Pacemaker-1.0とは違うのだよ、1.0とは！
～Pacemaker-1.1新機能のご紹介～

2015年2月28日 OSC2015 Tokyo/Spring

Linux-HA Japan
竹下　雄大

HighAvailability Pacemaker

本日の内容

- Pacemakerってなに？
- Pacemaker-1.1の特徴
- Pacemaker-1.1の性能
  - 最大ノード数
  - 最大リソース数
  - スイッチオーバ時間
- Pacemaker-1.1の新機能
  - kdump連携機能
  - Pacemaker Remote
  - リソース配置戦略機能
- TIPS

Pacemakerってなに？

Pacemakerはオープンソースの
HAクラスタソフトです。

Pacemakerってなに？

High Availability = 高可用性
サービス継続性

つまり

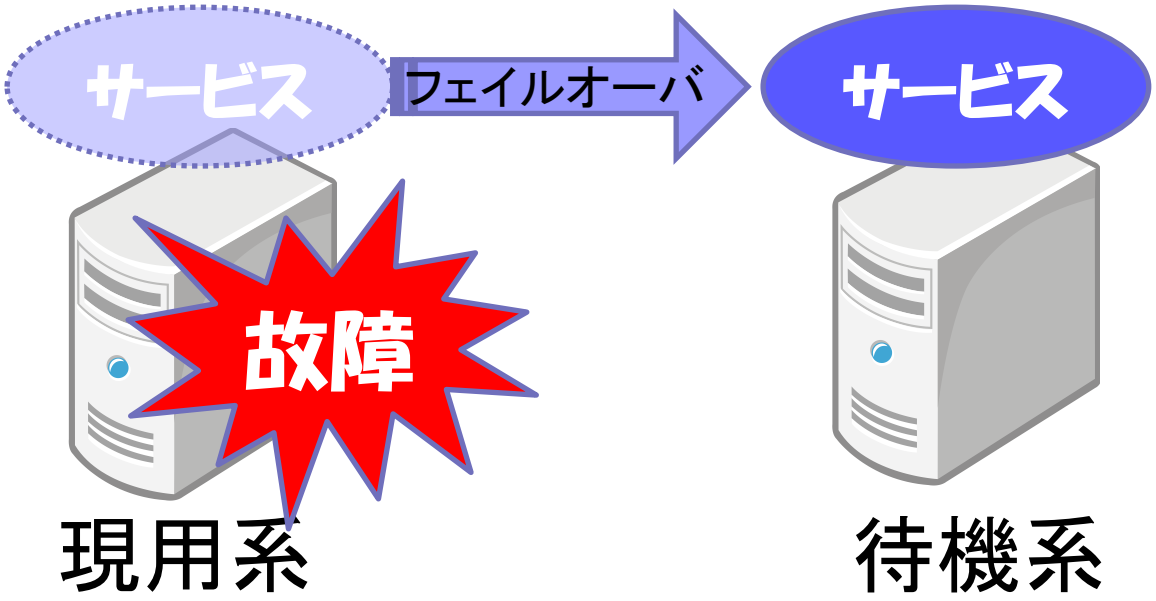
一台のコンピュータでは得られない高い信頼性を狙うために、
複数のコンピュータを結合(クラスタ化)し、
ひとまとまりとする…

ためのソフトウェアです

Pacemakerってなに？

HAクラスタを導入すると、
故障で現用系でサービスができなくなったときに、自動で待機系でサービスを起動させます

→このことを**「フェイルオーバ」**と言います
(以降、F.Oと表記)

