

『Rubyによる基幹業務開発』

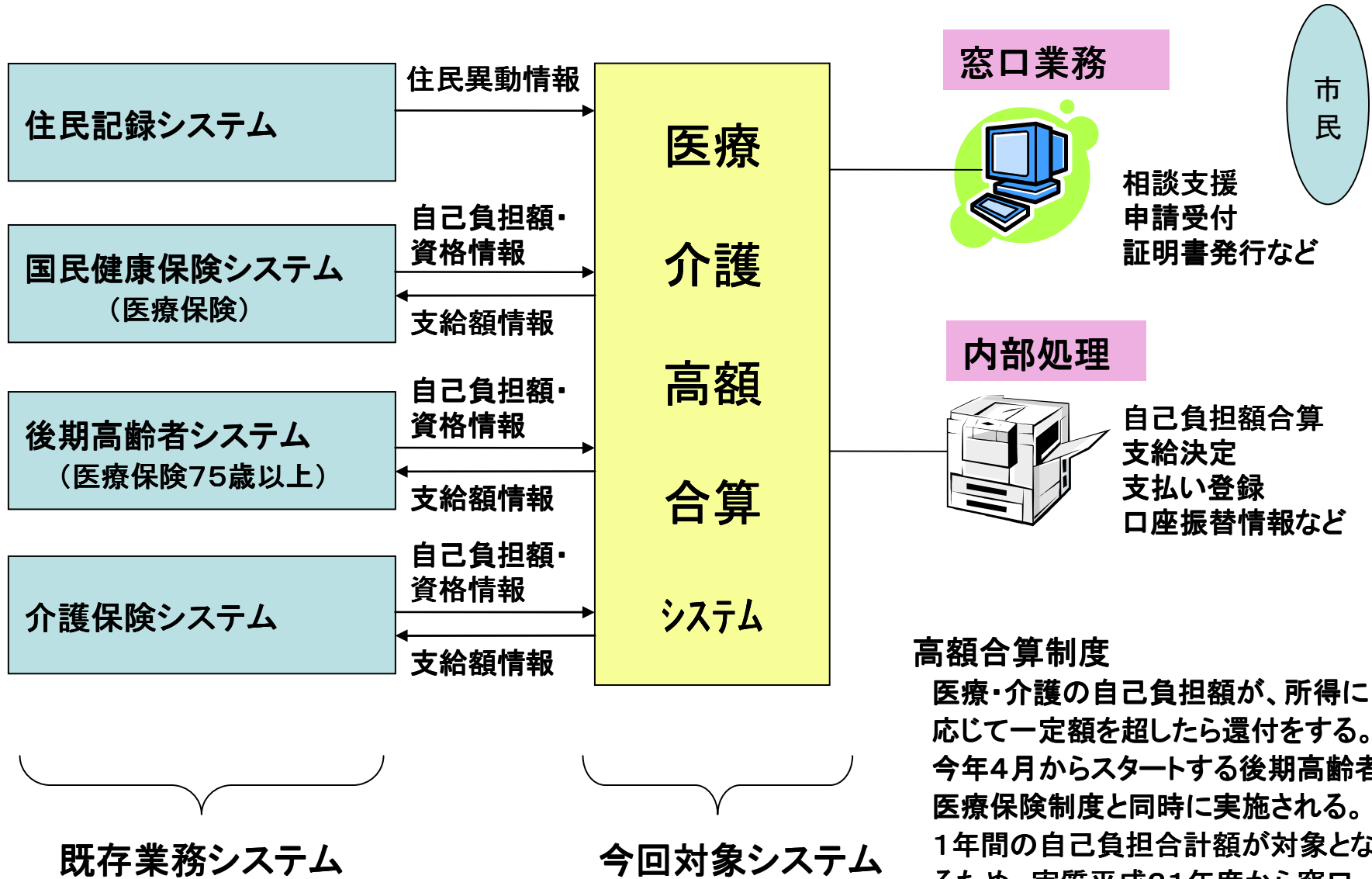
～松江市殿での高額合算システムの開発を通して～

2008. 9. 13

(株)テクノプロジェクト
吉岡 宏 高木 丈智

IPA公募事業

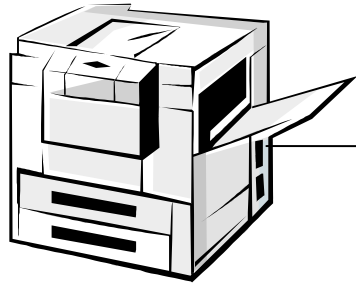
①医療・介護高額合算システム



IPA公募事業

②Rubyの新しい取り組み

- 大量のデータ処理
- 精度の高い帳票印刷
- 再印刷やページ指定印刷
- 夜間の無人運転支援
- プログラムの保守性向上
- COBOLエンジニアのシフト



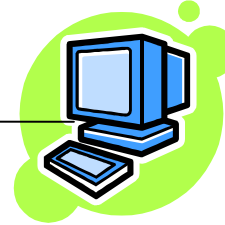
今回、新たに取り組む分野



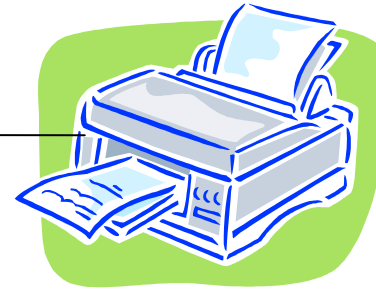
基幹業務システム(大規模システム)への拡大

高額
合算
システム

画面からのデータ登録



少量のデータ処理



これまでに実績のある分野

設計方式

Ruby ⇒ オブジェクト指向言語 ⇒ オブジェクト指向設計(UML)
COBOL ⇒ 手続き型言語 ⇒ 手続き型(構造化)設計
今回 手続き型(構造化)設計 … 業務フロー図、画面・帳票、機能設計書(詳細)

開発方式

ウォーターフォールを採用 ← アジャイルではない。
しっかり設計をやり、工程毎に顧客レビューを実施

プロジェクト体制

開発期間:2007. 9~2008. 2 オン:14本、バッチ:24本

業務開発チーム(TPJ:テクノプロジェクト・マツケイ中心)

