ココまでできるの！？
Amebaを支えるMySQLシステム構築

株式会社サイバーエージェント
新規開発局 インフラテクノロジーグループ
岡田 達典

●Amebaの開発組織

- ➢フロントクリエイティブグループ
- ➢システムクリエイティブグループ
- ➢インフラテクノロジーグループ
  - ◆ネットワークエンジニア
  - ◆サーバ・プラットホームエンジニア
  - ◆データベースエンジニア

• データベースエンジニアの仕事

- 各システムのパフォーマンス改善
- OS,サービスに合わせたMySQLなどのRPM作成
- MySQLサーバの運用方針,監視設定管理
- データセンター移設(物理,論理)
- 新技術の検証と導入検討
- サーバのラッキング及びサーバ構築
- サーバ選定(動作＆負荷検証)

Amebaでは、ほぼ全てサービスで MySQLを利用しています

年間50台以上の

MySQLサーバが増え

約3年でサーバ台数が

250台以上になりました

- アジェンダ

➢1.レプリケーション

1-1.レプリケーションを利用した分散について

1-2.用途に合わせたレプリケーション

1-3.レプリケーションを効率的に使うための設計

1-4.ストレージエンジンとレプリケーション

1-5.多階層のレプリケーション

➢2.MySQLの運用

2-1.バックアップ

2-2.障害監視

2-3.性能監視

➢3.今後の取り組み

1. レプリケーション

データベースの運用で

問題になる事として

I/O負荷が問題で

SELECTのレスポンスが

低下することがあります

解決としてI/Oを分散させるために

Diskの数を増やしたり

I/Oを減らすために

Memory増設で

解決することが出来ます

ですが、それでは

コストを抑えることが出来ないし、

DBエンジニアとしては

悲しいので

MySQLの特徴である

レプリケーションを使用した

I/O負荷の軽減例や

Amebaで実際に行っている

負荷分散について説明したいと思います

データベースの

改善としては

SQLのチューニングや

インデックスの見直し

などもありますが

ここでは、レプリケーションを利用した

負荷分散を中心に

説明したいと思います

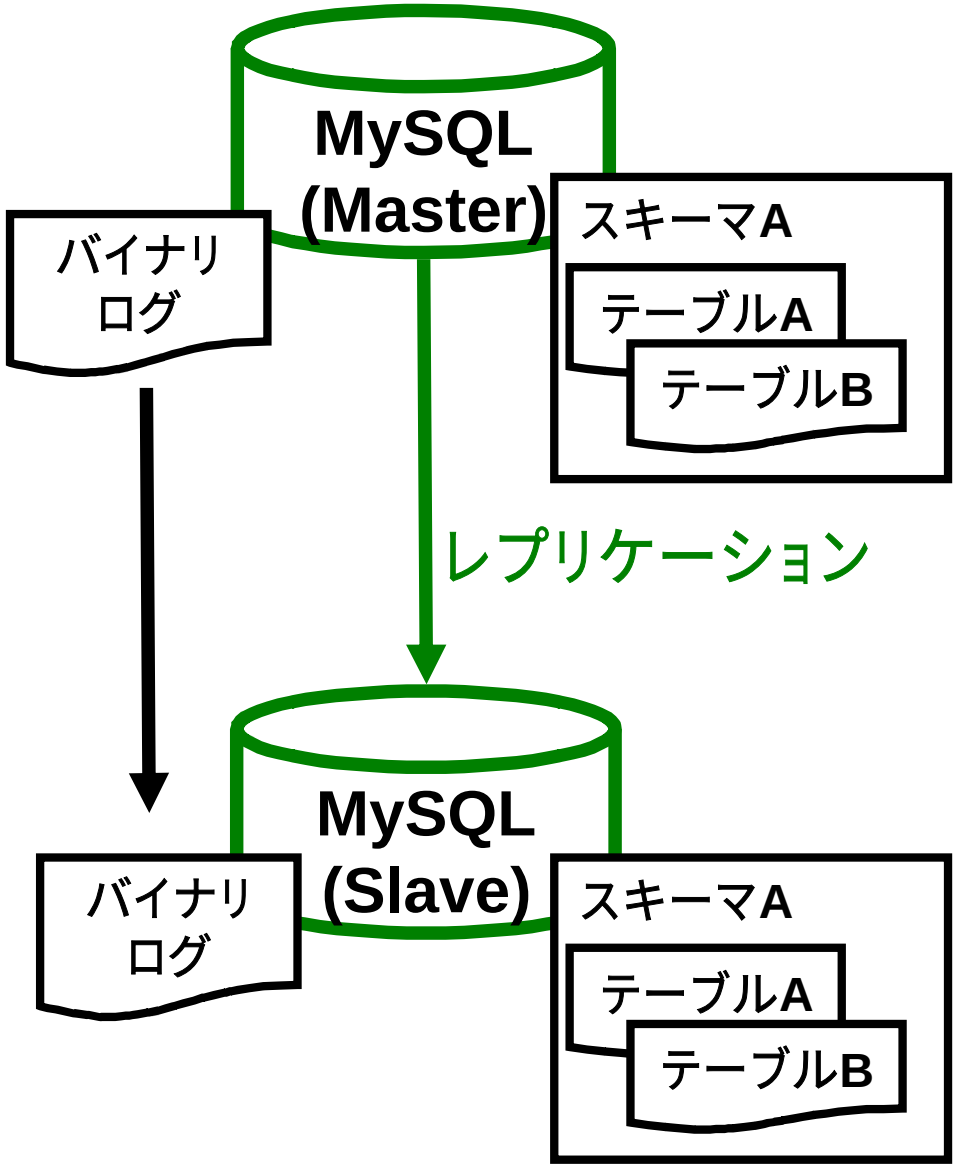
レプリケーションとは

一般的にあるデータベースから

他のデータベースに複製を作成する

事をいいます

- レプリケーションについて



MySQLは他のデータベースと違い

レプリケーション機能を使用することで

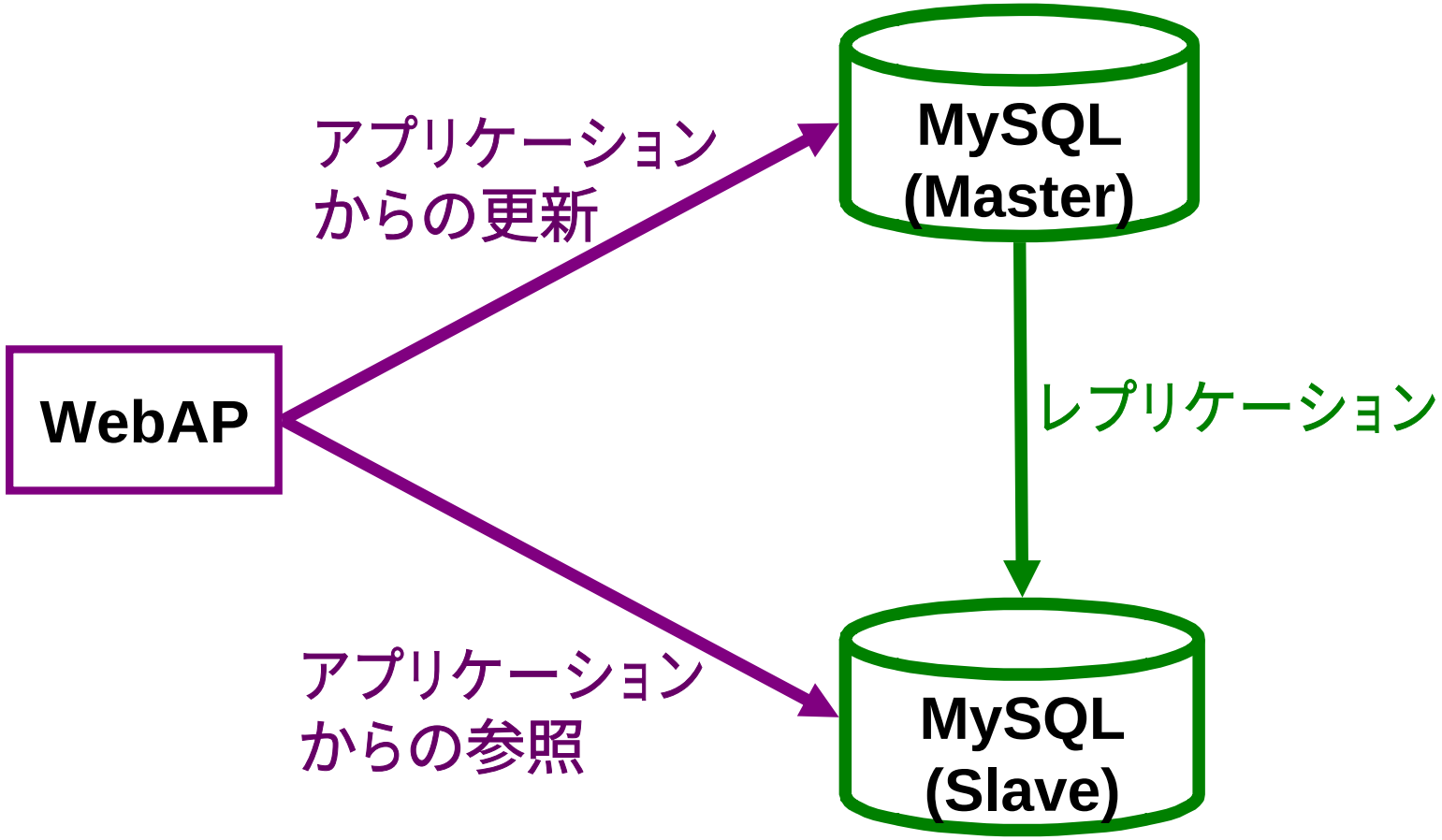
簡単に負荷を分散することが出来ます

Amebaでもっとも

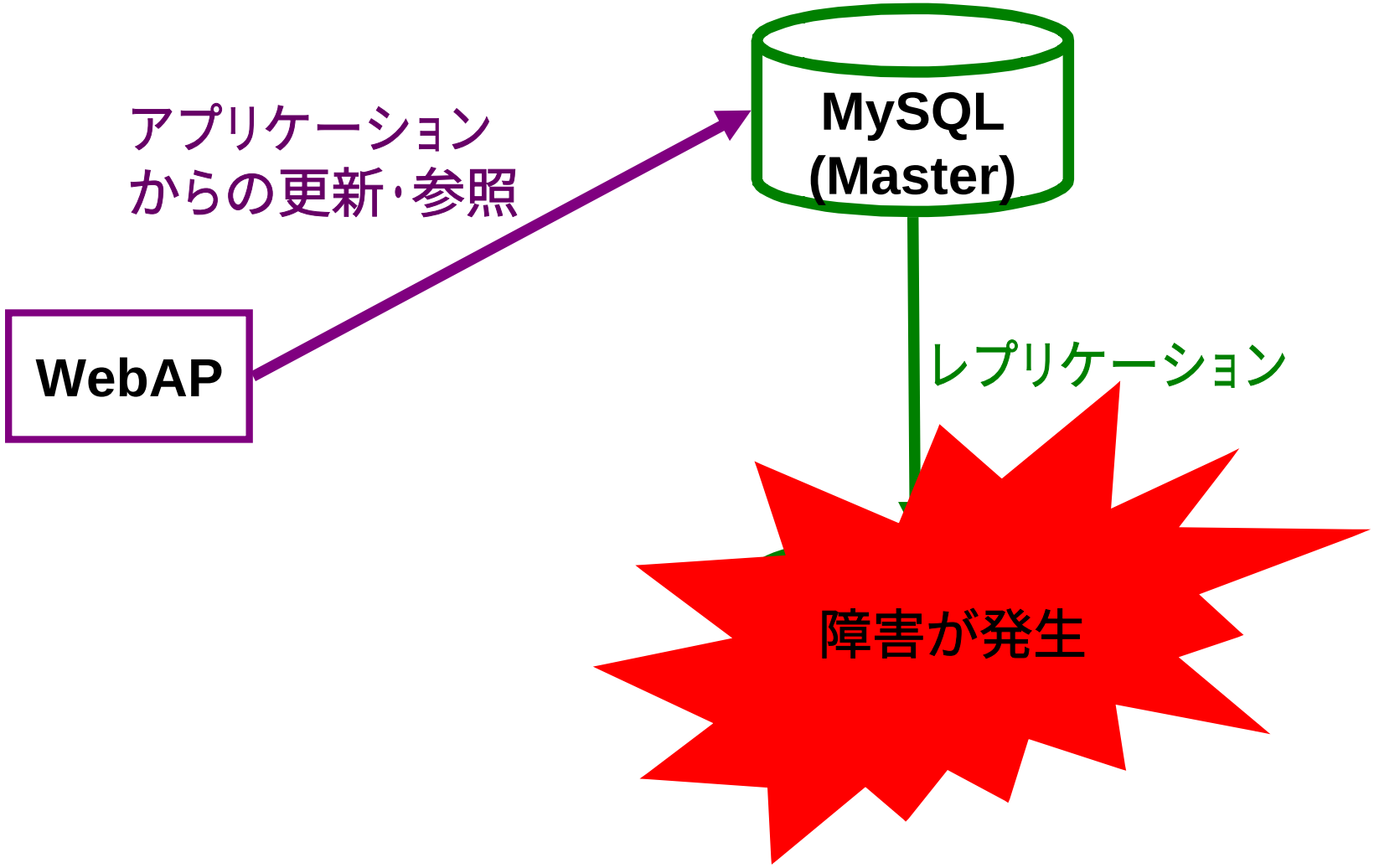
多くあるMySQLの

サーバ構成として

- 単純なレプリケーションと障害対応



- 単純なレプリケーションと障害対応



1-1.レプリケーションを利用した分散について

レプリケーションは
設定によって
分散方法を選択することが出来ます

スキーマ別やテーブル別で
Slaveサーバを構築する
事が可能です

始めにスキーマ別で分散する方法について
説明したいと思います

スキーマ別での分散は
「アメばた会議」で
利用しています

- スキーマ別のレプリケーション

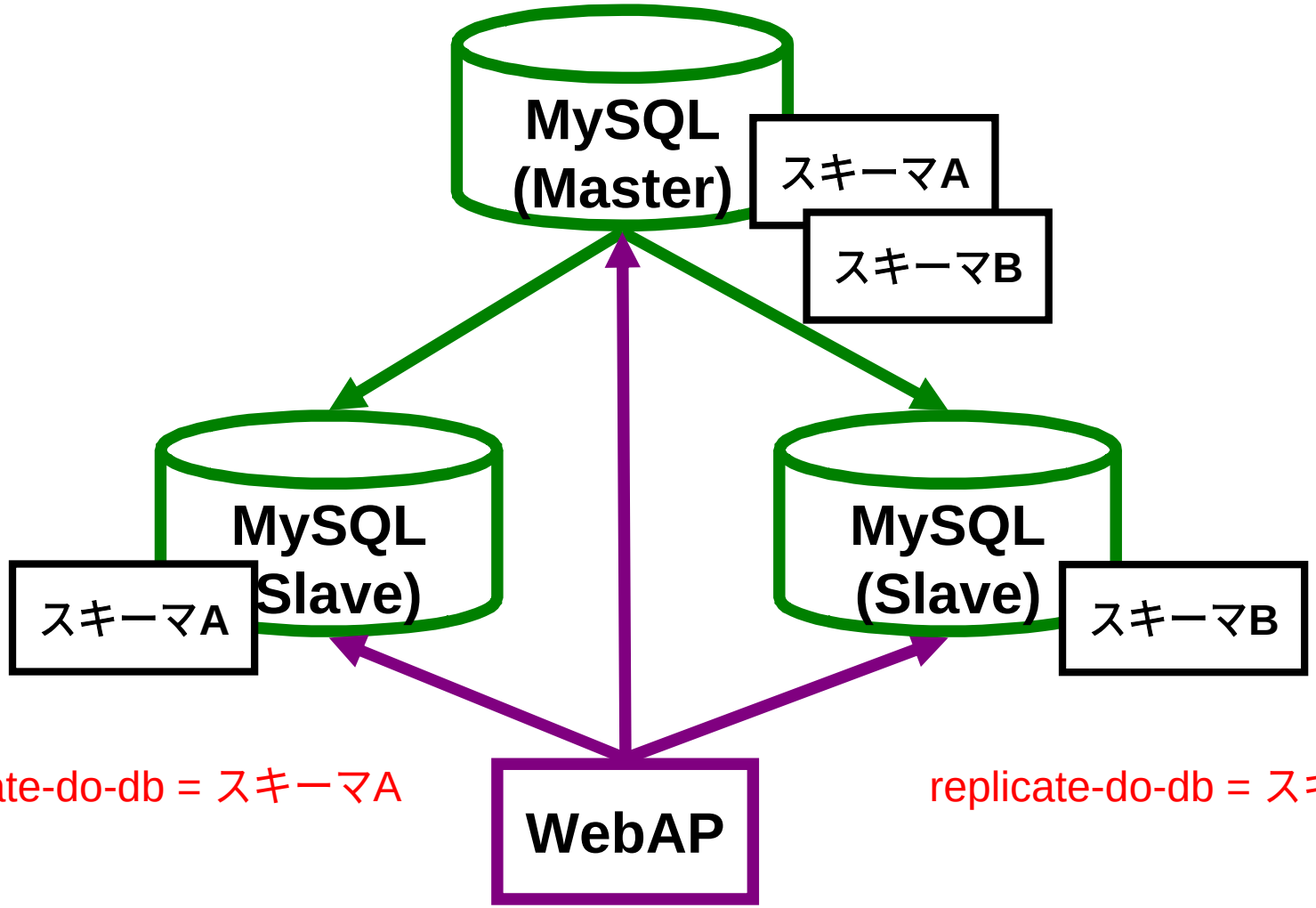
「**Replicate_Do_DB**」を**my.cnf**に

設定することで

スキーマ別に**Slave**サーバを

分散する構成です

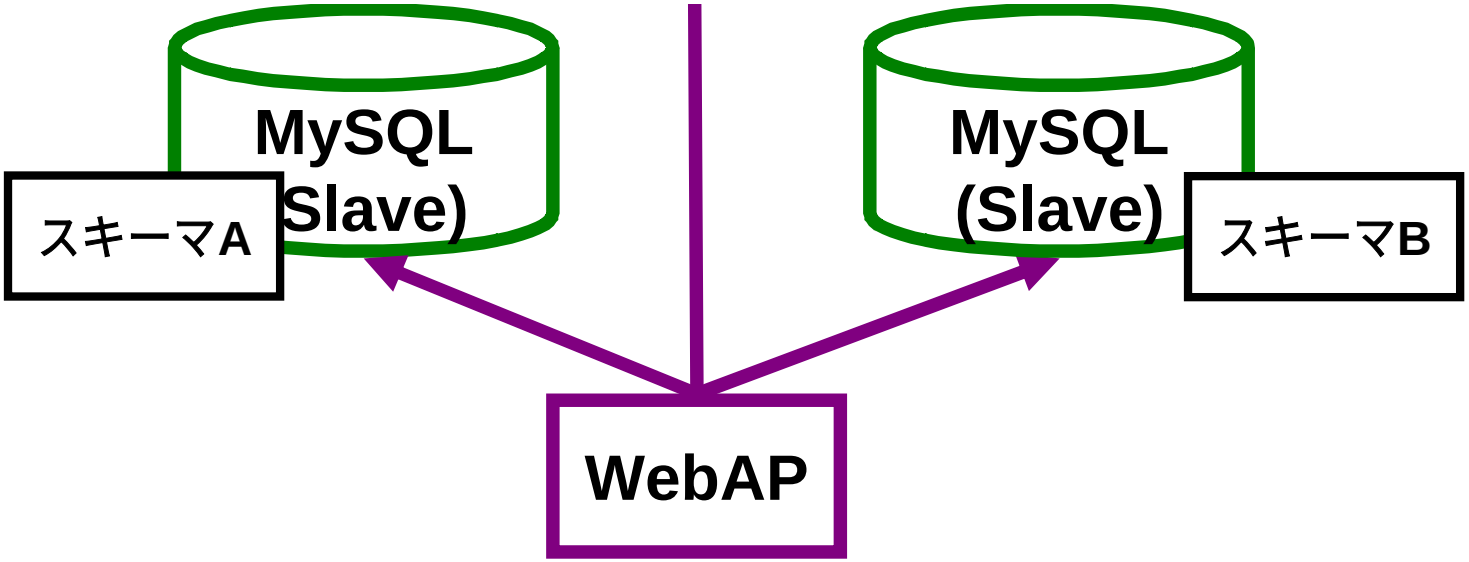
- スキーマ別のレプリケーション



スキーマ別にSlaveサーバを

構築できるので

管理面から見てもメリットがあります



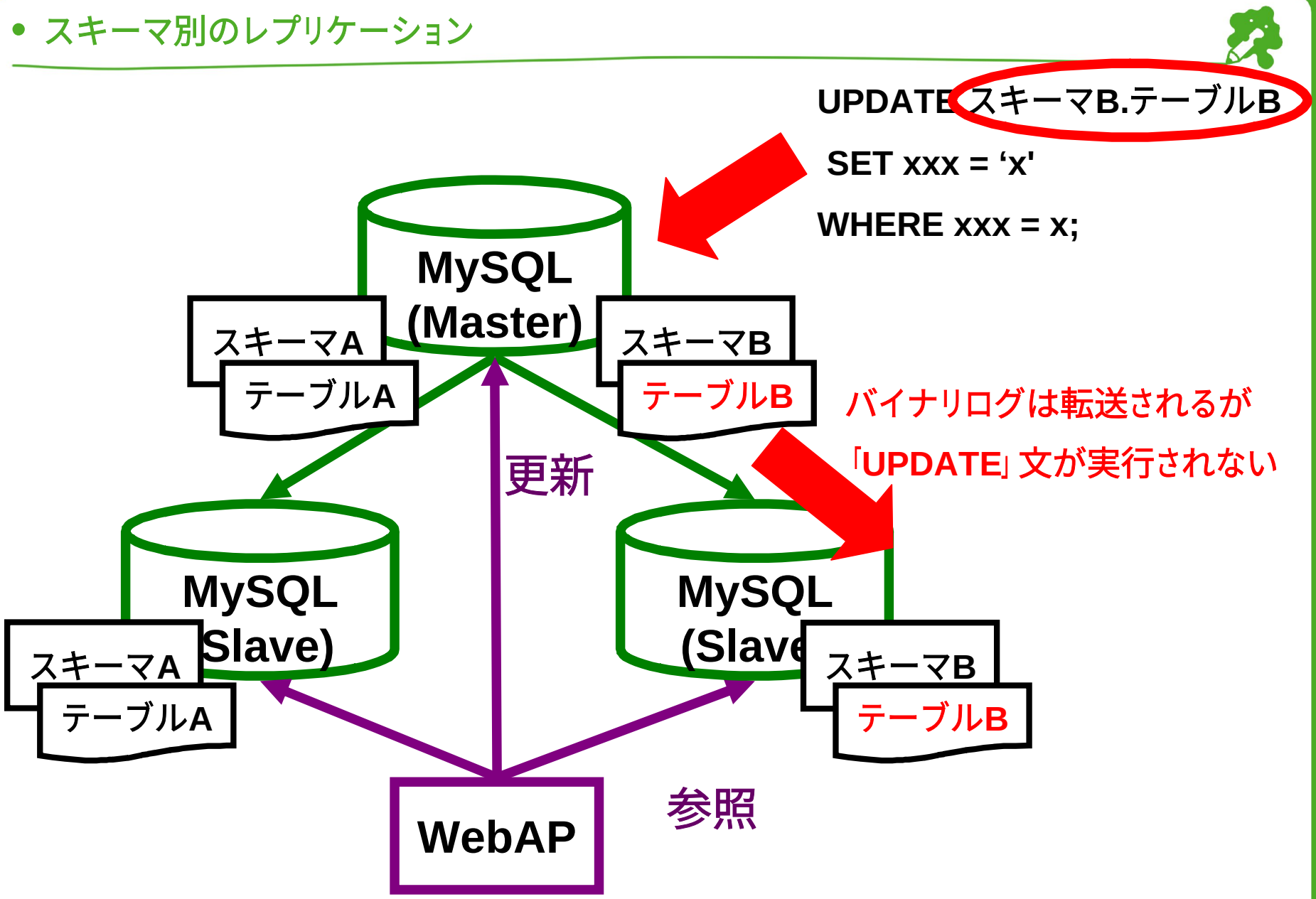
MySQL4.1で、この様な構成にした場合

更新系**SQL**の記述によっては

Slaveで**SQL**が実行されないことが

あったので確認を行う必要があります

- スキーマ別のレプリケーション



次にテーブル別で分散する方法について
説明したいと思います

テーブル別での分散は
「ブログ」や「なう」など多くのサービスで
利用しています

- テーブル別のレプリケーション

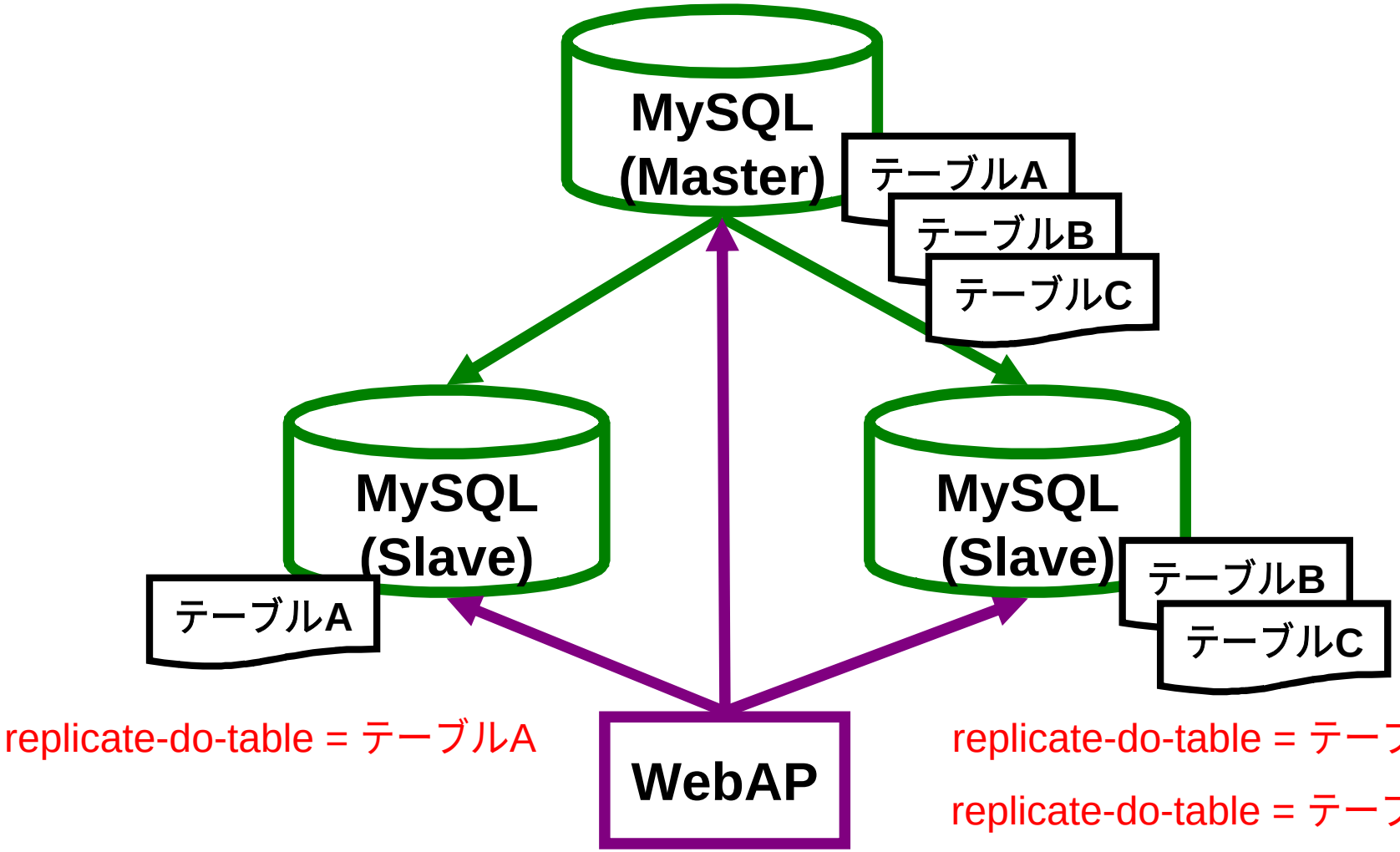
「**Replicate_Do_Table**」を**my.cnf**に

設定することで

テーブル別に**Slave**サーバを

分散する構成です

- テーブル別のレプリケーション



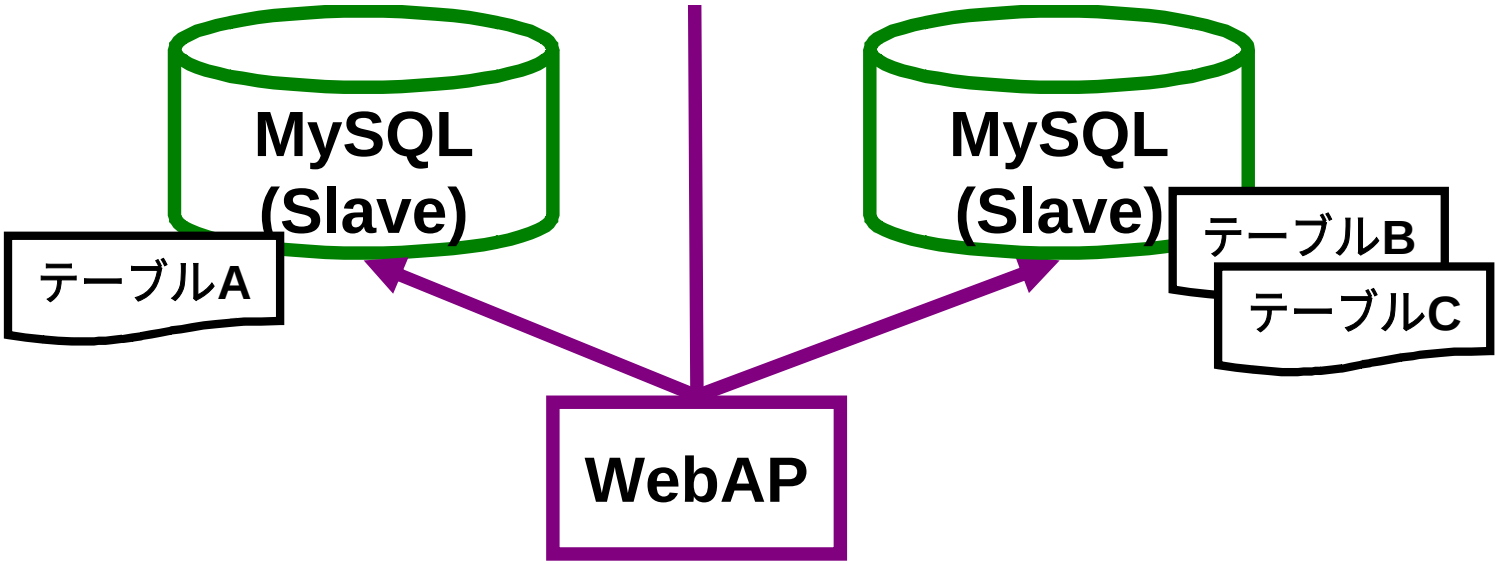
- テーブル別のレプリケーション

テーブルの特性を考え(参照が多い、更新が多いなど)