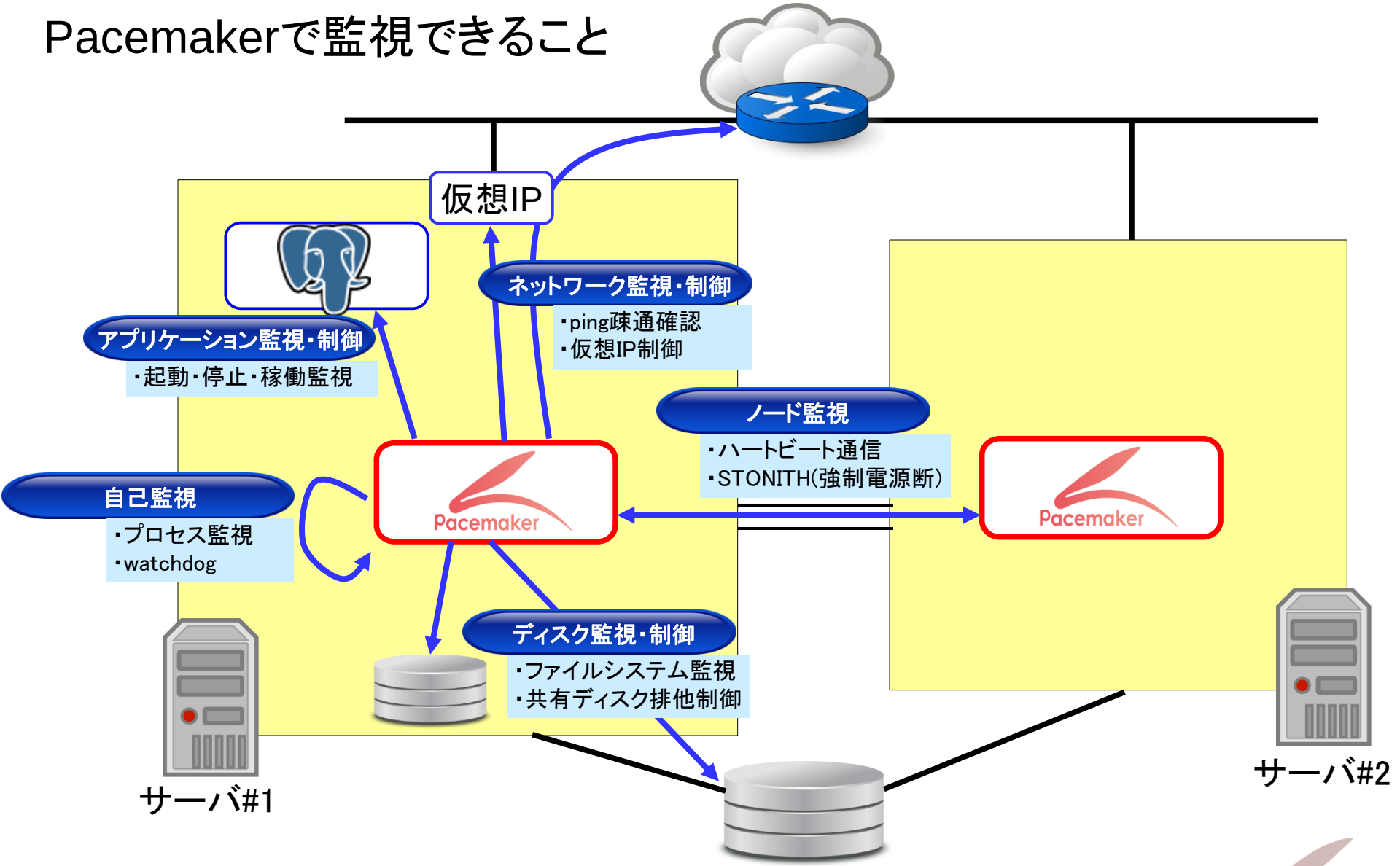


Pacemakerってなに？

はこのHAクラスタソフトとして実績のある「Heartbeat」と呼ばれていたソフトの後継です。

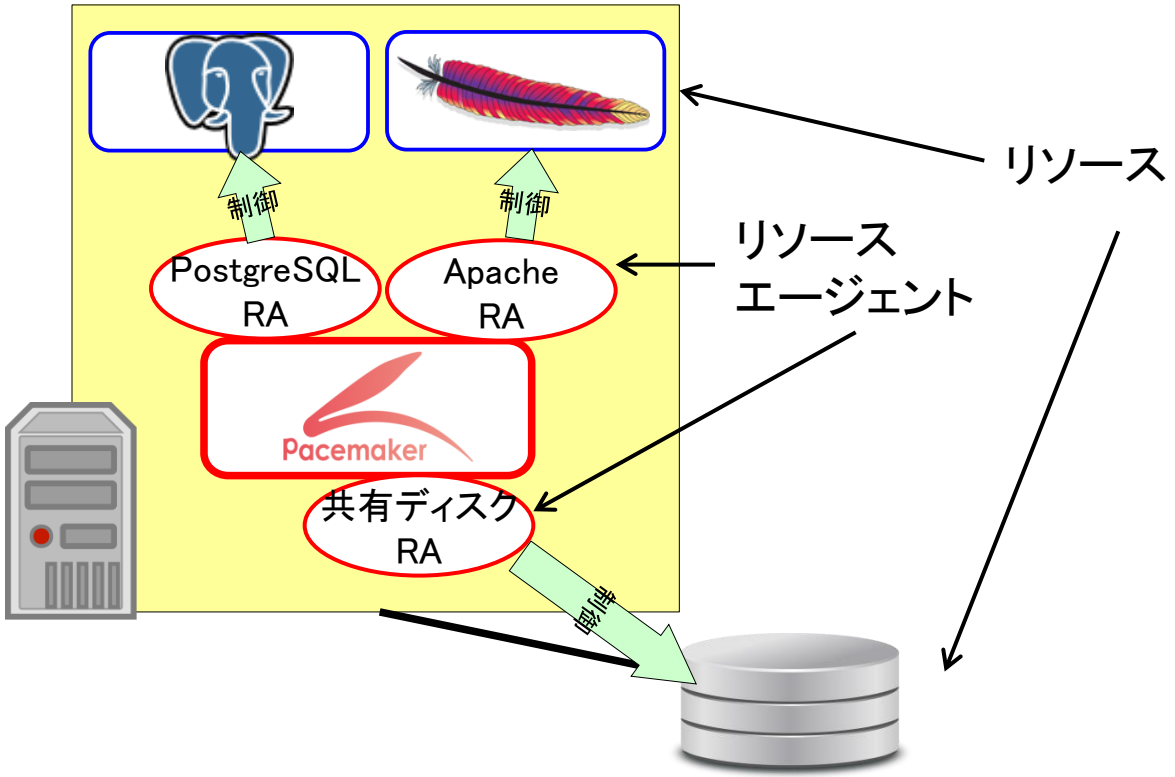
Pacemakerってなに？

Pacemakerで監視できること



Pacemakerってなに？

- ✓ Pacemakerが起動/停止/監視を制御する対象を**リソース**と呼ぶ
  - ✓ 例：Apache、PostgreSQL、共有ディスク、仮想IPアドレス…
- ✓ リソースの制御は**リソースエージェント(RA)**を介して行う
  - ✓ RAが各リソースの操作方法の違いをラップし、Pacemakerで制御できるようにしている
  - ✓ 多くはシェルスクリプト



ここまではPacemaker-1.0も
Pacemaker-1.1も同じです。
次から、違いをお話しします。

Pacemaker-1.1の特徴

まず最初に・・・

【重要なお知らせ】
Pacemaker-1.0は今後、本家コミュニティでのメンテナンス、およびリリースがされません。

Andrew氏(※1)の発言(2014/05/15):

> Code for the older 1.0 series of Pacemaker
> After lingering on in a zombie like state for a number of years, this codebase is now officially retired.
> (略)
> It had a good run but like all good things must come to an end.

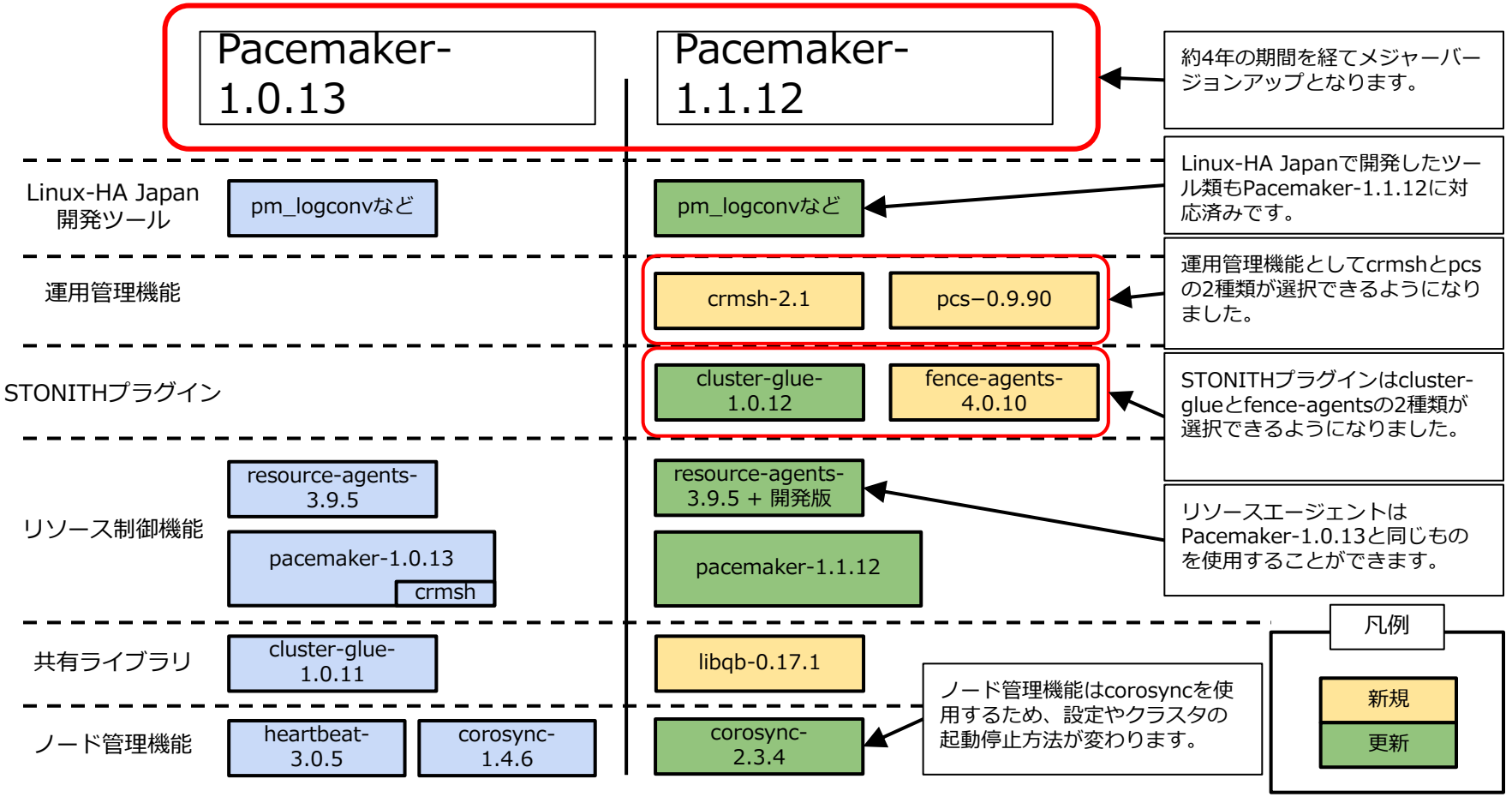
全文は下記参照
https://github.com/ClusterLabs/pacemaker-1.0/blob/master/README.md

これから新規導入を検討されている方は、Pacemaker-1.1系の利用をお勧めします！
MLでは1.0系の話題も歓迎です！

(※1) Pacemakerコミュニティを立ち上げた偉い人

Pacemaker-1.1の特徴

Pacemaker-1.1では、コンポーネントが変わりました！ (※1)



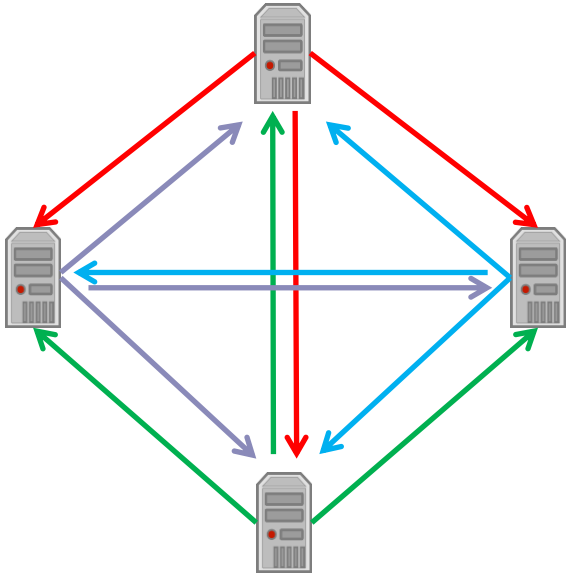
(※1) 図はOSC 2014 Tokyo/Fallの講演資料より引用

✓ 運用管理機能にはcrmshを利用する前提でお話しします

Pacemaker-1.1の性能

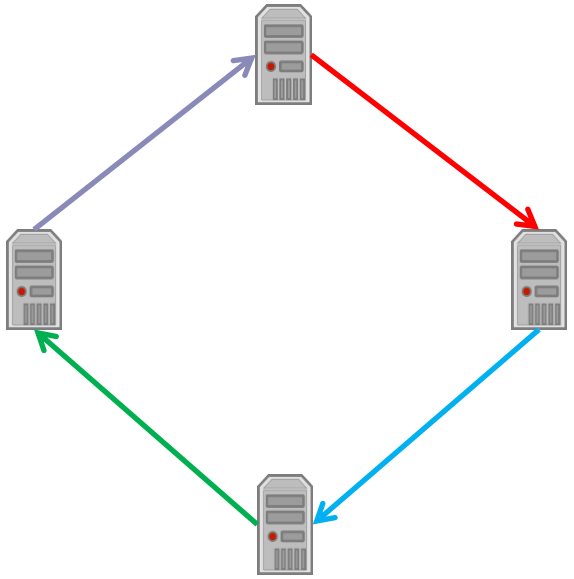
- ノード管理部の変更(Heartbeat → Corosync)等により、Pacemaker-1.1は大幅な性能向上を果たしました！
  - ✓ ノード間通信方式の変更
  - ✓ クラスタ/リソース構成情報に関する処理の高速化
  - ✓ throttle機能(※1)
  - ✓ etc…

