Ameba

pigg

サーバーサイド編

名村　卓

ソフトウエアエンジニア

アメーバピグ

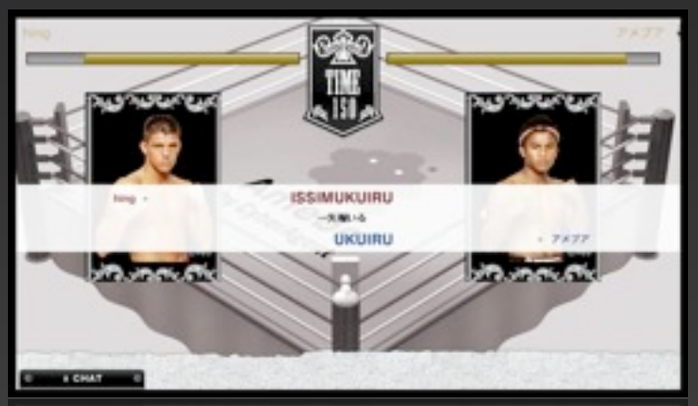
NinjaTrick

プーペガール

みんなの絵文字

メロメロパーク

K1-MAX タイピングバトル



今日話すこと


アメーバピグについて


アメーバピグのサーバー技術


Flash と Java の連携


開発運用について


アメリカ版アメーバピグについて

Ameba

1

アメーバピグについて

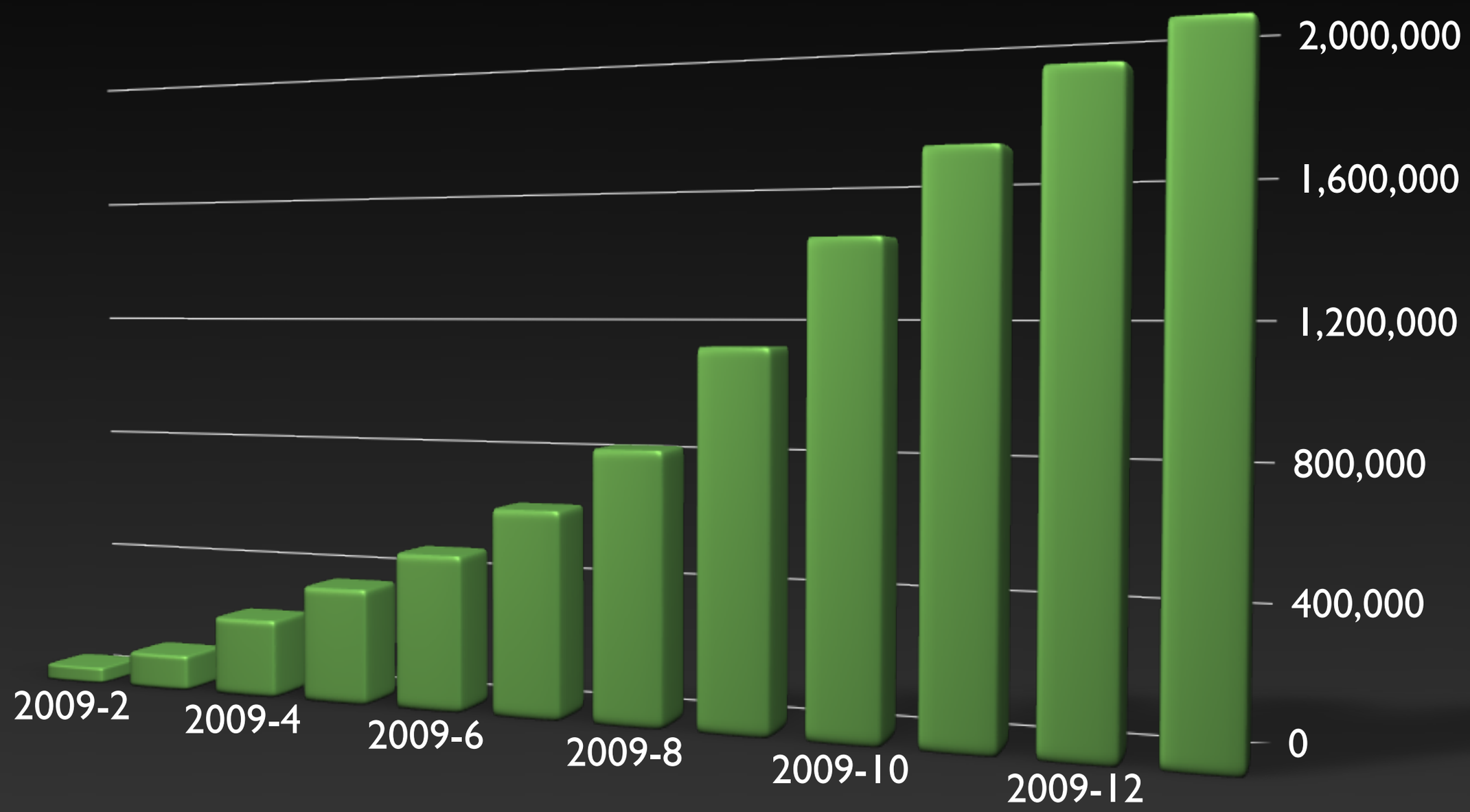
Ameba

2009年 2月 オープン

Ameba

登録者数は 約200万人

Ameba


2,000,000
1,600,000
1,200,000
800,000
400,000
0
2009-2
2009-4
2009-6
2009-8
2009-10
2009-12

