



In-Process Clinical Intelligence (IPCI)紹介資料

群馬大学医学部附属病院システム統合センター
鳥飼 幸太



IPCIの実体は
「Ubuntu仮想マシン」です

考察：永続化対象と利用ユーザ維持



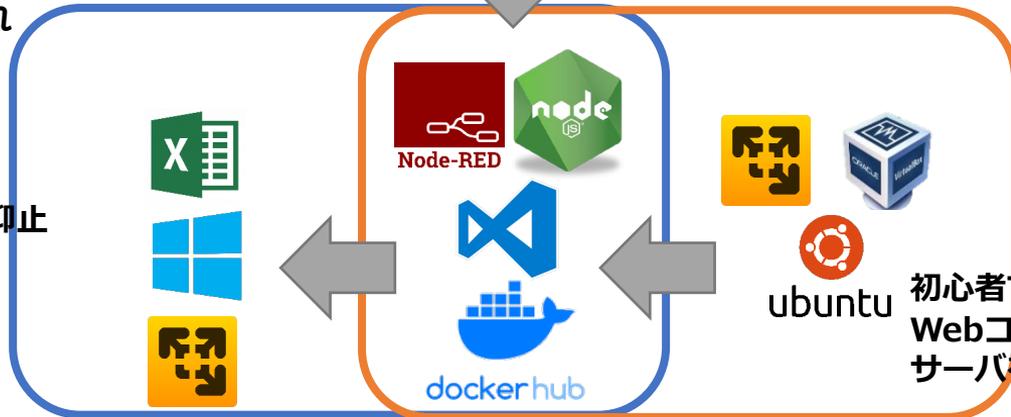
Androidは現時点では必ずしも成功した
開発プラットフォームとは言えない



Linux上で「ツールとして」Windowsを使う例はまだ少ないが、永続化視点では意味がある
LTS以降の「載せ替え」が課題→dockerまで抽象化？

データよりもExcel関数/UI/VBA資産が
Windowsで大きな位置を占めている？
→仮想化して永続化の流れ

永続的なバージョンの宣言
OS変更時のユーザー離れを抑止



Windowsは常に
端末内でのワンストップを目指す
仮想化の脅威

初心者でも理解しやすいGUIの獲得
Webコンピューティングでは多数の
サーバを必要とする→ライセンス費問題

pip/npmなどのコマンドインストーラが
再び浸透してきた



ECMAScriptはHTML/OSSと相まって
「次世代POSIX」の地位を確立



Macはハードウェアと一体の
デバイスとして領域を拡充
「使いやすいUnix」または
「ツール」または
「HTMLアプリの土台」



クラウドによる「囲い込み」
スケーリング用途での拡大
医療情報に適するか要検討

IPCIの実体 = 医療情報コンセプトに基づいて設計・実装された仮想マシン



仮想マシン (IPCI)



物理的実体 (PCやサーバ)



今回実装したものは、
VMware : 仮想化プラットフォーム
Ubuntu : OS



Node.js : スクリプティングベース
Docker : コンパイラベース



として、

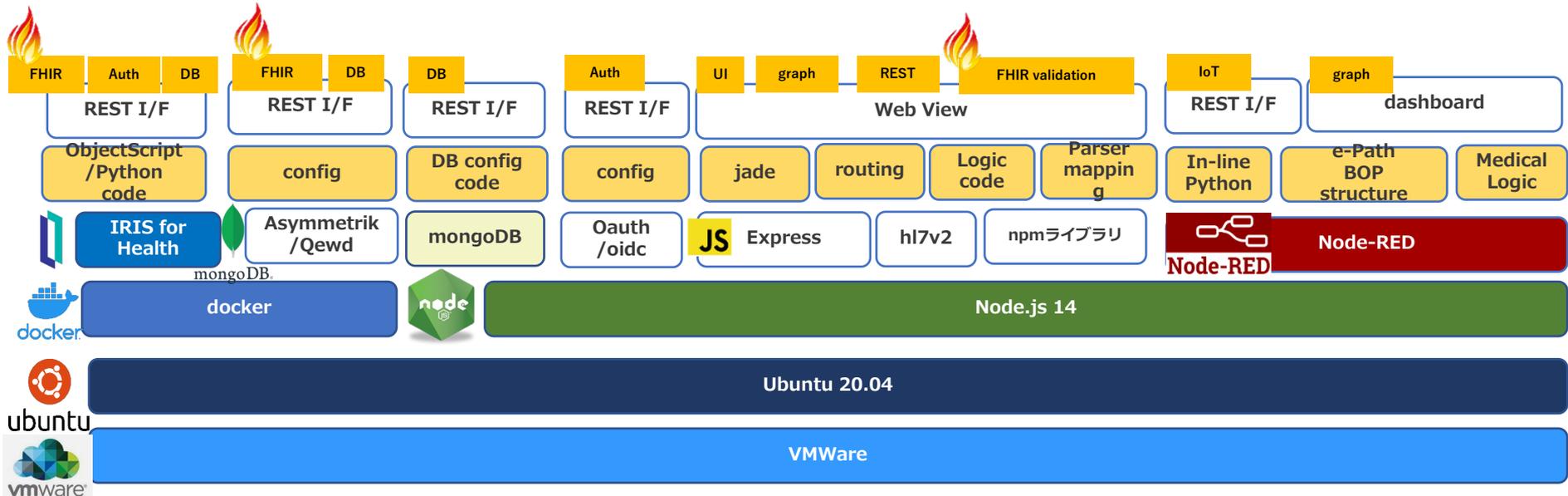
dockerhub



Node-RED : 医療ロジック・Webインターフェース
上に医療アルゴリズムを永続的に蓄積・利用するサーバ

前回チュートリアルでは、要素の一つであるNode.jsサーバの作り方を紹介した
今回チュートリアルでは、HL7 FHIRの送受信・変換サーバとしての使い方を紹介する

IPCIアーキテクチャ

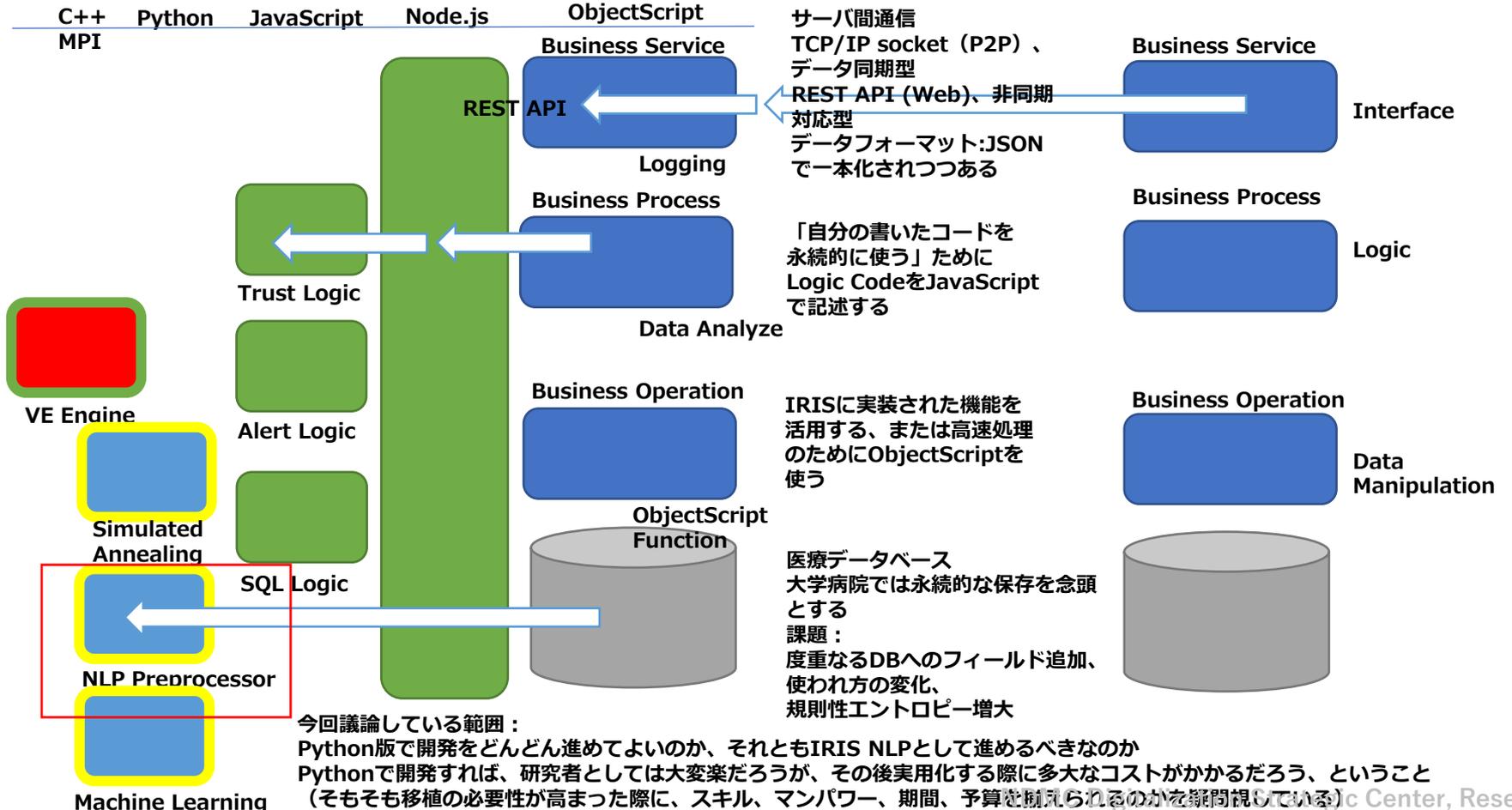


「継続的に開発できる」病院情報システムとは何か



使用言語

C++ Python JavaScript Node.js ObjectScript
MPI



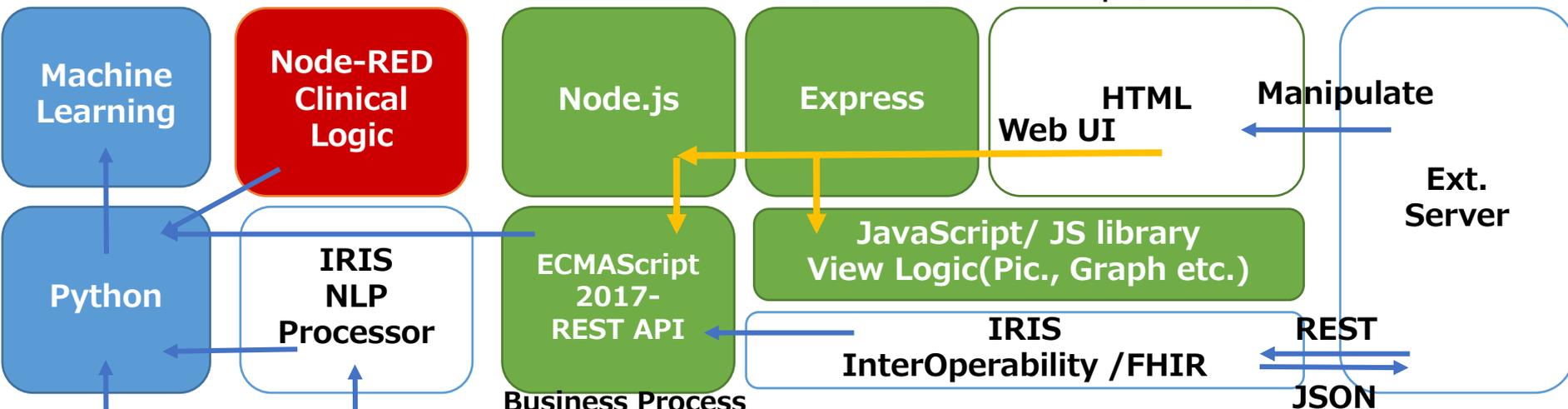
Medical Informatics/ DX System Framework

病院システムにおける In-Process Clinical Intelligence (IPCI)



ベクトルプロセッサ等接続 **クリニカルパスの記述箇所**

開発全体のグルー言語をJavaScriptでエコシステム構築



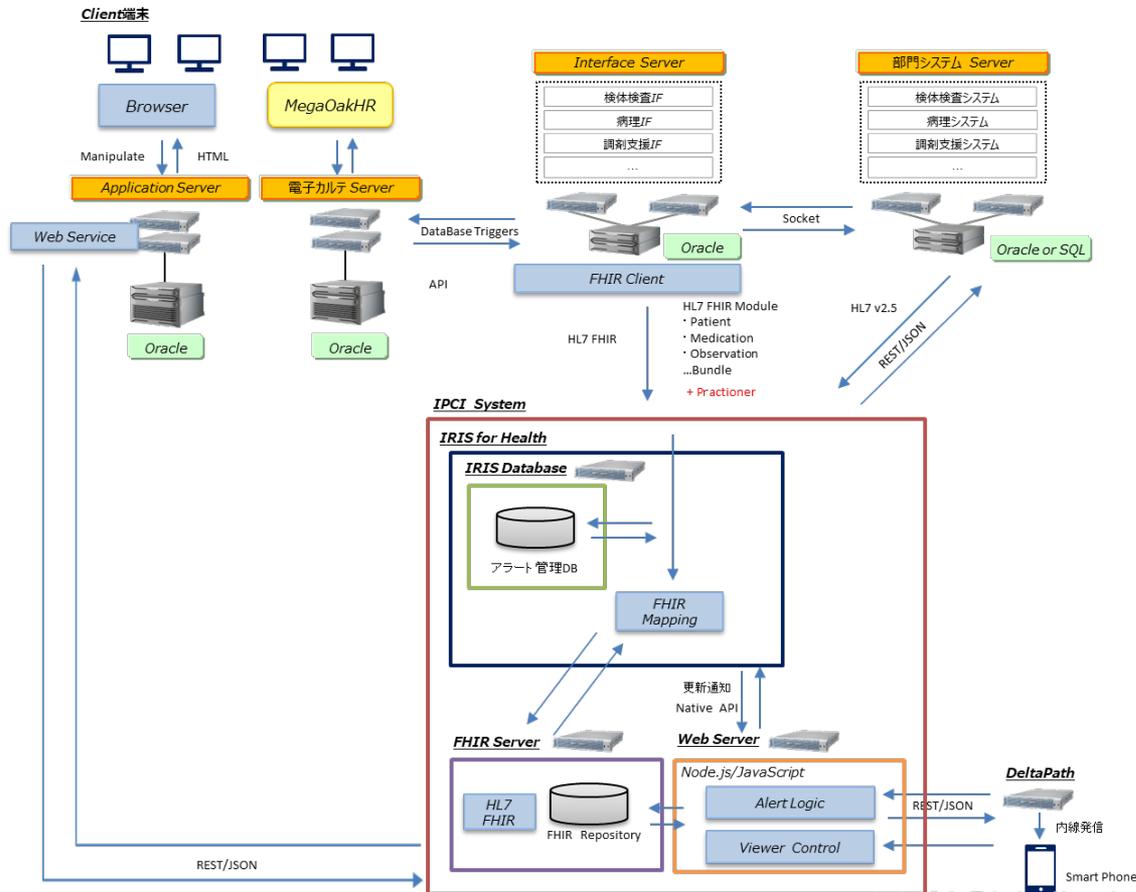
ObjectDataBase言語による
FHIRマスタバージョンアップの
「テーブル継承」概念を用いた
DB再構築の省略、データキューブ化
高速DBにOptaneを使用
不揮発性と高速性を両立

**UltraFast
DataBase**

中期作業として、MegaOak中の医療アルゴリズムをJavaScriptに
移植し、企業の知的資産として再利用
→電子カルテの段階的解体と再構築を促進
→HL7 FHIRのlabel規則でアルゴリズムをJavaScriptで構築し、
コード再利用性を格段に改善