Pacemaker-1.0の通信方式



各ノードがブロードキャストで全ノードと通信

Pacemaker-1.1の通信方式



各ノードはユニキャストで次のノードと通信

(※1) Pacemakerが30秒ごとにCPU負荷などを計測し、負荷状況に応じてジョブの同時実行数を制限する機能

Pacemaker-1.1の性能

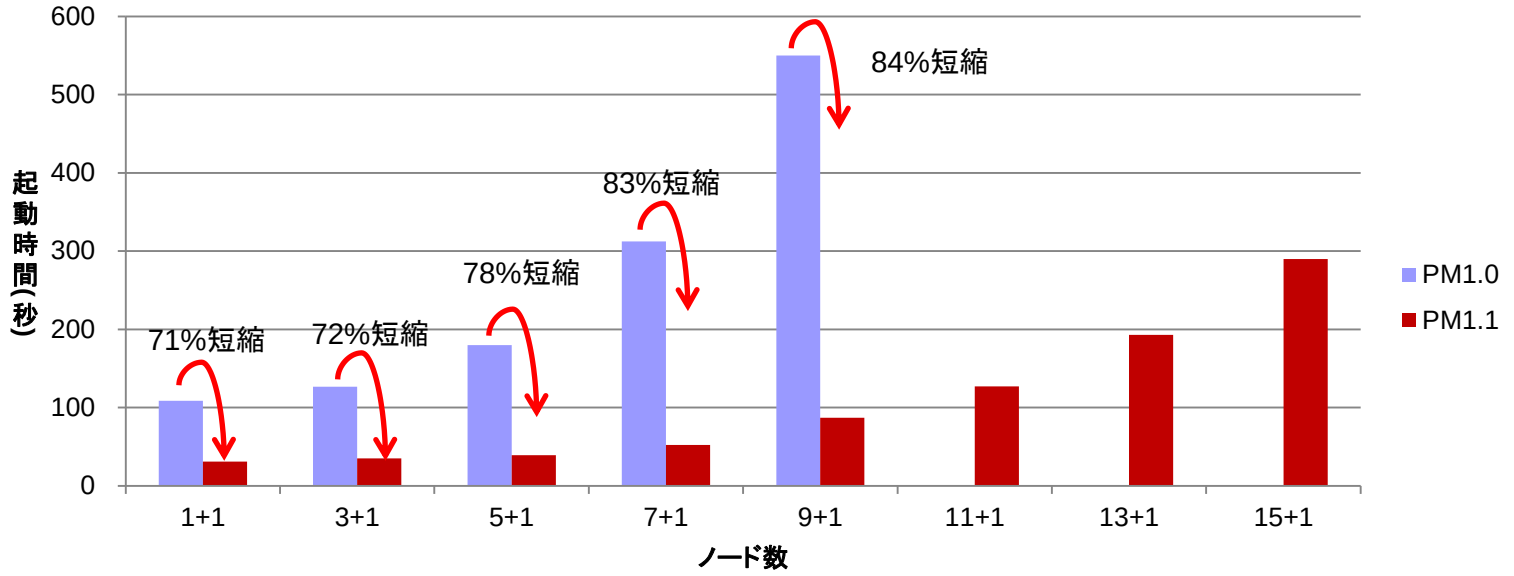
- どのくらい変わったのか、下記の環境で測定しました
  - ✓ 最大ノード数
  - ✓ 最大リソース数
  - ✓ 起動時間
  - ✓ スイッチオーバ時間

少し古いデータですが、1.0、1.1共に性能面で大きな変化はないはず・・・

Pacemaker	Pacemakerリポジトリパッケージ 1.0.13-1.1	pacemaker.x86_64 1.1.12-0.1.7f96b00.git.el6
OS	RHEL6.4(x86_64)	
ハードウェア	CPU：Xeon E5-2603 1.80GHz × 4	
	メモリ：16GB	
仮想マシン	OS同梱のKVMを使用	
	ノードあたりのHWリソースは以下	
	CPU：1コア	
	メモリ：2GB	

Pacemaker-1.1の性能：最大ノード数

■ 最大ノード数：クラスタ起動開始からリソース起動完了までの時間を計測
  ✓ 1ノード15リソースを稼働

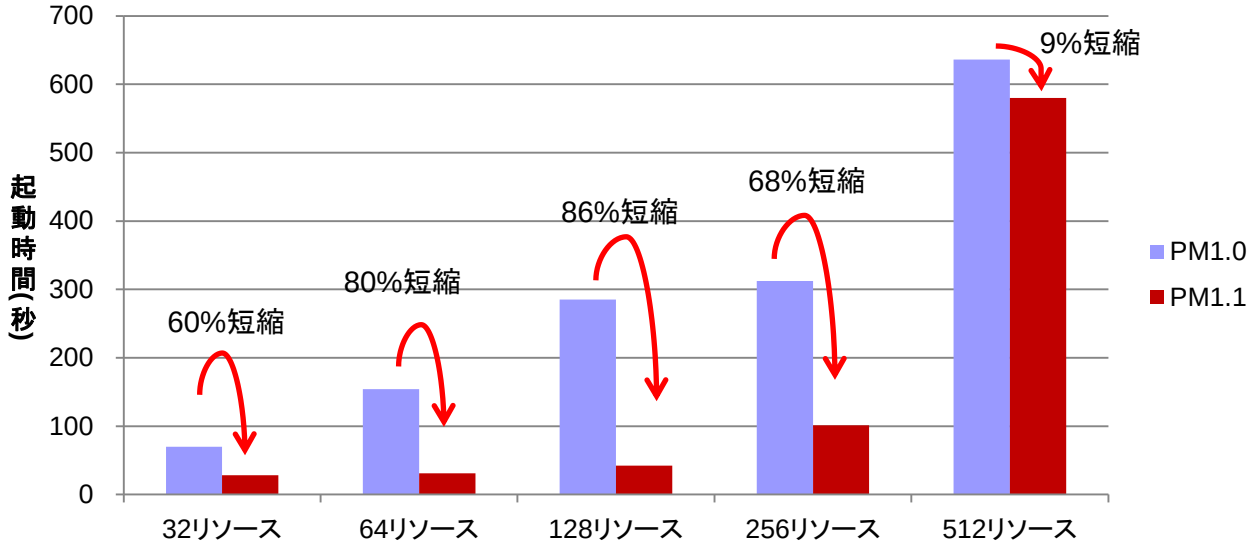


起動時間(秒)	1+1	3+1	5+1	7+1	9+1	11+1	13+1	15+1
1.0	108	127	180	312	550	起動不可		
1.1	31	35	39	52	87	127	193	290

✓ Pacemaker-1.1では、12ノード以上でも起動可能
✓ Pacemaker-1.1の起動時間は、Pacemaker-1.0より7～8割程度短縮

Pacemaker-1.1の性能：最大リソース数

■ 最大リソース数：クラスタ起動からリソース起動完了までの時間を計測
  ✓ 1＋1構成

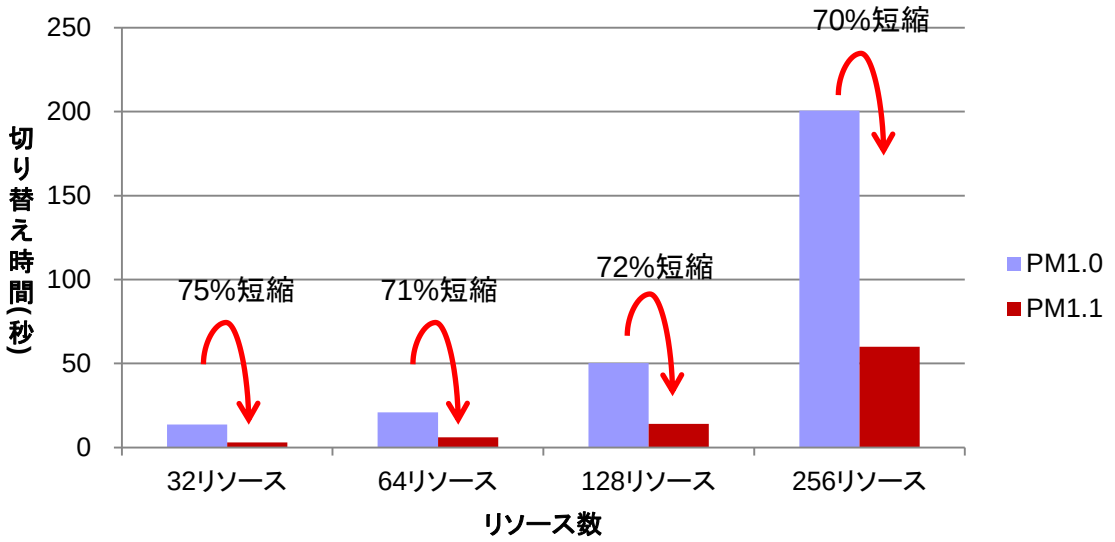


起動時間(秒)	32リソース	64リソース	128リソース	256リソース	512リソース
PM 1.0	70	154	285	312	636
PM 1.1	28	31	42	101	580

✓ Pacemaker-1.1では、256リソースでも現実的な時間で起動
✓ Pacemaker-1.1のリソース起動時間は、1.0より6～8割程度短縮
  ✓ 512リソースでは差がほぼない
    ✓ throttle機能の影響