レプリケーションの設定を行うことで

「**Replicate_Do_DB**」より

効率的に**I/O**負荷軽減を行うことが出来るでしょう



- テーブル別のレプリケーション

「**Replicate_Do_Table**」は、

管理が大変になってしまう

と言ったデメリットがあります

1-2.用途に合わせたレプリケーション

アプリケーションの機能に合わせて

Slaveサーバの構成を

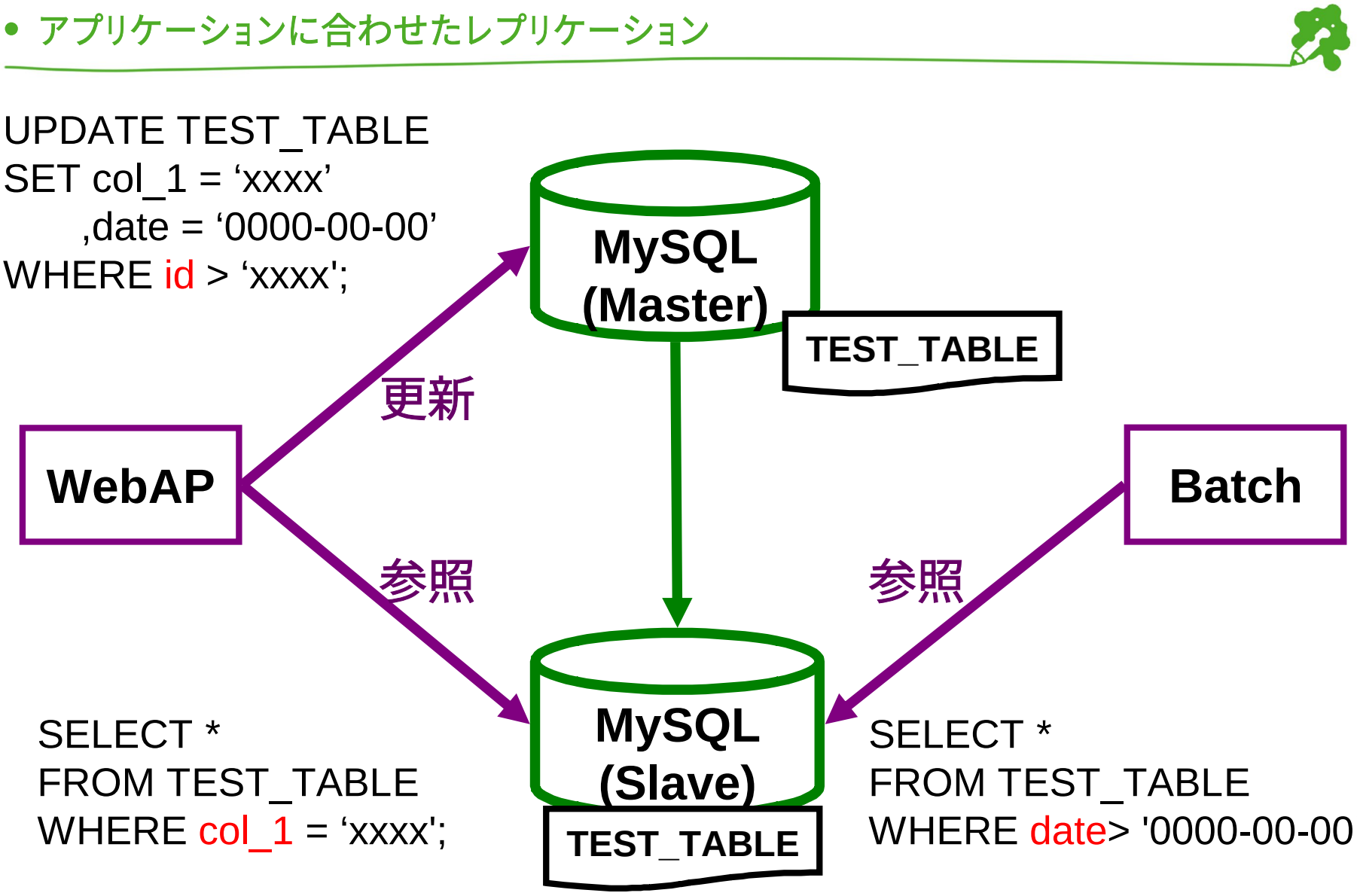
決めることがあります

以下のテーブル(TEST_TALBE)

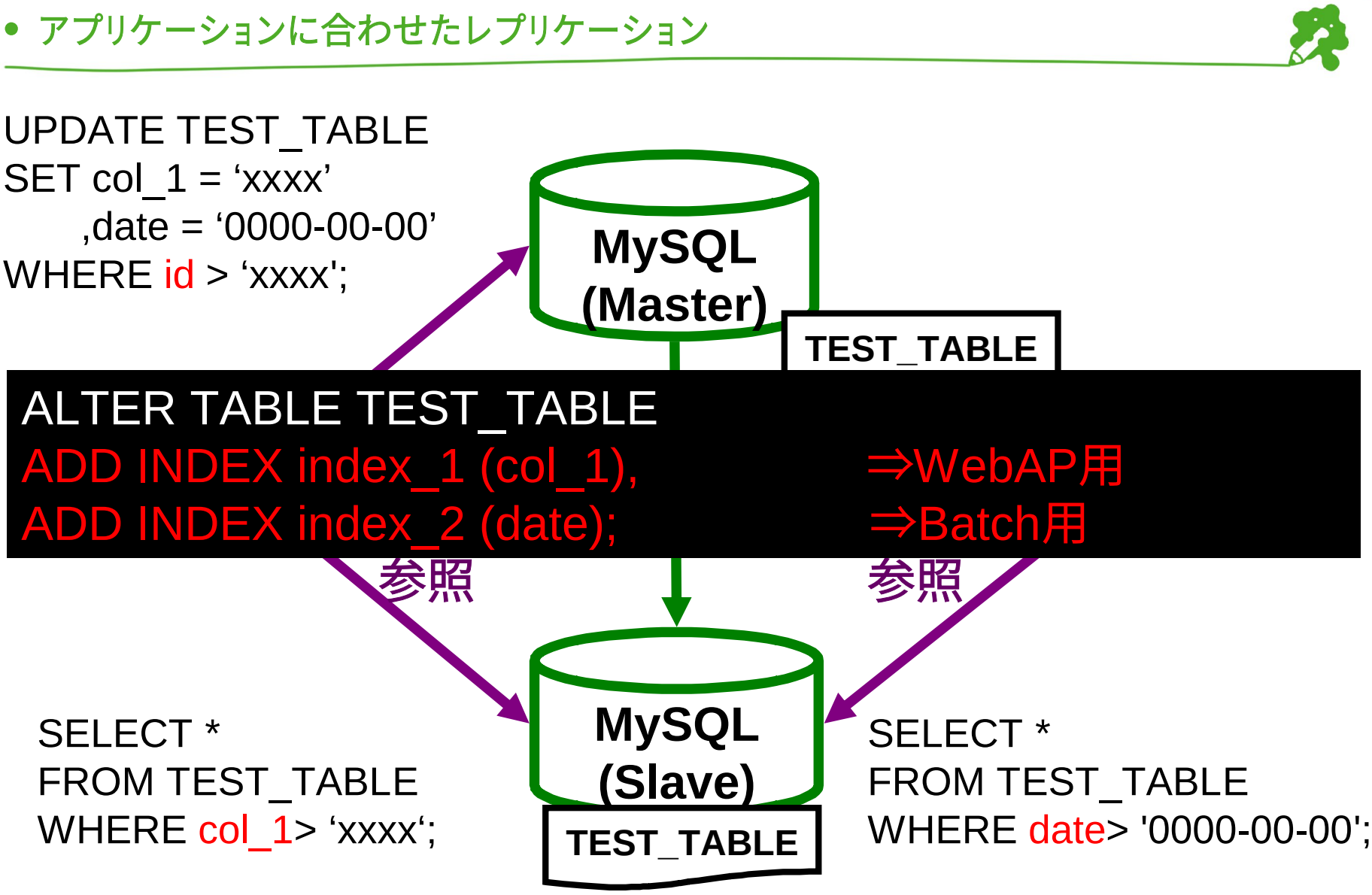
を使用して説明してきます

```
mysql> SHOW CREATE TABLE TEST_TABLE\G
*************************** 1. row ***************************
       Table: TEST_TABLE
Create Table: CREATE TABLE `TEST_TABLE` (
  `id` int(10) NOT NULL default '0',
  `col_1` varchar(10) default NULL,
  `date` datetime default NULL,
  PRIMARY KEY  (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8
1 row in set (0.01 sec)
```