50代SE 2名、30代SE/PG 3名、20代PG 3名

業務分析:顧客要件整理(含む厚生労働省の新制度説明資料)

設計(業務フロー、画面、帳票、詳細機能説明書)

開発(プログラミング、単体テスト、ソースレビュー)

テスト(結合テスト、システムテスト=実証実験)

共通技術チーム(NaCl:ネットワーク応用通信研究所中心+TPJ)

Ruby教育、プロトタイプ作成、コーディング規約作成&ソース点検

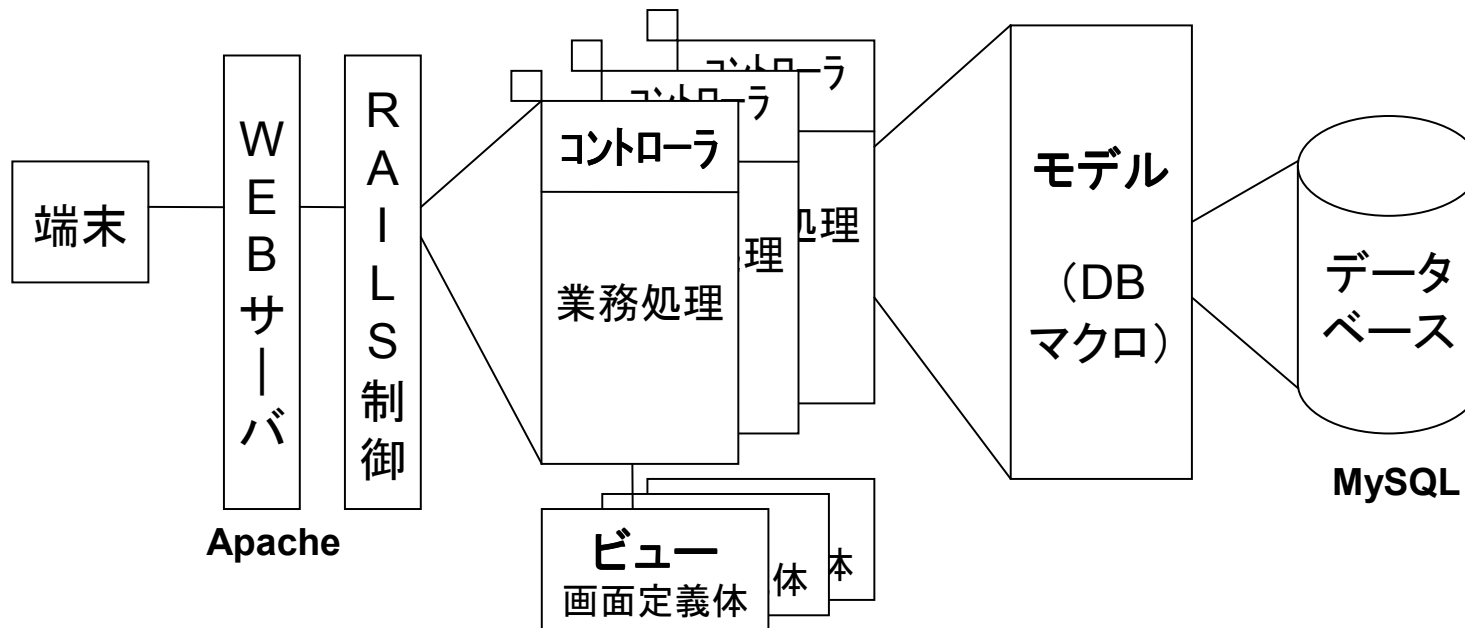
ツール作成(大量帳票印刷・バッチ自動運用)

ライブラリの組み込みと使用方法整理…祝日判定・DB複合キーなど

共通ルーチン作成(部品化)、開発環境整備

オンライン開発(画面)

Ruby on Rails(機能分割:モデル・ビュー・コントローラ)



※Ruby on Rails

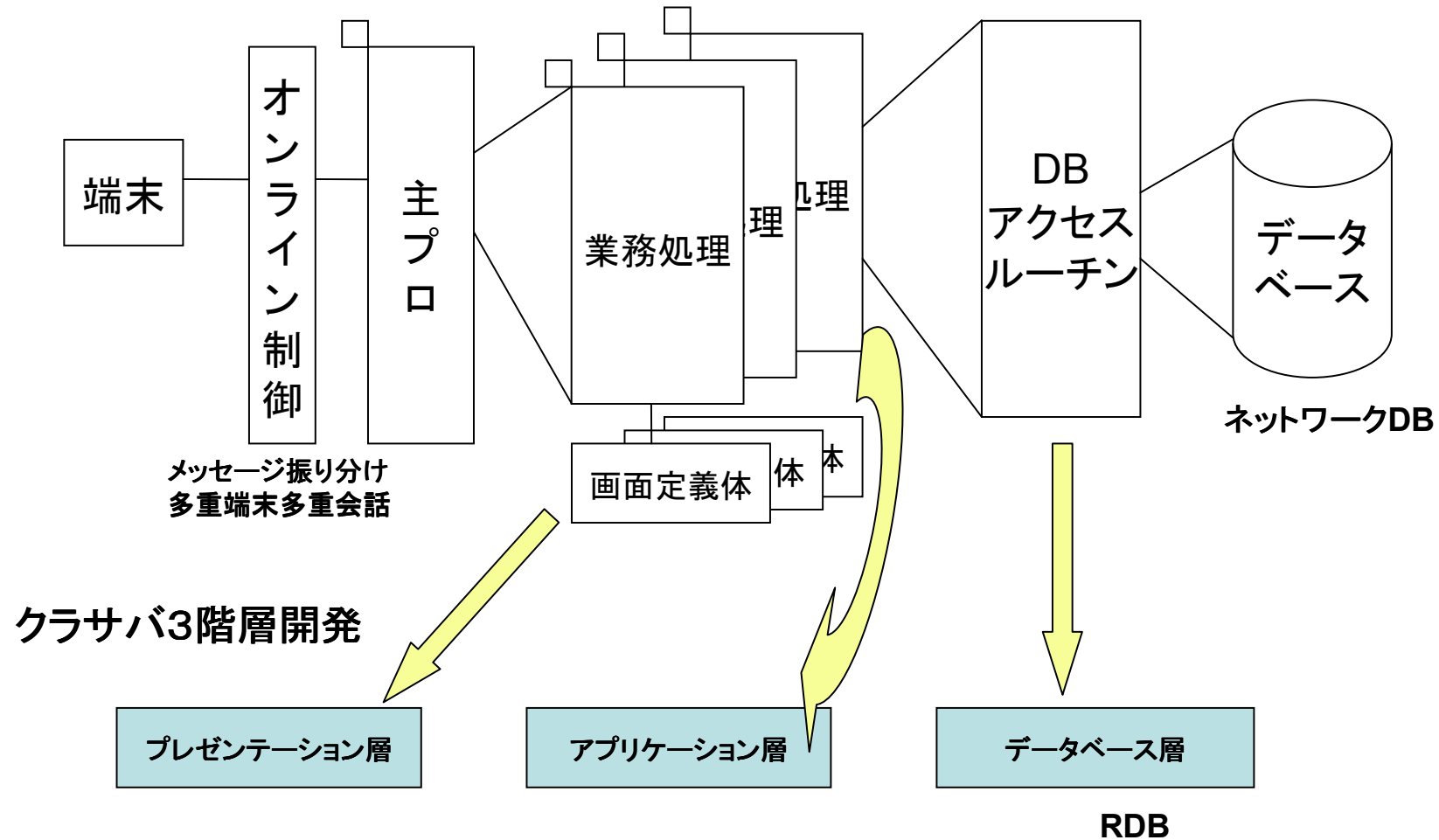
- ・モデル(DBマクロ) 検索・挿入・更新・削除のSQL
- ・ビュー(画面定義) DBの追加・更新・削除の画面HTML
- ・コントローラ(オンライン制御)