IPCIと群大病院情報システム(HIS)との接続

2022.9-群馬大学医学部附属病院内で稼働中





IPCIの特徴

リファクタ性を重視したプラットフォーム



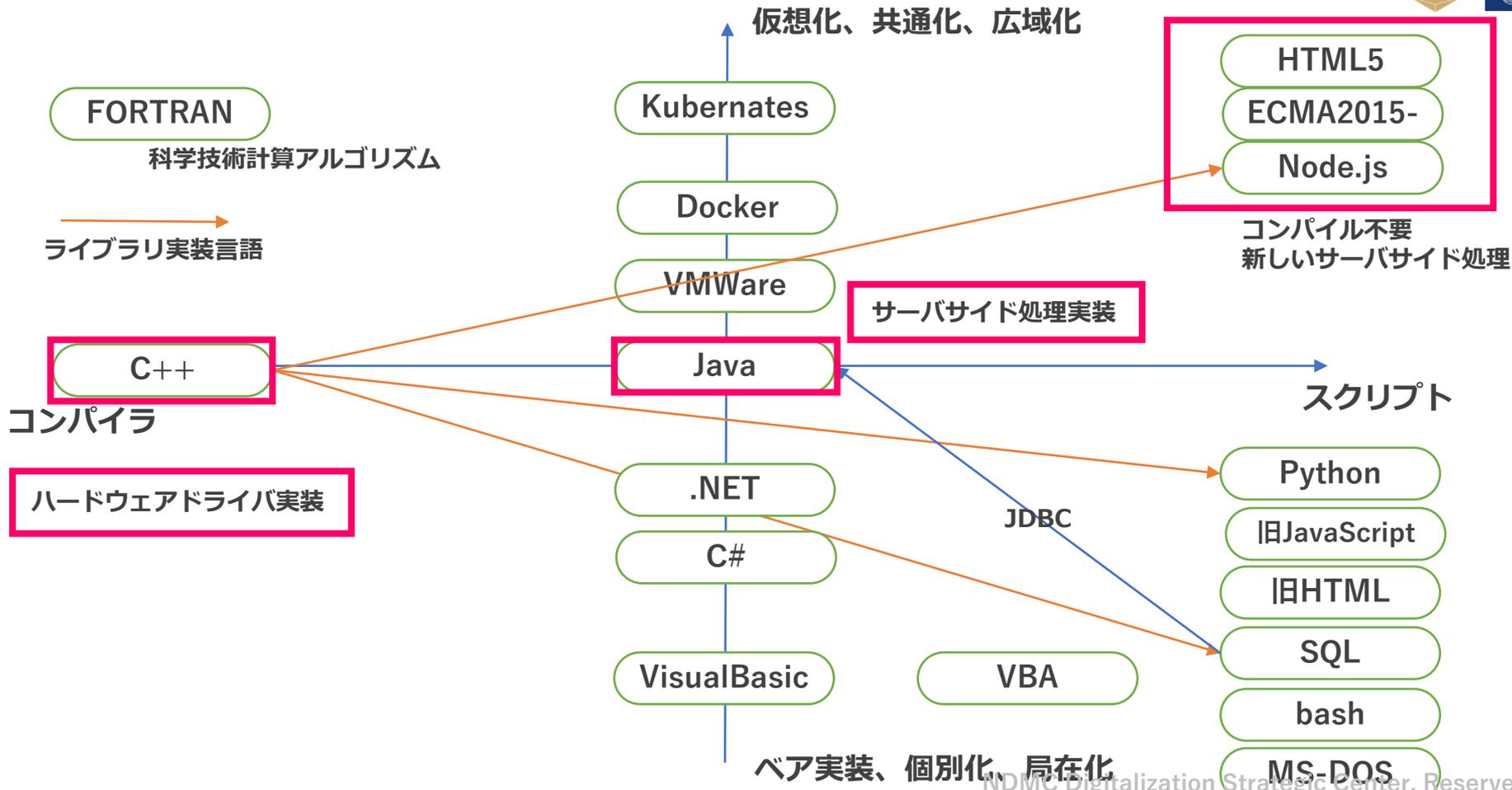
- 「環境構築」自体をオートメーションする→スキルの高いエンジニアの作業結果を複製して恩恵を受ける
- (Linuxのルート環境レベルで独立した)構造として独立することで、機能実装する際の相互関連性を減らす
- OS依存のプログラム (= マシン語に近いコンパイラを使う) を使わないようにすること
- マイクロコード思想：単純なプログラムやサービスは、再利用性が高くできる可能性がある
- 「コード作成者自身でないスタッフが、コードを容易に理解でき、迅速に活用できること」
- 文法ができるだけ簡素であること
- ライブラリが使いやすく、利用すると便利になる→エコシステムが成立している



express



採用言語の検討：仮想化と永続化の可能性から



Web/RESTに即したフレームワーク



Javaが早期にVM概念を確立していたにも関わらずデバイス言語にならなかったのは「学習コストの多さ」であると考えられる：

オブジェクト指向言語は細かな定義が行え、コードは十分高速だが、リファクタコストが高い

プロトタイプベースは「RAD的」な使い方と「厳格」な使い方の両方を可能とする

医療ワークフロー永続性とIoTに適したコーディング



- 普遍性の高いアルゴリズムをこそ永続化しようと試みられるべき
- 条件判断、分岐、中止、繰り返し、変更を記述するのに適した方式が望ましい
- コード肥大に伴うコード（クラスやライブラリ）を保守する手間が少ないことが望ましい
 - コンパイラがいない
- IoTが医療行為の起点になるシグナルを発するコードが望ましい
- ドライバ等を通じた接続性をサポートしている

Node-RED Con 2021

—HOME ABOUT SPEAKERS SESSIONS GALLERY SUPPORT

Japanese / English

23, Oct 2021 - Digital online event

Node-RED Con 2021

Node-RED Con 2021

HOME ABOUT —SPEAKERS SESSIONS GALLERY SUPPORT ORGANIZER CONTACT US



山崎 亘
株式会社ウフル



TODD SHARP
Oracle



伊藤 大輔
株式会社 日立製作所



MARC POUS
balena.io



青木 隆雄
株式会社ウフル



高野 幸太
群馬大学医学部附属病院シ
ステム統合センター



JONATHAN "PEL" DE
HALLEUX
Microsoft Research



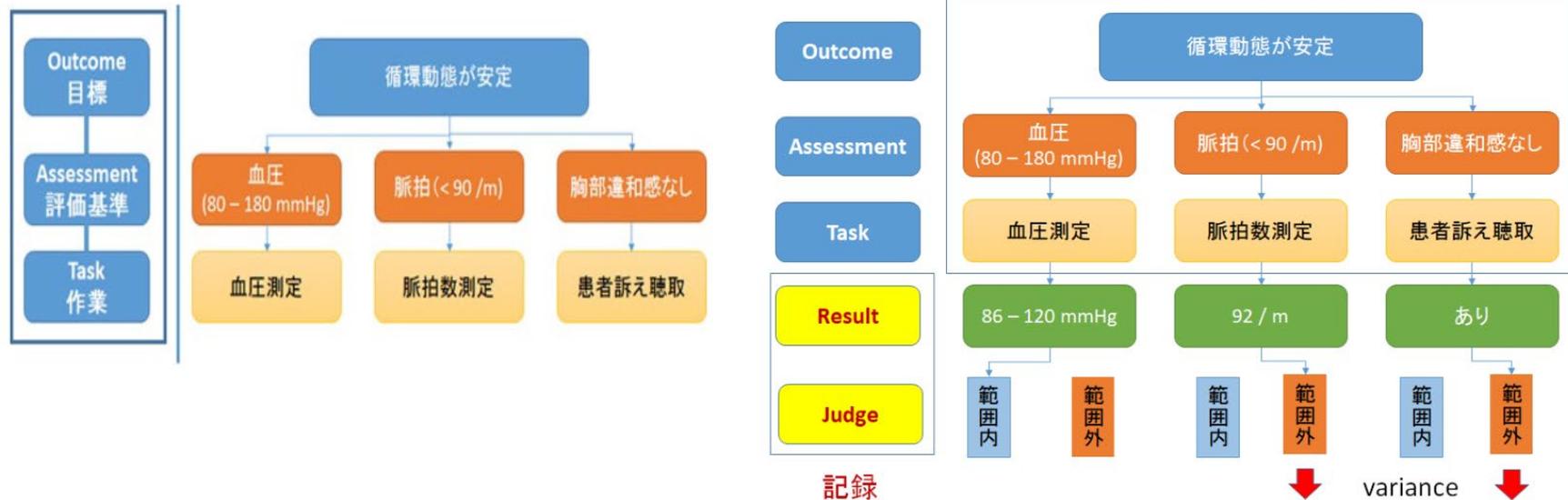
小路 慎浩
株式会社ウフル

e-Pathが提唱するアウトカムユニット構造



REST APIを

アウトカムユニットの構造と記録の例

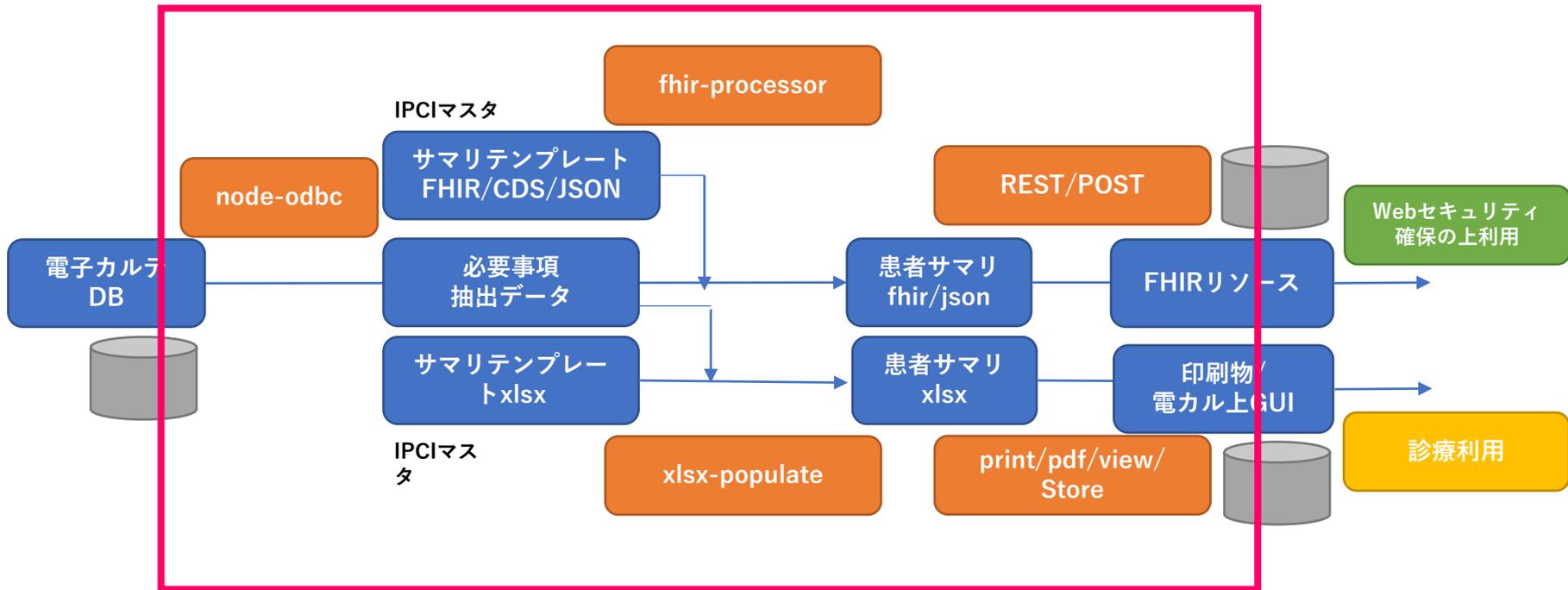


https://e-path.jp/img/prjt/outcum_unit.pdf

診療録サマリを生成するプロセスをIPCI機能で表現する



- Node-REDのフローに表現する流れ



プロトタイプオブジェクト指向の活用



msgオブジェクトの発生



function1

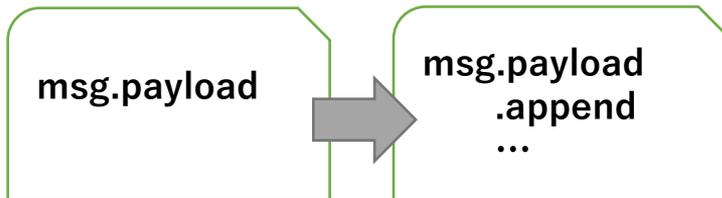
各ノード自身で独立して動かせる
切り離れたまま置いていてもエラーにならない
内部的に固有のIDを持つ
ノード自身は内部的にNode.jsのJSONデータとなっている (非常に重要)
→テキストデータによってノードの移植が可能

function1

JSの機能でプロパティを動的に付与

```
msg.append = msg.payload;  
return msg;
```

msgオブジェクト全体



function2

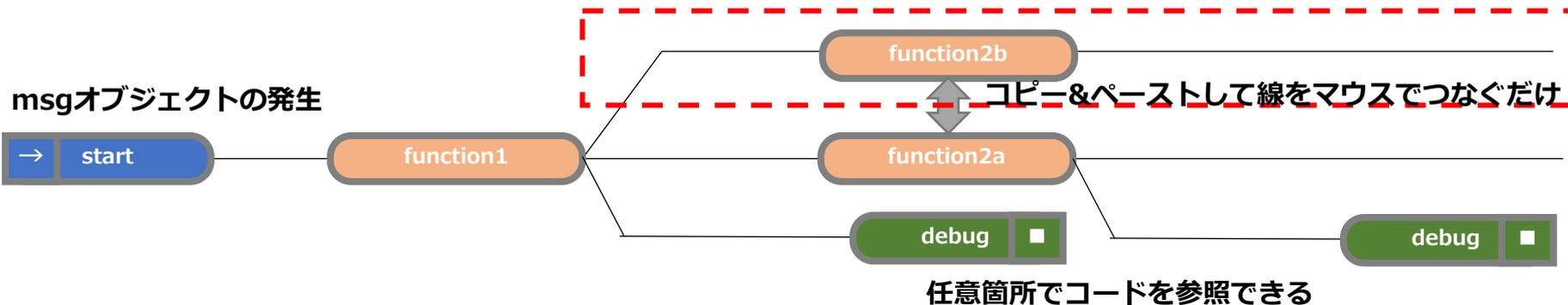
チームコーディング上の課題：
functionの記述によってはmsgオブジェクトの内容をクリアできるので、msgのオブジェクト名と保持/削除ルールをわかりやすくする必要あり

多数の同じデータ処理を行う場合：
Node-REDでシグナル分割するsplit/join機能を使う
msg.aaa.lengthプロパティでの繰り返し
(処理速度が気にならないければどちらで記述してよい)

コードブロックの入れ替えや変更が容易



新しい機能を試したい場合、ほぼGUI上のみで並列ノードを作成して比較できる
(コード記述型だと関連する処理ファイルの移動など含め工数が増える作業)



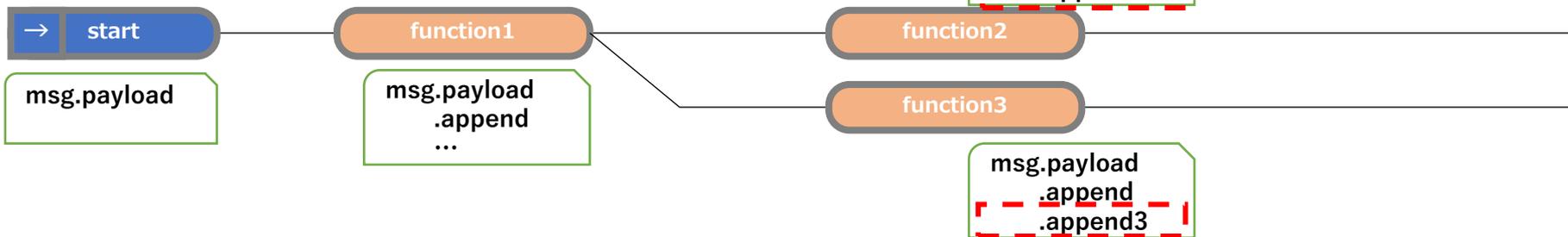
msgオブジェクトを通じたモジュール化が容易



Node-RED

フロープログラムのUIでは、マウスによりコードの実行順や分岐自身を取り扱くと、キーボードでこの変更を記述するよりも正確で手数の少ない変更ができる

例：



分岐後は独立したオブジェクトとして振舞う

医療ロジック自身が、演算途中のデータからさらに処理を分岐させたい場面が多い

例：検査結果の疾患解釈は診療科によって、また注目する疾患によって「組み合わせ」や「閾値」が異なる
これらを複数解釈させようとする、フロープログラムの「処理できるデータをマウスで複製できる」利点が活きる

