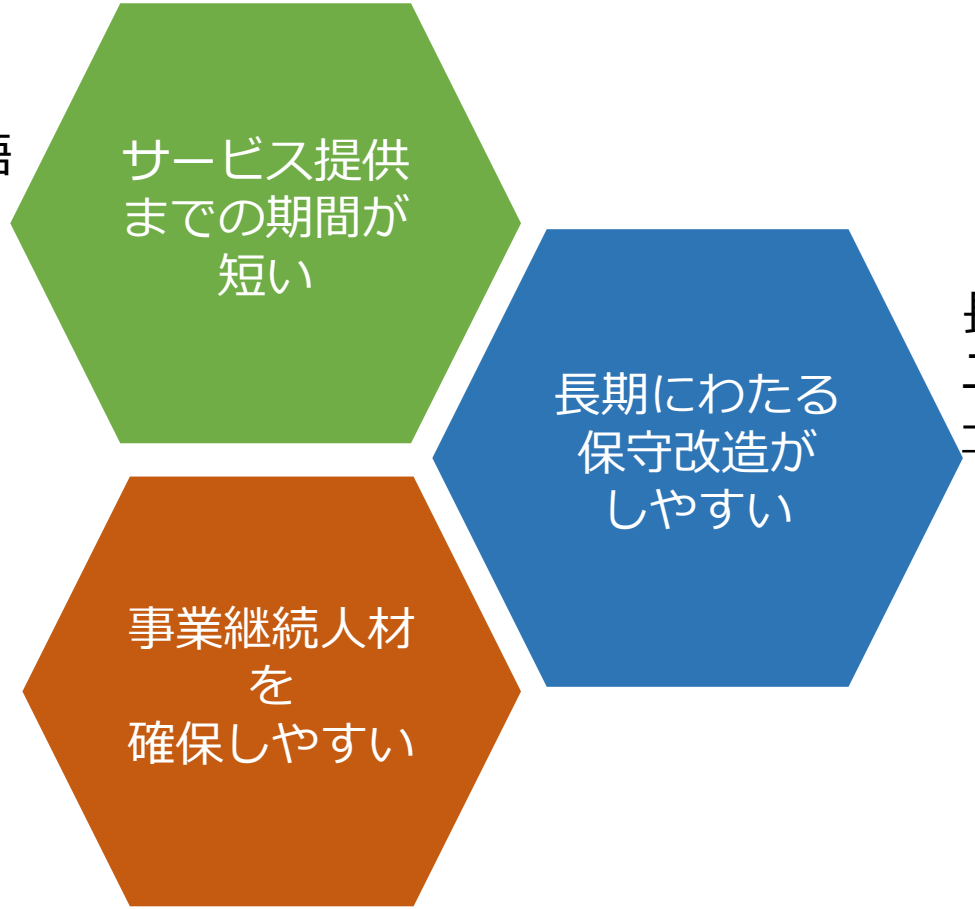
HL7セミナー
Node.jsでWebサーバを作る

群馬大学医学部附属病院システム統合センター
防衛医科大学校デジタル化推進本部推進補佐官
鳥飼　幸太

重要な背景:
開発者はどのように開発言語を選ぶのか

システム環境に求められる特徴の衝突

ライブラリの充実した言語
新しいサービスに特化した言語
→新規言語？



長期的に使われている言語
ユーザー数の多い言語
→古参の言語？

人気のある言語
単純な言語
→独自でない言語？

「医療ソフトウェア」のライフサイクルと経済性

医療ITが
有する素因

- 医療の現状は「半分市場、半分福祉」？
- ITベンダーが本来収益性を高くできるのは、「ソフトウェアは容易に複製できる」から
- ＜医療ノウハウ＞の詰まったコードを「棄てざるを得ない」事情がベンダーの収益性の障壁になってきた

病院で利用されるベンダー製ソフトウェアの始まりから終わりまで

「試してくれる」
医療者と
コンタクトする
機会の獲得

大学病院：医療情報部
診療科
民間は手探り
凄い苦労とコスト

医療者に
ITベンダーが
ヒアリング

言語が通じ合わない壁
医療知識のベース＋
ITスキルのベース
凄い苦労とコスト

テストソフトの
作成

開発コスト
＜医療ノウハウ＞
捻出元が必要
優れた
アーキテクトが
寿命を決める
多くはボツ、歩留不良

「注文の多い料理店」
＜医療ノウハウ＞

機能追加
接続変更

バグ改修

「非常に高い信頼性」
→改修が強く忌避される理由

実地検証

医療現場IT能力
凄い苦労とコスト

病院が
「ソフトに診療を合わせず、
診療にソフトを合わせる」
よう要求

他サイト拡販

OS等
環境変化
追従の移植

使えなくなる関数やコード
を探して移植するコストが
ユーザーに理解されない
開発主の離職

ローカル
バーション
保守困難
→終売

貴重なヒアリングや
医療現場知見が棄てられる

事業継続

移植困難
→終売

貴重なヒアリングや
医療現場知見が棄てられる
知見が積みあがらず
同じ事の繰り返し
開発者が楽にならない
コスト回収が困難に
→医療IT最大の問題

知的労働としても、費用としても、期間としても多額の投資
（電子カルテでは＞20年）

ソフトウェアの収益性

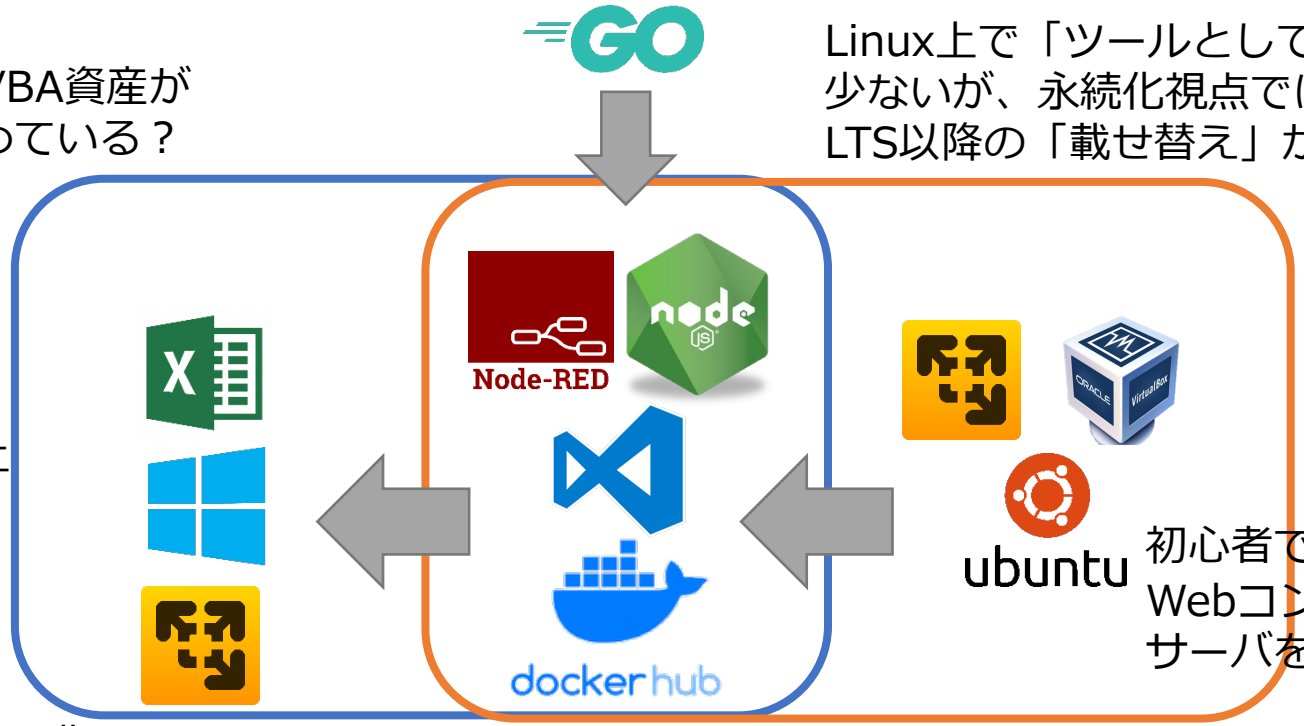
多くの電子カルテベンダー、PACSベンダーなどが、Web化に際して古いプラットフォーム（＋ノウハウ資源）を棄てた
新しいプラットフォームのアーキテクトは「標準化対応するには複雑独自に作りこみ過ぎた」状態？

考察：永続化と利用ユーザ維持の両立

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

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

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？



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

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

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

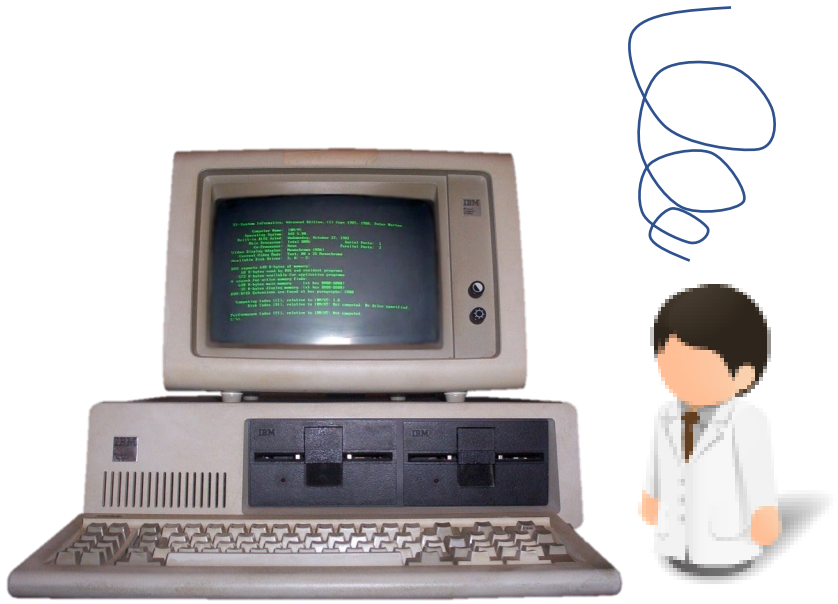
pip/npmなどのコマンドインストーラが再び
浸透してきた

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」

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

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

診療録入力で支援できること



診療を行う時間よりも、情報を探し、
整理し、記載する時間の方が長い？

病棟から電話呼び出し

病室に向かい、
患者に耳を傾ける

診察・診療を行う

病棟や医局まで歩いて戻る

診療録記録のために、
PCを立ち上げる

電子カルテにログインする

患者IDを入力する

検査記録が出ているかを探す

診療録アプリケーション
立ち上げ

診療録記載

ログオフ

多大な待ち時間
頻回な操作を
減少させる工夫

シンクライアントノート端末や
スマートフォンの導入

顔や虹彩認証、ICカード

検査結果Push通知
検索履歴の活用
機械学習による推論

音声記録の下書き活用
LLMによるサマリ生成

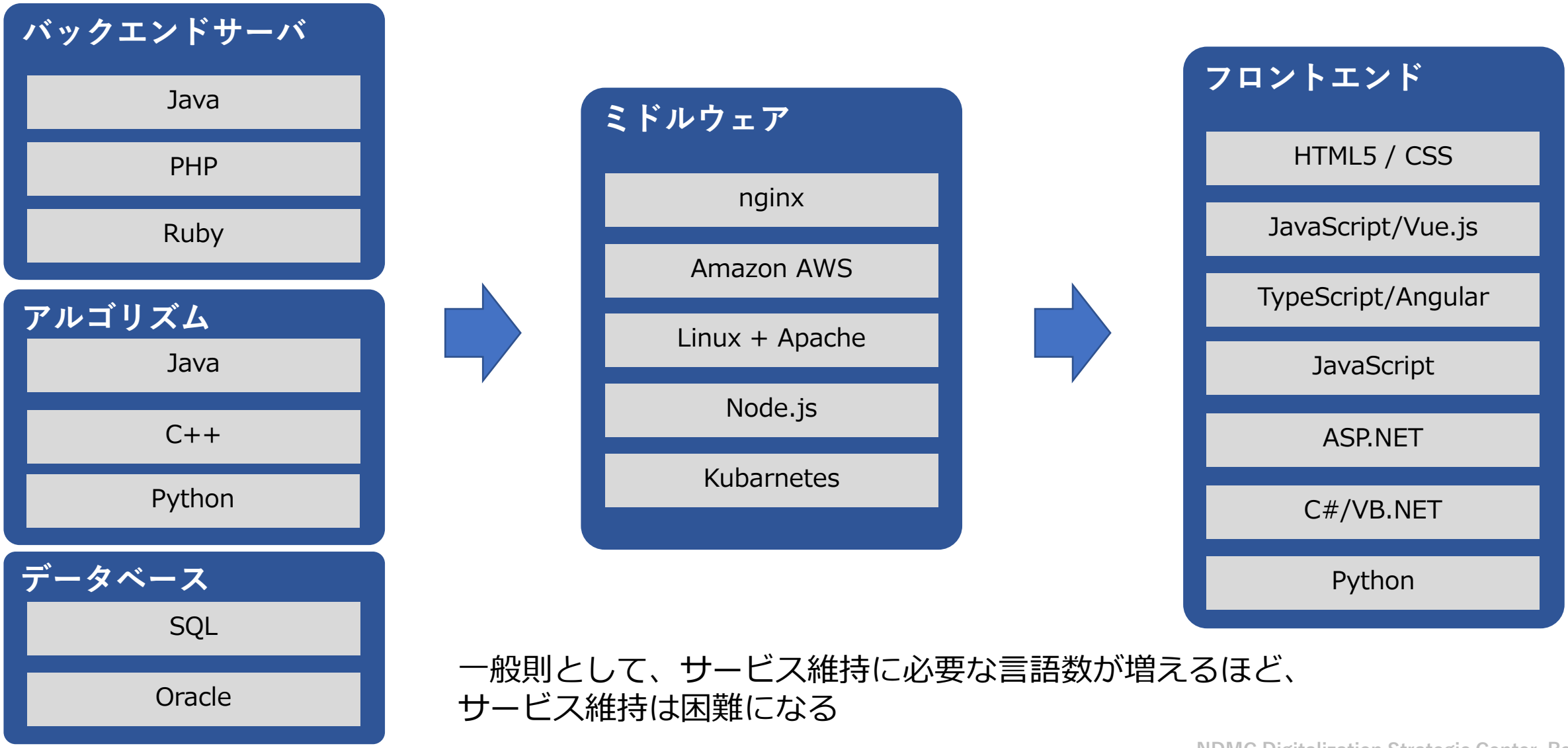
いわゆる”Web型開発”について

Web開発はコンピュータ市場の知恵の精錬である

- サーバとダム端末の時代：ストレージが一か所
- 分散型開発：フォーマットがばらばら、協調しない
- 集中型開発：型が古くなる、管理コストが合わない
- Web型開発：REST, URI, Hookなどを使って、上記の欠点を克服してきた

