

PG-REX 発展編

～マルチシンクレプリケーションで3重化～

2018年 2月 24日
OSC2018 Tokyo/Spring

Linux-HA Japan
竹下 雄大



本日の内容

- 最新版Pacemaker-1.1.17-1.1のご紹介
- PG-REX マルチシンクレプリケーションで3重化
- デモ
- 【参考】マルチシンクレプリケーション用設定について

最新版Pacemaker-1.1.17-1.1の ご紹介

Pacemakerはオープンソースの
HAクラスタソフトです

High **A**vailability = 高可用性

つまり

サービス継続性

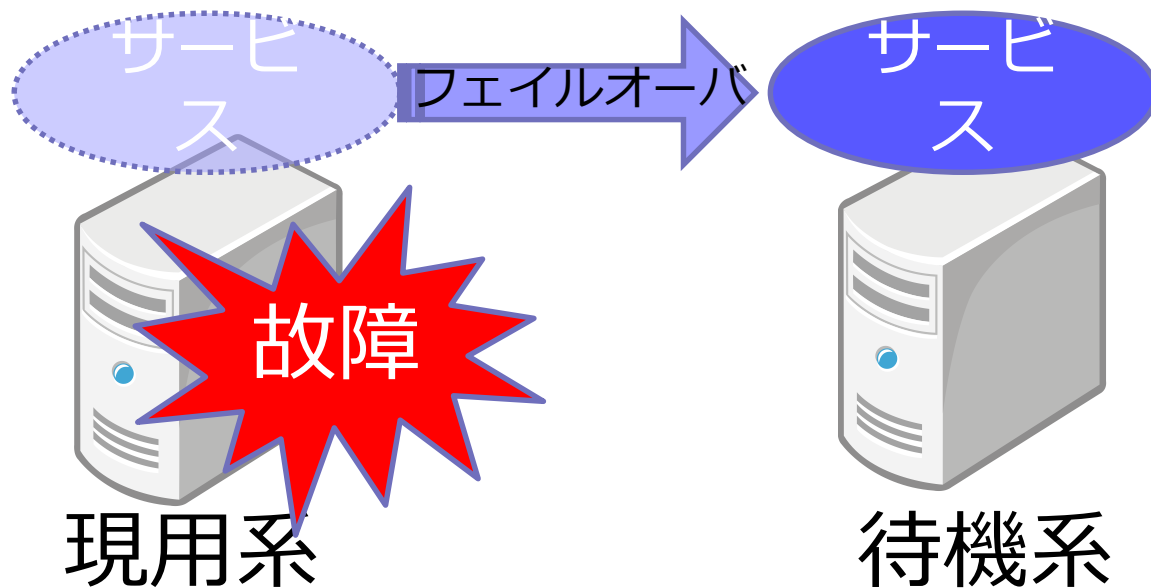
一台のコンピュータでは得られない高い信頼性を得るために、複数のコンピュータを結合(クラスタ化)し、ひとまとまりとする...

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運用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「**フェイルオーバー**」と言います

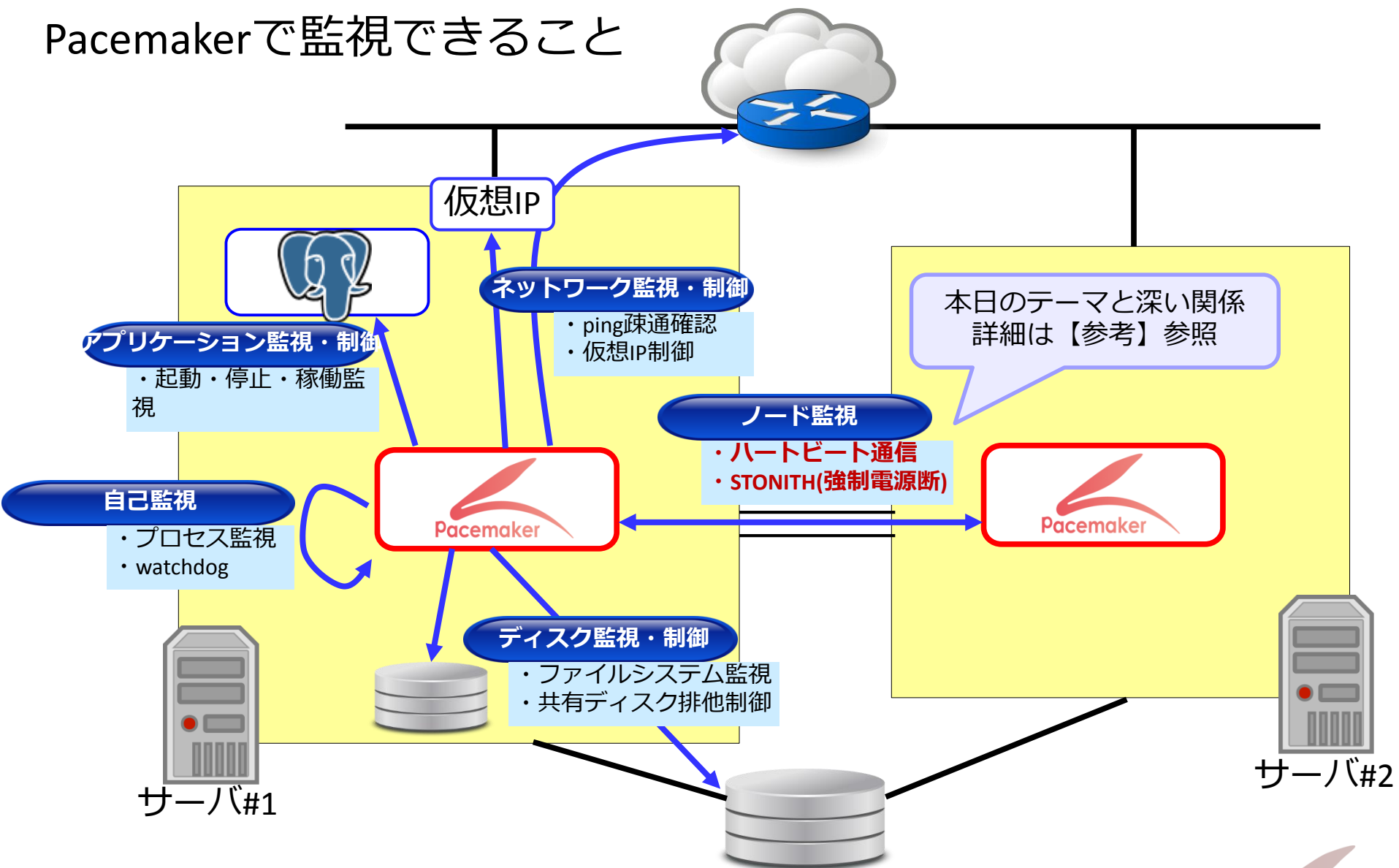


Pacemakerってなに？

 Pacemaker 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

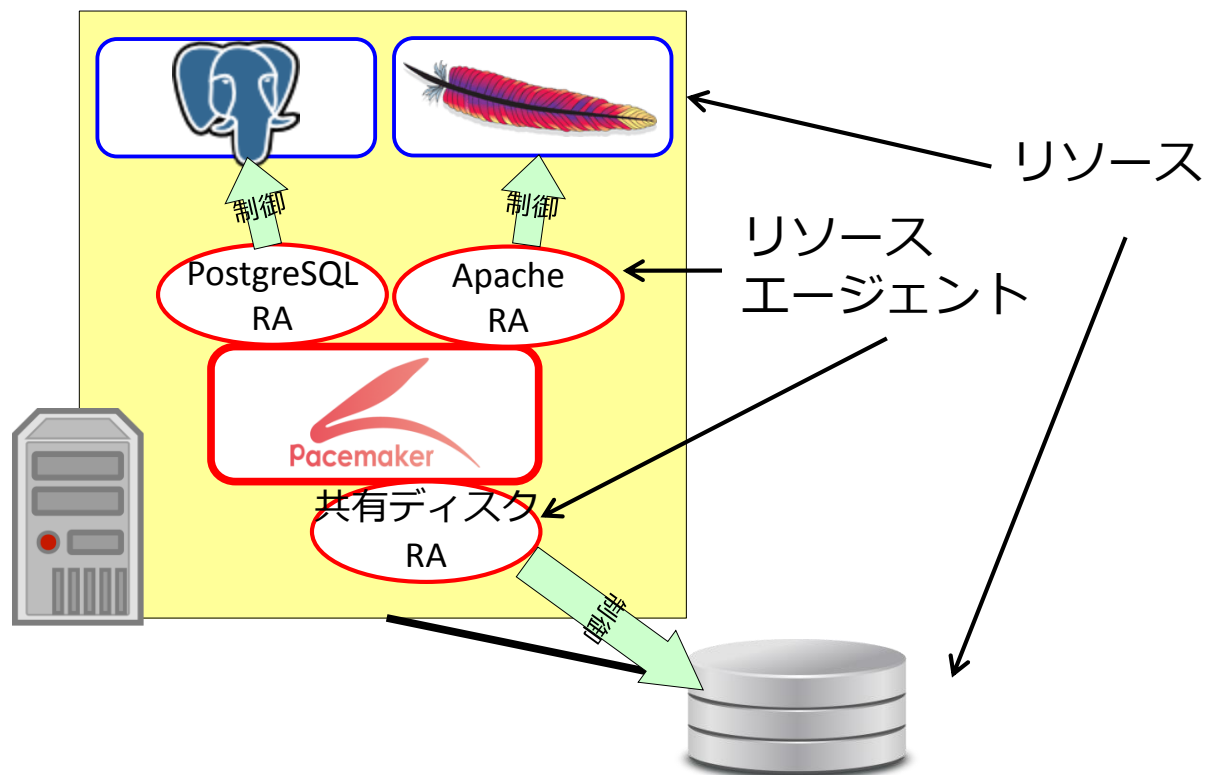
Pacemakerってなに？

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





Pacemaker-1.1.17-1.1

2018.2.14 リリース！

Pacemaker-1.1.17-1.1の主な変更点

- PostgreSQL 10対応
- bundle リソースタイプの追加
 - Docker コンテナ管理専用のリソースタイプ

- PostgreSQL 10 が 2017/10/5 にリリース
 - 約 7年ぶりの大型アップデート
 - 一部、過去バージョンとの非互換あり
 - Pacemaker-1.1.16-1.1 + PostgreSQL 10 ではPG-REX構築不可
- Pacemaker 1.1.17-1.1 でPostgreSQL 10対応を実施
 - postgresql RAで非互換を吸収
 - Pacemaker 1.1.17-1.1 + PostgreSQL 10でPG-REXが利用できるよう改善

