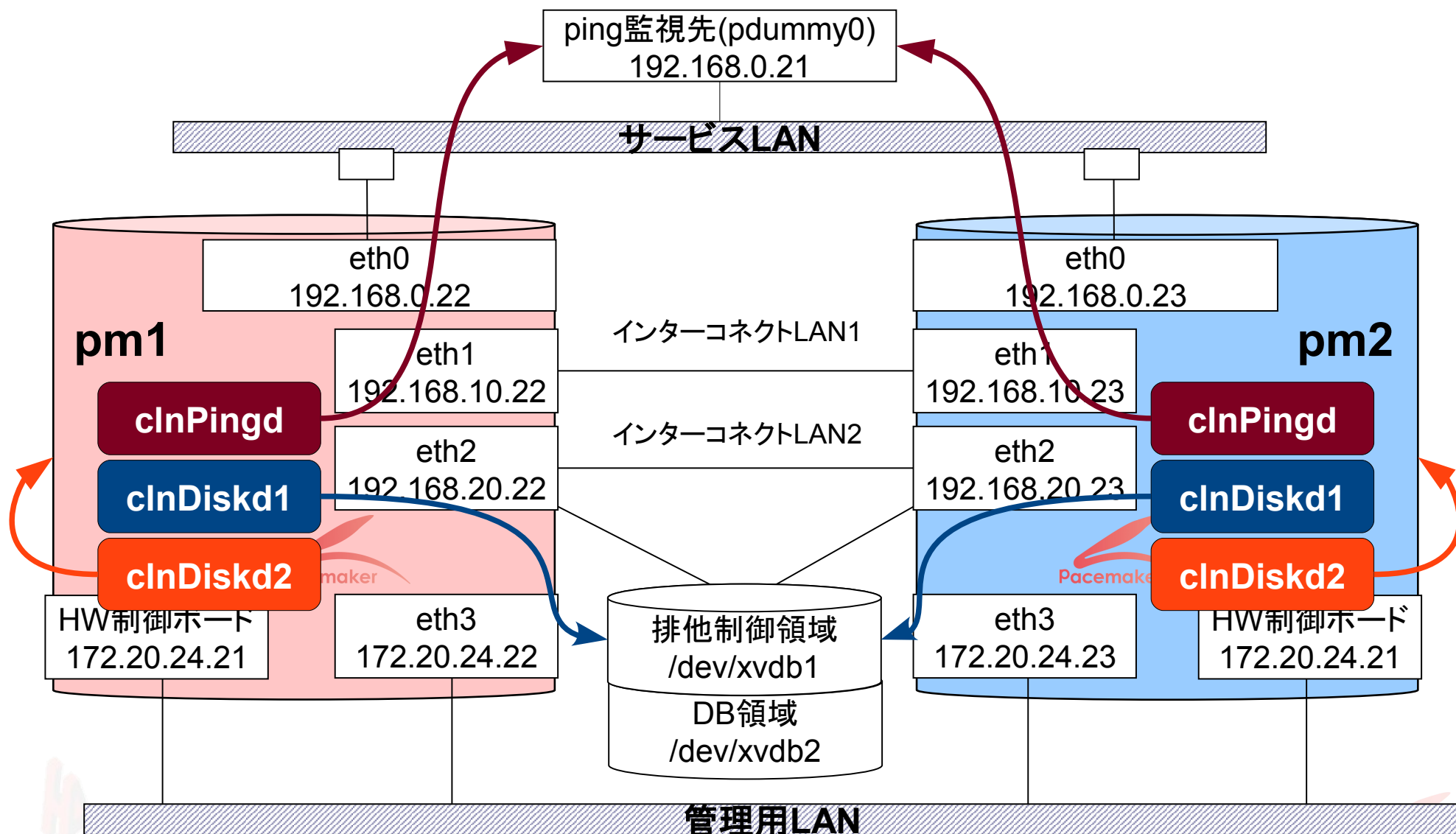
ディスク状態監視、ネットワーク状態監視は、両方のサーバで動作させるように Cloneで登録します。

「Started: [ pm1 pm2 ]」などと表示され、リソース稼動状態と稼働中のサーバ名がわかります。

```
Clone Set: clnDiskd1
  Started: [ pm1 pm2 ]
Clone Set: clnDiskd2
  Started: [ pm1 pm2 ]
Clone Set: clnPingd
  Started: [ pm1 pm2 ]
```

- clnDiskd1: 共有ディスク監視 (diskd)
- clnDiskd2: 内蔵ディスク監視 (diskd)
- clnPingd: ネットワーク監視 (pingd)

# 監視系リソース



**-fA** オプションを付与すると、インターコネクト LAN等の状況も確認可能です。

```
# crm_mon -fA
```

```
* Node pm1:
```


+ default_ping_set	: 100
+ diskcheck_status	: normal
+ diskcheck_status_internal	: normal
+ pm2-eth1	: up
+ pm2-eth2	: up

ネットワーク監視  
の状況

ディスク監視  
の状況

インターコネクトがUPされ  
ているのが確認可能



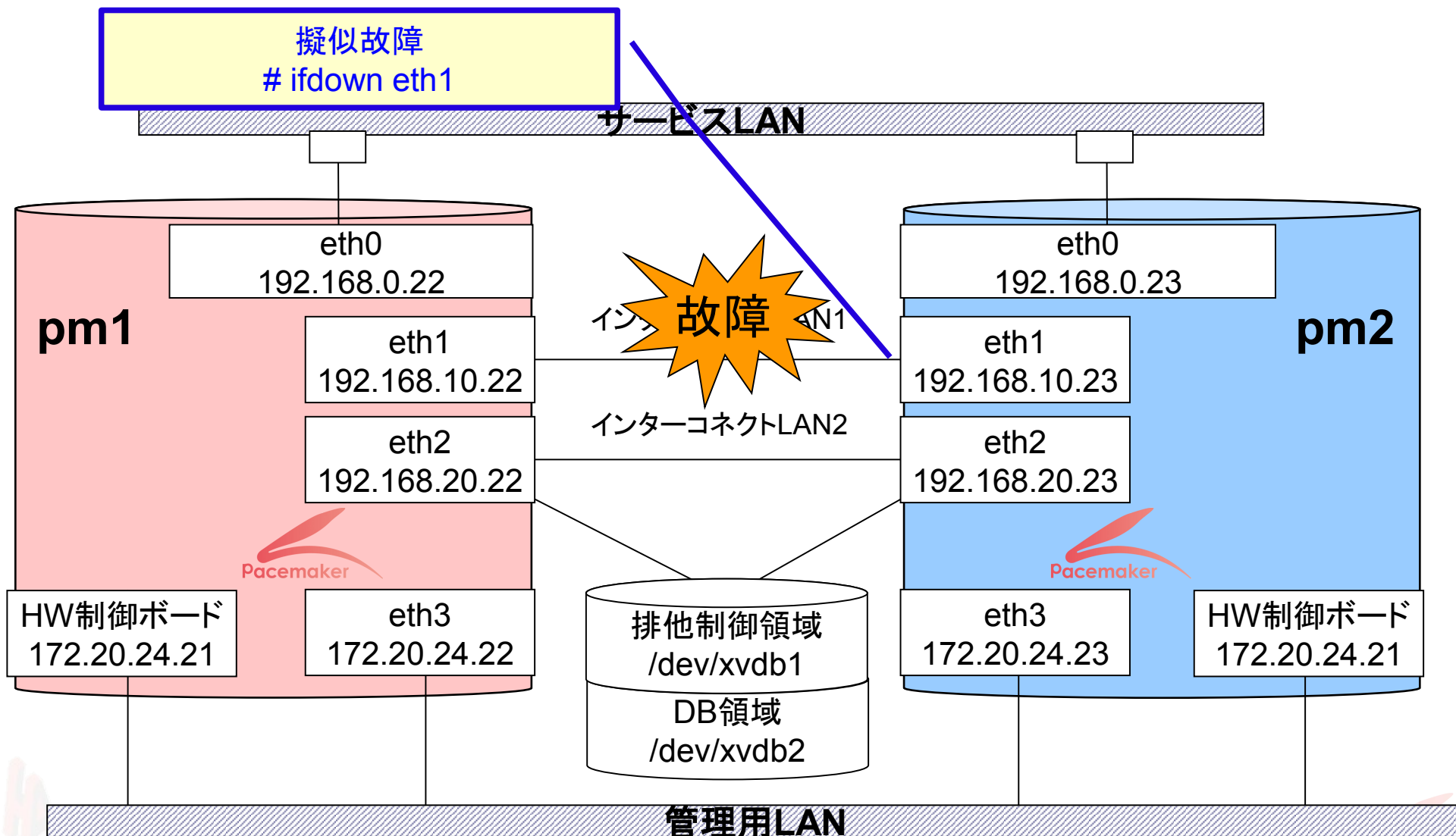


# ここで、 Pacemaker状態表示と インターコネクトLAN故障を デモします！

故障デモ例は  
次ページ



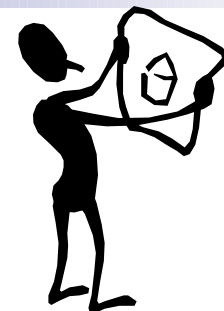
# インターコネクトLAN1を故障させてみる...