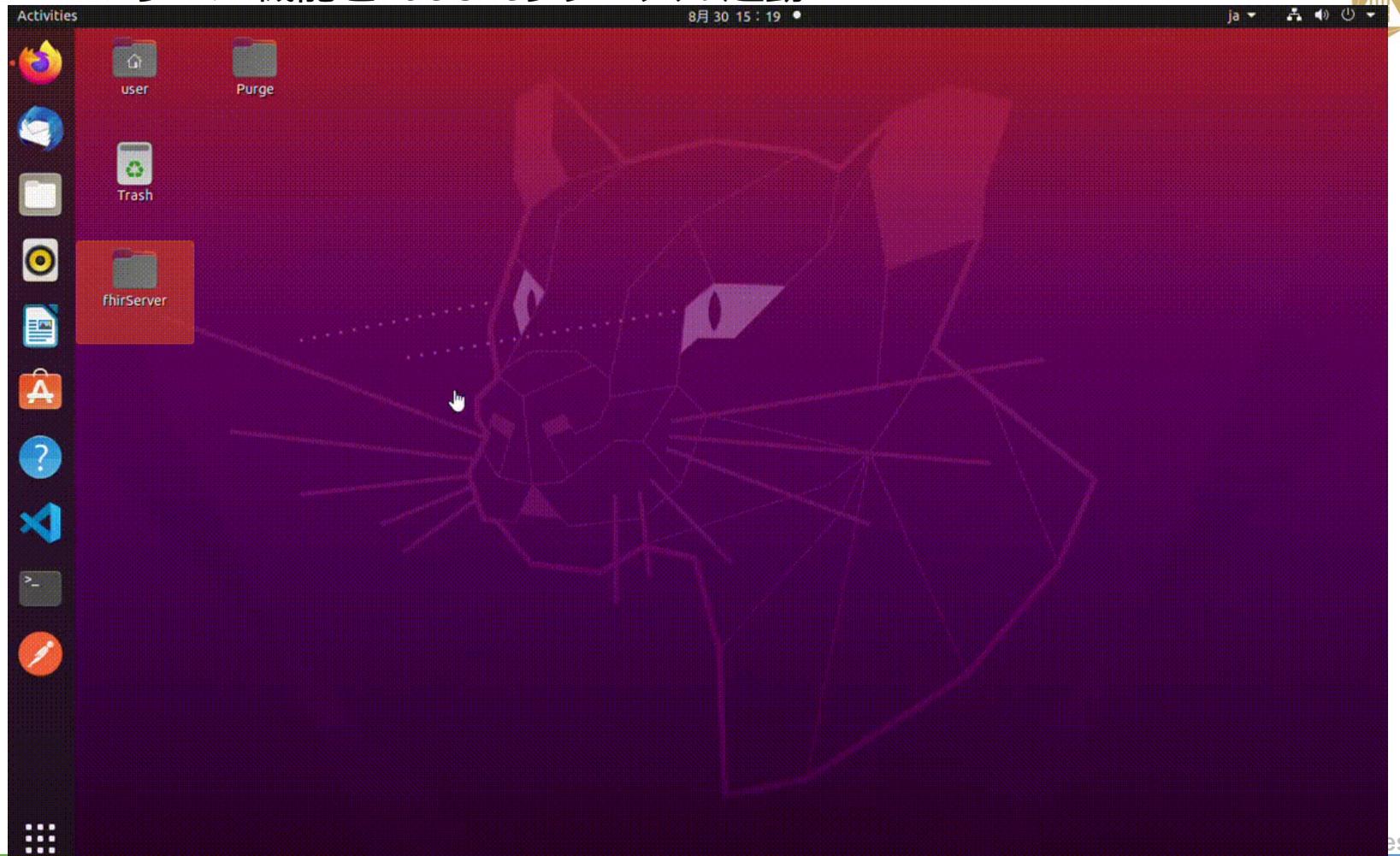


IPCIチュートリアル

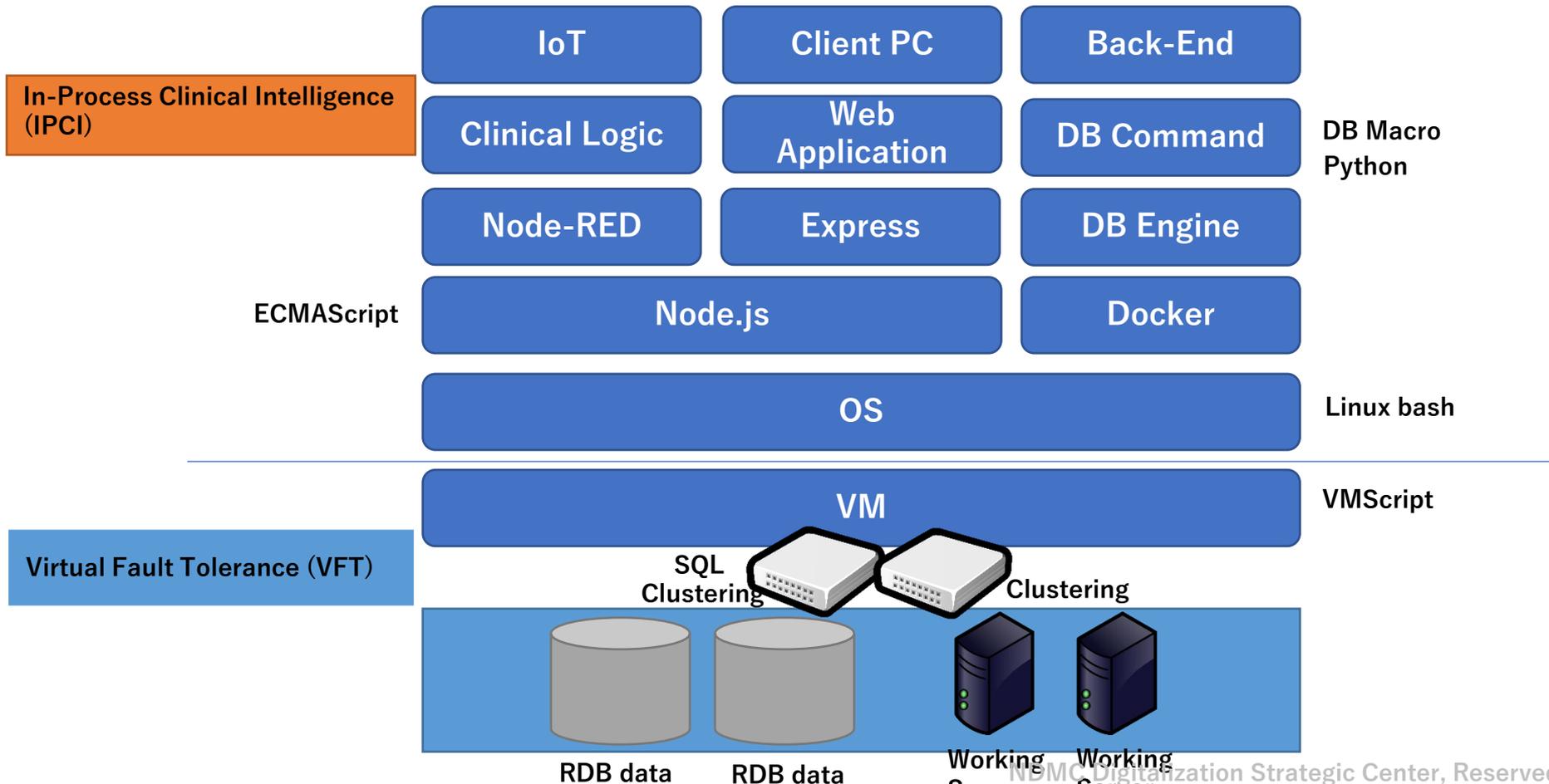
Node-REDによる医療ロジックフローのコピー&ペースト



IPCI-FHIRサーバ機能とPatientリソースの起動



IPCIをコアとしたスケーリング構成（大規模サーバへの応用）





触ってみましょう

作業の流れ



1 VMWareの準備

2 IPCI仮想マシンデータのダウンロード

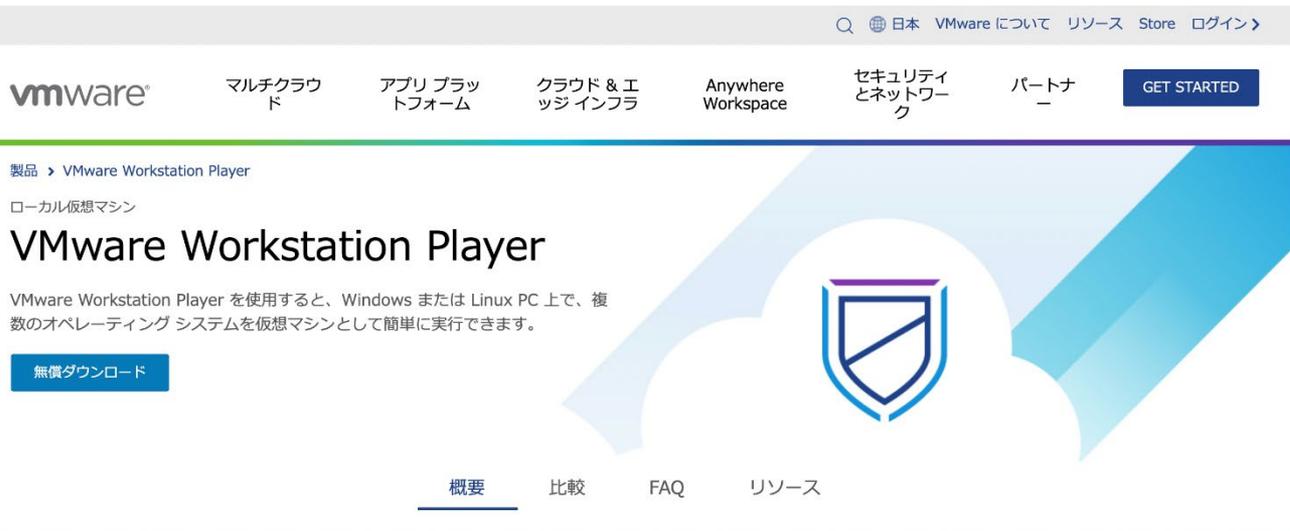
3 VM起動

4 Node-RED起動

5 Node-REDフロー作成

6 FHIRサーバ利用

<https://www.vmware.com/jp/products/workstation-player.html>



The screenshot shows the VMware Workstation Player product page. At the top, there is a navigation bar with the VMware logo and menu items: マルチクラウド (Multi-Cloud), アプリ プラットフォーム (App Platform), クラウド & エッジ インフラ (Cloud & Edge Infra), Anywhere Workspace, セキュリティとネットワーク (Security & Network), and パートナー (Partners). A 'GET STARTED' button is also present. Below the navigation bar, the page title is '製品 > VMware Workstation Player' and 'ローカル仮想マシン' (Local Virtual Machine). The main heading is 'VMware Workstation Player'. A sub-heading reads: 'VMware Workstation Player を使用すると、Windows または Linux PC 上で、複数のオペレーティング システムを仮想マシンとして簡単に実行できます。' (Using VMware Workstation Player, you can easily run multiple operating systems as virtual machines on Windows or Linux PC). A '無償ダウンロード' (Free Download) button is visible. Below the main content, there are tabs for '概要' (Overview), '比較' (Comparison), 'FAQ', and 'リソース' (Resources). The '概要' tab is selected.

- Intel CPU 2コア以上
- メモリ2GB以上 (4GB以上推奨)
- ディスク20-30GB

1 台の PC 上で複数の仮想オペレーティング システムを実行

授業で効率的な仮想インターフェイスを使用したい場合でも、BYO デバイスで会社のデスクトップを保護する方法が必要な場合でも、Workstation Player は、VMware vSphere Hypervisor テクノロジーを利用したシンプルでセキュアなローカル仮想化ソリューションを提供します。

シンプルで強力なローカル環境の仮想化

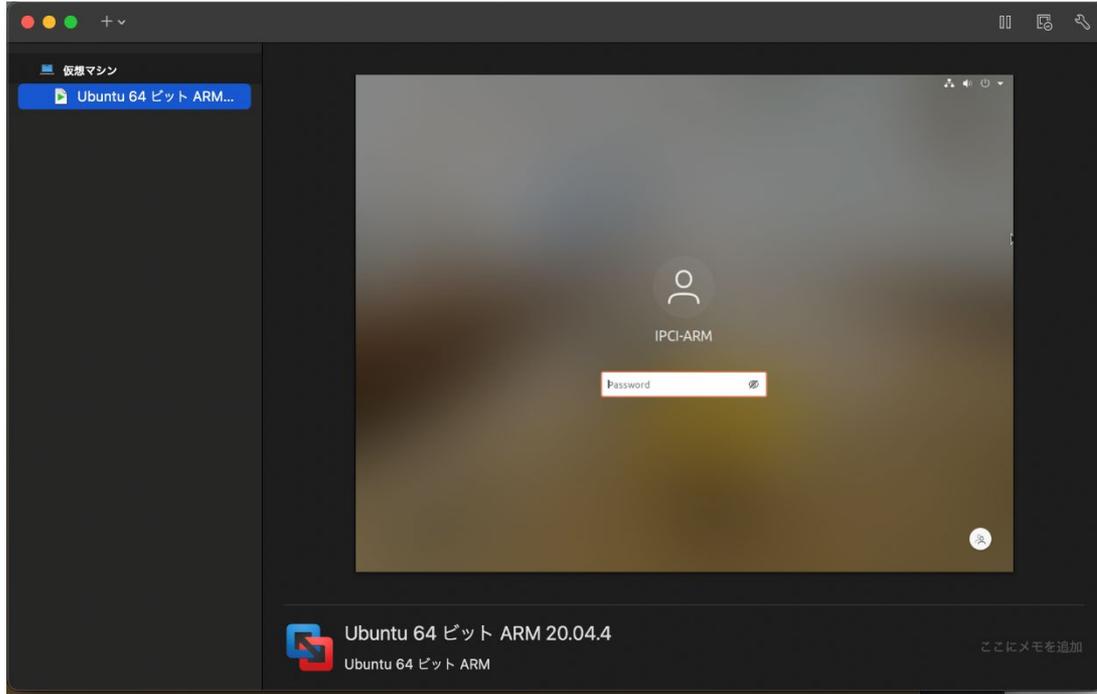


20 年以上にわたって開発され、vSphere と同じハイパーバイザー



「場所を選ばない働き方」の実現

ほぼすべての Windows PC または Linux PC 上でセキュアな仮想



- **VMWare WorkstationPlayerで起動**

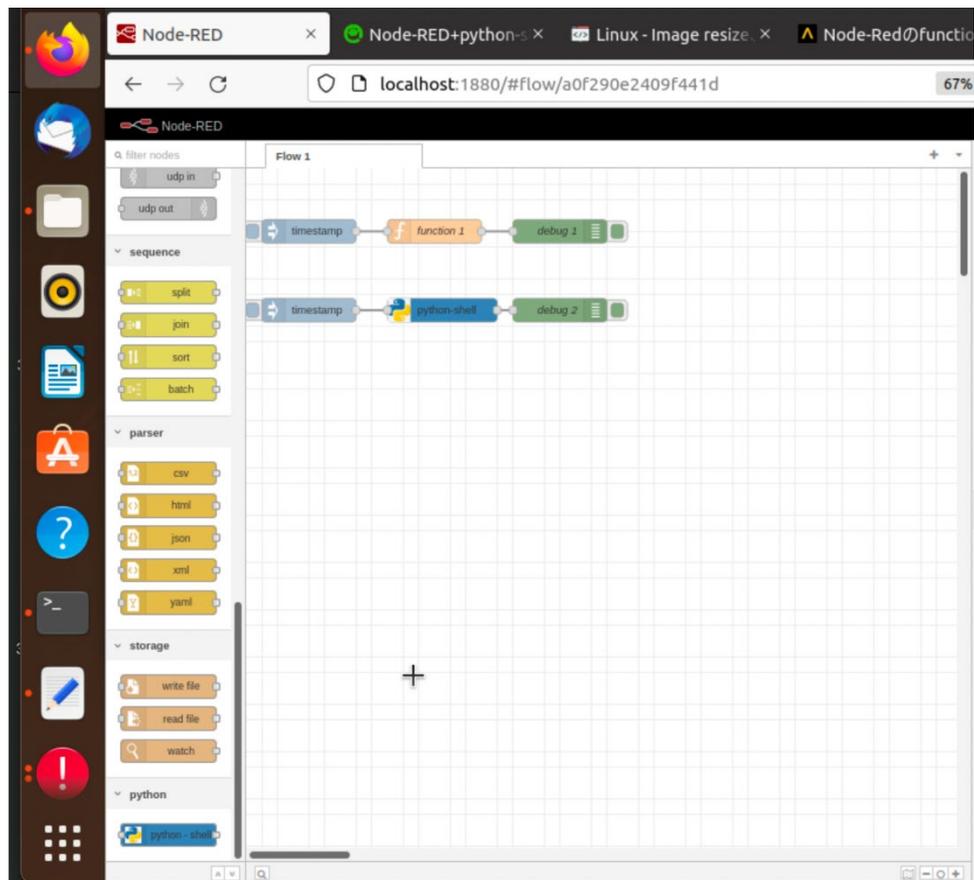
ターミナル（コマンド入力窓） 立ち上げ



```
ipci@ipci-arm-virtual-machine: ~  
ipci@ipci-arm-virtual-machine: ~$ node-red  
Welcome to Node-RED  
=====  
18 Aug 14:53:15 - [info] Node-RED version: v3.0.2  
18 Aug 14:53:15 - [info] Node.js version: v10.19.0  
18 Aug 14:53:15 - [info] Linux 5.13.0-30-generic arm64 LE  
18 Aug 14:53:15 - [info] Loading palette nodes  
18 Aug 14:53:16 - [info] Settings file : /home/ipci/.node-red/settings.js  
18 Aug 14:53:16 - [info] Context store : 'default' [module=memory]  
18 Aug 14:53:16 - [info] User directory : /home/ipci/.node-red  
18 Aug 14:53:16 - [warn] Projects disabled : editorTheme.projects.enabled=false  
18 Aug 14:53:16 - [info] Flows file : /home/ipci/.node-red/flows.json  
18 Aug 14:53:16 - [info] Creating new flow file  
18 Aug 14:53:16 - [warn]  
  
-----  
Your flow credentials file is encrypted using a system-generated key.  
  
If the system-generated key is lost for any reason, your credentials  
file will not be recoverable, you will have to delete it and re-enter  
your credentials.
```

起動コマンドは
node-red↩

Node-RED起動法



- Webブラウザ(ここではFireFox)を起動し、URLに“localhost:1880”と入力

Node-REDコントロールの利用法



- 左側のタブからノードをドラッグ&ドロップする

The screenshot displays the Node-RED web interface in a browser window. The address bar shows the URL `localhost:1880/#flow/a0f290e2409f441d`. The main workspace shows a flow named "Flow 1" with two parallel paths. The top path consists of a "timestamp" node, a "function 1" node, and a "debug 2" node. The bottom path consists of a "timestamp" node, a "python-shell" node, and a "debug 2" node. The left sidebar contains a search bar and several categories of nodes: "sequence" (split, join, sort, batch), "parser" (csv, html, json, xml, yaml), "storage" (write file, read file, watch), and "python" (python-shell). The right sidebar shows a "debug" console with the following error messages:

```
8/18/2022, 3:53:58 PM node: 7b45054c64814e09
msg: msg[0]
"Cannot find module '/home
/ipci/.node-red/node_modules
/nodered-contrib-mypython/python-
shell/'"

8/18/2022, 3:54:32 PM node: 7b45054c64814e09
msg: msg[0]
"Cannot find module '/home
/ipci/.node-red/node_modules
/nodered-contrib-mypython/'"

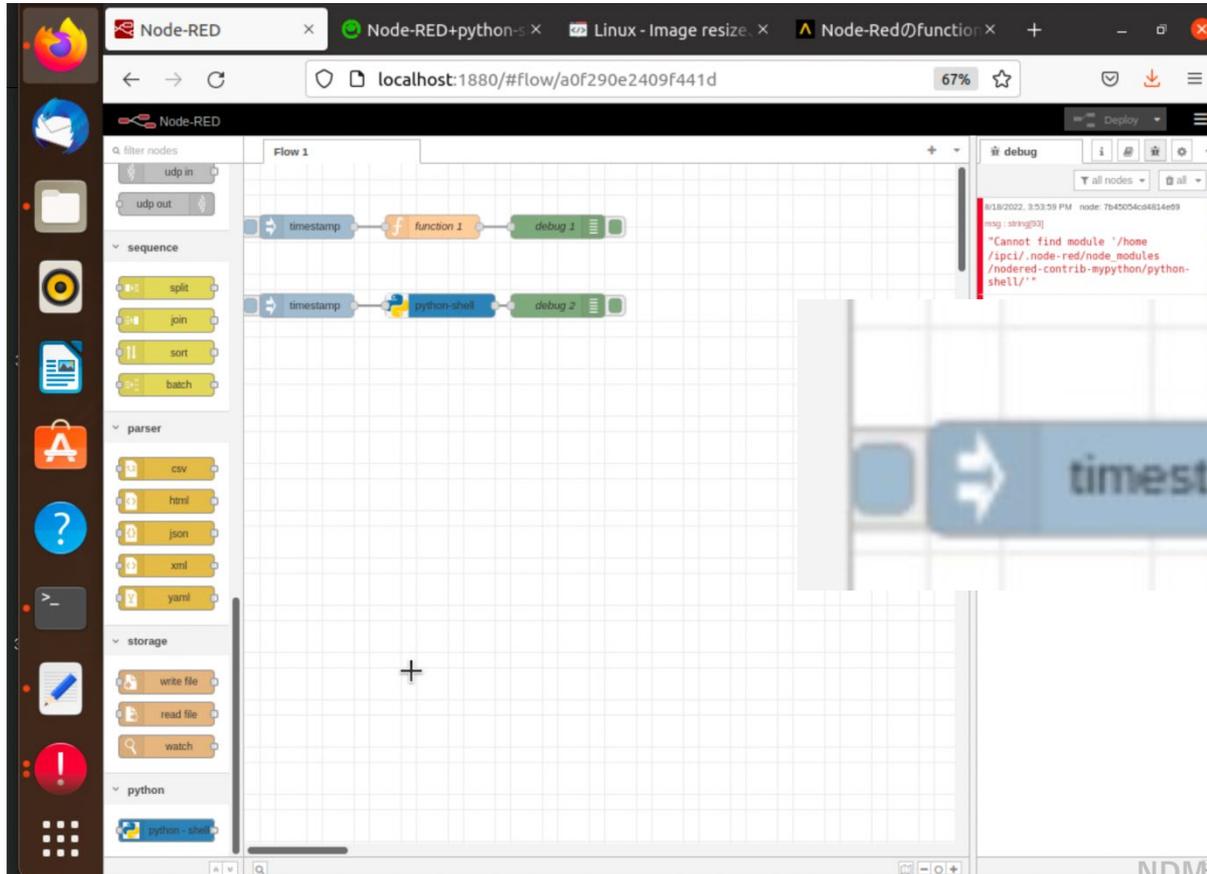
8/18/2022, 3:57:08 PM node: 7b45054c64814e09
msg: msg[45]
"Cannot read property 'runString' of
undefined"

8/18/2022, 3:57:40 PM node: 7b45054c64814e09
msg: msg[45]
"Cannot read property 'runString' of
undefined"
```