Web系：バックエンドからフロントエンドまで

- どの箇所に、どのような言語を、幾つ使うかの構成が、アーキテクト設計の要諦



一般則として、サービス維持に必要な言語数が増えるほど、
サービス維持は困難になる

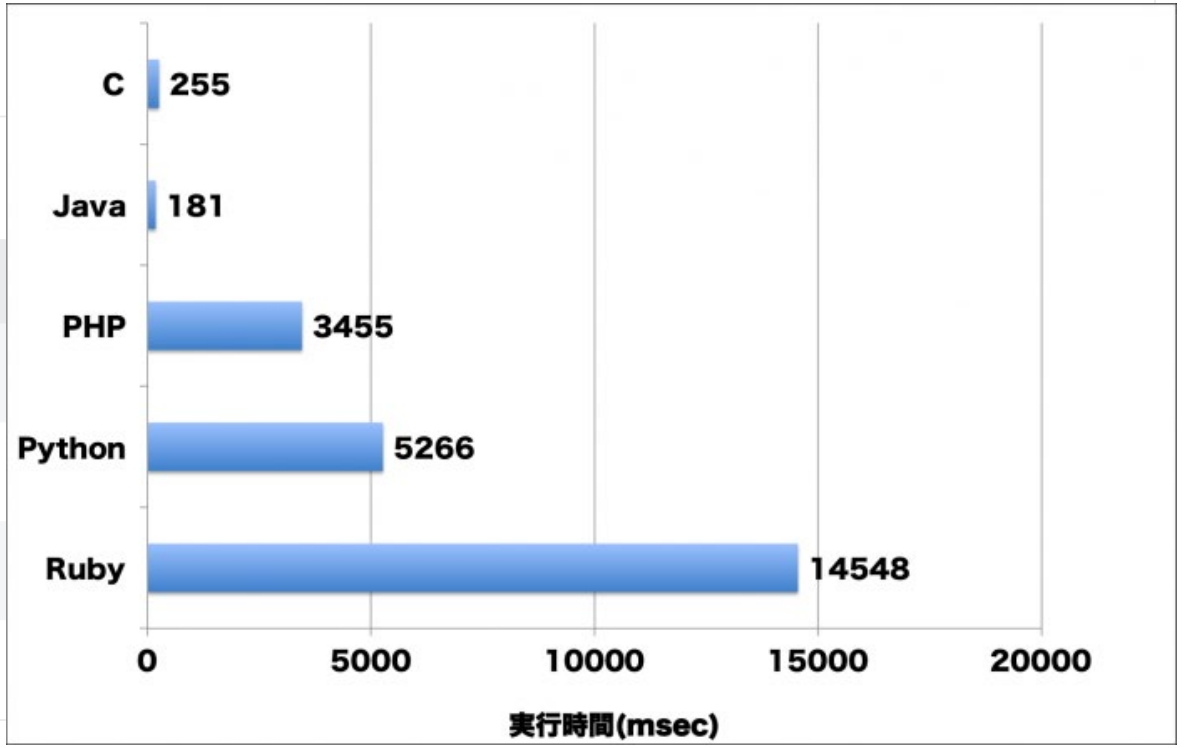
クラウド：時代は、改めて「アルゴリズムの良さ＝軽さ」を求めている

Azure ・ クラウド上の「仮想マシン」のディスク価格

ディスク サイズ (GiB)	4	8	16	32	64	128	256	512	1,024-65,536 (1 TiB ずつ増加)
最大 IOPS	1,200	2,400	4,800	9,600	19,200	38,400	76,800	130,000	
最大スループット (MB/秒)	300	600	1,200	2,400					

*ディスクを構成する方法の詳細については、Ultra Disk のドキュメント ページをご覧ください。

Ultra ディスク構成	単位
ディスク容量 (GiB)	GiB
プロビジョニング済み IOPS	IOPS
プロビジョニング スループット (MB/秒)	MBps
プロビジョニング済み vCPU 予約料金*	vCPU

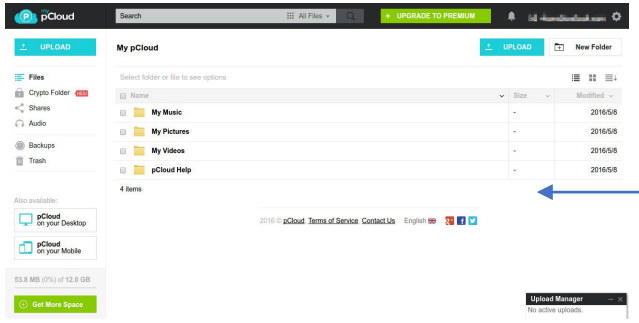


https://web.archive.org/web/20140802192155/http://maplesystems.co.jp/blog/all/programming/2015

よくあるWebアプリケーション（＝小さなサーバ）

- 古い言葉では「割り込み待ちの無限ループ」プログラム

コマンドプロンプト（ターミナル）で
Webアプリのプログラムを実行



IPアドレス
ポート設定

常時http呼び出し待ち状態

http://localhost:3000/

ルーティング
コード

引数が何もない場合
（デフォルト）

FHIR
サーバ

デフォルト
処理コード

表示に必要なデータを渡す

ビュー生成
コード

HTML+CSS+JavaScriptのセットが送られる

フレームワークは何をしてくれるのか

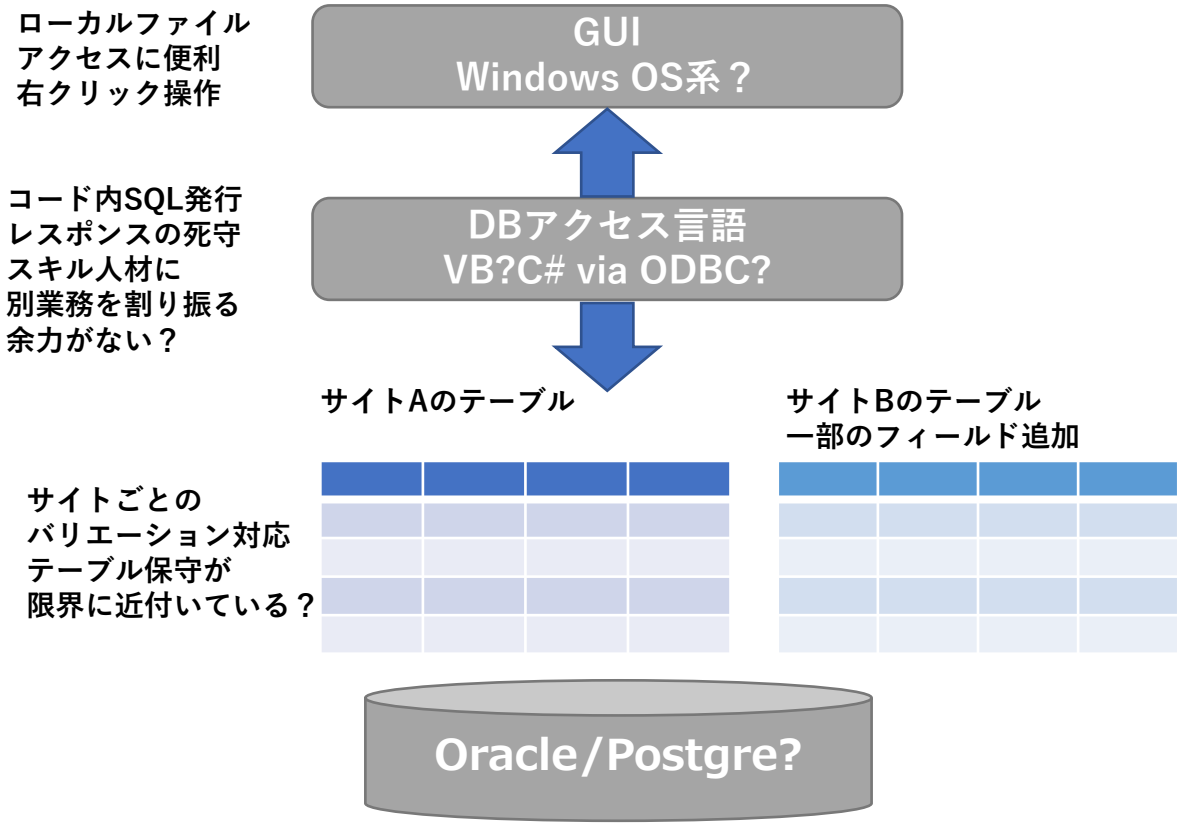
- プログラムをMVC(Model-View-Controller)に近づける支援
- 頻用される機能を短いコード量で記述できるように簡略化
- データ構造（アルゴリズム、ルーティング、ビュー）の配置と暗黙的リンクの作成
- 独自の文法・単語制約（標準以外の関数名、予約語など）が追加される
- 作成したプログラムの寿命が、言語由来ではなくフレームワーク由来になる
- チームコーディングにおいては１からデータ構造を思案するよりも効率的

言語の選択とFHIR/JSONの活用

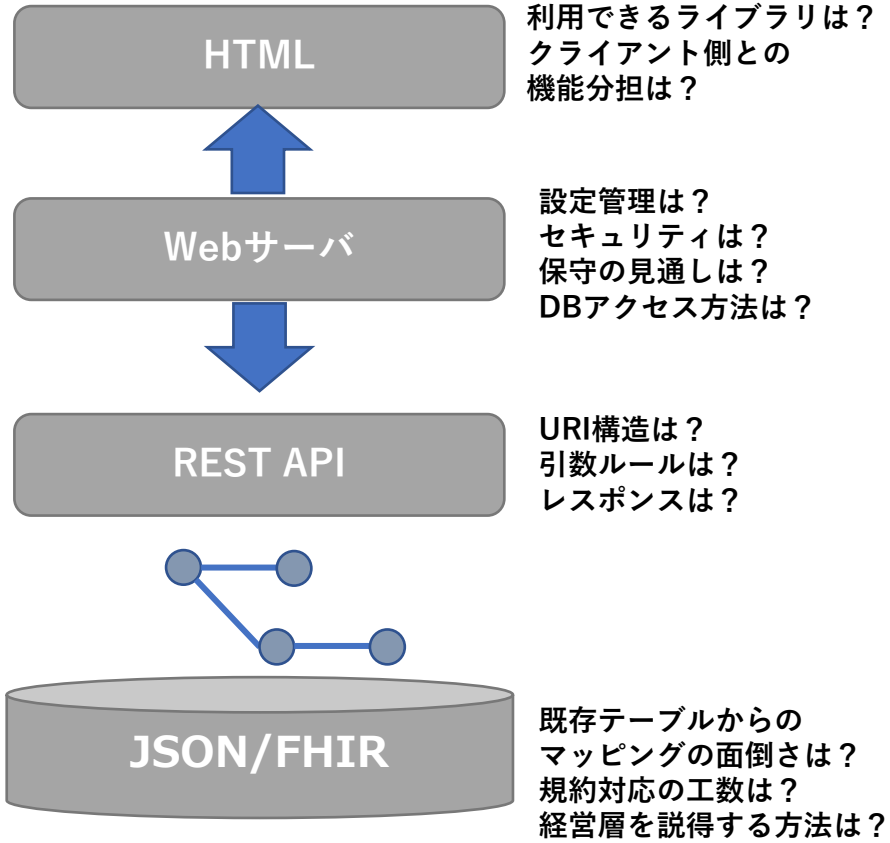
（想像の）医療情報システム構造のWeb化の課題

- **私がCIOだったら感じること：Web化モデルとは、既存のシステム構造とは似ても似つかない？**
- **途方もない工数（とストレス）がかかるような恐怖感**

把握している現状



提案されたモデル



ショートサマリ：データ分析とアルゴリズム記述

データ分析

- 分析目的・仮説の明確化
- データの収集
- データのクレンジング
- データの加工
- データの分析
- 仮説検証と探索的発見

アルゴリズム記述

- 記述に必要な文法が少ない
- オープンソースソフトウェア
- ライブラリの充実
- Git等を通じた版管理
- コード記述の適切な棲み分け
- 分析関連ツールとの連携