PostgreSQL 10 使えます！

bundle リソースタイプの追加

- Dockerコンテナを管理するための新たなリソースタイプ
 - Pacemaker-1.1.16まではDocker RAで管理
 - Docker RAによるコンテナ管理については以下を参照
 - <http://linux-ha.osdn.jp/wp/archives/4601>

- Docker RA との比較

- スケーラビリティを強化
 - コミュニティでは 1500 コンテナまで報告有
- コンテナ内リソースのヘルスチェックを強化
 - RAによるサービスレベルの複雑な監視が可能
 - Docker RAでは任意のワンライナー、またはHEALTHCHECKで監視
- Pacemaker-1.1.17-1.1ではcrmshによるリソース設定不可
 - がんばってXMLで設定する
 - pcsを使う



Linux-HA Japan では Technology Preview (お試し版) の位置づけ

bundle リソースタイプの追加

- bundleでApacheを3コンテナ起動した例

Online: [pm03 pm04]
GuestOnline: [httpd-bundle-0@pm03 httpd-bundle-1@pm04 httpd-bundle-2@pm03]

Full list of resources:

Docker container set: httpd-bundle [pcmkttest:http] (unique)

httpd-bundle-0 (192.168.0.200)	(ocf::heartbeat:apache):	Started pm03
httpd-bundle-1 (192.168.0.201)	(ocf::heartbeat:apache):	Started pm04
httpd-bundle-2 (192.168.0.202)	(ocf::heartbeat:apache):	Started pm03

Node Attributes:

- * Node httpd-bundle-0@pm03:
- * Node httpd-bundle-1@pm04:
- * Node httpd-bundle-2@pm03:
- * Node pm03:
- * Node pm04:

Migration Summary:

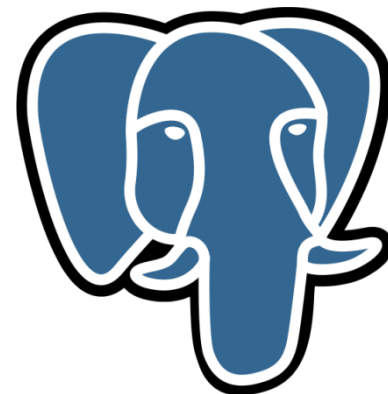
- * Node pm03:
- * Node pm04:
- * Node httpd-bundle-1@pm04:
- * Node httpd-bundle-0@pm03:
- * Node httpd-bundle-2@pm03:

Apacheがコンテナ内でRAによって
制御される

コンテナ



PG-REX マルチシンクレプリケーションで3重化



PG-REXってなに？

PostgreSQLレプリケーション機能 + Pacemakerの構成を

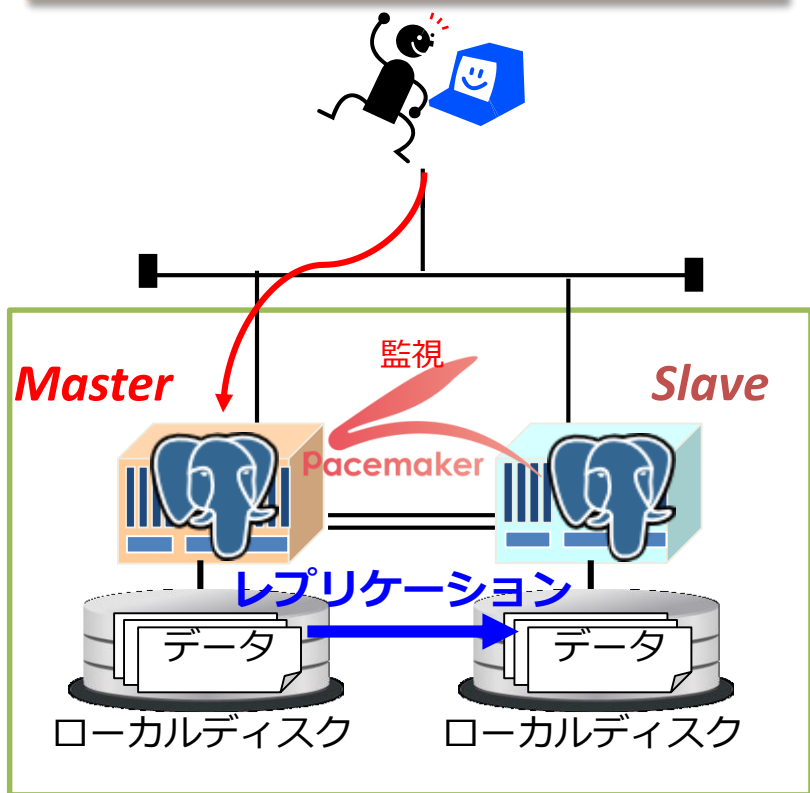


「PG-REX」と呼ぶ

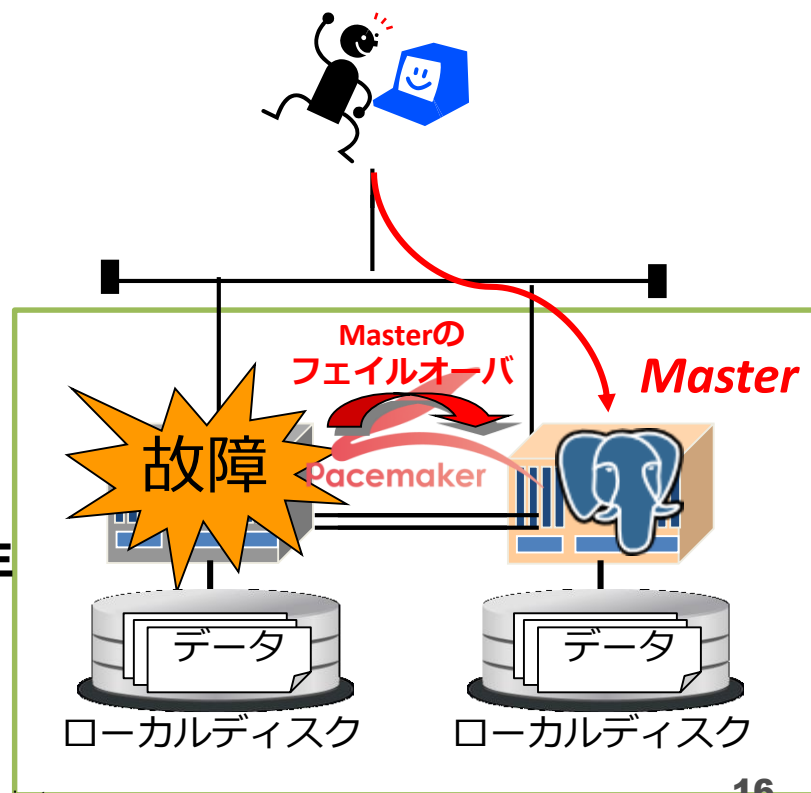
PostgreSQLのストリーミングレプリケーション機能を用いてデータを常に両系にコピー



故障をPacemakerが監視・検知
SlaveをMasterに昇格させることで自動的にサービスを継続



故障発生



共有ディスク構成 vs PG-REX



	共有ディスク	PG-REX
ハードウェア費用	共有ディスク(相当のもの)必須	 普通のHDDでよい
運用のしやすさ	 データは1箇所のみ	2箇所のデータの整合性を考慮
データの安全性	最新データは 共有ディスク上のみ	 最新データは2箇所に分散
サービス継続性	 フェイルオーバ時に リカバリに時間を要する	 フェイルオーバは早い が、Slave故障がサービスに影響
DB性能	 レプリケーションの オーバヘッド(※)なし (※)ディスクI/O、ネットワーク	レプリケーションの オーバヘッドあり
負荷分散	構成上不可能	 ReadOnlyクエリを Slaveで処理可能
実績	 多数あり	 近年増加