フローの接続



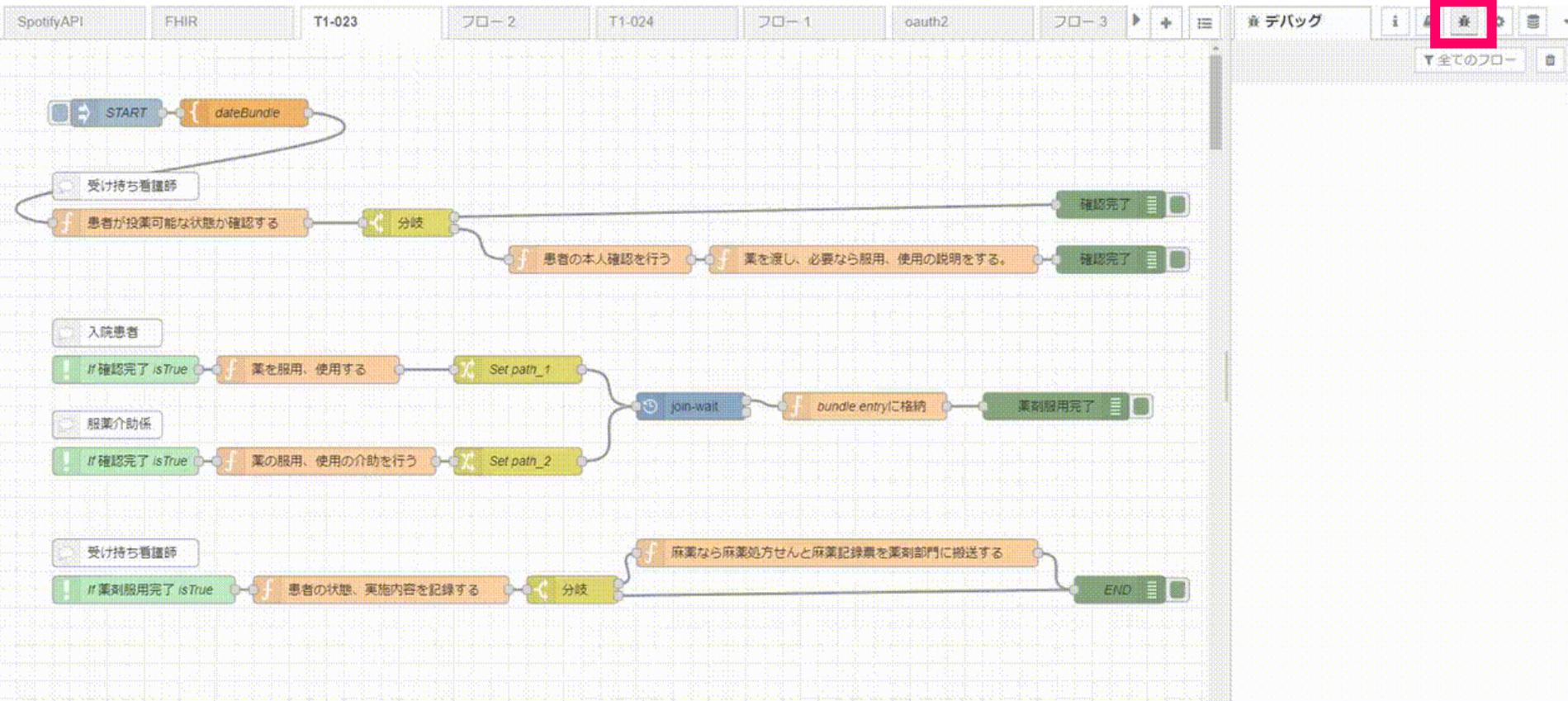
- マウスでコネクタ部分[・]をドラッグすると配線が発生する



Node-REDコントロールの利用法



- “start”の左側をクリックするとフローが実行され、debugボタン押下で出力が表示される





UMLモデル要素の Node-REDフローでの表現法

Node-REDにおける業務フロー(UMLモデル)実装性の確認

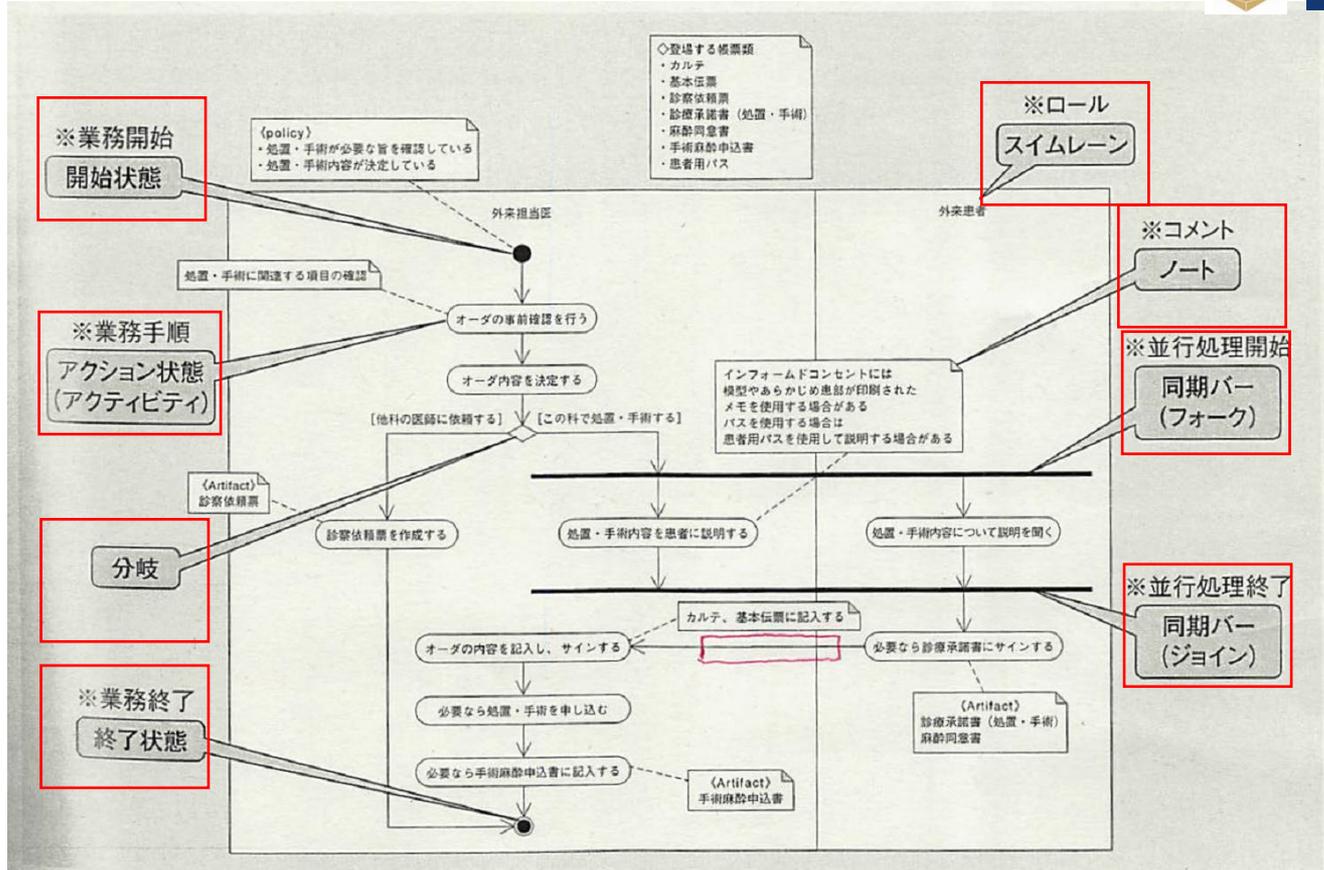
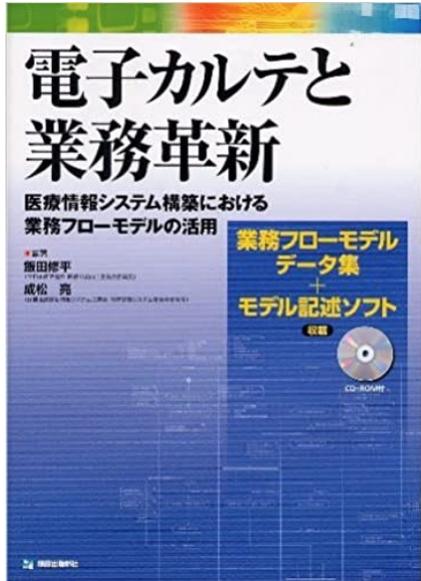


図8-1 アクティビティ図の構成要素

業務フロー記述UMLで必要とされる機能とNode-RED要素の対応調査



モデル要素の対応付け

(1) 開始状態

開始状態は処理の開始を表す。アクティビティ図上では黒で塗りつぶした丸で記述する。



図8-2 開始状態

(2) 終了状態

終了状態は処理の終了を表す。アクティビティ図上では黒と白の二重丸で記述する。



図8-3 終了状態

(3) アクション状態 (アクティビティ)

アクション状態は何かの処理を行っている状態を表し、アクティビティとも呼ばれる。アクティビティ図上では角の丸い長方形で記述する。

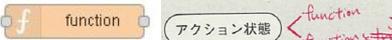


図8-4 アクション状態

(4) 遷移

遷移はある処理の状態から別の処理の状態へ遷移することを表す。アクティビティ図上では遷移する方向への矢印で記述する。



図8-5 遷移

(5) 分岐

分岐は何かの条件によって変化する処理の流れを表す。アクティビティ図上では遷移の矢印に、[] を付けた分岐条件で表記する。なお、分岐のポイントはひし形で記述する。

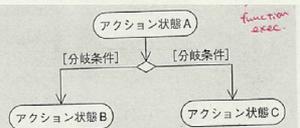


図8-6 分岐

(6) 同期バー

同期バーは複数の処理が並行して行われる流れを表す。並行処理の開始を表す同期バーを「フォーク」、並行処理の終了を表す同期バーを「ジョイン」と呼ぶ。アクティビティ図上では太線で記述する。

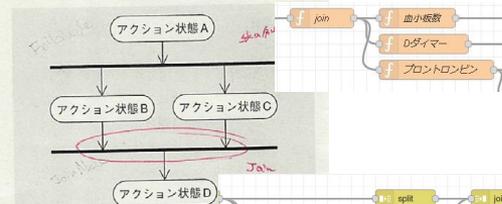
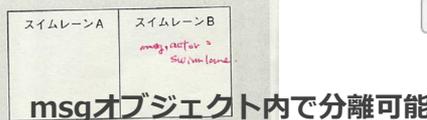


図8-7 同期バー

(7) スイムレーン

スイムレーンはアクション状態を実行する担当者を表す。アクティビティ図上では大きな長方形で記述する。スイムレーンにアクション状態を配置することで、その担当者が明確に表現できる。



msgオブジェクト内で分離可能

(8) サブアクティビティ状態

サブアクティビティ状態はその中にアクティビティ図が内包されていることを表す。アクティビティ図上では角の丸い長方形の右下にサブアクティビティ状態を表すアイコンを付記して記述する。

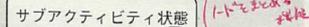


図8-9 サブアクティビティ状態

(9) オブジェクトフロー状態

オブジェクトフロー状態はアクション状態間では何かのオブジェクト (情報等) の受け渡しが行われることを表す。アクティビティ図上ではアクション状態間にオブジェクトを表す長方形を配置し、破線の矢印でそれが受け渡される方向を記述する。

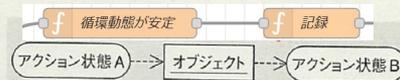


図8-10 オブジェクトフロー状態

(10) ノート

ノートはモデルに対するコメントを表す。UMLの全てのアクティビティ図で使用できる。

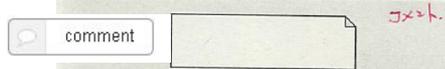


図8-11 ノート

(11) 割り込み可能アクティビティ領域 (UMLバージョン 2.0)

割り込み可能アクティビティ領域は、その領域内のアクティビティで特定のイベントが発生した場合、通常のフローとは別の処理 (アクティビティ) へシグナルを送ることができる仕組みである。特定の状況を感じ取る範囲を、角の丸い破線の長方形で囲んで表現する。領域内部からのシグナルは、ジグザグの実線に矢印を付け、それを内部から外部に出して表す。

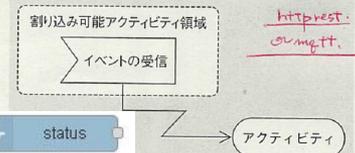


図8-12 割り込み可能アクティビティ領域

(12) コネクタ (UMLバージョン 2.0)

コネクタは複雑なアクティビティ図をシンプルに整理するための要素である。コネクタは2つ1組で使用される。一方のコネクタに対して接続したフローを、もう一方のコネクタから再開できる。コネクタは丸の中にコネクタ名を記入して表す。コネクタ名は、主に「A」などのアルファベット1文字が使われる。

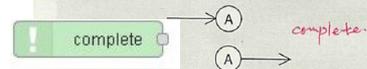


図8-13 コネクタ

必要機能に対応する要素があることを確認できた

Node-REDを用いた病院業務フローモデル



- <https://gist.github.com/ryumtym/b091d78c7035328919fe2283e2b965b8>

GitHub Gist Search... All gists Back to GitHub Sign in Sign up

Instantly share code, notes, and snippets.

ryumtym / node-red.md Secret ☆ Star 0 🍴 Fork 0

Last active 2 days ago

<> Code ↻ Revisions 9 Embed <script src="https://...> Download ZIP

node-red.md Raw

電子カルテと業務革新—医療情報システム構築における業務フローモデルの活用

TI-023: 投薬実施(内服・外用)プロセスの大まかな流れ

1. 受け持ち看護師のプロセスが開始
2. 受け持ち看護師のプロセス完了後、入院患者と服薬介助係のプロセスが同時に開始する
3. 患者と服薬介助係のプロセス終了後、看護師が実施入力を行い、TI-023フロー完了。

※内服薬、外用薬を対象とする

薬の種類、用法、用量を説明する
服薬の管理伝票については
薬剤師が適宜記録を適切に説明する



要素の解説

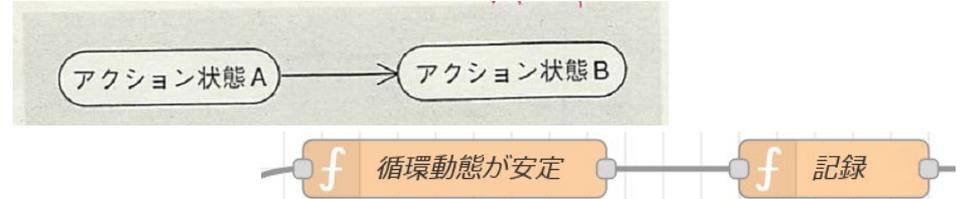


(1)開始状態

開始状態は処理の開始を表す。
アクティビティ図上では黒で塗りつぶした丸で記述する。

(2)開始状態

終了状態は処理の終了を表す。
アクティビティ図上では白と黒の二重丸で記述する。

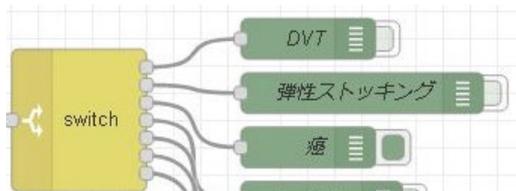
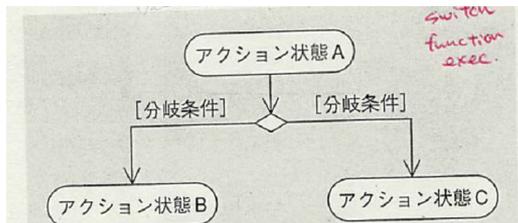


(3) アクション状態(アクティビティ)

アクション状態は何かの処理を行っている状態を表し、アクティビティとも呼ばれる。アクティビティ頭上では角の丸い長方形で記述する。

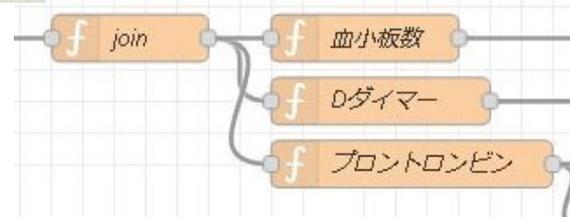
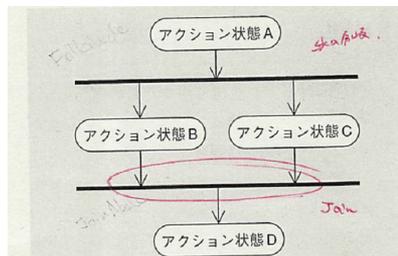
(4) 遷移

遷移はある処理の状態から別の処理の状態へ遷移することを表す。アクティビティ頭上では遷移する方向の矢印で記述する。



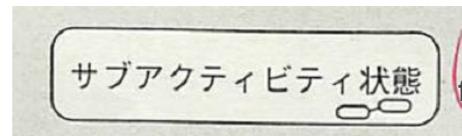
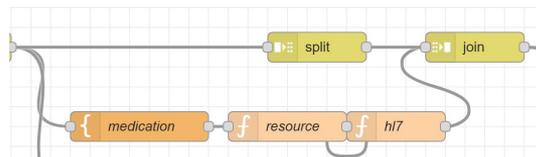
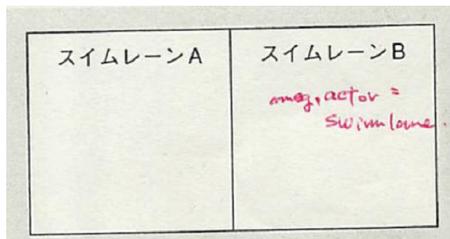
(5)分岐