フレームを自動生成してくれ、Webアプリが動くようになる。

※ビュー

Webのホームページ作成ツールでHTML定義のひな形を作成する。画面の統一性を出すため、1人のデザイナーが全画面作成する。プログラマーはHTML定義にプログラムとの変数を挿入し、ビューを完成させる。

レガシー開発(画面)



業務処理に集中して開発できる環境があった。
オンラインやDBで、予期せぬ動きはない。安心して、開発できる環境であった。

バッチ業務

必要性

複数部門で利用するシステムの場合、オンライン中に大量のデータ処理を実行すると ①オンラインのレスポンス悪化 ②排他制御によりオンライン停止あるいは ③デッドロックが起こる。
このため、大量のデータ処理は、オンライン終了後に実行する。

スケジューリング

他システムとのデータ受け渡しやバックアップ、当日の登録データ一覧表作成などの定常処理とその日に担当者が要求する特定帳票の出力などの処理がある。
⇒ 自動的に運用したい。実行する順番がある。

帳票出力

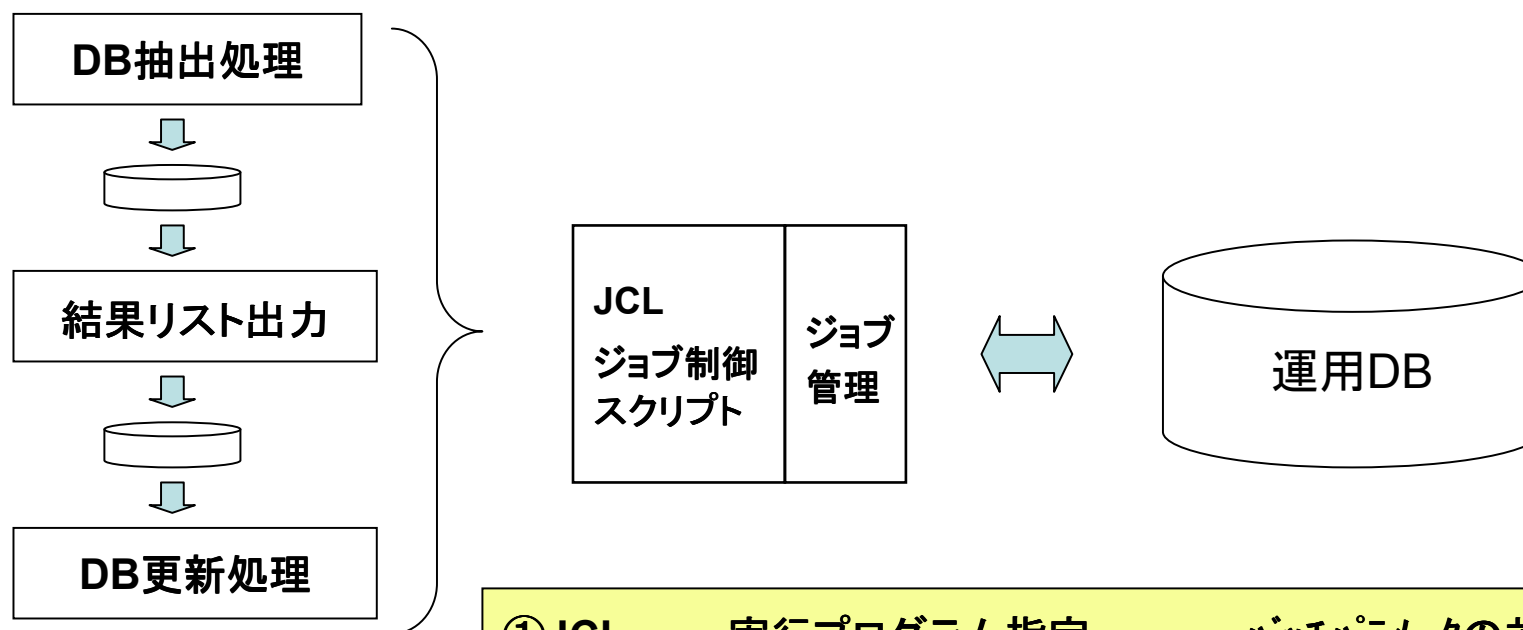
大量の帳票出力処理は、特定のページのみ出力したいケースやプリンターのトラブルのために再印刷する機能が必要である。
住民に渡す帳票なので、フォームを使用した判り易さが求められる。