それぞれ一長一短。サービスの要件に応じて選択すること。

OSCで普及活動やっています！

- Pacemaker + PostgreSQLレプリケーションで共有ディスクレス高信頼クラスタの構築
 - OSC 2013 Tokyo/Spring
 - <http://linux-ha.osdn.jp/wp/archives/3589>
- Pacemaker + PostgreSQLレプリケーション構成(PG-REX)のフェイルオーバー高速化
 - OSC 2014 Hokkaido / OSC 2014 Kansai@Kyoto
 - <http://linux-ha.osdn.jp/wp/archives/4010>
- PG-REXで学ぶPacemaker運用の実例
 - OSC 2015 Fukuoka
 - <http://linux-ha.osdn.jp/wp/archives/4137>
- Pacemaker + PostgreSQLレプリケーション構成(PG-REX)の運用性向上 ～スロットの覚醒～
 - OSC 2016 Tokyo/Spring
 - <http://linux-ha.osdn.jp/wp/archives/4421>
- 試して覚えるPacemaker入門『PG-REX構築』
 - OSC 2017 kyoto
 - <http://linux-ha.osdn.jp/wp/archives/4627>
- 試して覚えるPacemaker入門『PG-REX運用』
 - OSC 2018 Osaka
 - <http://linux-ha.osdn.jp/wp/archives/4664>

PG-REXプロジェクト

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

PG-REX  Fork

概要 ▾ ダウンロード ソースコード ▾ チケット ▾ 文書 ▾ コミュニケーション ▾ ニュース

KDE Plasma 5を搭載した「Mageia 6」が登場 [Magazine]

プロジェクトの説明





PostgreSQLとPacemakerを組み合わせた高可用ソリューション


 画像一覧

ダウンロード

 Windows [pg-rex96-1.0.0-1.tar.gz](#) (日付: 2017-01-25, サイズ: 1.46 MB)

 Mac [pg-rex96-1.0.0-1.tar.gz](#) (日付: 2017-01-25, サイズ: 1.46 MB)

 Linux [pg-rex96-1.0.0-1.tar.gz](#) (日付: 2017-01-25, サイズ: 1.46 MB)

 UNIX [pg-rex96-1.0.0-1.tar.gz](#) (日付: 2017-01-25, サイズ: 1.46 MB)

最新リリース

[PG-REX9.6 1.0.2_CentOS7](#) (日付: 2017-07-21)

[pg-rex_operation_tools 1.8.1](#) (日付: 2017-07-21)

[pg-rex_operation_tools 1.8.0](#) (日付: 2017-01-25)

[PG-REX9.6 1.0.0](#) (日付: 2017-01-25)

[PG-REX9.5 1.1.1](#) (日付: 2016-09-01)

PG-REXの構築手順書や設定例、PG-REX運用補助ツールを提供しています。

本セミナーでは割愛している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！

免責

- 本日は話す内容は、将来的にボツになる可能性があります
 - 現在 パッチをコミュニティへ pull request 中
 - Rejectされないように頑張ります！
- とりあえず使ってみたいという方は以下からpgsql RAを個別にダウンロードしてください
 - https://github.com/ytakeshita/resource-agents/blob/support_multiple_synchronous_standby/heartbeat/pgsql

マルチシンクレプリケーションとは

マルチシンクレプリケーションとは

- 1つのMasterに対して、複数のSlaveを同期レプリケーションさせる機能
 - priorityベースとquorumベースの二つの方式が存在する
 - priorityベース：設定したノードの先頭N個が同期レプリケーションする
 - quorumベース：設定したノードのうち、少なくともN個が同期レプリケーションする
- PostgreSQL 9.6以上で利用可能
 - 9.6：priority ベースのみ
 - 10：priorityベース、quorumベース
- PostgreSQL 9.5までは、1つのMasterに対して、1つのSlaveのみしか同期レプリケーションできなかった
 - 2Slave目以降はAsyncまたはPotential
 - カスケードレプリケーション(PostgreSQL 9.3以降)でも、2Slave目以降は非同期

設定方法と動作 - priorityベース -

- postgresql.confのsynchronous_standby_namesで設定

- **priorityベース**

```
synchronous_standby_names = 'N (node_1, node_2, ..., node_M)'  
                           or  
synchronous_standby_names = 'FIRST N (node_1, node_2, ..., node_M)'
```

- node_1, node_2, ..., node_Mのうち、先頭Nノードがsync
 - 残りのノードはpotentialになる

```
synchronous_standby_names = '2 (node_1, node_2, node_3)'
```

sync potential

- synchronous_standby_namesに記載のないノード(node_4)はasync
 - 「FIRST」は省略可能(PostgreSQL 9.6ではFIRSTなしのsyntaxのみ)
 - $N \leq M$
 - » $N > M$ も記載はできるが、レプリケーションが完了しない

PostgreSQLの状態 - priorityベース -

Master: pm01 / Slave: pm02, pm03

synchronous_standby_names='2 (pm02, pm03)'

```
# select * from pg_stat_replication;
```

```
-[ RECORD 1 ]-----
```

Pid	1002
usesysid	16384
username	repuser

application_name | pm03

client_addr	192.168.2.140
client_hostname	
client_port	48140
backend_start	2017-09-29 09:18:07.399915+09
backend_xmin	600
state	streaming
sent_lsn	0/41000060
write_lsn	0/41000060
flush_lsn	0/41000060
replay_lsn	0/41000060
write_lag	
flush_lag	
replay_lag	
sync_priority	2

sync_state | sync

```
-[ RECORD 2 ]-----
```

pid	923
usesysid	16384
username	repuser

application_name | pm02

client_addr	192.168.2.60
client_hostname	
client_port	35100
backend_start	2017-09-29 09:15:55.836521+09
backend_xmin	600
state	streaming
sent_lsn	0/41000060
write_lsn	0/41000060
flush_lsn	0/41000060
replay_lsn	0/41000060
write_lag	
flush_lag	
replay_lag	
sync_priority	1

sync_state | sync

synchronous_standby_names='1 (pm02, pm03)'

```
# select * from pg_stat_replication;
```

```
-[ RECORD 1 ]-----
```

pid	1002
usesysid	16384
username	repuser

application_name | pm03

client_addr	192.168.2.140
client_hostname	
client_port	48140
backend_start	2017-09-29 09:18:07.399915+09
backend_xmin	600
state	streaming
sent_lsn	0/41000140
write_lsn	0/41000140
flush_lsn	0/41000140
replay_lsn	0/41000140
write_lag	
flush_lag	
replay_lag	
sync_priority	2

sync_state | potential

```
-[ RECORD 2 ]-----
```

pid	923
usesysid	16384
username	repuser

application_name | pm02

client_addr	192.168.2.60
client_hostname	
client_port	35100
backend_start	2017-09-29 09:15:55.836521+09
backend_xmin	600
state	streaming
sent_lsn	0/41000140
write_lsn	0/41000140
flush_lsn	0/41000140
replay_lsn	0/41000140
write_lag	
flush_lag	
replay_lag	
sync_priority	1

sync_state | sync

synchronous_standby_names='1 (pm02)'

```
# select * from pg_stat_replication;
```

```
-[ RECORD 1 ]-----
```

pid	1002
usesysid	16384
username	repuser

application_name | pm03

client_addr	192.168.2.140
client_hostname	
client_port	48140
backend_start	2017-09-29 09:18:07.399915+09
backend_xmin	600
state	streaming
sent_lsn	0/41000140
write_lsn	0/41000140
flush_lsn	0/41000140
replay_lsn	0/41000140
write_lag	
flush_lag	
replay_lag	
sync_priority	0

sync_state | async

```
-[ RECORD 2 ]-----
```

pid	923
usesysid	16384
username	repuser

application_name | pm02

client_addr	192.168.2.60
client_hostname	
client_port	35100
backend_start	2017-09-29 09:15:55.836521+09
backend_xmin	600
state	streaming
sent_lsn	0/41000140
write_lsn	0/41000140
flush_lsn	0/41000140
replay_lsn	0/41000140
write_lag	
flush_lag	
replay_lag	
sync_priority	1

sync_state | sync

設定方法と動作 - quorumベース -

- quorumベース

```
synchronous_standby_names = 'ANY N (node_1, node_2, ..., node_M)'
```

- node_1, node_2, ..., node_Mの**少なくともNノード**が同期レプリケーション
 - node_1, node_2, ..., node_Mの状態は全て「quorum」
 - ノードの状態(sync_status)では、どのノードが同期状態なのか(あるいは、遅れているのか)を判別できない

```
synchronous_standby_names = 'ANY 2 (node_1, node_2, node_3)
```

quorum

- synchronous_standby_namesに記載のないノード(node_4)はasync
 - 「ANY」は省略不可
 - $N \leq M$
 - » $N > M$ も記載はできるが、レプリケーションが完了しない