分岐は何らかの条件によって変化する処理の流れを表す。アクティビティ図上では遷移の矢印に、[]をつけた分岐条件で表記する。なお、分岐のポイントはひし形で記述する。



(6)同期バー

同期バーは複数の処理が進行して行われる流れを表す。並行処理の開始を表す同期バーを「ジョイン」と呼ぶ。アクティビティ頭上では太枠で記述する。



functionの集約可能

(7) スイムレーン

スイムレーンはアクション状態を実行する担当者を表す。アクティビティ図上では大きな長方形で記述する。スイムレーンにアクション状態を配置することで、その担当者が明確に表現できる。

(8) サブアクティビティ状態

サブアクティビティ状態はその中にアクティビティ図が内包されていることを表す。アクティビティ図上では角の丸い長方形の右下にサブアクティビティ状態を表すアイコンを付記して記述する。

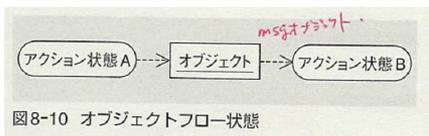
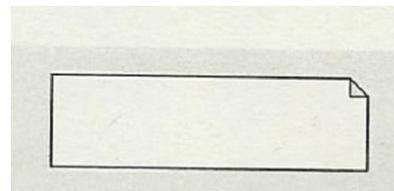


図8-10 オブジェクトフロー状態



(9) オブジェクトフロー状態

オブジェクトフロー状態はアクション状態間で何らかのオブジェクト（情報等）の受け渡しが行われることを表す。アクティビティ図上ではアクション状態間にオブジェクトを表す長方形を配置し、破線の矢印でそれが受け渡される方向を記述する。

(10) ノート

ノートはモデルに対するコメントを表す。UMLの全てのアクティビティ図で使用できる。

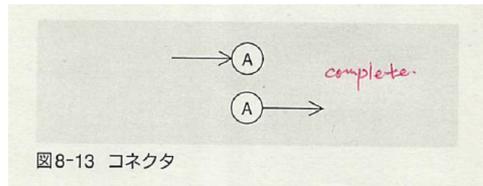
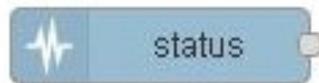
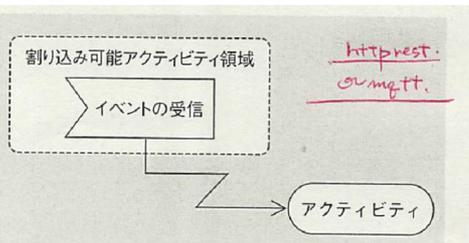


図8-13 コネクタ



(11) 割り込み可能アクティビティ領域 (UMLバージョン 2.0)

割り込み可能アクティビティ領域は、その領域内のアクティビティで特定のイベントが発生した場合、通常のフローとは別の処理（アクティビティ）へシグナルを送ることができる仕組みである。特定の状況を感知する範囲を、角の丸い破線の長方形で囲んで表現する。領域内部からのシグナルは、ジグザグの実線に矢印を付け、それを内部から外部に出して表す。

(12) コネクタ (UMLバージョン 2.0)

コネクタは複雑なアクティビティ図をシンプルに整理するための要素である。コネクタは2つ1組で使用される。一方のコネクタに対して接続したフローを、もう一方のコネクタから再開できる。コネクタは丸の中にコネクタ名を記入して表す。コネクタ名は、主に「A」などのアルファベット1文字が使われる。



Node-REDフロー内での msgオブジェクトの取り扱い



- フローの実行に伴ってmsgオブジェクトが生成される
- msgオブジェクトは平たく言えば、JavaScriptの変数と同じ取り扱いができる
- M言語風と言えば「label : value」の形式
- Propertyの追加
- Propertyの編集
- 配列の利用
- 連想配列の利用



Node-RED OSSフローの利用



- ターミナルよりnode-redとタイプ

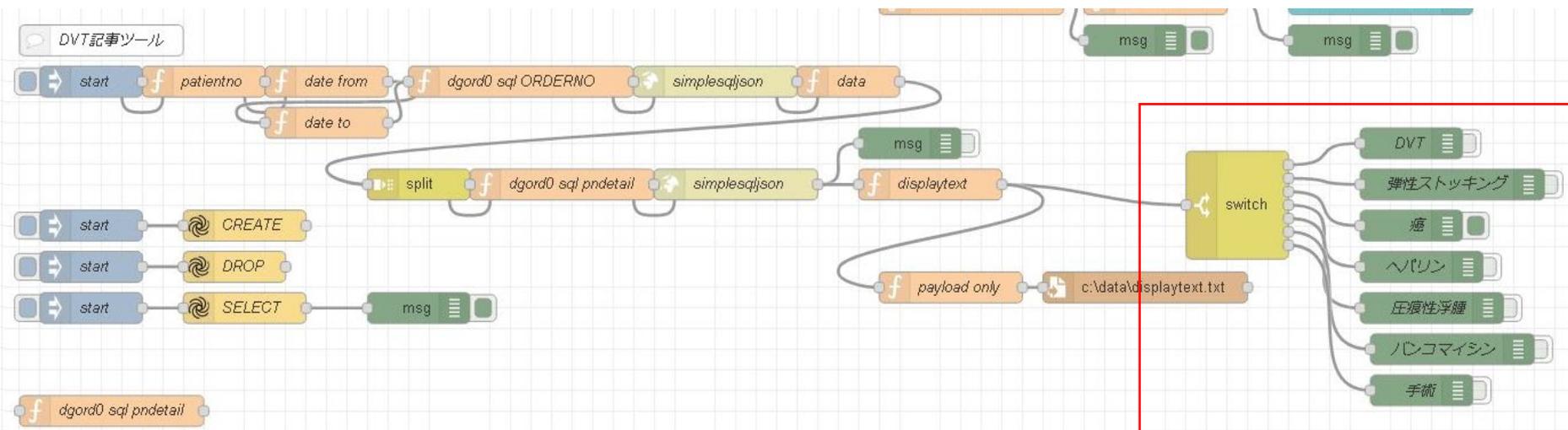


Node-REDフロー組み立て例

記事中のキーワードを用いたスイッチアクション



患者ID、期間をもとに診察記事を抽出し、含まれるキーワードによってフラグを複数生成するデモ
このようなコード体系では、運用後のロジック追加の際に機能全体の稼働保証を憂慮せずに済むメリットが大きい



結果一覧から目的の検査項目を抽出する方法



検査結果一覧から特定の項目（ここではDダイマー）をフィルタする書き方

The screenshot shows a Node-RED workflow in a browser window. The workflow is organized into three flows:

- フロー-1:** Starts with a 'start' node, followed by 'patientno' and 'date' nodes.
- フロー-2:** Contains a 'dgord0 sql resdetail' node connected to a 'simplesqjson' node, which is then connected to a 'resdetail' node. This flow outputs to a 'msg' node.
- フロー-3:** Contains a 'dgord0 sql resultitem' node connected to a 'simplesqjson' node, which is then connected to a 'resultitem' node. This flow outputs to a file node 'c:\data\resultitem.json'.
- フロー-4:** Contains a 'dgord0 sql testitem' node connected to a 'simplesqjson' node, which is then connected to a 'testitem' node. This flow outputs to a 'join' node.

The 'join' node is connected to three output nodes: '血小板', 'Dダイマー', and 'フロン'. A red box highlights the JavaScript code in the 'function ノードを編集' panel, which filters the 'resdetail' array for 'Dダイマー' based on the 'TESTITEMCODE'.

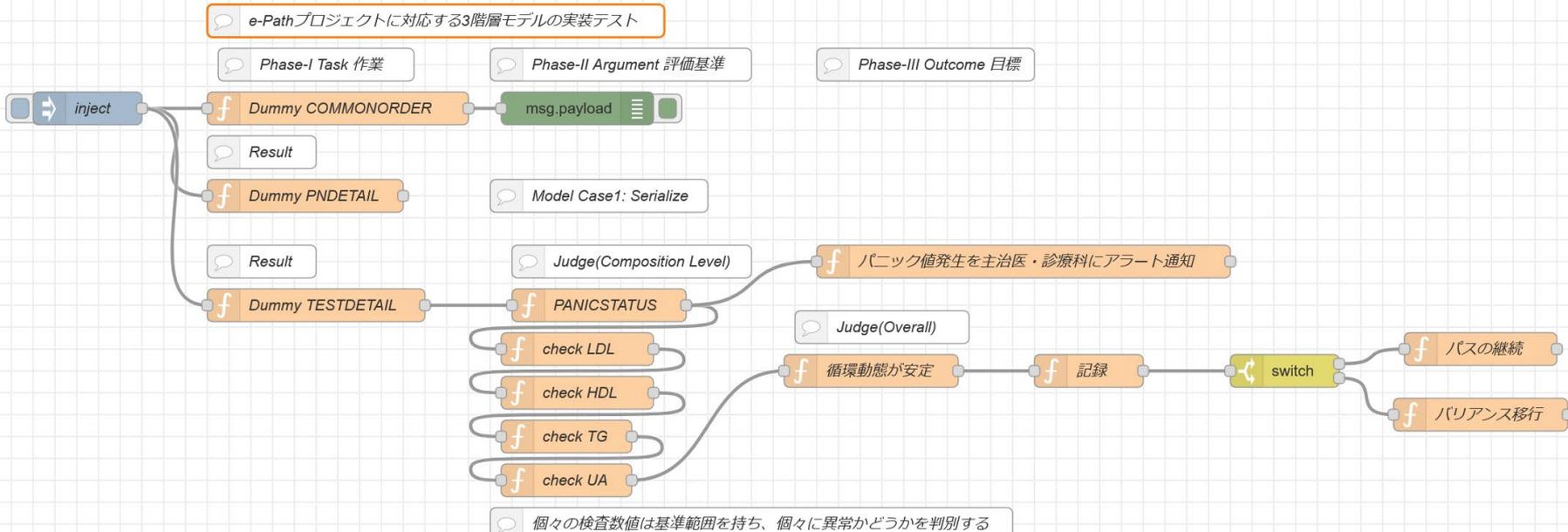
```
1 - var result = msg.resdetail.filter(function(item,index){
2   var str = item.TESTITEMCODE;
3   if(str.indexOf('227300') > -1){return true;}
4   //以下条件を書き連ねてよい(OR)
5 - });
6 msg.out=result;
7 return msg;
```

IPCIアーキテクチャによるe-Path3階層の実装テスト



NEC MegaOAKHRとのオーダリングDB接続を実現

フロー 1

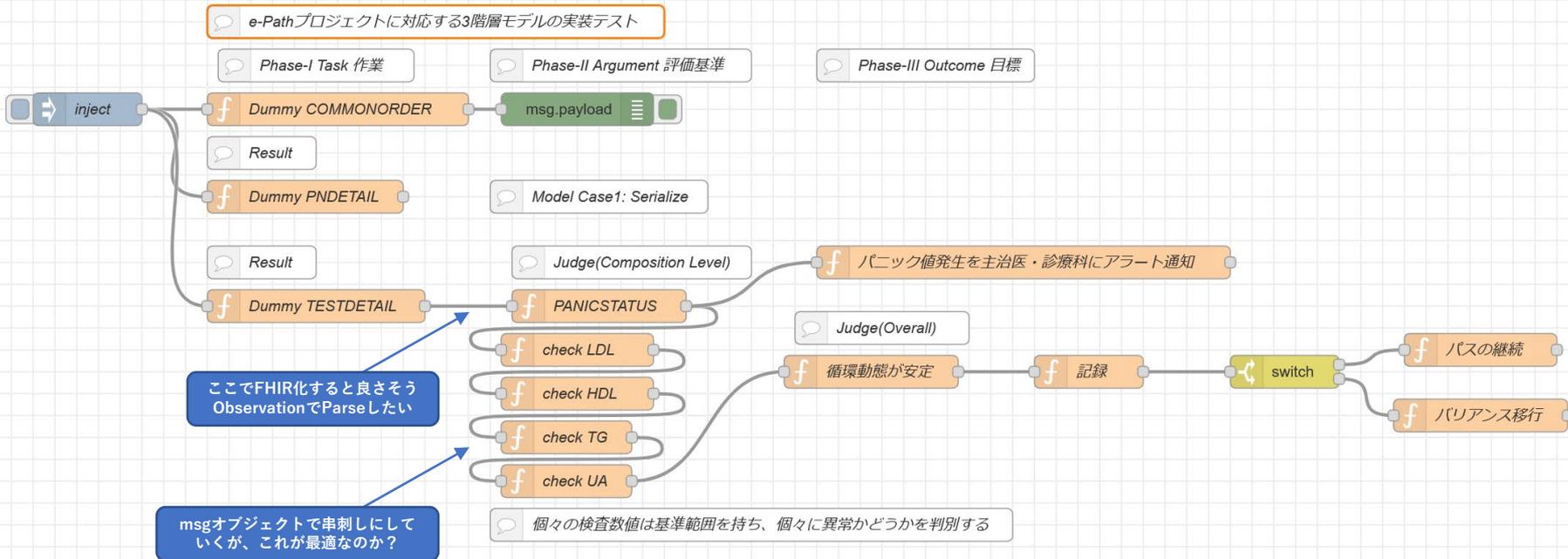


IPCIアーキテクチャによるe-Path3階層の実装テスト



NEC MegaOAKHRとのオーダリングDB接続を実現

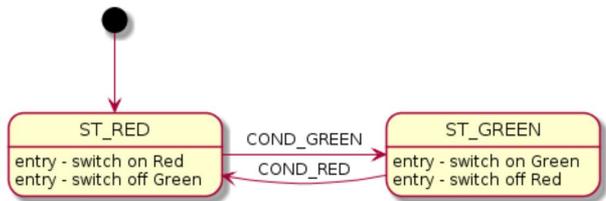
フロー 1



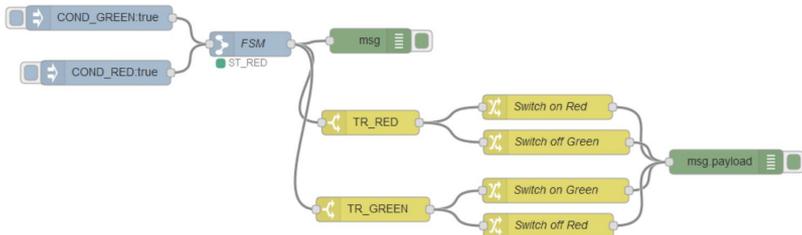
Example

Simple pedestrian light with manual trigger

The following example demonstrates the FSM node for a simple pedestrian traffic light state machine. It has two states ST_RED and ST_GREEN, where the related light gets switched on. The ST_RED state is entered at startup. The transitions between the two states are triggered by COND_GREEN and COND_RED respectively.



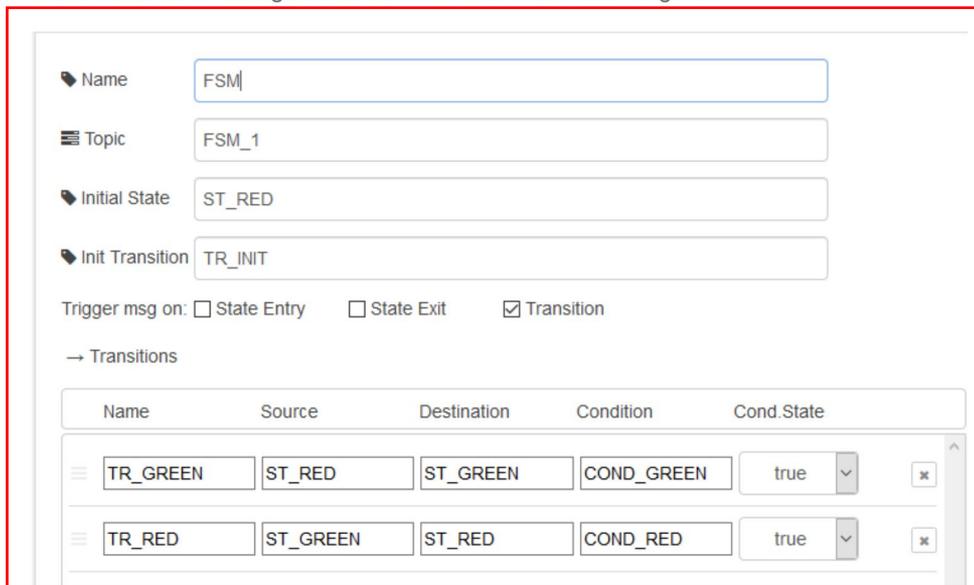
The flow in node-red:



The switch nodes filter the result message of FSM for the entry event of the states and the change node generates the final action of switching the traffic lights.

The exported flow can be downloaded from here: [FLOW](#).

The FSM node is configured as shown in the UML state diagram.