これだけでは、  
リソース設定が無いので  
見てのとおり  
アプリケーションは  
なーんにも起動していません…



# 計画



## ■ リソース制御するには事前に計画が必要

### □ リソースの選択

Apache、PostgreSQL、NW監視など、何を使用するか？  
リソースエージェント(RA)がなければ、予め自作してみるか？

### □ リソースの動作の定義

リソースの監視(monitor)間隔は何秒にするか？タイムアウトは？  
故障時はどのように動作させるか？  
リソースエージェント(RA)に与えるパラメータは？

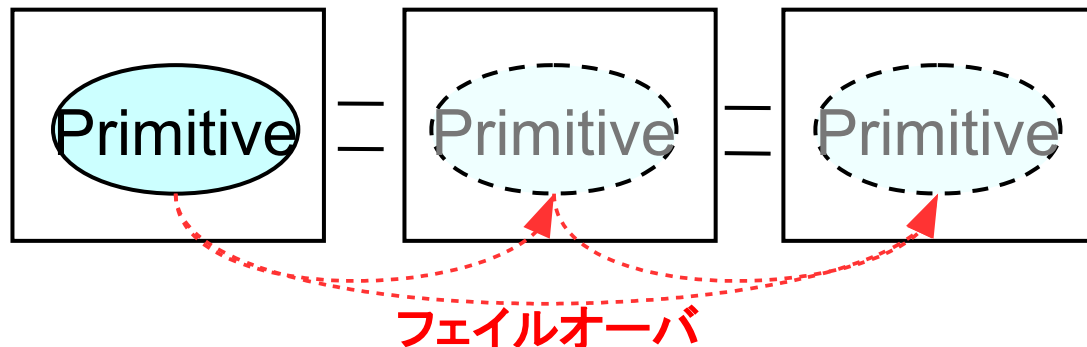
### □ リソース配置・連携の定義

リソースをどのノードで起動させるか？  
リソースの起動順番は？



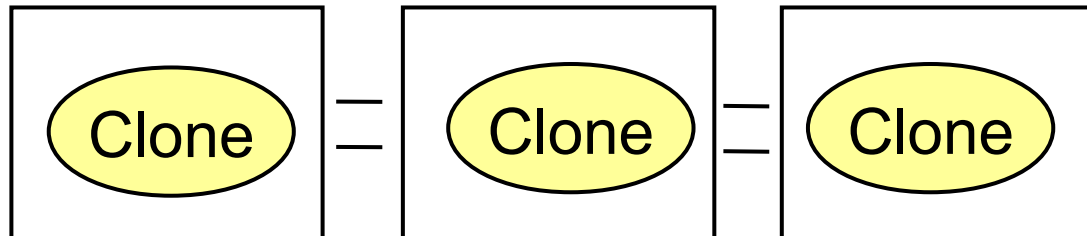
# リソースの種類

## ■Primitive



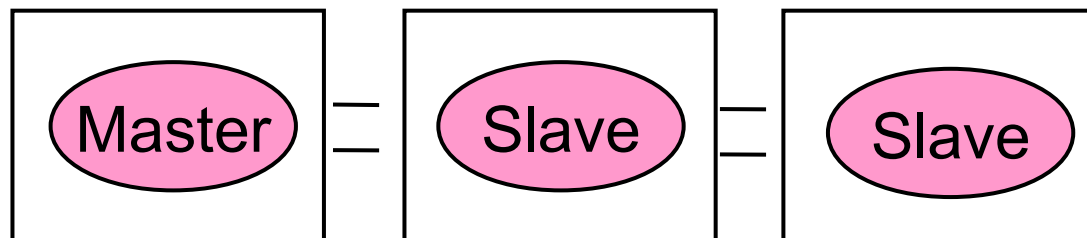
全てのリソース定義の基礎。  
どこか一つのノードで動作させ、  
故障時にフェイルオーバさせたい  
リソースに使用。  
(例) PostgreSQL, 仮想IPアドレス

## ■Clone



複数のノードで動作させたいリソースに  
使用。Primitive を  
定義した後に Clone化する。  
(例) NW監視, ディスク監視

## ■Master / Slave



複数のノードで動作させ、親子関係のある  
リソースに使用。Primitive を定義した  
後にMaster/Slave化する。  
RAには、Masterに昇格、Slaveに降格さ  
せる関数の実装が必要  
(例) PostgreSQL9.1 Streaming  
Replication

# 設定方法

## ■ 主に2通り

- cib.xml ファイルにXML形式で設定を記述
  - 従来のHeartbeat 2での方法
  - XMLを手で書く必要があり面倒
- crmコマンドで設定
  - Pacemakerからの新機能

# cib.xml

- /var/lib/heartbeat/crm/cib.xml


リソースの定義等を設定するXMLファイルを作成します。

(..略..)

```
<primitive class="ocf" id="prmlp" provider="heartbeat" type="IPaddr2">
  <instance_attributes id="prmlp-instance_attributes">
    <nvpair id="prmlp-instance_attributes-ip" name="ip" value="192.168.0.100"/>
    <nvpair id="prmlp-instance_attributes-nic" name="nic" value="eth0"/>
    <nvpair id="prmlp-instance_attributes-cidr_netmask" name="cidr_netmask"
value="24"/>
  </instance_attributes>
  <operations>
    <op id="prmlp-start-0s" interval="0s" name="start" on-fail="restart" timeout="60s"/>
    <op id="prmlp-monitor-10s" interval="10s" name="monitor" on-fail="restart"
timeout="60s"/>
    <op id="prmlp-stop-0s" interval="0s" name="stop" on-fail="block" timeout="60s"/>
  </operations>
</primitive>
(..略..)
```




XMLの記法を知る  
必要があり難しい...



Heartbeatバージョン2を  
使おうとして、  
このXMLで挫折した人は  
多いはずです...



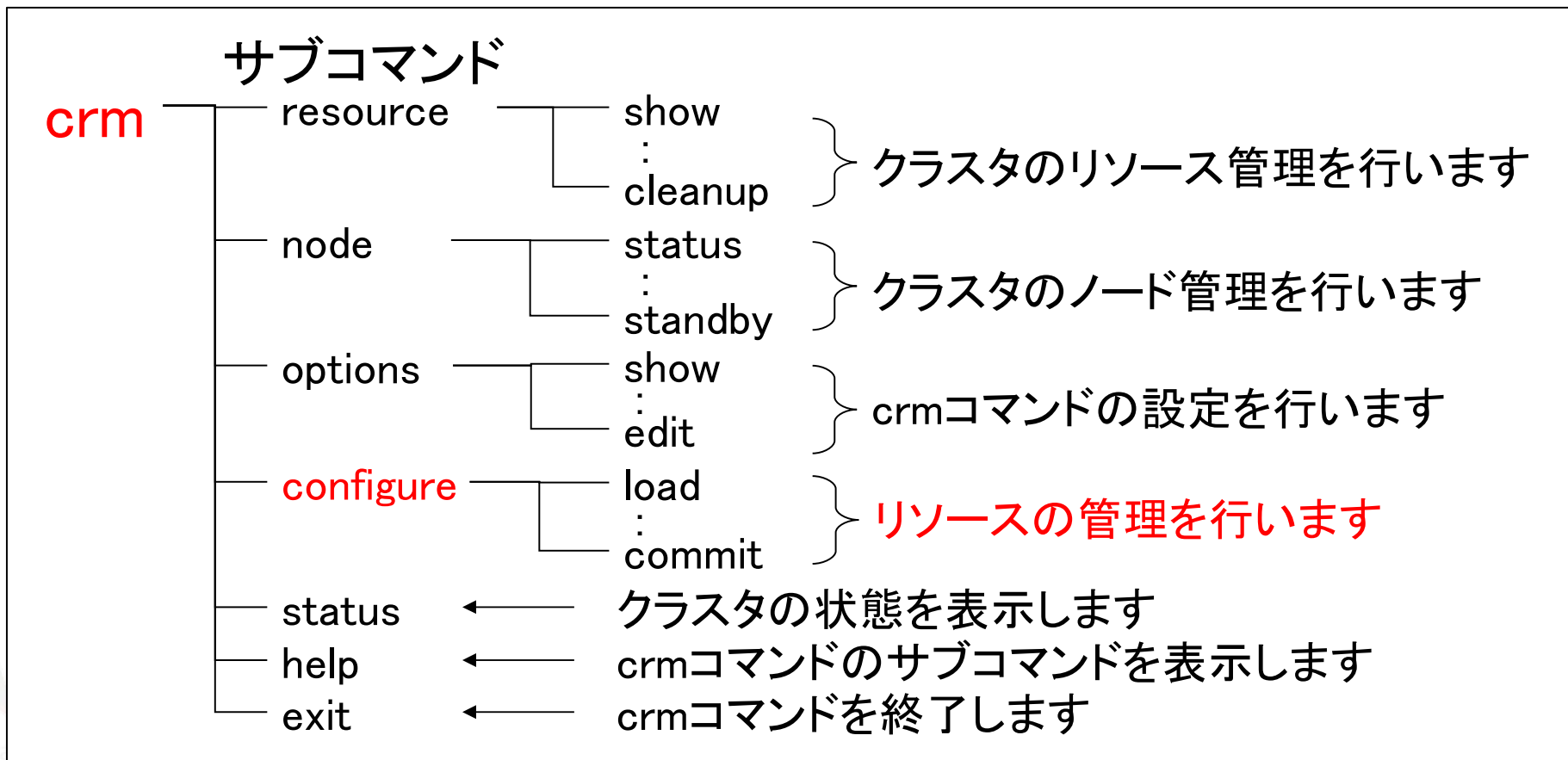




そこで、  
Pacemaker での新機能  
crmコマンドを  
使ってみよう！

# crm コマンド

- 階層構造をもったコマンドラインインターフェイス
- 設定だけでなく、Pacemakerの状態把握や管理も可能
- TABキーで入力内容の補完可能



# crmコマンド実行例


```
# crm
```

```
crm(live)# configure
```

```
crm(live)configure# primitive prmlp ocf:heartbeat:IPaddr2 \  
params ip="192.168.0.100" nic="eth0" \  
cidr_netmask="24" \  
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="fence" \  
:  
crm(live)configure# commit
```


「IPaddr2」リソースエージェント  
を使用して仮想IPを設定をす  
るcrmコマンド例です

コミットされると、cib.xmlに反映されてリソースが起動されます。  
(つまりリソース設定の根っこは cib.xml なのです)



これでも設定方法が  
わかりにくいって人には、





crmコマンドがわからなくても  
まとめて設定できる  
簡単ツールを紹介します！



# crmファイル編集ツール pm\_crmgen

Linux-HA Japanで  
crmファイル編集ツールを開発！

Excelのテンプレートファイルから簡単に  
crm用設定ファイルを生成してくれるツール

リポジトリパッケージに含まれていますし、  
個別にダウンロードも可能です。

<http://sourceforge.jp/projects/linux-ha/>

- どのサーバが優先的にActive？
  - NW監視は？
  - NWが壊れた時の挙動は？
  - STONITHの設定は？
- など細かい挙動の設定も  
可能です！



# 設定イメージ

## 1) Excelのテンプレートファイルにリソース定義を記載

`/usr/share/pacemaker/pm_crmgen/pm_crmgen_env.xls`

ファイルをExcel が使用できるPCにコピーします。  
テンプレート青枠の中に値を記入していきます。

83 #表 5-3 クラスタ設定 ... Primitiveリソース (id=prmlp)

84 PRIMITIVE

85 P

86 #

87

88 A

89 #

90

91

92

93 O

94 #

95

96

97

PRIMITIVE					
P	id	class	provider	type	
#	リソースID	class	provider	type	
	prmlp	ocf	heartbeat	IPaddr2	
A	type	name		value	
#	パラメータ種別	項目	設定内容		
	params	ip	192.168.0.100		
		nic	eth0		
		cidr_netmask	24		
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	on_fail (障害時の動作)	起動前処理
	start	60s	0s	restart	
	monitor	60s	10s	restart	
	stop	60s	0s	fence	