バッチ業務(ジョブ管理)

- 機能
- ・バッチジョブの実行パラメータを事前に設定できること(オンライン中)
 - ・バッチ処理の実行結果の確認を翌日以降にできること(夜間自動運用)

ツール新規作成

バッチプログラムの初期処理・終了処理時に、自動運用の関数(サブルーチン)



- | | | |
|----------|-------------------|----------------|
| ①JCL | ・実行プログラム指定 | ・バッチパラメータの対応付け |
| ②プログラム | ・ログの書き出し | ・バッチパラメータの受け取り |
| | ・ワークファイルの定義(SQL) | |
| ③ジョブ管理機能 | ・実行結果の判定 | ・バッチパラメータの事前設定 |
| | ・実行結果のログ参照(ジョブ単位) | |

バッチ業務(スプール機能)

帳票印刷

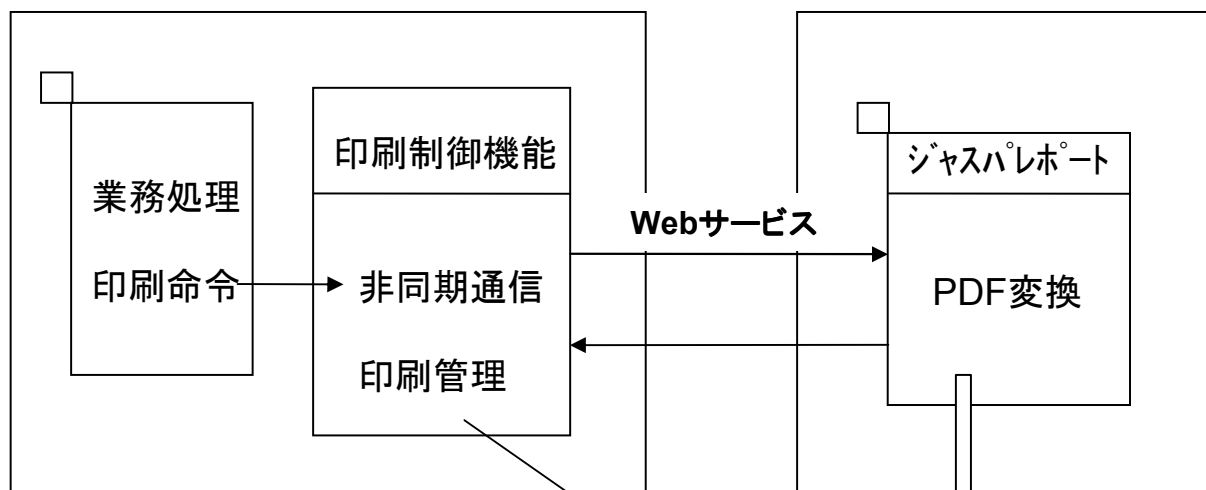
iReport と ジャスパーレポートの活用 (Javaオープンソース、フォーム作成・PDF変換)