Ameba

1人あたりの平均滞在時間

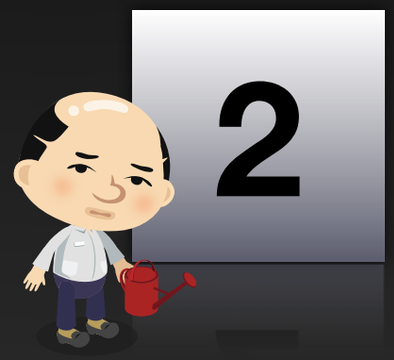
1日 60分~70分

1ヶ月 600分~700分

Ameba

リアルタイム
コミュニケーション

Ameba



アメーバピグの

サーバー技術

設計方針

省エネ・省コスト

壊れてもいい

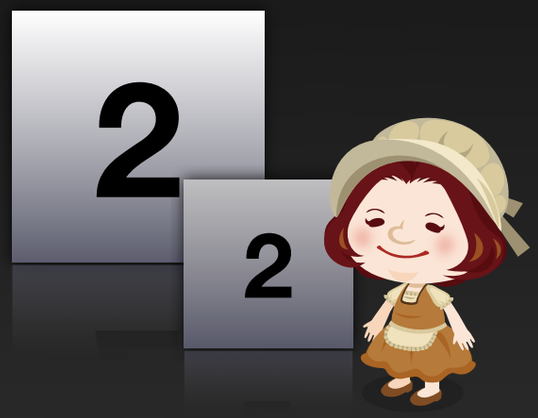
リニアにスケールできる

Ameba

1つの処理を 0.00001 秒

10,000 の同時処理を 0.1秒

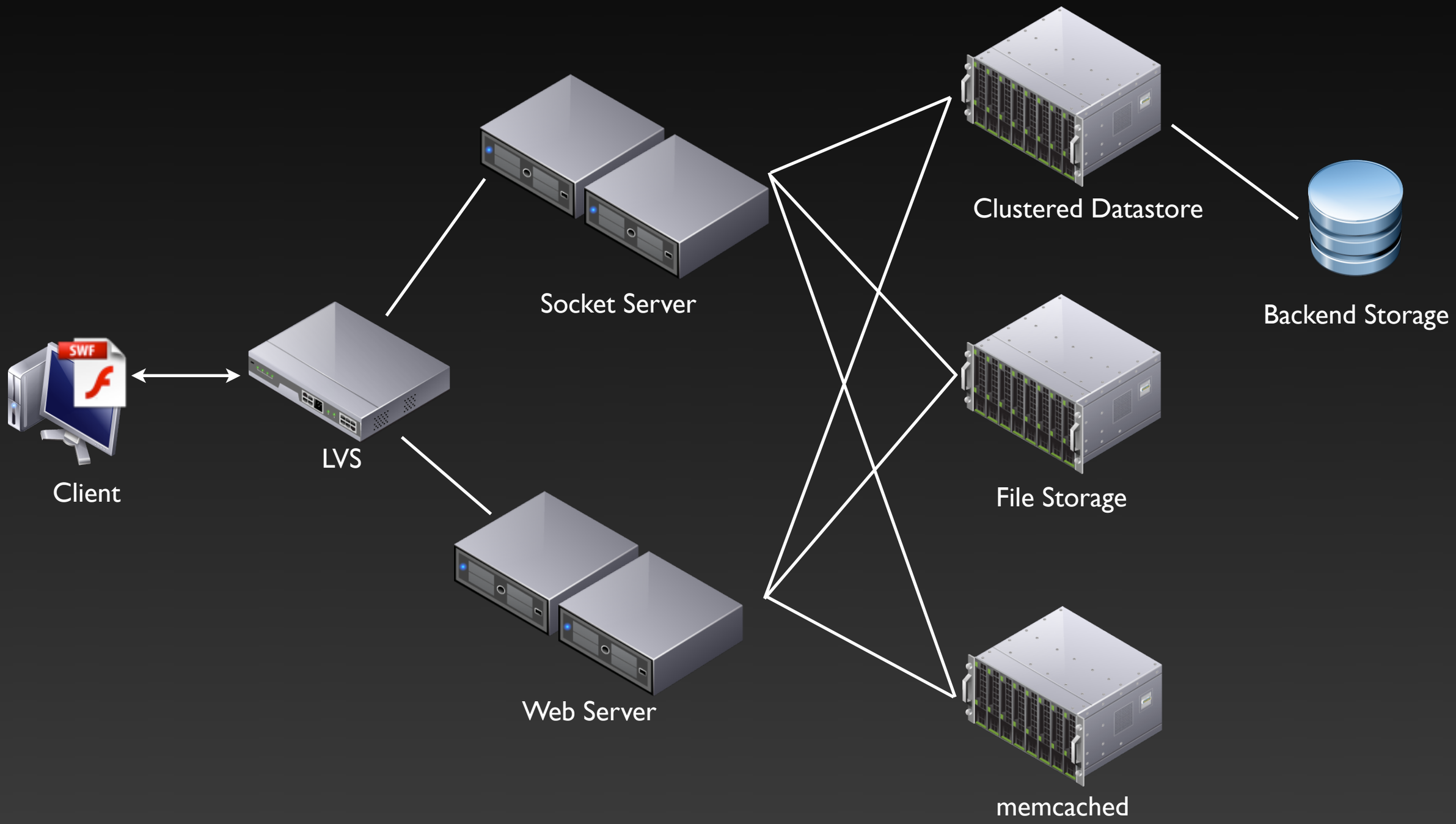
Ameba



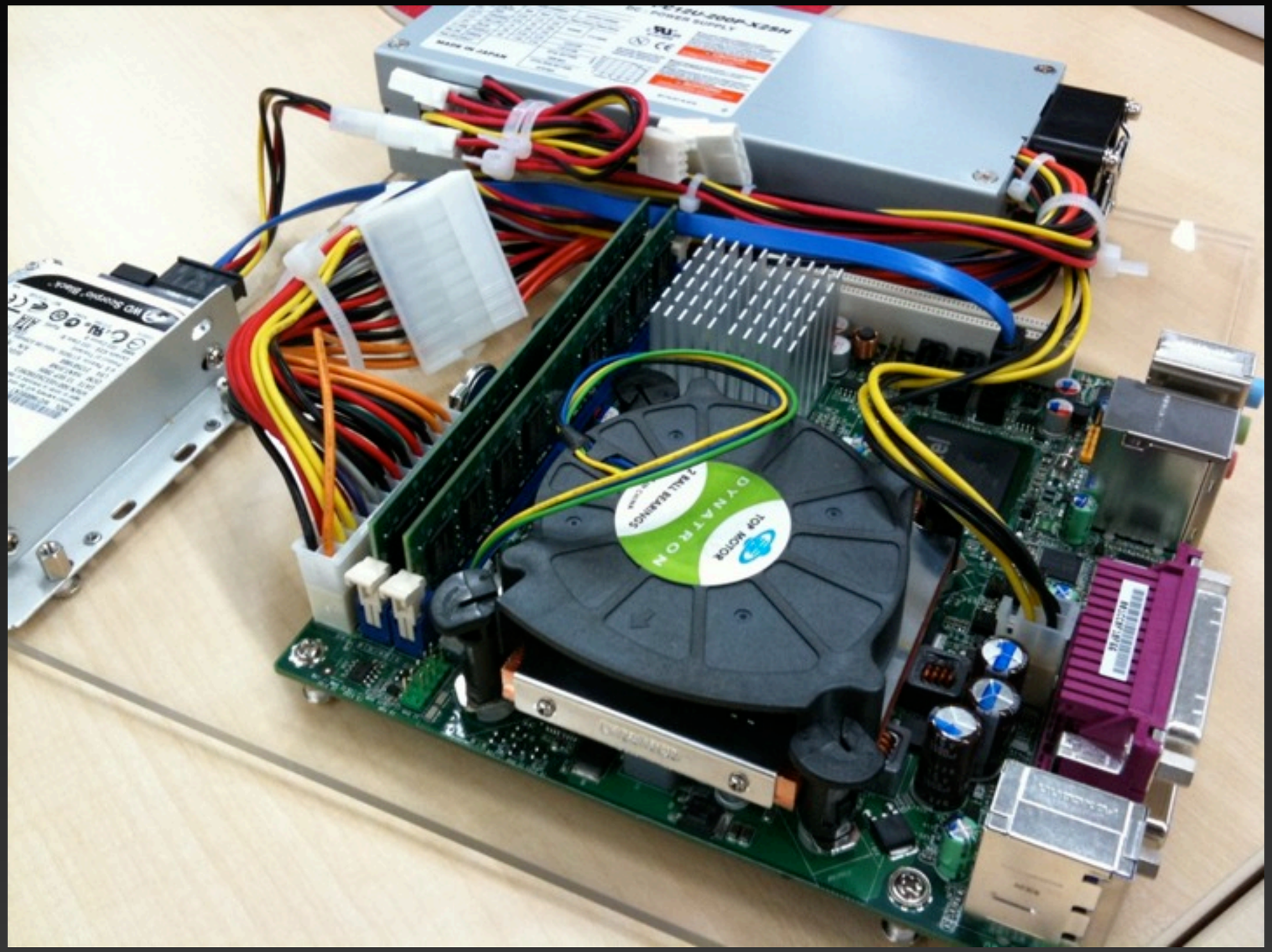
サーバー構成

Ameba

サーバー構成



Ameba



Ameba

Intel Core2 Quad™ Q8400S

4GB

160GB 7200rpm