- アプリケーションに合わせたレプリケーション



- アプリケーションに合わせたレプリケーション



こうした場合、

「**TEST_TABLE**」のデータが増加した場合に

バッチ処理のテーブルロック時間が

長くなってしまい、トランザクション処理で

レスポンスが低下しまう事があります

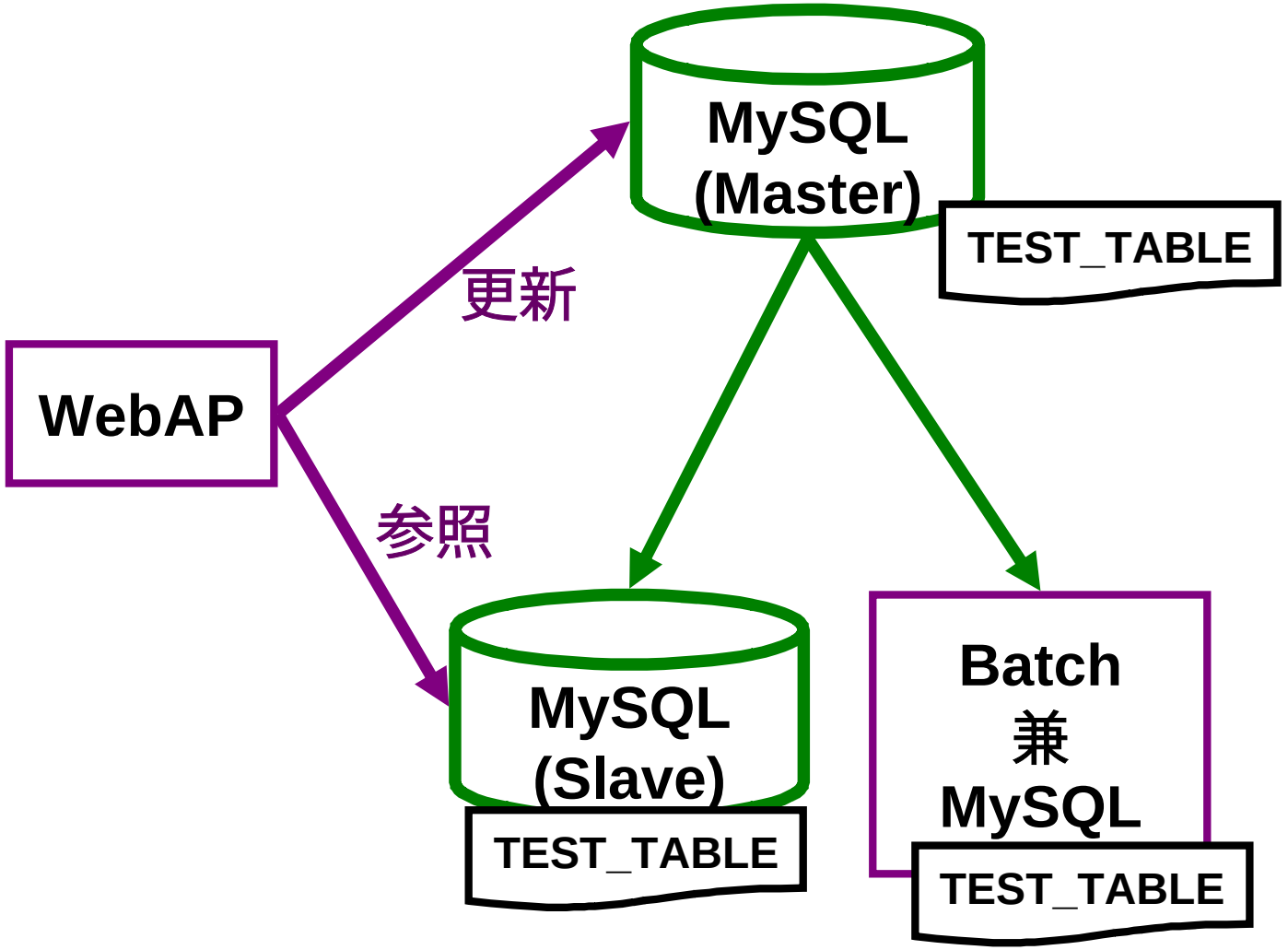
解決案として

次のような構成に

することで検索レスポンスを

維持することが出来ます

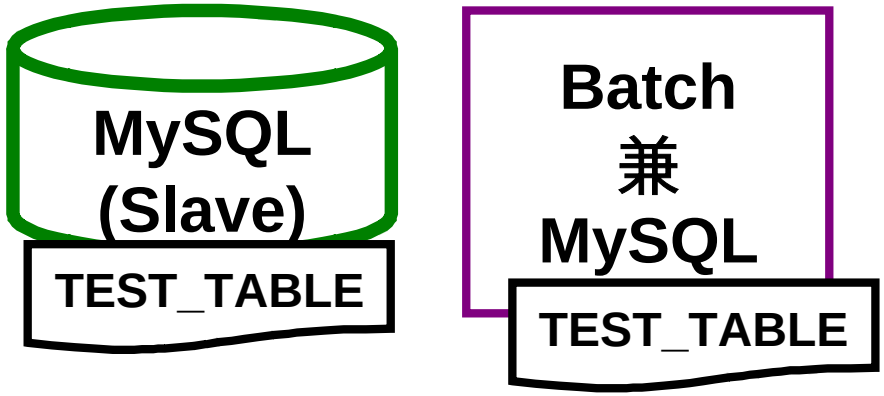
- アプリケーションに合わせたレプリケーション



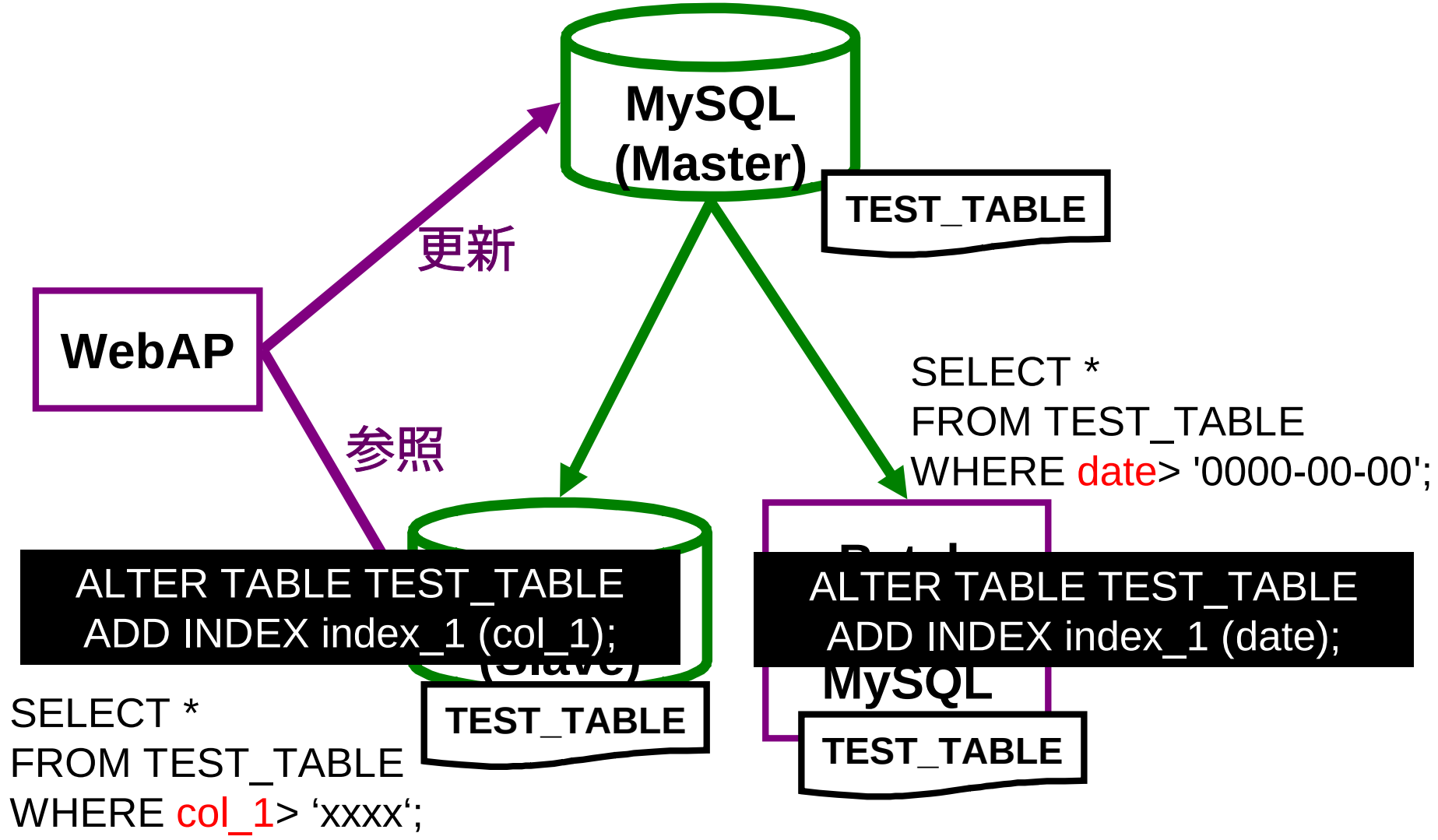
サーバを用途ごとに分けることで

検索のレスポンスを

維持することが出来るでしょう



- アプリケーションに合わせたレプリケーション

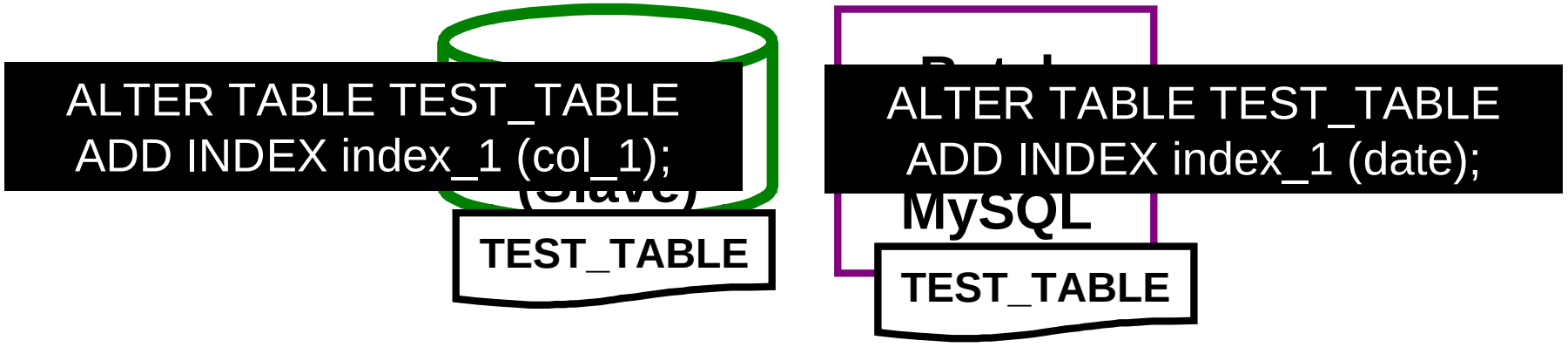


インデックスを

個別に作成することで

Disk容量の確保や

更新時のI/O負荷軽減にもつながります



1-3.レプリケーションを効率的に使う設計

レプリケーションをより効率的に

使用するためのテーブル設計や

SQLを考えてみましょう

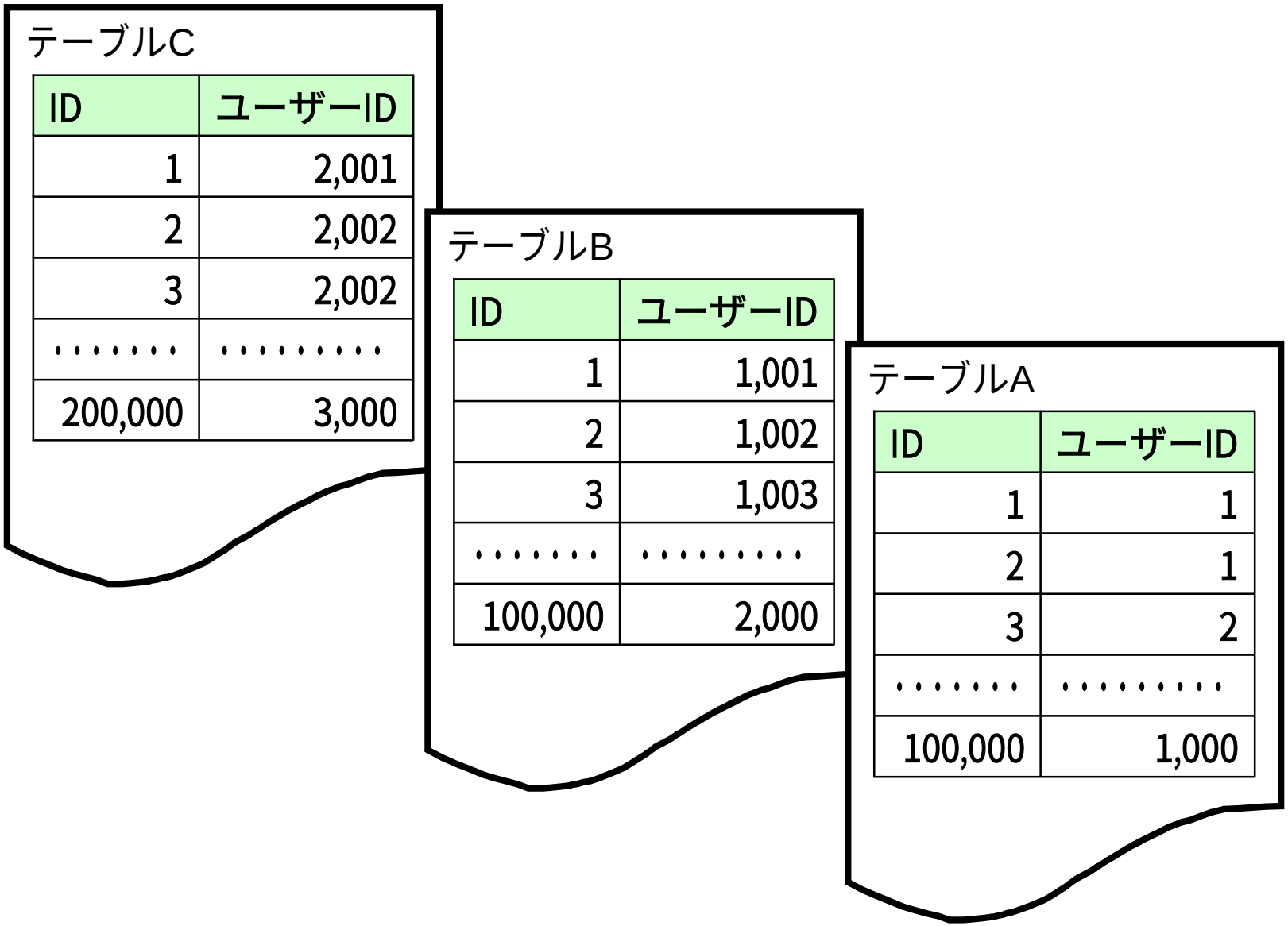
- レプリケーションを効率的に使うためには

ユーザーのIDを

キーにしてテーブルの分割を行い

レプリケーションで分散する

- レプリケーションを効率的に使うためには



テーブルC

ID	ユーザーID
1	2,001
2	2,002
3	2,002
・・・・・・・	・・・・・・・・・
200,000	3,000

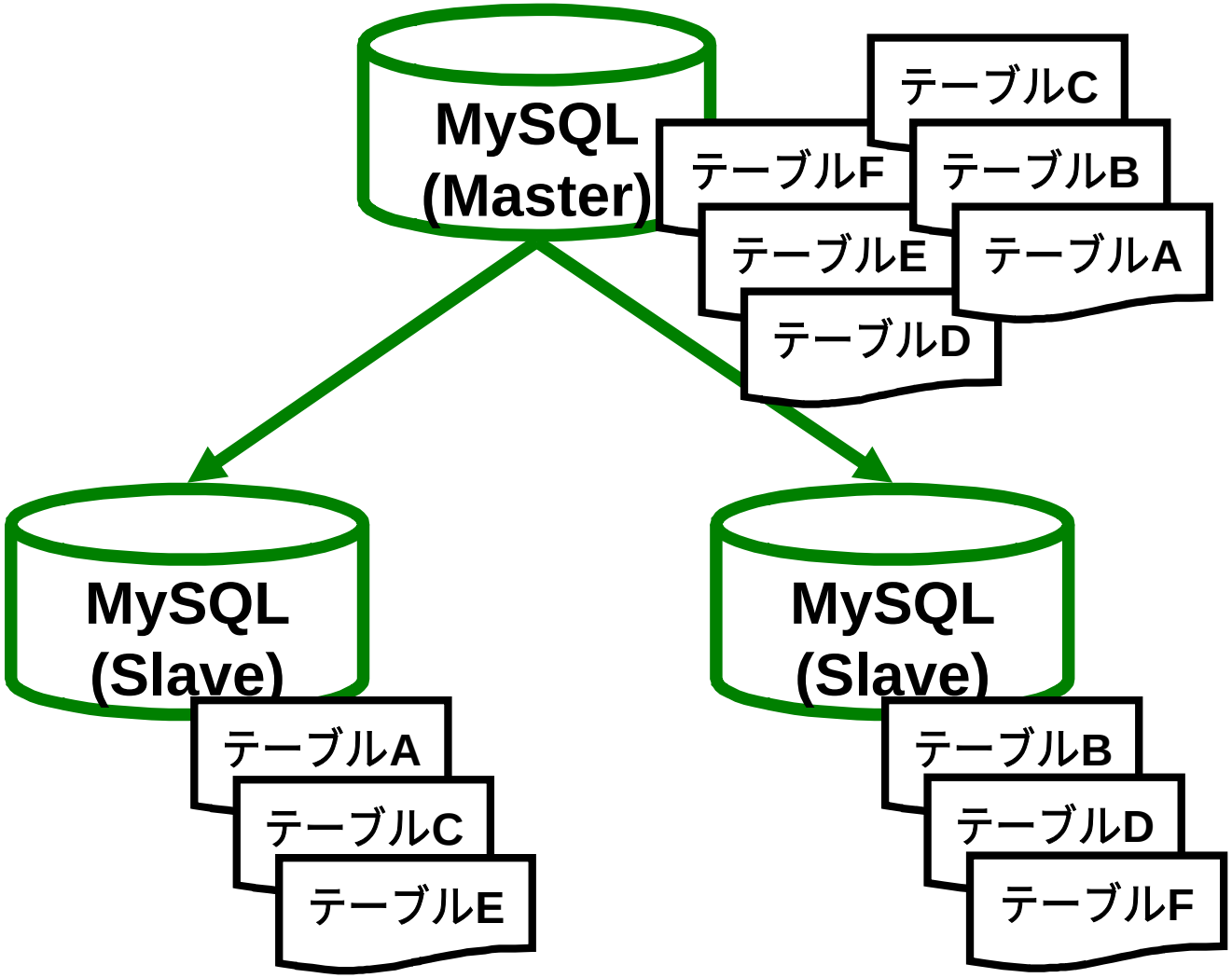
テーブルB

ID	ユーザーID
1	1,001
2	1,002
3	1,003
・・・・・・・	・・・・・・・・・
100,000	2,000

テーブルA

ID	ユーザーID
1	1
2	1
3	2
・・・・・・・	・・・・・・・・・
100,000	1,000

- レプリケーションを効率的に使うためには

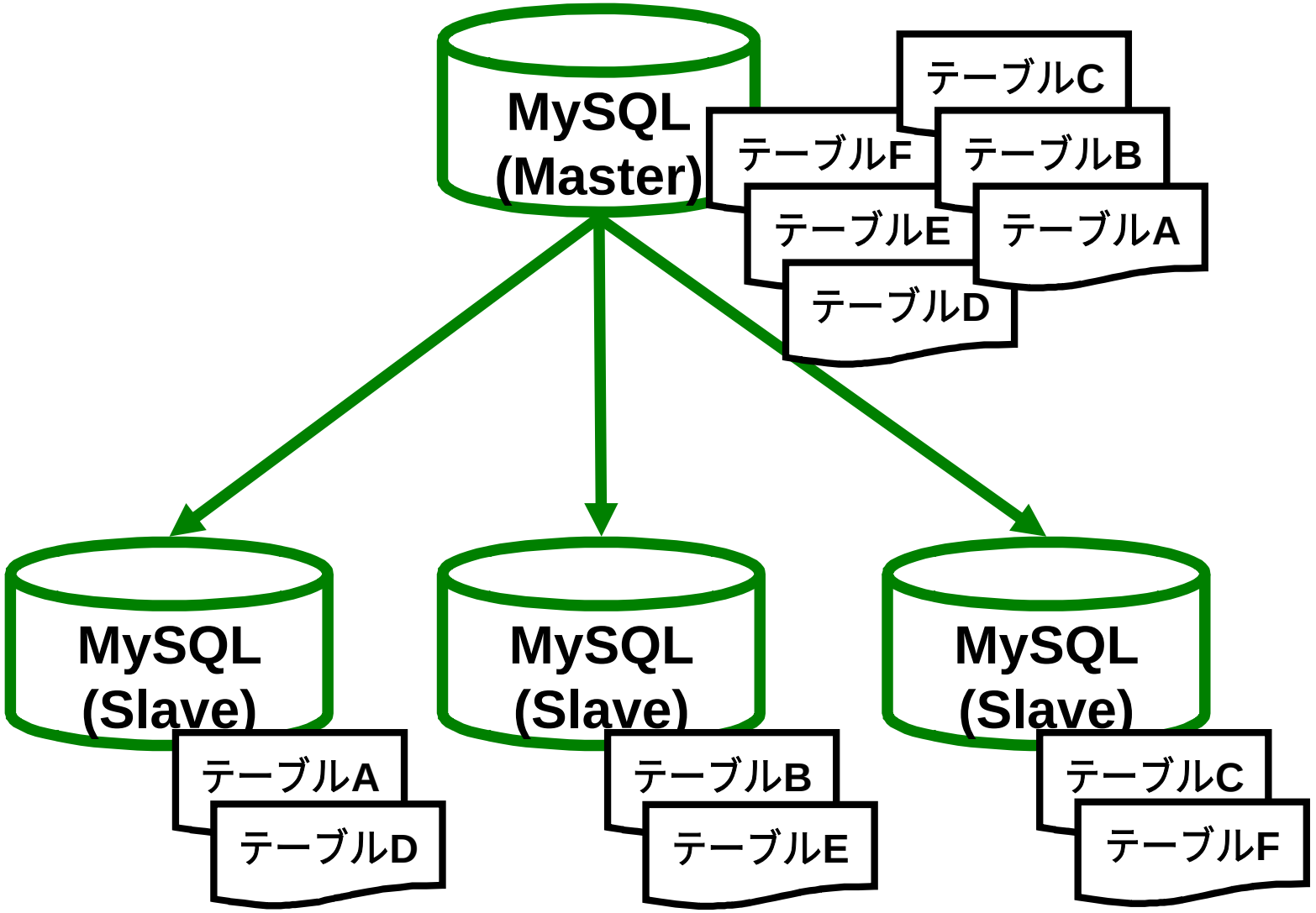


SlaveサーバのI/O負荷が増加したら

さらにテーブルを分散させることで

負荷を下げることが出来ます

- レプリケーションを効率的に使うためには



- レプリケーションを効率的に使うためには

導入からテーブルを分割する必要は

ありませんが、

設計段階からスケールアウト

出来る設計をしておくことで

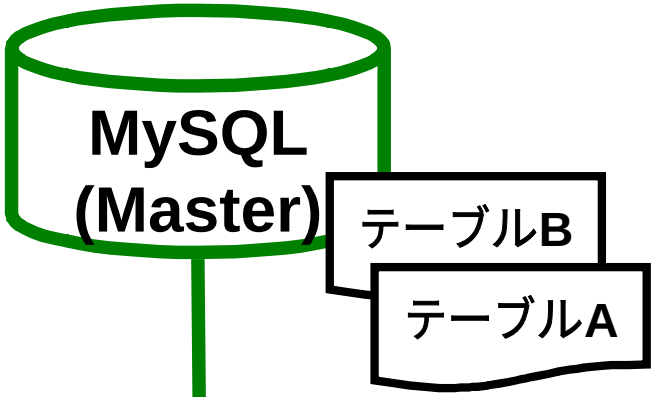
迅速に対応を行うことが出来ます

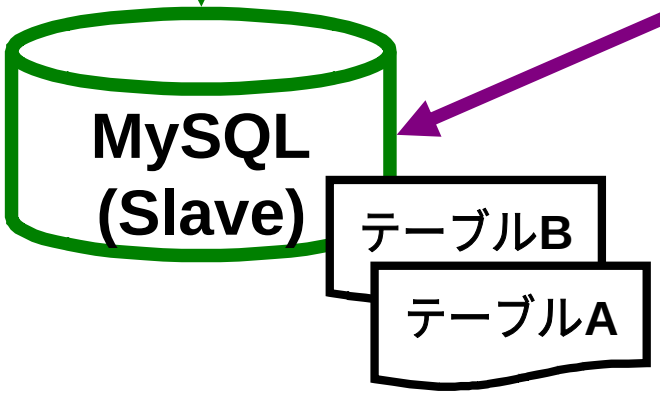
また、テーブルの分割以外にも

JOINの利用についても

考えておく必要があります

- レプリケーションを効率的に使うためには



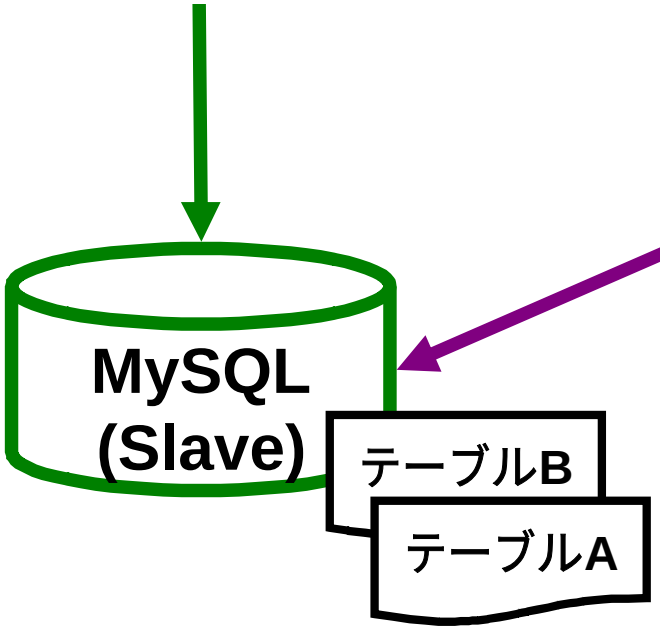



```
SELECT A.*
            ,B.*
FROM テーブルA A
        ,  テーブルB B
 WHERE A.ID = B.ID
   AND ・・・・
```

JOINを行ってしまう事で、

レプリケーションでの分散が困難に

なってしまいます




```
SELECT A.*
       ,B.*
FROM テーブルA A
     , テーブルB B
 WHERE A.ID = B.ID
   AND ・・・・・
```

JOINをしている場合、

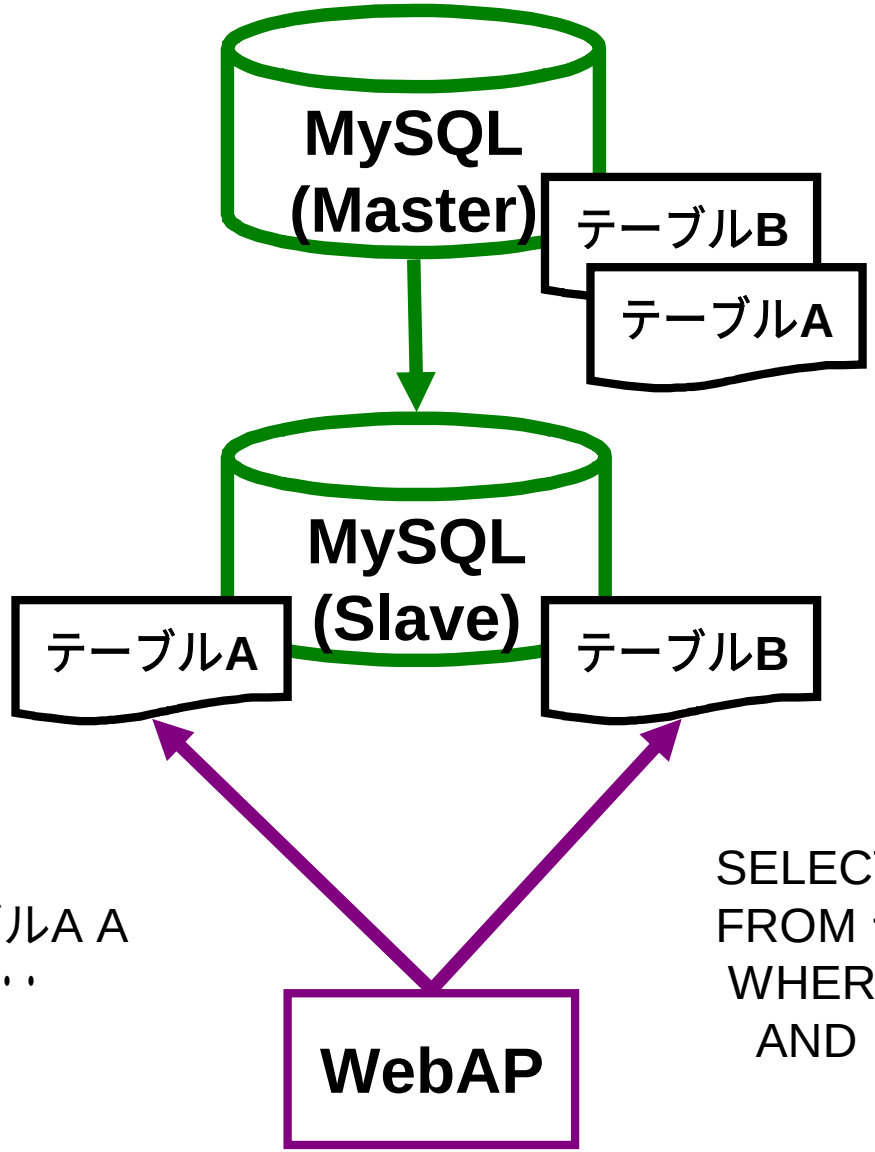
一度**SQL**を分離し、

サーバを分割する必要が

出来てしまうので対応までに

時間が掛かってしまいます

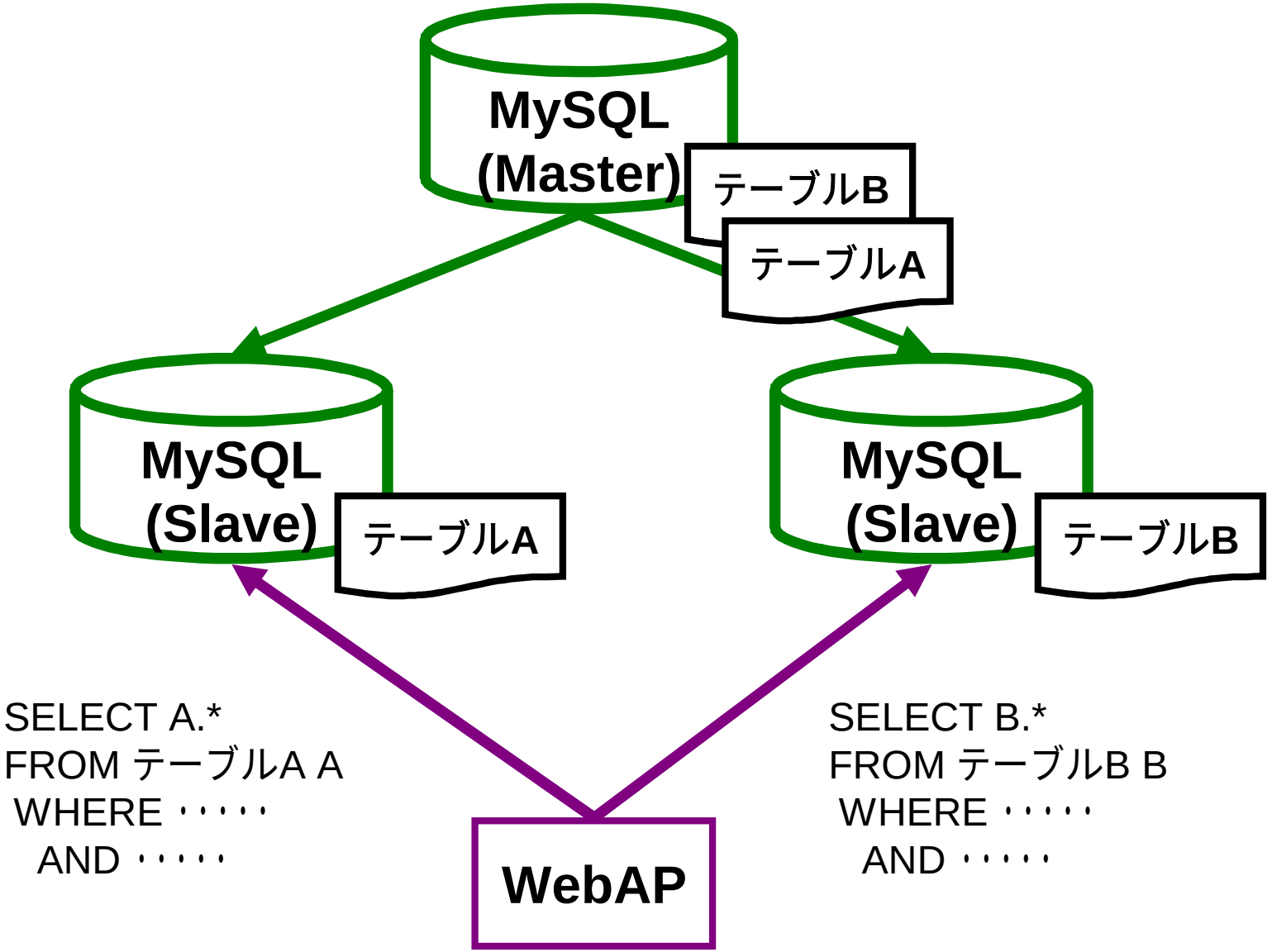
- レプリケーションを効率的に使うためには




```
SELECT A.*
FROM テーブルA A
 WHERE ・・・・・
   AND ・・・・・
```

```
SELECT B.*
FROM テーブルB B
 WHERE ・・・・・
   AND ・・・・・
```