個人的な2大言語：ECMAScriptとPython

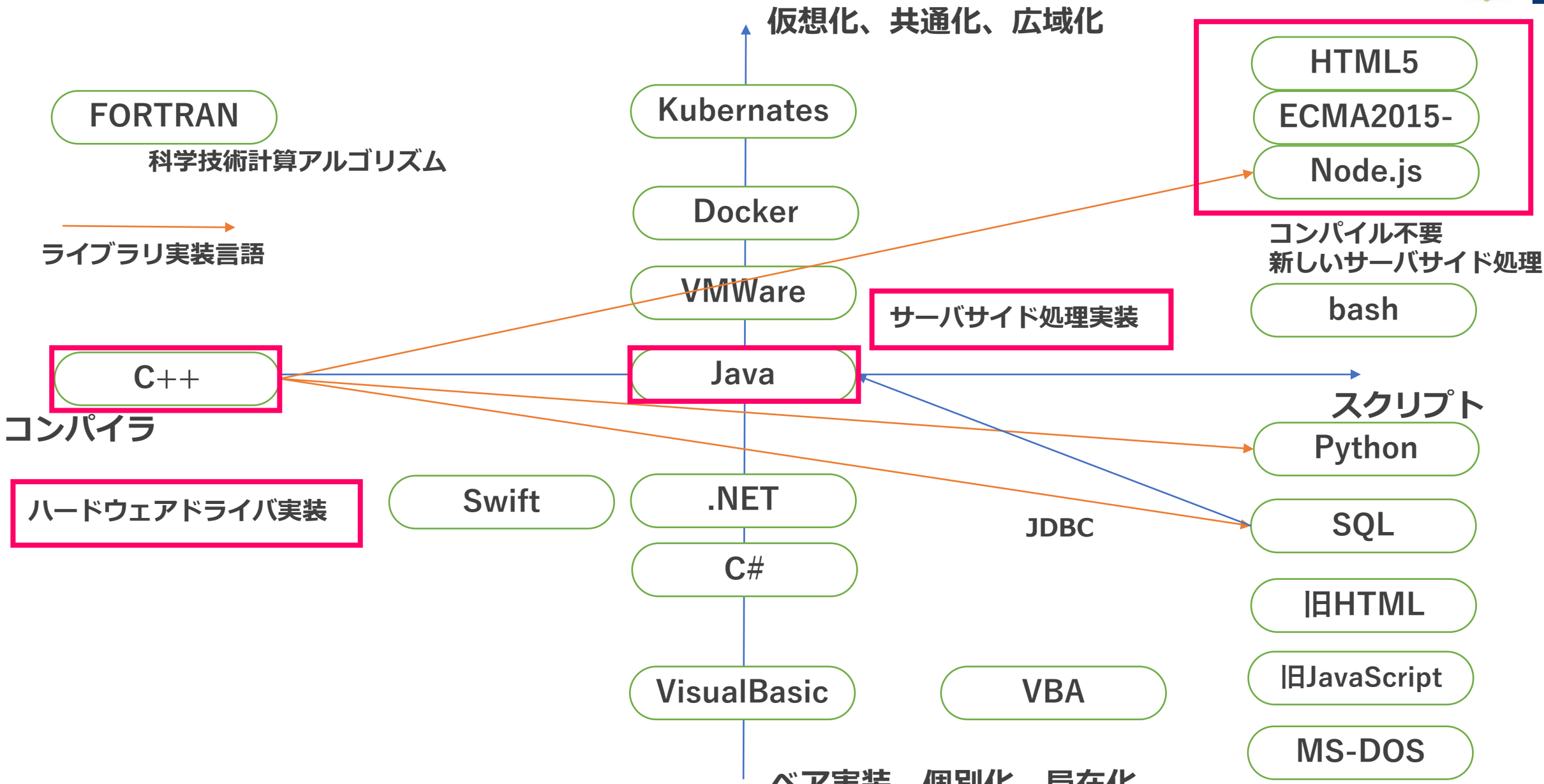
ECMAScript

- C言語的文法
- 非同期が標準仕様(Node.js)
- プロトタイプオブジェクト指向
- HTML5(UI)との親和性
- 充実し続けるライブラリ
- Web系グルー言語候補

Python

- Fortran言語/C言語に部分的に類似
- 3.X系から言語が安定
- 必要ならオブジェクト指向で構成可
- 機械学習との親和性
- Git等を通じたライブラリの充実
- バックエンド用グルー言語の候補

採用言語の検討：計算機資源と言語における仮想化と永続化の可能性



内容	C++	VisualBasic	ECMAScript	Python

変数宣言	int i;	Dim i as Integer	var i ; //型宣言なし
			let i //セミコロンなしでも動く
				宣言不要

関数、引数、戻り値	（定義：戻り値なし）
	void fct(int i, int *j){
	}
	（定義：戻り値あり）
	int fct(int i, int *j){
	return nInt;
	}
	
	使用(戻り値なし)
	cls1::.fct(i, &j);
	使用（戻り値あり）
	k = cls1.fct(i, &j);
		（定義：戻り値なし）
		Sub fct(byval i as integer,
		byref j as integer)
		End Sub
		（定義：戻り値あり）
		Function fct(byval i as integer,
		byref j as integer) As Integer
		Return nInt
		End Function
		使用（戻り値なし）
		cls1.fct(i, j)
		使用（戻り値あり）
		k = cls1.fct(i, j)
			(定義：戻り値なし)
			function fct(i, j)
			{}
			
			(定義：戻り値あり)
			function fct(i, j)
			{ return nInt; }
			
			使用（戻り値なし）
			fct(i, j);
			使用（戻り値あり）
			k = fct(i, j);
				定義
				class cls1:
				def fct(i, j=0):
				return nInt
				#タブで関数内記述
				
				使用
				k = cls1.fct(i)

値の代入	i = 15;	i = 15	i = 15;	i = 15
ライブラリ関数 特殊変数	include<cmath> f = cos(0.25);	f = Math.Cos(0.25)	f = Math.cos(0.25);	import math math.cos(0.25)
マクロ	#define DEBUG 1	#Const DEBUG=1	キーワードなし	キーワードなし（大文字表記が慣例）
クラス内オブジェクト	this->obj	Me.obj	this.obj（引数）	self.obj
コマンド(ターミナルで用いる)	system(“C:test.txt");	System.Diagnostics.Process.Start(objProcIfm)	console.log(“samplestring”)	subprocess.call(“C:test.txt")

繰り返し	int i;
	for(i = 0; i < 10; i=i+1){
	}
		Dim i as Integer
		For i = 0 to 10 Step 1
		Next i
			for (var i=0; i < 10; i++){
			}
				for i in range(10):
				#タブで関数内記述

クラス定義	C++	Visual Basic .NET	ECMAScript	Python

	#include <stdio>
	#include <string>
	class Cls1 : public ClsBase
	{
	public:
	int i;
	/*コメントtype1*/
	//コメントtype2
	void NewID()
	{
	//処理を記述
	}
	//戻り値のある関数
	std::string
	NewID2(std::string
	filename)
	{
	
	std::string s;
	//文字列操作
	s = std::string("a") +
	std::string("b");
	return s;
	}
	};
		Public Class Cls1
		
		Inherits ClsBase
		
		Public i as Integer
		
		‘コメントtype1
		Sub NewID()
		
		‘処理を記述
		End Sub
		‘戻り値のある関数
		Function NewID2(ByVal
		filename as String) as
		String
		
		Dim s as String
		‘文字列操作
		s = “FHIR” + “IRIS”
		
		Return s
		End Sub
		End Class
			Class Cls1 extends ClsBase
			{
			var i;
			
			/*コメントtype1*/
			///コメントtype2
			function NewID()
			{
			//処理を記述
			}
			
			//戻り値のある関数
			function NewID2(filename)
			{
			
			var s;
			//文字列操作
			s = “FHIR”+”IRIS”
			
			return s;
			}
			}
				class cls1:
				
				#変数宣言不要
				
				#コメント
				
				def NewID():
				#処理を記述
				
				#戻り値のある関数
				def NewID2(filename):
				#処理を記述
				
				#変数宣言不要
				s = “FHIR”+”IRIS”
				
				return s

APIとライブラリ環境

FHIR APIを用いたクライアントアプリケーションの開発技術

- アプリケーションの実装方法：
  - 必要機能の決定
  - 将来的拡張・改造の可能性または必要性
  - 使用年数の推定
  - 使用者数の推定
  - 使用言語の選定（ひとつとは限らない）
  - フレームワークの選定（言語によって限定される）

Webアプリケーションの一例：

Python系

Flask
web development,
one drop at a time

django

JavaScript (TypeScript、Node.js)系

React

Express

ANGULAR

Go系

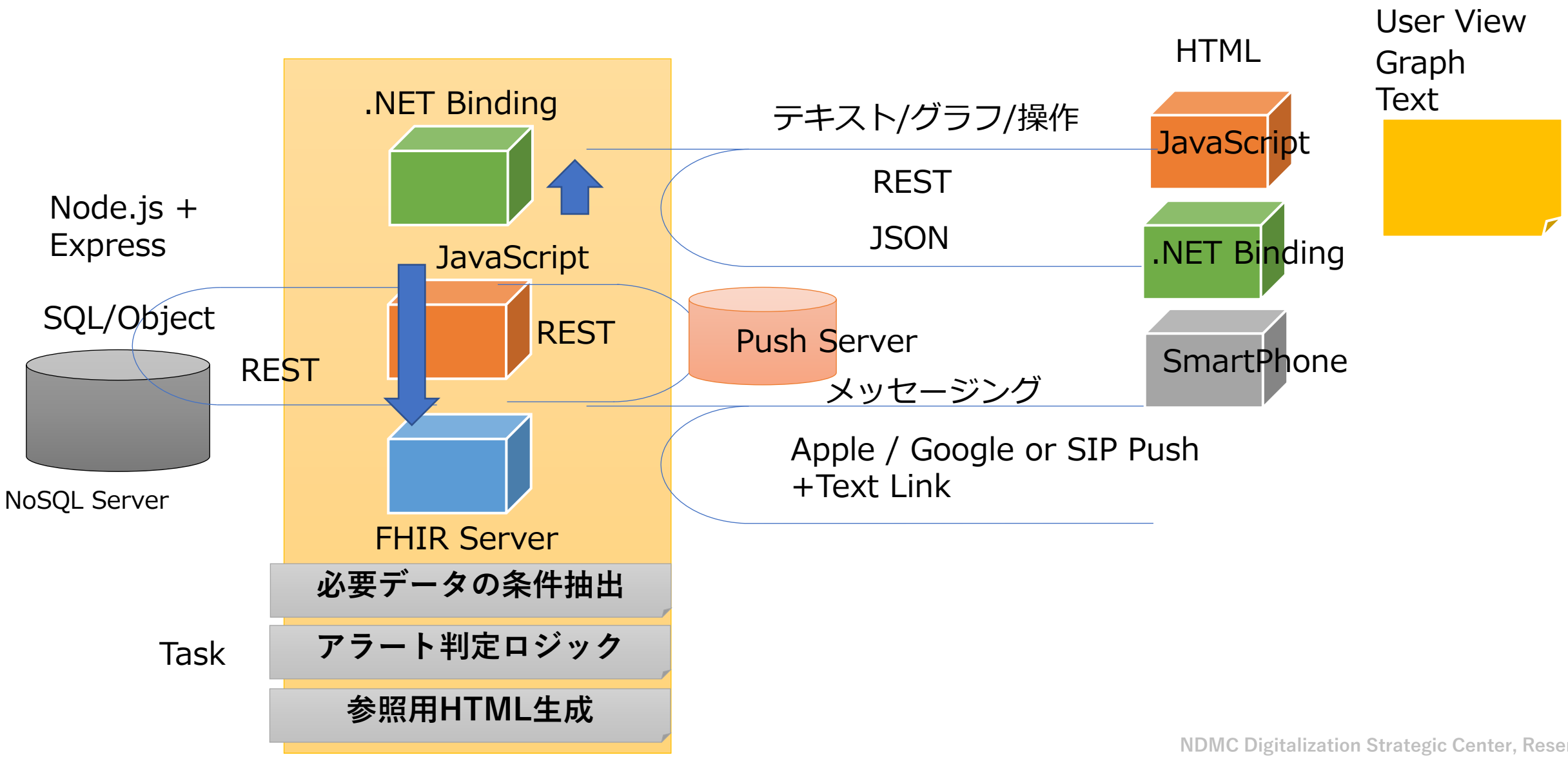
Gin Web Framework

Java系

spring®

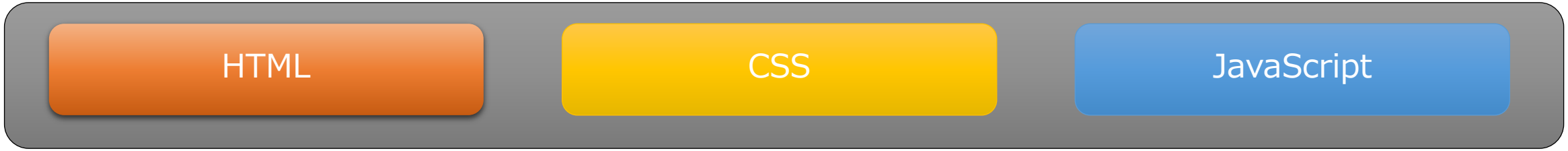
JAKARTA EE

サーバ側の中身の話（データ検索、演算、可視化、表示について）



APIとJSONの親和性

- Webブラウザ、HTML5における標準要素



JavaとXMLの組み合わせ:2000年代

クライアントの演算性能が低い時代
サーバサイドの演算性能を高めるためにコンパイラ言語が好まれた
ライセンスで利益を出すビジネス

Java実行形式　利用データ(XML)

Java実行形式内にあるパラメータを変化させたい場合

Javaコード（全体再送）、変化パラメータと不可分 → コンパイル → 利用データ(XML)

JavaScriptとJSONの組み合わせ:2010年代

クライアントの演算性能が向上した時代
サーバサービスの保守性を高めるための仕様が好まれた
プラットフォームや保守サービスで利益を出すビジネス

JavaScriptコード　利用データ(JSON)

JavaScript内にあるパラメータを変化させたい場合

HTML内のJavaScriptスクリプトの一部のみ追加 → 利用データ(JSON)