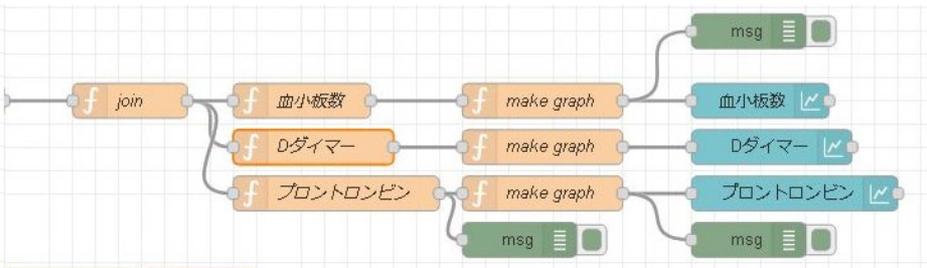
Name	Source	Destination	Condition	Cond.State
TR_GREEN	ST_RED	ST_GREEN	COND_GREEN	true
TR_RED	ST_GREEN	ST_RED	COND_RED	true

UML-FSMのストレートな記述性を確認できた

Dashboard機能



Node-RED上に実装されている機能



グラフテンプレートとデータ構造の記述の仕方

名前: make graph

初期化処理: コード: 終了処理:

```
1 var temp = [{
2   "series": ["prontronbin"],
3   "data": [[]],
4   "labels": [""]
5 }];
6 for(var i = 0; i < msg.out.length; i++){
7   var plotdata = {"x" : msg.out[i].UPDATEDATE, "y" : parseFloat(msg.out[i].TESTRESULT)};
8   temp[0].data[0][i] = plotdata;
9 }
10
11 msg.payload = {};
12 msg.payload = temp;
13 return msg;
```

DVT検査指標

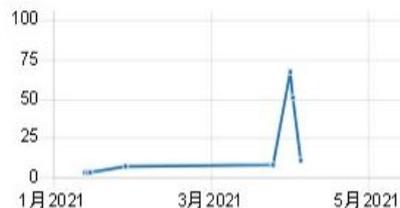
プロトロンビン

プロトロンビン



Dダイマー

Dダイマー



血小板数

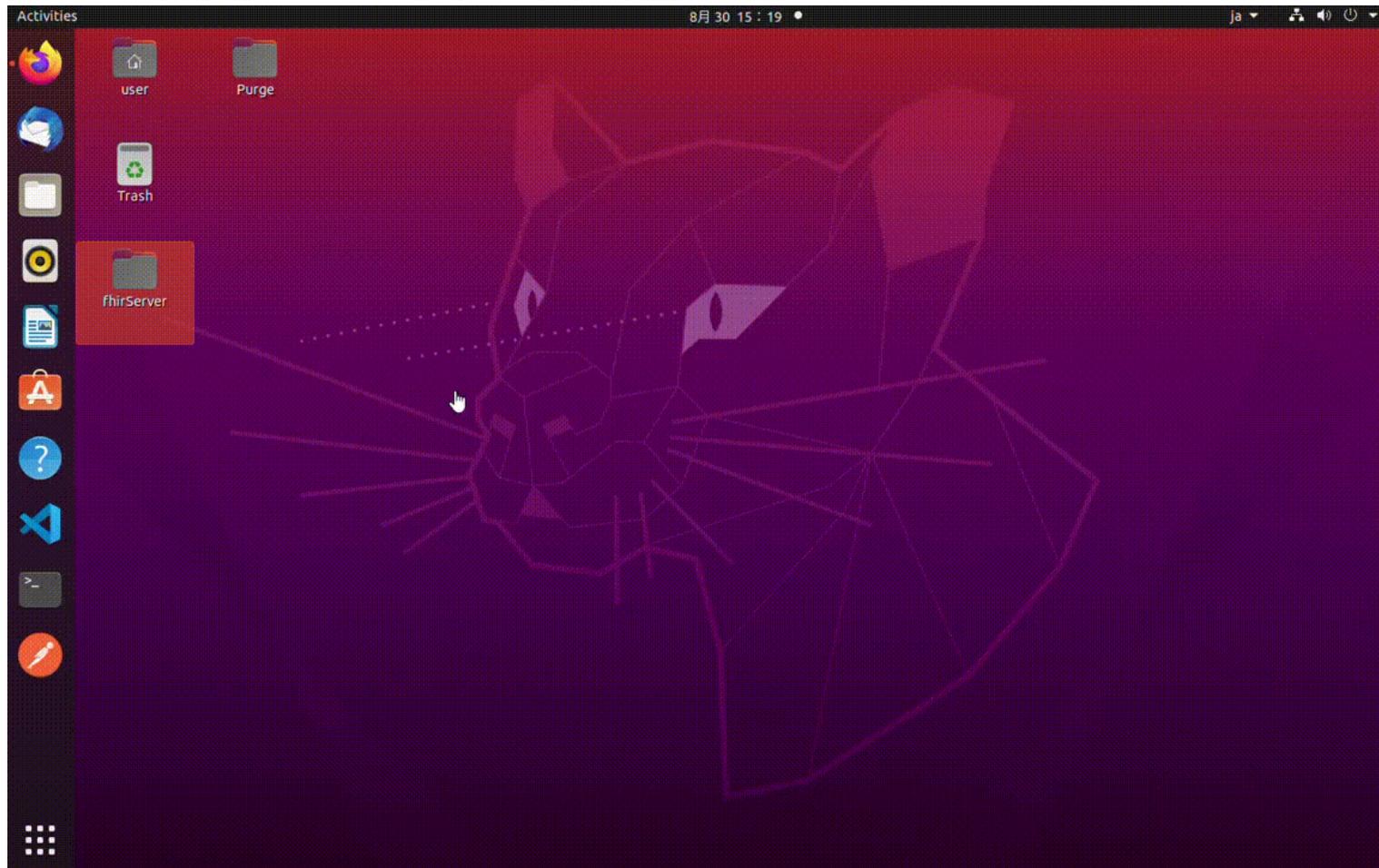
血小板数





IPCI-FHIRサーバーの起動と利用

FHIRサーバ機能とPatientリソースの起動





VSCodeを用いたNode.jsコーディング

IPCIサーバ 開発系画面の一例



Node-RED

②injectでRun

①コード編集

③動作をモニタ

The Node-RED interface shows a workflow starting with an inject node, followed by a 'timeout-false' node, a 'date' node, and a 'date to' node. The flow then goes to a 'global timeout + false' node, a 'Read File' node, a 'simplejsonfile' node, a 'delay ts' node, and finally an 'misp payload' node. A 'restart after timeout' node is also present. The right sidebar shows a debug console with logs from 2021/8/19 14:52:01, including 'no responses from server' and 'Error: ENOENT: no such file or directory, open 'c:\data\raydetail.json''.

VisualCode

The Visual Studio Code editor displays the source code for `simplejsonfile.js`. The code includes a `router.get` function that processes a request, checks for a file, and writes it to a database. The terminal at the bottom shows the command `node simplejsonfile.js` and its output, including logs for 'result.count', 'result.length', and 'filepath'. The terminal also shows a warning about the Windows LWS extension.

Node.js
npm



route一覧

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

//Todo 1: routerの追加 (BasicTest)
var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');
var pictureTestRouter = require('./routes/picturetest');

//InterSysteme IRIS
var irisouttestRouter = require('./routes/irisouttest');
var irissapitestRouter = require('./routes/irissapitest');
var irisDebugRouter = require('./routes/irisdebug');
var irisDebug2Router = require('./routes/irisdebug2');
var irisDebug3Router = require('./routes/irisdebug3');
var irisDebug3sonRouter = require('./routes/irisdebug3son');
var watchRayReportRouter = require('./routes/watchrayreport');
var irisDebug3Router = require('./routes/irisdebug3');
var irisCallSqlRouter = require('./routes/iriscallsql');

//RUIFILM Medical FRIS Report
var frisReportRouter = require('./routes/frisreport'); //error
var processCsvRouter = require('./routes/processcsv'); //error
var readFrisReportListRouter = require('./routes/readfrisreportlist'); //error
var checkOutPatientRouter = require('./routes/checkoutpatient');
var checkInPatient2Router = require('./routes/checkinpatient2');

//NEC MegaOAK HR (Data Guard Server: DGORD) via ODBC
var dgordRouter = require('./routes/dgord');
var dgordTestRouter = require('./routes/dgordtest');
var dgordTest2Router = require('./routes/dgordtest2');
var dgordLoopTestRouter = require('./routes/dgordlooptest');
var calcVteRiskRouter = require('./routes/calcvterisk');
var simpleSqlShowRouter = require('./routes/simplesqlshow');
var dvContInpatientRouter = require('./routes/dvcontinpatient');
var simpleSaveRouter = require('./routes/simplesave');
var disasterModeRouter = require('./routes/disastermode');
var rakupicRouter = require('./routes/rakupic');
var megaOakDisasterRouter = require('./routes/megaokdisaster');
var findadRouter = require('./routes/findad');
var findad1Router = require('./routes/findad1');
var findad1saveRouter = require('./routes/findad1save');
var findad1savedataonlyRouter = require('./routes/findad1savedataonly');
var carecomkangoRouter = require('./routes/carecomkango');
var getAppointEntabRouter = require('./routes/getappointentable');
var cartviewRouter = require('./routes/cartview');
var dvTextFindRouter = require('./routes/dvtextfind');
var cartview2Router = require('./routes/cartview2');
var departmentTodayRouter = require('./routes/departmenttoday');
var todayCarteFrmRouter = require('./routes/todaycartefrm');
var dvStatisticsRouter = require('./routes/dvstatistics');
var simpleJsonRouter = require('./routes/simplejson');
var simpleJsonToFileRouter = require('./routes/simplejsonstofile');

//Utility
var simpleSaveRouter = require('./routes/simplesave');
var folderFileListRouter = require('./routes/folderfilelist');
var convertTouF8Router = require('./routes/converttouf8');