- レプリケーションを効率的に使うためには



- レプリケーションを効率的に使うためには

データ量が多くなるテーブル

同士での**JOIN**は

避けたほうがよいでしょう

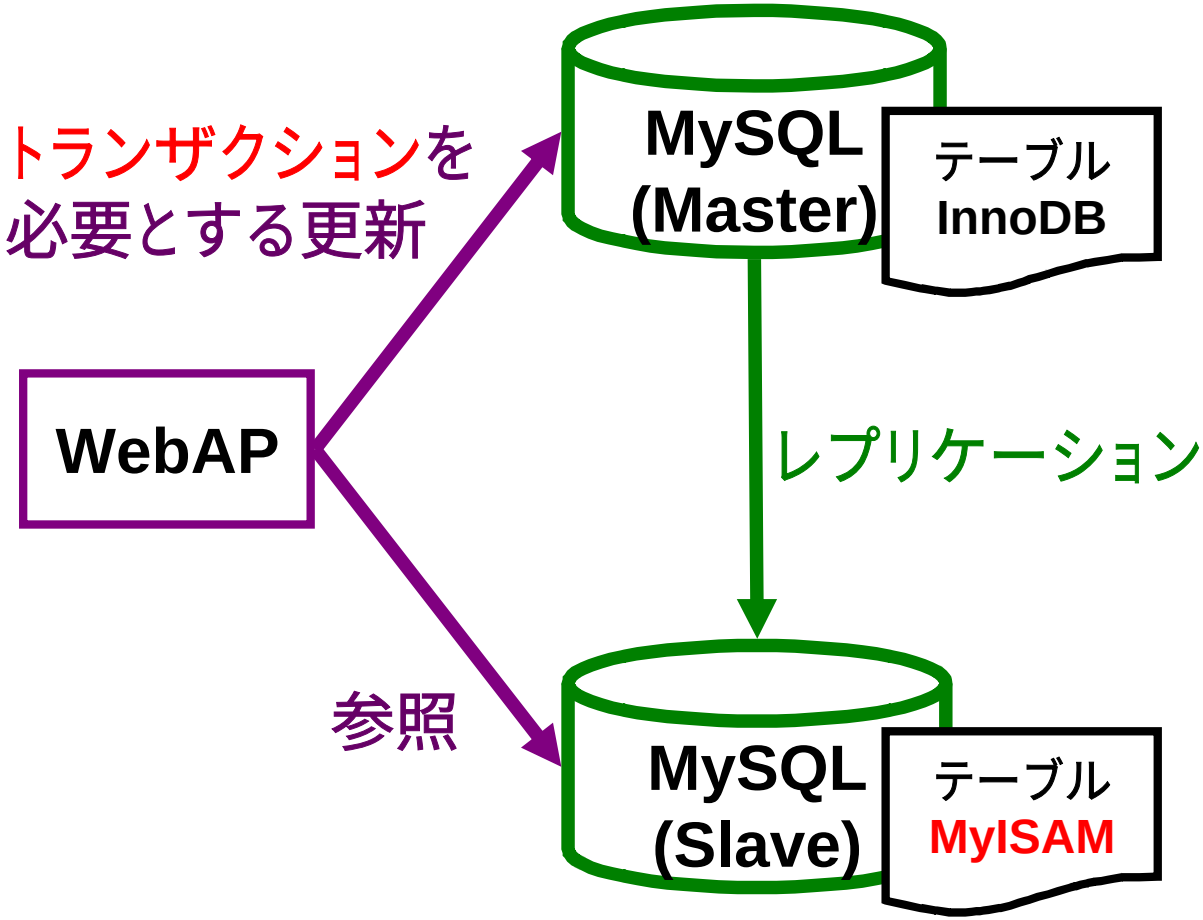
1-4.ストレージエンジンとレプリケーション

ストレージエンジンと

レプリケーションを用いた

負荷軽減について説明します

- MasterとSlaveでストレージエンジンを変える



- MasterとSlaveでストレージエンジンを変える

SELECTの発行回数が多い場合は

Slaveサーバのストレージエンジンを

Memoryエンジンにすることで

パフォーマンスを出すことが出来ます

- MasterとSlaveでストレージエンジンを変える

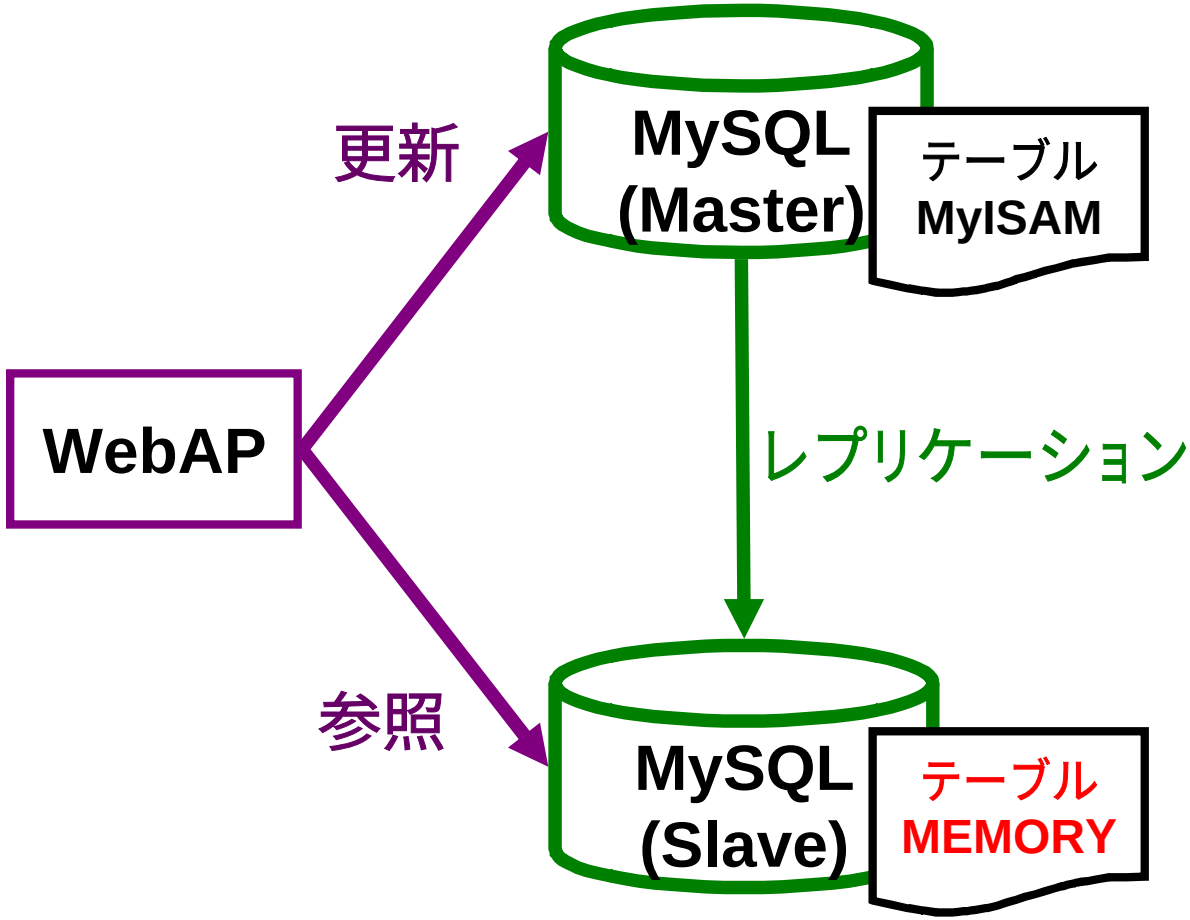
Memoryエンジンはデータ、インデックスを

メモリー上で管理するので

書き込み、読み込み共に

最も早いストレージエンジンと言えるでしょう

- MasterとSlaveでストレージエンジンを変える



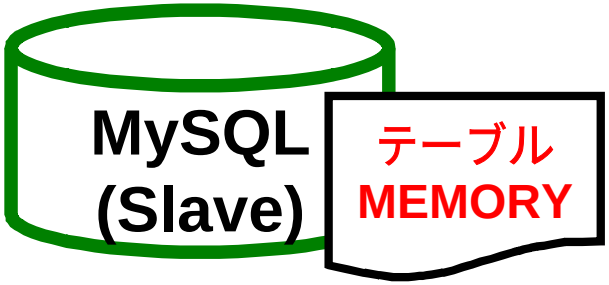
ただし、**Memory**エンジンは

MySQLの再起動で

データが消えてしまうので

再起動の方法や障害時のデータ復旧

など予め考えておく必要があります



- MasterとSlaveでストレージエンジンを変える

Masterサーバには不要な

履歴などのテーブルが

ある場合は

- MasterとSlaveでストレージエンジンを変える

Masterサーバのストレージエンジンを

BLACKHOLEエンジンにすることで

Masterサーバの

I/O負荷を軽減することが出来ます

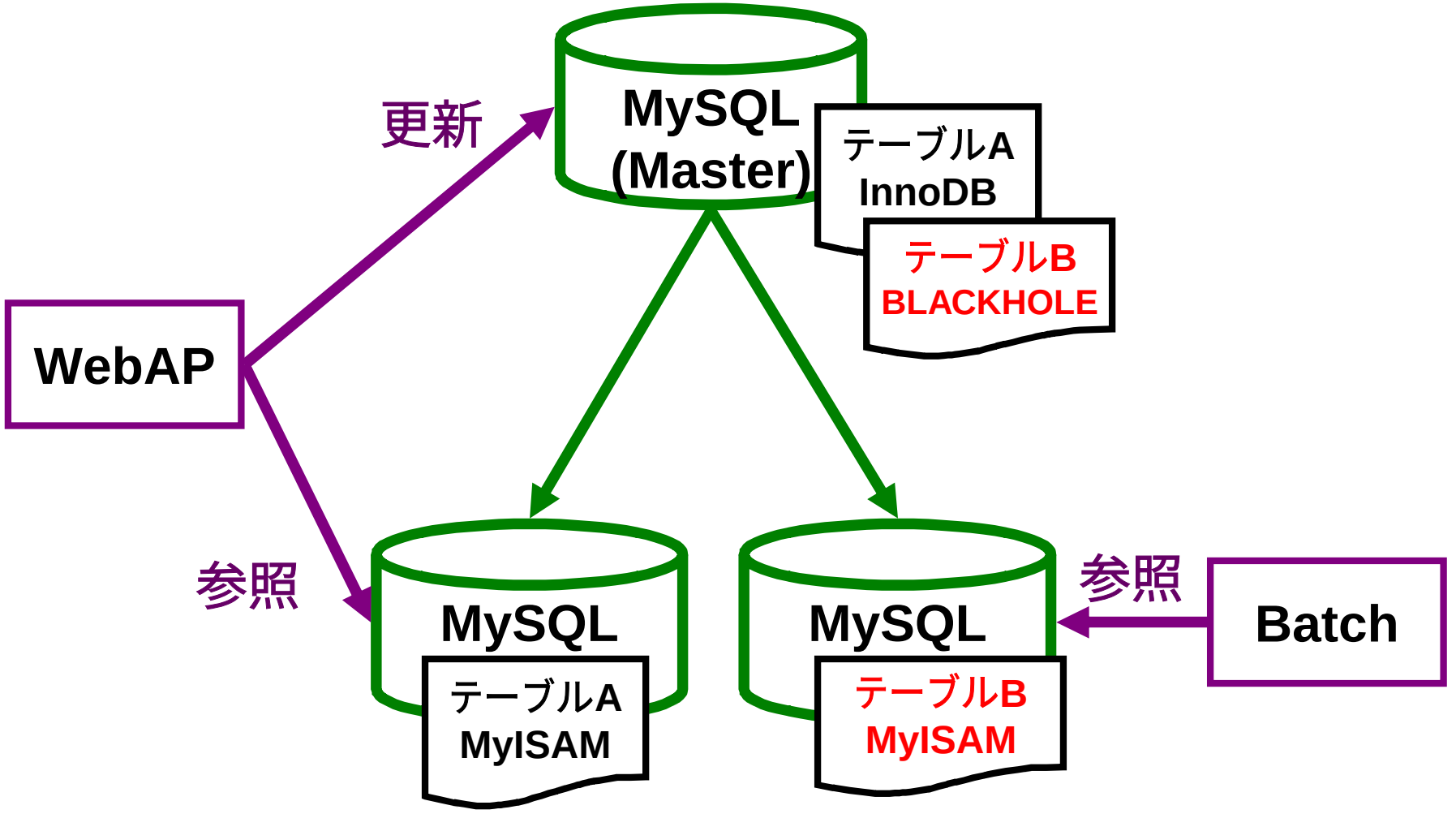
- MasterとSlaveでストレージエンジンを変える

BLACKHOLEエンジンは

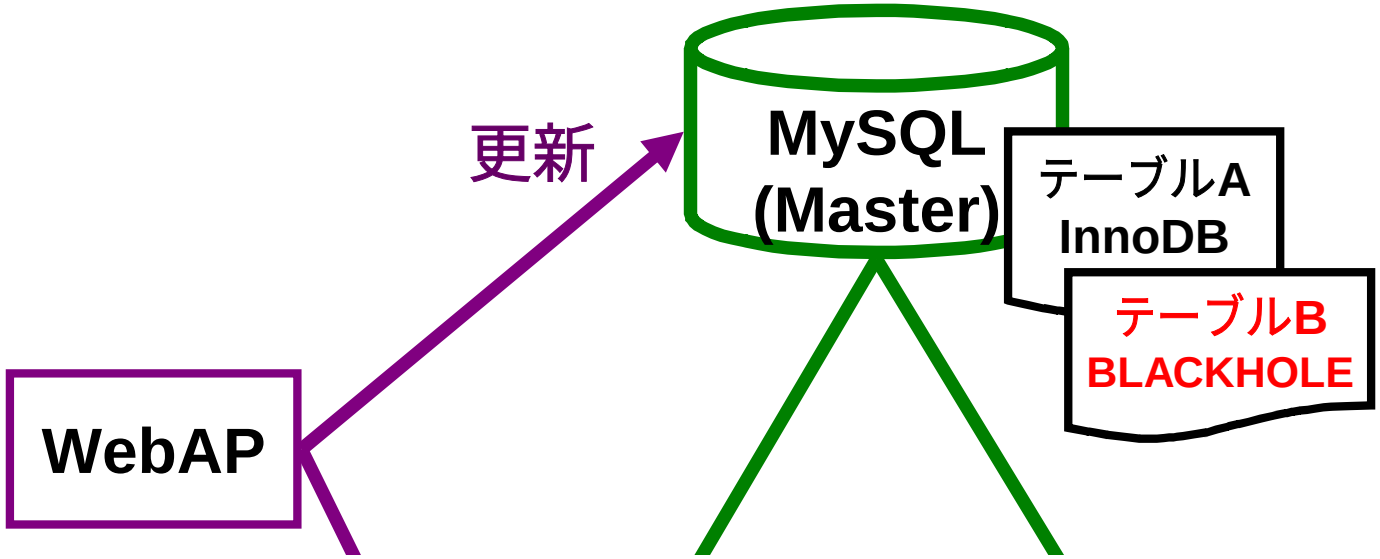
データを保存･更新しない

ストレージエンジンになります

- MasterとSlaveでストレージエンジンを変える



- MasterとSlaveでストレージエンジンを変える



BLACKHOLEストレージエンジンを

使用することで、**Master**の**Disk**容量を

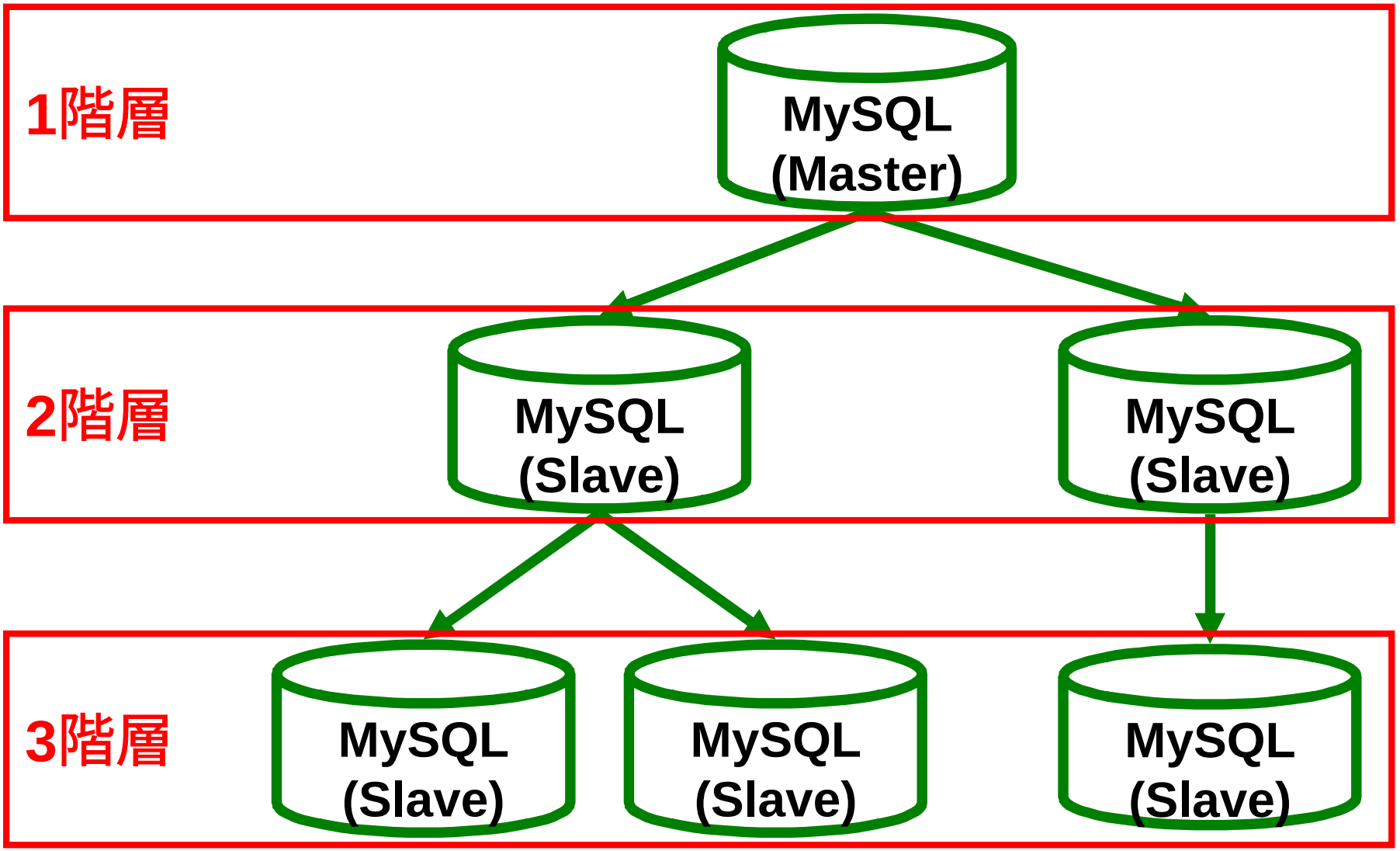
確保すると共に**I/O**を減らすことが出来ます

1-5.多階層のレプリケーション

最後に多層型のレプリケーション

について説明します

- 多階層のレプリケーション

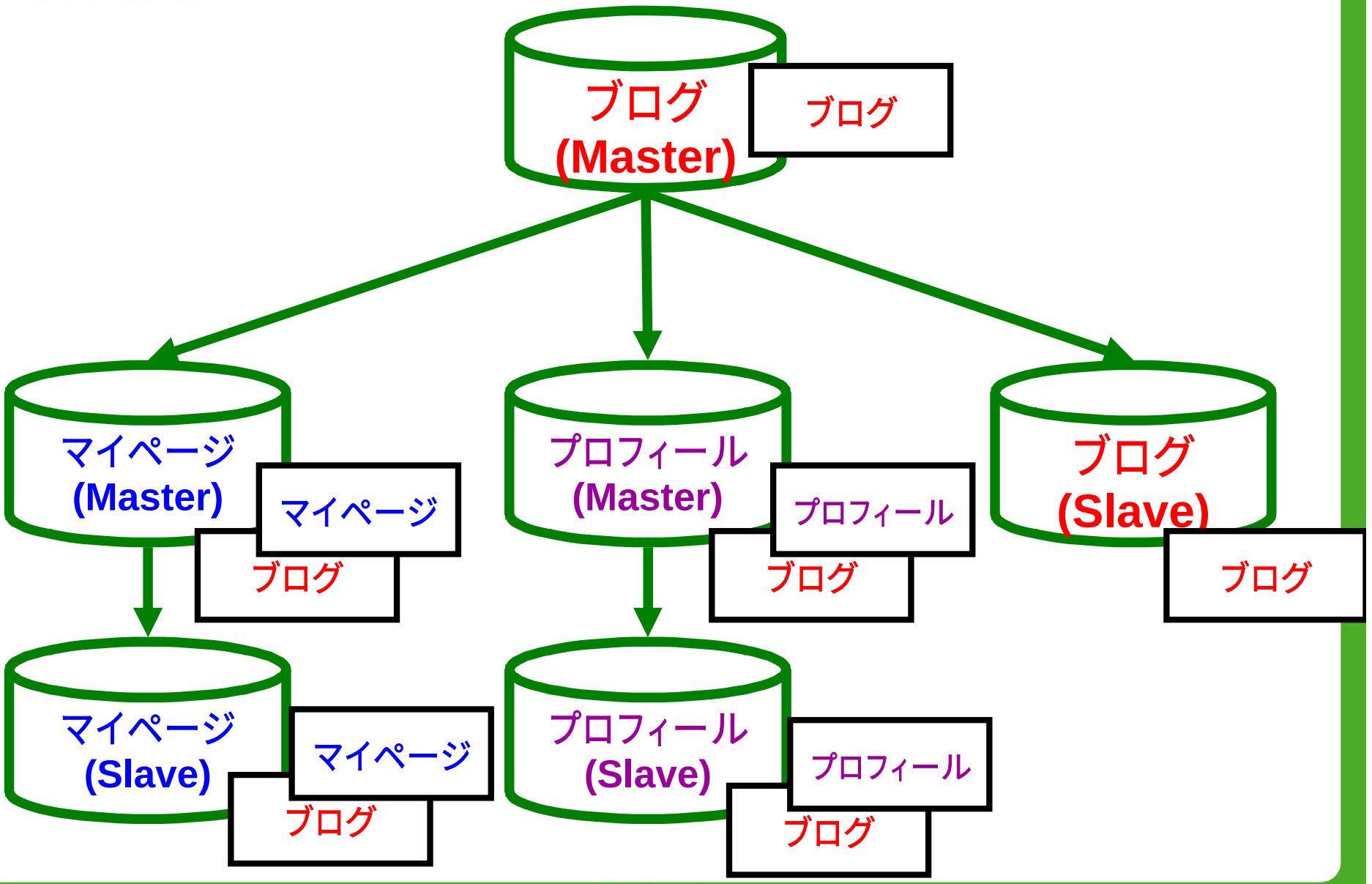


- 多階層のレプリケーション

Amebaでは次のようなケースで

使用しています

- 多階層のレプリケーションの利用例

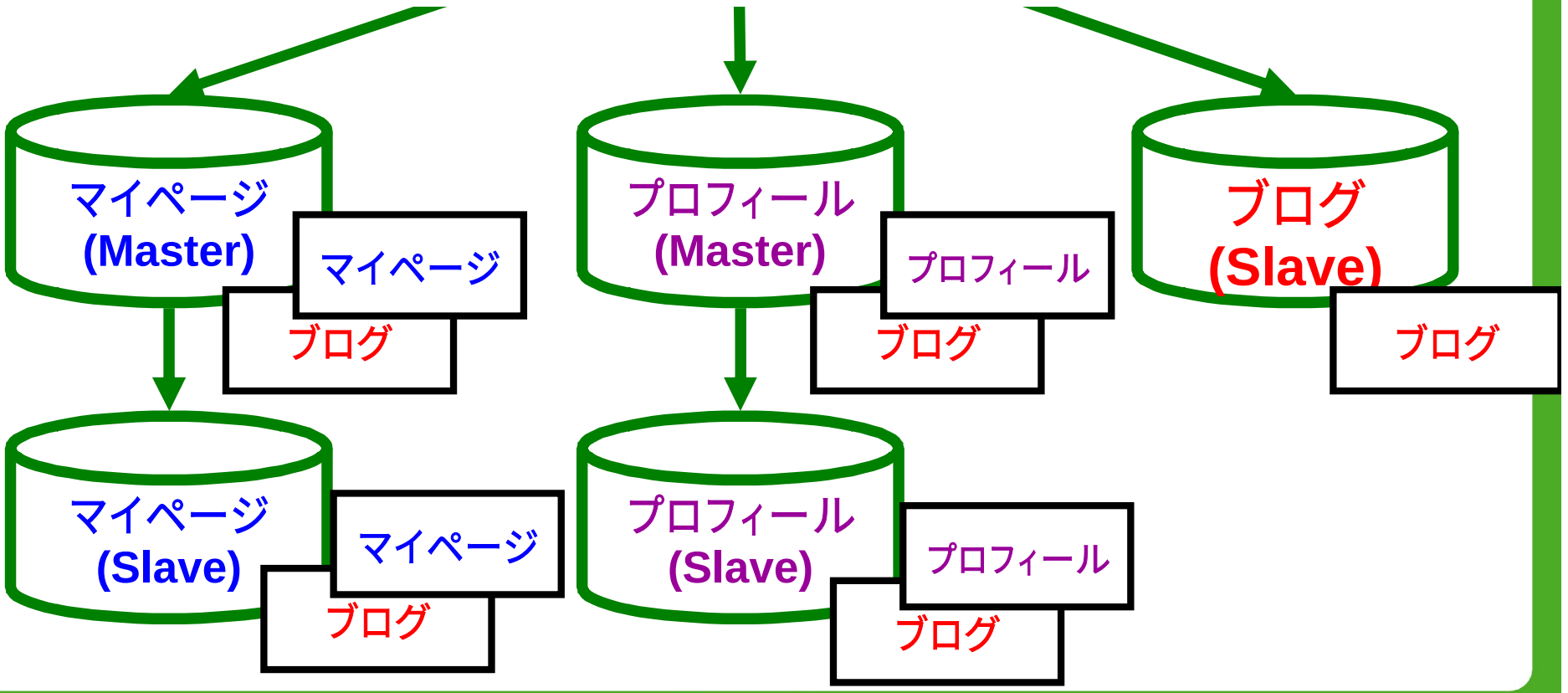


- 多階層のレプリケーションの利用例

この様にあるサービスのデータを

他のサービスで使用する場合に

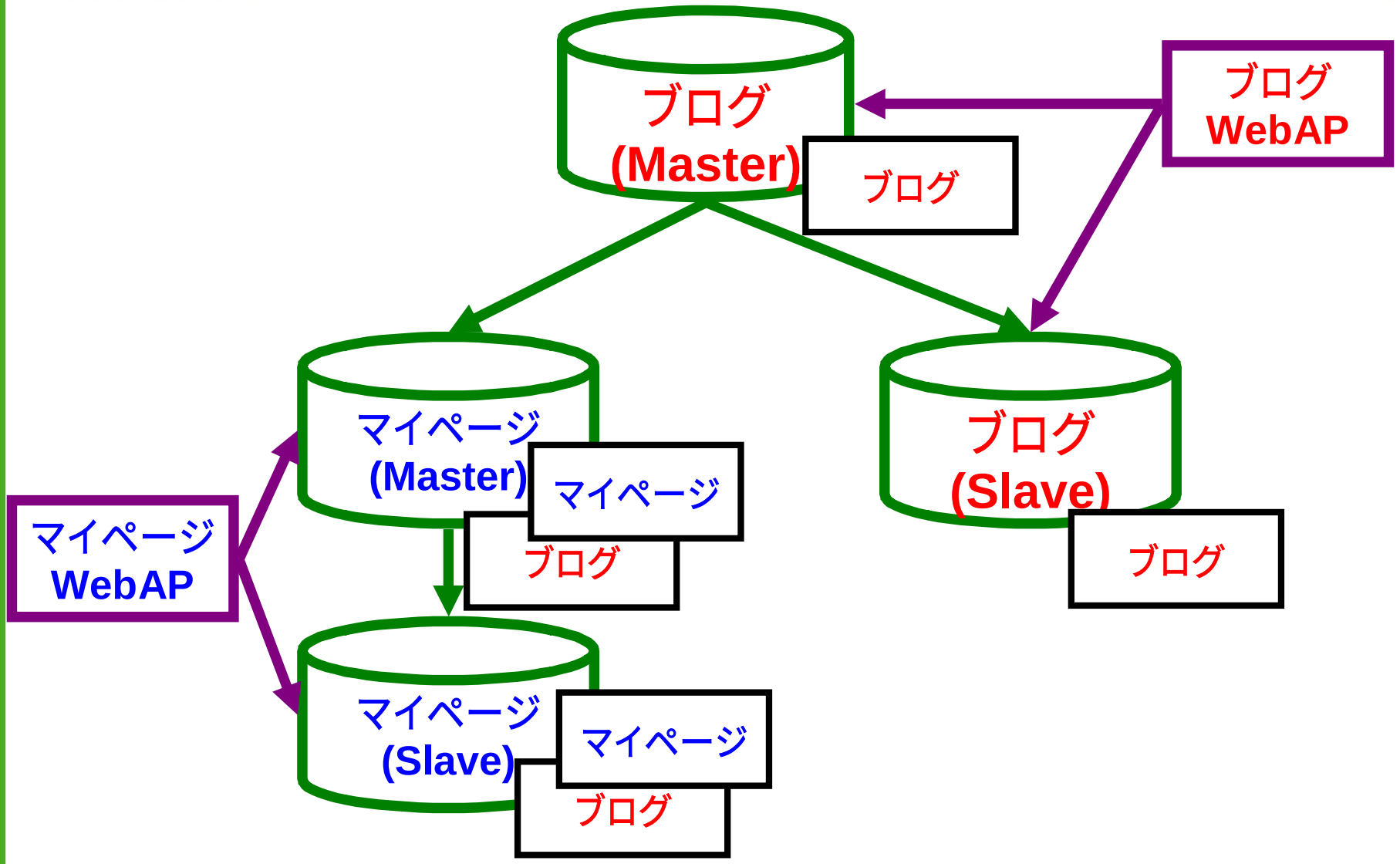
便利です



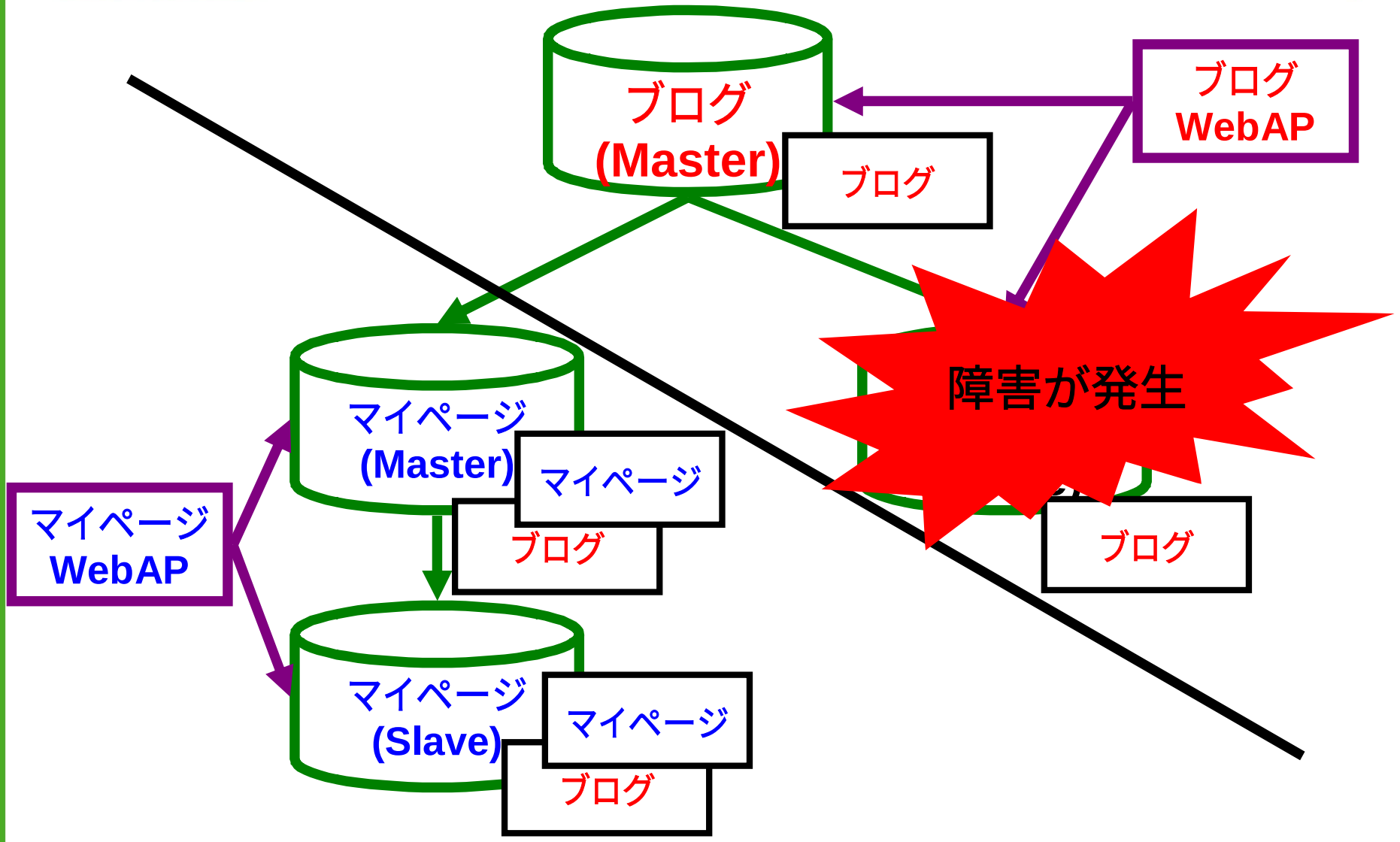
- 多階層のレプリケーションの利用例

また、耐障害性にも優れています

- 多階層のレプリケーションの利用例

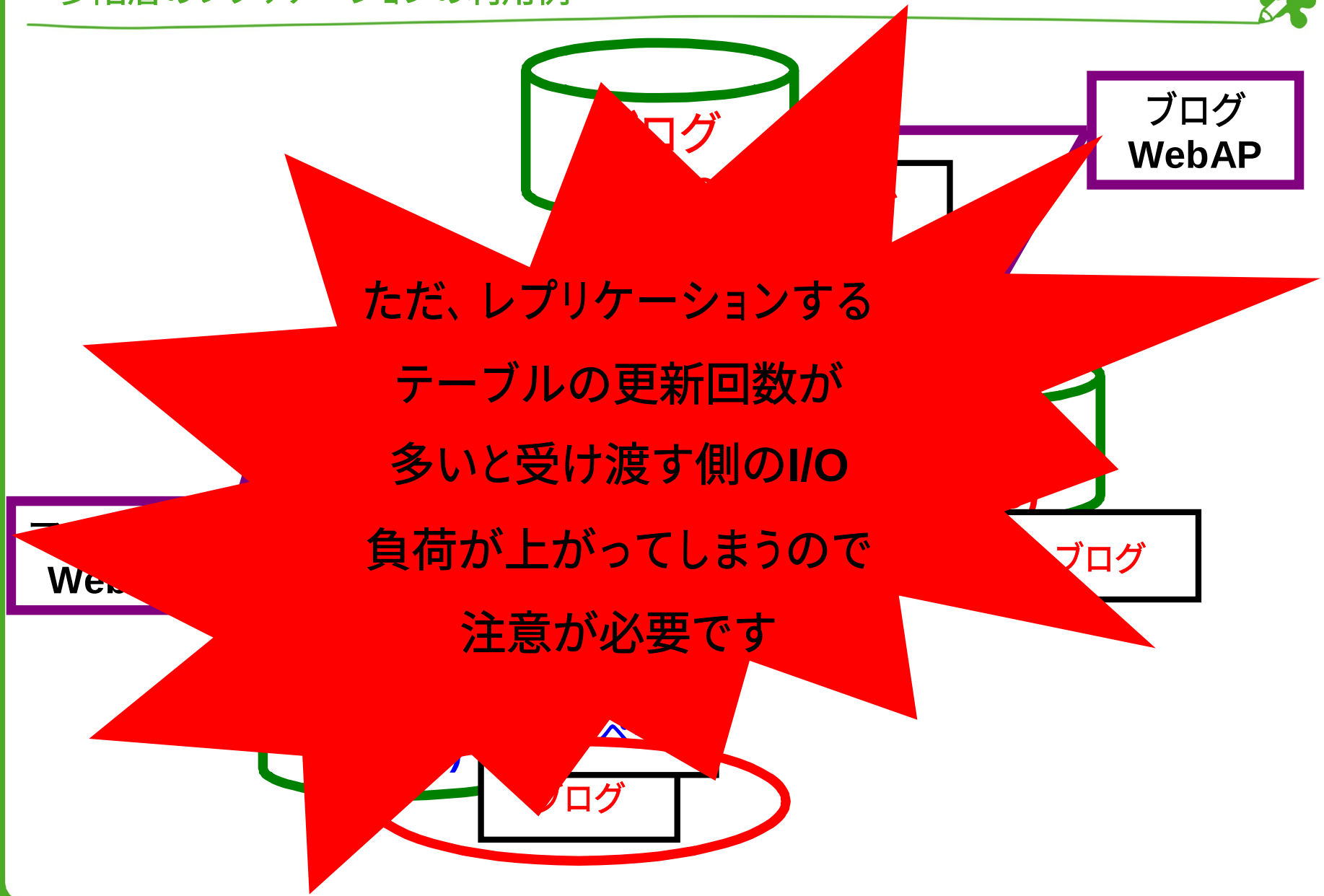


- 多階層のレプリケーションの利用例



影響を受けずにサービスを継続することが可能

- 多階層のレプリケーションの利用例



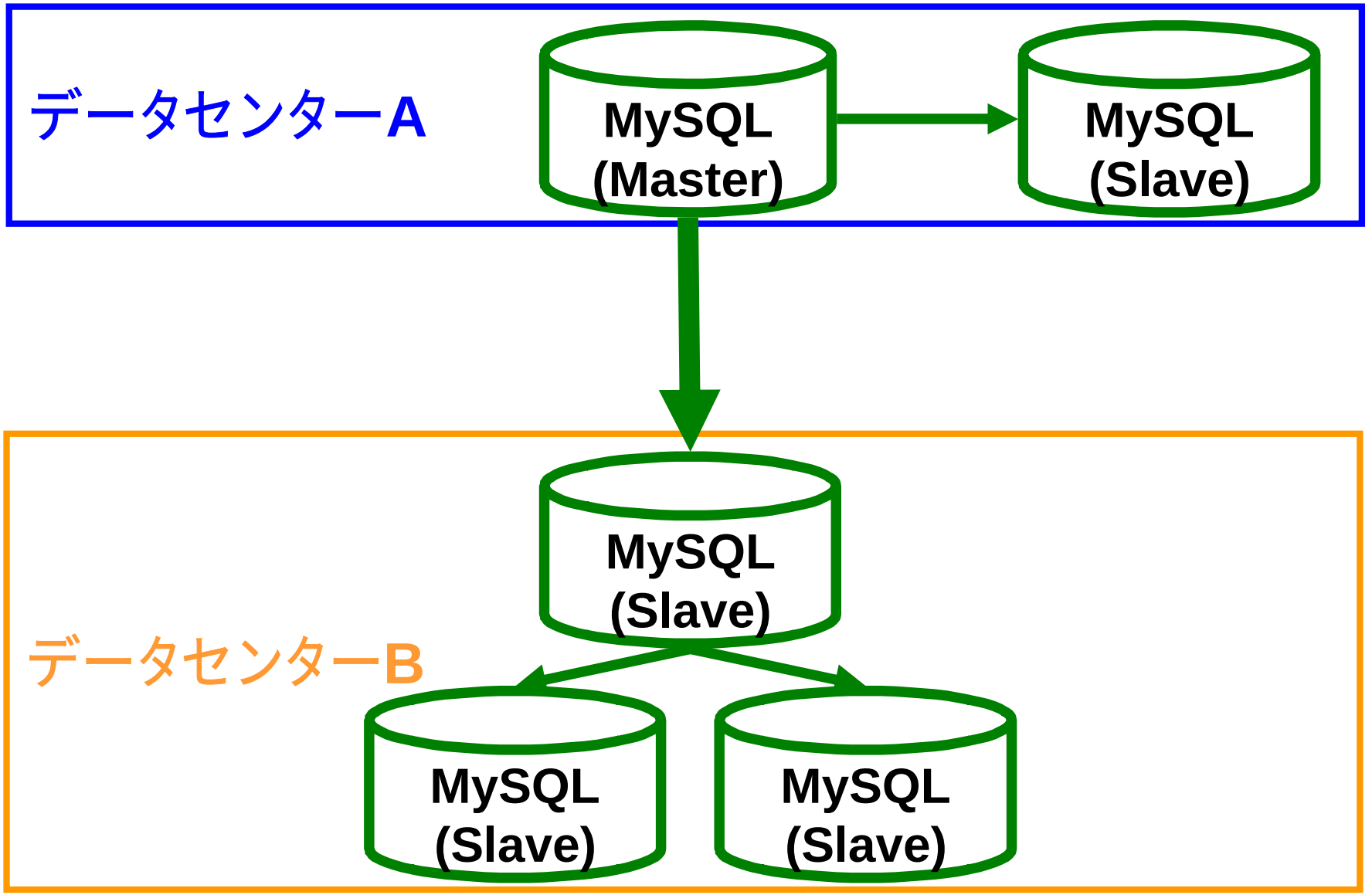
ただ、レプリケーションする
テーブルの更新回数が
多いと受け渡す側のI/O
負荷が上がってしまうので
注意が必要です