Pacemaker-1.1の性能：リソース数とスイッチオーバ時間

■ スイッチオーバ時間：スイッチオーバ開始からスイッチオーバ完了までの時間を計測
  ✓ １＋１構成



切り替え時間(秒)	32リソース	64リソース	128リソース	256リソース
PM1.0	14	21	50	201
PM1.1	3	6	14	60

✓ Pacemaker-1.1では、256リソースでも1分でスイッチオーバ完了
✓ Pacemaker-1.0と比較して、約7割ほどの短縮

以上、性能向上のお話しでした。

次から、利便性向上に大きく貢献する(と思われる)新機能についてお話しします！

新機能　その1
～kdump連携～

Pacemaker-1.1の新機能：kdump連携機能

■ kdump連携機能

- ✓ 故障ノードでkdump実行中の場合、STONITH(※)を実行したものとみなしてF.O処理を行う機能
- ✓ これにより、故障ノードでkdumpを取得しつつ、速やかなサービス継続が可能

STONITHによりkdump処理が失敗する課題を解決！

■ Pacemaker-1.0では・・・

- ✓ 故障ノードでkdump実行中でも、容赦なくSTONITHされる
  - ✓ F.Oは速やかに完了するが、kdumpは取得失敗
- ✓ kdumpを取得するために、kdumpの完了までSTONITHおよびF.Oを遅延させる
  - ✓ stonith-helperプラグインのstandby_wait_timeを十分長く設定する(=サービス停止時間が延伸)
    - ✓ STONITH発動時、kdump処理の実行有無に関わらず、standby_wait_time分、F.Oは遅延する
  - ✓ それでも確実ではない
    - ✓ サービス停止した上にkdumpも失敗するという目も当てられない状態・・・

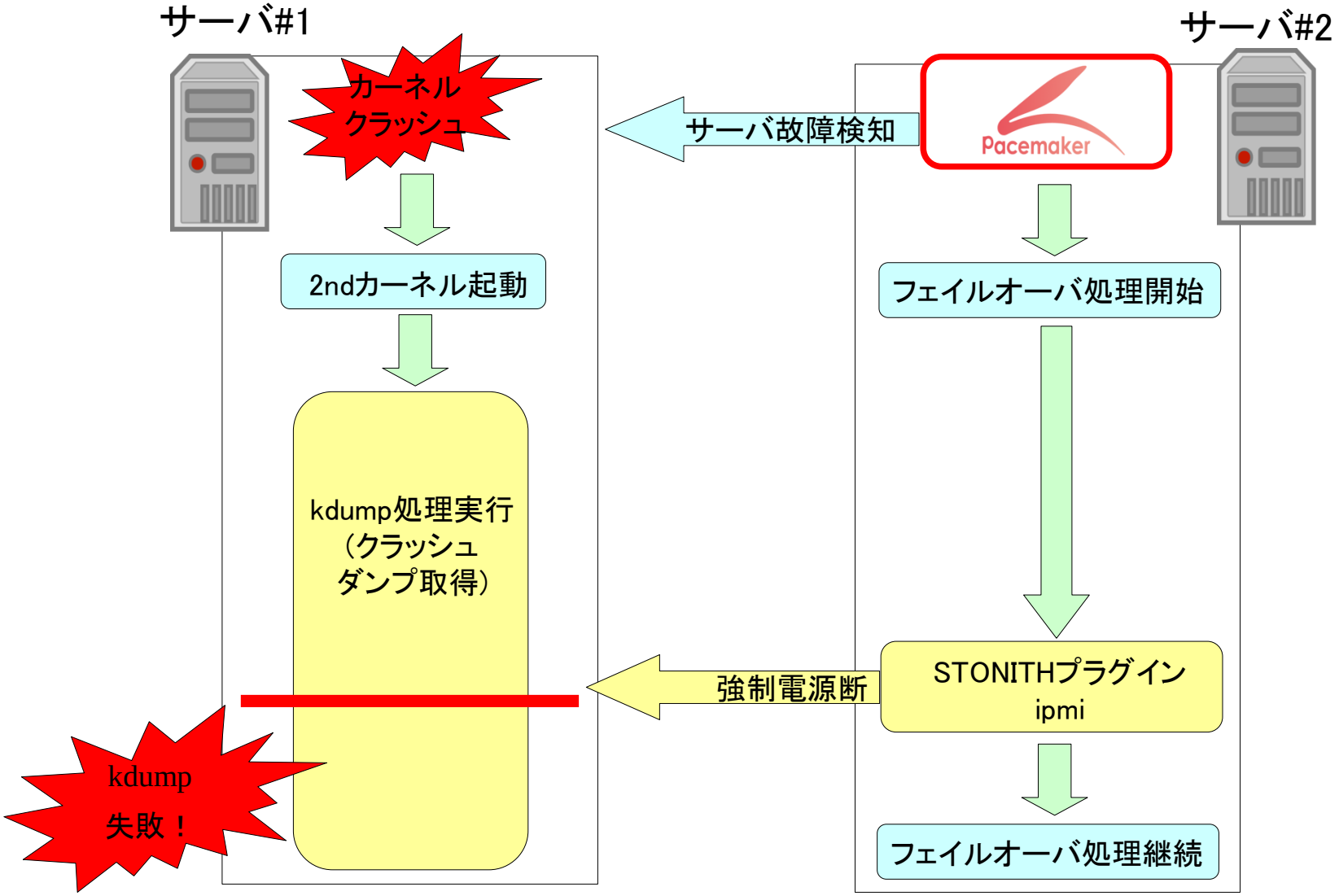
kdumpは失敗した！なぜだ！？

STONITHされたからさ・・・

(※)STONITHとは
対向ノードの状態が分からなくなった(スプリットブレイン)、リソース停止処理の失敗等、クラスタにとって致命的な状態が発生した場合に、安全のために対向ノードを強制電源断すること

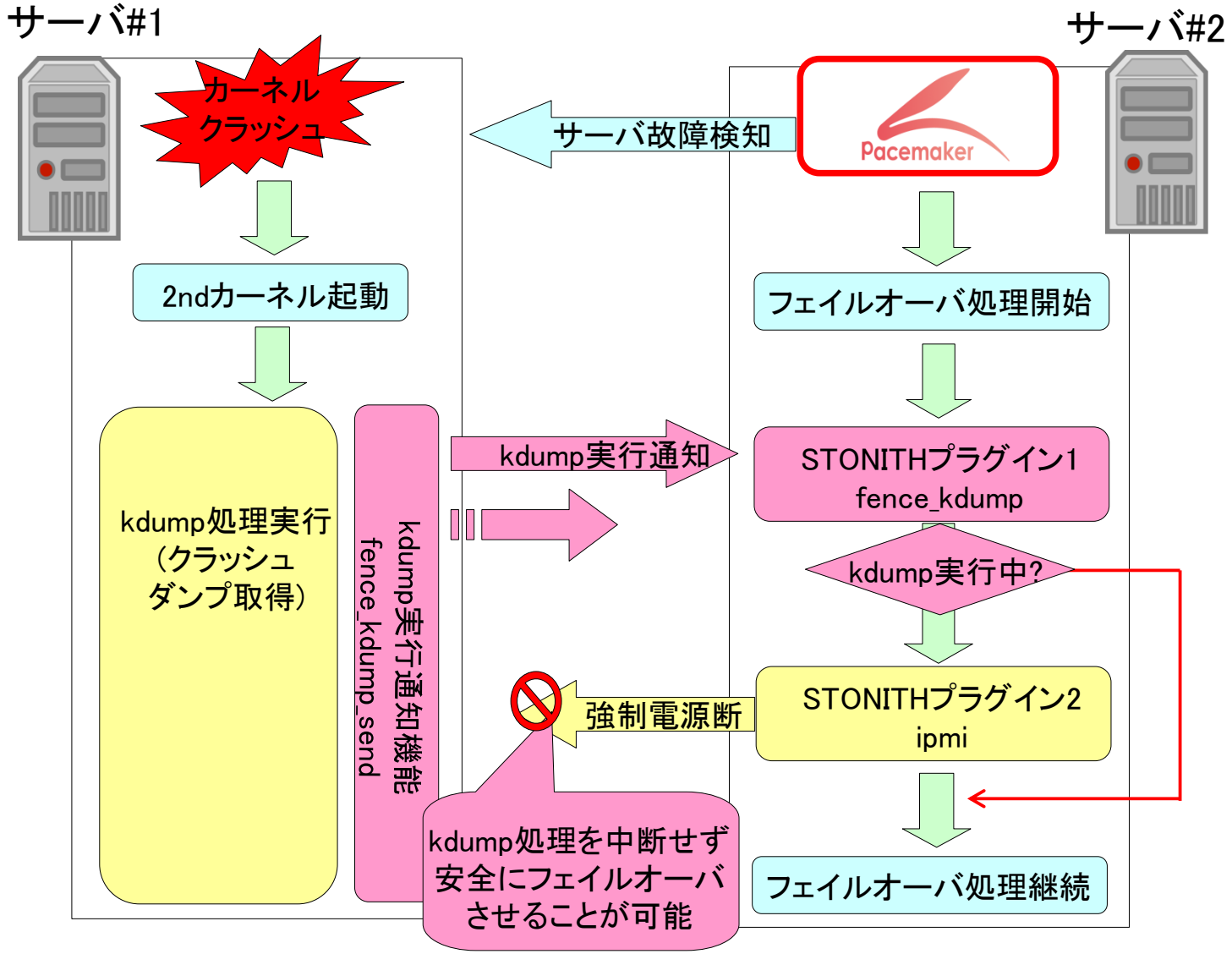
Pacemaker-1.1の新機能：kdump連携機能

✓Pacemaker-1.0の場合



Pacemaker-1.1の新機能：kdump連携機能

✓ Pacemaker-1.1の場合



使い方

- 前提条件
  - kdump機能が有効になっていること
  - セカンドカーネルと対向ノード間で通信が可能であること
    - ifconfigコマンドの先頭インタフェースと対向ノードが通信可能なこと