ツール新規作成

ジャスパーレポートの運用管理機能を新規作成する。JRubyで開発。

Web・アプリサーバ

プリントサーバ

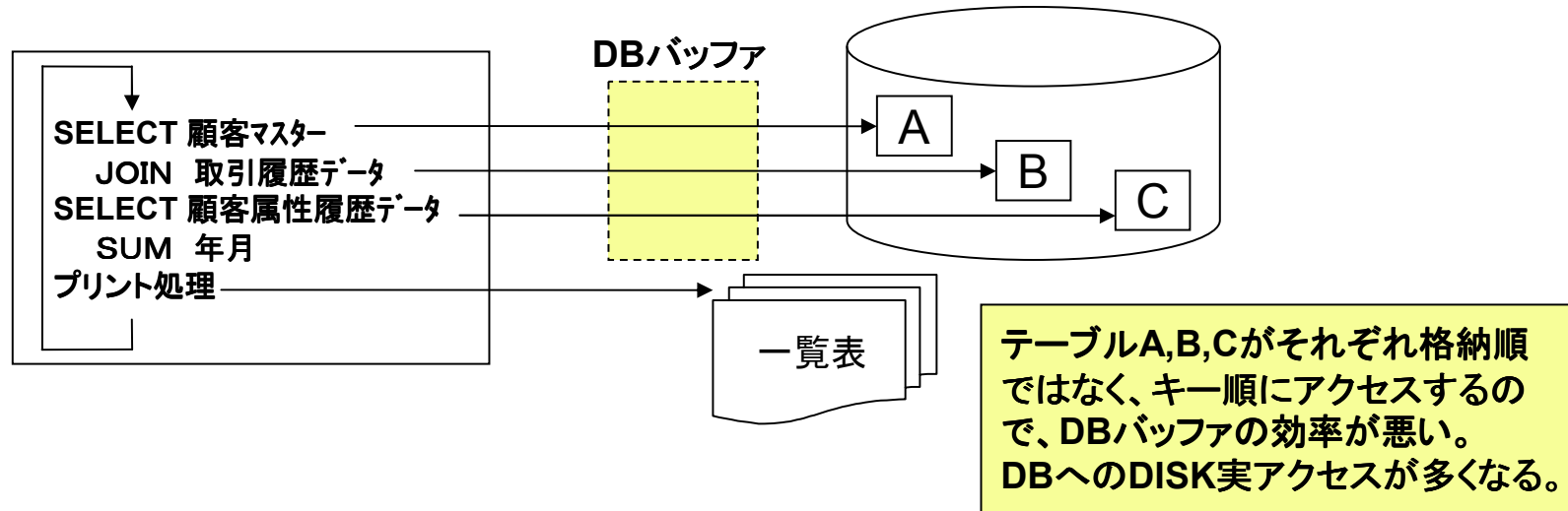


スプール機能

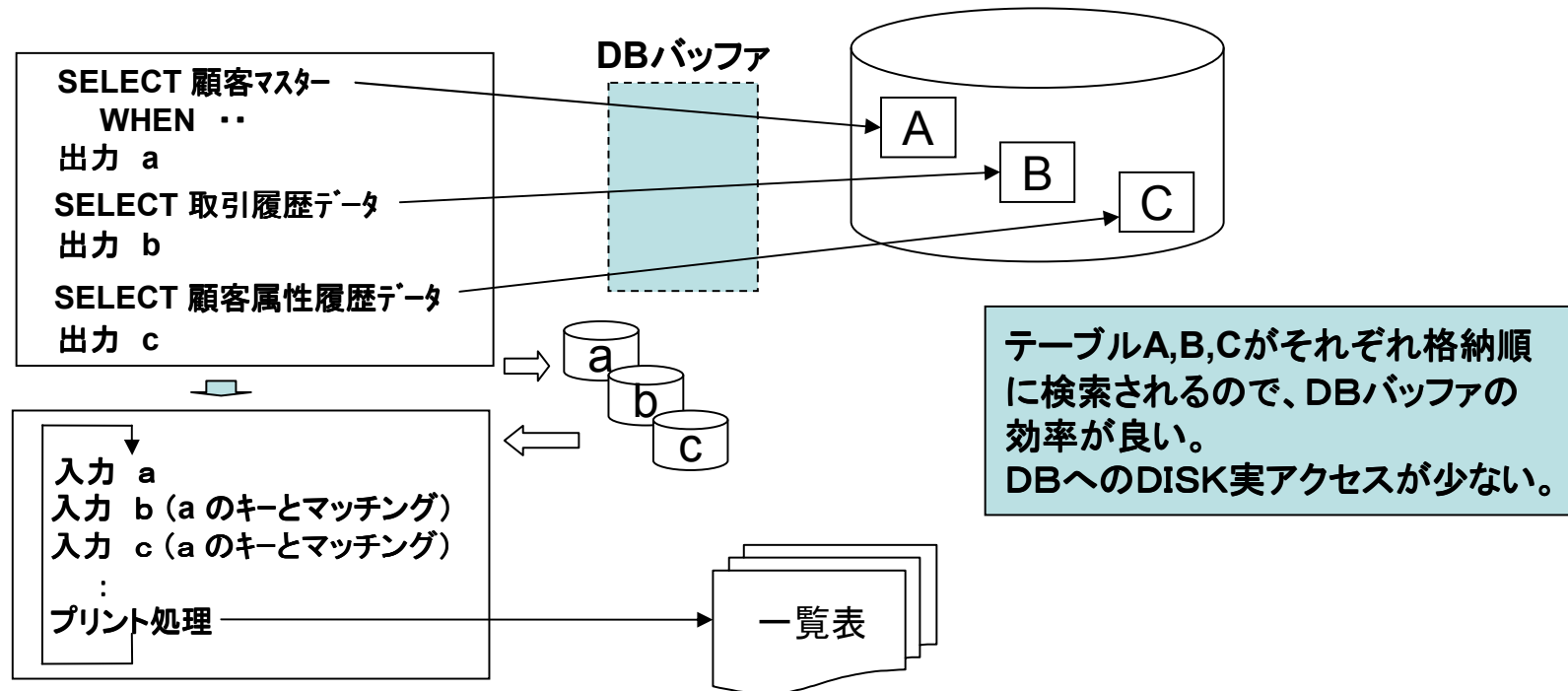
- ・プレビュー
- ・ページ指定印刷
- ・再印刷
- ・印刷中キャンセル
- ・検索・削除

バッチ業務(大量データ処理)

遅くなる例



速くなる例



バッチ業務(大量データ処理)

DISKアクセスとロジック(CPU)の処理時間比較 ... 数値は想定値

DISKアクセス1回の処理時間 : 10m秒程度 = 10×10^{-3} 秒 ← ①

サーチ時間: 7200RPM(秒当り120回転= $8\text{m秒} \times 1/2$) + シーク時間 + 転送時間

Ruby 100Stepの実行時間

= 100(Rubyステップ数) × 10(倍) × 5(マシン語のステップ数/C言語1ステップ)
× 5(クロック数/マシン語1ステップ) × 0.5×10^{-9} (秒)

= 12.5×10^{-6} 秒 ← ②

※Rubyは実測レベルで、Cの数倍から10倍の実行速度である(インタプリタ)

※CPU: 2GHz (1クロックの時間= 0.5×10^{-9} 秒)

⇒ DISKアクセスとRuby100ステップの実行速度は1000倍程度の差がある。

20年程度前と現在の比較

	CPU速度	DISK アクセス	メモリ	大量 データ	ファイル	DB チューニング	言語
20年 程度前	MHz (FLOPS)	20~40 mS	MB	数十万 ~数百万	NDB 順編成	SEノウハウ ダイナミック ステップ	COBOL PL/I
現 在	GHz	10mS	GB	数千万 ~数億	RDB	自動化 SEノウハウ	VB,Java Ruby

Ruby基幹業務開発(生産性)

①システム開発の工程別比率

設計工程(30%) + 製造工程(PG設計~単体テスト40%) + テスト工程(30%) ⇒ 全工程(100%)

②Rubyによる生産性(Rails研修後)

設計工程(30%) + 製造工程(40%) × 1/2 + テスト工程(30%) × 2/3 ⇒ 全工程(70%)

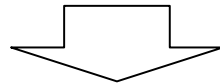
Ruby on Railsは、DBのテーブルを定義するとデータ操作のWebアプリが簡単にできる。

テストは手順を登録すると、自動テストが可能である。(障害時の再テスト・レグレッションテスト)

③Rubyによる生産性(開発を終えて、開発者の感想)

設計工程(30%) + 製造工程(40%) × 3/4 + テスト工程(30%) × 1/1 ⇒ 全工程(90%)