￥70,000 ～ 80,000

Ameba

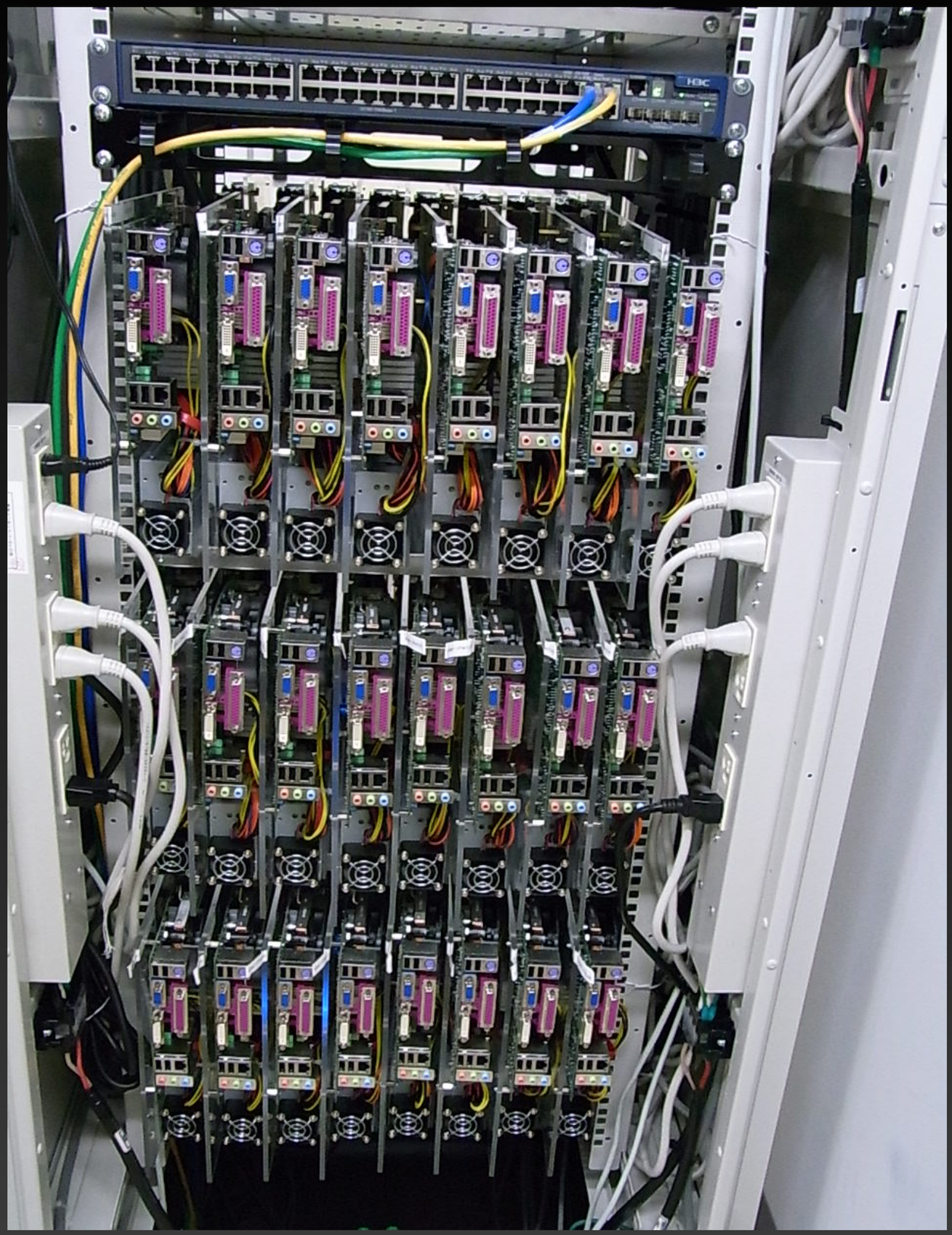
Intel ATOM™ N330

2GB

160GB 7200rpm

￥30,000 ～ 40,000

Ameba

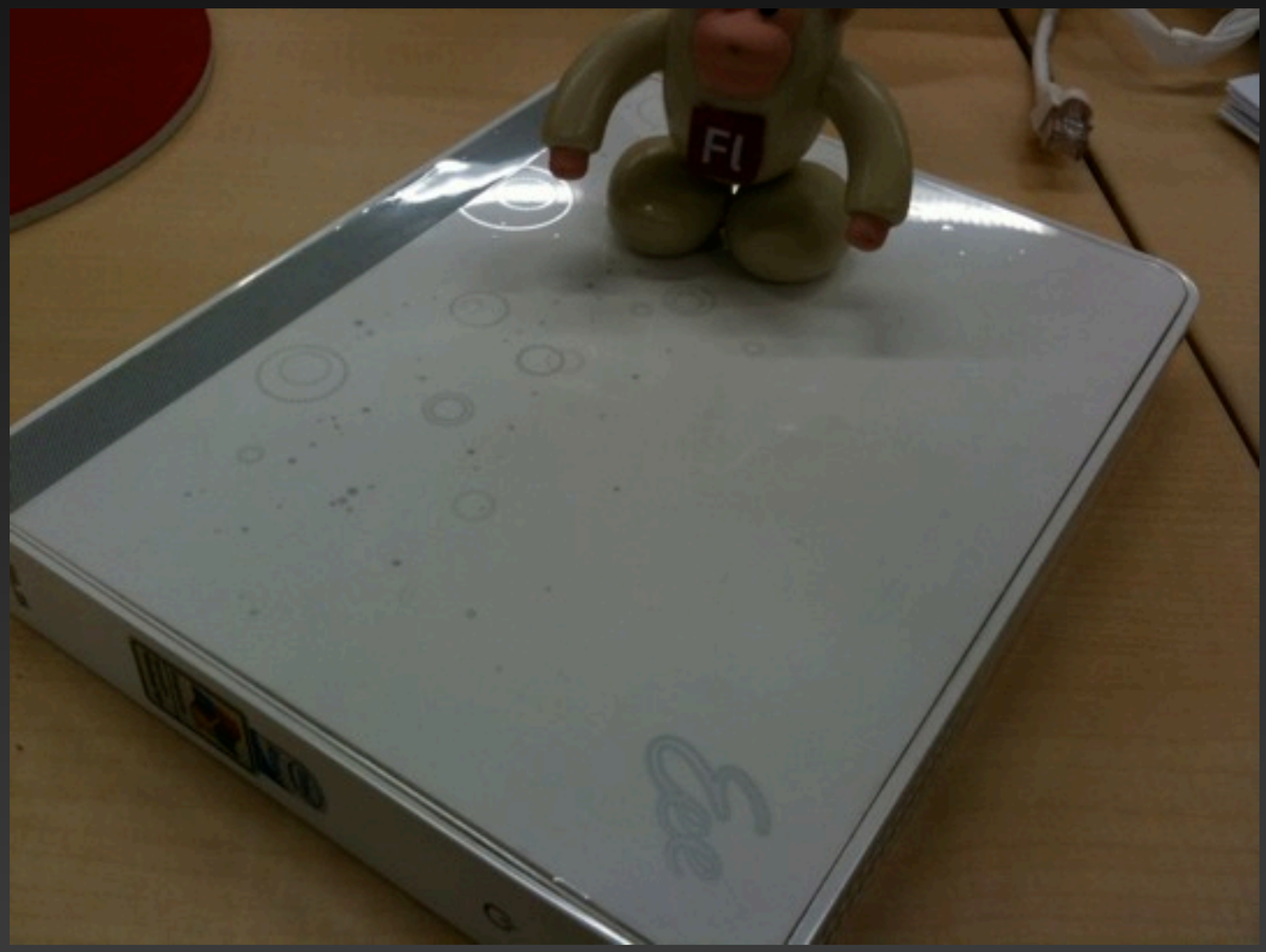

H3C

オープン時のサーバーコスト

￥2,000,000くらい

開発サーバー

開発環境は eee Box でした（自腹で 5台）



Ameba

データベース

RDBの問題点

- ボトルネックになりやすい
- IO分散がわりと面倒
- Horizontal Partitioning が面倒
- 分散戦略が、アプリの設計に影響する
- クラスターは高い(MySQL Cluster, Oracle RAC)

アメーバピグの特徴


データ更新の頻度が極端に多い


ユーザーごとにデータ範囲が独立しやすい


全体のデータに対するソートや
複雑な検索があまりない

Ameba

Key Value Storage

ソート

分散

“Ryan”	....

put

Key	Value
“Alex”	....
“Ben”	....
“Charles”	....
“Daniel”	....
“Ethan”	....
“John”	....
“Tyler”	....
“William”	....
”Zachary”	....

Node1

Ameba

Key	Value
“Alex”	....
“Ben”	....
“Charles”	....
“Daniel”	....
“Ethan”	....
“John”	....
”Ryan”	....
“Tyler”	....
“William”	....
“Zachary”	....

Node1

Ameba

Key	Value
“Alex”	....
“Ben”	....
“Charles”	....
“Daniel”	....
“Ethan”	....