- インストール
  - 各ノードにfence-agentsを追加インストール
    - fence-agentsはリポジトリパッケージとOSメディアの両方に含まれるため注意

```
# yum –y install fence-agents
```

  - fence_kdump_sendの配置先を確認
    - /usr/sbin配下にない場合はシンボリックリンクを作成

```
# ln –s /usr/libexec/fence_kdump_send /usr/sbin
```

使い方

- セカンドカーネルへの組み込み(kexec-tools-2.0.0-280未満)
  - ✓ セカンドカーネルからfence_kdump_sendを起動させる設定を行う
  - ✓ /etc/cluster/cluster.confを編集 or 新規作成
    - ✓ Kexec-tools-2.0.0-280以降は/etc/kdump.confを編集

```
<cluster name="mycluster" config_version="1">
<clusternodes>
  <clusternode name="vm01" nodeid="1"/>
  <clusternode name="vm02" nodeid="2"/>
</clusternodes>
<fencedevices>
    <fencedevice name="kdump" agent="fence_kdump"/>
</fencedevices>
</cluster>
```

  - ✓ <clusternode name>には、セカンドカーネルが利用するインタフェースと通信可能なホスト名(またはIPアドレス)を指定
    - ✓ ホスト名の場合は名前解決できること
  - ✓ <fencedevice>のagentにはfence_kdumpを指定
    - ✓ fence_kdump_sendでないことに注意
    - ✓ この設定により、initrdの再作成が行われる

使い方

- セカンドカーネルへの組み込み(続き)
  - ✓ kdumpサービスの再起動
    - ✓ initrdの再作成
      - ✓ kdumpサービスの起動時に、initrdとcluster.confを比較
      - ✓ initrdよりcluster.confの方が新しい場合に、initrdを再作成する

```
# service kdump restart
```

  - ✓ initrdにfence_kdump_sendが組み込まれ、セカンドカーネル起動時にfence_kdump_sendが実行される
    - ✓ initrdにfence_kdump_sendが組み込まれていることを確認

```
# lsinitrd /boot/initrd-2.6.32-431.el6.x86_64kdump.img  | grep
fence_kdump_send
-rwxr-xr-x   1 root     root        10896 Feb  5 18:30 usr/sbin/fence_kdump_send
```

使い方

- Pacemakerリソース設定

```
primitive prmFenceKdump1 stonith:fence_kdump ¥
        params pcmk_reboot_retries=1 pcmk_reboot_timeout=60s
                pcmk_reboot_action=off pcmk_monitor_action=metadata
                pcmk_host_list=vm01 ¥
        op start interval=0s timeout=60s ¥
        op stop interval=0s timeout=60s
(中略)

fencing_topology ¥
        vm01: prmStonithHelper1 prmFenceKdump1 prmIpmi1¥
        vm02: prmStonithHelper2 prmFenceKdump2 prmIpmi2
```

  - ✓ pcmk_monitor_action=metadataは必須
    - ✓ op monitorの実装がないため
  - ✓ fencing_topology(※1)はstonith-helperと実プラグイン(ipmi等)の間に指定

(※1)ノードに対して複数のSTONITHリソースを利用する場合に、**そのノードに対して実行するSTONITHリソースの実行順序**を定義する。
また、実行順序はノード毎に定義する。

新機能　その2
～Pacemaker Remote～

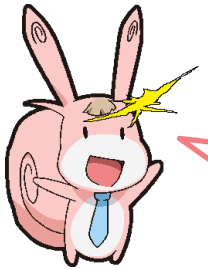
Pacemaker-1.1の新機能：Pacemaker Remote

- Pacemaker Remote
  - ✓ 仮想マシン、コンテナ、物理サーバ内のサービスを遠隔監視する機能

従来の監視方式の課題を解決！

・仮想マシン上のサービスを監視/管理できない
・ノード毎に複雑なPacemakerのインストール・設定作業が必要

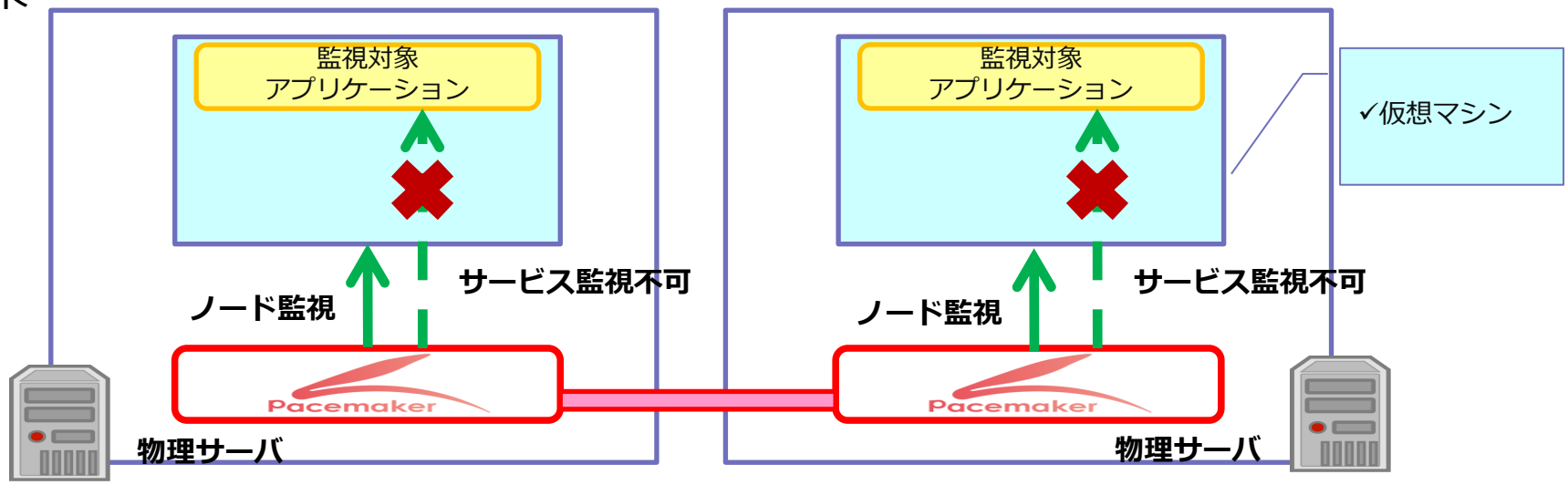
- ✓ Pacemaker Remoteを導入すると・・・
- ✓ 仮想マシン自体の監視だけでなく、仮想マシン上のサービスも監視可能
  - ✓ 物理サーバも監視可能
- ✓ 監視対象ノードへのPacemaker導入なしにノード監視・サービス監視が可能
- ✓ より大規模な構成に対応可能



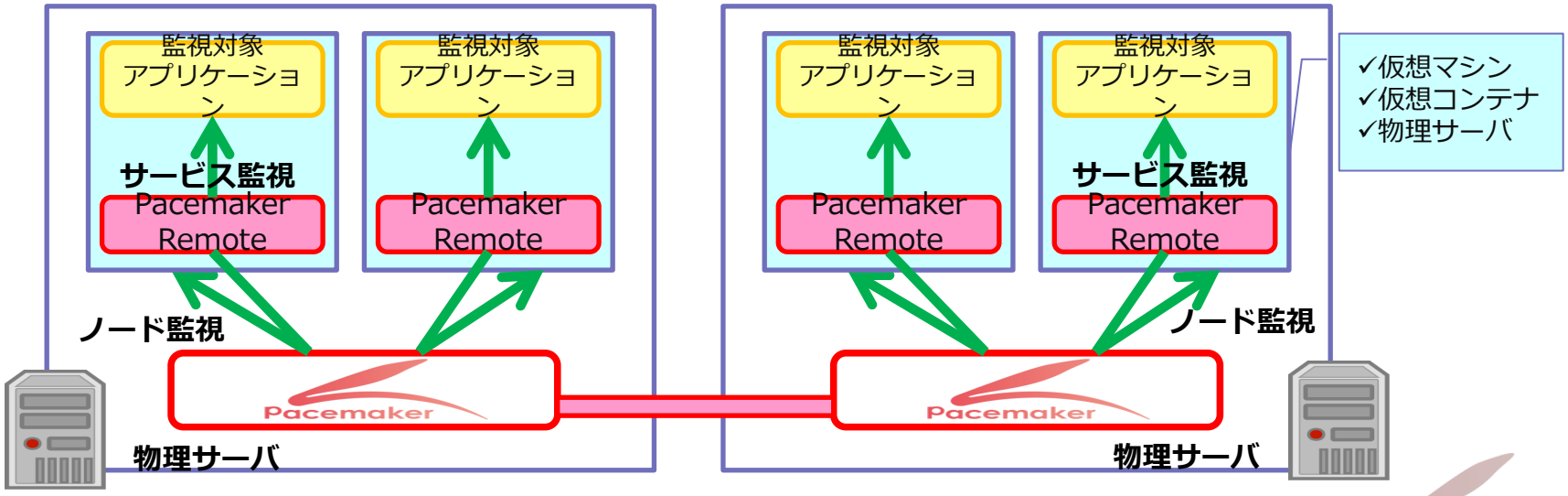
見えるぞ！私にもサービスが見える！

Pacemaker-1.1の新機能：Pacemaker Remote

■ 従来



■ Pacemaker Remote



使い方

■ 前提条件

- ✓ [ホストノード] – [リモートノード]間の通信が可能なこと
  - ✓ IPアドレス/ホスト名、ポート

■ インストール

- ✓ リモートノードに「pacemaker-remote」、「resource-agents」をインストール
  - ✓ Pacemaker-1.1.12 リポジトリパッケージに同梱
  - ✓ Pacemaker本体のインストールは不要
  - ✓ 依存関係解消のため、OSメディアおよびyumリポジトリを準備