社会全体でコード保守と再利用性を高めやすくする効果

FHIRにおけるWeb/FHIRアプリケーションのサポート

- Java: HAPI FHIR

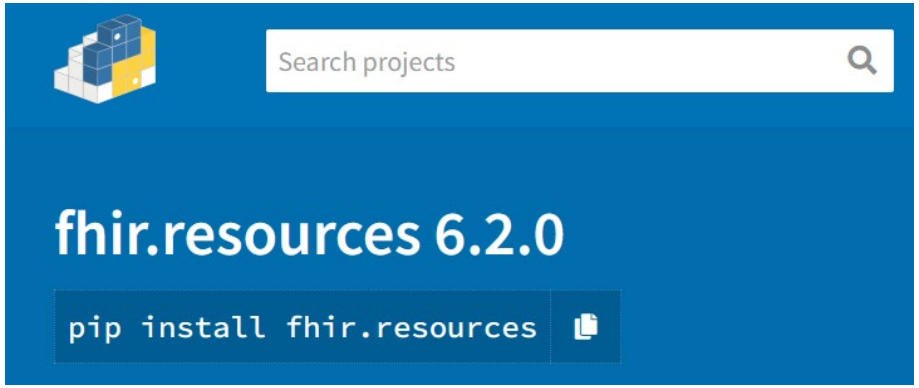
- JavaScript: SMART

SMART®

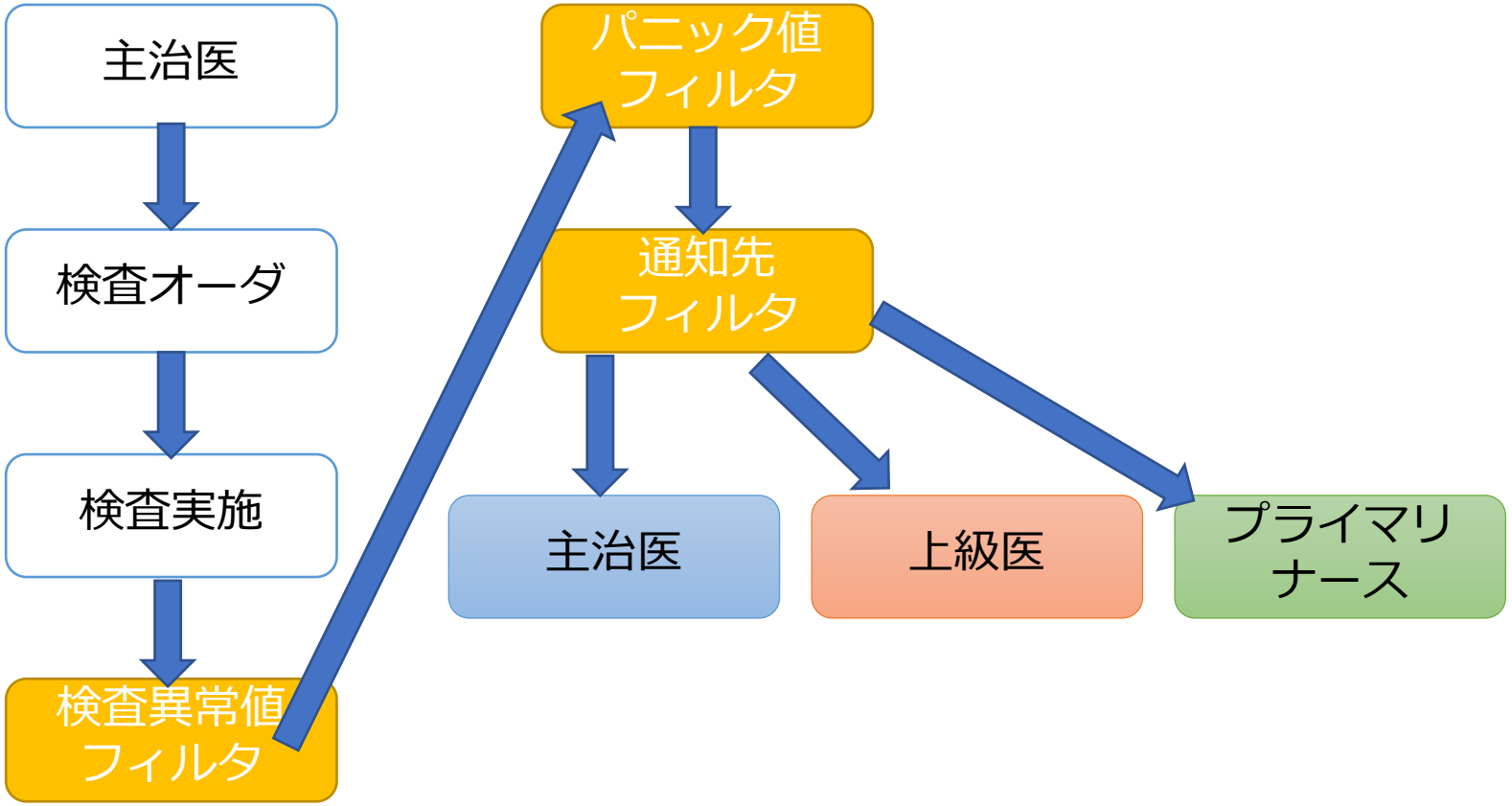
Software Libraries

- JavaScript or TypeScript: Client-side and server-side library with support for SMART App Launch
- Node.js from Vermonster: An alternative Node.js implementation
- Python: Server-side Python library with support for SMART App Launch
- R
- Ruby
- Swift (iOS)
- Java
- .NET: FHIR client library from Firely

- Python: fhir.resources



情報を規則性へ変換する＝フィルタリングすることの重要性



「必要な人に、必要な時に、必要な場所で、必要な内容を、
必要な形式で、必要なデバイスで」

FHIRはRESTに使用規則を追加する代わりに、長期運用と広大な非医療系システムの連携に道を拓く

RESTの自由度

URI、ステートレス、CRUDなど

FHIRの自由度

header追加
”context=json+fhir”

ドメイン以下の階層名指定
/Patient, /Observation …

クエリパラメータの指定
?Patient = 1

IoTの自由度

各種センサ

各種端末

各種アクチュエータ

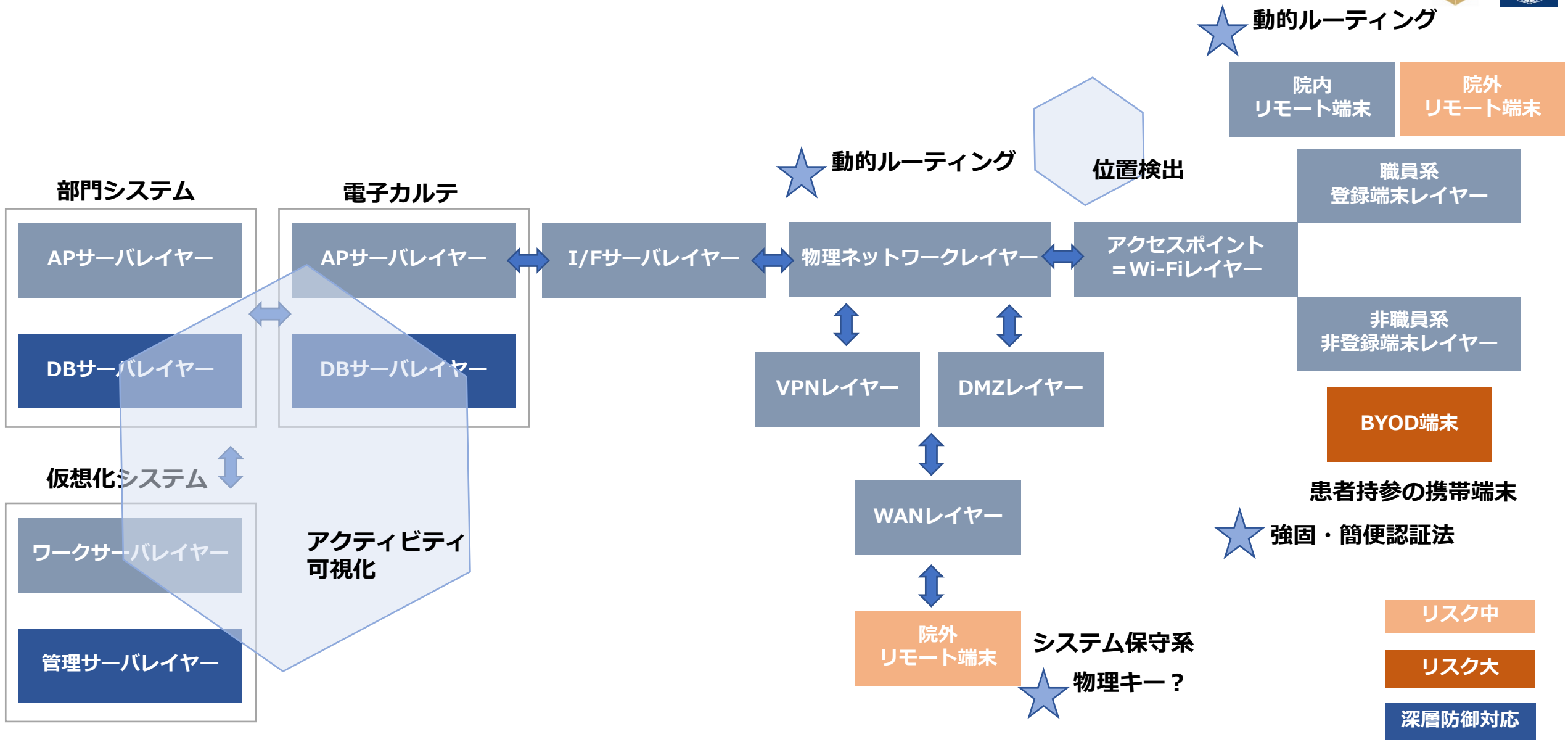
中核サーバ構築/運用者
Web型サーバとして安定した長期運用を提供しやすくなる
サーバーサーバ間通信接続を実装する工数の削減

端末側アプリケーション製作者
サーバデータの容易な参照、端末交換後のサービス持続
医療者の利用変化に対応できる機会を提供する

アプリケーション操作者（殆どの医療者）
RESTであること自体も（今後も）意識されないケースが多い
（GUIでテキストボックス、ボタン、グラフなどを操作する）

FHIRに特に期待されるのは「ラスト1マイル」の「プログラマ」への時間創出である

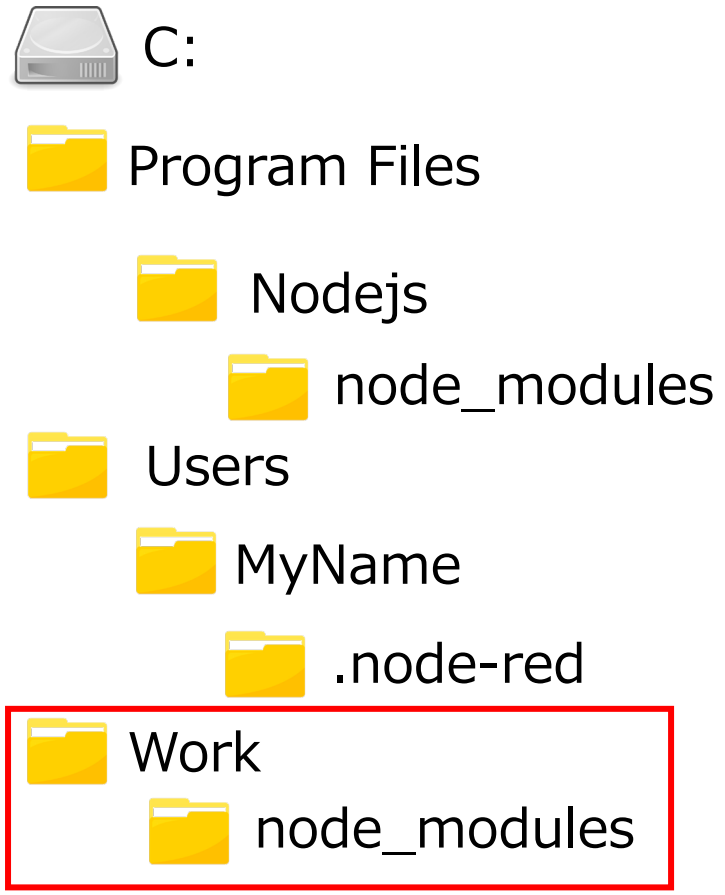
ネットワークセキュリティ向上のためにWeb型アーキテクチャを活用する



Node.jsを使ってみよう
(実演デモ)

npm: Node Package Manager

- 利用にはインターネット接続環境が必要
- コマンドラインから操作する
- ライブラリ位置は[node_modules]フォルダに集められる
- npm install […] でカレントディレクトリにインストールされる
- npm install -g […] でNode.js基幹側にインストールされる



Express：広く使われているフレームワーク

- npm install –g express-generator (Node.jsインストール後1度のみ実行)
- 環境を作りたいフォルダに移動
- express ↩ で右のファイルが生成される
- Webサーバ実行は npm start ↩

この時点でWeb(HTML)サーバがlocalhost:3000で稼働している

- bin　(アプリケーションを実行するためのwwwファイルが配置)
- public　(公開ディレクトリ images javascripts stylesheets配置)
- routes　(ルーティング)
- views　(テンプレートファイルを配置)
- app.js　(メインプログラム)
- package.json　(npmを管理)

実演デモ
app.jsを編集する

実演デモ
ルーティング(routes)を編集する

実演デモ
viewを編集する

実演デモ
同期処理を組み込む

幸福な医療情報システム開発を考える

医療情報を構成する環境の持続性に関する長年の課題

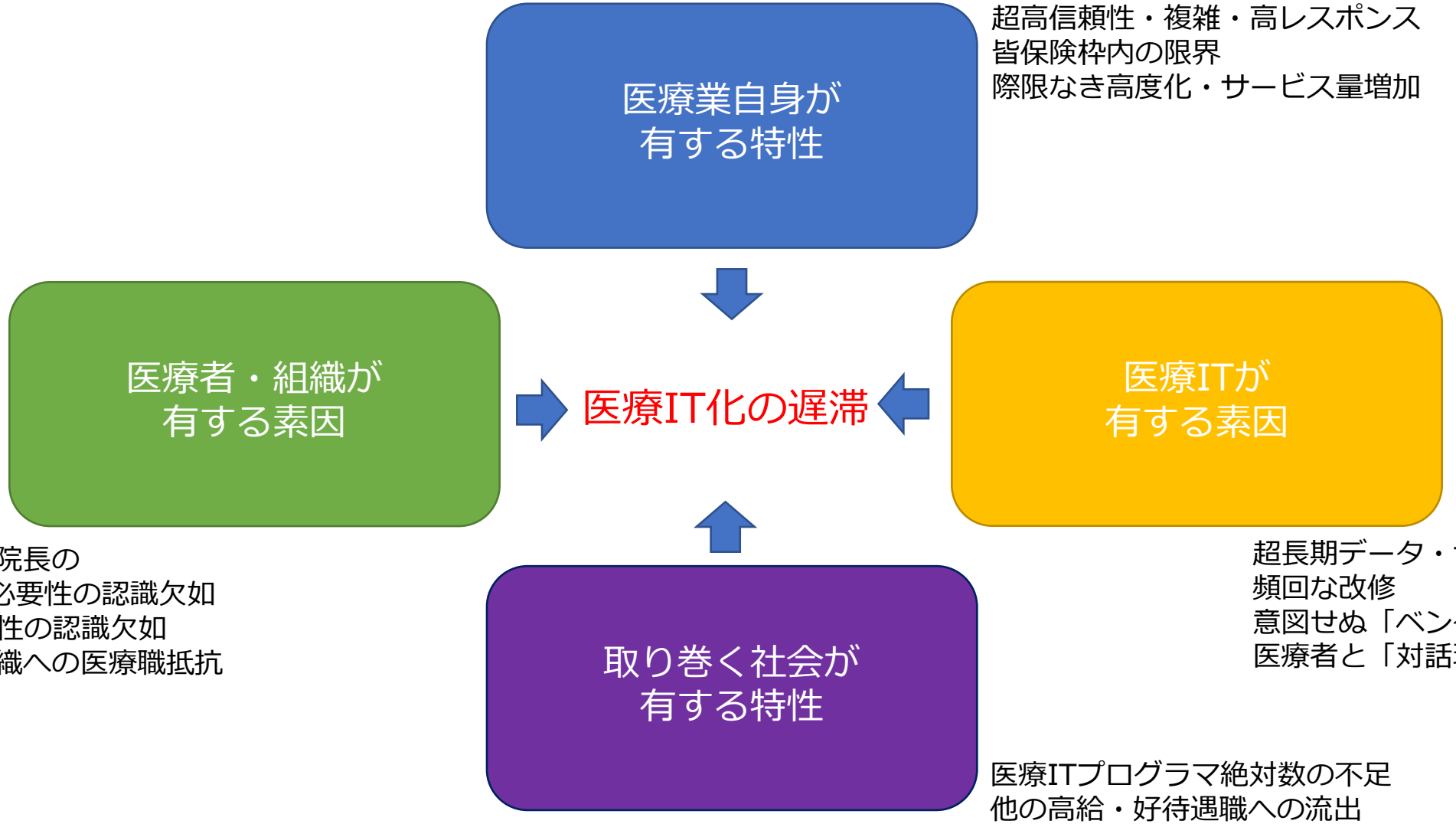
「ニーズの充足を目的とした絶え間ない高度化自身が機能積み上げの阻害要因になっている」