PostgreSQLの状態 - quorumベース -

Master: pm01 / Slave: pm02, pm03

synchronous_standby_names

= 'ANY 2 (pm02, pm03)'

```
# select * from pg_stat_replication;
```

```
-[ RECORD 1 ]-----
```

pid	1002
usesysid	16384
username	repuser
application_name	pm03
client_addr	192.168.2.140
client_hostname	
client_port	48140
backend_start	2017-09-29 09:18:07.399915+09
backend_xmin	600
state	streaming
sent_lsn	0/41000140
write_lsn	0/41000140
flush_lsn	0/41000140
replay_lsn	0/41000140
write_lag	
flush_lag	
replay_lag	
sync_priority	1

sync_state | **quorum**

```
-[ RECORD 2 ]-----
```

pid	923
usesysid	16384
username	repuser
application_name	pm02
client_addr	192.168.2.60
client_hostname	
client_port	35100
backend_start	2017-09-29 09:15:55.836521+09
backend_xmin	600
state	streaming
sent_lsn	0/41000140
write_lsn	0/41000140
flush_lsn	0/41000140
replay_lsn	0/41000140
write_lag	
flush_lag	
replay_lag	
sync_priority	1

sync_state | **quorum**

synchronous_standby_names

= 'ANY 1 (pm02, pm03)'

```
# select * from pg_stat_replication;
```

```
-[ RECORD 1 ]-----
```

pid	1002
usesysid	16384
username	repuser
application_name	pm03
client_addr	192.168.2.140
client_hostname	
client_port	48140
backend_start	2017-09-29 09:18:07.399915+09
backend_xmin	600
state	streaming
sent_lsn	0/41000140
write_lsn	0/41000140
flush_lsn	0/41000140
replay_lsn	0/41000140
write_lag	
flush_lag	
replay_lag	
sync_priority	2

sync_state | **quorum**

```
-[ RECORD 2 ]-----
```

pid	923
usesysid	16384
username	repuser
application_name	pm02
client_addr	192.168.2.60
client_hostname	
client_port	35100
backend_start	2017-09-29 09:15:55.836521+09
backend_xmin	600
state	streaming
sent_lsn	0/41000140
write_lsn	0/41000140
flush_lsn	0/41000140
replay_lsn	0/41000140
write_lag	
flush_lag	
replay_lag	
sync_priority	1

sync_state | **quorum**

どちらもquorum

synchronous_standby_names

= 'ANY 1 (pm02)'

```
# select * from pg_stat_replication;
```

```
-[ RECORD 1 ]-----
```

pid	1002
usesysid	16384
username	repuser
application_name	pm03
client_addr	192.168.2.140
client_hostname	
client_port	48140
backend_start	2017-09-29 09:18:07.399915+09
backend_xmin	600
state	streaming
sent_lsn	0/41000140
write_lsn	0/41000140
flush_lsn	0/41000140
replay_lsn	0/41000140
write_lag	
flush_lag	
replay_lag	
sync_priority	0

sync_state | **async**

```
-[ RECORD 2 ]-----
```

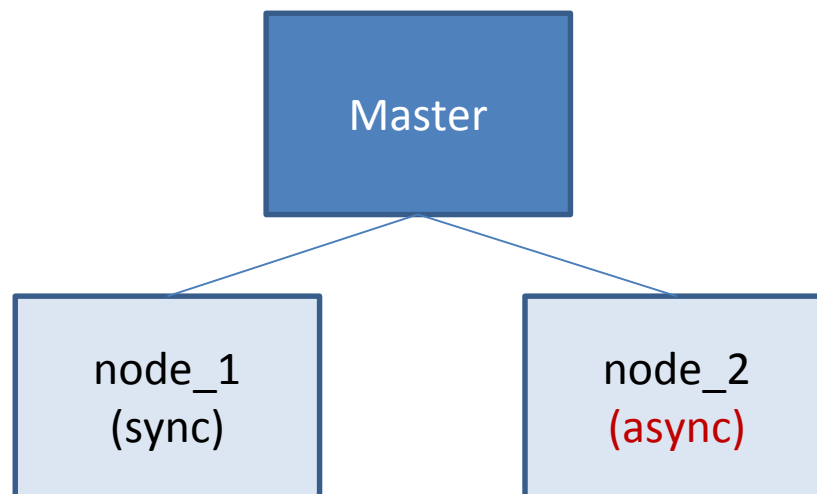
pid	923
usesysid	16384
username	repuser
application_name	pm02
client_addr	192.168.2.60
client_hostname	
client_port	35100
backend_start	2017-09-29 09:15:55.836521+09
backend_xmin	600
state	streaming
sent_lsn	0/41000140
write_lsn	0/41000140
flush_lsn	0/41000140
replay_lsn	0/41000140
write_lag	
flush_lag	
replay_lag	
sync_priority	1

sync_state | **quorum**

PG-REXでマルチシンクレプリケーション

現在のpgsql RA(resource-agents-4.0.1-1)では・・・

- 2Slave構成の構築は可能だが、2ノード目は非同期レプリケーションのため可用性はあまり向上しない
- 可用性が向上しない例
 1. node_1が故障した場合、node_2はasyncのまま、非同期レプリケーションを継続する
 2. 次に、Masterが故障した場合、node_2は**asyncのためMasterに昇格できない**
 3. **サービス停止**
 - **3重化しているのに2重故障に耐えられない！**



node_2がsyncであればMasterに昇格できるため、サービスは継続される

マルチシンク実装してみた

- まずはpriorityベースのみ
- 2018/02/23現在、本家コミュニティに未マージのため、お試し用のパッチは以下からダウンロード(コピペ or git clone)してください
 - https://github.com/ytakeshita/resource-agents/blob/support_multiple_synchronous_standby/heartbeat/pgsql

基本的な仕様

- PostgreSQL 9.6 以上で利用可能
- 設定するパラメータ
 - **sync_num (integer, [none]):**
 - 同期レプリケーションするノード数を指定 (最大値 : 998)
 - “2” 以上を指定した場合、マルチシンクレプリケーションを行う
- sync_num=“1” の場合の動作
 - 現在の pgsql RA と同様
 - synchronous_standby_names=“node_1”
- sync_num=“N” (N ≥ 2) の場合の動作
 - 1ノード目
 - synchronous_standby_names=“1(node_1)”
 - 2ノード目以降
 - synchronous_standby_names=“N (node_1, … , node_N)”
- Master故障時に、どのノードが昇格するの？
 - masterスコアの高いノード

動作の例(crm_mon -DfA)

- Master: pm03
- Slave: pm04, pm05 (どちらもSTREAMING|SYNC)
- Masterが故障した場合、pm04がMasterに昇格する
 - masterスコア(master-prmPostgresql)がpm04の方が高い

Online: [pm03 pm04 pm05]

Full list of resources:

vip-master (ocf::heartbeat:IPaddr2): Started pm03

vip-rep (ocf::heartbeat:IPaddr2): Started pm03

Resource Group: grpStonith1

prnStonith1-1 (stonith:external/stonith-helper): Started pm04

prnStonith1-2 (stonith:external/libvirt): Started pm04

Resource Group: grpStonith2

prnStonith2-1 (stonith:external/stonith-helper): Started pm03

prnStonith2-2 (stonith:external/libvirt): Started pm03

Resource Group: grpStonith3

prnStonith3-1 (stonith:external/stonith-helper): Started pm03

prnStonith3-2 (stonith:external/libvirt): Started pm03

Master/Slave Set: msPostgresql [prmPostgresql]

Masters: [pm03]

Slaves: [pm04 pm05]

Clone Set: clnDiskd-Inner [prmDiskd-Inner]

Started: [pm03 pm04 pm05]

Clone Set: clnPingd [prmPingd]

Started: [pm03 pm04 pm05]

Node Attributes:

* Node pm03:

+ default_ping_set : 100
+ diskcheck_status_internal : normal
+ master-prmPostgresql : 1000
+ prmPostgresql-data-status : LATEST
+ prmPostgresql-master-baseline : 0000000046000280
+ prmPostgresql-status : PRI
+ ringnumber_0 : 192.168.1.150 is UP

* Node pm04:

+ default_ping_set : 100
+ diskcheck_status_internal : normal
+ **master-prmPostgresql** : **101**
+ prmPostgresql-data-status : **STREAMING|SYNC**
+ prmPostgresql-status : **HS:sync**
+ ringnumber_0 : 192.168.1.150 is UP

* Node pm05:

+ default_ping_set : 100
+ diskcheck_status_internal : normal
+ **master-prmPostgresql** : **100**
+ prmPostgresql-data-status : **STREAMING|SYNC**
+ prmPostgresql-status : **HS:sync**
+ prmPostgresql-xlog-loc : 0000000046000390
+ ringnumber_0 : 192.168.1.160 is UP

昇格優先度 :
高

どちらも
SYNC(同期)

もうちょっとだけ続くんじゃない

- Q : pgsql RAでマルチシンクレプリケーション対応を行うだけで、本当に2重故障に耐えられるのか？

• A: No

- PostgreSQLの制御は出来るが、クラスタとしての制御は別
- 本来、2重故障に耐えるためには、**5ノード**でクラスタを構築する必要がある
- これを3ノードで (無理やり) 実現するために、**特別な設定**が必要
 - PG-REXプロジェクトや過去のOSCセミナーで公開しているPacemakerの設定ではできない

従来のPacemaker設定で発生する問題

- 従来のPacemaker設定(過去に公開しているPacemaker設定)では、以下のような問題が発生する
 1. Master孤立時にSlaveがSTONITH(強制電源断)される
 - 3重化しているにも関わらず、Master以外は停止するため2重故障に耐えられない
 - 最悪の場合は、MasterもSlaveも全部停止する(相撃ち)
 2. ハートビート(IC-LAN)通信の全断時に何が起こるか分からない
 - 1ノードを残して全てのノードが停止する
 - どのノードが生き残るかは不明
 - せめてMasterが生き残ってほしいが・・・
 - 最悪の場合はやはり相撃ち

詳細は【参考】を御参照ください

現時点の最適(と思われる)設定

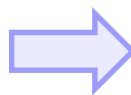
○ 通常の PG-REX (2ノード)