【リモートノード】
```
# yum –y install pacemaker-remote resource-agents
```

使い方

■ 認証ファイルの作成

- ✓ ホストノードで認証ファイルを作成
  - ✓ /etc/pacemaker配下に作成
  - ✓ 作成後リモートノードにも配布

```
【ホストノード】
# mkdir /etc/pacemaker
# dd if=/dev/urandom of=/etc/pacemaker/authkey bs=4096 count=1
# scp –pr /etc/pacemaker REMOTE_NODE:/etc
```

■ Pacemaker Remoteの起動

- ✓ リモートノードでPacemaker Remoteを起動する

```
【リモートノード】
# service pacemaker_remote start
Starting Pacemaker Remote Agent:                    [  OK  ]
```

使い方

■ Pacemakerリソースの設定

- ✓ ホストノードのPacemakerに、リモードノードを通知する
  - ✓ remote RA (VirtualDomain RA(※1)を利用しない場合) or
  - ✓ VirtualDomain RAのmeta remote-nodeオプション
    - ✓ IPアドレスまたはノード名で通知

```
primitive vm02-remote ocf:pacemaker:remote ¥
  params server="vm02" ¥
  op monitor interval=3 timeout=15

primitive Host-rsc1 Dummy ¥
  op start interval=0s timeout=60s ¥
  op monitor interval=30s timeout=60s ¥
  op stop interval=0s timeout=60s

primitive Remote-rsc1 Dummy ¥
  op start interval=0s timeout=60s ¥
  op monitor interval=30s timeout=60s ¥
  op stop interval=0s timeout=60s

location loc1 Remote-rsc1 ¥
 rule 200: #uname eq vm02-remote
location loc2 Host-rsc1 ¥
 rule 200: #uname eq vm01
```

remote RAでvm02がリモートノードであることを通知

監視対象リソースは通常通り定義

定義したリソースは配置制約によって、監視対象ノードに配置
リモートノード or ホストノード？

(※1) libvirt準拠の仮想マシンを制御するRA

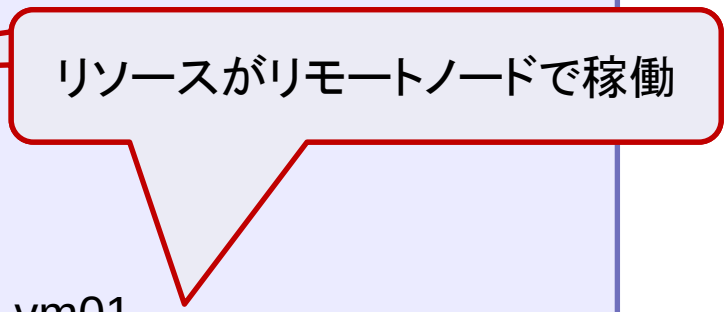
使い方

■ ホストノードでPacemaker起動

```
【ホストノード】
# initctl start pacemaker.combined
# crm configure load update remote.crm
```

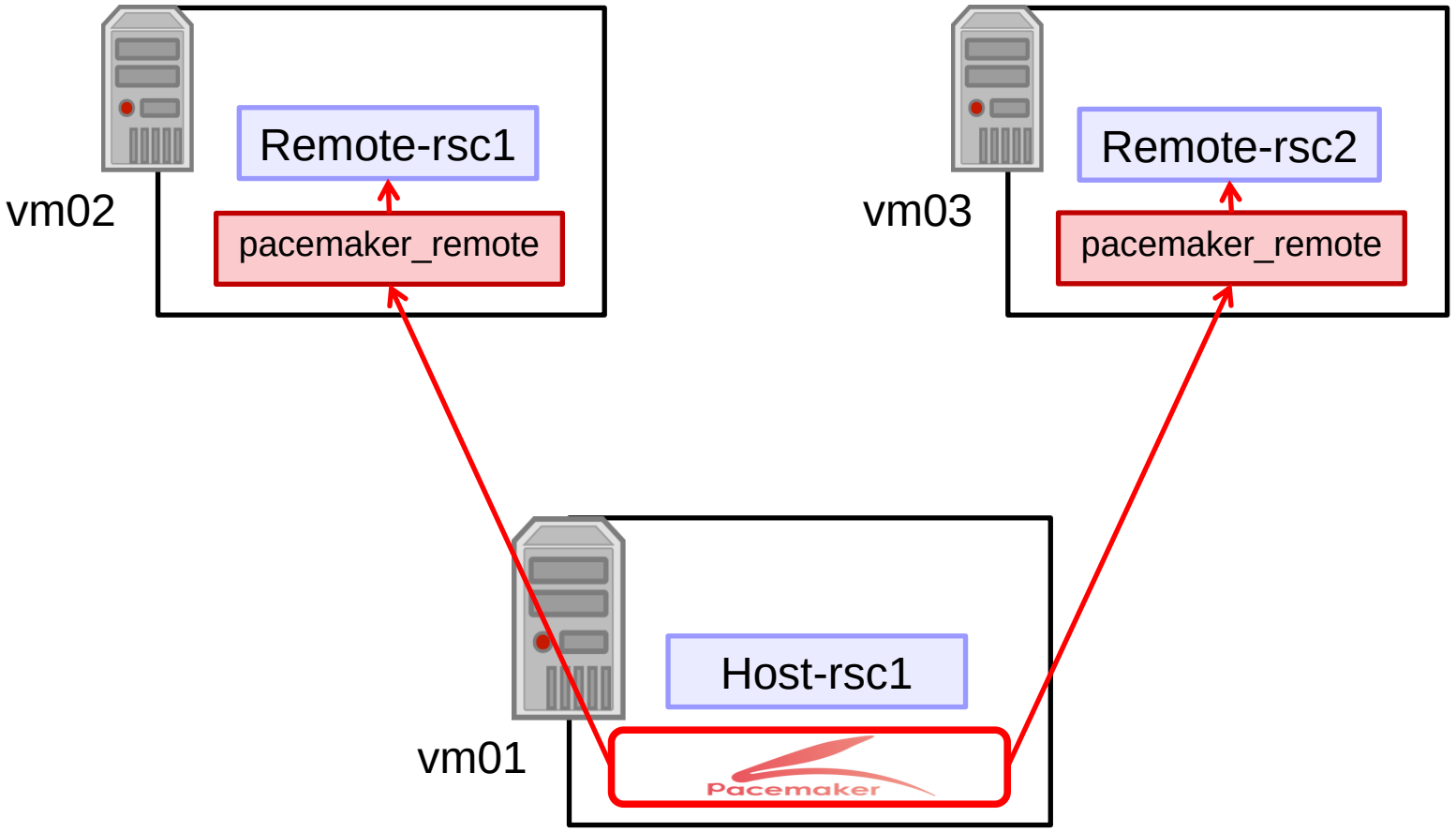
■ crm_mon

```
Online: [ vm01 ]
RemoteOnline: [ vm02-remote ]

Full list of resources:

Host-rsc      (ocf::heartbeat:Dummy): Started vm01
Remote-rsc1    (ocf::heartbeat:Dummy): Started vm02-remote
vm02-remote     (ocf::pacemaker:remote):       Started vm01

Migration summary:
* Node vm01:
* Node vm02-remote:
```