- 多階層のレプリケーションの利用例

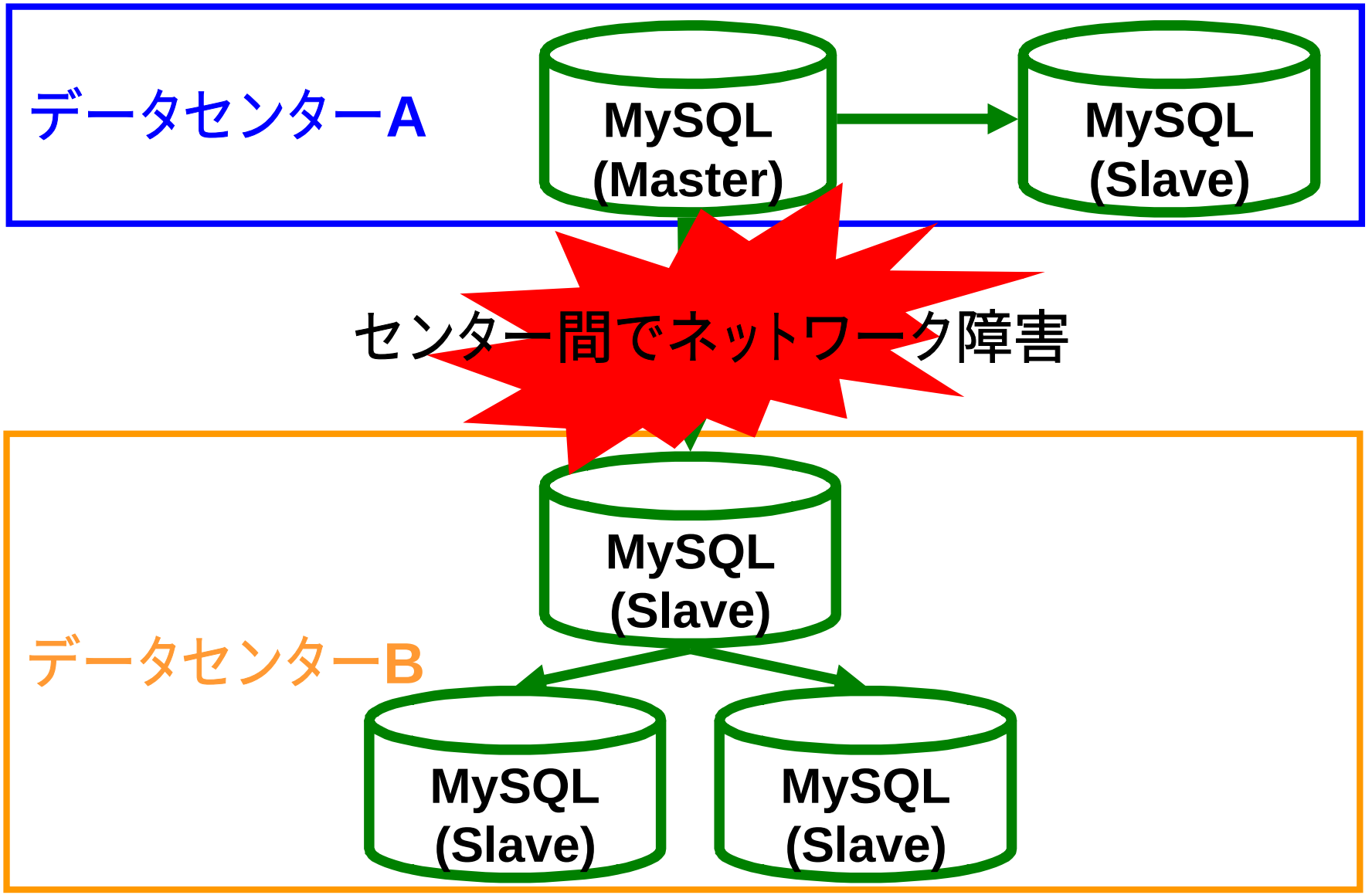
また、次のようなケースでも

使用しています

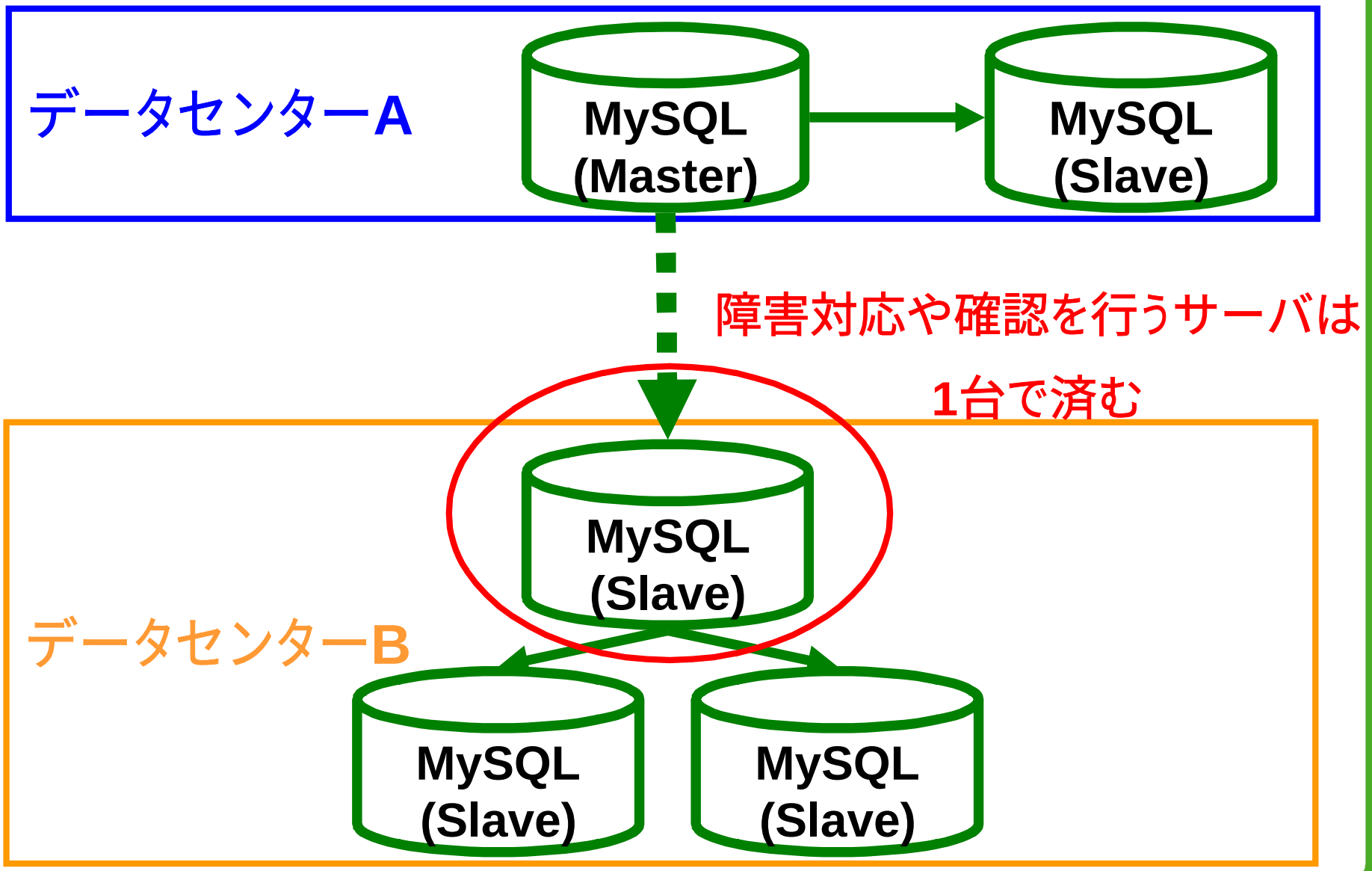
- 多階層のレプリケーションの利用例



- 多階層のレプリケーションの利用例



- 多階層のレプリケーションの利用例



MySQLは

レプリケーションを使用することで

負荷分散を簡単に行うことができ、

また、障害に強い構成を

組むこともできます

2.MySQLの運用

MySQLサーバの運用で重要な

バックアップ

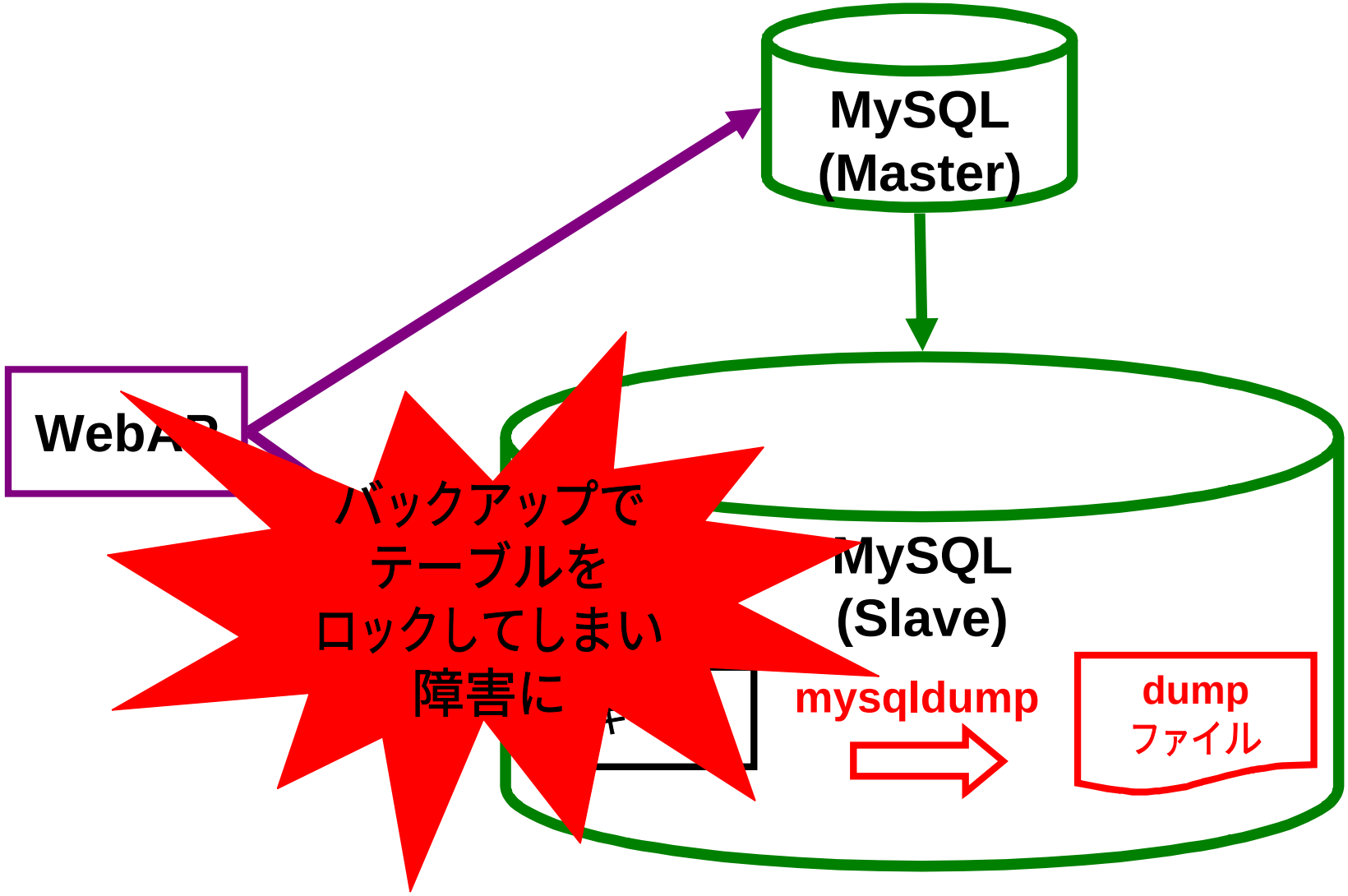
障害監視

性能監視

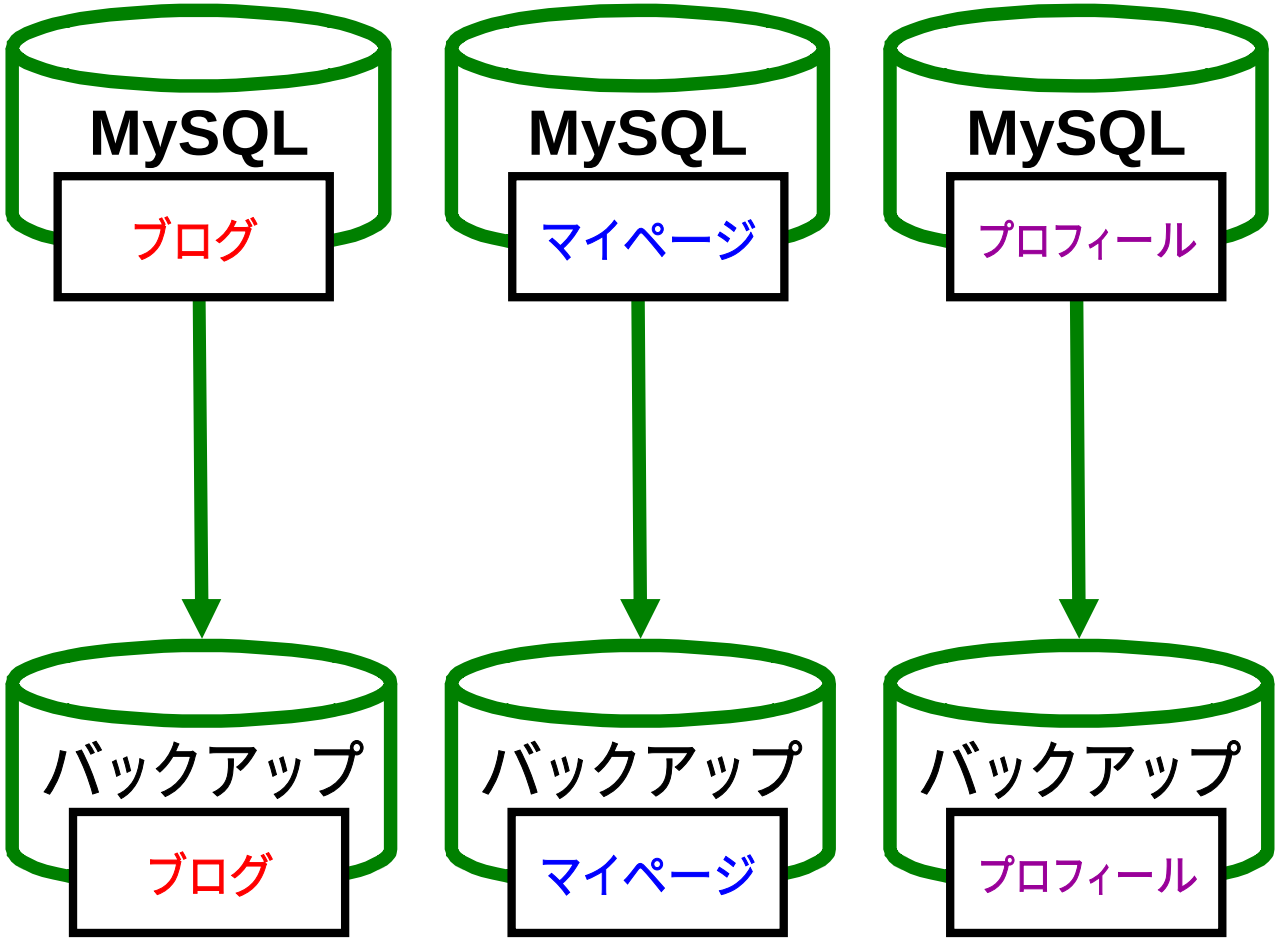
について説明します

2-1. バックアップ

- バックアップ


MySQL
(Master)
WebA
バックアップで
テーブルを
ロックしてしまい
障害に
MySQL
(Slave)
mysqldump
dump
ファイル

- バックアップ

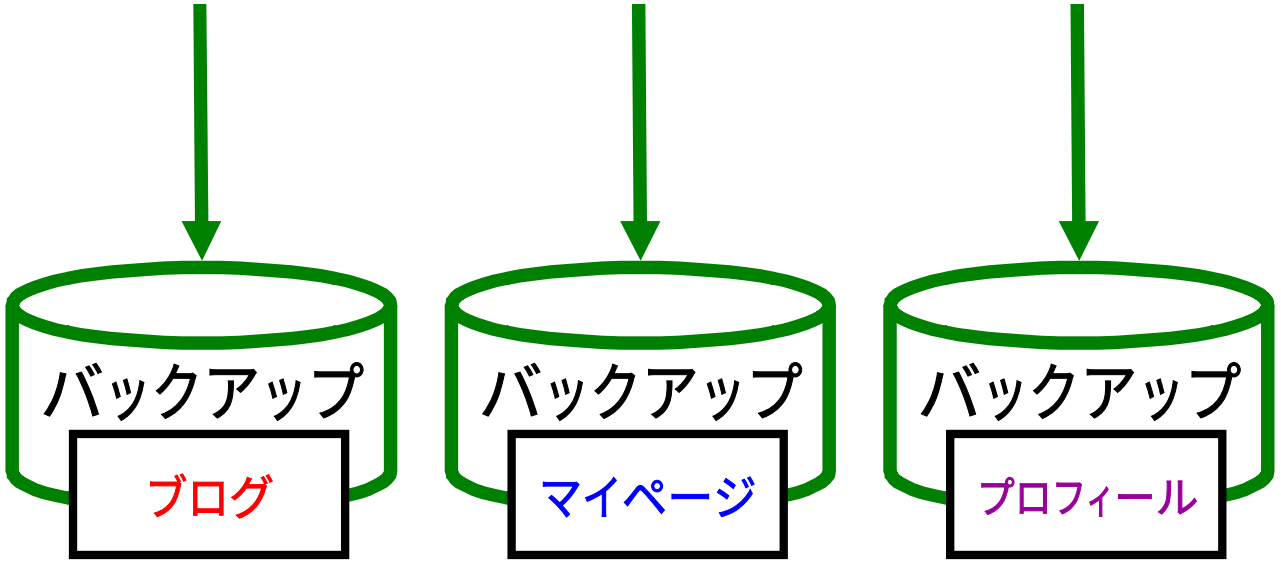

MySQL
ブログ
MySQL
マイページ
MySQL
プロフィール
バックアップ
ブログ
バックアップ
マイページ
バックアップ
プロフィール

- バックアップ

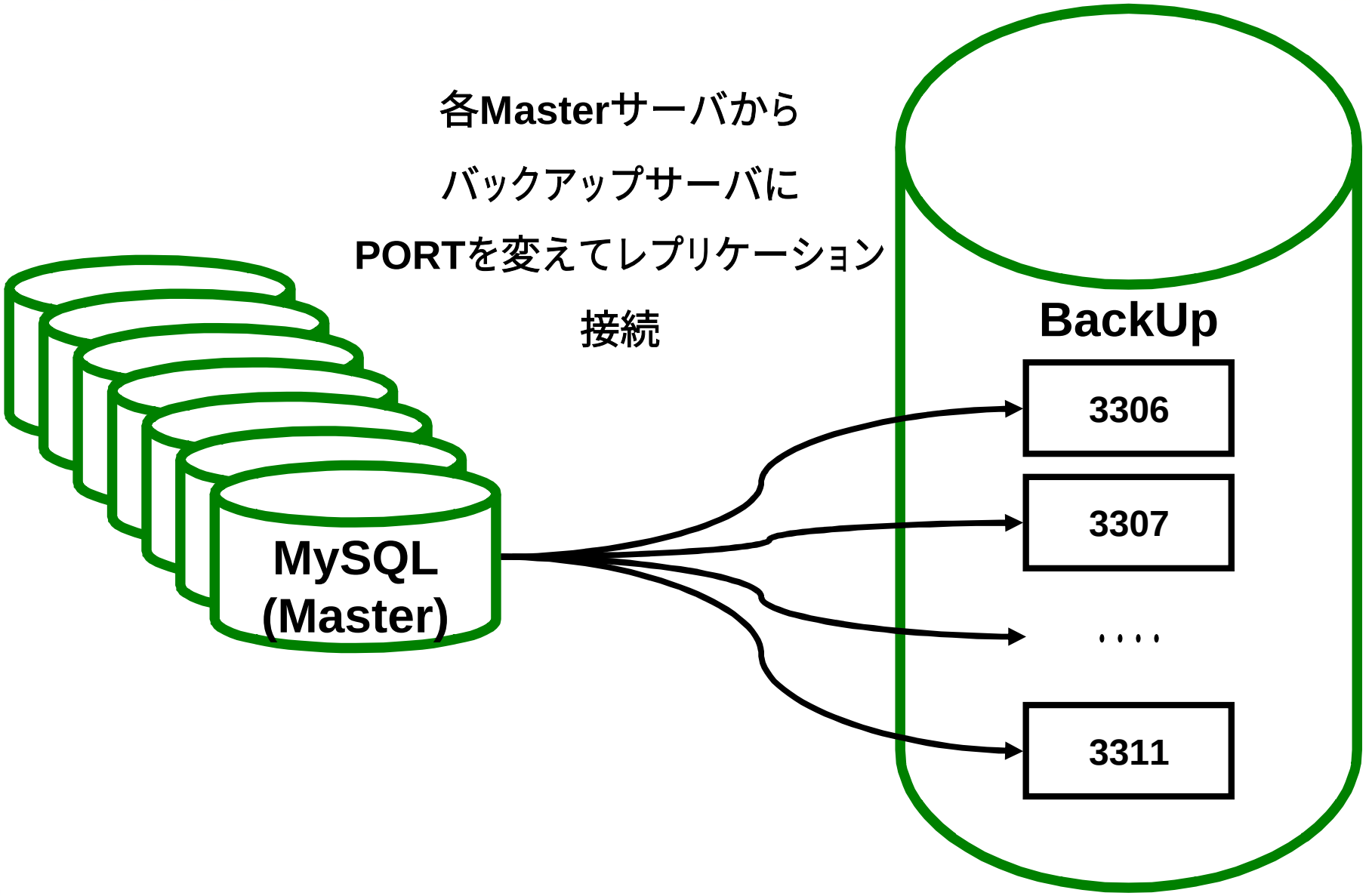
サービス毎にバックアップサーバを

用意してはコストがかかってしまうし、

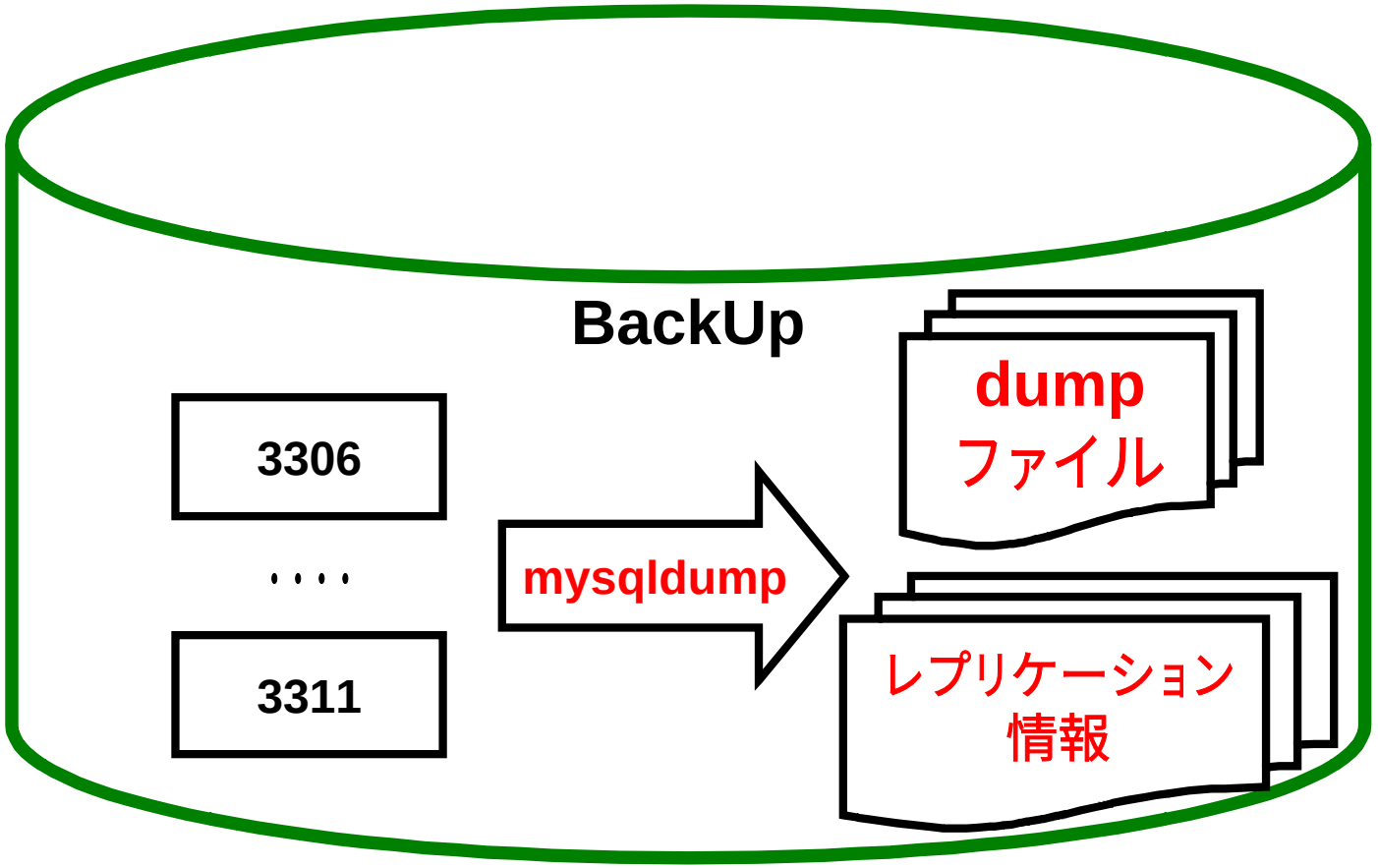
障害率も上がってしまうので



- バックアップ例


各Masterサーバから
バックアップサーバに
PORTを変えてレプリケーション
接続
MySQL
(Master)
BackUp
3306
3307
....
3311