- Pacemakerの設定
 - **no-quorum-policy="ignore"**
- Corosyncの設定(corosync.conf)

```
totem {
  version: 2
  rrp_mode: active
  token: 1000
  rrp_problem_count_timeout: 2000
  interface {
    ringnumber: 0
    bindnetaddr: 192.168.1.0
    mcastaddr: 239.255.1.1
    mcastport: 5405
  }
  interface {
    ringnumber: 1
    bindnetaddr: 192.168.3.0
    mcastaddr: 239.255.1.2
    mcastport: 5405
  }
}

logging {
  syslog_facility: local1
  debug: off
}

quorum {
  provider: corosync_votequorum
  expected_votes: 2
}
```



○ マルチシンク

- Pacemakerの設定
 - **no-quorum-policy="freeze"**
- Corosyncの設定(corosync.conf)

```
totem {
  version: 2
  rrp_mode: active
  token: 1000
  rrp_problem_count_timeout: 2000
  interface {
    ringnumber: 0
    bindnetaddr: 192.168.1.0
    mcastaddr: 239.255.1.1
    mcastport: 5405
  }
  interface {
    ringnumber: 1
    bindnetaddr: 192.168.3.0
    mcastaddr: 239.255.1.2
    mcastport: 5405
  }
}

logging {
  syslog_facility: local1
  debug: off
}

quorum {
  provider: corosync_votequorum
  expected_votes: 1
  auto_tie_breaker: 1
}
```

現時点の最適(と思われる)設定

○ 通常の PG-REX (2ノード)

- Pacemakerの設定
 - **no-quorum-policy="ignore"**
- Corosyncの設定(corosync.conf)

```
totem {  
  version: 2  
  rrp_mode: active  
  token: 1000  
  rrp_problem_count_timeout: 2000  
  interface {  
    ringnumber: 0  
  }  
}  
int  
  
}
```

```
logging {  
  syslog_facility: local1  
  debug: off  
}
```

```
quorum {  
  provider: corosync_votequorum  
  expected_votes: 2  
}
```

○ マルチシンク

- Pacemakerの設定
 - **no-quorum-policy="freeze"**
- Corosyncの設定(corosync.conf)

```
totem {  
  version: 2  
  rrp_mode: active  
  token: 1000  
  rrp_problem_count_timeout: 2000  
  interface {  
    ringnumber: 0  
  }  
}
```

設定の詳細については【参考】を御参照ください

```
logging {  
  syslog_facility: local1  
  debug: off  
}
```

```
quorum {  
  provider: corosync_votequorum  
  expected_votes: 1  
  auto_tie_breaker: 1  
}
```

デモ

設定を変更してもまだ問題は発生する

- 以下のケースが発生した場合は、サービス停止となる(2重故障に耐えられない)可能性がある
 - 3ノードの場合
- **最後に残ったノードが、「高ノードID_(※)かつSlaveである」場合**
 1. 2ノード同時故障
 2. 1ノードずつ故障
- 【暫定対処】
 - 残ったノードで以下のコマンドを実行する
 - # corosync-cfgtool -R

根本対処にはCorosyncの機能追加が必要
現時点では不可

(※) Corosyncがノードを識別するためのID。以下のコマンドで確認可能
corosync-cfgtool -s

設定を変更してもまだ問題は発生する

- 以下のケースが発生した場合は、サービス停止となる(2重故障に耐えられない)可能性がある
 - 3ノードの場合
- **最後に残ったノードが、「高ノードID_(※)かつSlaveである」場合**
 1. 2ノード同時故障
 2. 1ノードずつ故障
- 詳細は【参考】を御参照ください
 - # corosync-cfgtool -R

根本対処にはCorosyncの機能追加が必要
現時点では不可

(※) Corosyncがノードを識別するためのID。以下のコマンドで確認可能
corosync-cfgtool -s

- Pacemaker-1.1.17-1.1

- 2018.2.14 リリース
 - PostgreSQL 10対応
 - bundle リソースタイプ追加 (Technology Preview 扱い)

- PG-REX マルチシンクレプリケーション対応

- PostgreSQL 9.6以上で利用可能
- 3ノード以上で同期レプリケーションが可能
 - 設定や運用は従来のPG-REXのものから変更が必要

- Pacemakerの設定
 - no-quorum-policyをfreezeに
- Corosyncの設定
 - expected_votesを“1”に
 - auto_tie_breakerを有効(“1”)に

- RAは現在pull request中

さいごに

Linux-HA Japan URL

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

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



The screenshot shows the Linux-HA Japan Project website. At the top is the logo with the text "LINUX-HA JAPAN High-Availability Clustering on Linux". Below the logo is a navigation bar with links: HOME, メーリングリスト, ダウンロード&インストール, マニュアル, デスクトップテーマ・壁紙等, コミュニティ概要. Below the navigation bar is a section titled "Linux-HA Japan プロジェクト" with a sub-header "Linux-HA Japan 成果物ダウンロード". The main content area lists various resources: "Linux-HA Japan 成果物ダウンロード" (RHEL/CentOS向けPacemaker RPMパッケージ, yumのリポジトリ形式や設定ファイル(crm)作成支援ツール, ディスク監視機能などをダウンロードできます。), "マニュアル" (本家コミュニティ提供の公式マニュアルやLinux-HA Japan提供の翻訳マニュアル。), "メーリングリスト" (インストール方法や設定方法等の質問はMLまで。), "イベント情報" (カンファレンスへの出席や講演、勉強会開催情報、講演時のスライド公開など。), "開発者向けサイト" (Linux-HA Japan開発者向けサイトです。Linux-HA Japan独自開発機能のソースコードやバイナリのダウンロード等。). At the bottom, there is a Twitter link and a footer with contact information.

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.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

ご清聴ありがとうございました。
May the Pacemaker be with you !



Linux-HA Japan

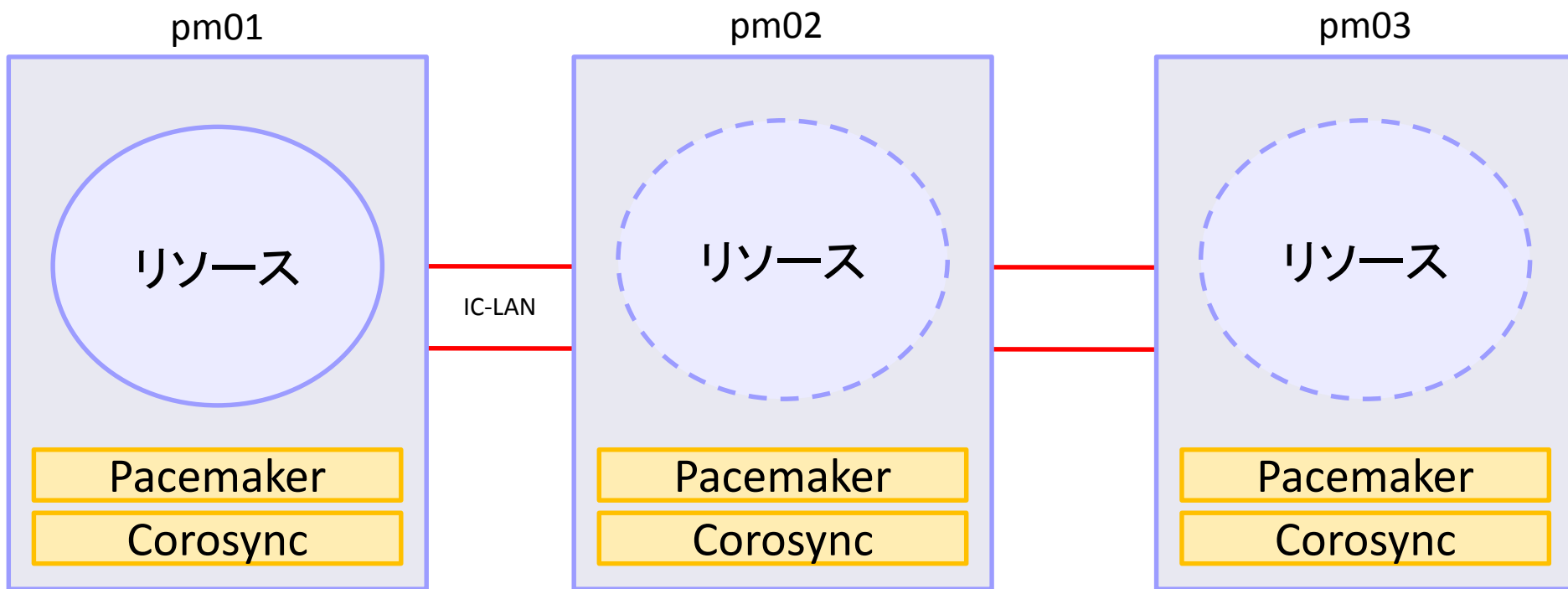
検索



【参考】 マルチシンクレプリケーション用設定 について

【本題の前に予備知識】 QuorumとSTONITH

- Pacemaker + Corosync はインターコネクトLAN(IC-LAN)に token と呼ばれるパケットを送受信する(※)ことで、クラスタメンバの状態を管理している