監視間隔  
故障時の

監視間隔やタイムアウト値、  
故障時の動作などを入力



どのサーバをActiveにするかといった  
リソース配置制約の設定も、サーバ名を記述  
するだけで可能です。

270 #表 6-1 クラスタ設定 ... リソース配置制約

271 LOCATION

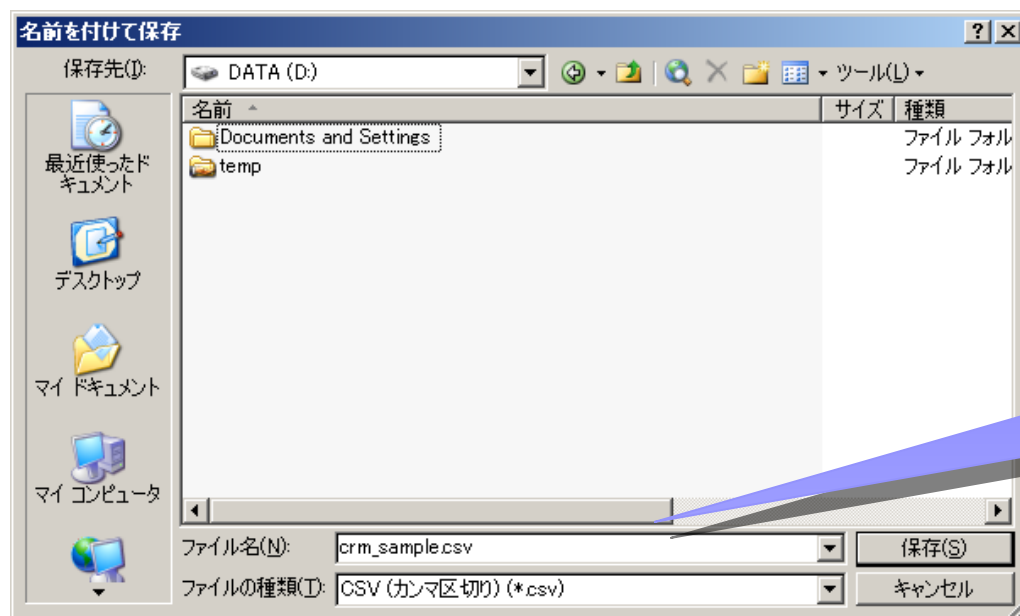
272	rsc	score:200	score:100	score:-inf
273 #	リソースID	Activeノード	Standbyノード	非稼働ノード
274	erpPe	pm1	pm2	
275	erpStonith1			pm1
276	erpSto			pm2

リソースID

ActiveとStandbyサーバを指定

# crm用設定ファイルに変換

## 2) CSV形式でファイルを保存



## 3) CSVファイルをサーバへ転送

CSVファイル保存後、SCPやFTP等でpm\_crmgenがインストールされたサーバへ転送

# crm用設定ファイルに変換

4) pm\_crmgenコマンドでcrmファイルを生成

```
# pm_crmgen -o crm_sample.crm crm_sample.csv
```

生成する設定ファイル名

3)で転送した  
CSVファイル

5) crmコマンドを実行してリソース設定を反映

```
# crm configure load update crm_sample.crm
```

# 出来上がった crmファイル例

(..略..)

### Primitive Configuration ###

primitive prmlp ocf:heartbeat:IPaddr2 \

params \

ip="192.168.0.100" \

nic="eth0" \

cidr\_netmask="24" \

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="fence"

(..略..)

Excelファイルで記述した  
仮想IPを設定する  
crmコマンドが  
ファイルに記述されます

# ログメッセージ制御機能

pm\_logconv-hb

Linux-HA Japanで  
ログメッセージ制御機能を提供中！

Pacemaker標準ログ(ha-log)は出力が多くわかりにくいですが、pm\_logconv-hbを使用すると、運用上必要なログだけを出力することができます。

さらにフェイルオーバーが発生した際に、「Start Fail-over」のログが出力されるようになります。

※クラスタ制御部がHeartbeat3である必要性があります。(Corosyncは未対応です)

# 動作設定

/etc/pm\_logconv.conf

[Settings]

**ha\_log\_path = /var/log/ha-log**

**output\_path = /var/log/pm\_logconv.out**

#hostcache\_path = /var/lib/heartbeat/hostcache

#syslogformat = True

#reset\_interval = 60

attribute\_pingd = not\_defined default\_ping\_set or default\_ping\_set  
lt 100

attribute\_diskd = not\_defined diskcheck\_status or diskcheck\_status  
eq ERROR

attribute\_diskd\_inner = not\_defined diskcheck\_status\_internal or  
diskcheck\_status\_internal eq ERROR

#logconv\_logfacility = daemon

**act\_rsc = prmEx, prmlp**

変換元のログファイル名  
を指定

変換後のログ  
ファイル名を指定

サービスリソースの最上位と  
最下位のリソースIDを設定

# 起動設定

inittabに pm\_logconv-hb 起動設定を追加し、respawnで起動させます。

## /etc/inittab

(省略)

:

```
logc:2345:respawn:/usr/share/pacemaker/pm_logconv/pm_logconv.py
```

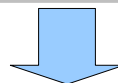
※ RHEL6系では、Upstartを使用する必要性があります。

# ログ変換例 (PostgreSQL起動時)

/var/log/ha-log

```
Jul 11 18:53:34 pm1 crmd: [1996]: info: do_lrm_rsc_op: Performing key=18:14:0:54ec38e9-bfac-4b29-9256-a9b9587456c6  
op=prmPg_start_0 )  
Jul 11 18:53:34 pm1 lrmd: [1993]: info: rsc:prmPg:63: start  
Jul 11 18:53:34 pm1 crmd: [1996]: info: process_lrm_event: LRM operation prmIp_monitor_10000 (call=62, rc=0, cib-update=68,  
confirmed=false) ok  
Jul 11 18:53:35 pm1 pgsql[19130]: INFO: server starting  
Jul 11 18:53:35 pm1 pgsql[19130]: INFO: PostgreSQL start command sent.  
Jul 11 18:53:35 pm1 pgsql[19130]: WARNING: psql: could not connect to server: No such file or directory Is the server  
running locally and accepting connections on Unix domain socket "/tmp/.s.PGSQL.5432"?  
Jul 11 18:53:35 pm1 pgsql[19130]: WARNING: PostgreSQL template1 isn't running  
Jul 11 18:53:35 pm1 pgsql[19130]: WARNING: Connection error (connection to the server went bad and the session was not  
interactive) occurred while executing the psql command.  
Jul 11 18:53:37 pm1 pgsql[19130]: INFO: PostgreSQL is started.  
Jul 11 18:53:37 pm1 crmd: [1996]: info: process_lrm_event: LRM operation prmPg_start_0 (call=63, rc=0, cib-update=69,  
confirmed=true) ok
```

/var/log/pm\_logconv.out



運用上必要なログだけを出力

```
Jul 11 18:53:34 pm1 info: Resource prmPg tries to start.  
Jul 11 18:53:37 pm1 info: Resource prmPg started. (rc=0)
```




# フェイルオーバー時のログ出力例

/var/log/pm\_logconv.out

```
Jul 11 19:02:15 pm2 ERROR: Start to fail-over.  
Jul 11 19:02:23 pm2 info: Resource prmEx tries to start.  
Jul 11 19:02:24 pm2 info: Resource prmEx started. (rc=0)  
Jul 11 19:02:24 pm2 info: Resource prmFs tries to start.  
Jul 11 19:02:24 pm2 info: Resource prmFs started. (rc=0)  
Jul 11 19:02:24 pm2 info: Resource prmPg tries to start.  
Jul 11 19:02:26 pm2 info: Resource prmPg started. (rc=0)  
Jul 11 19:02:24 pm2 info: Resource prmlp tries to start.  
Jul 11 19:02:24 pm2 info: Resource prmlp started. (rc=0)  
Jul 11 19:02:26 pm2 info: Resource prmEx : Move pm1 -> pm2  
Jul 11 19:02:26 pm2 info: Resource prmlp : Move pm1 -> pm2  
Jul 11 19:02:26 pm2 info: fail-over succeeded.
```

※ fail-overのログは、DCノード側のみ出力されます。



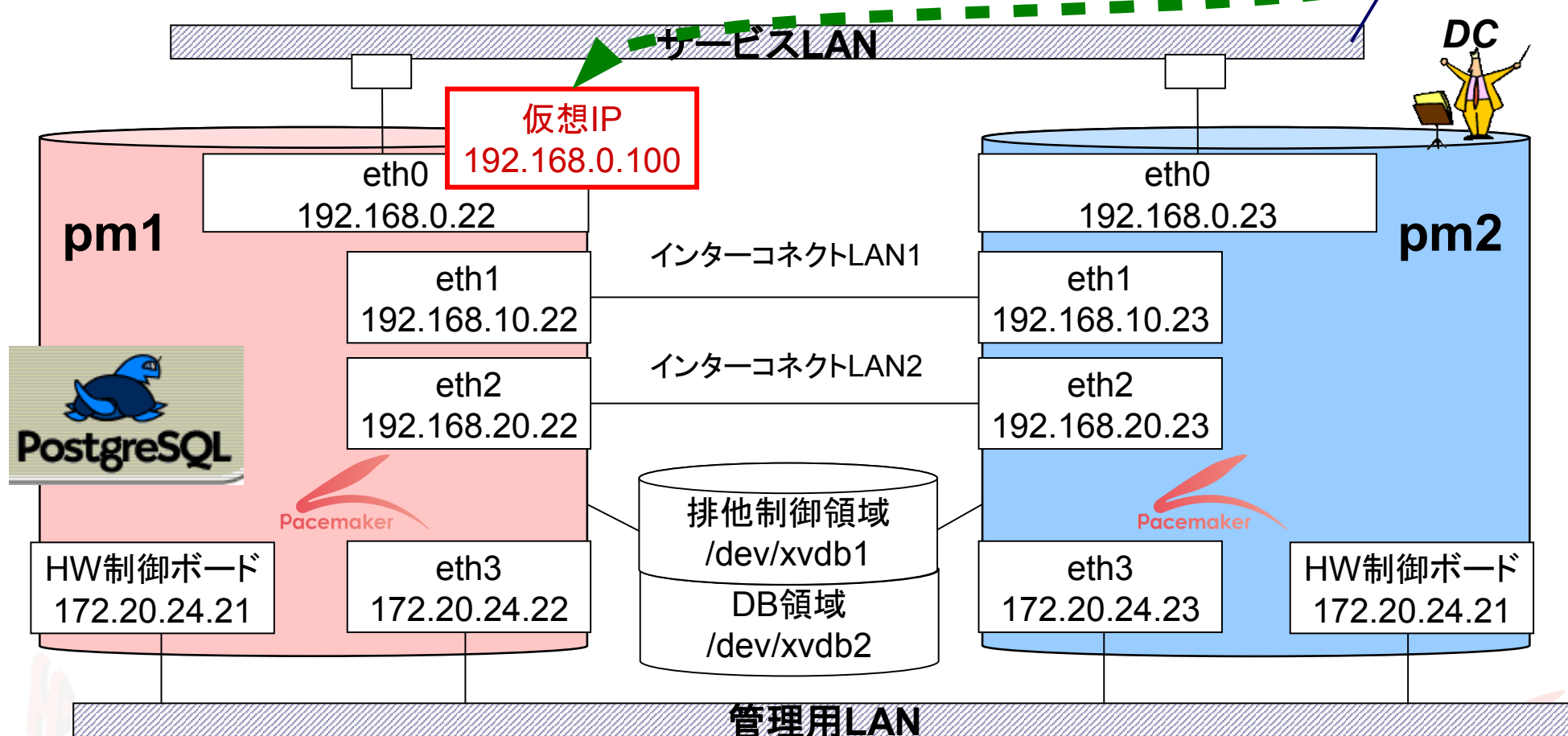
リソース設定をして  
サービスの起動と、  
本当にサービス  
が起動しているか  
デモします！