Node2

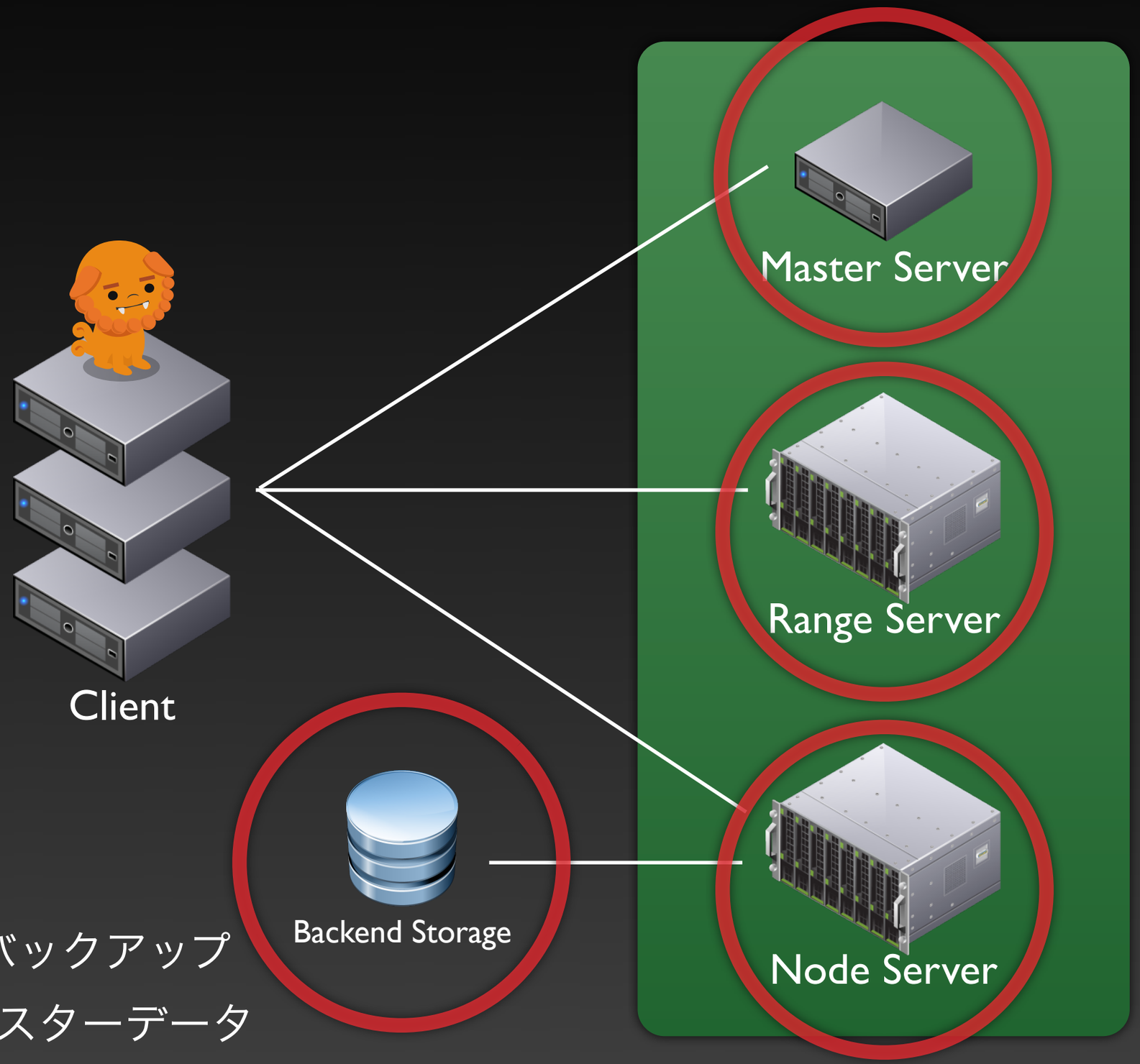
Key	Value
“John”	....
”Ryan”	....
“Tyler”	....
“William”	....
“Zachary”	....

Node3

ポイント

- キーがソートされることによって、レンジサーチが可能になる
- 一般的なRDBのインデクスに相当する機能がKVSで利用可能になる
- 自動分散によって、アプリケーションは分散を意識しなくてよい

構成



サーバーの死活監視
ノードの割り当て

ノードの配置情報

データの実態の管理

バックアップ
マスターデータ

実装に使った技術

Ameba

JDK1.6
(100% Pure Java)

Ameba

NIO フレームワーク

(java.net.nio)

Ameba

java.util.concurrent

Ameba

ConcurrentSkipListMap

ConcurrentLinkedQueue

BlockingQueue

AtomicInteger

ReentrantLock

Ameba

BerkeleyDB Java Edition

Ameba

Live Coding

Ameba

Persister
POJO Object

```
@Persistable(“pigg_user”)
public class User {

    private int userId;
    private String nickname;

    @IndexKey
    @Store(index=0)
    public int getUserId() {
        return userId;
    }

    @Store(index=1)
    public String getNickname() {
        return nickname;
    }
}
```

Ameba

Persister
Load/Save

```
@Autowired
private IndexPersister persister;

public void saveUser(User user) {
    persister.save(user);
}

public User getUser(int userId) {
    return persister.load(userId, User.class);
}
```

Ameba

Persister
Fetch

```
@Autowired
private IndexPersister persister;

public List<UserItem> getUserItem(int userId) {
        IndexQuery query = IndexQuery.startsWith(userId);

        List<UserItem> itemList =
            persister.fetch(query, UserItem.class);

        return itemList;

    }
```

Ameba

オープンソースでは

Ameba

MongoDB

MckoiDDB

HBase(Hadoop)

データ解析

Ameba

Map Reduce (Hadoop)