- ・チェック処理が多い。1つのデータ登録に、100ステップ程度の整合性チェックをする。
- ・自動テスト機能は、プログラムの状態やデータベースの状態が同じでなければ、使えない。
短期開発では、わざわざテスト手順を登録し、データをバックアップする余裕はない。
- ・部品化：設計中に共通部品の洗い出し ⇒ 多くはできない。数個程度。
Railsの組み込み部品が有効。例) 20件のデータを検索表示 → 次の20件が部品。
- ・プログラム修正から単体テストがスムーズにできる。
- ・基本的に、1度限りの開発である。繰り返しによる部品の充実が図れない。

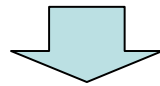


COBOL開発(ホスト・オフコン・クラサバ3階層)とRuby(Rails)は、ほぼ同じである。

Ruby基幹業務開発(メンテナンス性)

前提

- ・基幹業務は、企業活動になくてはならないシステムであり、変化に対応して長い期間利用される。ロングライフであること。
- ・長いシステムの利用の中では、プログラムの担当SEが替わり、メンテナンスが継続して行なわれる。誰でもメンテナンスできること。



管理された開発 ⇒ 可読性が良くなる

標準化=コーディング規約を厳しく設ける。

ソースレビューの徹底(スパゲティ状態にならないこと)

豊富なRubyの言語仕様を制限して利用する。

Ruby:プログラミングのたのしさ
(プログラマーが想像力豊かに、自由に組める)

封印

プロジェクトチームによるものづくりの楽しさ
基幹業務システム構築 by Ruby

Ruby基幹業務開発(その他のポイント)

①データのテーブル名・項目名 : 日本語名称を使用しない = ソースの可読性が低下する

受給申請者 → jukyu-shinsei-sha

日本語サポート(RubyとMySQLは単独でOK、Railsもモデルに事例ナシ)、HTMLの中に、日本語名称が入るとエラーとなる。Input-id="product_日本語名称" ⇒ XHTMLを利用すればOK。

Railsはテーブル名:複数形、クラスは単数形に自動変換する。

例)日本語のケース : 顧客 → 顧客s 英語のケース : Customer → Customers

②計算精度 : 固定小数点演算をサポートしている。

(RubyとMySQLでDecimal属性をサポート)

小数点2桁目を四捨五入し、小数点1桁目までを求める。

③データ交換 : スペースを詰め込む関数で対応。

レガシーや金融機関とのデータ交換に固定長のデータ形式が必要となる。

④Ajax : 全面的に採用した。

画面の一部を書き換える処理が多い。画面のチラツキを押さえた使いやすい画面を実現。

⑤コーディング規約 : インデント(字下げ)、1行の桁数80、メソッドのカッコを必須、改行コードはLF

文字列のクォートは'(シングル)、unlessは使用禁止、グローバル変数の使用、SQL記述 など

⑥データベースの主キー : 複合キー (複数の項目を組み合わせたものを主キーとする)

例) 年度+申請者番号+履歴番号

⑦オープンソースを複数種類組み合わせた開発は、初期コストが確実に安くなる。但し、それぞれでバージョンアップをするので、要注意。 ⇒ 実績のある組み合わせ

Ruby基幹業務開発(COBOL'erの復活)

業種・業務ノウハウ

・自治体の特性

住民サービス ⇒ 起算日処理が必要(申請や届け出の遅れを取り戻す)

データ修正の記録(履歴管理)

フォームを使った判りやすい帳票

職員の定期的な異動 ⇒ 他システムと同等な使い勝手

判りやすさ = 画面の階層を浅くする(細かな処理に分けない)

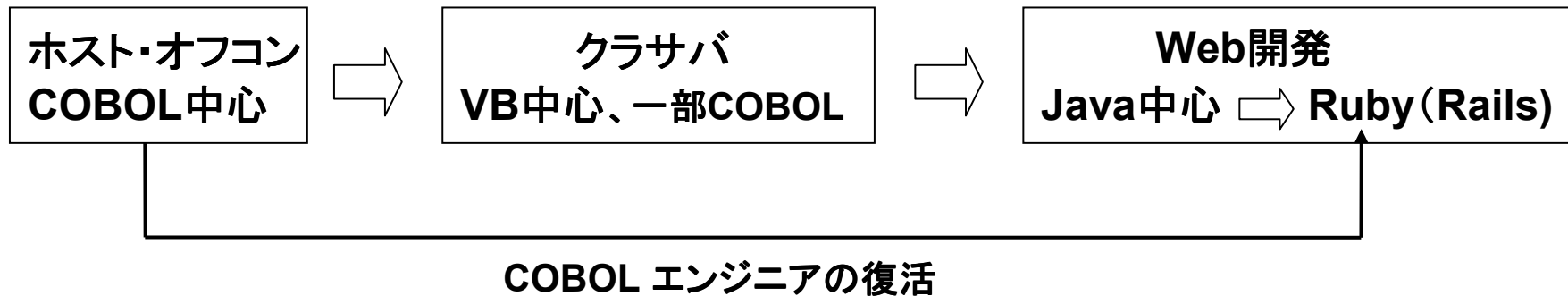
= 1つの画面が複雑になる(登録・訂正・削除が1画面)

・業務の特性

医療保険や介護保険の自己負担の還付制度と類似している。

法律のあいまいさ = 詳細な箇所が記載されていない = 厚労省とのQAがロジック化

基幹業務開発:ベテランSE(COBOL'er)のノウハウが重要



※レガシー:過去の遺産・時代遅れ

ITのトレンド

① 70~80年代(昭和) プロプライエタリイの時代

1ベンダ(メーカ)の技術で全てが提供される。 I・F・N・H
ハード+OS+ミドル+言語+システム開発 … 高額

② 90年代(平成) OPEN化の時代

クライアントサーバシステム(Windows・UNIX・データベース・言語)
プロプライエタリイの世界にマイクロソフト社,サンマイクロ社,オラクル社など
サードパーティが参入し、価格破壊が起こる。… 低価格化

③ 2000年代 オープンソースの時代

LAMP(ランプ)を活用したWebアプリの構築が主流となる。

オープンソースを代表する製品:Linux、Apache、MySQL、Perl/PHP

OS・ミドル・言語の費用がゼロに近くなる。… 無償(サポート料あり)