「mysqldump」コマンドを利用したバックアップ方式



- 「mysqldump」を利用したバックアップ例

Amebaでは運用当初

「mysqldump」でバックアップを

行っていたのですが

データ量の増加と

サービスの増加で

バックアップが完了するまでに

20時間近くかかってしまう結果に

そこで

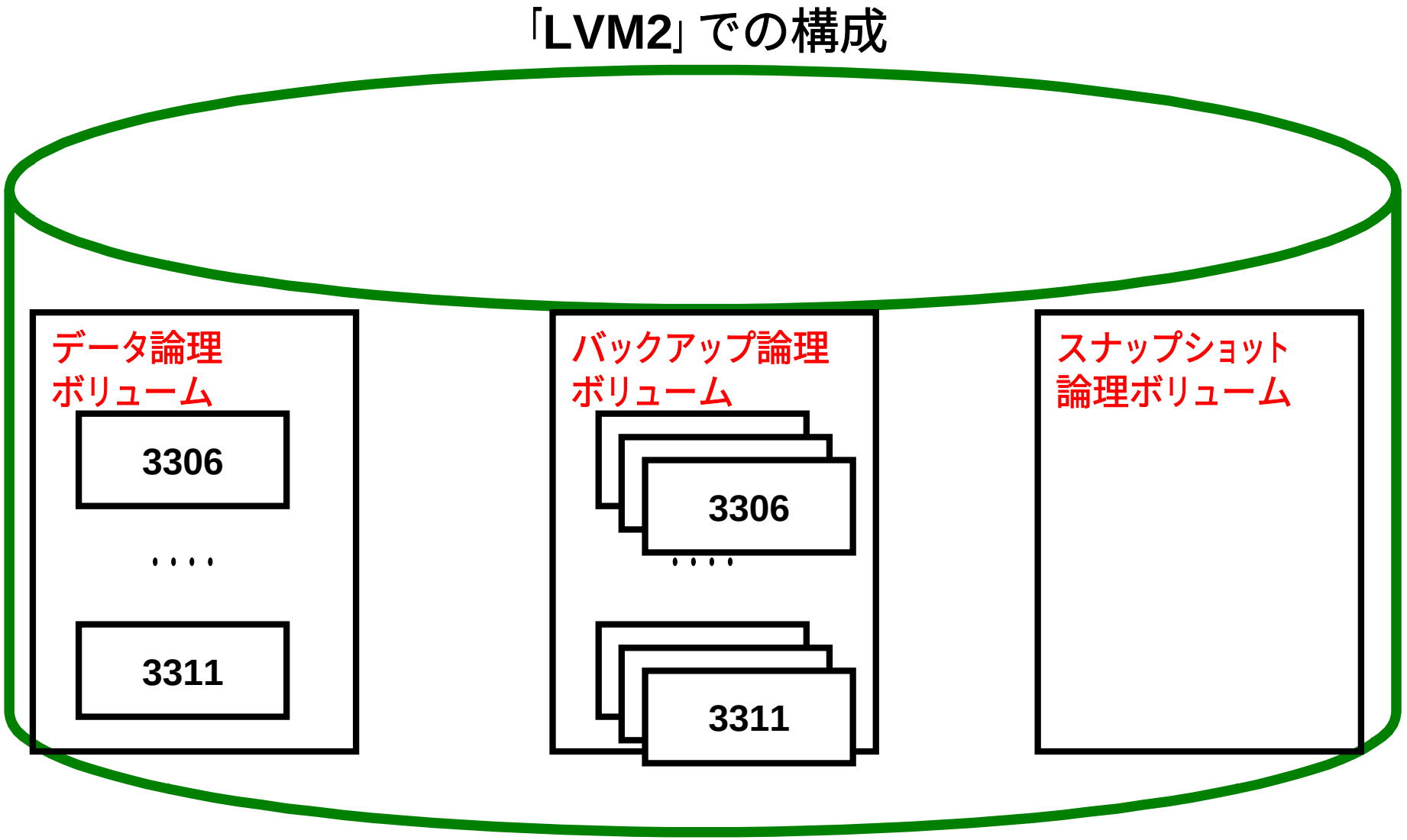
「LVM2(論理ボリュームマネージャー)」

のスナップショット機能を

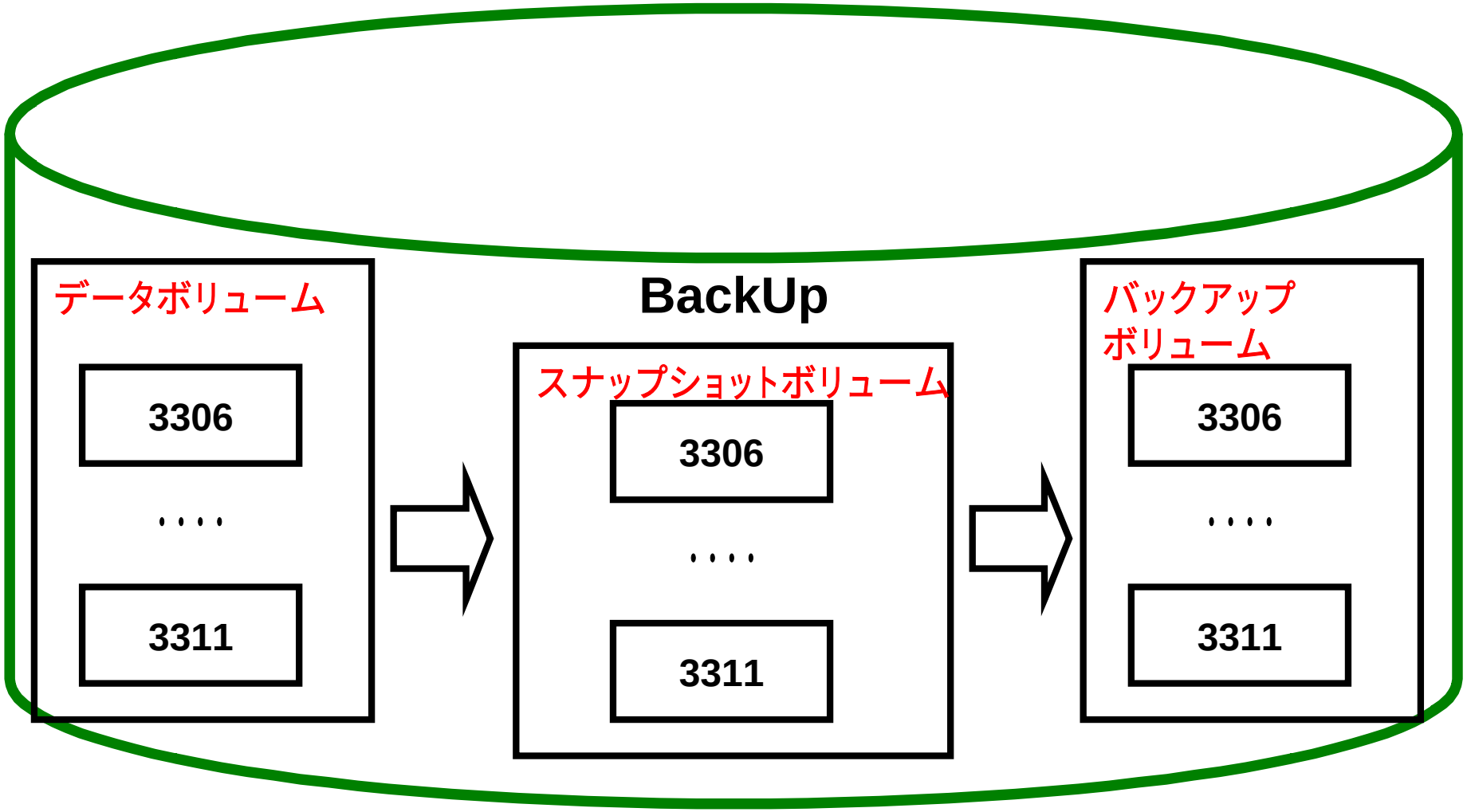
利用したバックアップ方式に

変更しました

- 「LVM2」を利用したバックアップ例



- 「LVM2」を利用したバックアップ例



- 「LVM2」を利用したバックアップ例

LVM2スナップショットを

利用することで

従来のmysqldumpの

1/6程度でバックアップが完了

参考までに「LVM2」での
バックアップの流れを確認
してみましょう

- 「LVM2」を利用したバックアップ例：前提

✓ スレーブであること

✓ データファイルが格納されている，
LVM論理ボリュームを対象とする

✓ ファイルシステムは ext3

手順1） MySQLを停止（面倒なので）
※ LOCK TABLES でもOK

手順2） sync（確実に）

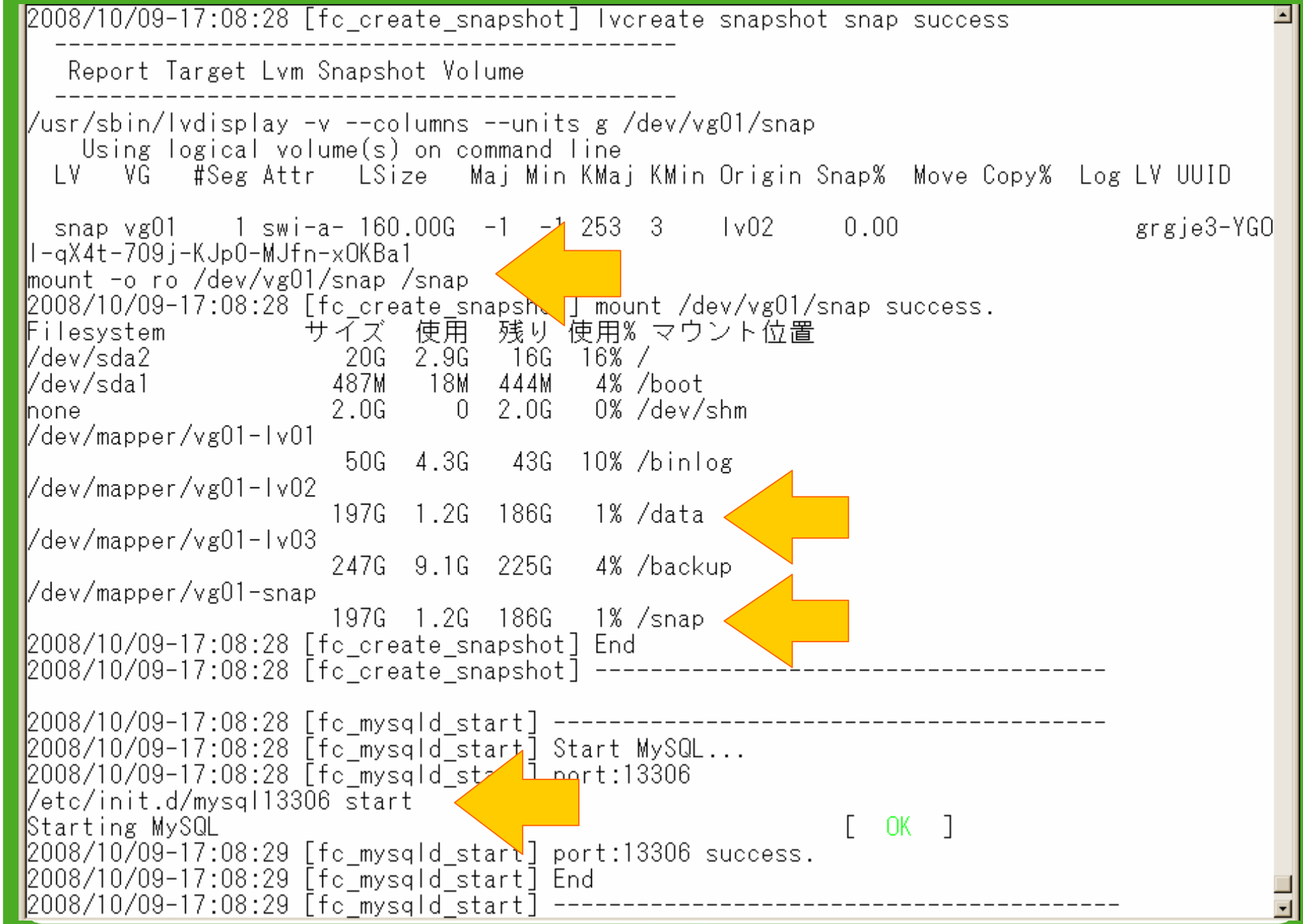
手順3） 対象ボリュームの
スナップショット作成
>> lvcreate --snapshot …

```
2008/10/09-17:08:26 [fc_stop_slave] ----------------------------------------
2008/10/09-17:08:26 [fc_stop_slave] Stop Slave...
2008/10/09-17:08:26 [fc_stop_slave] port:13306
/usr/local/mysql13306/bin/mysql -uroot --socket=/tmp/mysql13306.sock -e "stop slave"
2008/10/09-17:08:26 [fc_stop_slave] port:13306 success.
2008/10/09-17:08:26 [fc_stop_slave] End
2008/10/09-17:08:26 [fc_stop_slave] ----------------------------------------

2008/10/09-17:08:26 [fc_mysqld_stop] ----------------------------------------
2008/10/09-17:08:26 [fc_mysqld_stop] Stop MySQL...
2008/10/09-17:08:26 [fc_mysqld_stop] port:13306
/etc/init.d/mysql13306 stop
Shutting down MySQL                                        [  OK  ]
2008/10/09-17:08:27 [fc_mysqld_stop] port:13306 success.
2008/10/09-17:08:27 [fc_mysqld_stop] End
2008/10/09-17:08:27 [fc_mysqld_stop] ----------------------------------------

2008/10/09-17:08:27 [fc_create_snapshot] ------------------------------------
2008/10/09-17:08:27 [fc_create_snapshot] Create Snapshot...
/bin/sync
/bin/sync
/bin/sync
/usr/sbin/lvcreate -L 160G --snapshot --name snap /dev/vg01/lv02
  Logical volume "snap" created
```

手順4） スナップショットボリュームを
マウント（/snap）

手順5） スナップショットを作成済なので、
MySQLはスタート

ここまでで数秒


2008/10/09-17:08:28 [fc_create_snapshot] lvcreate snapshot snap success
Report Target Lvm Snapshot Volume
/usr/sbin/lvdisplay -v --columns --units g /dev/vg01/snap
Using logical volume(s) on command line
LV VG #Seg Attr LSize Maj Min KMaj KMin Origin Snap% Move Copy% Log LV UUID
snap vg01 1 swi-a- 160.00G -1 -1 253 3 lv02 0.00 grgje3-YGO
I-qX4t-709j-KJp0-MJfn-xOKBa1
mount -o ro /dev/vg01/snap /snap
2008/10/09-17:08:28 [fc_create_snapsh] mount /dev/vg01/snap success.
Filesystem サイズ 使用 残り 使用% マウント位置
/dev/sda2 20G 2.9G 16G 16% /
/dev/sda1 487M 18M 444M 4% /boot
none 2.0G 0 2.0G 0% /dev/shm
/dev/mapper/vg01-lv01
50G 4.3G 43G 10% /binlog
/dev/mapper/vg01-lv02
197G 1.2G 186G 1% /data
/dev/mapper/vg01-lv03
247G 9.1G 225G 4% /backup
/dev/mapper/vg01-snap
197G 1.2G 186G 1% /snap
2008/10/09-17:08:28 [fc_create_snapshot] End
2008/10/09-17:08:28 [fc_create_snapshot] ----------
2008/10/09-17:08:28 [fc_mysqld_start] ----------
2008/10/09-17:08:28 [fc_mysqld_start] Start MySQL...
2008/10/09-17:08:28 [fc_mysqld_sta] port:13306
/etc/init.d/mysql13306 start
Starting MySQL [OK]
2008/10/09-17:08:29 [fc_mysqld_start] port:13306 success.
2008/10/09-17:08:29 [fc_mysqld_start] End
2008/10/09-17:08:29 [fc_mysqld_start] ----------

手順6） MySQLの対象データディレクトリをコピー

手順7） アンマウントして、スナップショットを削除

>> lvremove …

- 「LVM2」を利用したバックアップ例：実行例

```
2008/10/09-17:08:29 [fc_backup] ------------------------------------------
2008/10/09-17:08:29 [fc_backup] Backup MySQL Data Directory...
(cd /snap; cp -pr mysql13306 /backup/mysql13306.20081009_1708_29)
2008/10/09-17:09:03 [fc_backup] copy /snap/mysql13306 success
2008/10/09-17:09:03 [fc_backup] End
2008/10/09-17:09:03 [fc_backup] ------------------------------------------
umount /snap
2008/10/09-17:09:04 [fc_delete_snapshot] umount /snap success
Filesystem            サイズ  使用  残り 使用% マウント位置
/dev/sda2              20G  2.9G   16G  16% /
/dev/sda1             487M   18M  444M   4% /boot
none                  2.0G     0  2.0G   0% /dev/shm
/dev/mapper/vg01-lv01
                       50G  4.3G   43G  10% /binlog
/dev/mapper/vg01-lv02
                      197G  1.2G  186G   1% /data
/dev/mapper/vg01-lv03
                      247G   11G  224G   5% /b
/usr/sbin/lvremove -f /dev/vg01/snap
  Logical volume "snap" successfully removed
2008/10/09-17:09:06 [fc_delete_snapshot] lvremove /dev/vg01/snap success
2008/10/09-17:09:06 [fc_delete_snapshot] End
```

2-2. 障害監視

250台以上の

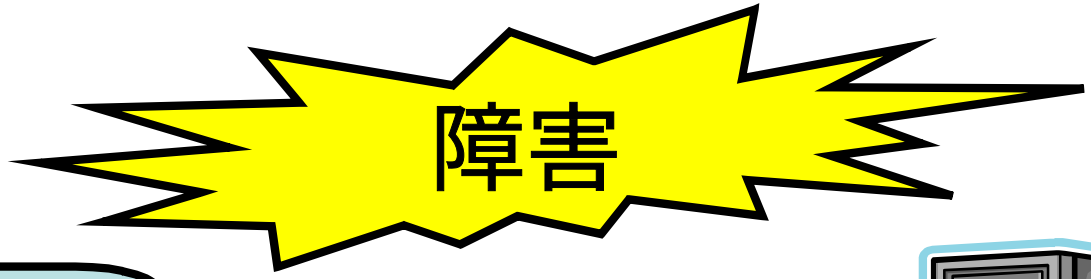
MySQLサーバ

どう守りますか？

私たちは、

こうして監視してます

- 障害を知るまで



OSSで構築

監視サーバが検知

アプリ担当者　インフラ担当者　DB担当者

- 障害からの復旧までのルール

監視サーバから携帯にメール

各サービス担当者が

責任をもって確認し，一時対応・報告

すぐに解決できない場合は

インフラ，DB担当も対応

- 結果として

✓ ダウンタイムの短縮に成功

← 担当者へリアルタイムでアラート

✓ 個々人のスキルアップ

← 障害内容は様々，自己スキル内に閉じない

障害監視の
監視項目とツールについて

MySQLサーバでは主に

・PING監視

・PORT監視

・レプリケーション監視

・RAID監視

を行っています

・「mon」で障害監視を行っています

MON: Operation Status: Summary View

Show Operational Status (summary) (failures only)	Show Alert History	Load scheduler state	Start scheduler	List Disabled Hosts/ Watches/ Svcs		Test Mon Config File
Show Operational Status (full)	Show Downtime Log	Save scheduler state	Stop scheduler	Reload auth file	List Mon PIDs	Reset Mon

This information was presented at 14:00:42 on Wednesday, 12-Aug-2009 (log in).
The scheduler on **localhost:2583** is currently running. This page will reload every 180 seconds.

Host Group	Service (legend)	Last Checked	Est. Next Check

Service color legend: (top of table)	Unchecked	Good	Failed (no alerts sent)	Failed (alerts sent)	Disabled

Show Operational Status (summary) (failures only)	Show Alert History	Load scheduler state	Start scheduler	List Disabled Hosts/ Watches/ Svcs		Test Mon Config File
Show Operational Status (full)	Show Downtime Log	Save scheduler state	Stop scheduler	Reload auth file	List Mon PIDs	Reset Mon

For questions about this server,
contact BOFH@your.domain

mon.cgi v1.4.2.1

「mon」はデフォルトで“PING”,

“PORT”監視をするコマンドが

用意されています

レプリケーション監視については

独自に作成しています

参考までにどのように

レプリケーション監視を

行っているか確認してみましょう

MySQLのレプリケーション監視は

MySQLコマンドと

「SNMP (Simple Network Management Protocol)」を

使用して行っています

レプリケーションが

問題なく稼動しているかは、

MySQLのコマンド

「**SHOW SLAVE STATUS**」

で確認出来ます

- 「mon」で障害監視

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
             Slave_IO_State: Waiting for master to send event
                Master_Host: 192.168.1.21
                Master_User: replication
                Master_Port: 3306
              Connect_Retry: 60
            Master_Log_File: mysql-m1-bin.000001
        Read_Master_Log_Pos: 79
             Relay_Log_File: mysql-s1-relay-bin.000001
              Relay_Log_Pos: 79
      Relay_Master_Log_File: mysql-m1-bin.000001
           Slave_IO_Running: Yes
          Slave_SQL_Running: Yes
            Replicate_Do_DB:
..........................................................

             Master_SSL_Key:
      Seconds_Behind_Master: 0
```

- 「LVM2」を利用したバックアップ例

Slave_IO_Running、Slave_SQL_Runningをチェックする

シェルコマンド作成

```
# vi mysql_repl.sh

#!/bin/sh

CNT=0

CNT=`mysql -uroot -e'show slave status¥G' | grep Running | grep No -c`

if [ "$CNT" != 0 ]
then
    echo 1
    exit 1;
fi

echo 0
exit 0;
```

「SNMP」にカスタムコマンドとして追加

```
# vi snmpd.conf

extend .1.3.6.1.4.1.3306.10 mysql_repl /bin/sh mysql_repl.sh

# /etc/init.d/snmpd reload
```

追加した値で実行して、値がかえってきたら

```
# snmpwalk -v 2c -c casnmp localhost .1.3.6.1.4.1.3306.10

SNMPv2-SMI::enterprises.3306.10.1.0 = INTEGER: 1
‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘
```

あとは、「**mon**」側で

追加した**SNMP**を実行する

スクリプトを作成して完了となります

- 「mon」で障害監視

```
#!/bin/sh

RET=''
ERRORWORD='MySQL replication check failed.'
ERRORWORD1='MySQL replication has stopped.'

for host in "$@"
do

   #EXEC
   RET=`snmpwalk -v2c -c test ${host} .1.3.6.1.4.1.3306.10 | gawk '{print $4}'`

   #RET CHECK
   if [ "${RET}" = 0 ]; then
         :
   else
      if [ "$SummaryOutput" = "" ]; then
            SummaryOutput=${host}
            eval DetailedText=¥${host}¥" : ¥"¥${ERRORWORD${RET}}
      else
            SummaryOutput="${SummaryOutput} ${host}"
            eval DetailedText=¥${DetailedText}¥" ¥¥n¥"¥${host}¥" : ¥"¥${ERRORWORD${RET}}
      fi
   fi

done

# ERROR CHECK
if [ "${SummaryOutput}" != "" ]; then
      echo "${SummaryOutput}"
      echo -e "${DetailedText}"
      exit 1
fi

exit 0
```

- 「mon」で障害監視

それ以外にも、“Seconds_Behind_Master”の値を

チェックすることでレプリケーションの遅延を

検知することも可能です

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
             Slave_IO_State: Waiting for master to send event
                Master_Host: 192.168.1.21
                Master_User: replication
                Master_Port: 3306
.....................................................

             Master_SSL_Key:
       Seconds_Behind_Master: 0
```

ただ、

Master - Slave間のネットワーク瞬断した時

「Slave_IO_Running」が“No”にならず、

障害検知出来ない場合がありました

この場合、

「**slave_net_timeout**」の

設定で対処することが可能です

- レプリケーションのトラフィック

「**slave_net_timeout**」は、マスターから設定した値(秒数)の間、バイナリログを受信しない場合、再接続を行う設定になります

- 『slave_net_timeout』の設定

オンラインで設定・反映が可能です(Slaveのみ)

ここでは、設定を30秒にしています

```
mysql> set global slave_net_timeout = 30;
Query OK, 0 rows affected (0.00 sec)
```

確認は「SHOW VARIABLES」で行えます

```
mysql> SHOW VARIABLES LIKE 'slave_n%';
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| slave_net_timeout | 30    |
+-------------------+-------+
```

- 『slave_net_timeout』の設定

後は、**MySQL**を再起動しても設定が反映されるよう

my.cnfへ追記しましょう

```
[mysqld]
・・・・・・・・・・・・
slave_net_timeout = 30
```

「**slave_net_timeout**」設定を行うことで、

ネットワーク障害の対応を

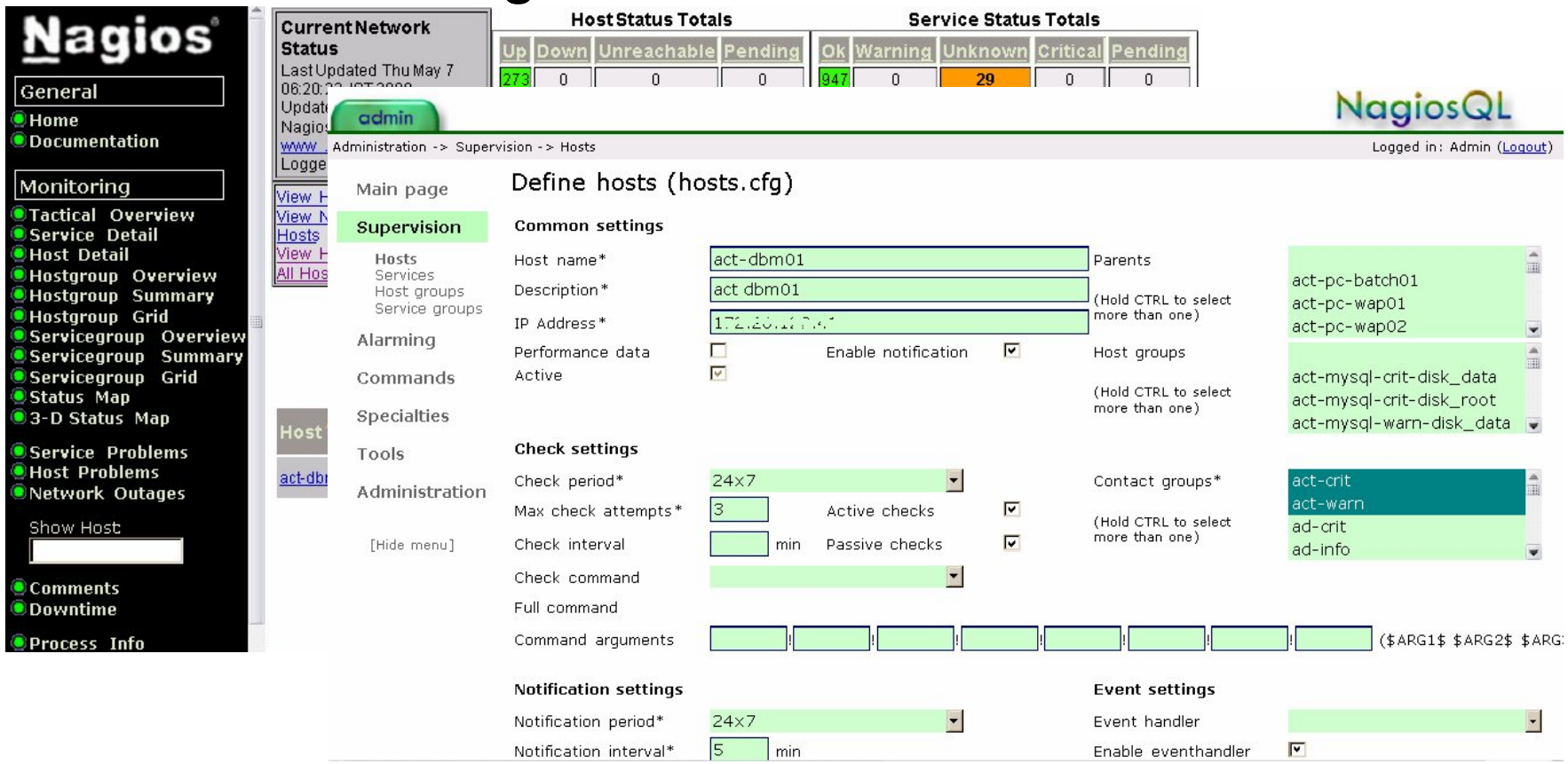
より短縮することが可能になります

- その他の障害監視ツール

また、「**mon**」以外に

「**Nagios**」を使用して

“Load Average”と“Diskの使用率”も監視しています



2-3. 性能監視

障害監視以外の

日々の性能も

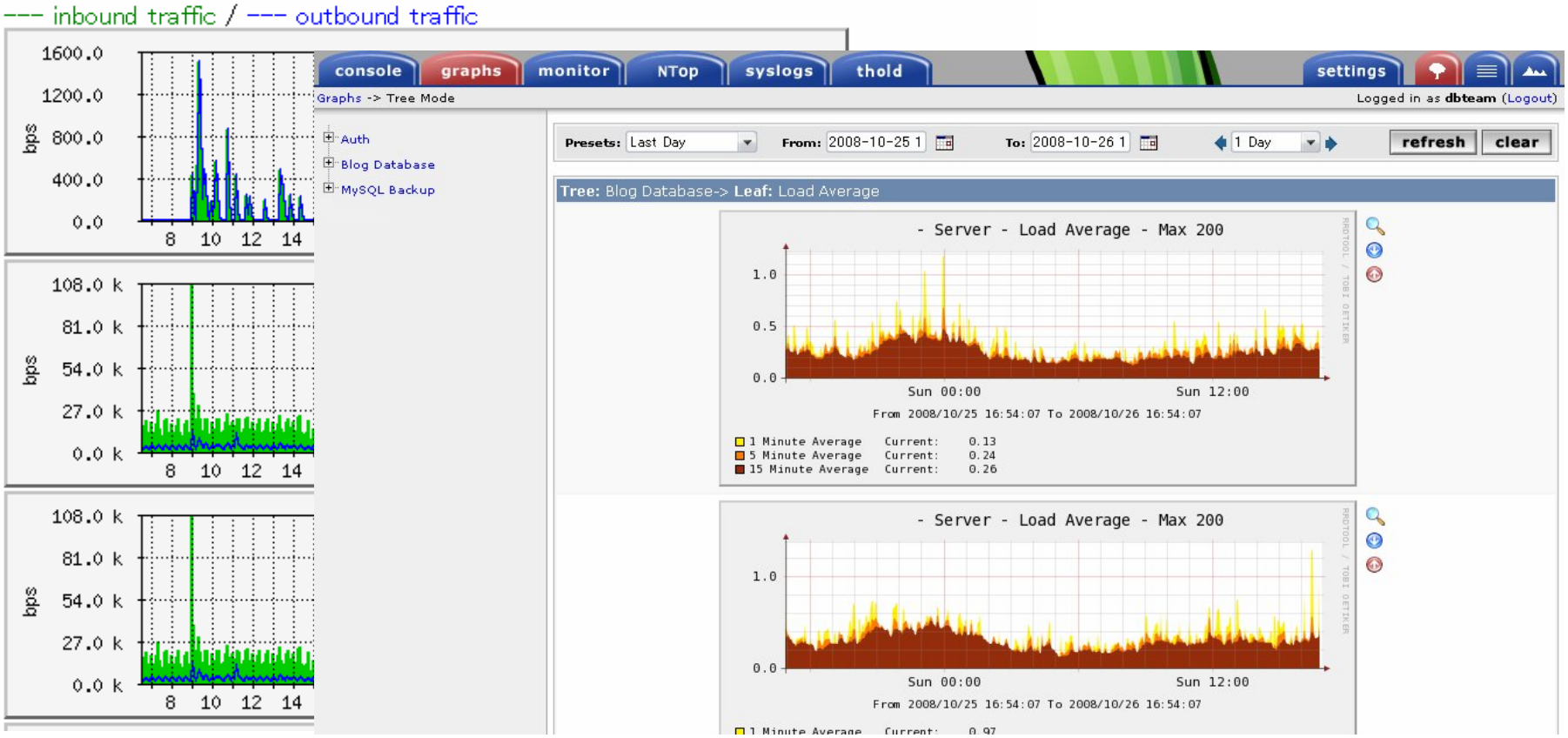
確認する必要があります

性能監視は

サーバ側と**MySQL**側で

行う必要があります

サーバの性能監視の多くは「MRTG」か「Cacti」を使用しています



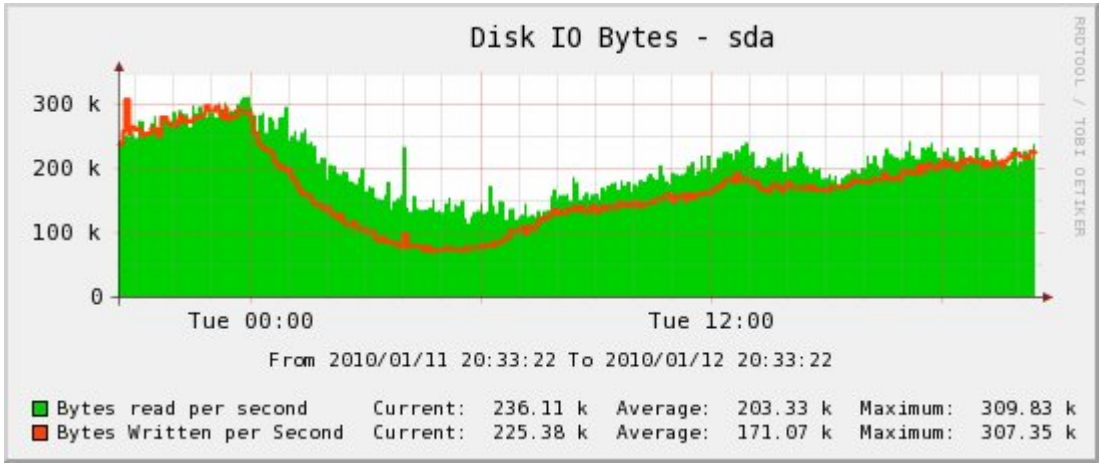
サーバの性能監視は主に

- **Load Average**
- トラフィック量
- **Disk I/O**
- **Memory使用量**
- **Disk使用量**
- レプリケーション遅延
- **vmstat**