Ameba

Sample Code

Ameba

ソケットサーバー

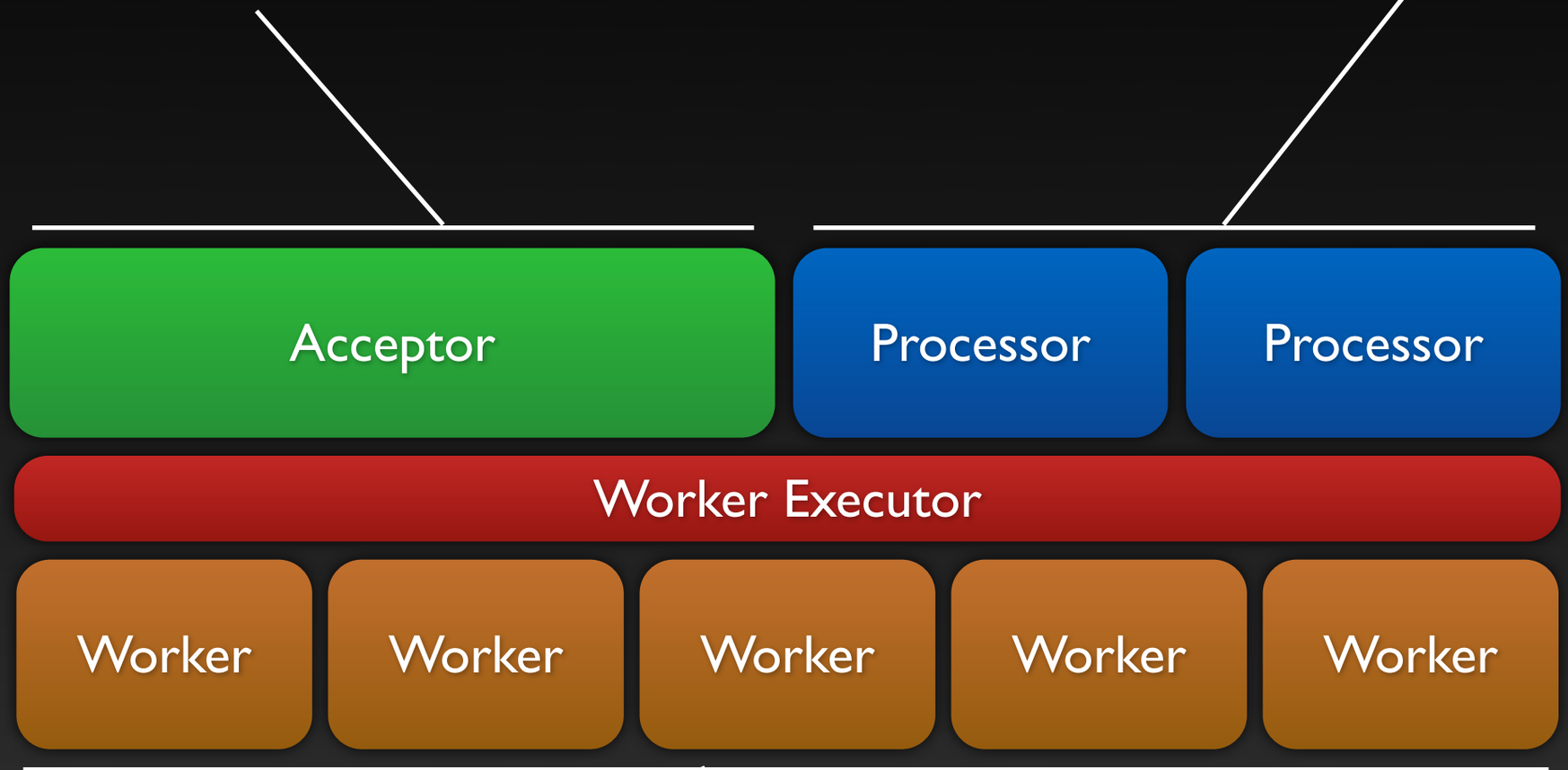
Ameba

Flash Player との
ソケット通信

Ameba

接続の待受

通信データの送受信



通信内容に応じたロジック処理を実行

Ameba

Flashが接続する
ソケットサーバーは
2 種類にわけた

Info Server

- ユーザーのオンライン情報の管理
- ピグを開いている間は、常時接続
- ユーザーの持ち物などの基本情報の配信
- メッセンジャーや、情報通知の配信

Chat Server

- 仮想空間のエリア情報を管理
- エリア移動ごとに接続を切り替える
- チャットメッセージの配信
- 同一エリア内のユーザーの動きを配信

Ameba

Info Server

Chat Server

接続

ログイン

エリア入室

エリア割り当て

割り当て完了

エリアサーバー情報

接続

入室

エリア情報

実装のポイント

- できるだけオンメモリでデータを持つ
- 常時接続を活かして非同期で永続化する
- 情報配信は全てOSに任せて非同期化

  SocketChannel.write(ByteBuffer src);
- Executors, ExecutorService は、 並列処理でとても便利

Ameba

オープンソースで似ている技術

Ameba

Apache MINA
JBOSS netty
Grizzly
xSocket

Ameba

Flash と Java

HTTP (80)
画像, SWF

バイナリ (1935)
各種情報

Ameba

ActionScript 3.0

flash.net.Socket

Ameba

ActionScript 3.0

flash.utils.ByteArray

Ameba

オブジェクトの直列化

Ameba

```
out.writeUTF(“Test”);
out.writeInt(3000);
out.writeNumber(100.5);
out.writeInt(28);
```

```
var name:String = in.readUTF();
var value1:int = in.readInt();
var value2:Number = in.readDouble();
var size:int = in.readShort();
```

Ameba

Live Coding

Ameba

Flash → Java

TalkData.as

Ameba

```
public class TalkData implements CommandData
{
  private var message:String;

  public function TalkData(message:String) {
    this.message = message;
  }

  public function readData(in:ByteArray):void
  {
    message = in.readUTF();
  }

  public function writeData(out:ByteArray):void
  {
    out.writeUTF(message);
  }
}
```

Ameba

```
var client:CommandClient = new CommandClient();
client.open(host, port);

var data:TalkData = new TalkData(“Hello!”);
client.sendCommand(data);
```