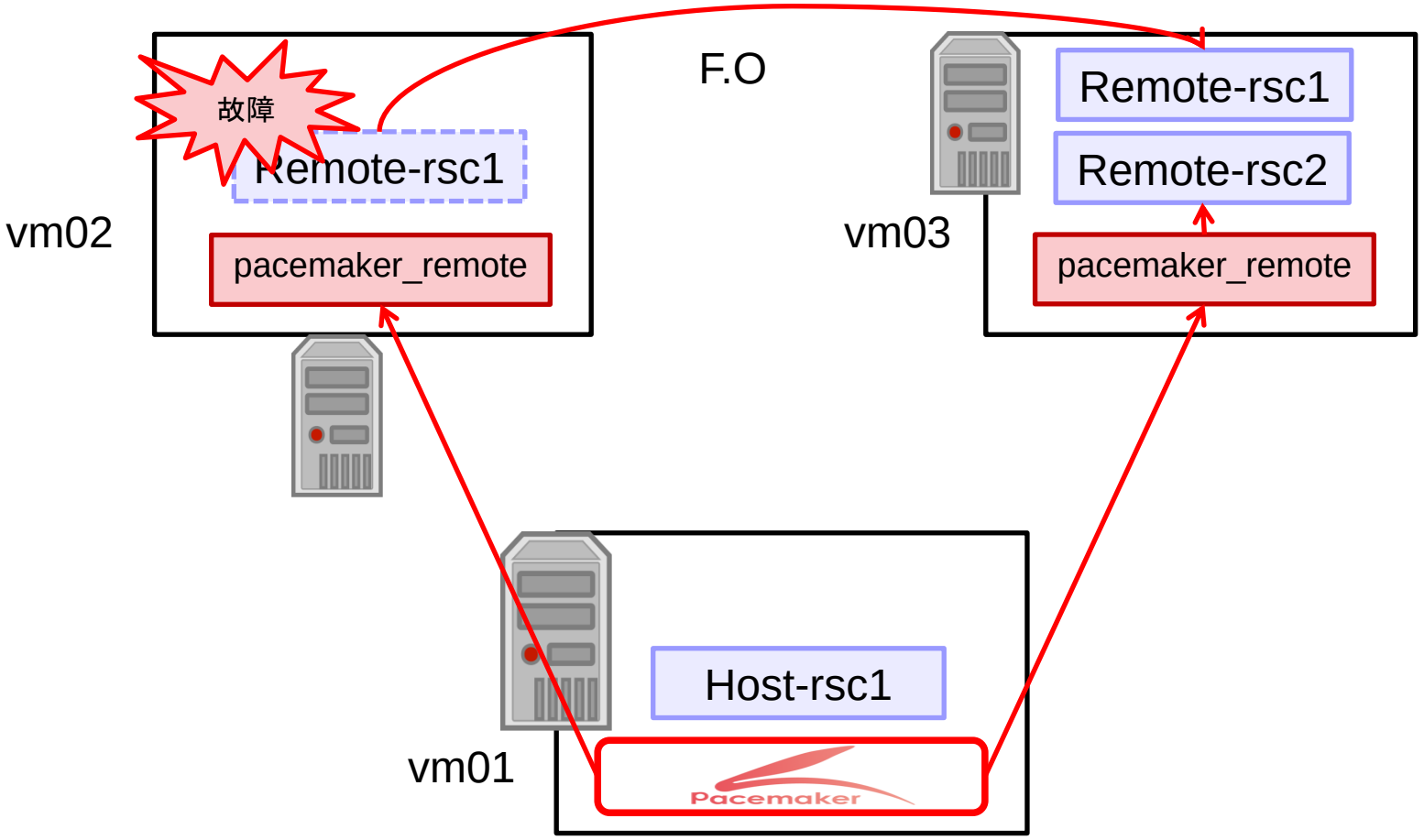
デモ

- Pacemaker Remoteのデモ
  - ✓ 仮想マシン上のリモートノードでリソースが稼働していることを確認
  - ✓ リモートノード上のリソースがフェイルオーバすることを確認



デモ

- Pacemaker Remoteのデモ
  - ✓ 仮想マシン上のリモートノードでリソースが稼働していることを確認
  - ✓ リモートノード上のリソースがフェイルオーバすることを確認



新機能　その3
～リソース配置戦略機能～

Pacemaker-1.1の新機能：リソース配置戦略機能

■ リソース配置戦略機能

- ✓ ノードとリソースに「容量」という概念を導入し、ノードの空き容量、リソースの使用容量に応じてリソースの配置先を決定する機能
  - ✓ 「リソースの使用容量 > ノードの空き容量」となったノードでは当該リソース配置不可
- ✓ 従来のリソース配置より柔軟な配置が可能

従来のリソース配置方式の課題を解決！

- リソース配置先の偏り(※1)
- マシン性能以上のリソース稼働による性能劣化

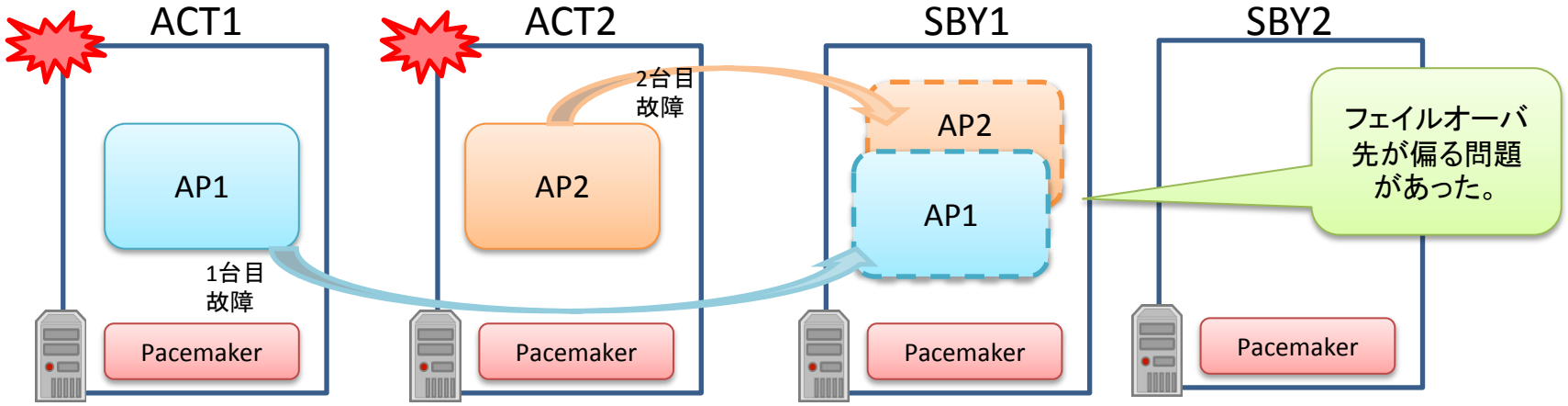
- ✓ 注意点
  - ✓ 「リソースの容量」はprimitiveリソースのみ設定可能
    - ✓ group、clone、msリソースには設定不可
    - ✓ ただし、groupの場合は先頭のprimitiveリソースに設定すればOK

primitive、clone、groupについては下記参照
http://linux-ha.sourceforge.jp/wp/wp-content/uploads/OSC2013TokyoFall.pdf