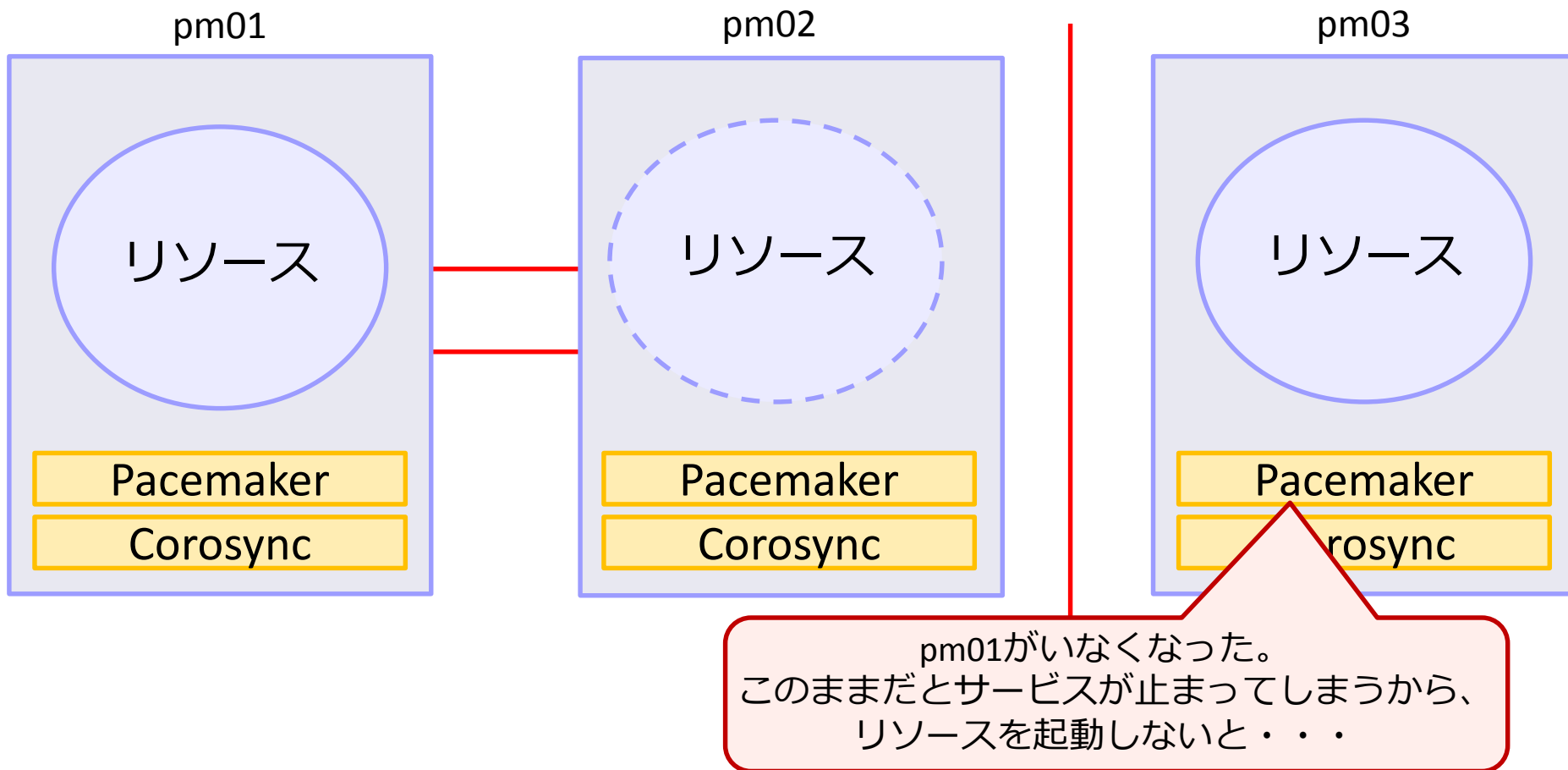


(※)本来はリング状にパケットを送信しているが、簡単のために横並びで図示

【本題の前に予備知識】 QuorumとSTONITH

Q: IC-LAN通信が一部途絶し、pm03と疎通できなくなったら？

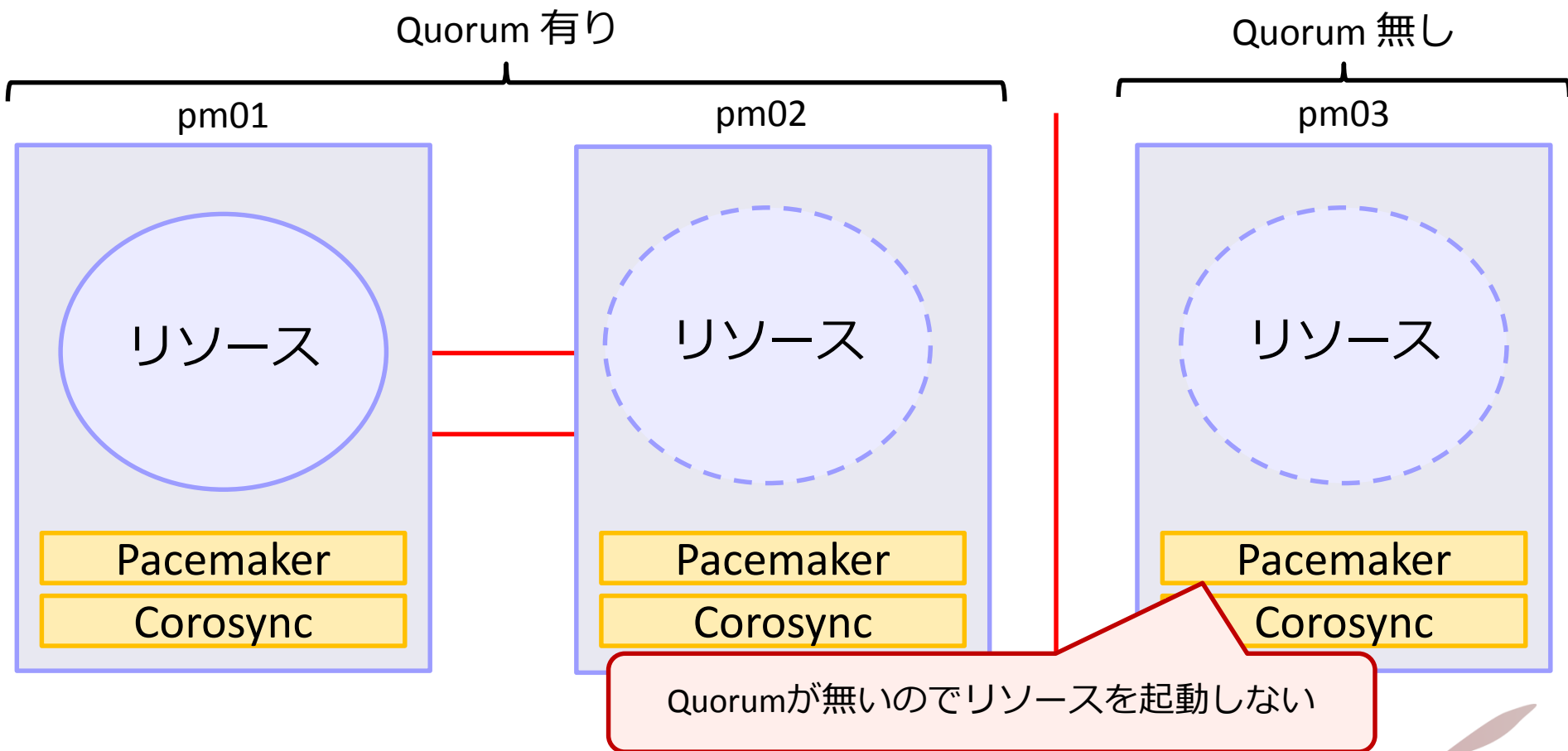
A: pm01, pm02のクラスタとpm03のクラスタに分離する



- 両クラスタでリソースが稼働 (スプリットブレイン)
- **最悪の場合、データ破壊が発生**する致命的な状態

【本題の前に予備知識】 QuorumとSTONITH

- Quorum (PostgreSQLのQuorumとは意味合いが異なる)
 - ざっくりいうと、多数決によるリソース制御権獲得の仕組み
 - 1. ノード総数の過半数以上が属するクラスタがQuorumを獲得する(※)
 - 2. Quorumを獲得したクラスタがリソースを制御する権利を持つ