接続デモ例は  
次ページ



# PostgreSQLに接続...

```
demo# pgsql -U postgres -h 192.168.0.100 -c "SELECT now();"
demo(Domain-0)
```





⑤

いろいろ故障デモします！

# 故障デモに あたって...

”crm\_mon -fA”の結果は  
デモでは表示しきれないので、  
”crm\_mon -fA1”というワン  
ショットモードコマンドから  
一部をデモ目的に必要な部分を  
スクリプトで抜き出し、1秒毎に  
表示してデモを行います。

推奨ではないですが、デモの関  
係上Pacemakerは自動起動設  
定しています。

```
=====
Last updated: Fri May 25 14:59:57 2012
Stack: Heartbeat
Current DC: pm2 (7f1b5dcb-e696-414d-8fca-da79274b0a74) - partition with quorum
Version: 1.0.12-066152e
2 Nodes configured, unknown expected votes
6 Resources configured.
=====

Online: [ pm1 pm2 ]

Resource Group: grpPg
  prmEx      (ocf::heartbeat:sfex):    Started pm1
  prmFs      (ocf::heartbeat:Filesystem): Started pm1
  prmPg      (ocf::heartbeat:pgsql):   Started pm1
  prmIp      (ocf::heartbeat:IPaddr2):  Started pm1
Resource Group: grpStonith1
  prmStonith1-1 (stonith:external/stonith-helper): Started pm2
  prmStonith1-2 (stonith:external/libvirt):      Started pm2
  prmStonith1-3 (stonith:meatware):                    Started pm2
Resource Group: grpStonith2
  prmStonith2-1 (stonith:external/stonith-helper): Started pm1
  prmStonith2-2 (stonith:external/libvirt):      Started pm1
  prmStonith2-3 (stonith:meatware):                    Started pm1
Clone Set: clnDisk1
  Started: [ pm1 pm2 ]
Clone Set: clnDisk2
  Started: [ pm1 pm2 ]
Clone Set: clnPingd
  Started: [ pm1 pm2 ]

Node Attributes:
* Node pm1:
  + default_ping_set           : 100
  + diskcheck_status           : normal
  + diskcheck_status_internal  : normal
  + pm2-eth1                   : up
  + pm2-eth2                   : up
* Node pm2:
  + default_ping_set           : 100
  + diskcheck_status           : normal
  + diskcheck_status_internal  : normal
  + pm1-eth1                   : up
  + pm1-eth2                   : up

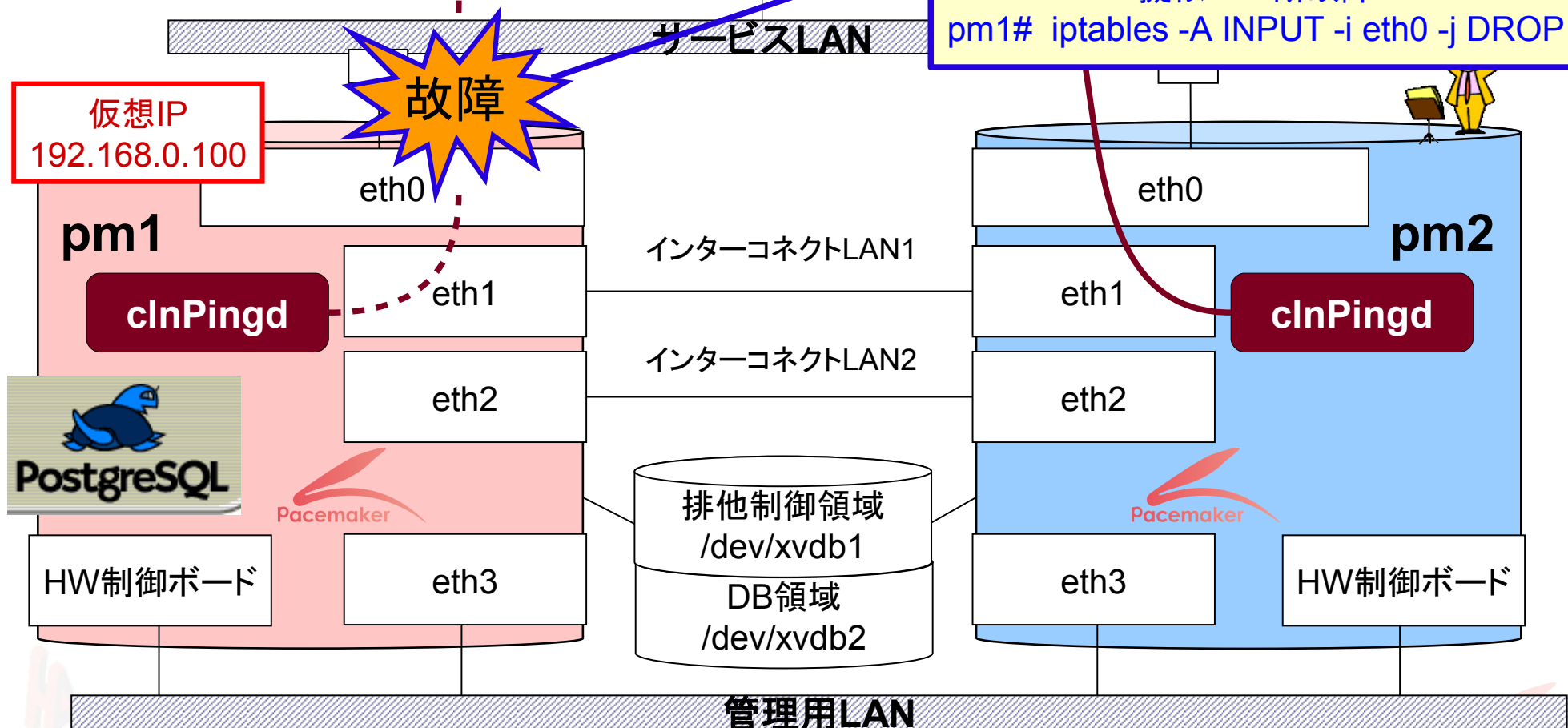
Migration summary:
* Node pm2:
* Node pm1:
```

# サービスLAN故障させてみる…

crm\_mon結果は  
次ページ

ping監視先(pdumy0)  
192.168.0.21

擬似LAN断故障  
pm1# iptables -A INPUT -i eth0 -j DROP



```
# crm_mon -fA
```

～ 省略 ～

Node Attributes:

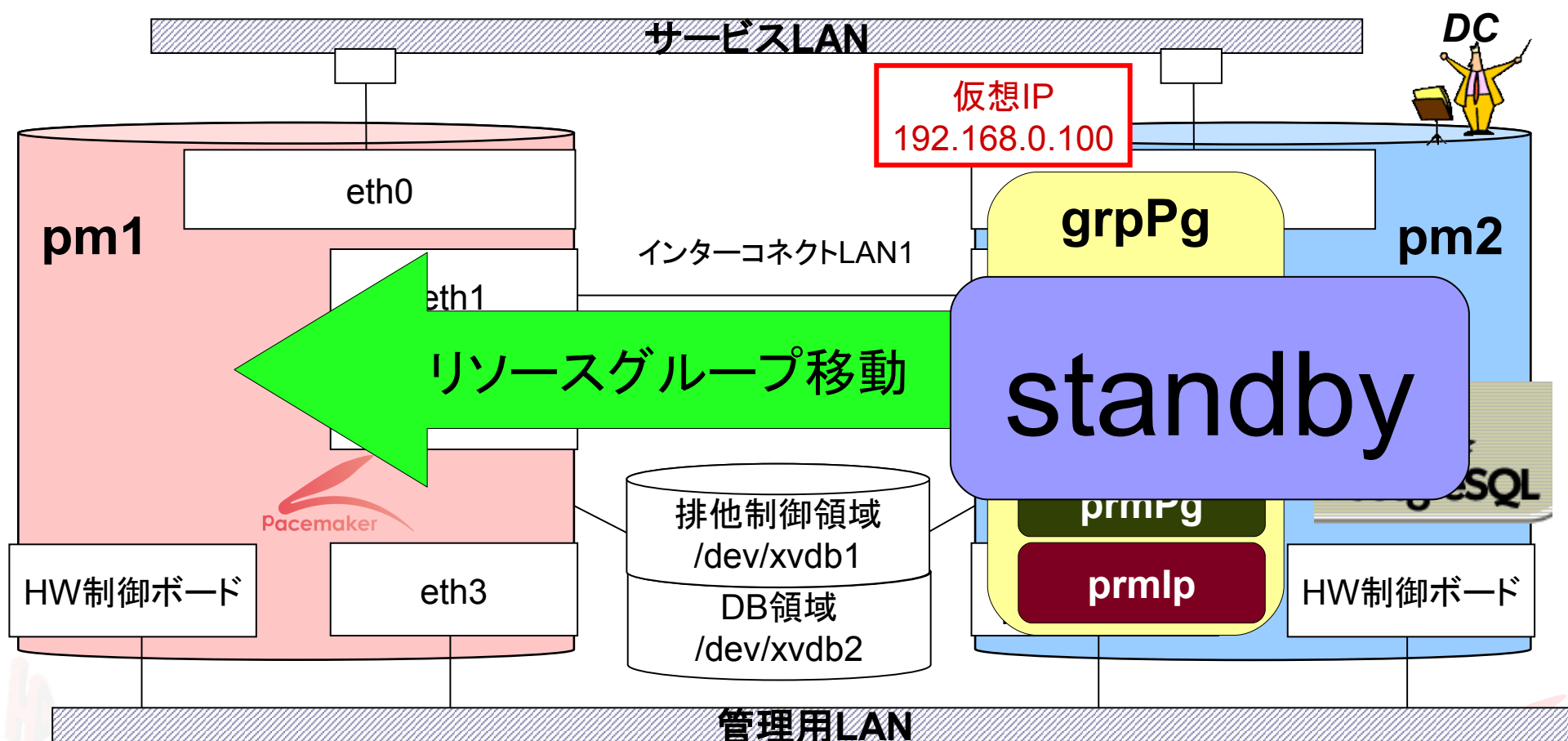
\* Node pm1:

+ default_ping_set	: 0	: Connectivity is lost
+ diskcheck_status	: normal	
+ diskcheck_status_internal	: normal	

サービスLAN故障  
を表示

# スタンバイ化して リソースグループ移動させてみる…

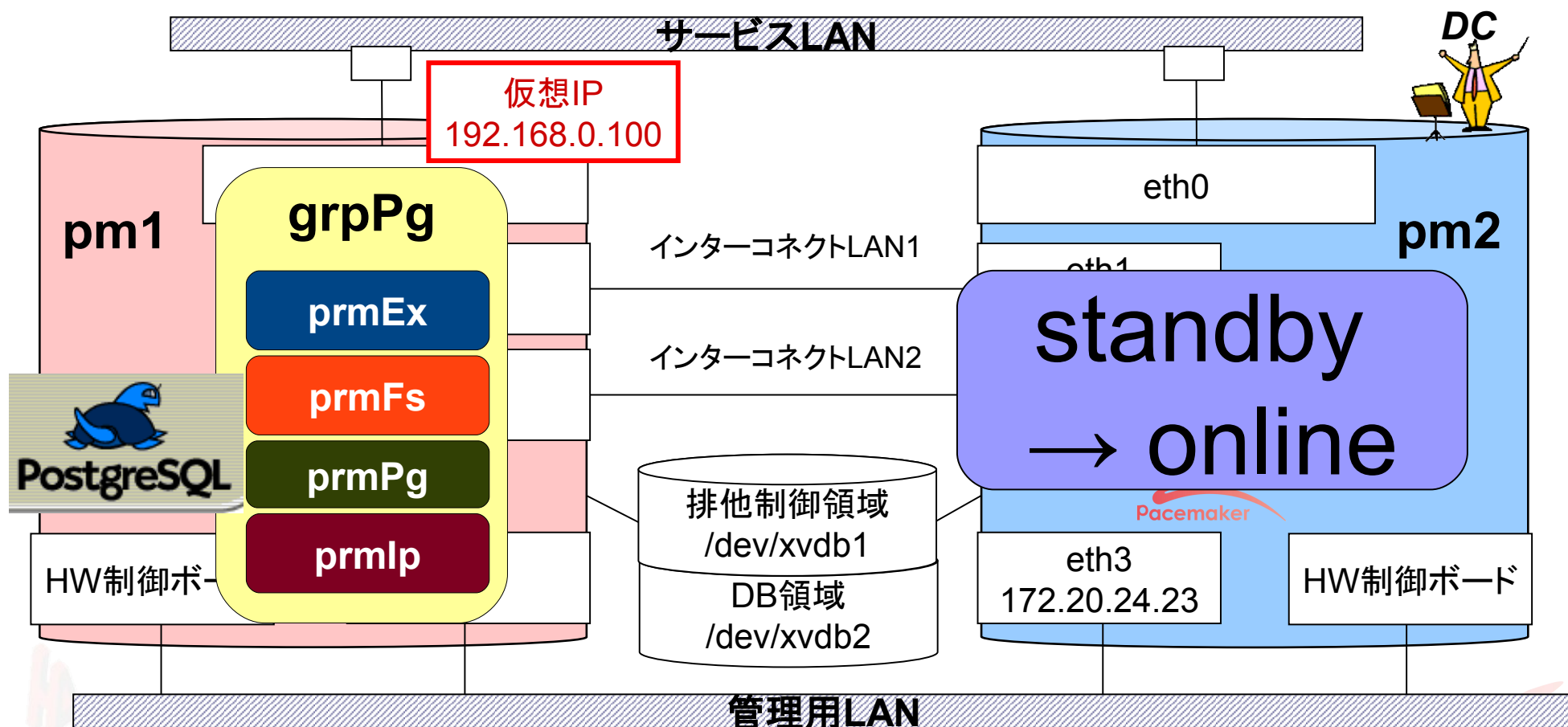
# crm node standby pm2





# オンライン化を忘れずに

# crm node online pm2



# リソース故障させてみる…

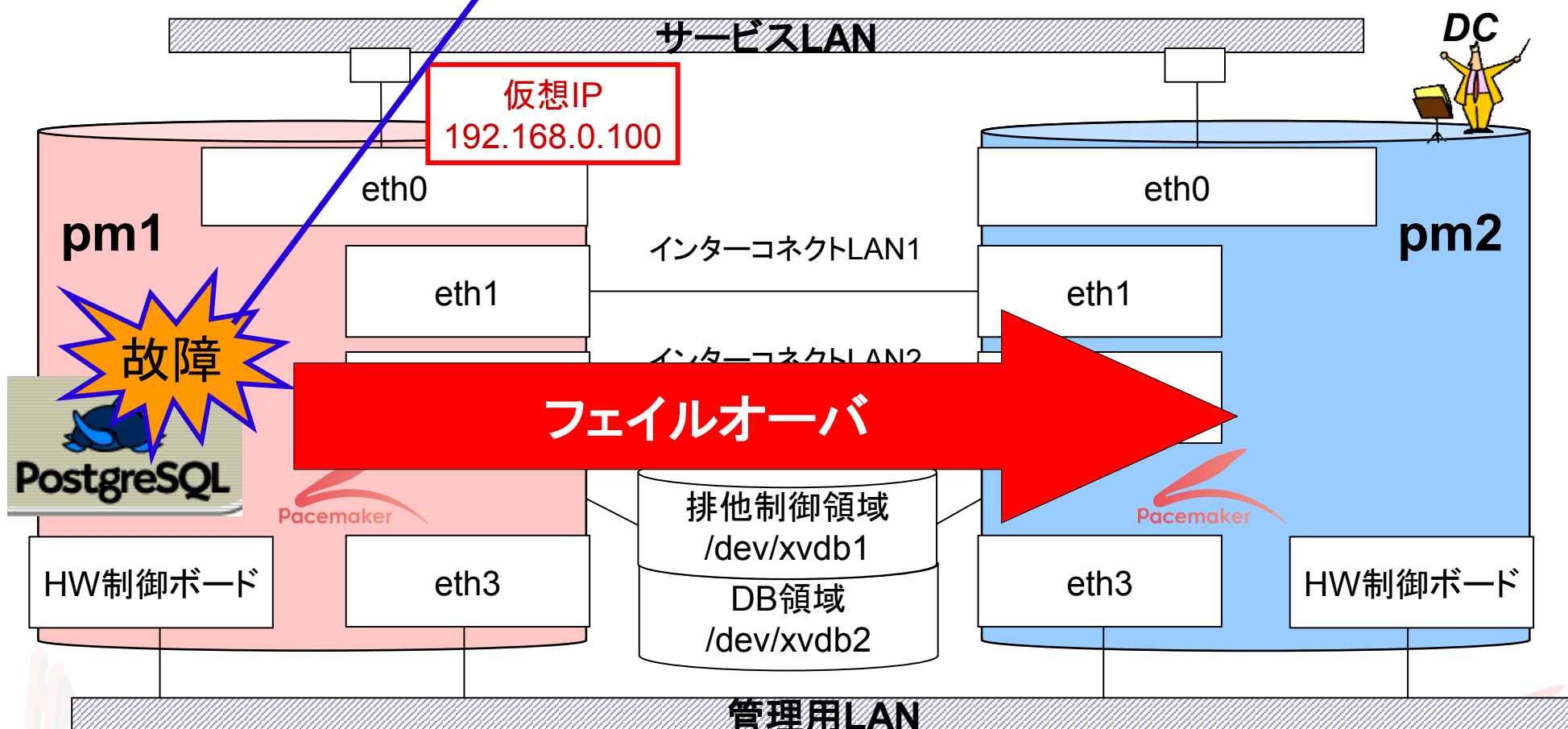
crm\_mon結果は  
次ページ

擬似故障  
# kill -9 postgresql親プロセス

仮想IP  
192.168.0.100

故障

フェイルオーバー



```
# crm_mon -fA
```

～ 省略 ～

Online: [ pm1 pm2 ]