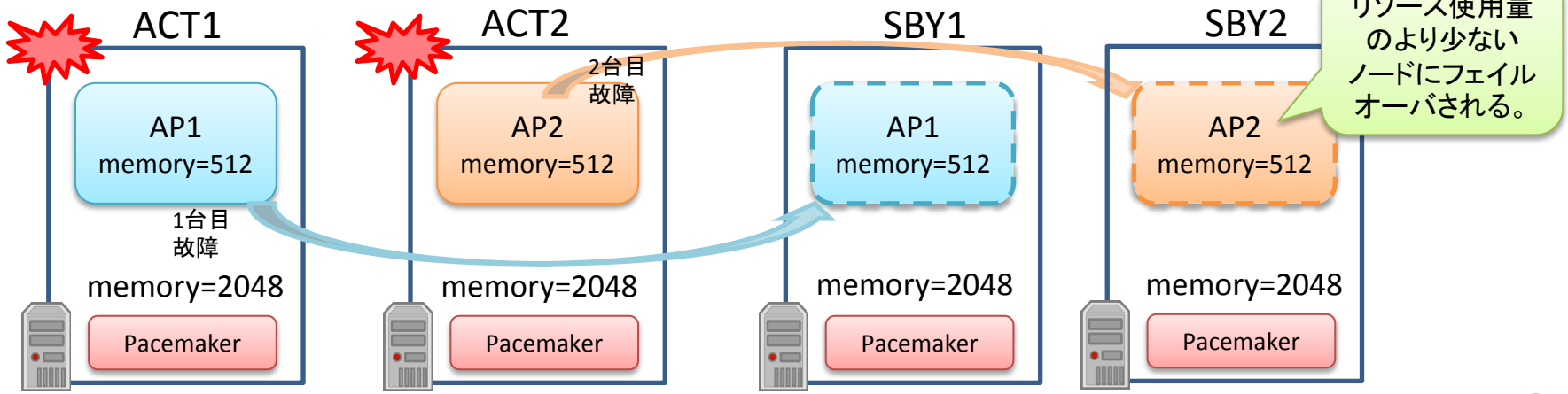
(※1) リソース名や故障順序などに依存します

Pacemaker-1.1の新機能：リソース配置戦略機能

1. フェイルオーバ先ノードが偏る
   - ✓ 従来の動作イメージ

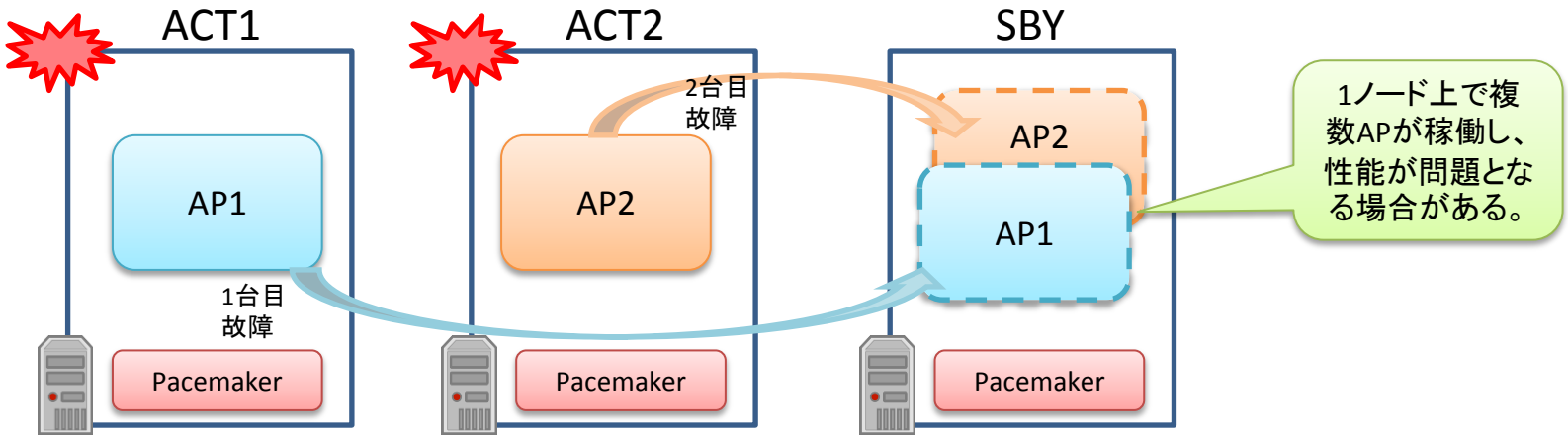


   - ✓ リソース配置戦略機能の動作イメージ

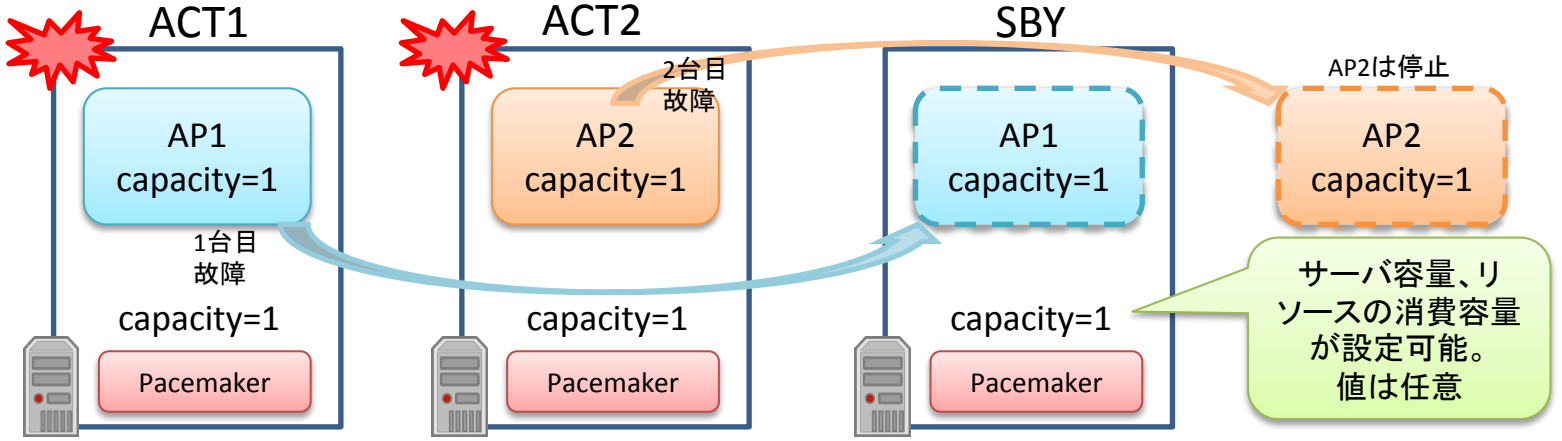


Pacemaker-1.1の新機能：リソース配置戦略機能

2. 複数APが1ノード上で稼働
   - ✓ 従来の動作イメージ



   - ✓ リソース配置戦略機能の動作イメージ



Pacemaker

使い方

- crmファイルに下記を定義する
  1. ノードの容量
  2. 配置戦略
  3. リソースの容量

- ノードの容量を定義

```
node XXX utilization capacity="1"
node YYY utilization capacity="1"
```

任意の名前でOK
ただし、リソース容量も同じ名前で定義すること

- 配置戦略を定義

```
property ・・・
        placement-strategy="balanced"
```

- リソース容量を定義

```
primitive rscDummy ocf:heartbeat:Dummy ¥
        utilization capacity="1" ¥
```

ノードの容量と同一の名前で定義する

Pacemaker

Pacemaker-1.1の情報が少ないので、
お勧めの設定、MLで話題になったことを
ピックアップしてご紹介します！

TIPS：Pacemakerのログをまとめよう

- Pacemakerはコンポーネント群なので、ログ出力先がバラバラ
  - ✓ messages、corosync.log、pacemaker.log
  - ✓ 大量のログを見比べて突き合わせるのは大変・・・

rsyslogでログをまとめよう！

(例) ログを/var/log/ha-logにまとめて出力する

- ✓ ファイル名、facility、priority等は適宜修正してください

- /etc/corosync/corosync.conf

```
logging {
        syslog_facility: local1         # corosyncのログ facilityをloca1に設定
        debug: off
}
```

- /etc/sysconfig/pacemaker

```
export PCMK_logfile=none          # pacemaker.logは出力しない
export PCMK_logfacility=local1    #  pacemakerのログ facilityをlocal1に設定
export PCMK_logpriority=info      #  pacemakerのログレベルをinfo
export HA_LOGFACILITY=local1 # stonithdのメッセージをlocal1に出力する設定
```

- /etc/rsyslog.conf

```
*.info;mail.none;authpriv.none;cron.none;local1.none          /var/log/messages
local1.info                                                    /var/log/ha-log
```

TIPS：tokenの推奨値ってありますか？

- Token(corosync.conf)とは・・・
  - ✓ corosync間の通信(token)の受信タイムアウト
  - ✓ ノードの故障検知時間に影響

故障検知時間 = token + consensus

  - ✓ consensusは新たなリングを形成するまでのタイムアウト時間
  - ✓ デフォルトはtoken * 1.2

- 1+1構成の場合の推奨値
  - ✓ デフォルトの1000msです
    - ✓ 高負荷時などにタイムアウトが発生した場合は、要調整

- N+M構成の場合の推奨値
  - ✓ Linux-HA Japanでの実績のある値はありません
    - ✓ tokenはリング状に巡回するため、ノード数が増えるほど受信までの必要時間が伸びる
    - ✓ 必要に応じて値を調整してください
    - ✓ nodelistを定義している場合は、1ノード増加するごとに自動的に+650msされる
      - ✓ 詳しくはman corosync.confの「token_coefficient」項参照

Linux-HA Japan URL

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

http://sourceforge.jp/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。)

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

ご清聴ありがとうございました。

Linux-HA Japan

検索

HA Cluster Summit 2015 参加報告

Linux-HA Japanのメンバ3名が、チェコ　ブルーノで世界中のLinux-HA界隈の方々と熱い議論をしてきました！

HA Cluster Summit 2015

- 概要
  - Pacemaker, corosync などの HAクラスタに関連する開発者が一同に会するF2Fミーティング
    - http://plan.alteeve.ca/index.php/Main_Page
  - 開催は不定期。前回は 2011年開催(プラハ)
  - Red Hat, SUSEのメンバが持ち回りで主催
- 日時
  - 2015年2月4日(水)～2015年2月5日(木)
- 場所
  - Red Hat 社チェコオフィス(Brno)
- 参加者
  - 合計26名程度
  - 大部分は Red Hat 開発者
  - SUSE(4名), LINBIT(3名), SAP(1名)
  - その他個人会社など
  - 日本からは3名

Pacemaker

参加者

Pacemaker

トピック1: 今後のHAコミュニティ全体の方針について

- 現状の課題
  - HAクラスタに関するWebサイト、ML等が多数分散しており、情報源や問い合わせ先がわかりづらい。
  - 過去の経緯(Red Hatクラスタ機能とのマージ等)により開発者・コンポーネントが細分化されているため。

- 主な議論内容
  - 初めて参加する人への入口となるポータルのようなものが欲しい。
  - 理想は OpenStack コミュニティのように全体の統括とサブプロジェクトに分かれているような体制が望ましいが、現時点ではそこまでのリソースはない。
  - まずは全体の入口となる名前を決定し、既存の情報源へのリンクを設けるところから始めたい。

- 決定事項
  - clusterlabs の名前を全体の名前とする。
    - wiki (clusterlabs.org), github, メーリングリスト、IRC を全て clusterlabs の名前で統一
  - 次回開催予定
    - 時期: 2016年夏頃目途
    - 場所: プラハ、SUSEオフィスが候補(詳細未定)

トピック2: HAクラスタ新機能について

- 最近開発された機能、および今後に向けた新機能の要望と議論。
  - UIのエラーチェック改善、ライブマイグレーションの動作改善など
    比較的細かい改善についての議論がほとんど

- docker リソースエージェントの紹介
  - dockerコンテナを管理するためのRA。コミュニティ最新版で公開済み。
    - ※ Linux-HA Japanリポジトリパッケージ版にはまだ含まれていません(1.1.12-1.1時点) 。
  - コンテナ内サービスの監視と構築の容易化が可能
  - 例:
    - Webサーバのグループリソース(RA、ミドルウェア)をコンテナ内に予め構築
    - → docker RAの引数として IP、ポート番号等を指定してサービスインスタンスを作成可能
    - → サービス監視は pacemaker_remoted経由