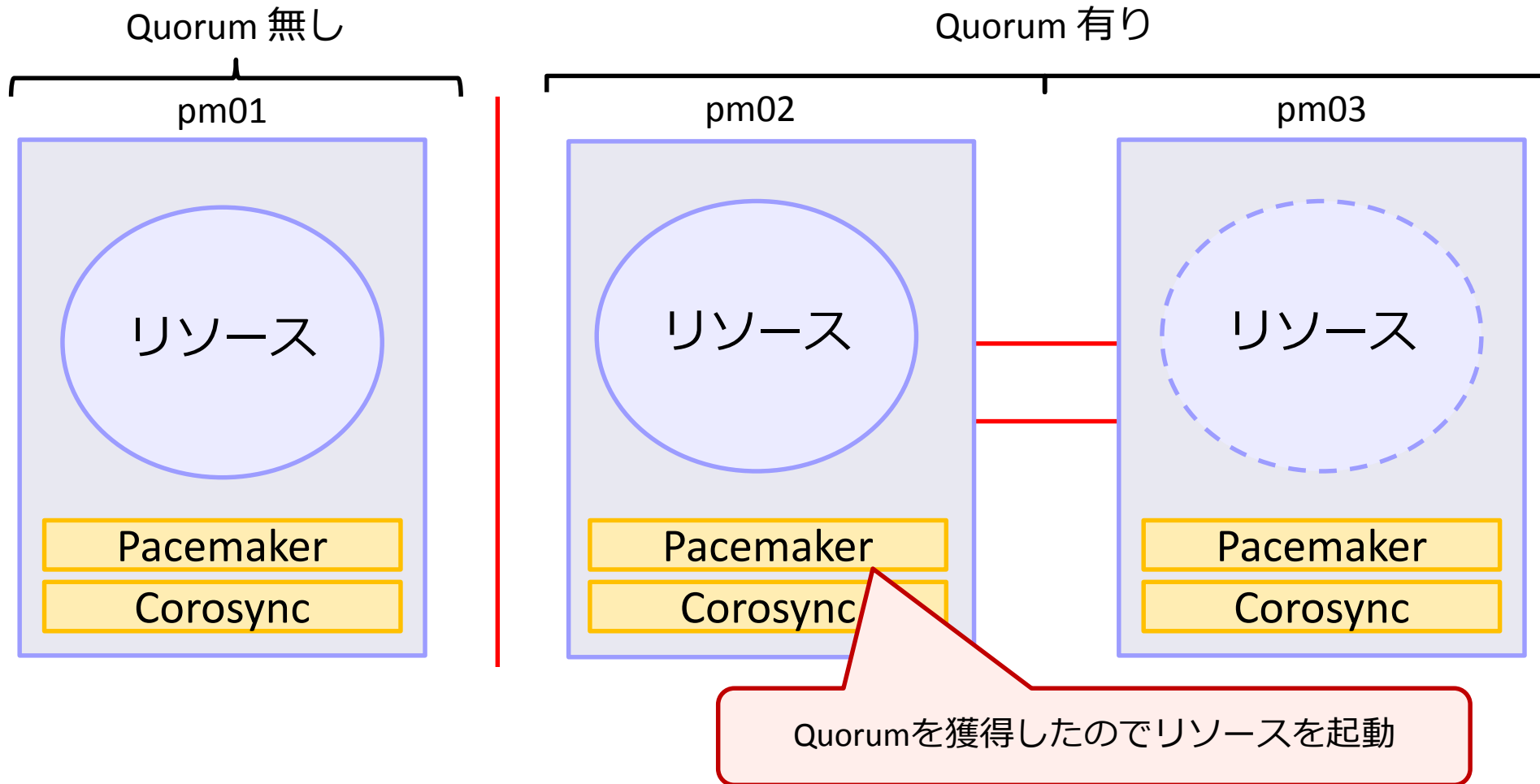


(※) 各ノードの投票数(重み)が 1 の場合

【本題の前に予備知識】 QuorumとSTONITH

Q: pm01が孤立すると両クラスタでリソースが起動するのでは・・・？

A: Exactly (そのとおりでございます)

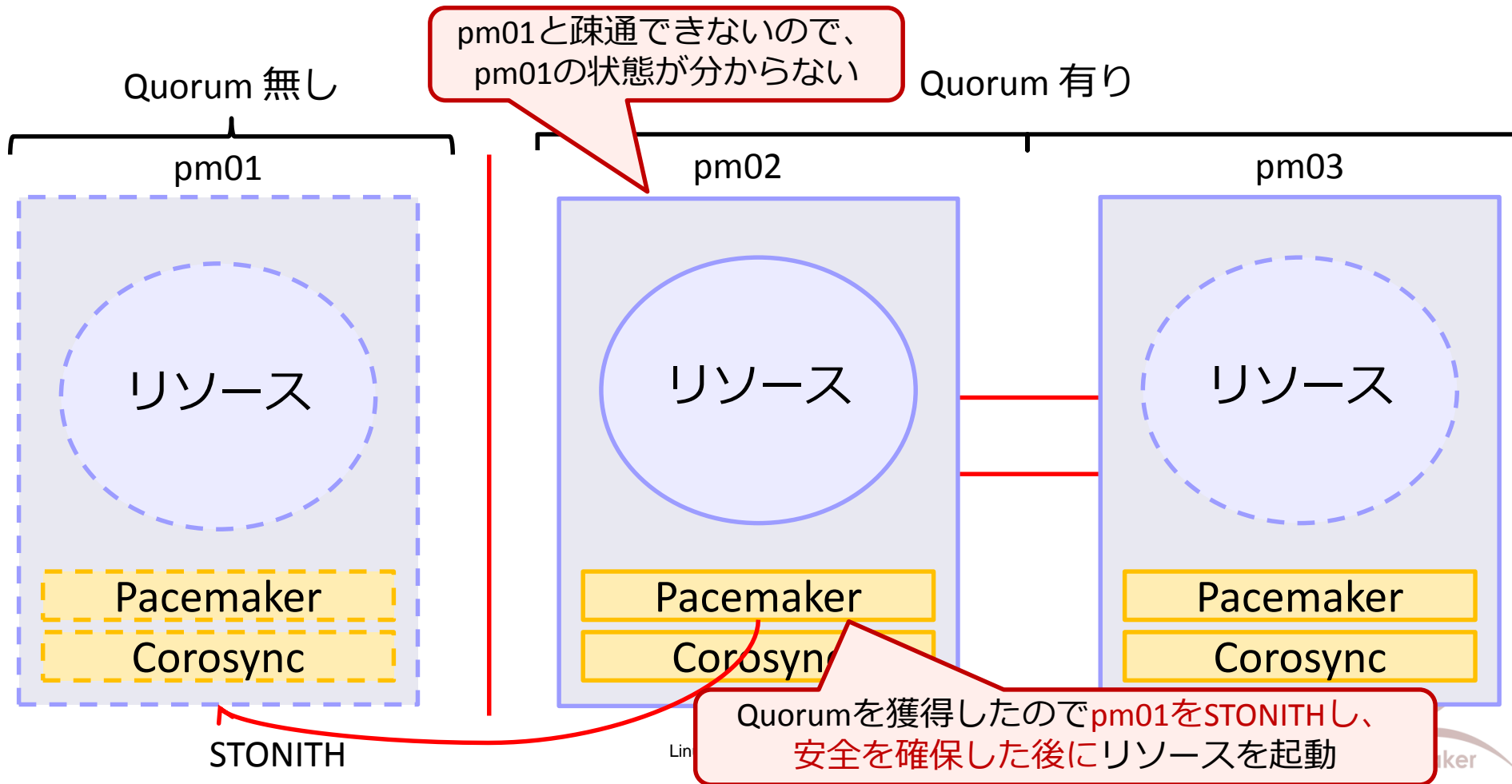


- STONITHによって、リソースの2重起動を防ぐ

【本題の前に予備知識】 QuorumとSTONITH

• STONITH

- 状態が分からなくなったクラスタメンバを強制的に電源断する機能
 - 電源断することで、リソースの2重起動を防ぐ
- 物理環境ではハードウェア制御ボード(iLOなど)を利用
- 仮想環境、AWSなども対応可能



【本題の前に予備知識】 QuorumとSTONITH

- ノードがQuorumを失った場合の動作は、Pacemakerの設定に依存している
- no-quorum-policy

パラメータ	動作概要
ignore	Quorumの有無に関わらず、リソース管理を継続する
freeze	Quorumを失ったノードでリソース管理を継続するが、新たなリソースの起動は行わない
stop (デフォルト)	Quorumを失ったノードで稼働している全てのリソースを停止する
suicide	Quorumを失ったノードは停止する

- ここまで説明してきた動作は、**no-quorum-policy="freeze"**を設定した場合の動作

【本題】

- PG-REX マルチシンクレプリケーション(3ノード構成)で2重故障に耐えられるようにするためには、どのような設定をすればよいだろう

通常のPG-REX(2ノード構成)の設定

- Corosyncの設定 (corosync.conf)

```
totem {
  version: 2
  rrp_mode: active
  token: 1000
  rrp_problem_count_timeout: 2000
  interface {
    ringnumber: 0
    bindnetaddr: 192.168.1.0
    mcastaddr: 239.255.1.1
    mcastport: 5405
  }
  interface {
    ringnumber: 1
    bindnetaddr: 192.168.3.0
    mcastaddr: 239.255.1.2
    mcastport: 5405
  }
}

logging {
  syslog_facility: local1
  debug: off
}

quorum {
  provider: corosync_votequorum
  expected_votes: 2
}
```

expected_votes: 2
2ノードでクラスタを構成することを期待
(1ノードではQuorumを獲得できないので単
体起動できない)