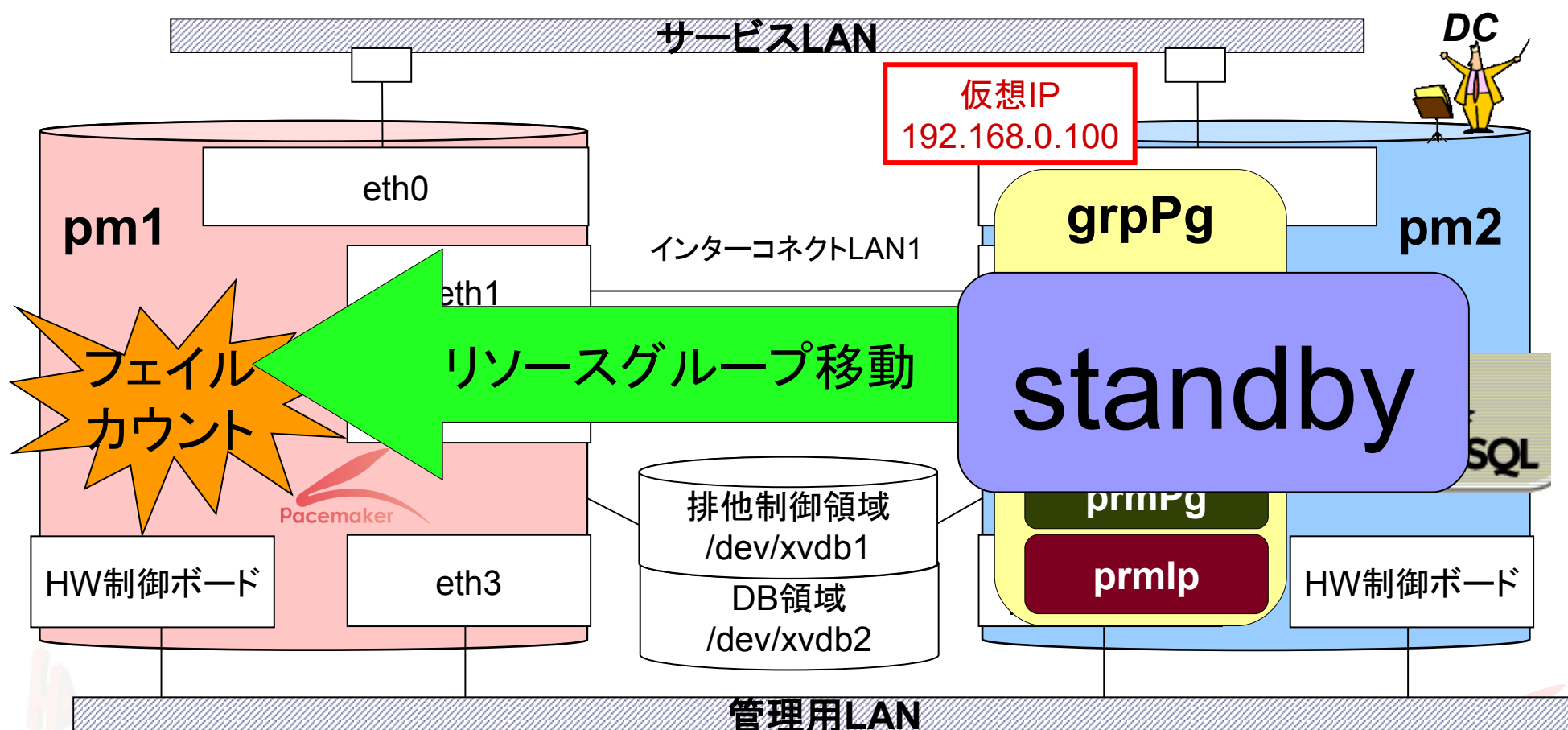
Resource Group: grpPg


prmEx	(ocf::heartbeat:sfex):	Started pm1
prmFs	(ocf::heartbeat:Filesystem):	Started pm1
prmPg	(ocf::heartbeat:pgsql):	Started pm1 FAILED
prmlp	(ocf::heartbeat:IPaddr2):	Started pm1

故障検出を表示

# この状態でスタンバイ化により リソースグループ移動させてみる…

# crm node standby pm2





切り替わらないのは  
ミスではありません！



故障を示す**フェイルカウント**がカウントアップされているため、クリアしなければ切り替わりません。

```
# crm_mon -fA
```

```
=====
```

```
~ 省略 ~
```

```
=====
```

Migration summary:

\* Node pm1:

prnPg: migration-threshold=1 fail-count=1

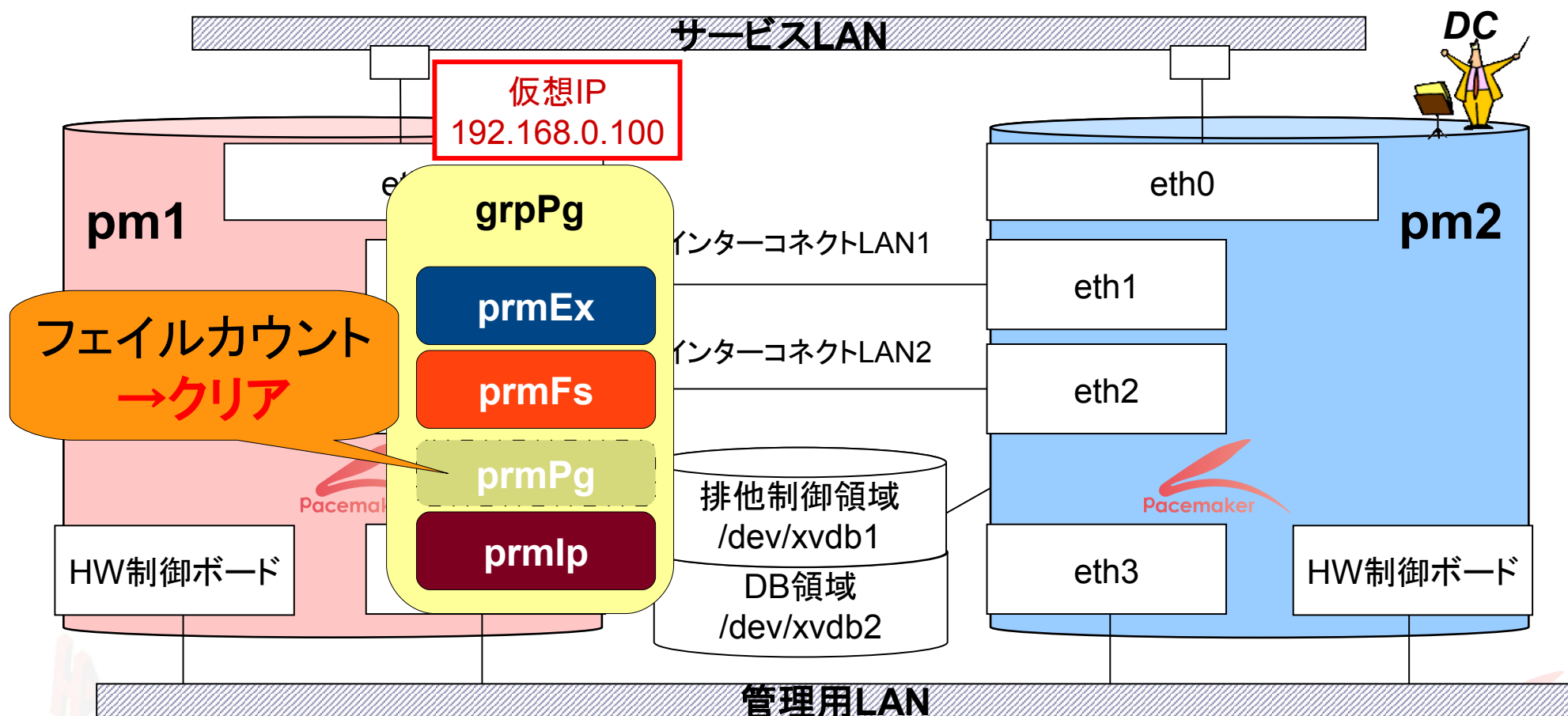
\* Node pm2:

Failed actions:

prnPg\_monitor\_10000 (node=pm1, call=34, rc=7, status=complete):  
not running

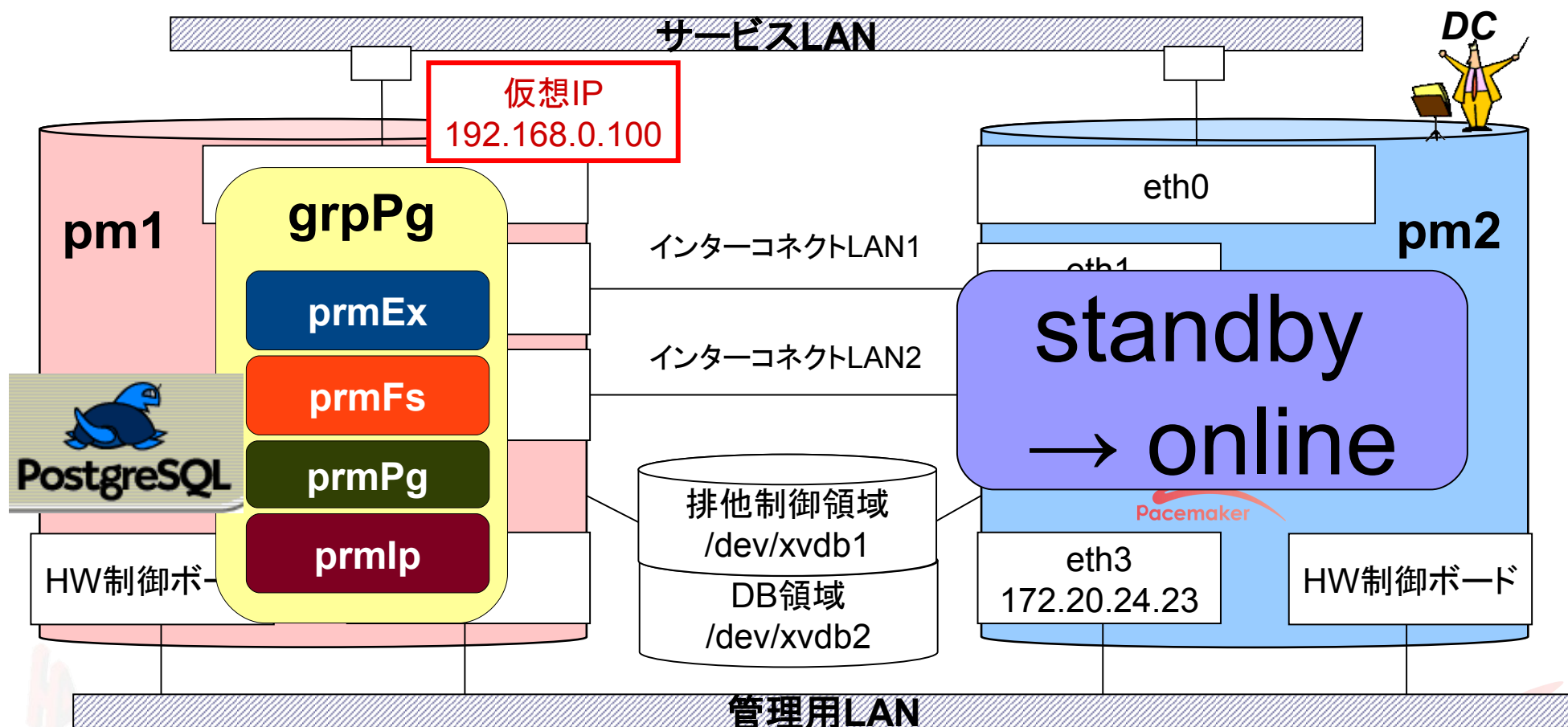
# フェイルカントをクリアしてみる…

```
# crm resource cleanup prmPg pm1
```