//Use Node Express
var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
```

```
routes > simplesqlsontofile.js > ...
1 var express = require('express');
2 var router = express.Router();
3 var fs = require('fs');
4
5 var hrobj = require('./db/dgord@general'); //oracle dgord@ odbc login
6 let result = {};
7 let result2 = [];
8 let str;
9 var iconv = require('iconv-lite')
10
11 //TestRouteの引数変更入力
12 router.get('/', function(req, res) {
13   console.log('simpleshow start');
14   console.log(req.query.word);
15   if (req.query.word){
16     const execute = async () =>{
17       var sqlstr = req.query.word;
18
19       //see MegaOakHRテーブル説明書(H_Phdetail)
20       //CONVERT(UTL_RAW.CAST_TO_VARCHAR2(ODBC_LOG.SUBSTR(DISPLAYTEXT,
21         result = {};
22       const startTime = Date.now();
23       result = await hrobj.myFunc(sqlstr); //execute sql command
24       const endTime = Date.now();
25       console.log('sql process time = ',(endTime - startTime));
26
27       var elements = ['statement', 'parameters', 'return', 'count', 'c
28         for(var key in result){
29           if(elements.indexOf(key) > 0){
30             delete result[key];
31           }
32         }
33       if(result){
34         console.log('result.count = ',result.length);
35         //console.log(result);
36         result2 = [];
37         if(result.length > 0){
38           for(var i = 0; i < result.length; i++){
39             //var str2;
40             //str2 = await result[i].DISPLAYTEXT.replace(/A/g, '');
41             result2.push(result[i]);
42           }
43         }
44         if (req.query.filePath){
45           await res.json(result2);
46           console.log('filePath saved');
47           console.log(req.query.filePath);
48           await fs.writeFileSync(req.query.filePath, JSON.stringify
49             }
50           else{
51             console.log('filePath not assigned');
52             await res.json({"filePath": "notsaved"});
53           }
54         }
55         else{
56           console.log('result.length not exist');
57           await res.json({"filePath": "notsaved"});
58         }
59       }
60       execute();
61     }
62     else{
63       res.json({"result": "not found"});
64     }
65   }
66 });
```

app.js

simplesqlsontofile.js



操作が理解できた方は
周りの方の操作支援をお願い致します

解説：
病院情報システム内でIPCIを利用する
セットアップ・設定の検討

医療情報学の永続性を妨げる要素



医療情報学はなぜこうも「落ち着かない」感じがするのか？

課題点		対応策
低い耐久性/高い独自性	デバイス アプリケーション OS	エッジ側規格化(Linux)
(特に無線) 方式の変化	ネットワーク	802.11系
UIのOS依存	アプリケーション	Web UI
下位互換性の喪失	OS	仮想化/コンテナ化
演算能力	CPU/メモリ	メニーコア化/専用回路
スピンドル脆弱性	ストレージ	半導体ストレージ

どの領域にも「耐久性」の主張が困難 - 維持費ショート問題

IPCIコンセプト導入による開発ワークフローの変化



- 既存型

ユーザーインターフェース
(テキスト/ボタン)

医療ロジックの記述

OS依存開発環境
(例: MS.NET)

OS依存ドライバ

OS

この状況で、OSのバージョンアップが起きると、UIが追従できなくなると同時に、医療スタッフからヒアリングした（普遍性の高い）医療ロジックが継承できなくなり、喪失リスクが生じる

可能な反論：

「医療ロジックだけをUIと別にコーディングしておけばよい」
(Doc-View概念に忠実な実装)

だがUI自体が「医療における安全性ノウハウ」を同時に提供している点において、医療ロジックはUIにも埋め込まれている（ここで医療ロジックとUIの分離あいまい性が生じる）

また現場での「迅速な」実装を求められた際、Doc-Viewはリファクタが困難であった→オブジェクト指向自身の性質に由来

医療機器現場では、さらに「ハードウェアと通信するドライバのOS依存と、記述できるエンジニアの長期雇用の課題」を内包する

IPCIコンセプト導入による開発ワークフローの変化



- 導入後



Web化は「UIのクロスプラットフォーム」をもたらした
→UI自身がOS非依存になり、耐久性に貢献する

医療ロジック自身を「高速なスクリプト言語」かつ
「リファクタが容易になる言語」で記述する
→Node.js
(サーバサイドJavaScript+プロトタイプベースオブジェクト指向)
を活かすことで、耐久性とリファクタ性に貢献する

機器との通信部分はOS以上をすべてコンテナ化することで、ドライバの稼働をOSを更新せずに担保することで、耐久性に貢献する (ただし適切なセキュリティでIOを包むことが必須)

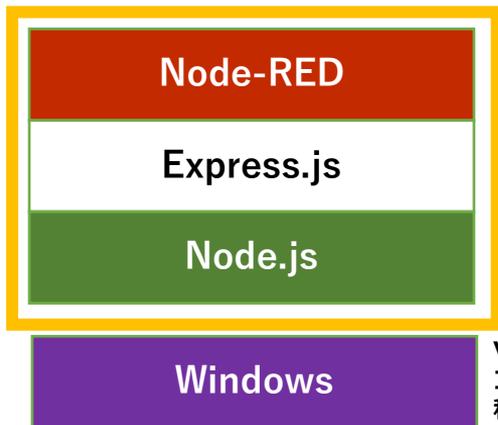
OSS(特にOpenJS系)の特徴と開発方針



- ・基本はスクリプト言語である→ソースコードレベルで移植性が高い
- ・ソフト間の関係性はNode.js+Expressフレームワークに準じている
- ・個人が開発したソースは検証が多くされているものとそうでないものが混在、見分ける必要あり
- ・言語仕様がかわらない、また必要機能が増えない限り遡及した保守は不要
- ・レスポンスとしては低-中速まではRESTでよい、それ以上は単一サーバ内に集約する

- ・これまで推進してきたNode.js+Expressの医療ワークフローとして、同じJavaScript上で完全に（字義通り）統合される

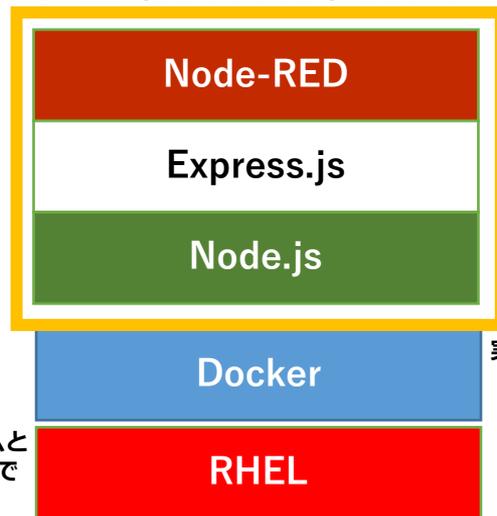
RESTサービス全体（サーバ間通信設定）まで保存したい場合に
Docker（必要に応じてK8）を使う→プログラムの永続性



パッケージとして考える
医療ロジックの記述箇所

RESTライブラリを構成

V8エンジンの上で動いているプログラムと
コンパイラ互換のコードで構成することで
移植性を担保している



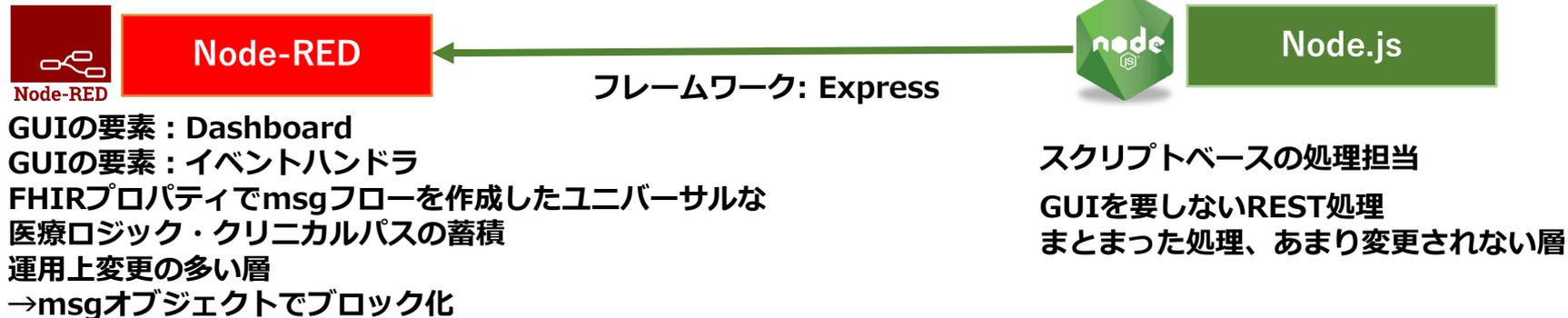
実行速度の低下が少ない
(有償版ではVMのハイパーバイザ)

求める効果：「医療ロジックの永続化」→共通クリニカルパス策定のプラットフォーム、シームレスなIoT/AI連携が見込まれる

Node.jsとNode-REDの開発階層



役割の分担



いずれの処理においても、共通して使用する言語はJavaScript(ECMAScript2015以降,JS)で一貫している
言語選択の重要性 :

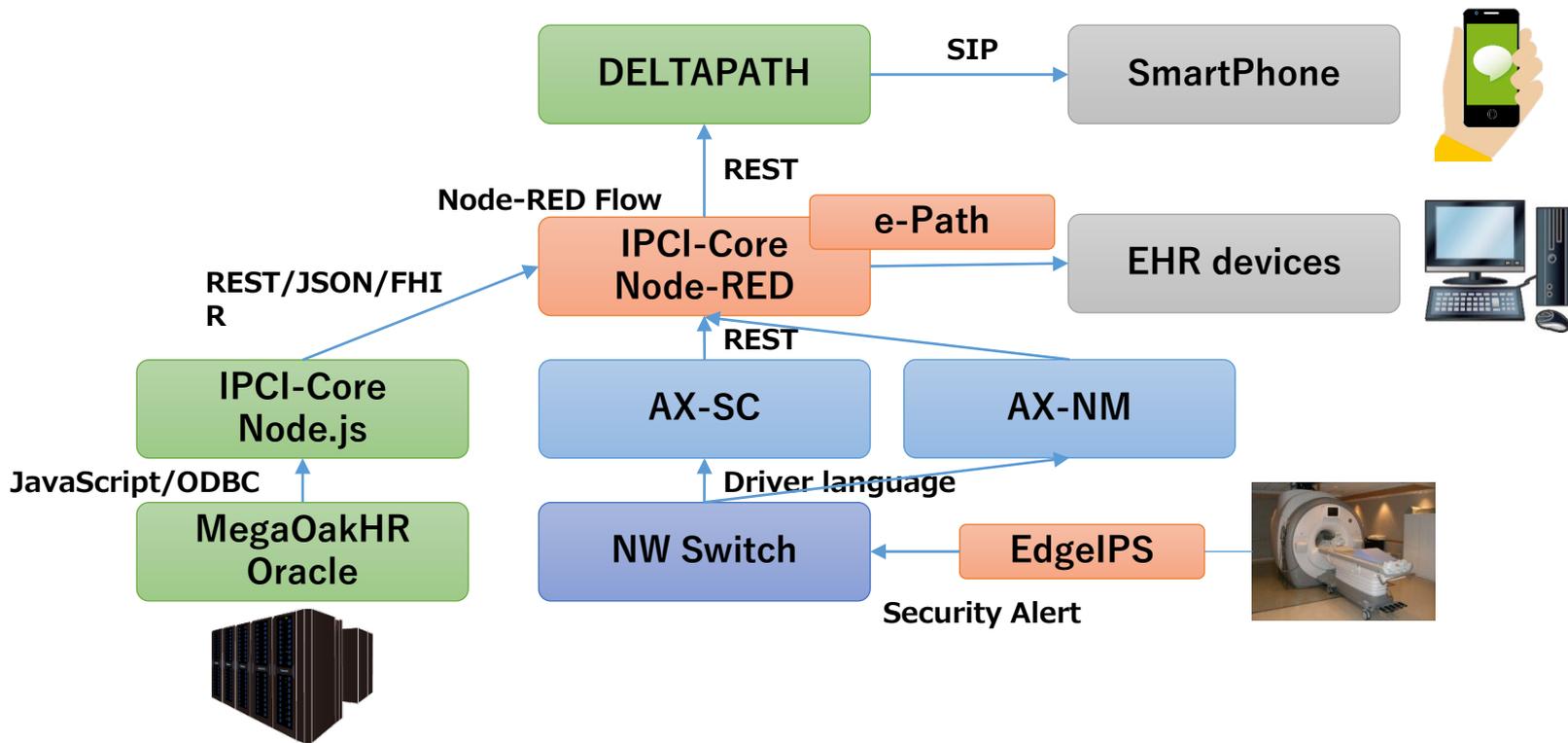
PythonではJSONネイティブな書き方が存在するが、UIの生成については別途HTML/CSS/JavaScriptを組むことになる→処理文法が増える欠点あり

重要な提案は、Node-RED内の医療ロジックを”FHIR準拠形式”で記述していくことで、コードブロック単体での再利用性まで高める

JSON形式の高速データベースの必要性→Mテクノロジー学会主体で開発 (Mアーキテクチャ+ObjectScript) 、ただしその他のデータベース形式にも広く接続性を担保するような活動にすること

JSはスクリプト式でコンパイラよりは演算処理が遅い→コンパイラプログラムをサーバのどこに位置付けるかを定めるハードウェアを含む演算速度の増強が不可欠

手段：IPCIを中核とするロジックシステム・ロジック接続形態の設計



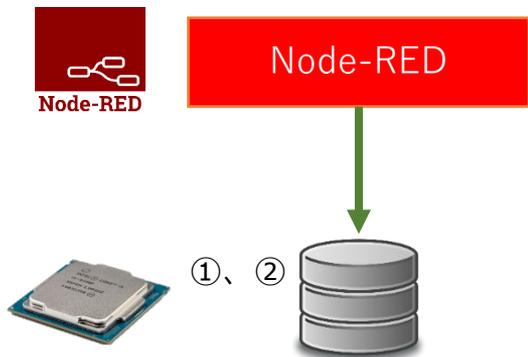
実装上の工夫



Node-REDではrequireでnode_moduleを直接呼び出すコードはfunctionに書けない
対処法

- ①requireを含むスクリプトをフローオブジェクト化
- ②requireを含むスクリプトをコマンドフローから呼ぶ
(ただし戻り値処理が少しややこしい)
- ③RESTでラップしてNode.js-Expressへ投げ、戻り値をJSONオブジェクトとして受ける
(記述はしやすいが、RESTコールはそれほど高速でない)

Node-REDサーバは一つとは限らない

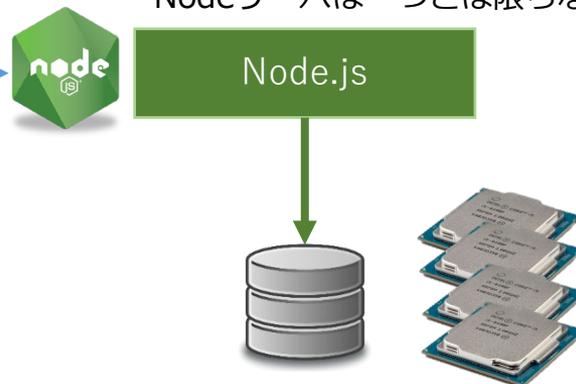


①②は高レスポンスを強く求める場合



ネットワーク性能が決定的に
レスポンスに影響する
→Webプログラミングがもつ
普遍的性質

Nodeサーバは一つとは限らない



③は処理の重いタスクを
分担させるのに向いている



IPCI-Nodeオーケストレーション

「病院まるごとIoT化」計画
 個々のサーバからの機械的情報と医療情報をIoTと同等のレイヤーで操作する
 医療ロジック地震をNode-REDレイヤーに蓄積することで、知的財産の継承を可能にし、
 時間の経過を味方につけられる

Windows UIや
 既存オーダーの仕組みのみ
 再利用

電子カルテ



IPCI-Node
 Node-RED



DELTAPATH frSIP
 RESTでメッセージ生成、通話
 音声認識インターフェース
 スマートフォン通知



ロードバランサーとしても機能する

Node-RED



Node-RED



Node-RED



例：医師がスマートフォンで承認する際
 には、操作者の権限とともに、
 なりすまし防止のトラストエンジン
 （操作権限）チェックが同時に必要

位置情報をRESTで収集



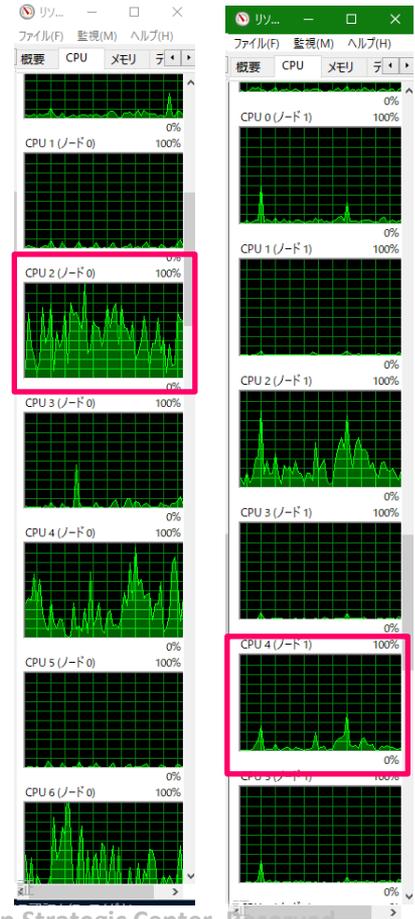
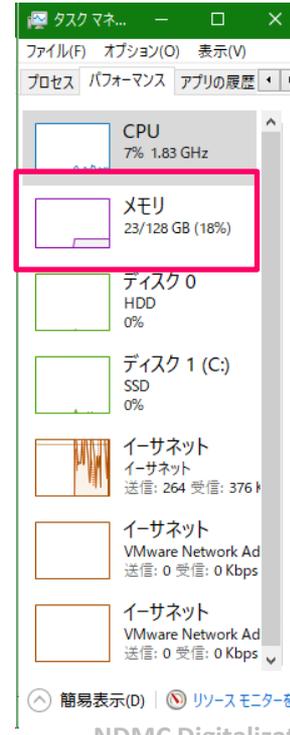
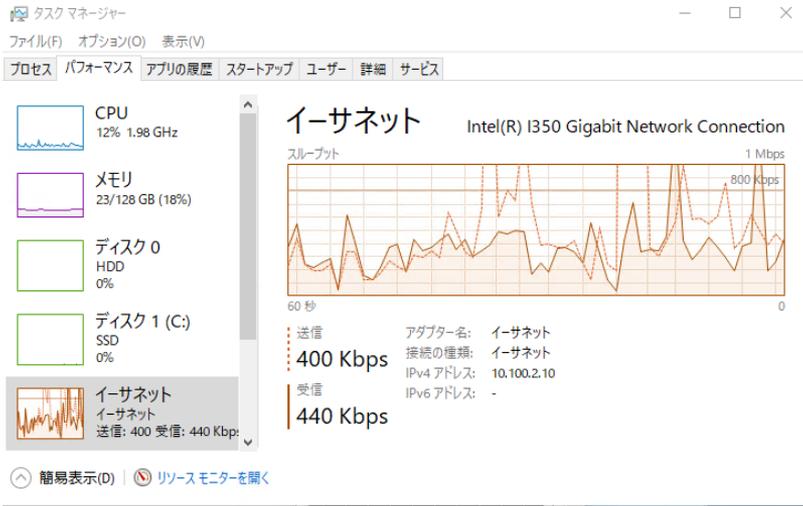
Alaxala



IPCIに適したCPU/メモリ/ストレージ/ネットワークの考え方



- SUPERMICROでIPCI構成をテストした際の検討(物理16コア)
- 非同期処理を行うために、複数コアは必要である
- しかし実例では4コア程度に負荷が集中している
- →動作周波数が高く(できれば常時5GHz)、かつ4-6コア程度のマシンが適切と考える
- メモリは23GB程度であり、最小構成でも32GBあればOK
- ディスク容量は数TBで、増設を可能とする構成が望ましい
- 安価に抑えるなら10Gbps-NASなど
- Thunderboltは高速だが技術的に枯れていない?
- テキストはファイルサイズが小さい→速度より遅延のほうが大きく影響



検討：CUI/GUI/Flowプログラミング、それぞれの利点



アプローチ	CUI	GUI	Flow
言語例	C++ bash	Windows Form マウス操作/RPA	LabView Node-RED
命令単位	テキストファイル	ユーザー操作によるロガー	ノード
コード独立性	完全に独立	汎用化には手入れが必要	フレームワーク上での独立性
シーケンス記述	OK	RPA自身の制御は平易とは言えない	OK
パラレル記述	コード記述にスキルが必要	マルチタスク性という意味ではよい	OK
オブジェクトマニピュレーション	キーボード操作と画面切り替えが煩雑	視覚的で効率よい	ノード単位で可能
データフロー	同一コードに記述すると分かりにくい	ユーザー自身で都度判断	分岐が容易
利用例	数学アルゴリズム的 規則単位の多用	繰り返しが少ない処理（ファイル整理）	処理が分岐する場合 将来変更が多い場合

検討：Node-REDとPythonモジュールの分担について



各モジュールに分担する機能と理由の検討

f function python shell

	JavaScript	Python	exec	VBA	VBScript	各社独自スクリプト
タイプ	スクリプト	スクリプト	スクリプト	スクリプト	スクリプト	スクリプト
動作ベース	Node.js	Python	MS-DOS	Excel	MS-DOS	各ソフト上
ハードウェアアクセス	可	可	可	可	可	可
ML	△	○	×	×	×	(さまざま)
JSON	○	○	×?	△	?	(さまざま)
ロジックGUI	Node-RED	×	×	デバッグ容易	×	独自アプリ上
マルチOS	○	○	×(bashに近い)	×	×	(さまざま)
中核サーバ	Express	http.server Flask	×	×	×	(さまざま)
単体アプリケーション	Electron Angular	Kivy Tkinter PyQt wxPython PySimpleGUI →PyInstaller	.bat MS-DOS窓で可能	VBへの移植以外不可 (画面表示をさせずに実行することは可能)	.vbs	(あまり聞いたことがない)
単体App作成効率	△(Node.jsスクリプトレベルならPythonと効率はそれなり)	○	△	○	△	×(独自言語文法の習得が必要)
WebApp作成効率	○(但し単体Appより低い)	△(HTMLを書くことになる)	×	△-×(HTMLを書くことになる)	×	(さまざま)

プロトタイプベースオブジェクト指向の特長



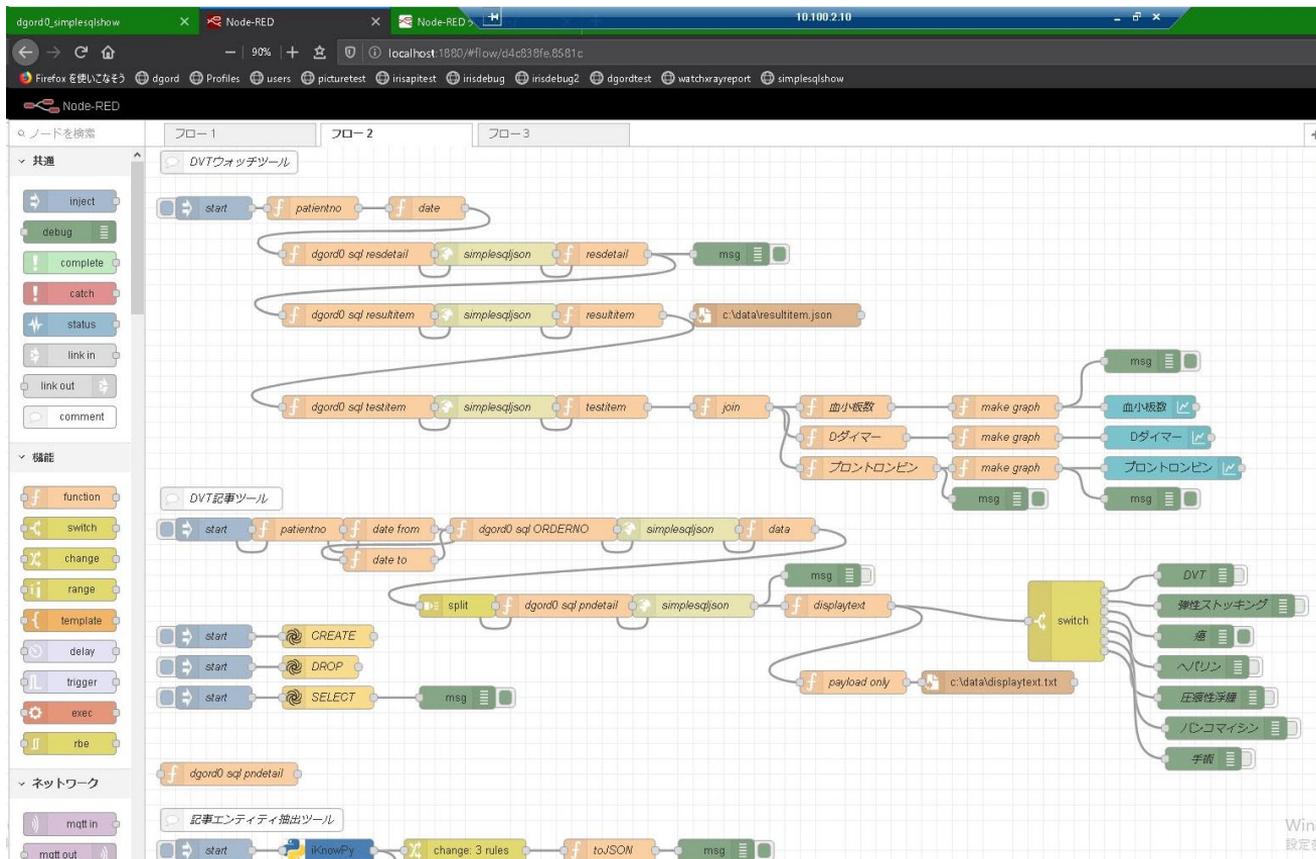
InterSystemsがGitで公開しているiKnowPyを呼び出し（ファイルで渡し、結果はmsg.payloadで戻せる）
将来的な機械学習機能の実装にも対応できる

The screenshot displays the Node-RED web interface. In the center, a workflow is visible with several nodes. One node, labeled 'iKnowPy', is highlighted with a red rectangular box. To the right of the main workspace, a 'python-shell' node editor is open, showing the following Python code:

```
python-shell ノードを編集
削除 中止 完了
プロパティ
名前
iKnowPy
コード
1 import iknowpy
2 import sys
3 import codecs
4 import json
5 engine = iknowpy.iKnowEngine()
6
7 #sys.stdout = codecs.getwriter('utf-8')(sys.stdout)
8 #reload(sys)
9 #sys.setdefaultencoding('cp932')
10
11 #index text
12 text = '新型コロナウイルスの121件の感染拡大に伴い、'
13 f = codecs.open('C:\data\displaytext.txt', 'r', 'utf-8')
14 text = f.read()
15 engine.index(text, 'ja')
16 f.close()
17 text2 = json.dumps(engine.m_index)
18
19 print(engine.m_index)
```

フロープログラミングの特徴

医療ロジックや複数の情報要素を束ねる際に有効な手法
コーディングでは表現しづらい並列処理や、変換ロジックだけのモジュール化などが容易



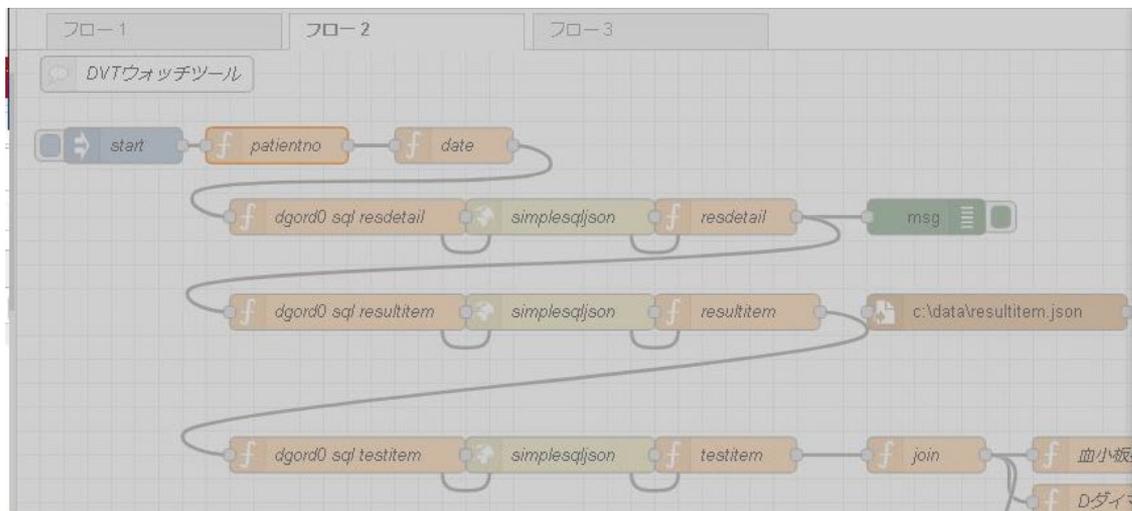
プロトタイプオブジェクト指向の特長



動的に要素を定義、追加、編集、削除できる

Node-REDの中ではmsgオブジェクトを（多くはmsg.payload）使ってメッセージ処理を行う

例：患者ID、期間を指定して、検査結果をMegaOAK HRのDBから抽出し、Dダイマー等のフィルタで分類し、これらをグラフに表示するデモ



function ノードを編集

削除 中止 完

プロパティ

名前 patientno

初期化処理 コード 終了処理

```
1 msg.pid = '06219282';
2 return msg;
```

**患者ID用にmsg.pidを動的に指定
日付はmsg.dateで指定している**

SQLのみをロジック化する手法

msg.payload中にSQL構文を記述



```
function ノードを編集
  削除 中止 完了
  プロパティ
  名前 dgord0 sql resdetail
  初期化処理 コード 終了処理
  1 sqlstr = `
  2 SELECT
  3 *
  4 FROM RESEDETAIL
  5 WHERE
  6 PATIENTNO = '' + msg.pid + ''
  7 AND
  8 UPDATEDATE > TO_DATE('' + msg.date + '', 'YYYY-MM-DD')
  9 ;
  10 `;
  11 msg.payload = { "word": sqlstr };
  12 return msg;
  13
```

ODBCラッパーとDB接続



Node.js + Expressで作成したREST-ODBCラッパーにSQL構文を渡し、REST/JSONで戻す
同一筐体内にポートの異なる2つのサーバがある状態



ODBC Connector

dgord0_simplesqshow

Node-RED

localhost:1880/#f1ow/d4c838fe.6581c

http request ノードを編集

URL http://localhost:3000/simplesqjson

出力形式 JSONオブジェクト

名前 simplesqjson

msg.resdetailの生成



msg.resdetailにオブジェクトを載せ替え

The screenshot displays a workflow editor interface. The main workspace shows a sequence of steps: 'start' followed by 'patientno' and 'date' functions. A loop contains three parallel paths: 1) 'dgord0 sql resdetail' -> 'simplesqljson' -> 'resdetail' -> 'msg'; 2) 'dgord0 sql resultitem' -> 'simplesqljson' -> 'resultitem' -> 'c:\data\resultitem.json'; 3) 'dgord0 sql testitem' -> 'simplesqljson' -> 'testitem' -> 'join'. The 'resdetail' step is highlighted with a blue dot. To the right, a 'FUNCTION' window is open, showing the 'resdetail' function's code:

```
1 msg.resdetail = msg.payload;  
2 return msg;
```

The code is enclosed in a red rectangular box. The function window also includes buttons for '削除' (Delete), '中止' (Cancel), and '完了' (Done), and tabs for 'プロパティ' (Properties), '名前' (Name), '初期化処理' (Initialization), 'コード' (Code), and '終了処理' (Finalization).

結果一覧から目的の検査項目を抽出する方法



検査結果一覧から特定の項目（ここではDダイマー）をフィルタする書き方

The screenshot shows a Node-RED workflow in a browser window. The workflow is organized into three flows:

- フロー-1:** Starts with a 'start' node, followed by 'patientno' and 'date' nodes.
- フロー-2:** Contains a 'dgord0 sql resdetail' node connected to a 'simplesqjson' node, which is then connected to a 'resdetail' node. The output of 'resdetail' is connected to a 'msg' node.
- フロー-3:** Contains a 'dgord0 sql resultitem' node connected to a 'simplesqjson' node, which is then connected to a 'resultitem' node. The output of 'resultitem' is connected to a file node 'c:\data\resultitem.json'. Below this, there is a 'dgord0 sql testitem' node connected to a 'simplesqjson' node, which is then connected to a 'testitem' node. The output of 'testitem' is connected to a 'join' node. The 'join' node has two inputs: one from the 'resultitem' node and another from a 'Dダイマー' node. The output of the 'join' node is connected to a 'msg' node.

The right-hand panel shows the 'function ノードを編集' (Edit Function Node) interface. The name of the function node is 'Dダイマー'. The code editor contains the following JavaScript code, which is highlighted with a red box:

```
1 - var result = msg.resdetail.filter(function(item,index){
2   var str = item.TESTITEMCODE;
3   if(str.indexOf('227300') > -1){return true;}
4   //以下条件を書き連ねてよい(OR)
5 - });
6 msg.out=result;
7 return msg;
```

Dashboard機能



Node-RED上に実装されている機能

グラフテンプレートとデータ構造の記述の仕方

名前: make graph

初期化処理 コード 終了処理

```
1 var temp = [{
2   "series": ["prontronbin"],
3   "data": [[]],
4   "labels": [""]
5 }];
6 for(var i = 0; i < msg.out.length; i++){
7   var plotdata = {"x" : msg.out[i].UPDATE_DATE, "y" : parseFloat(msg.out[i].TESTRESULT)};
8   temp[0].data[0][i] = plotdata;
9 }
10
11 msg.payload = {};
12 msg.payload = temp;
13 return msg;
```

DVT検査指標

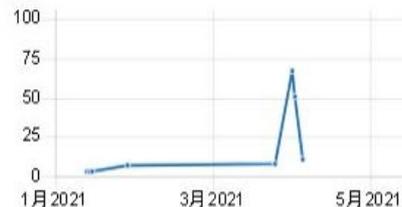
プロントロンピン

プロントロンピン



Dダイマー

Dダイマー



血小板数

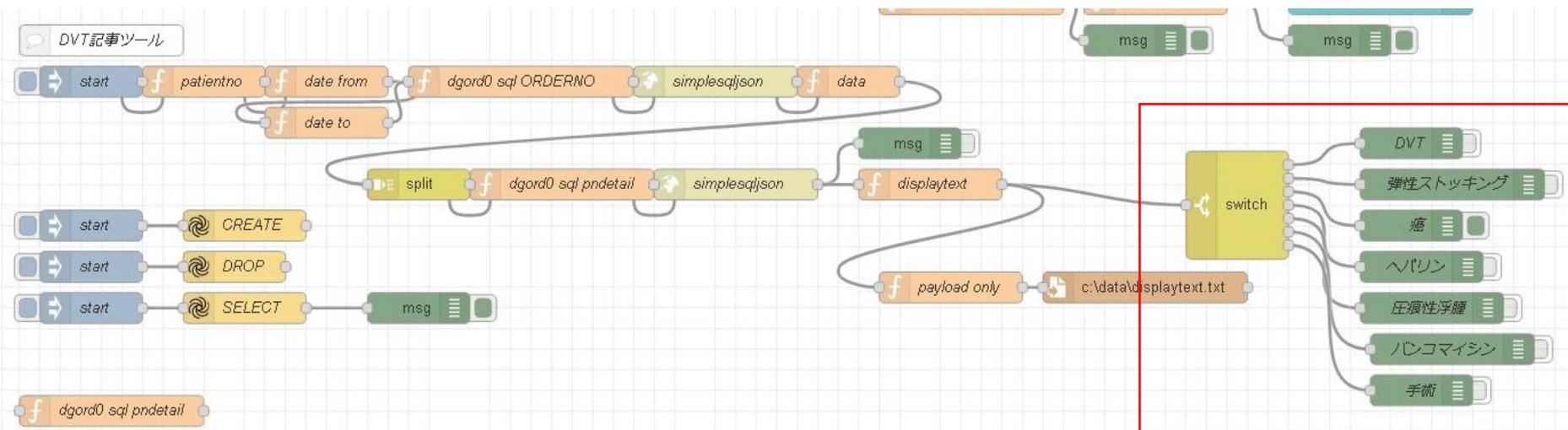
血小板数



記事中のキーワードを用いたスイッチアクション



患者ID、期間をもとに診察記事を抽出し、含まれるキーワードによってフラグを複数生成するデモ
このようなコード体系では、運用後のロジック追加の際に機能全体の稼働保証を憂慮せずに済むメリットが大きい



プロトタイプオブジェクト指向の特長



InterSystemsがGitで公開しているiKnowPyを呼び出し（ファイルで渡し、結果はmsg.payloadで戻せる）
将来的な機械学習機能の実装にも対応できる

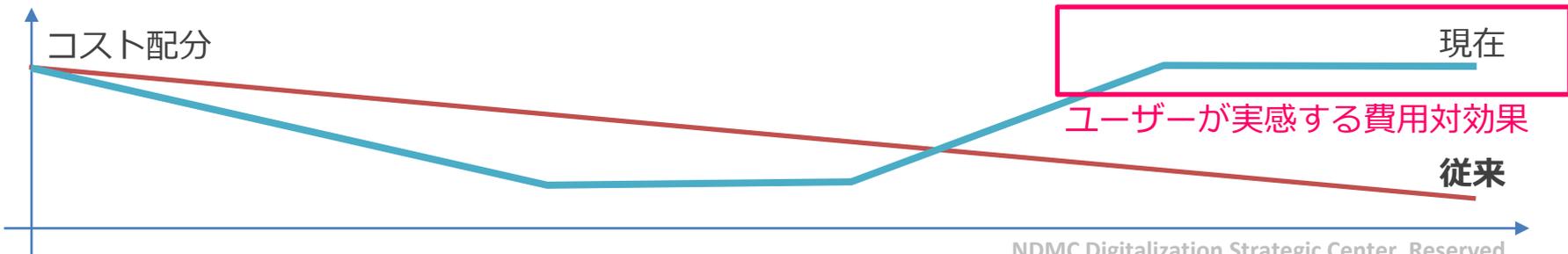
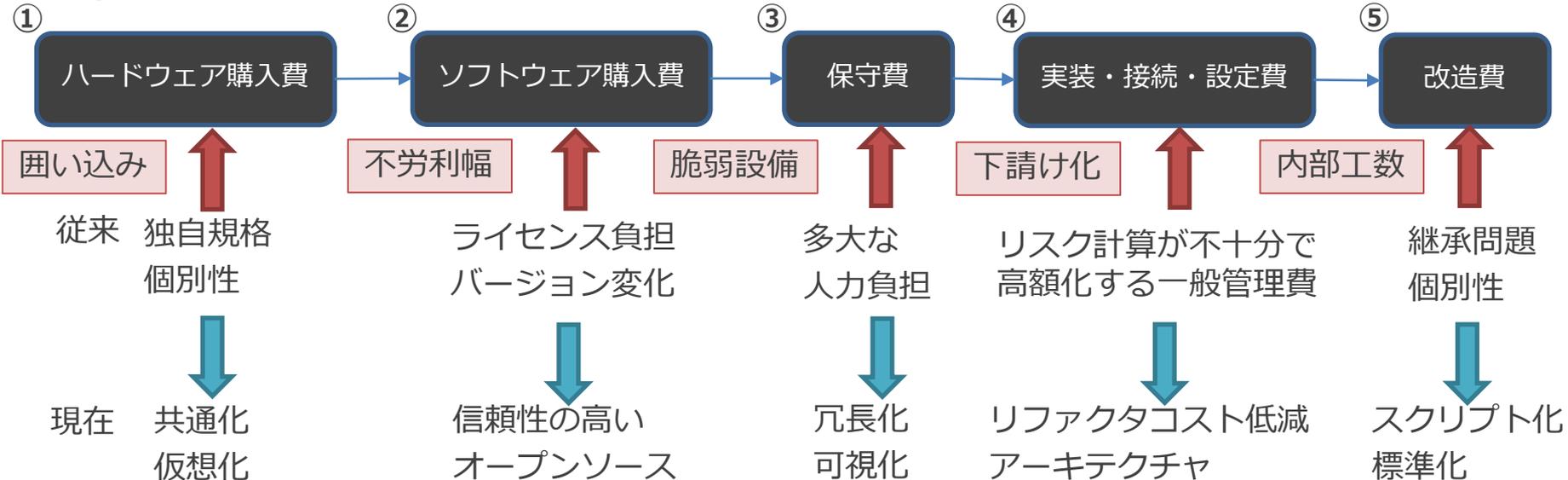
The screenshot displays the Node-RED web interface. The main workspace shows a workflow with several nodes. A node labeled 'iKnowPy' is highlighted with a red box. The right-hand panel, titled 'python-shell ノードを編集', shows the Python code for this node:

```
1 import iknowpy
2 import sys
3 import codecs
4 import json
5 engine = iknowpy.iKnowEngine()
6
7 #sys.stdout = codecs.getwriter('utf-8')(sys.stdout)
8 #reload(sys)
9 #sys.setdefaultencoding('cp932')
10
11 #index text
12 text = '新型コロナウイルスの121件の感染拡大に伴い、'
13 f = codecs.open('C:\data\displaytext.txt', 'r', 'utf-8')
14 text = f.read()
15 engine.index(text, 'ja')
16 f.close()
17 text2 = json.dumps(engine.m_index)
18
19 print(engine.m_index)
```

IPCIコンセプトによる医療情報ビジネスへのインパクト



“Right Business” = “人間が時間・技能をかけて労働する内容に合わせた費用拠出をする”方針の推進



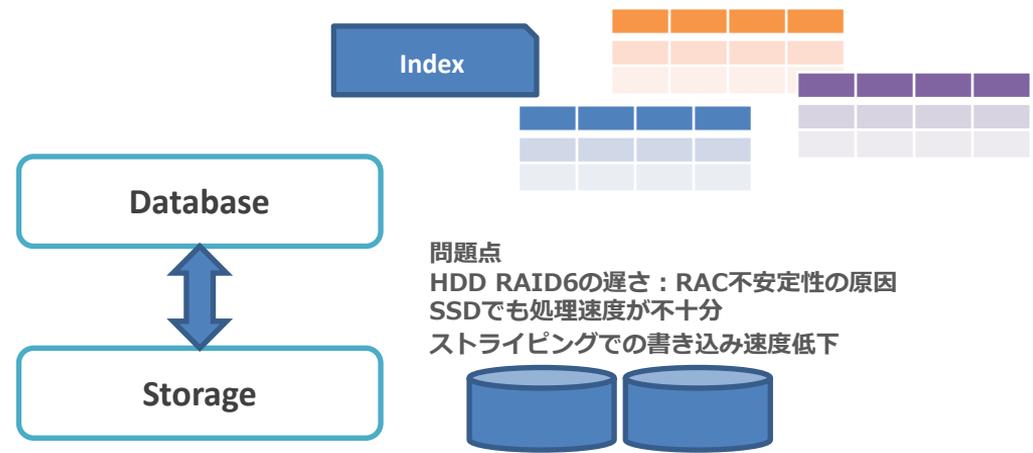
DBのindex構築の属人性、医療データベースが抱える問題、根本改善につ



医療に限らないデータベース設計の価値

- ・高レスポンス→DWH問題への解
- ・SQL構文のシンプルさ→属人性からの離脱
- ・保守・バージョンアップ省力化

問題点
Indexの最適化問題
技術者の減少
DB数が多大
Oracle独自API
改造工数の肥大化



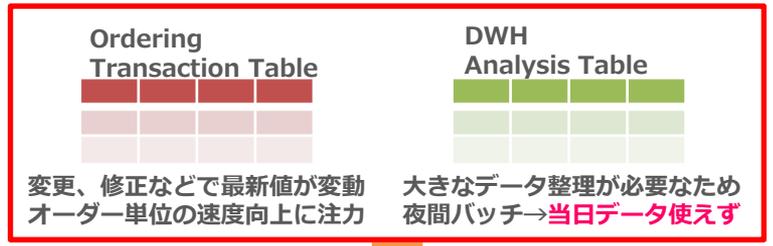
問題点
HDD RAID6の遅さ：RAC不安定性の原因
SSDでも処理速度が不十分
ストライピングでの書き込み速度低下

大幅な改造の停止
改造工数抑制

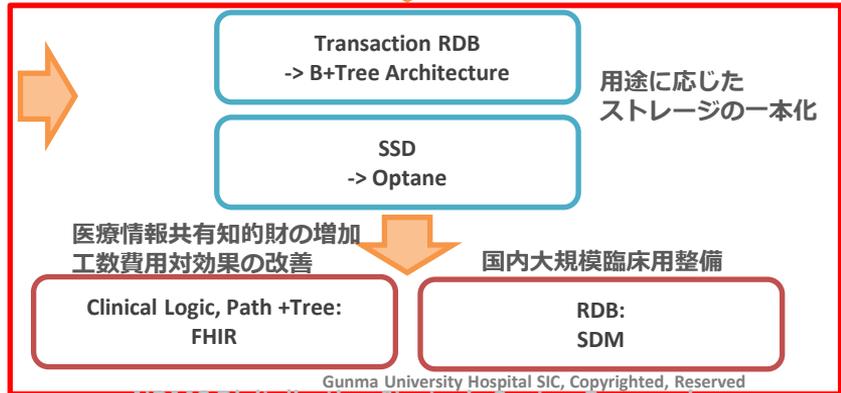
群大病院としての
中期的データ保持形態案

GUI/Transaction
Vender-Made DB

かつての取り組み：
別途DWHを持つ方法：DB容量の2重投資、SQLテーブルの複雑さが障壁
→集計したいデータ項目が分かりにくく、分かったとしても実用的な時間で抽出できない



IPCIコンセプト提案





The Tutorial is Completed
お疲れ様でした