IT基盤の高度化と度重なる標準仕様の変更

医療基盤の高度化と度重なる標準仕様の変更

医療ITの進歩が遅滞する要因の整理

- これらの素因がさらに交絡することで、医療IT化が進まない要因に



医療情報プロジェクトの方向性「本質を捉えて省力化し、危機に備える」

今日現在の危機とは何か
・労働者人口（人間の総労働可能時間）の減少によるサービス業における質の低下と安全性の低下
・分業、記録管理の増大による本質的業務時間の減少
・業務の専門化、分業化による、コミュニケーション（用語、手段、異職種の立場の相互理解）の困難化と円滑な遂行の困難化

医療情報分野における将来的な理想とは何か
・競争ではなく、協調によって維持される医療環境への貢献
・サスティナブルなワークサイクルでの医療の量および質、安全性の担保向上
・本質的でない労働中ストレスの低い職場環境の醸成
・労働者の自然なスキル向上を伴う、教育環境の整った職場環境

電子カルテの省力化：
HRを基本とし、大学病院向けは仮想化環境での提供を続ける
アルゴリズムとViewの分離→現場の困っている点を解決する点を探して実施
「Push（必要時お知らせ）の活用」→らくらく開発において埋め込んでおく
「以前のオーダーについて再送する」操作をスマートフォンで行う
医事-カルテ-物流マスタの連動、再編成→一般化アルゴリズムの開発

「次期大学病院発スタンダード」開発の核となる方針（今後5-10年の中で着手、段階的移行）

電子カルテ・ソフトウェア

「ノンカスタマイズ」移行（医療現場の独自性が統合されなければならない。ここにどう挑むか）
「ソフトのレイヤー移行」（大病院、中病院、小病院、クリニック）×（急性期・亜急性期・回復期）
「コンシェルジュ」移行（使う人に便利を提供する、必要時Push、頻用メモリ、マスタ再構成後AI）
「大更新を延期できる」運用への移行？（長期、何年に1回の更新だと、人材スキルが維持できる？）
「仮想化とハイパーコンバージド」「クラウド」「オンプレミス」の適応領域の明確化
ソフトウェア（AI,アルゴリズム）に十分な開発投資が回るための仕掛けづくり
海外向けにはクラウド＋スマホでのソリューション提供（維持管理を）
紙との共存領域の明確化（カメラ撮影後の手書き認識など）

医療ネットワーク

「レスポンス」化によるインフラ簡素化移行（仮想化、アンテナ削減）
「光ファイバー」＋「中継器」化移行（スイッチ、APを減らし、維持管理を容易に）
「スマホ長寿命化」（5-10年使える端末を目指す）
「送受信電波強化」（アンテナ少なくても使える、そういう用途に特化したスマホがあればいいのに）
「Wi-Fiビーム応用」（屋外通信技術を応用する、DENGYOなど）
「チャンネル整理移行」（できればシングルチャネル化）

機器、端末

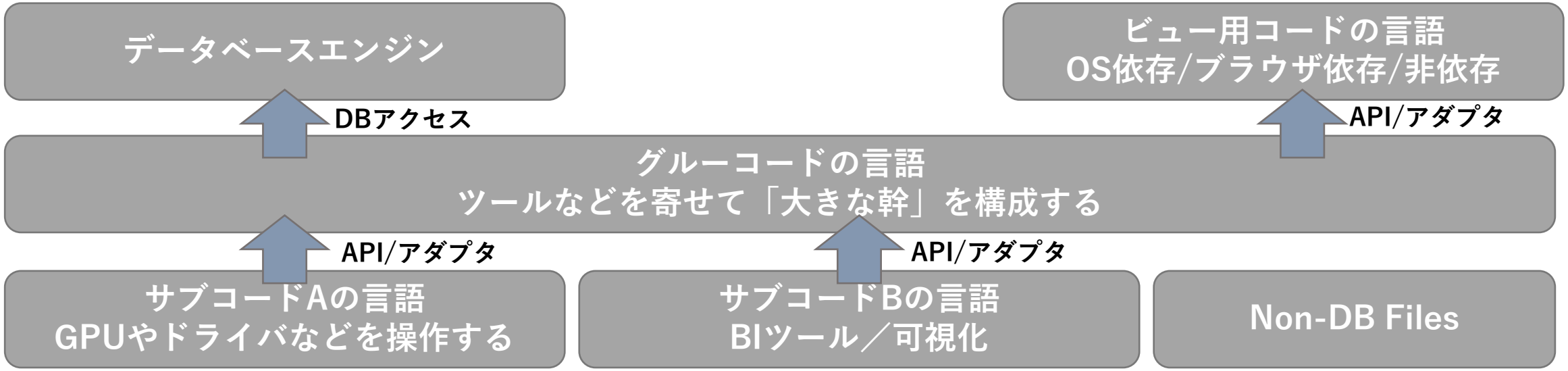
シンクラ端末「買取」での長寿命化移行（ハードウェアを長く大事に使う方針）
耐久性、耐災害性、汎用性の高い端末の選定
テクノストレスを低下させる装置選択テスト（電子ペーパー、反射型ディスプレイ、音声、ジェスチャー、VR）

コードの「機能」と特徴

- **サブコードが「継続的利用」に不可欠な機能を有する場合、ユーザーからみたソフトウェアの「価値」は「サブコードの基幹業務での必要性と長期稼働性」によって大きく左右される**

グルー言語の特徴では「アダプタ」の充実度と、
市場に出てくる新製品のキャッチアップが重要
グルー言語は「交換が極めて困難」→慎重を期した選定が不可欠

アーキテクチャ設計で考えること：
ビューを提供するOSの検討（OS依存を選ぶか、非依存を選ぶか）
市場で雇用できるエンジニアの技術力と待遇の検討
「長く使える」ビューの構成と検討
デバイスの調達のしやすさ



アーキテクチャ設計で考えること：
サブコードの言語の持続性と、市場から開発者を確保できる見通しは？
市場で開発者が少ない場合、自社に長く留まってもらう配慮ができるか？
最悪の場合、改修不能、移植不能となった場合の事業継続をどうするか？
１：ソフトの寿命と思って終売としてよいか？
２：将来的により簡便なソフトを用いて再開発するよう検討するか？

ハードウェア構成の種類と特徴

	オンプレミス	クラウド	仮想化＋ ハイパーコンバージド
現場での速度	◎	△	○
ストレージ許容 （10TBを超える）	◎	○	△
BCP能力	○	△	◎
ソフトウェアの 長期利用	△	○	◎
ネットワーク障害の 影響	◎	△	◎
多地点同時作業	△	◎	△
障害切り分け	◎	○	△
サイト被災害時の データ/システム保全	△	◎	○
保守要員の地域分散	○	◎	△
利用用途	レスポンスが 必要な業務（外来） 医療機器接続	データ送受信/保存/ プロセスの少ない用途 スマホの代替処理 地域連携	インターフェース サーバ、管理サーバ等 停止が許容されにくい 領域

ネットワーク構成の種類と特徴

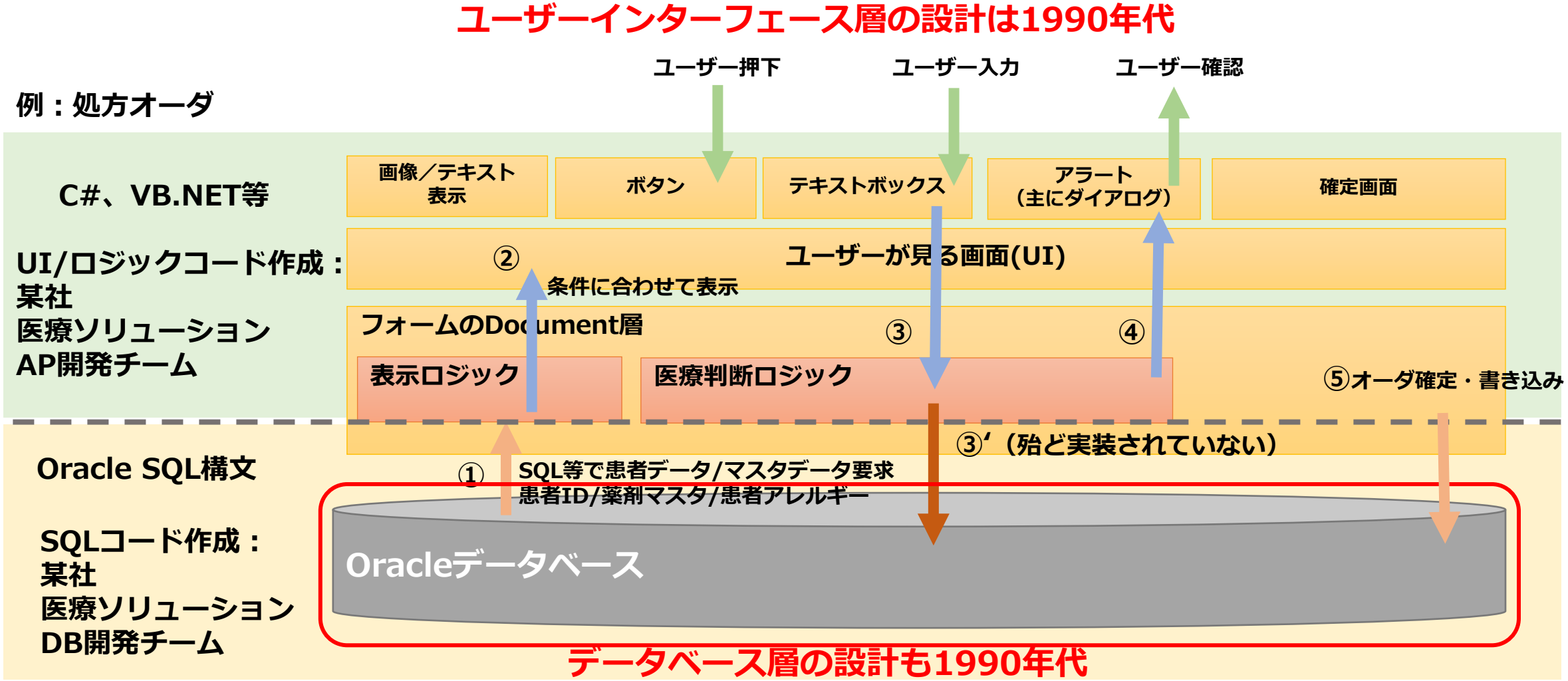
	光ファイバー	有線メタル	Wi-Fi(2.4GHz)	Wi-Fi(5GHz)
通信距離	◎	○	△	▲
通信速度	◎	○	▲	△
取り回し	△	◎	○	○
価格 （コンバータ含）	▲	◎	△	△
ノイズ干渉	◎	○	▲	△
接続しやすさ	▲	△	○	○
用途	サーバ間接続 ファットクライアント	プリンタ シンクラデスクトップ	シンクラノート スマホ通信	スマホ通信 （速度が必要な場合）

端末構成の種類と特徴

	有線ファット デスクトップ	有線シンクラ デスクトップ	無線シンクラ ノートPC	スマートフォン
プロセス速度	◎	○	△	▲
ハードウェアとの接続性	◎	○	△	▲
ネットワーク障害時の 独立動作	◎	▲	▲	▲
故障時 メンテナンス性	▲	○	◎	○
管理性	▲	○	○	○
情報の分散性	▲	◎	◎	○
価格	▲	◎	○	△
画質	◎	○	△	▲
レスポンス性	◎	○	△	△
可搬性	▲	△	○	◎
スペース/電源	▲	△	○	◎
用途	外来端末 機器接続 検査など独立動作	病棟端末	病棟端末 ラウンド端末	PDA代替 オンデマンド 新しい情報手段