TalkData.java

Ameba

```
public class TalkData implements CommandData {

  private String message;

  private void readData(ByteBuilder in) {
    message = in.getUTF();
  }

  private void writeData(ByteBuilder out) {
    out.putUTF(message);
  }
}
```

TalkCommand.java

Ameba

```
public class TalkCommand<TalkData>
        implements Command<TalkData> {

  public void execute(
       CommandRequest<TalkData> request,
       CommandResponse reponse) throws Exception{

     TalkData data = request.getData();
     response.reply(new TalkResultData(“World!”));

  }
}
```

Ameba

Flash ← Java

Ameba

Delegate

ChatDelegate.as

Ameba

```
public class ChatDelegate
{
  public function ChatDelegate() {}

  public function onTalk(data:TalkResultData):void
  { ... }

  public function onMove(data:MoveResultData):void
  { ... }
}
```

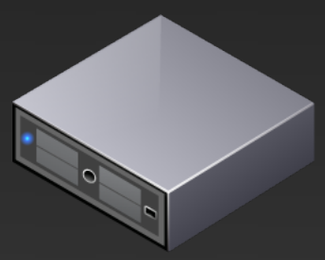
```
var delegate:ChatDelegate = new ChatDelegate();

var client:CommandClient = new CommandClient();
client.delegate = delegate;
client.open(host, port);
```

Ameba

Crossdomain サーバー

843番ポート


```
<policy-file-request/>\0
```



```
<cross-domain-policy>
<allow-access-from’*’ to-ports=’*’/>
</crossdomain-policy>\0
```

実際は改行なし

Ameba

```
byte[] content = “<crossdo.../>\0”.getBytes();

ServerSocket serverSocket = new ServerSocket();
serverSocket.bind(new InetSocketAddress(843));

while (true) {
   Socket socket = serverSocket.accept();
   OutputStream out = socket.getOutputStream();
   out.write(content);
   out.close();
}
```

Ameba

開発運用について

Ameba

4-1 運用ツール紹介

SVN

- デザイナやFlasherも利用しやすい
- チケット管理ツールと連動
- flaファイルとかたくさんコミットする
  と、リポジトリ壊れる(version古い？)

- eclipse + Subversive + SVNKit
- 個人的には git にしたい

Ameba

Maven2

- モジュールの依存管理が楽
- リリース関連の処理が自動化
- jar ファイルをコミットしなくていい
- Continuous Build と相性がよい
- Flash もビルドできる (FlexMojo)
- Eclipse IAM (q4e) が安定してきた

Ameba

Hudson

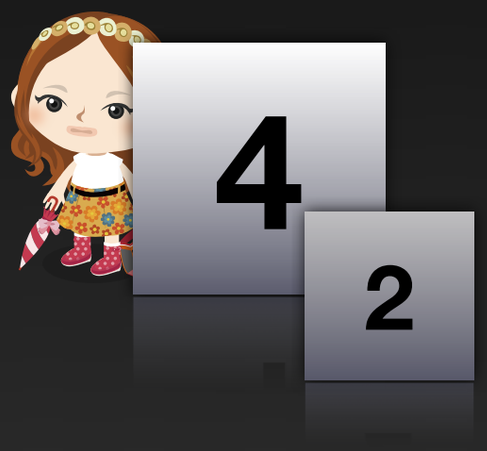
- 定期ビルドとテスト結果の保存
- Maven2 と相性がよい
- リリース処理もできる
- 定期運用ならなんでもできる

Ameba

Jopr

- Java製監視ツール
- 大量サーバー時にグラフが見やすい
- 設定が半自動
- 公式ドキュメントはそこそこ充実
- ネット調べても情報が少なすぎる
- カスタムモジュール作るの面倒