# またオンライン化を忘れずに

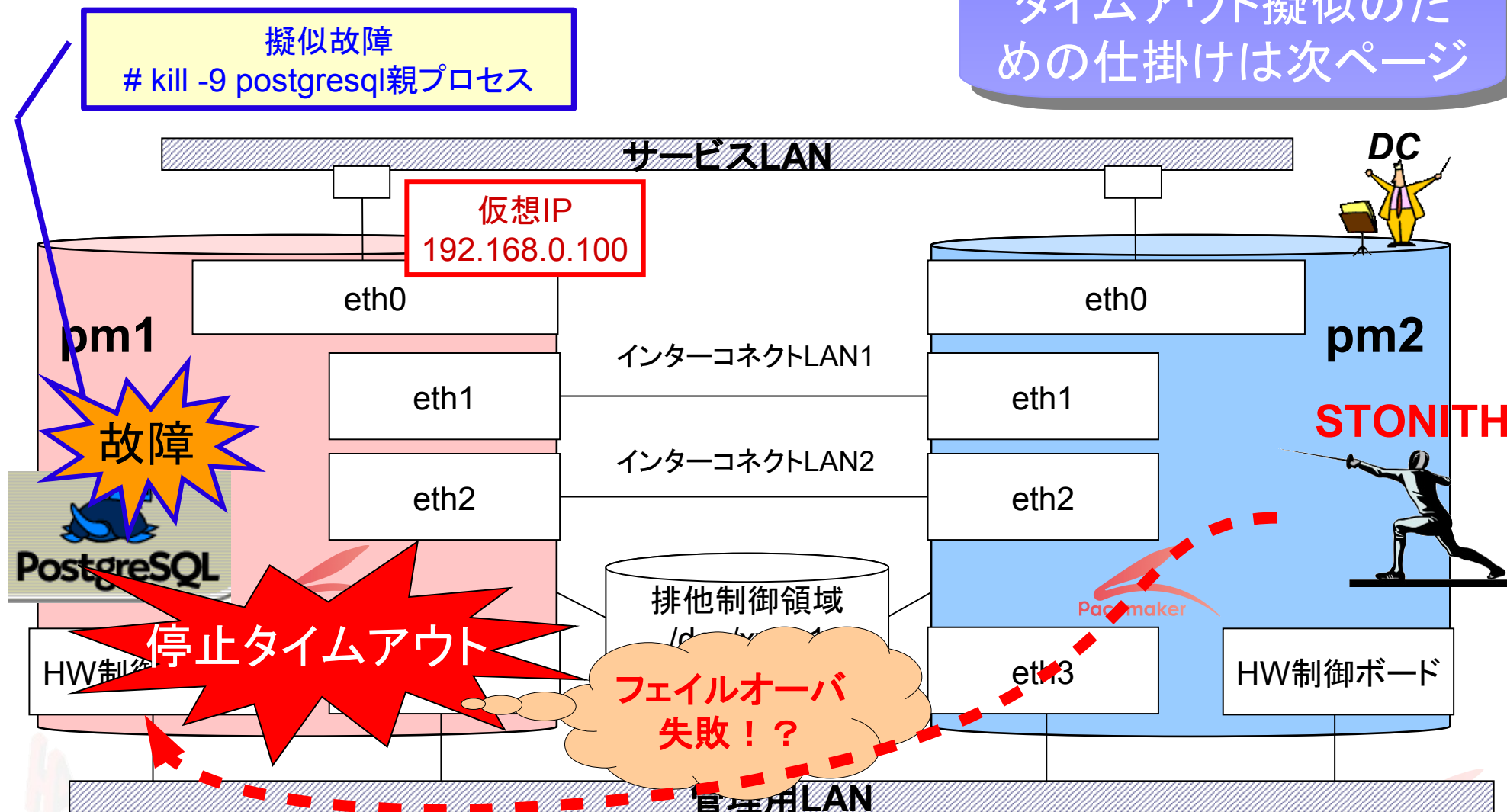
# crm node online pm2





# リソース故障時、停止タイムアウト...

タイムアウト擬似のための仕掛けは次ページ



# 停止タイムアウトデモのために、 こんな仕掛けします…

1. pgsqlリソースエージェントのstop制御部に sleep 60 をわざと入れます。

/usr/lib/ocf/resource.d/heartbeat/pgsql

```
381 #pgsql_stop: Stop PostgreSQL
382 pgsql_stop() {
383     local rc
384     sleep 60
385     if ! pgsql_status
386     then
387         #Already stopped
388         return $OCF_SUCCESS
389     fi
```

## 2. crmコマンドのeditモードで prmPg のストップタイムアウトを 60s から 10s に変更します。

初期構築後、値を変更したい場合に便利です

```
# crm configure edit
```

```
primitive prmPg ocf:heartbeat:pgsql ¥  
  params pgctl="/usr/pgsql-9.1/bin/pg_ctl" psql="/usr/pgsql-  
9.1/bin/psql" pgdata="/var/lib/pgsql/9.1/data"  
pgdba="postgres" pgport="5432" pgdb="template1" ¥  
  op start interval="0s" timeout="60s" on-fail="restart" ¥  
  op monitor interval="10s" timeout="60s" on-  
fail="restart" ¥  
  op stop interval="0s" timeout="10s" on-fail="fence"
```

```
# crm_mon -fA
```

Migration summary:

\* Node pm1:

prnPg: migration-threshold=1 fail-count=1

\* Node pm2:

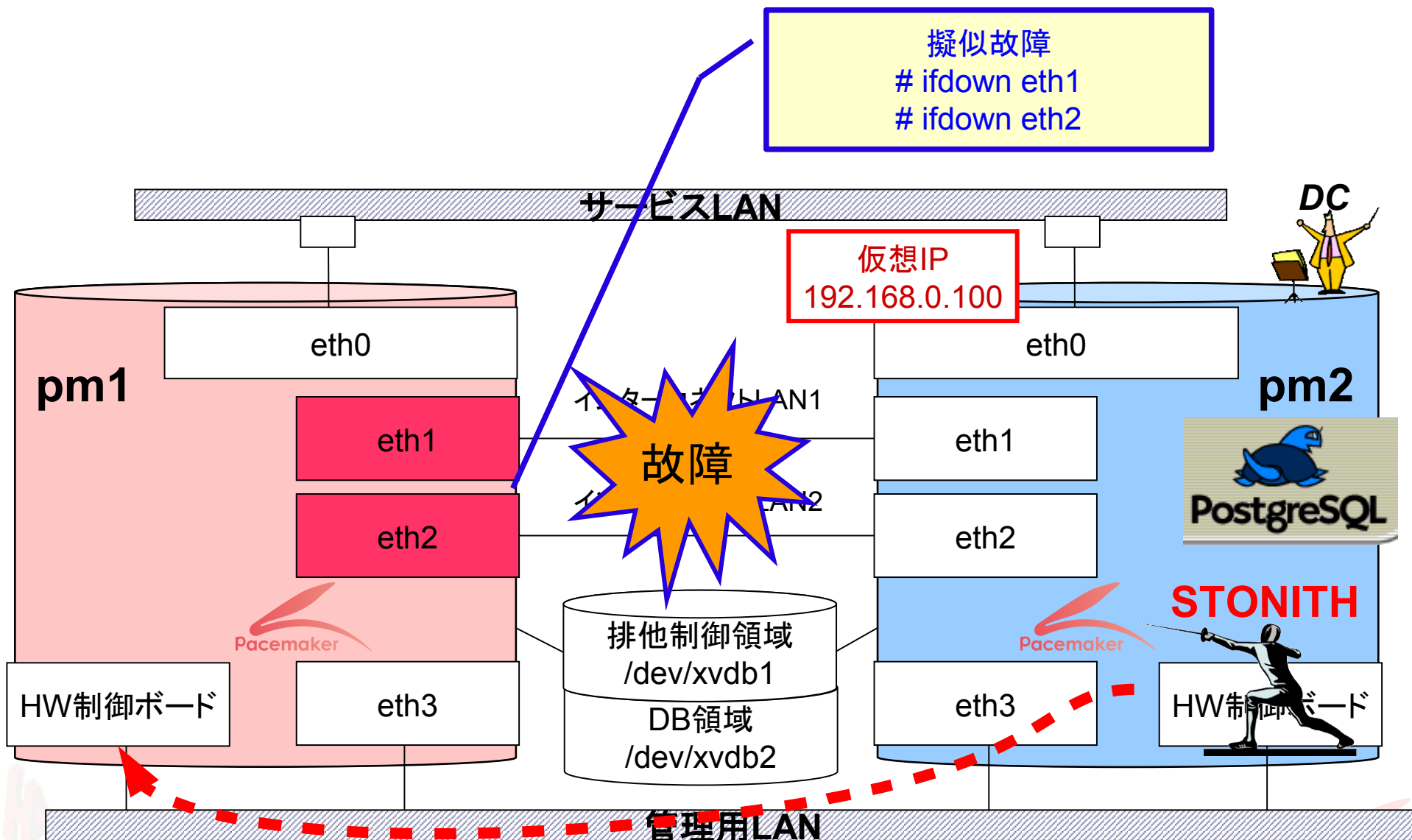
Failed actions:

prnPg\_monitor\_10000 (node=pm1, call=34, rc=7,  
status=complete): not running

prnPg\_stop\_0 (node=pm1, call=35, rc=-2, status=Timed Out):  
unknown exec error