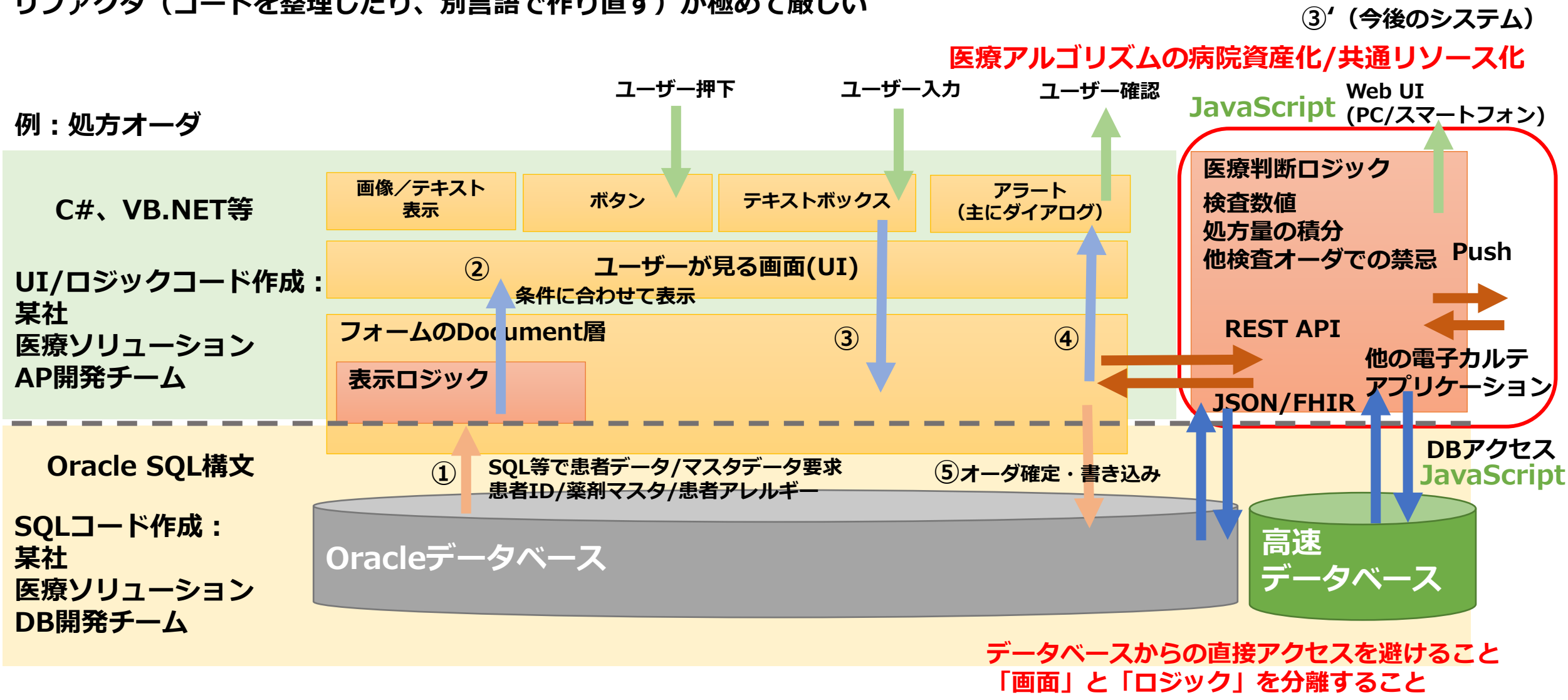
背景となる医療アプリケーションの内部構造

現在の多くの医療情報アプリケーションの構造



医療アプリケーション内部の改善提案

ユーザーインターフェースには、医療現場のノウハウが蓄積され過ぎており、
リファクタ（コードを整理したり、別言語で作り直す）が極めて厳しい



データベースからの直接アクセスを避けること
「画面」と「ロジック」を分離すること

提案する「Web系リファクタ・ロードマップ」

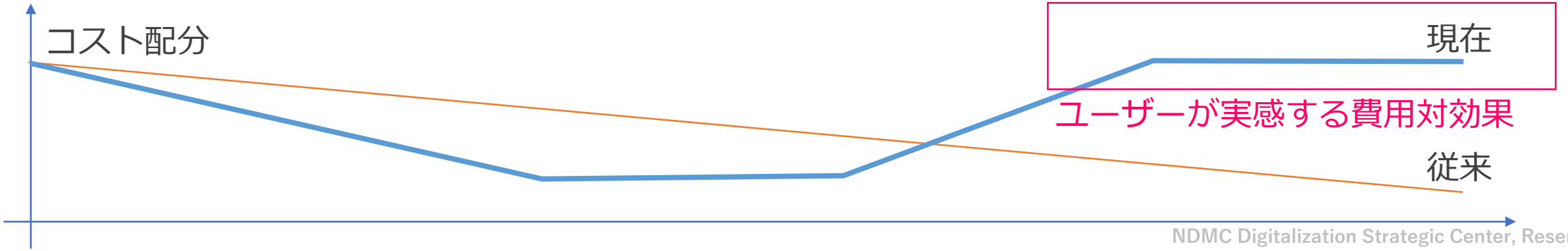
- 実運用側でのアクションと、アーキテクチャ側でのアクションを並行して進める

	対ユーザー事業／既存製品側アクション	アーキテクト戦略・長期開発側アクション
Step1	明らかに役立つ箇所：ビューで完結する インターフェースUIのWeb化	新規作成の機能について、 HTML化を優先する
Step2	OS依存のGUIから呼び出すアルゴリズムを REST APIに随時変更	長期に使うアルゴリズムの Node.js/Node-REDへの移植
Step3	DB・Commit箇所の作り替え GUIのHTML化	既存データベースのFHIRマッピング
Step4	クラウド上のWebフォーメーション運用	根本的なDBアーキテクトの再設計 マイクロサービス/NoSQL等の検討

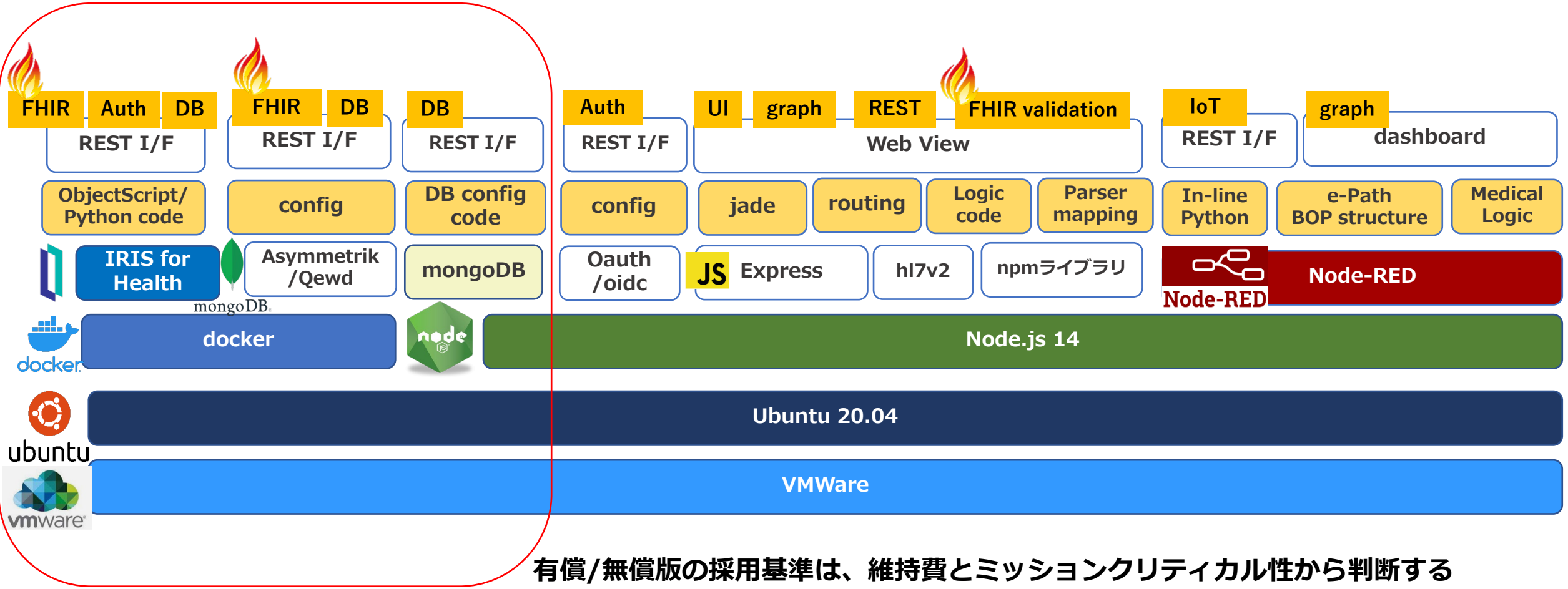
Web開発コンセプトによる医療情報ビジネスへのインパクト

“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

①	②	③	④	⑤
ハードウェア購入費	ソフトウェア購入費	保守費	実装・接続・設定費	改造費
囲い込み	不労利幅	脆弱設備	下請け化	内部工数
従来 独自規格 個別性	ライセンス負担 バージョン変化	多大な 人力負担	リスク計算が不十分で 高額化する一般管理費	継承問題 個別性
現在 共通化 仮想化	信頼性の高い オープンソース	冗長化 可視化	リファクタコスト低減 アーキテクチャ	スクリプト化 標準化



IPCI機能概要図



有償/無償版の採用基準は、維持費とミッションクリティカル性から判断する

IPCI: HL7 v2 parserにおけるNode.js→Node-REDの機能分担

Webから

GET
https://fhirtestjp.med.gunma-u.ac.jp/spreadsheettest

Node.js 14

Node-RED

render

jade

hl7v2

render

jade

POST
https://fhirtestjp.med.gunma-u.ac.jp/spreadsheettest

Input Text

```
MSH|^~¥&|LCS|LCA|LIS|TEST9999|199807311532OO||ORU^R01|3629|P|2.2
PID|2|2161348462|20809880170|1614614|鳥飼^幸太^3
世||19760924000000|M|||^^^^OOOOO-OOOO|||||||86427531^^^03|SSN# HERE
ORC|NW|8642753100012^LIS|20809880170^LCS|||||19980727000000|||HAVILAND
OBR|1|8642753100012^LIS|20809880170^LCS|008342^UPPER RESPIRATORY
CULTURE^L|||19980727175800|||||SS#634748641 CH14885 SRC:THROA
SRC:PENI|19980727000000|||||20809880170||19980730041800||BN|F
OBX|1|ST|008342^UPPER RESPIRATORY CULTURE^L||FINALREPORT|||||N|F|||19980729160500|BN
ORC|NW|8642753100012^LIS|20809880170^LCS|||||19980727000000|||HAVILAND
OBR|2|8642753100012^LIS|20809880170^LCS|997602^.^L|||19980727175800||||G|||19980727000000|||||
|20809880170||19980730041800|||F|997602|||008342
OBX|2|CE|997231^RESULT 1^L||M415|||||N|F|||19980729160500|BN
NTE|1|L|MORAXELLA (BRANHAMELLA) CATARRHALIS
NTE|2|L|HEAVY GROWTH
NTE|3|L|BETA LACTAMASE POSITIVE
OBX|3|CE|997232^RESULT 2^L||MR105|||||N|F|||19980729160500|BN
NTE|1|L|ROUTINE RESPIRATORY FLORA
```

Process v2 Message

HL7 v2 電文形式

table form **HL7 v2 表形式**

	Segment	#	title	label	#	value
1	MSH	1	FieldSeparator	data	1	|
2	MSH	1	EncodingCharacters	data	1	^~¥&
3	MSH	1	SendingApplication	data	1	LCS
4	MSH	1	SendingFacility	data	1	LCA
5	MSH	1	ReceivingApplication	data	1	LIS
6	MSH	1	ReceivingFacility	data	1	TEST9999
7	MSH	1	DateTimeOfMessage	data	1	
8	MSH	1	DateTimeOfMessage	items	1	1998-07-31T15:32:00.000Z
9	MSH	1	DateTimeOfMessage	items	2	
10	MSH	1	MessageType	data	1	
11	MSH	1	MessageType	items	1	ORU
12	MSH	1	MessageType	items	2	R01
13	MSH	1	MessageControlId	data	1	3629
14	MSH	1	ProcessingId	data	1	P
15	MSH	1	VersionId	data	1	2.2
16	PID	2	SetIdPatientId	data	1	2
17	PID	2	PatientIdExternal	data	1	
18	PID	2	PatientIdExternal	items	1	2161348462
19	PID	2	PatientIdExternal	items	2	
20	PID	2	PatientIdExternal	items	3	
21	PID	2	PatientIdExternal	items	4	