Ameba



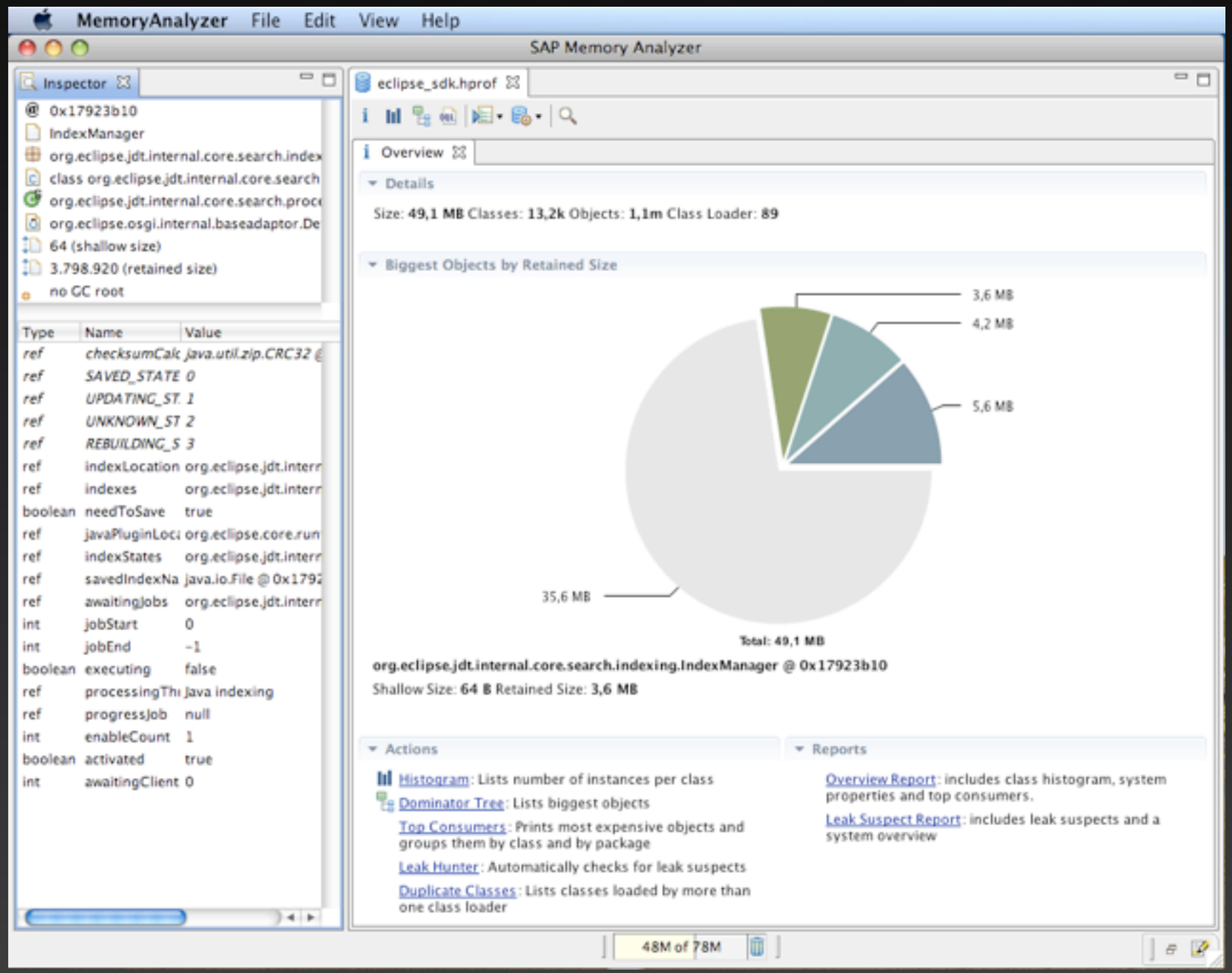
Javaサーバーの運用TIPS

Ameba

Memory Leak

Ameba

Eclipse Memory Analyzer



Heap Dump

稼働中のプロセスのダンプをとる

```
% jmap -dump:format=b,file=/tmp/heapdump.hprof $PID
```

JVM起動ユーザーで実行しないと取得できない

Heap Dump

OutOfMemory を出したときにダンプ取得

JVM起動オプションに下記を追加

```
-XX:+HeapDumpOnOutOfMemoryError
```

Ameba

Dead Lock

Ameba

HTTP経由のJMX監視ツール

OS

Arch	Name/Version	Processors	Load Average
amd64	Linux / 2.6.9-67.ELsmp	4	1.10

VM

Name	Vendor	Version	Uptime
Java HotSpot(TM) 64-Bit Server VM	Sun Microsystems Inc.	10.0-b23	1,324.01 hour

Memory

Type	Init	Used	Committed	Max
Heap	384.00 MB	834.30 MB	941.81 MB	1,109.38 MB
Non Heap	23.19 MB	22.66 MB	24.31 MB	132.00 MB

Memory Pool

Name	Init	Used	Commited	Ma
Code Cache	2.44 MB	3.21 MB	3.31 MB	48.0
PS Eden Space	96.00 MB	7.16 MB	77.88 MB	77.9
PS Survivor Space	16.00 MB	0.00 MB	25.06 MB	25.0
PS Old Gen	256.00 MB	827.25 MB	838.88 MB	1,024.0
PS Perm Gen	20.75 MB	19.45 MB	21.00 MB	84.0

Garbage Collection

Name	Count	Time
PS Scavenge	73,295.00	8,317.73 sec
PS MarkSweep	5,073.00	26,468.76 sec

Threads

Thread List

ID	Name	Status	Blocked Count/Time	Waited Count/Time	Lock Owner
58	Thread-43 java.lang.Thread.sleep (Thread.java:-2) sun.net.httpserver.SelectorCache$CacheCleaner.run (SelectorCache.java:96)	TIMED_WAITING	0 / -0.00 sec	1 / -0.00 sec	
57	pool-5-thread-1 sun.misc.Unsafe.park (Unsafe.java:-2) java.util.concurrent.locks.LockSupport.park (LockSupport.java:158) java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.await (AbstractQueuedSynchronizer.java:1,925) java.util.concurrent.LinkedBlockingQueue.take (LinkedBlockingQueue.java:358) java.util.concurrent.ThreadPoolExecutor.getTask (ThreadPoolExecutor.java:946) java.util.concurrent.ThreadPoolExecutor$Worker.run (ThreadPoolExecutor.java:906) java.lang.Thread.run (Thread.java:619)	WAITING	5 / -0.00 sec	23,856,391 / -0.00 sec	
56	pool-3-thread-1 sun.misc.Unsafe.park (Unsafe.java:-2) java.util.concurrent.locks.LockSupport.park (LockSupport.java:158) java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.await (AbstractQueuedSynchronizer.java:1,925) java.util.concurrent.LinkedBlockingQueue.take (LinkedBlockingQueue.java:358) java.util.concurrent.ThreadPoolExecutor.getTask (ThreadPoolExecutor.java:946) java.util.concurrent.ThreadPoolExecutor$Worker.run (ThreadPoolExecutor.java:906) java.lang.Thread.run (Thread.java:619)	WAITING	201 / -0.00 sec	27,866,488 / -0.00 sec	
55	pool-1-thread-1 sun.misc.Unsafe.park (Unsafe.java:-2) java.util.concurrent.locks.LockSupport.park (LockSupport.java:158) java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.await (AbstractQueuedSynchronizer.java:1,925)	WAITING	184 / -0.00 sec	39,438,982 / -0.00 sec	

アメリカ向けピグ

Ameba

Amazon EC2

サーバーが 1分で起動する

ツールが充実

結構高い

CPU 1個あたりは貧弱

日本からNW遠い

Ameba

Amazon S3

アイテムごとにURLが発行される

SSLも対応している

独自ドメインでも利用できる

アクセス制限がちょっと面倒

そのまま CDN に配信できる

Ameba

Amazon SimpleDB

文字列だけ

書き込んでもすぐに反映される保証がない

ページングを実装するのが難しい

Ameba

Amazon RDS

ハード性能をオンラインで変更できる

レプリケーションは対応予定

バックアップは、停止時間がある

Ameba

Facebook Application

仕組みがよくできている

Flashとも相性がよい

API仕様変更が多い

ローカルテストが難しいケースがある

質疑応答

ご静聴

ありがとうございました