「**Disk I/O**」が問題になると**SQL**の

レスポンスが急激に悪くなることがあるので

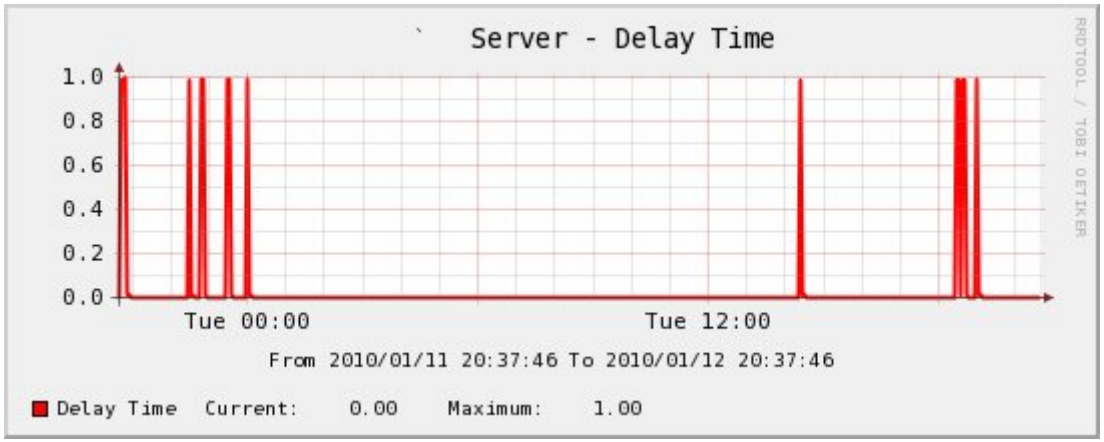
監視が必要です



また、「**Disk I/O**」が問題で

レプリケーションの遅延が発生することがあるので

合わせて監視するとよいと思います



最後に「vmstat」の監視ですが、

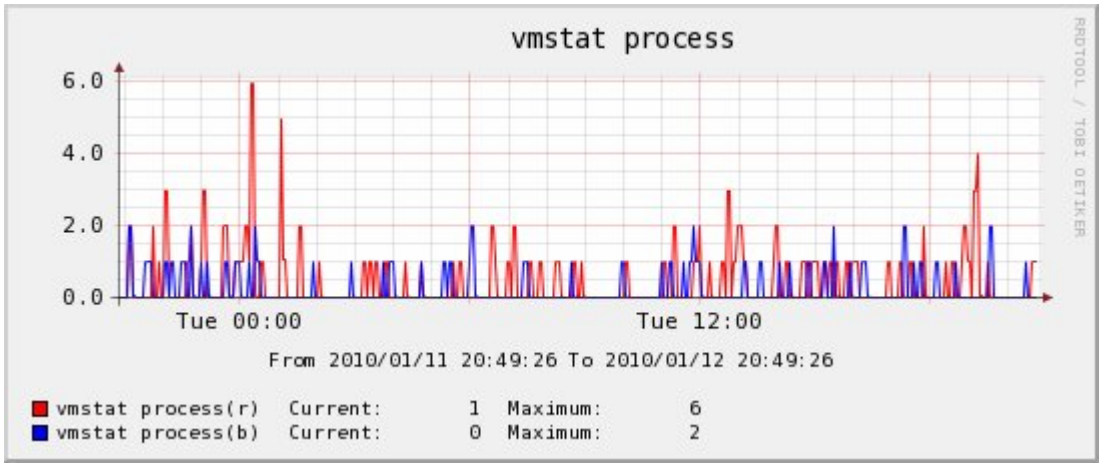
監視する項目は以下にしています

```
# vmstat 1
procs -----------memory---------- ---swap-- -----io---- --system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in    cs us sy id wa
 0  0    160 208452  73560 2714100    0    0     0     0 1619  2623  5  3 93  0
 3  0    160 190748  73560 2714100    0    0     0     0 1343  1948  4  7 90  0
 1  0    160 182116  73560 2714100    0    0     0  3460 1908  1843  8  6 73 13
 0  0    160 183924  73560 2714100    0    0     0     0 1641  3431 13  5 82  0
```

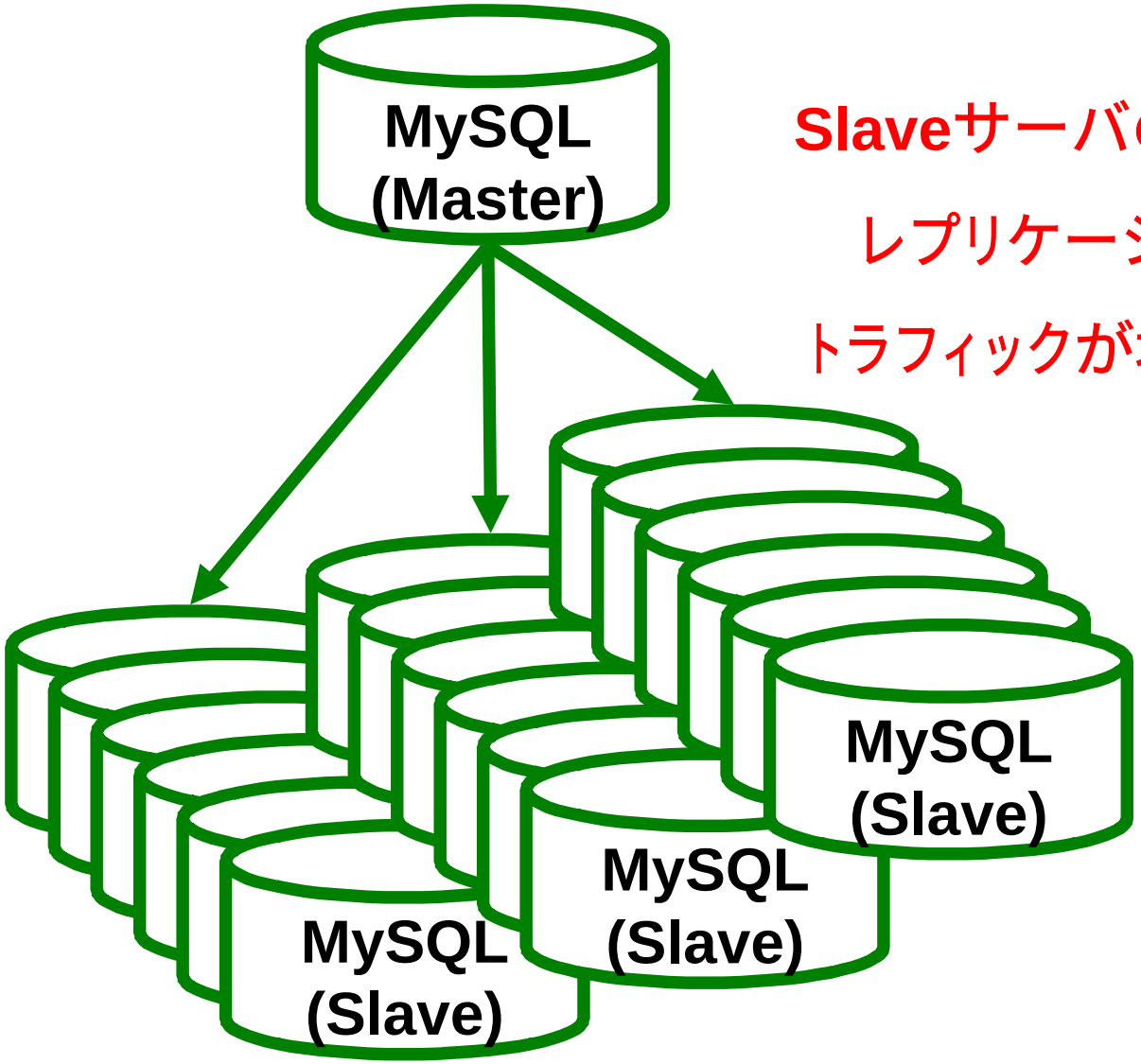
r：実行待ち状態にあるプロセス数
b：割り込み不可能なスリープ状態にあるプロセス数

「vmstat」の” procs”で

サーバのキャパシティの確認を行っています



- MySQLの性能監視



Slaveサーバの台数が増え

レプリケーションによる

トラフィックが増加した場合

- レプリケーションのトラフィック

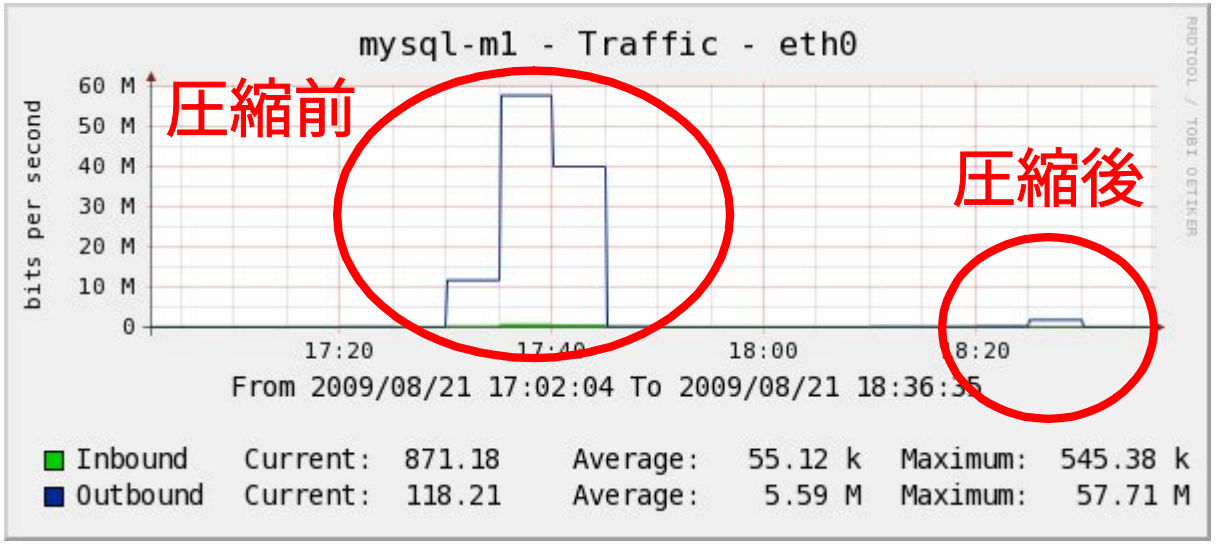
『slave_compressed_protocol』を

設定することで解消することが出来ます

実際に「INSERT」が多く

「LOAD DATA　INFILE」を使用している

サービスでは以下のような結果になりました



- 『slave_compressed_protocol』の設定

オンラインで設定が可能です(Master,Slave共に必要)

```
mysql> set global slave_compressed_protocol = 1;
Query OK, 0 rows affected (0.00 sec)
```

- MySQL4系の場合

```
mysql> select @@slave_compressed_protocol;
+-----------------------------+
| @@slave_compressed_protocol |
+-----------------------------+
|                           1 |
+-----------------------------+
```

- MySQL5系の場合

```
mysql> show variables like 'slave_c%';
+-----------------------------+-------+
| Variable_name               | Value |
+-----------------------------+-------+
| slave_compressed_protocol | ON    |
+-----------------------------+-------+
```

- 『slave_compressed_protocol』の設定

設定を反映するためにはI/Oスレッドかレプリケーションを一度停止する必要があります

```
mysql> stop slave io_thread;
OR
mysql> stop slave;
```

```
mysql> start slave io_thread;
OR
mysql> start slave;
```

後は、MySQLを再起動しても設定が反映されるようmy.cnfへ追記しましょう

```
[mysqld]
・・・・・・・・・・・・
slave_compressed_protocol = 1
```

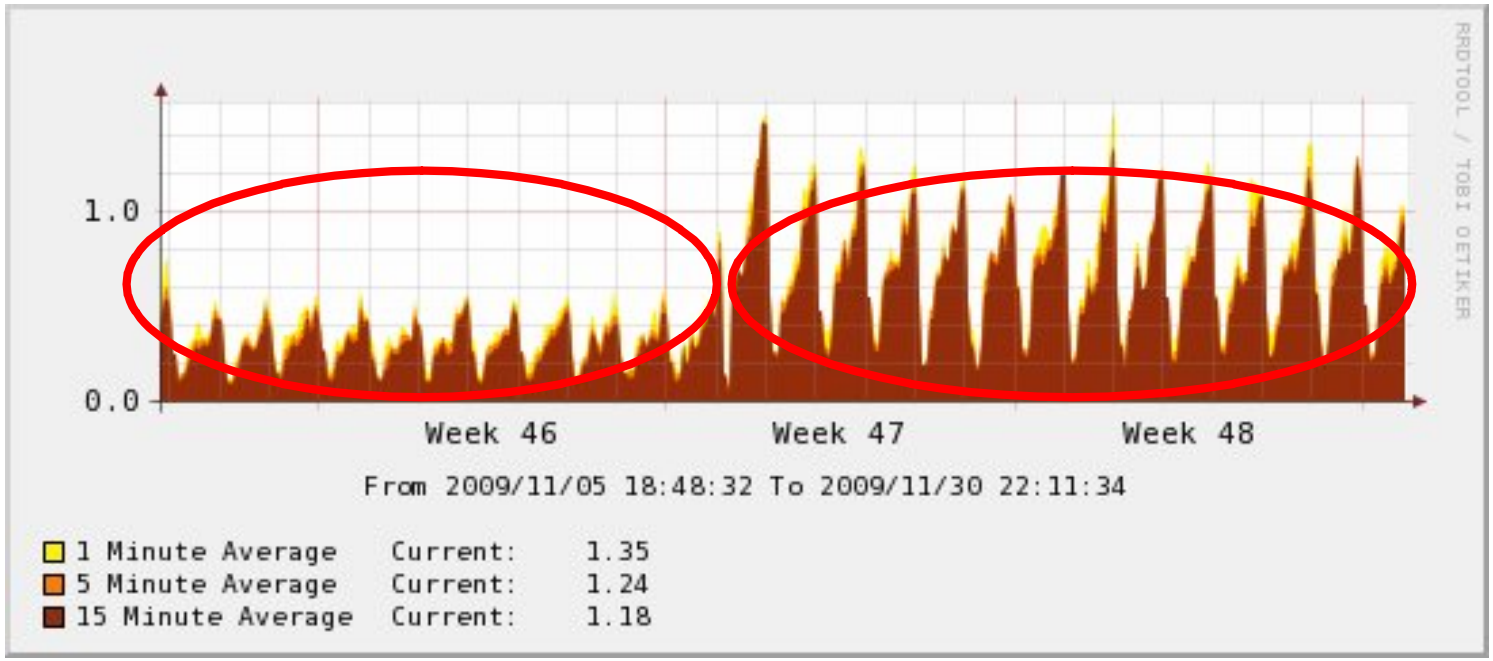
- レプリケーションのトラフィック

ただし、デメリットもあります

Slaveサーバが32台あり、ピーク時間の更新処理が

1分間に6,500～7,000のサーバで、

負荷が倍以上になっています



- レプリケーションのトラフィック

このように、更新回数

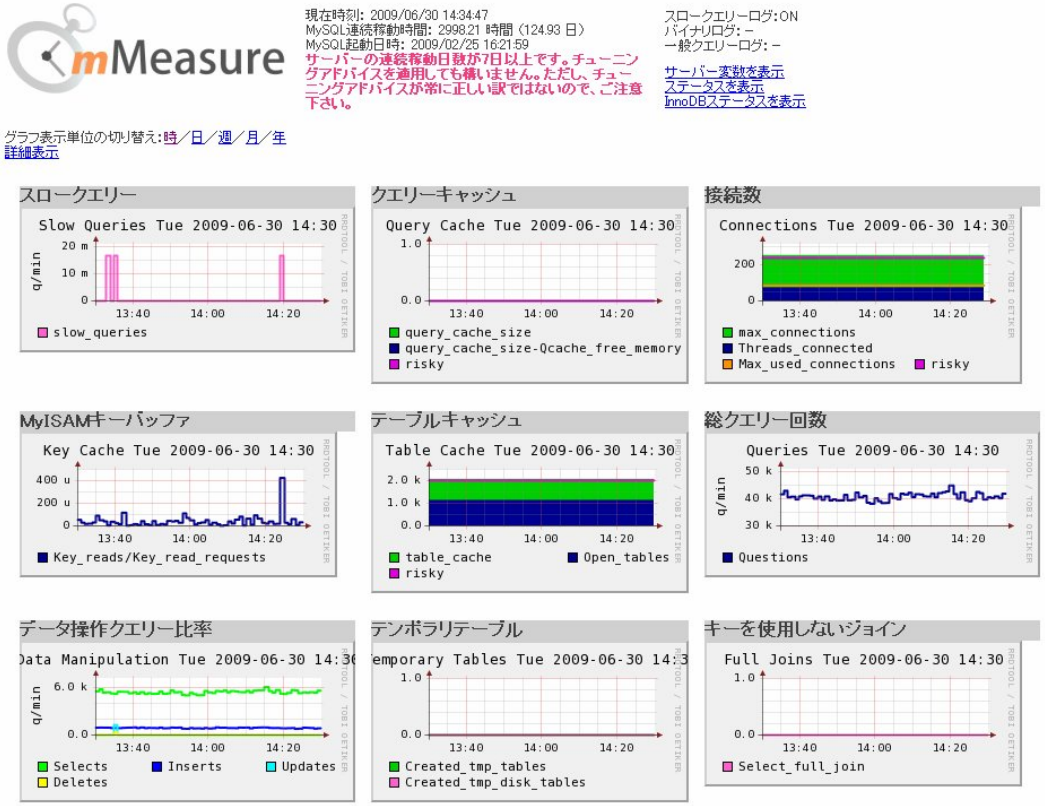
(バイナリログの転送回数)が

多い場合はMasterの負荷が上がって

しまう事があるので注意が必要です

MySQLの性能監視は

「mMeasure」を使用しています

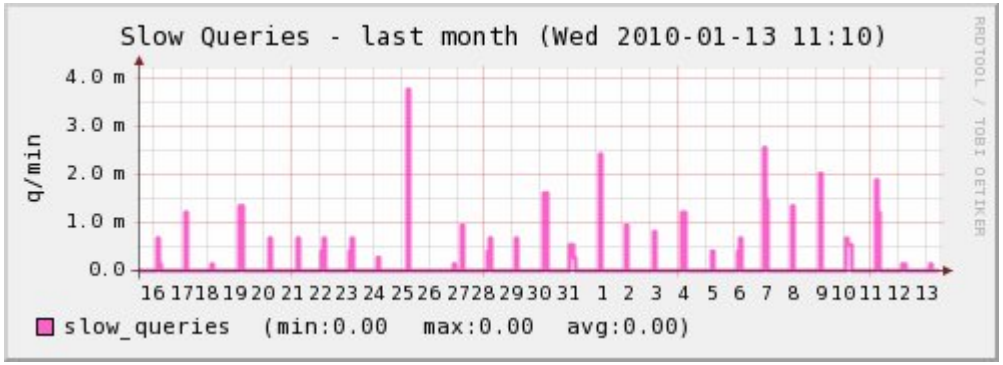


スロークエリーの状況で

現在のSQLに問題がないか

また、データ増加による影響を

受けていないか確認しています

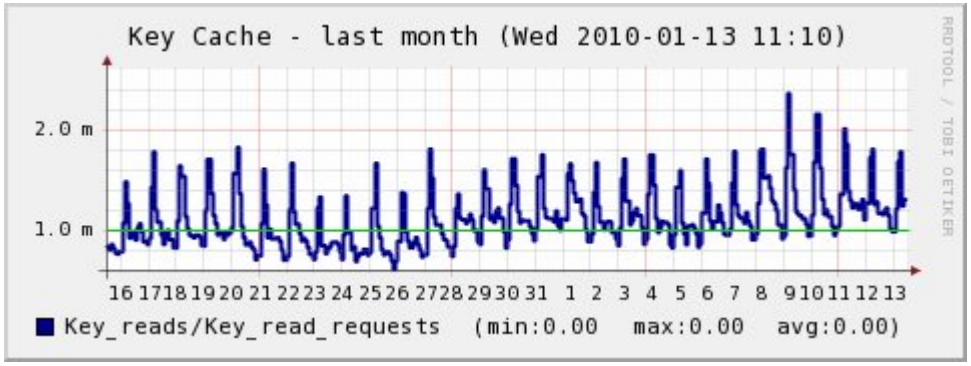


MyISAMの場合は

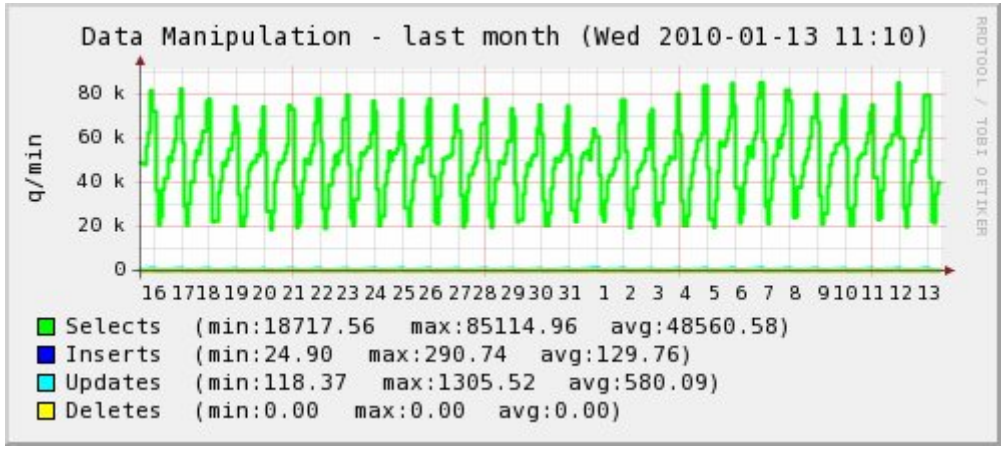
キャッシュのHit率を確認し

Hit率が低い場合は「key_buffer_size」

の見直すかサーバの分散を行います



最後に、SQLの発行回数を確認しています



以前、ブログで使用してる

サーバで秒間の

SQL発行回数が2,500回を

超えたところで処理が詰まってしまう

ということが起きました

サーバのスペックによって

SQLの発行回数に限界があるので、

ベンチマークなどを使用して

限界値を見極め

サーバの増強計画立てています

この様に日々、

監視を強化することで

安定したサービスを

提供できるでしょう

- ダウンロードサイト

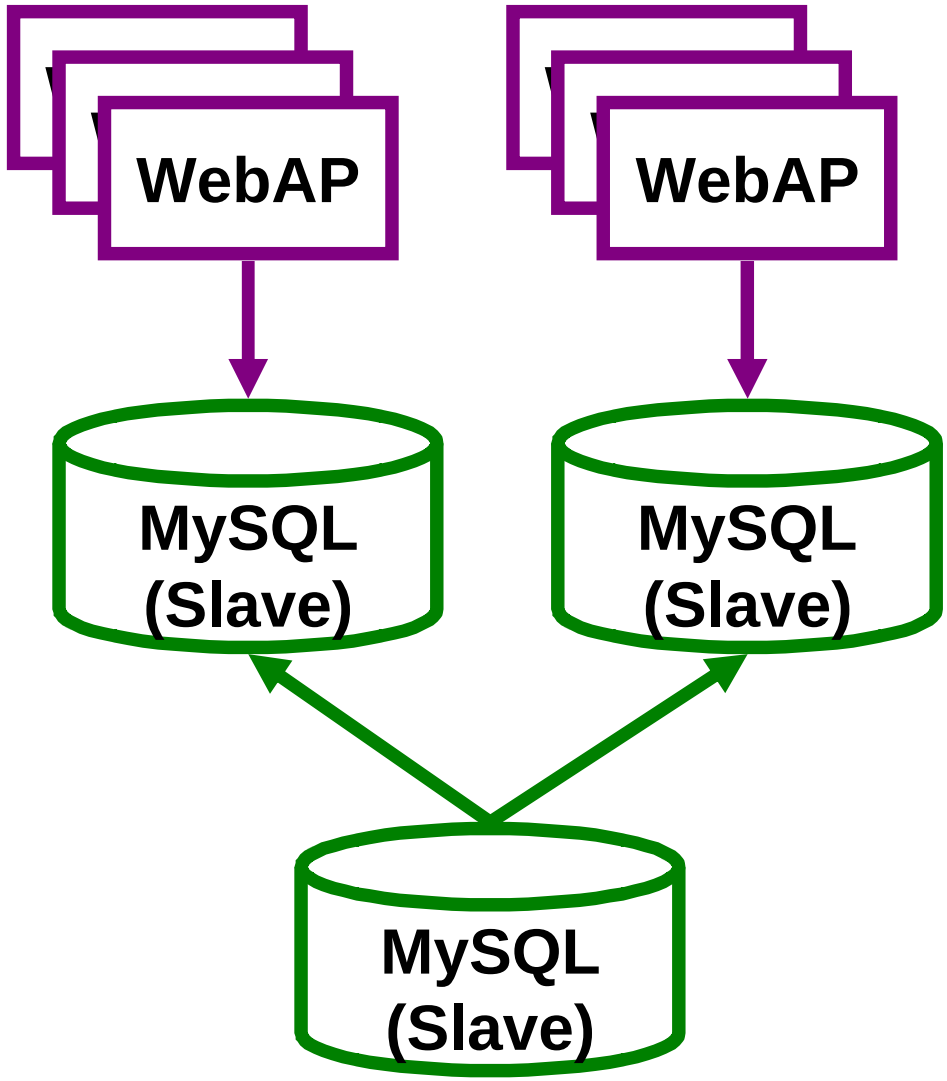
- ✓ mon
  - http://mon.wiki.kernel.org/index.php/Main_Page
- ✓ Nagios
  - http://www.nagios.org/
- ✓ Cacti
  - http://www.cacti.net/
- ✓ mMeasure
  - http://mmeasure.sourceforge.jp/
- ✓ munin
  - http://munin.projects.linpro.no/

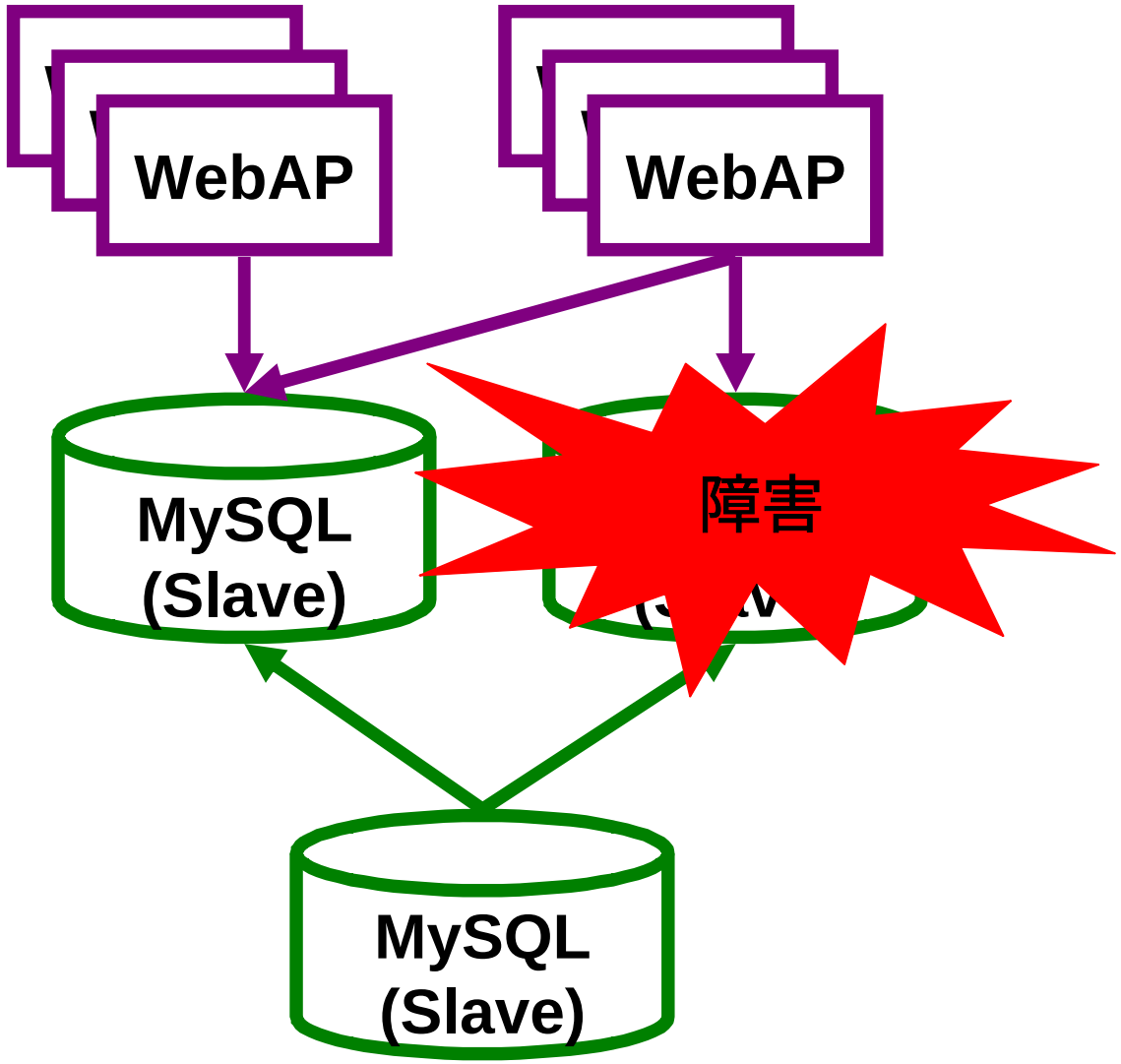
3.今後の取り組み

- MySQL5.1及び5.5の検証
- Percona、XtraDBの検証
- Key-Value型データベースの調査・検証
- WebAPとDBの接続の最適化