JSON形式

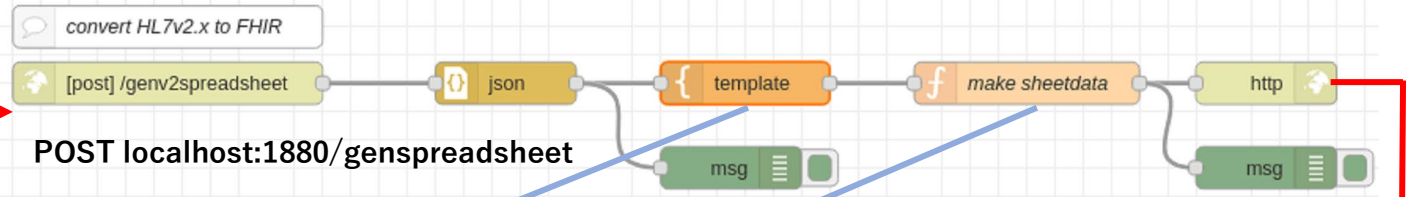
JSON form

1 result TEST9999

```
object ▶ _segments ▶
2 {3}
3 {3}
4 {3}
5 {3}
  _name : ReceivingFacility
  _def {5}
  _data [1]
    0 {4}
      _def {5}
      _level : 0
      _items : null
      _value : TEST9999
6 {3}
```

copy to clipboard

POST localhost:1880/genspreadsheet



field (column)表示形式設定

template ノードを編集

削除 中止 完了

プロパティ

名前 名前

プロパティ msg. coltemplate

テンプレート 構文: JSON

```
{
    "data": {},
    "columns": [
    {
    "type": "text",
    "title":"Segment",
    "width":90
    },
    {
    "type": "text",
    "title":"#",
    "width":90
    },
    {
    "type": "text",
    "title":"title",
    "width":180
    },
    {
    "type": "text",
    "title":"label",
    "width":90
    },
    {
    "type": "text",
    "title":"#",
    "width":50
    },
    {
    "type": "text",
    "title":"value",
    "width":280
    }
    ]
}
```

record (raw): JSONからテーブルへ

function ノードを編集

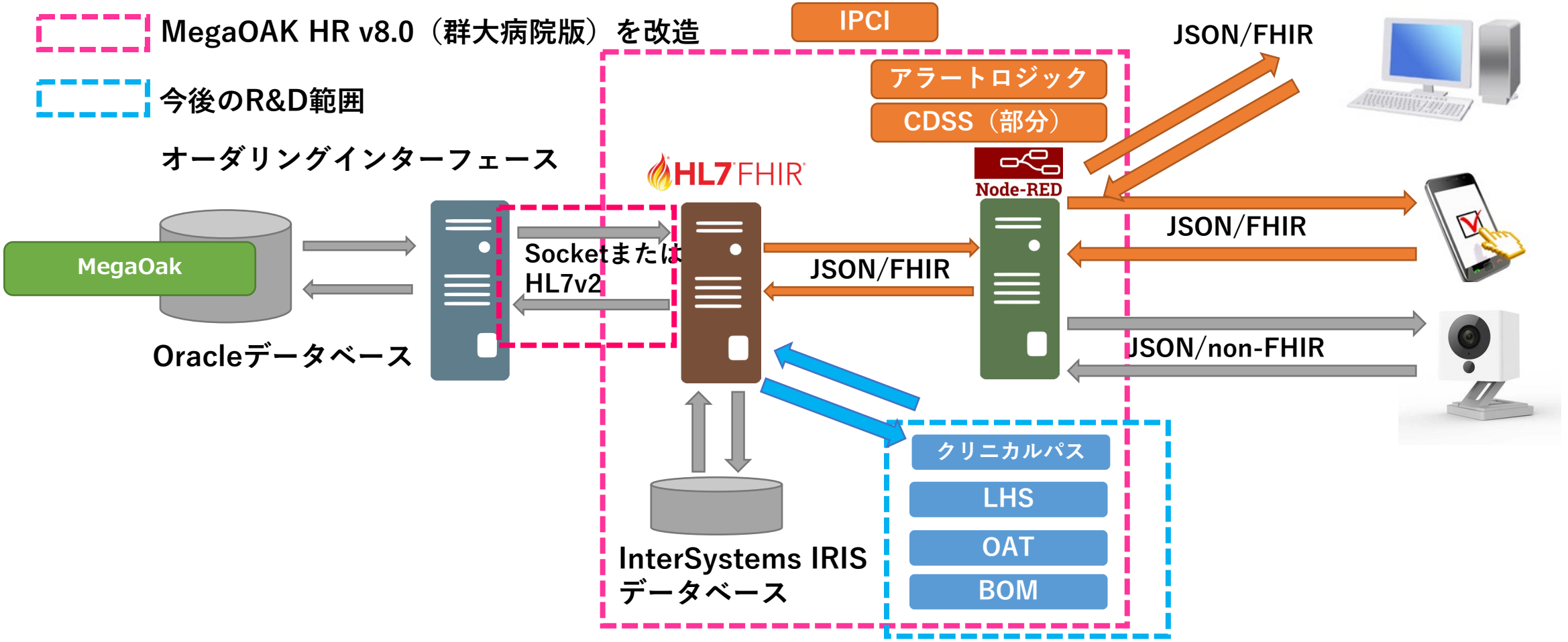
削除 中止 完了

名前 make sheetdata

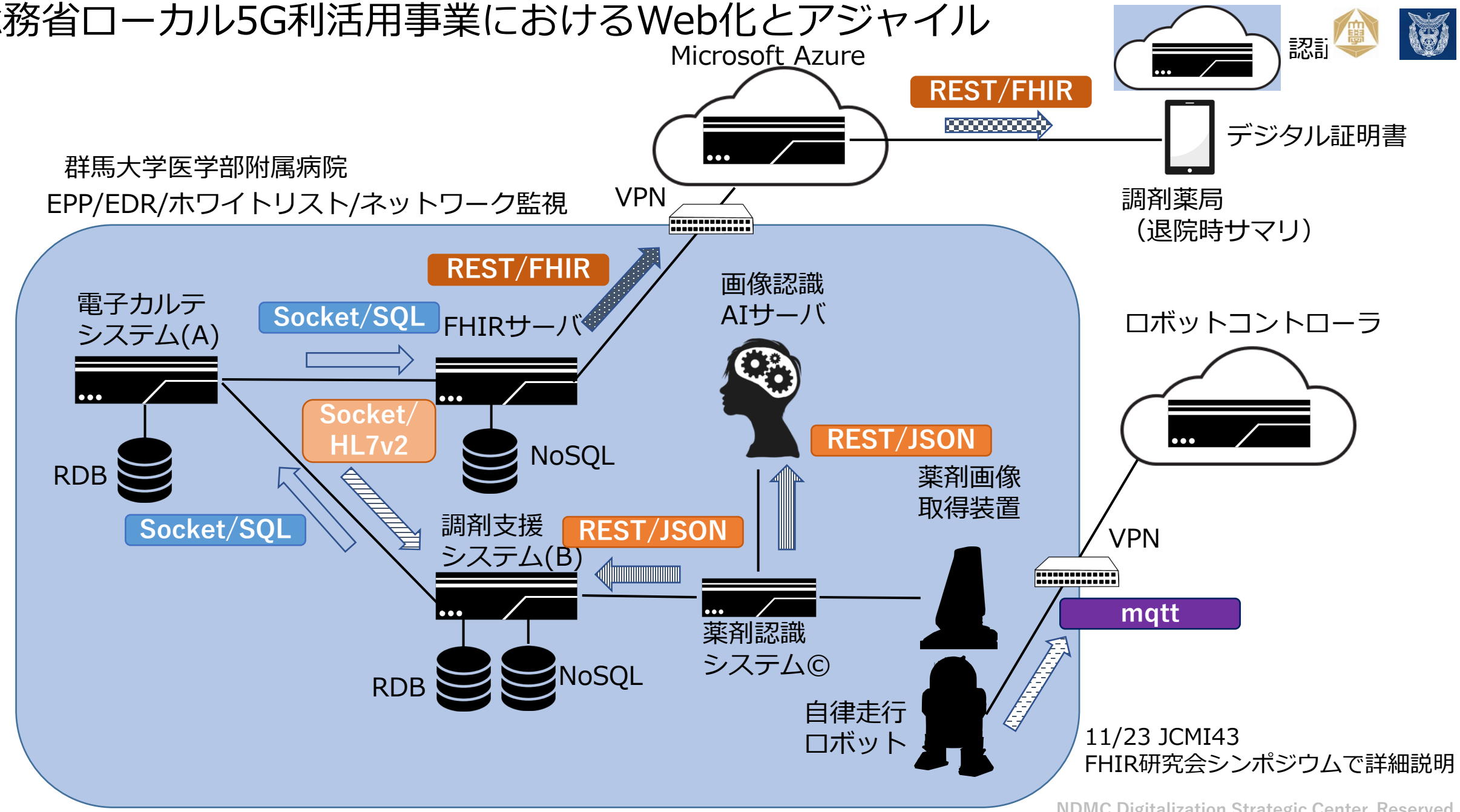
初期化処理 コード 終了処理

```
//filter PID segment
//var pid = msg.payload._segments.find(x => x._type ==='PID' );

msg.data = [];
// loop read sequence
    var record, _type, _fields_name, _fields_value, _label;

    //const execute = async() =>{

    for(var i = 0; i < msg.payload._segments.length; i++){
        _type = msg.payload._segments[i]._type;

        for(var j = 0; j < msg.payload._segments[i]._fields.length; j++){
            _fields_name = msg.payload._segments[i]._fields[j]._name;

            if( msg.payload._segments[i]._fields[j]._data !== null){
                _label = "data";
                for(var k = 0; k < msg.payload._segments[i]._fields[j]._data.length; k++){
                    _fields_value = msg.payload._segments[i]._fields[j]._data[k]._value;
                    record = [ _type, i+1, _fields_name, _label, k+1, _fields_value ];
                    //push
                    await msg.data.push(record);

                    if( msg.payload._segments[i]._fields[j]._data[k]._items !== null){
                    _label = "items";
                    for(var l = 0; l < msg.payload._segments[i]._fields[j]._data[k]._items.length; l++){
                        _fields_value = msg.payload._segments[i]._fields[j]._data[k]._items[l]._value;
                        record = [ _type, i+1, _fields_name, _label, l+1, _fields_value ];
                        //push
                        await msg.data.push(record);
                    }
                }
            }

            }

        }
    }

    //};
    //execute;

    msg.coltemplate.data = msg.data;
    msg.temp = msg.payload;
    msg.payload = {};
    msg.payload = msg.coltemplate;
```