但し、システム開発費や運用・保守に掛かる費用は、従来通りか一部低価格となる。

Rubyは、世界に普及したオープンソースの中で、唯一の国産であり、国の応援がある。

被保険者状況照会

窓口業務担当ユーザ

2008/01/22 16:24:18

制度: 年度: 被保険者番号: 個人番号: 被保険者番号検索

年度:

氏名	被保険者番号	8	9	10	11	12	1	2	3	4	5	6	7	計
高額太一	2003170000	主	主	主	主	主								
							75:							
		34,000	49,000	0	36,000	0								119,000
	3000005300													
							75:							
						0	30,000	0	52,000	0	39,000	29,000	52,000	202,000
	4000009700													
		5,000	3,000	3,000	4,000	7,000	5,000	3,000	4,000	8,000	3,000	4,000	5,000	54,000
高額幸子	2003170000					主	主	主	主	主	主	主	主	
						30,000	80,000	72,000	37,000	45,000	23,000	40,000	52,000	379,000
	4000009800													
		30,000	5,000	2,000	3,000	2,000	4,000	5,000	3,000	4,000	2,000	5,000	3,000	68,000
高額太一郎	2003170000													
														0

前年度

国保 後期 介護

次年度

検索

確認

戻る

実行

印刷

終了



支給申請入力

窓口業務担当ユーザ

2008/01/22 16:13:25

[ログアウト](#)

被保険者状況照会

自己負担証明書申請入力

 登録
 修正
 削除

状態:

 受付
 >
 合算済
 >
 確認済
 >
 審査済
 >
 発行済
 >
 依頼中
 >
 支払済

 制度:
 年度:
 被保険者番号:
 個人番号:

 受付番号:

 被保険者名:
 区分:
 世帯集約番号:
 給付制限状況:
 所得区分:

 生年月日:
 年齢:
 性別:
 住所:

主	個人番号	氏名(氏名カナ)	生年月日	年齢	性別	制度	保険者	被保険者番号	資格加入期間情報			支給申請情報	
									取得	喪失	得喪事由	受付番号	申請日
	1000003340	高額太郎(コカ^々々)	505.04.25	77歳	男	後期	後期島根県	3000003800	09/03/26		転入による資格取	0000000006	09/11/15
						介護	介護松江市	4000008200	09/03/26		転入による資格取		
						後期	鳥取県後期	3900000200					
						介護	境港市介護	4900000300					
	1000003350	高額花子(コカ^々々)	507.05.18	75歳	女	後期	後期島根県	3000003900	09/03/26		転入による資格取	0000000033	09/11/15
						介護	介護松江市	4000008300	09/03/26		転入による資格取		

自己負担額証明書登録:

申請日: 本人電話番号: 申請者: 郵便番号: 住所: 氏名: 電話番号: 送付先: その他の送付先: 郵便番号: 住所: 氏名: 電話番号: 支払方法: 口座情報: 金融機関: 店舗: 口座種別: 口座名義人: 口座名義人カナ: 口座番号:

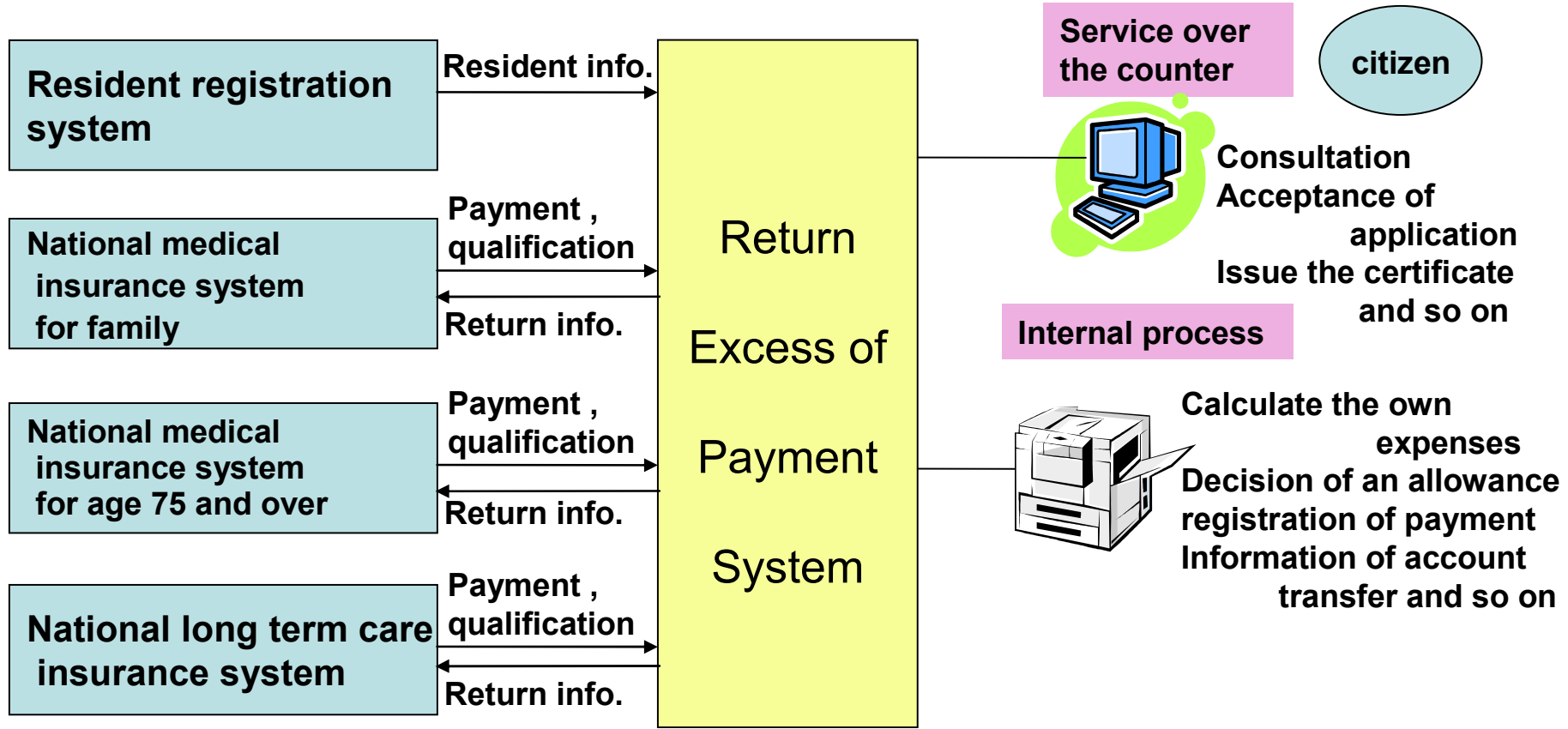


2008. 6. 23

JRuby 開発者
チャールズ・ナッター氏
松江訪問

**Ruby基幹業務開発を
プレゼン**

IPA's event (1) Return Excess of Payment System



Already existing system
 (Almost legacy system on main frame)

Target system

Back ground
 Japanese government set new values for payment of medical care and long term care expenses. When total cost of the individual payment is over the standard, return excess of payment.

Ruby development in Enterprise System (COBOLer's revival)

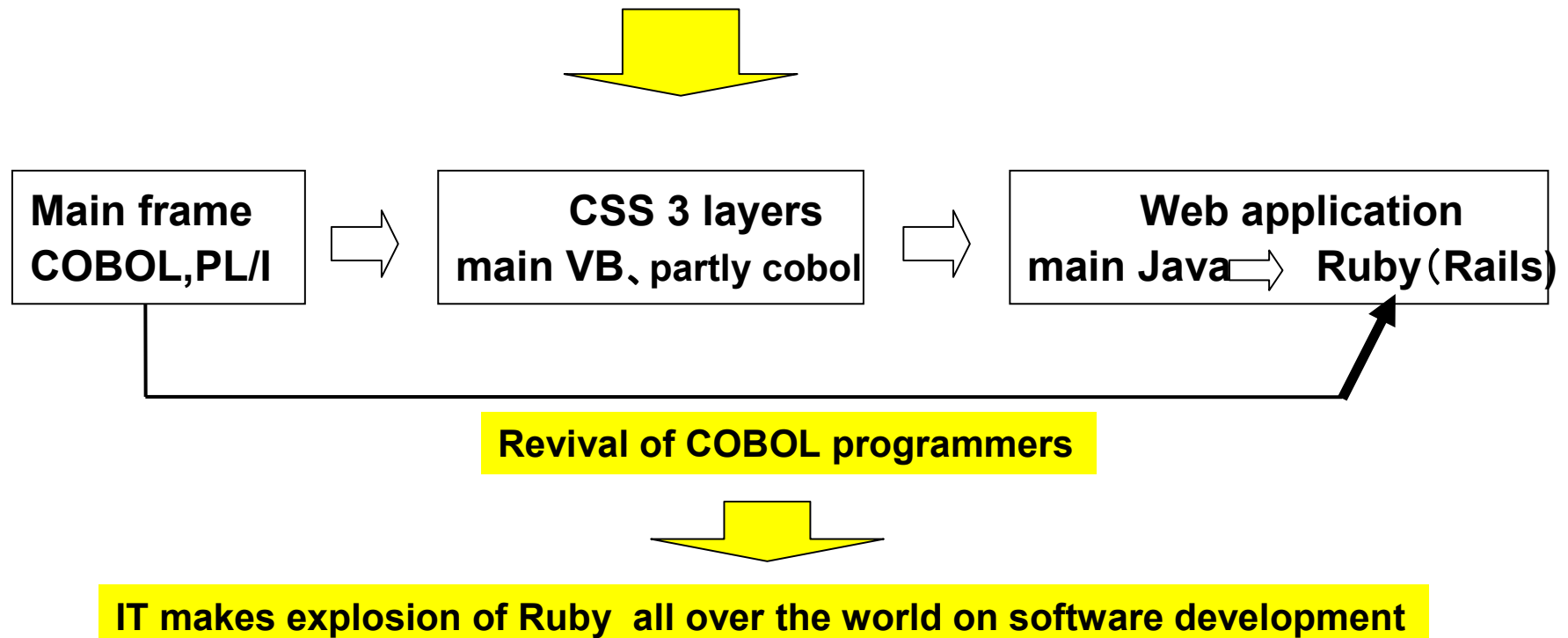
COBOL programmers

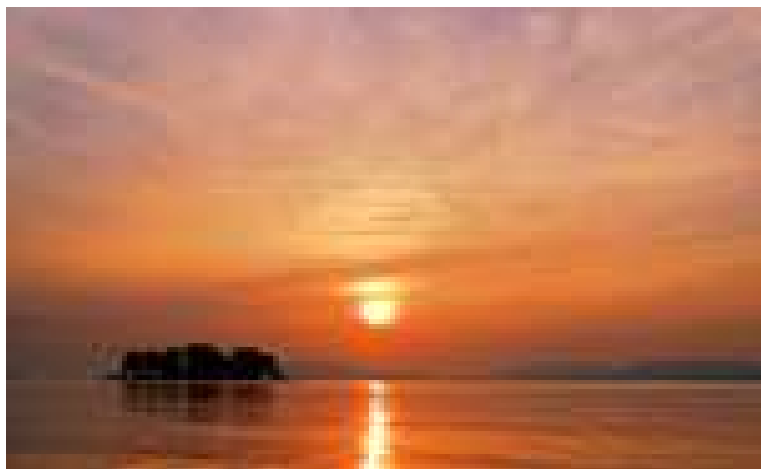
they have “Many Business Skills”.

rich experience of software development, human network of end users.

But, they are older, their technology is legacy.

Enterprise system development needs Many Business Skills





Sunset at Lake SINJIKO in Matsue City
where Matz lives.

公開ページ : www.tpj.co.jp

IPA公募事業成果「高額合算システム」

ITプロのWebマガジン:エンタープライズジン

今日の記事記事 ランキング 先頭に表示

“エンタープライズ開発の現場で感じたRubyのメリット”

<http://enterprisezine.jp/news/>

The END