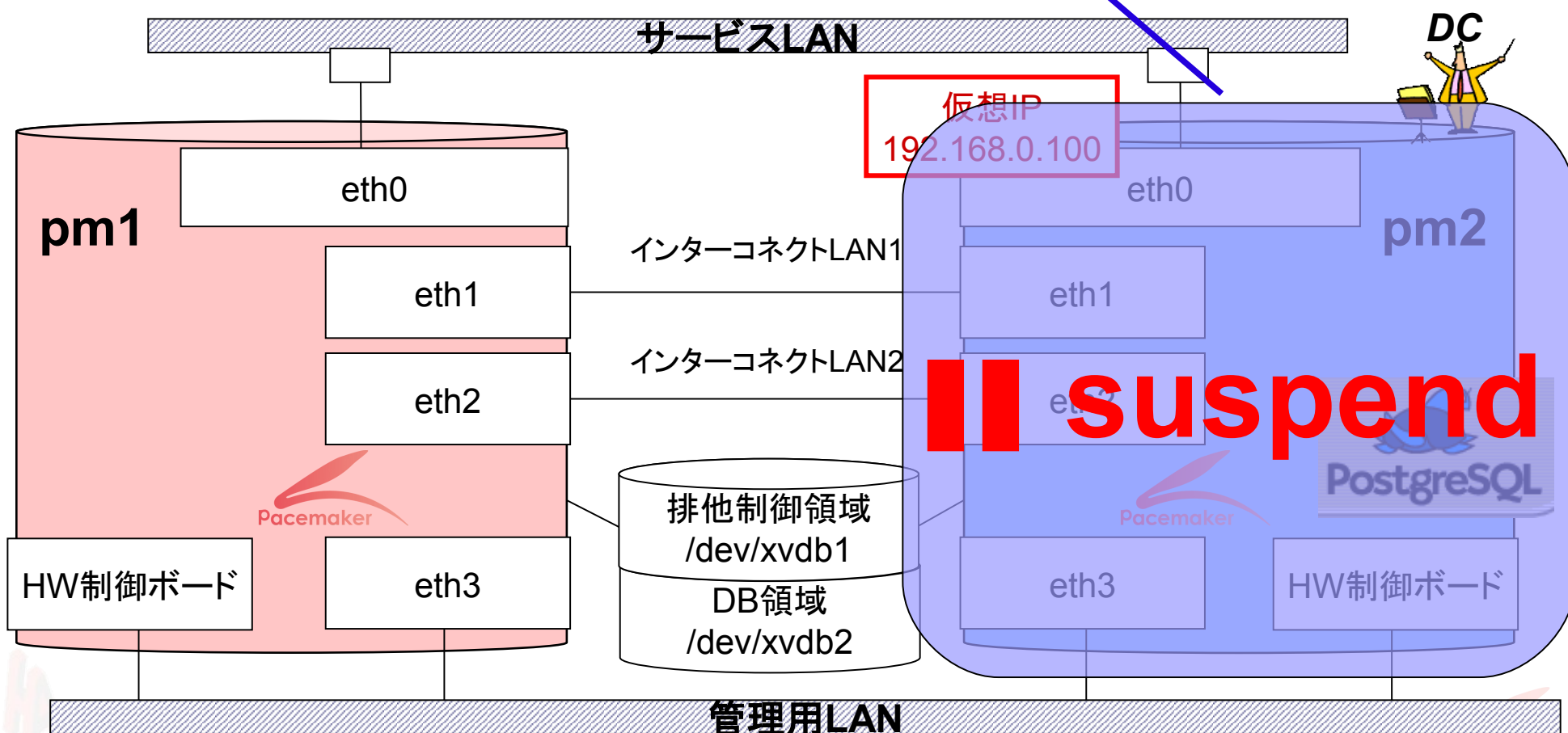
ストップタイムアウト状態  
を表示

# スプリットブレイン...

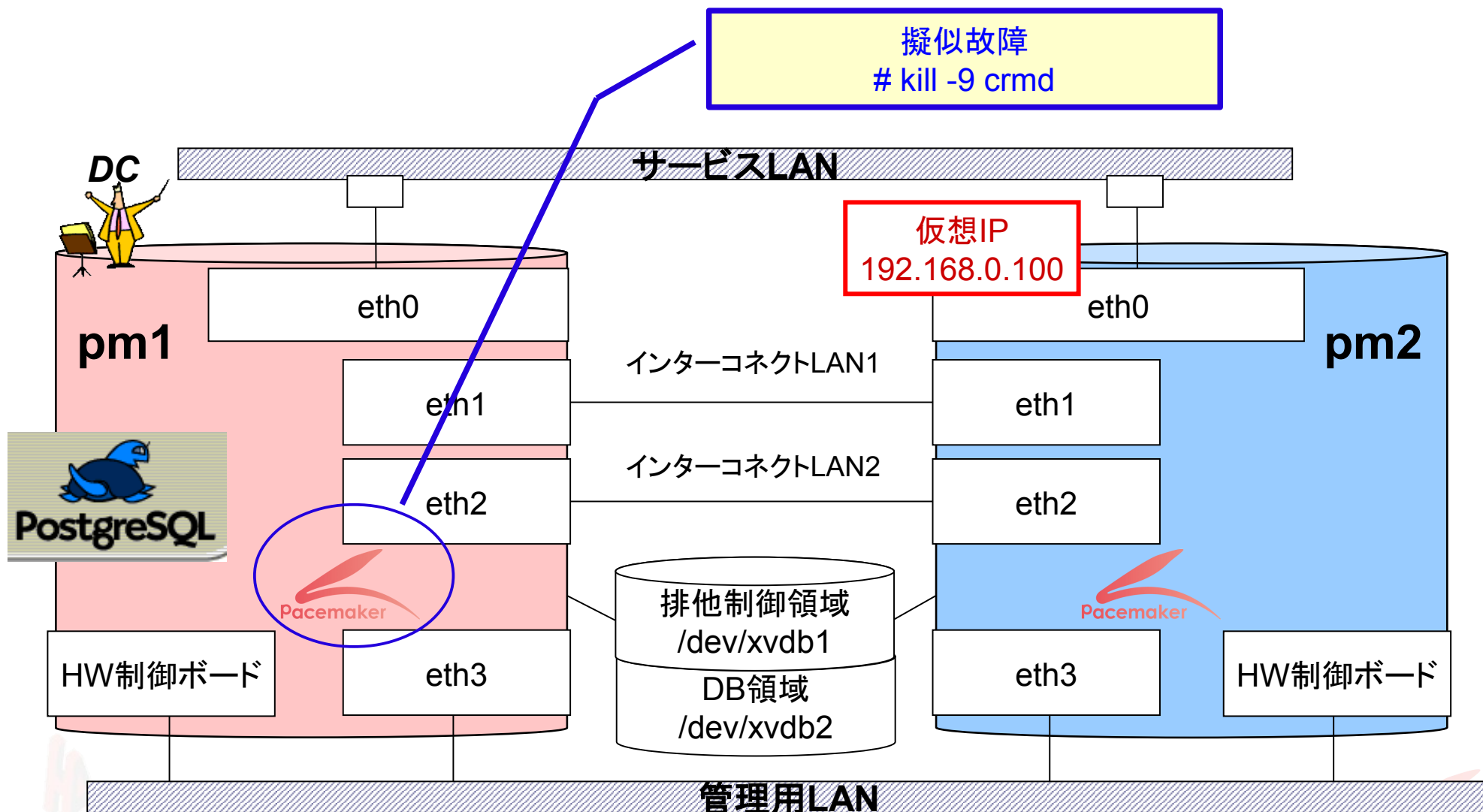


# ノード一時停止させてみる…

```
demo # virsh suspend pm2
```



# Pacemakerプロセス故障させてみる…



# Pacemakerプロセス故障時の挙動

(クラスタ制御機能にHeartbeat3使用の場合)

Pacemakerプロセス	プロセス故障時の挙動
heartbeat: master control process	サーバ再起動
ccm	
cib	
crmd	
lrmd	
pengine	
heartbeat: FIFO reader	プロセス再起動
heartbeat: write: bcast ethX	
heartbeat: read: bcast ethX	
stonithd	
attrd	
ifcheckd	





⑥

# Linux-HA Japanについて

# Linux-HA Japanの経緯

『Heartbeat(ハートビート)』の日本における更なる普及展開を目的として、2007年10月5日「Linux-HA (Heartbeat) 日本語サイト」を設立し、日本でのLinux-HA コミュニティ活動として、Heartbeat2のrpmバイナリや機能追加用パッケージを提供してきました。

現在は、Pacemakerリポジトリパッケージの提供や、PostgreSQL 9.1 Streaming Replication に対応したリソースエージェント開発などの活動を行っています。

このRAが入った resource-agents-3.9.3 は  
日本時間で本日までたくリリース！

# Linux-HA Japan URL

<http://linux-ha.sourceforge.jp/>

(一般向け)

<http://sourceforge.jp/projects/linux-ha/> (開発者向け)



Pacemaker情報の公開用として  
随時情報を更新中です。

このサイトより、Pacemakerリポジトリ  
パッケージがダウンロード可能です。

# 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、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.sourceforge.jp/>  
の「メーリングリスト」をクリック



- MLアドレス

**linux-ha-japan@lists.sourceforge.jp**

※スパム防止のために、登録者以外の投稿は許可制です

# 本家Pacemakerサイト

<http://clusterlabs.org/>

Fedora, openSUSE,  
EPEL(CentOS/RHEL)  
のrpmがダウンロード  
可能です。

## Pacemaker 1.0.x - Supported Versions/Distributions

Binary packages for current Fedora, OpenSUSE and EPEL compatible distributions (eg. RHEL, CentOS and Scientific releases):

### Fedora

- 10 [repository] [i386] [src] [x86\_64]
- 11 [repository] [i386] [src] [x86\_64]
- 12 [repository] [i386] [src] [x86\_64]
- 13 [repository] [i386] [src] [x86\_64]
- 14 [repository] [src] [x86\_64]
- rawhide [repository] [src] [x86\_64]

### openSUSE

- 11.0 [repository] [i386] [src] [x86\_64]
- 11.1 [repository] [i386] [src] [x86\_64]
- 11.2 [repository] [i386] [src] [x86\_64]
- 11.3 [repository] [i386] [src] [x86\_64]

### EPEL

- 4 [repository] [i386] [src] [x86\_64]
- 5 [repository] [i386] [src] [x86\_64]

<http://clusterlabs.org/rpm>

ぢつは本家の  
Pacemakerのロゴはこれでした...



しかし

これ  では、

いかにも医療機器っぽいので…



# Pacemakerロゴ

Linux-HA Japan では、  
Pacemakerのロゴ・バナーを独自に作成





# 本家Pacemakerロゴに勝負を挑みました！

<http://theclusterguy.clusterlabs.org/post/1551578523/new-logo>

Cluster Guy New logo

検索



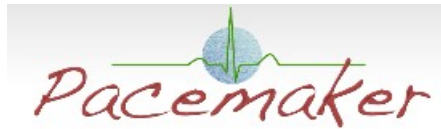
VS



# すると新ロゴが 圧倒的リードで勝利したのです！



VS



## Which Logo is Better?



New

74%



Old

26%



**YOU'RE  
WINNER!**

しかし！？

# 本家Pacemaker新ロゴは、 青になるというオチに…

The screenshot shows the Pacemaker website with the new blue logo. The header includes "clusterlabs menu" and "site-search:". The main banner features the logo and the text "A scalable High-Availability cluster resource manager". Below the banner, there are navigation tabs: "The Team", "Overview" (selected), "Features", "FAQ", and "Explore". The "Overview" section describes Pacemaker as an Open Source, High Availability resource manager. To the right, the "Deployment Examples" section shows a diagram titled "Active / Passive" illustrating a cluster architecture with Services, Cluster Software (Pacemaker, CoroSynchrony), and Hardware (Hosts).

clusterlabs menu site-search:

**Pacemaker**

A scalable High-Availability cluster resource manager

The Team **Overview** Features FAQ Explore

**Overview**

Pacemaker is an [Open Source](#), [High Availability](#) resource manager suitable for both small and large [clusters](#).

Hardware and application failures can result in prolonged downtime and impact your bottom line.

In the event of a failure, resource managers like Pacemaker automatically initiate recovery and make sure your application is available from one of the remaining machines in the cluster.

Your users may never even know there was a problem.

**Deployment Examples**

**Active / Passive**

Services: URL, Web Site, Files, Storage, D/base, D/base, Storage

Cluster Software: **Pacemaker**, CoroSynchrony

Hardware: Host, Host, Host, Host

Click image to enlarge

ご清聴ありがとうございました。



Linux-HA Japan

検索

<http://linux-ha.sourceforge.jp/>