NEC社製電子カルテ(MegaOAK HR)内IPCI機能



総務省ローカル5G利活用事業におけるWeb化とアジャイル



11/23 JCMI43
FHIR研究会シンポジウムで詳細説明

持参薬確認（持参薬確認場所で、薬剤師が操作）

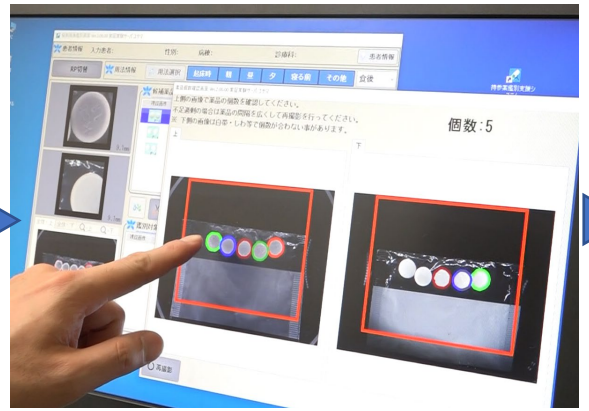
①持参薬確認場所より
持参薬確認ボタンを押す

②ロボットが持参薬確認場所へ来る

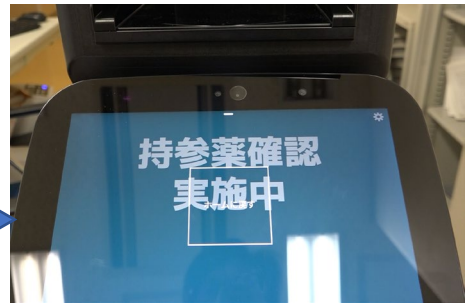
③ロボットからトレイを取出し、薬を入れる

④薬を入れたトレイをセット

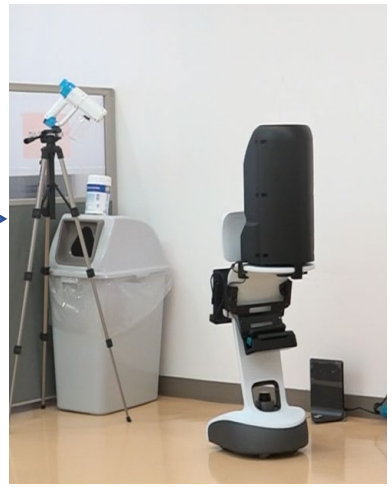
⑤撮影しローカル5Gにて画像送信



⑥候補の薬がパソコンに表示されるので、選択



⑦持参薬確認業務が終了した後、ロボット画面のホームにもどすボタンを押す



⑧ロボットがホームへ戻る

⑨ホーム到着

⑩到着後、消毒用アルコールスプレーが噴射される

配薬確認（配薬確認場所で、看護師が操作）

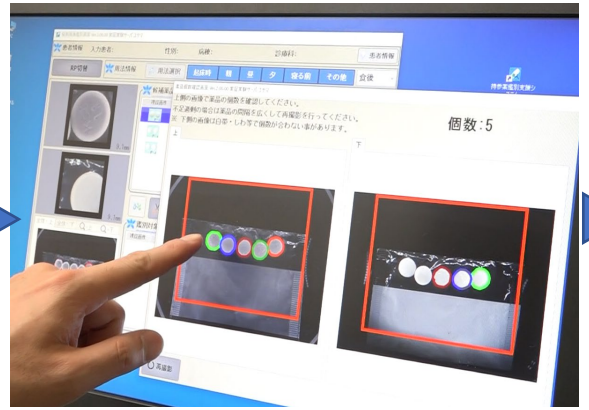
①配薬確認場所より
配薬確認ボタンを押す

②ロボットが配薬確認場所へ来る

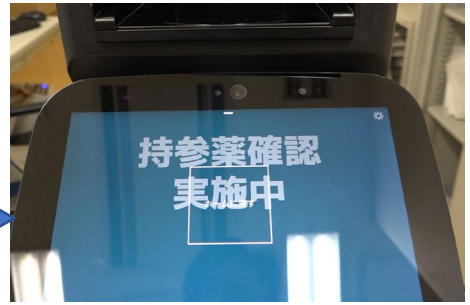
③ロボットからトレイを取出し、薬を入れる

④薬を入れたトレイをセット

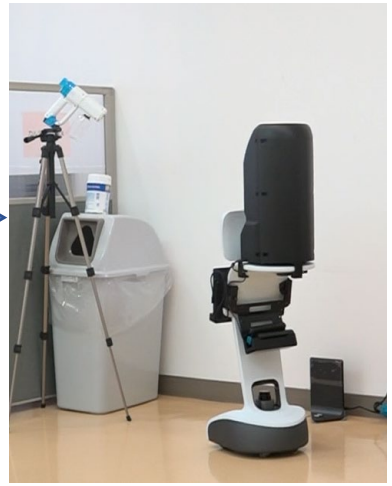
⑤撮影しローカル5Gにて画像送信



⑥候補の薬がパソコンに表示されるので、選択



⑦配薬確認業務が終了した後、ロボット画面のホームにもどすボタンを押す



⑧ロボットがホームへ戻る

⑨ホーム到着

⑩到着後、消毒用アルコールスプレーが噴射される

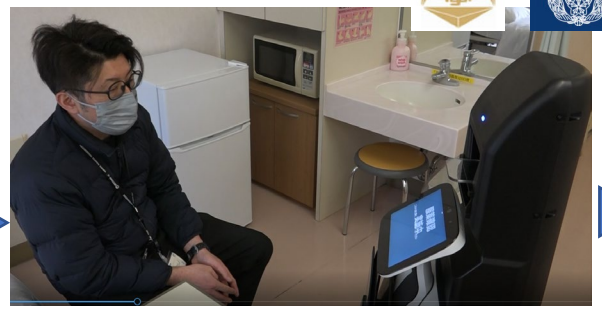
服薬前確認（病室で、患者が操作）

①看護師が服薬前確認をしたい患者ID
ボタンを押す

②ロボットが対象の患者病室へ向かう
♪服薬前確認のため●●●へ移動します♪

③ロボットが対象の患者病室へ入る。
（扉が開いている必要あり）

④ロボットがベッド横に向かう
※病室ごとに止まる地点が
決めてある）



⑤ロボットが事前に決められた位置で止まり
♪これから飲む薬を
トレイにセットしてください♪

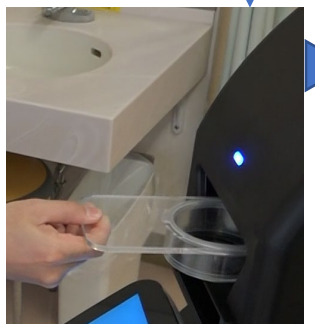
⑥ロボットからトレイを取り薬を入れる

⑦ロボットからトレイを取り薬を入れる

⑧ナースステーションで、看護師がシステム突合結果を確認する。

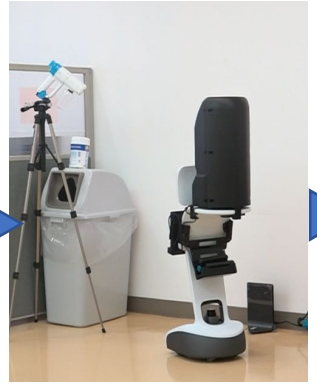
⑨薬を取り出す

⑩薬を飲む



⑪空のトレイを戻す

⑫ロボットが病室を出てホームへ戻る



⑬ロボットがホームへ戻る

⑩到着後、消毒用アルコールスプレー噴射される

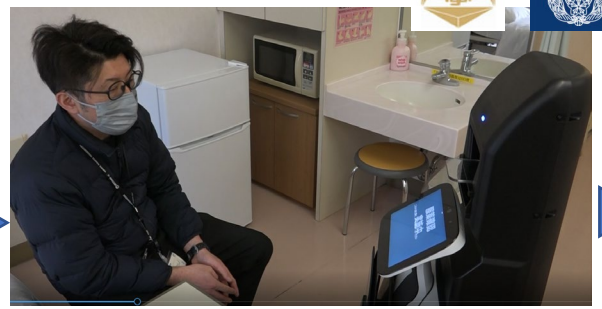
服薬後確認※飲み殻確認（病室で、患者が操作）

①看護師が服薬後確認をしたい患者ID
ボタンを押す

②ロボットが対象の患者病室へ向かう
♪服薬済み確認のため
●●●へ移動します♪

③ロボットが対象の患者病室へ入る。
（扉が開いている必要あり）

④ロボットがベッド横に向かう
※病室ごとに止まる地点が
決めてある）

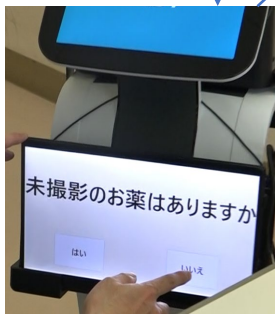


⑤ロボットが事前に決められた位置で止まり
♪飲み殻をトレイにセットしてください♪

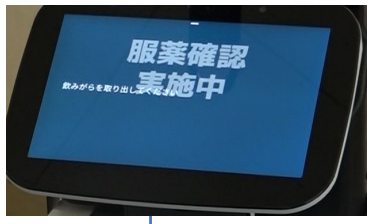
⑥ロボットからトレイを取り飲み殻を入れる

⑦トレイを入れると撮影

⑧解析サーバにて
飲み殻確認



⑨次の殻セットまたは、
「いいえ」押し終了



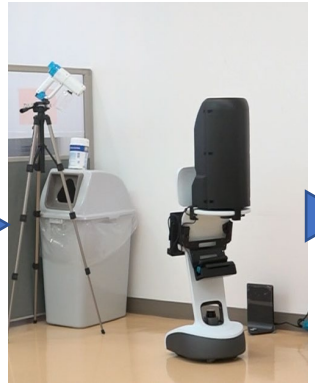
⑩♪飲み殻を取り出して
ください♪

⑪飲み殻を取り出す

⑫空のトレイを戻す

⑬空のトレイを戻す

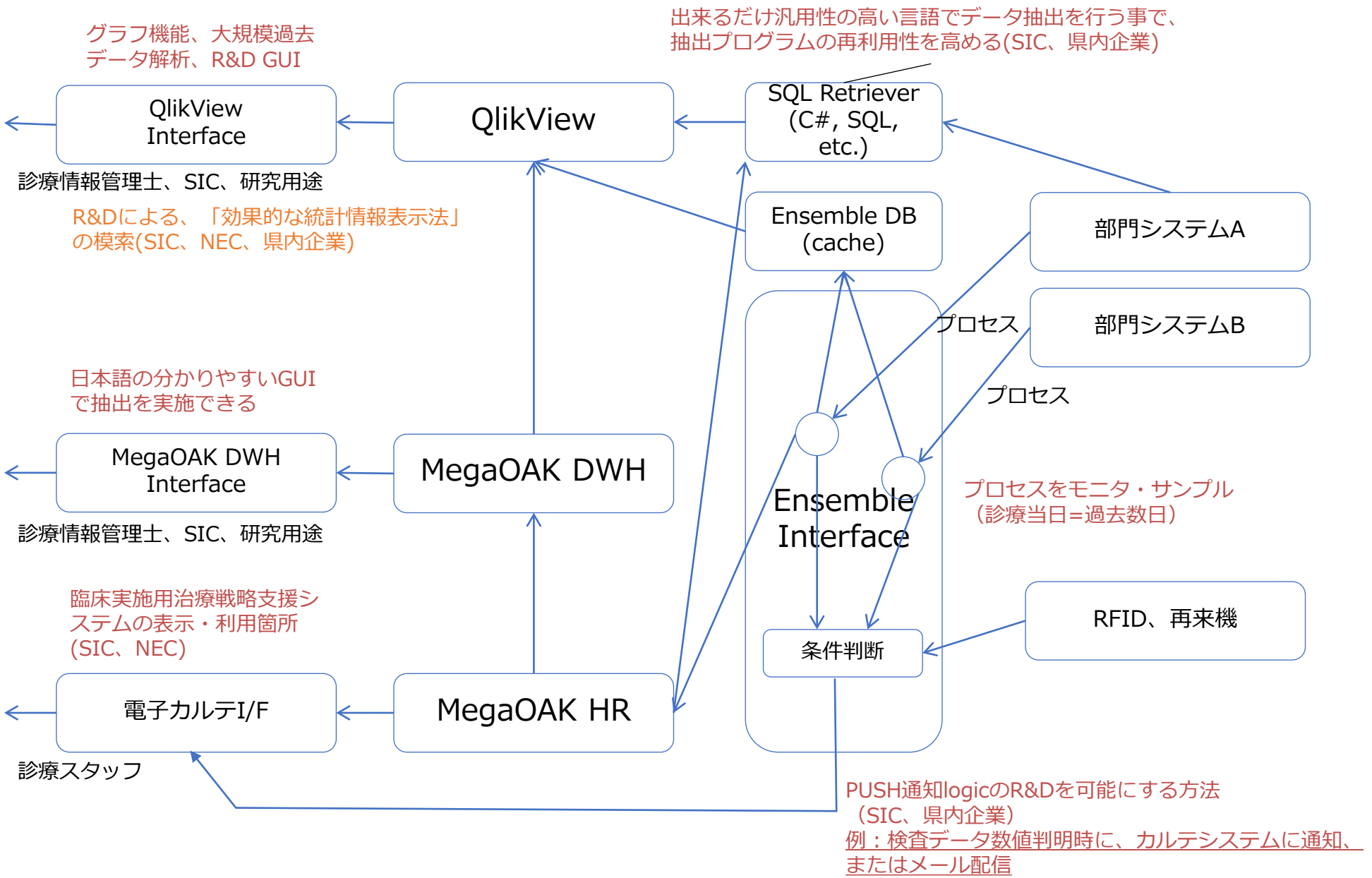
⑭ロボットが病室を
出てホームへ戻る



⑮ロボットがホームへ
戻る

⑯到着後、消毒用アルコール
スプレー噴射される

データベース観点からみたシステム接続概要



IPCIコンセプトのversatile（多態)性

効力時間　法則化

連携病院

ePath Project Thinking of medical future

FHIR®©

「面での医療の質の向上」

IPCI-2
クリニカル
パス共有

病病連携

国内／国際
臨床試験

FHIR®©

ePath Project Thinking of medical future

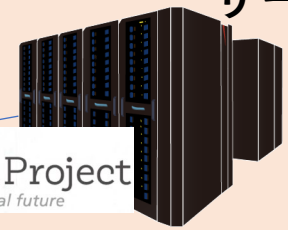
IPCI-3
地域拠点
リソース

FHIR®©

ePath Project Thinking of medical future

中核拠点病院
大学病院

電子カルテ
サーバ



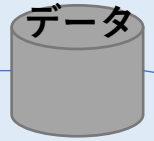
「研究を加速するデータ提供」

病診連携
（地域連携）

病診連携
（地域連携）

カルテツール

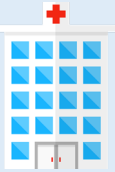
レセプト
データ



IPCI-1
タスクシフト

IPCI-4
ナショナルセンター/
インターナショナルセンター

病薬連携



FHIR®©

「電子処方箋、疑義照会、病歴照会の効率化」

In-Process Clinical Intelligence:
規模が違っても互換のある仕組み

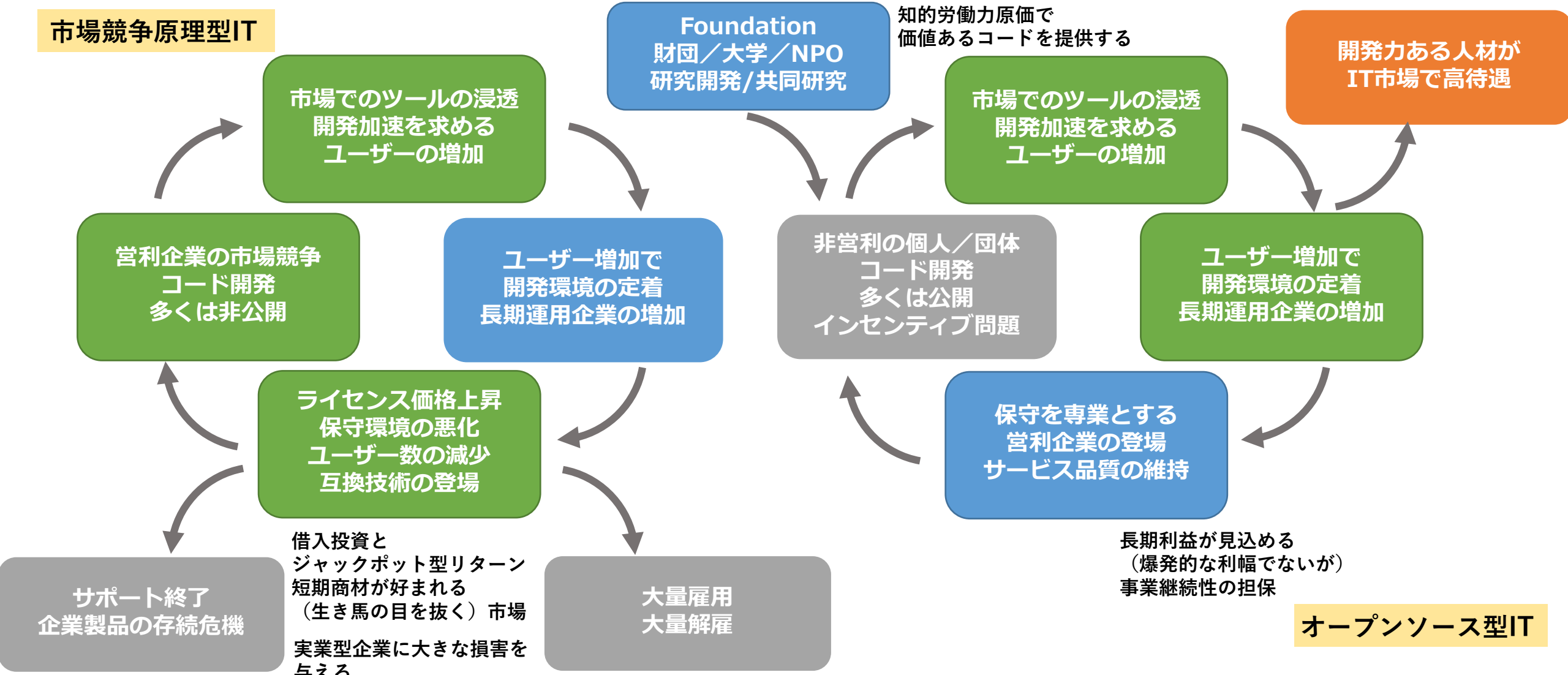
Phase-A　規模が小さく
費用対効果が高い

Phase-B　地域包括ケアレベルの
医療連携

Phase-C　大規模臨床試験
PHR（個人記録）
保管先

ITにおける有償市場の必要性と、オープンソースサイクルの必要性

- **過剰にインフレーションしやすい市場的性質**
- **医療ITでは信頼や継続性という「ITビジネス」の性質に衝突する特徴がある**



創造的破壊とは：さらなる時間創出のために既存の構造を変化させること

- シュンペーターによって紹介された
- 私たちがしばしば問題とするのは、「何を創造すべき」と定めるか
- 「創造すべき」は「未来」であり、「見通し」
  その前段として、「それらを考えるための時間」が必要
- 創造的破壊は、行うにあたり、
  「適用する現場の、どのような時間を創造」するかを
  予め考える必要がある

https://en.wikipedia.org/wiki/Joseph_schumpeter

ご清聴ありがとうございました

HL7セミナー
Node.jsでWebサーバを作る

群馬大学医学部附属病院システム統合センター
防衛医科大学校デジタル化推進本部推進補佐官
鳥飼　幸太