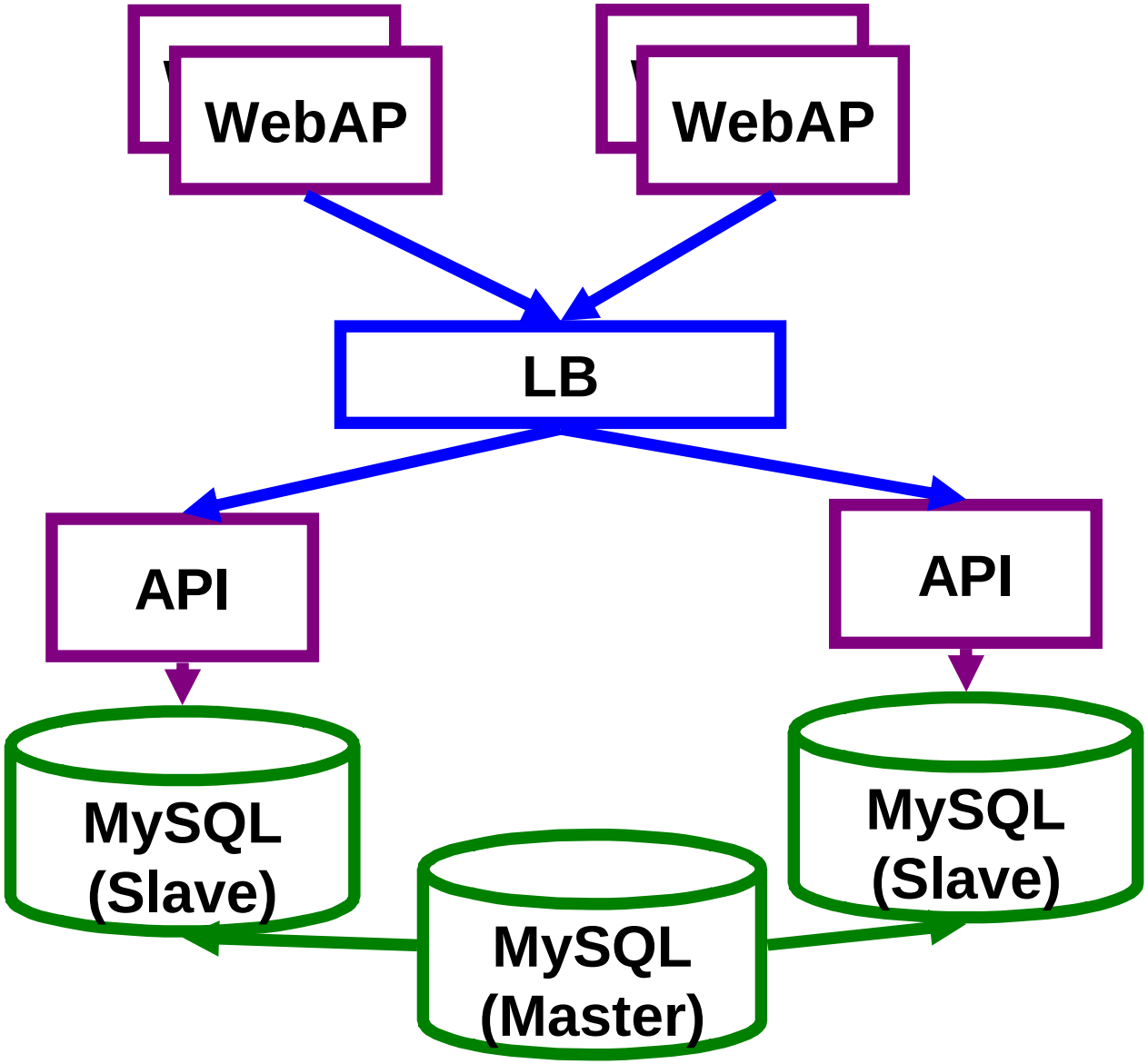
WebAPとデータベースの接続について

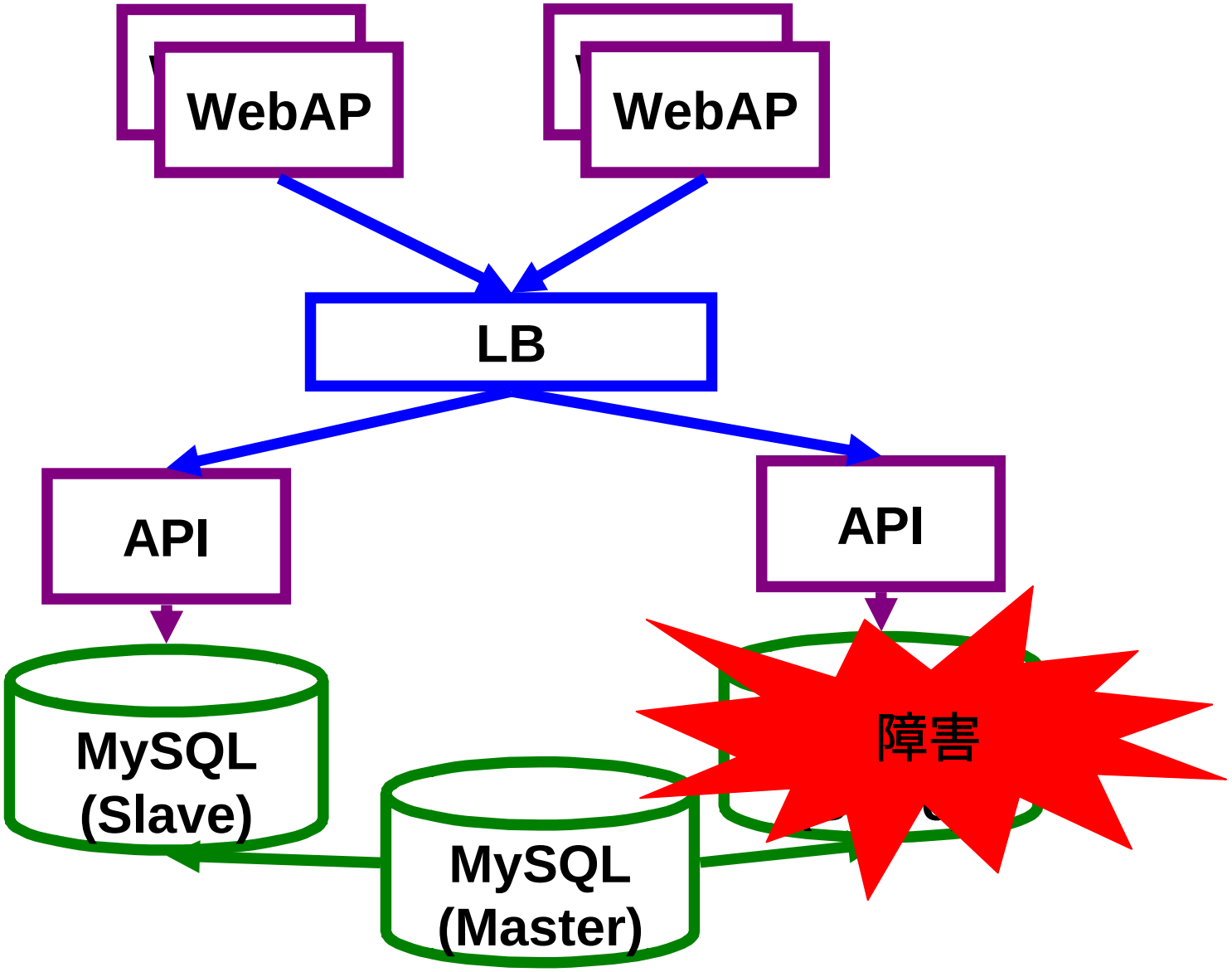
基本的に接続構成としては


WebAP
WebAP
MySQL
(Slave)
MySQL
(Slave)
MySQL
(Slave)

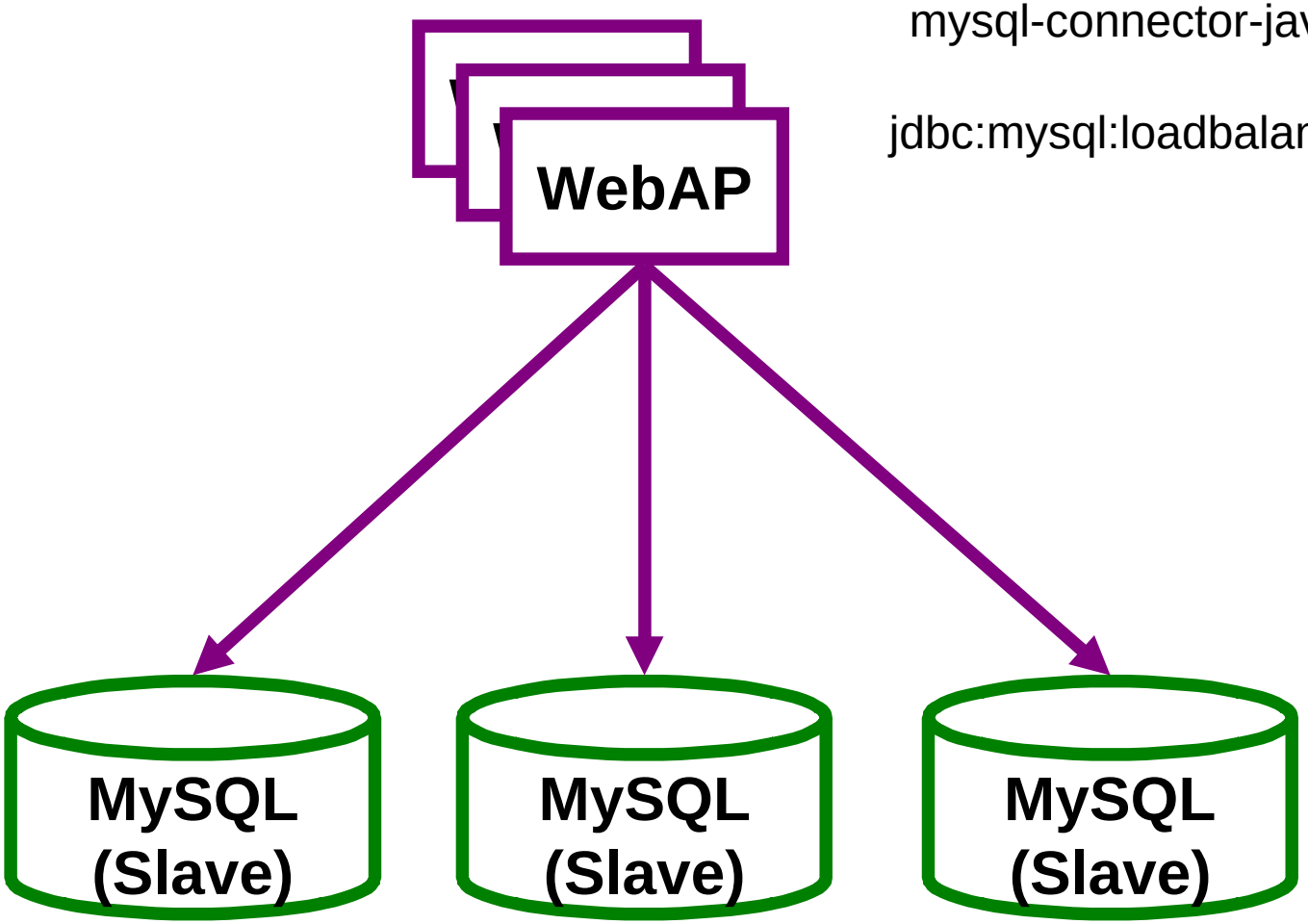

WebAP
WebAP
MySQL
(Slave)
障害
MySQL
(Slave)

LBとAPIを使用した方式として


WebAP
WebAP
LB
API
API
MySQL
(Slave)
MySQL
(Master)
MySQL
(Slave)

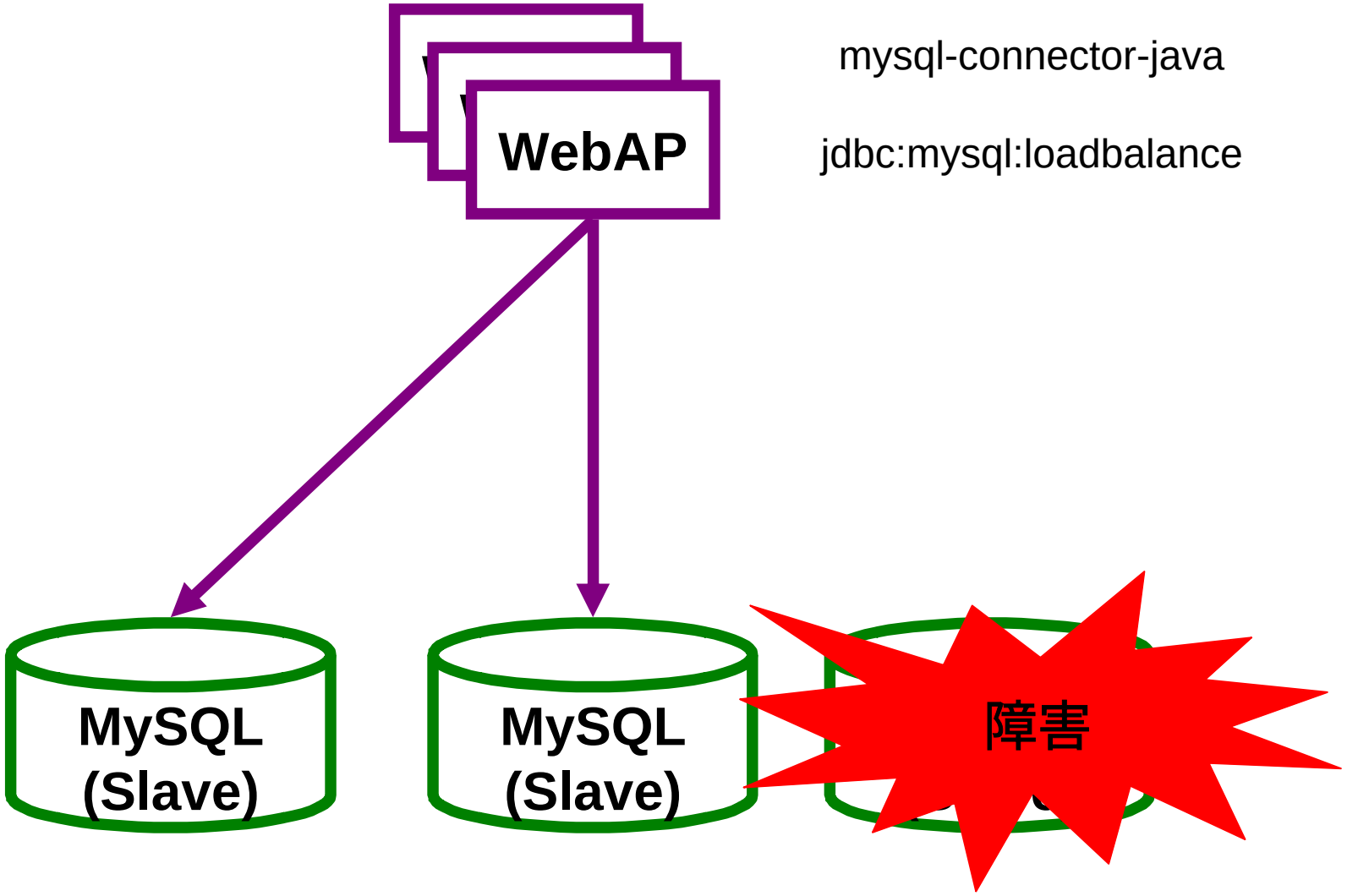

WebAP
WebAP
LB
API
API
MySQL
(Slave)
MySQL
(Master)
障害

mysql-connector-javaのloadbalance
を使用した方式

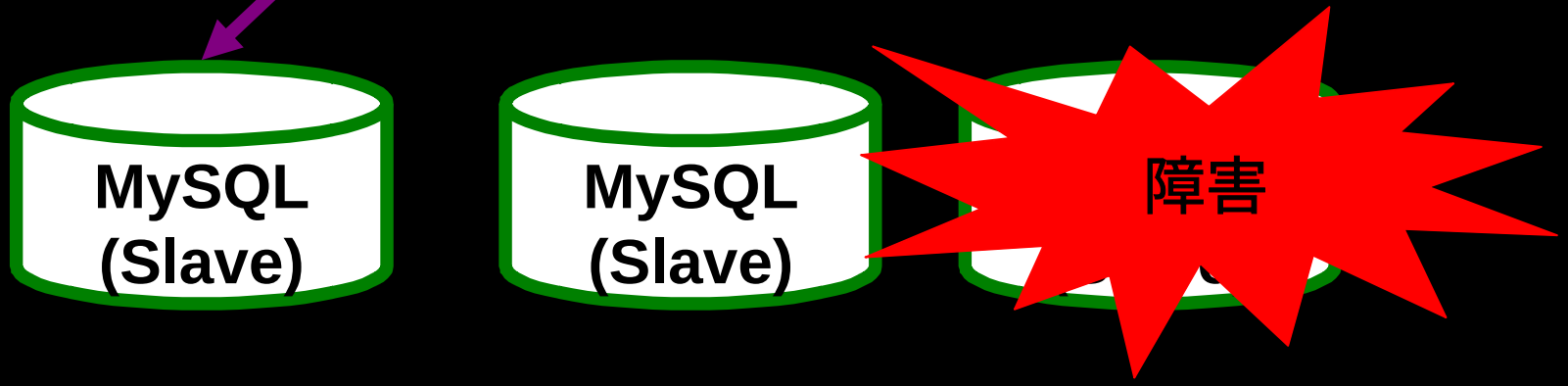


mysql-connector-java

jdbc:mysql:loadbalance


WebAP
mysql-connector-java
jdbc:mysql:loadbalance
MySQL
(Slave)
MySQL
(Slave)
障害

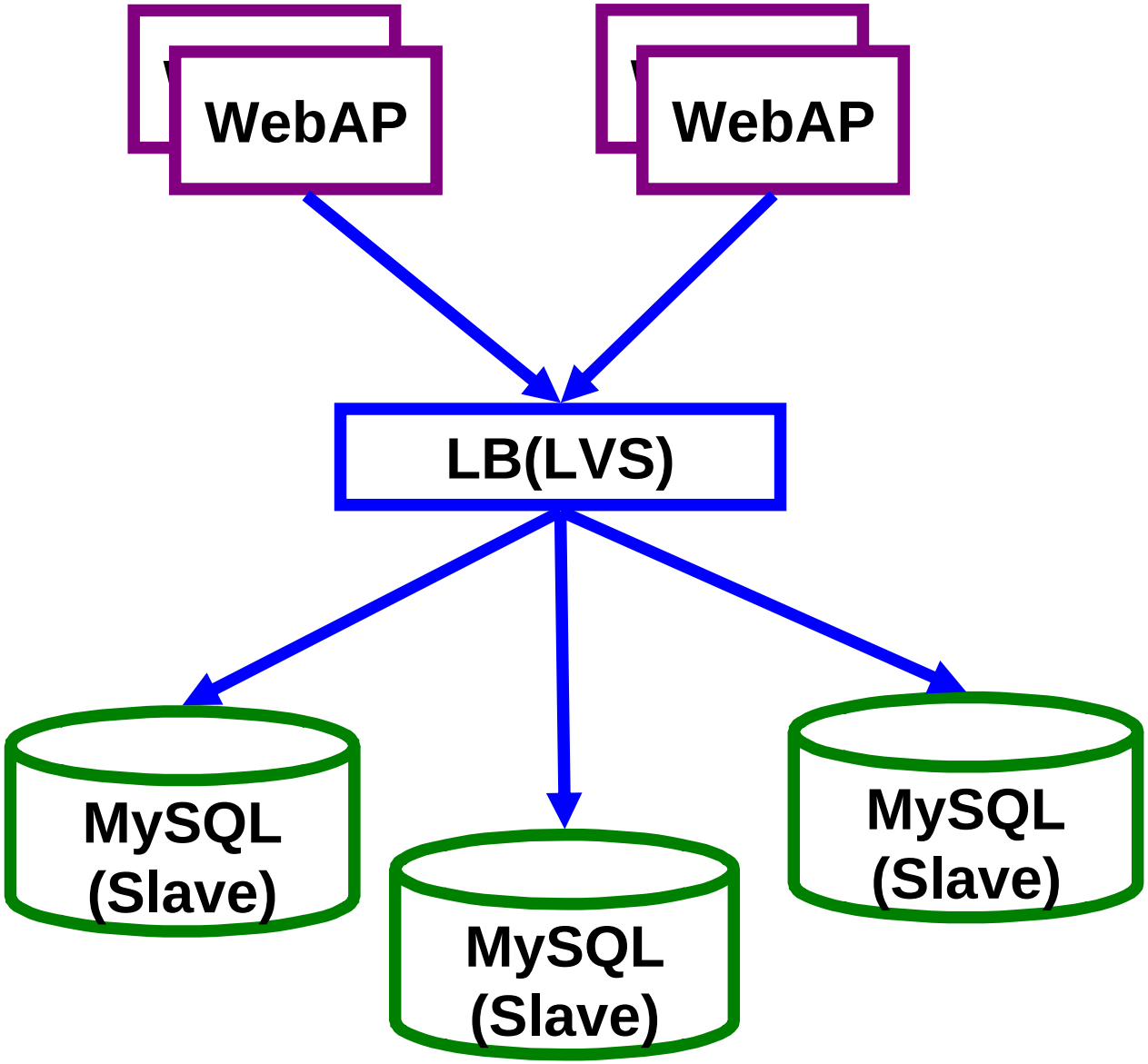
この方式は、コネクションプーリングが
出来ないのでアクセスが多くなった場合に
接続による負荷が上がることがあります

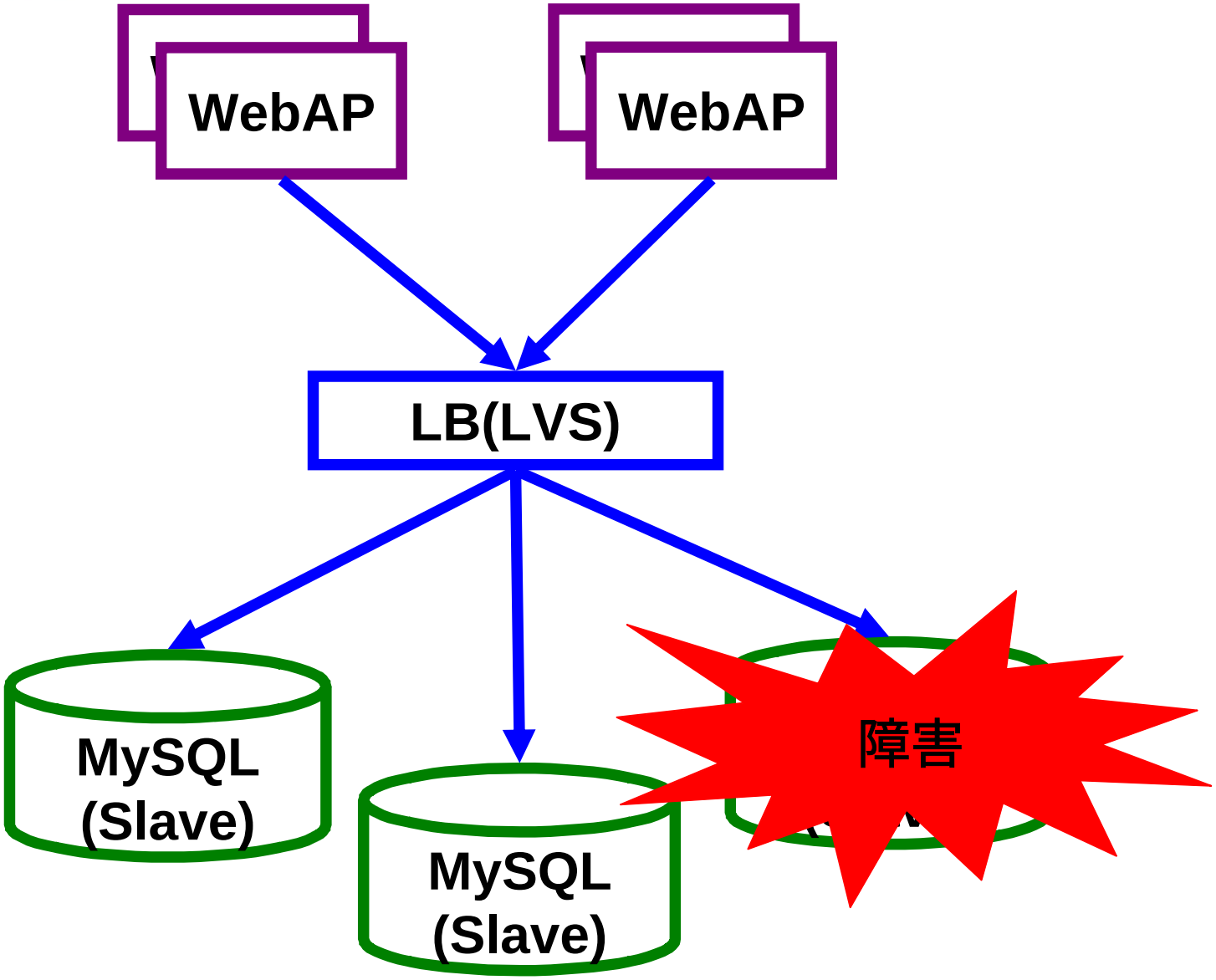


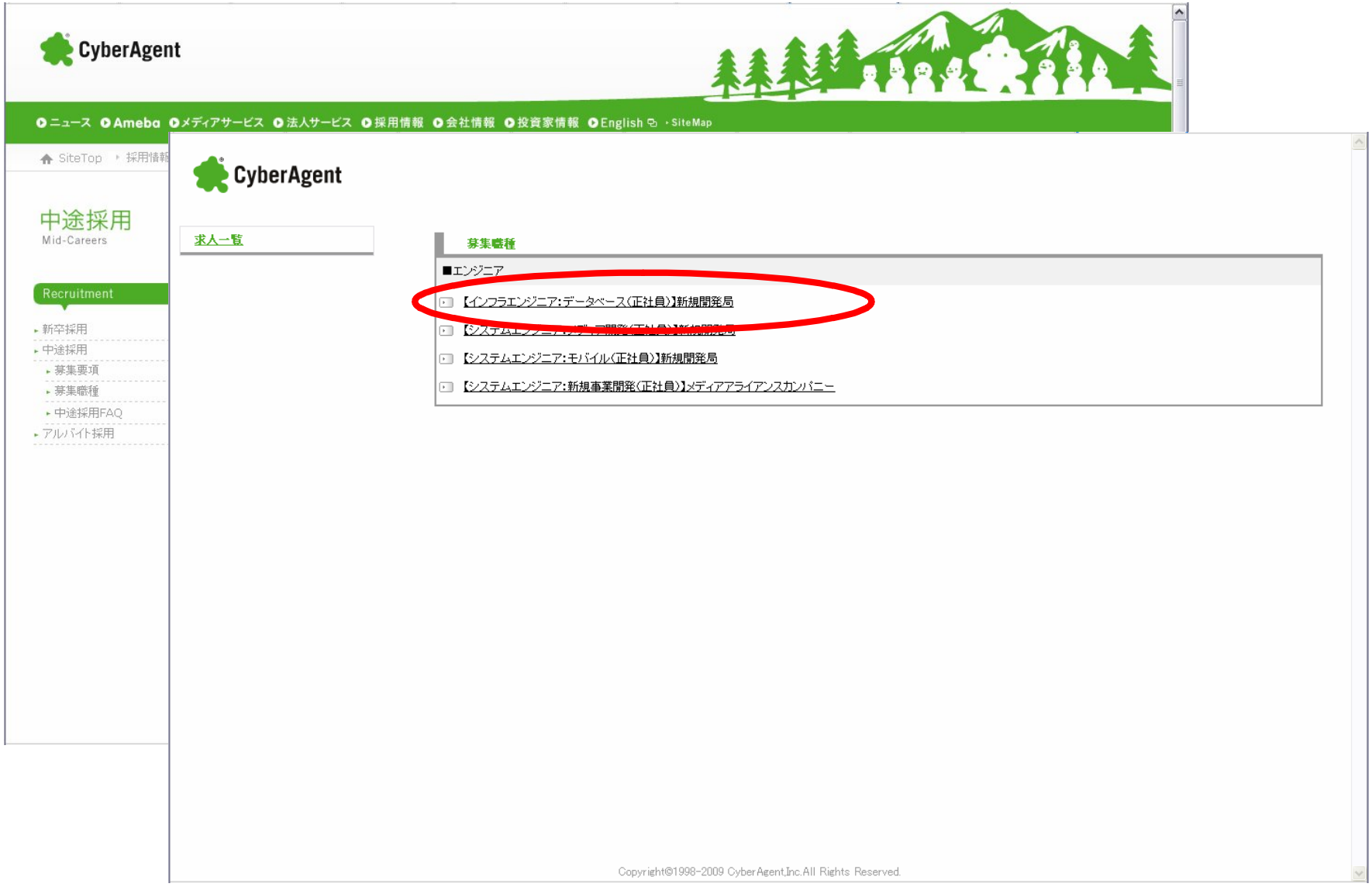
LVS(Linux Virtual Server)を使用した方式

LVSはフリーの

ロードバランサーになります


WebAP
WebAP
LB(LVS)
MySQL
(Slave)
MySQL
(Slave)
MySQL
(Slave)


WebAP
WebAP
LB(LVS)
MySQL
(Slave)
MySQL
(Slave)
障害


CyberAgent
ニュース Ameba メディアサービス 法人サービス 採用情報 会社情報 投資家情報 English SiteMap
SiteTop
中途採用
Mid-Careers
Recruitment
新卒採用
中途採用
募集要項
募集職種
中途採用FAQ
アルバイト採用
CyberAgent
求人一覧
募集職種
■エンジニア
【インフラエンジニア:データベース(正社員)】新規開発局
【システムエンジニア:モバイル(正社員)】新規開発局
【システムエンジニア:新規事業開発(正社員)】メディアアライアンスカンパニー
Copyright©1998-2009 CyberAgent,Inc.All Rights Reserved.

ご静聴ありがとうございました.