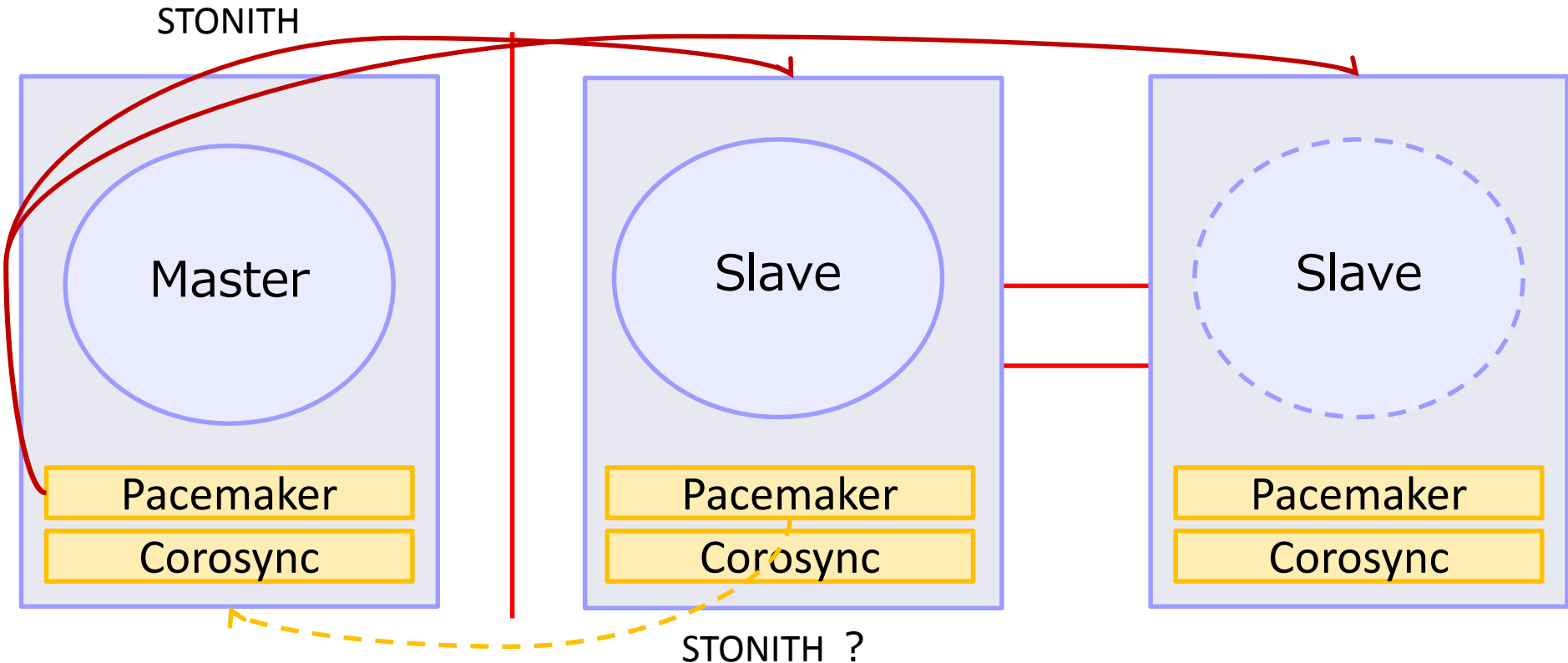
- Pacemakerの設定
 - no-quorum-policy="ignore" }

PG-REXはSlaveの初期同期のため、
Master単体で起動する必要がある

前ページの設定(※)でマルチシンクレプリケーション

※ expected_votesは3に変更する

- 発生する問題 1 : Master孤立時にSlaveがSTONITHされる(最悪相撃ち)



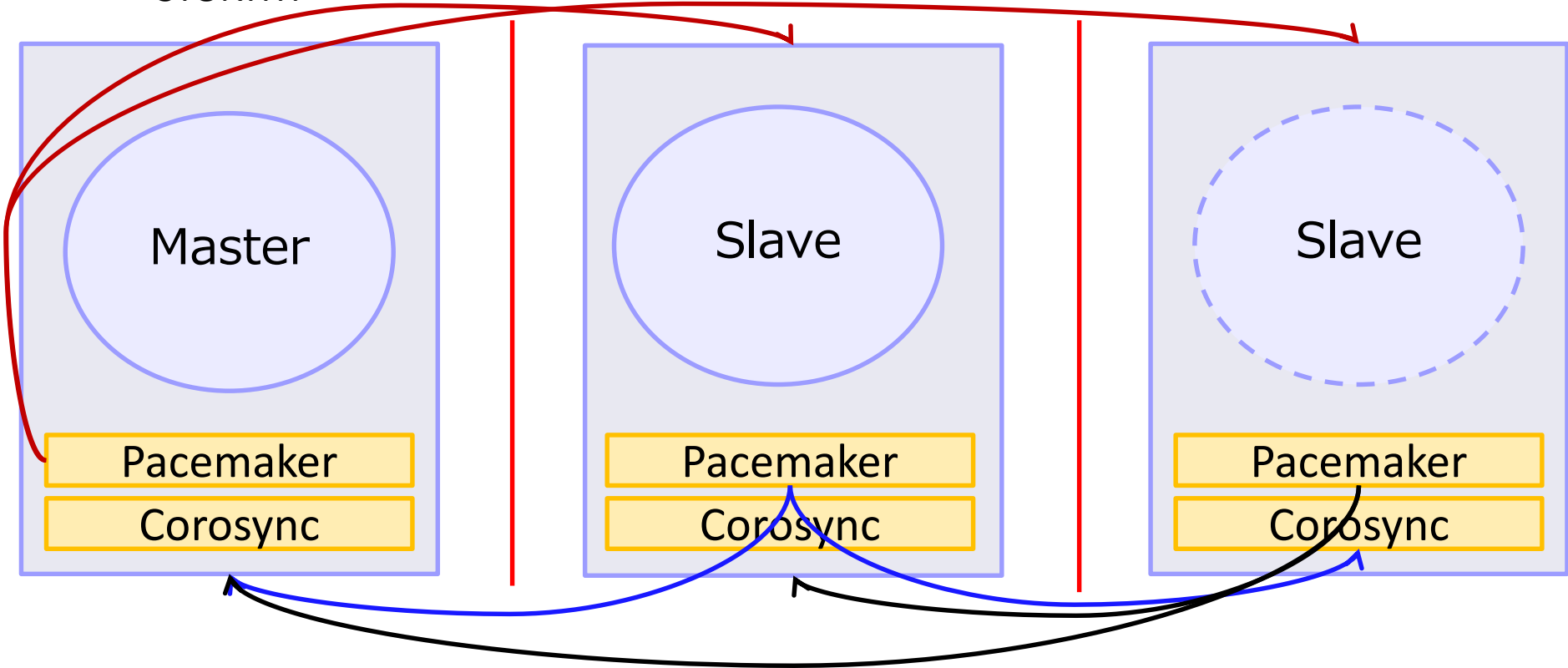
- 多重度の観点では、SlaveがMasterをSTONITHして昇格すべきだが・・・
- no-quorum-policy="ignore"なので、(Quorumのない)MasterもSTONITH実行可能
- Masterの方がSTONITHの実行が早いため、**Masterだけが生き残る**
 - 3重化しているのに、2重故障に耐えられない

前ページの設定(※)でマルチシンクレプリケーション

※ expected_votesは3に変更する

- 発生する問題 2 : IC-LAN全断時に何が起こるか分からない

STONITH



- 1ノードしか生き残らないのは仕方ない(せめてMasterが生き残ってほしい)
- 全てのノードが他のノードに対してSTONITHを実行しようとする
 - Masterが生き残る可能性は高いが、絶対ではない ⇒ 最悪相撃ち

no-quorum-policy="freeze"にできないか

- 前述の問題はno-quorum-policy="ignore"が原因
 - freezeにすれば解決するのでは・・・？
- freezeにすると、新たな問題が発生する
 1. freezeでは単ノード起動できない
 - PG-REXは初期同期のため、Master単ノード起動が必要
 2. ノードが順番に故障すると、2ノード目の故障でQuorumを失う
 - 2ノード目がMasterの場合、サービス停止
 - 3重化しても多重度が2ノードと変わらない！

解くべき課題：

1. no-quorum-policy="freeze"で単ノード起動を実現する
2. 2ノード故障してもQuorumを保持する

設定例

- Corosyncの設定 (corosync.conf)
 - expected_votes: 1
 - 初期値を1にすることで、単ノードでもQuorum獲得
 - 他ノードが参加した場合は自動的に加算される
 - auto_tie_breaker: 1
 - 偶数ノードでクラスタを構成し、半々に分離した場合にクラスタメンバのnode id (※)の低い方がQuorumを持つ
 - 奇数ノードのクラスタでauto_tie_breakerを有効にした場合は、「条件付きで」**過半数 - 1**までQuorumを持つことができる

```
totem {
  (snip)
}

logging {
  (snip)
}

quorum {
  provider: corosync_votequorum
  expected_votes: 1
  auto_tie_breaker: 1
}
```

- Pacemakerの設定
 - **no-quorum-policy="freeze"**

(※) CorosyncがIPアドレスから算出する、ノードを一意に識別するID。以下のコマンドで確認可能
corosync-cfgtool -s

奇数ノードでのauto_tie_breakerの条件

- 奇数ノードのクラスタでauto_tie_breakerを有効にした場合は、「条件付きで」過半数 - 1までQuorumを持つことができる
- 残ノードがQuorumを獲得できないケース (3ノードの場合)
 - IC-LANの同時全断：
 - no-quorum-policy="freeze"のため、STONITHは実行されない
 - Masterが停止しないため、サービス停止にはならない
 - 2ノードが同時に停止
 - 残ノードがSlaveの場合はサービス停止
 - 残ノードがMasterの場合はサービス継続
 - 2ノード目故障が、低ノードIDのCorosync異常停止(ノード故障、Corosync故障)の場合：
 - 残ノードがSlaveの場合はサービス停止
 - 残ノードがMasterの場合はサービス継続

【暫定対処】

- サービス停止した場合は、残ノードでcorosync.confを再読み込み！
 - # corosync-cfgtool -R