JINS MEME

powered by amazon web services

バイタルデータの意味付けという荒波を乗り越える！適切な処理分担のためのサーバーレスアーキテクチャー

(株)ジンズ JINS MEME Gr. 菰田 泰生

Outline

バイタルデータを測定するウェアラブル端末とサーバーレスアーキテクチャの協調処理に関する事例紹介です

- 製品紹介＆自己紹介
- バイタルデータって
- サーバー側アーキテクチャへの落とし込み
- 開発プロセス
- まとめ

現行製品・アプリ紹介

JINS MEMEのさまざまなアプリケーションが可能にするのは、今の「自分」を客観的に見ること。より良い自分を目ざすためのアプリケーションです。

JINS MEME

JINS MEME OFFICE

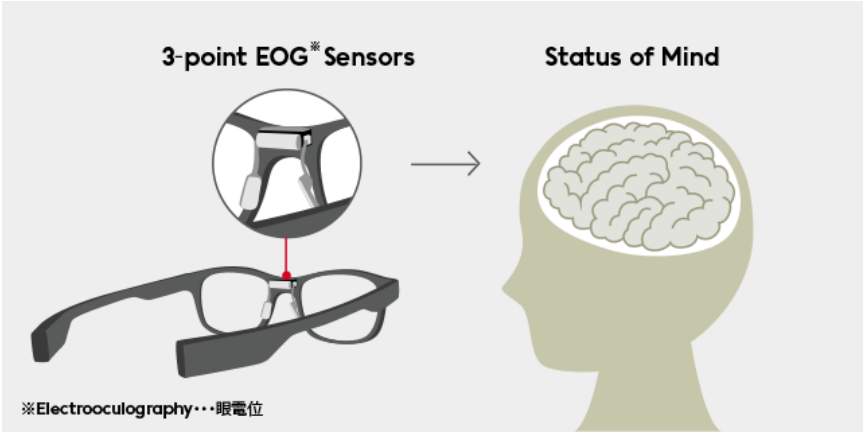
JINS MEME ZEN

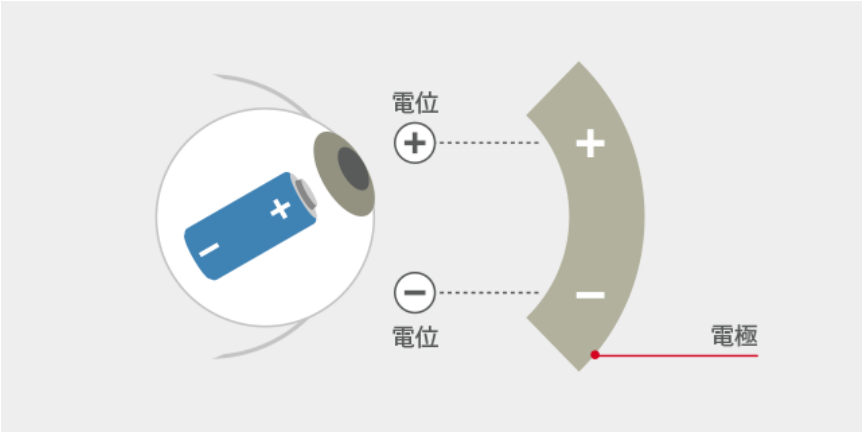
JINS MEME RUN

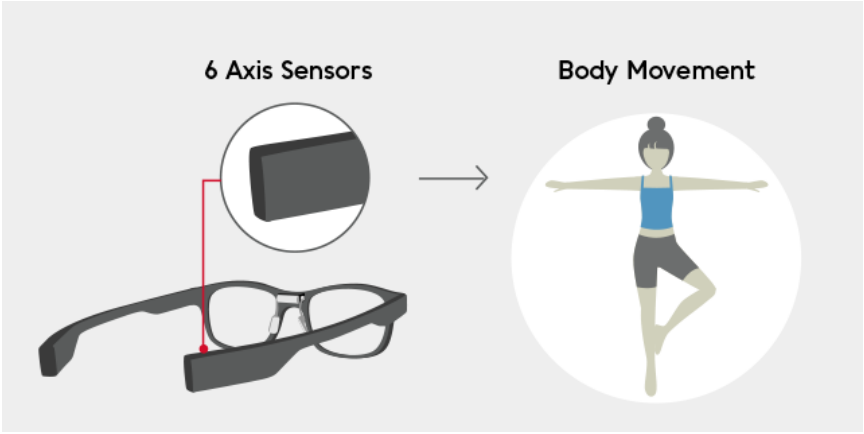
JINS MEME TAIKAN

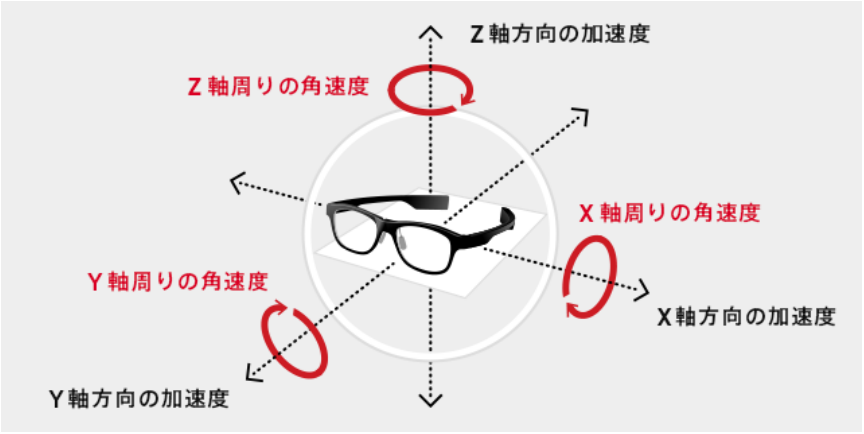
JINS MEME DRIVE

搭載センサー…眼電位センサー＆モーションセンサー









SDK(iOS/Android) から利用できるデータ (layer1)

- 頭部の加速度（x,y,z）
- 頭部の回転角（roll,pitch,yaw）
- まばたき　強さ(uV) / 閉眼時間(mSec)
- 視線移動　上下左右を3段階　※絶対位置ではなく、相対的な移動を検出します
- 周波数　20Hz

R&D中

JINS MEME

SDKのデータを意味づけすることで、静止時・動作時の質を評価できる指標群が利用できます

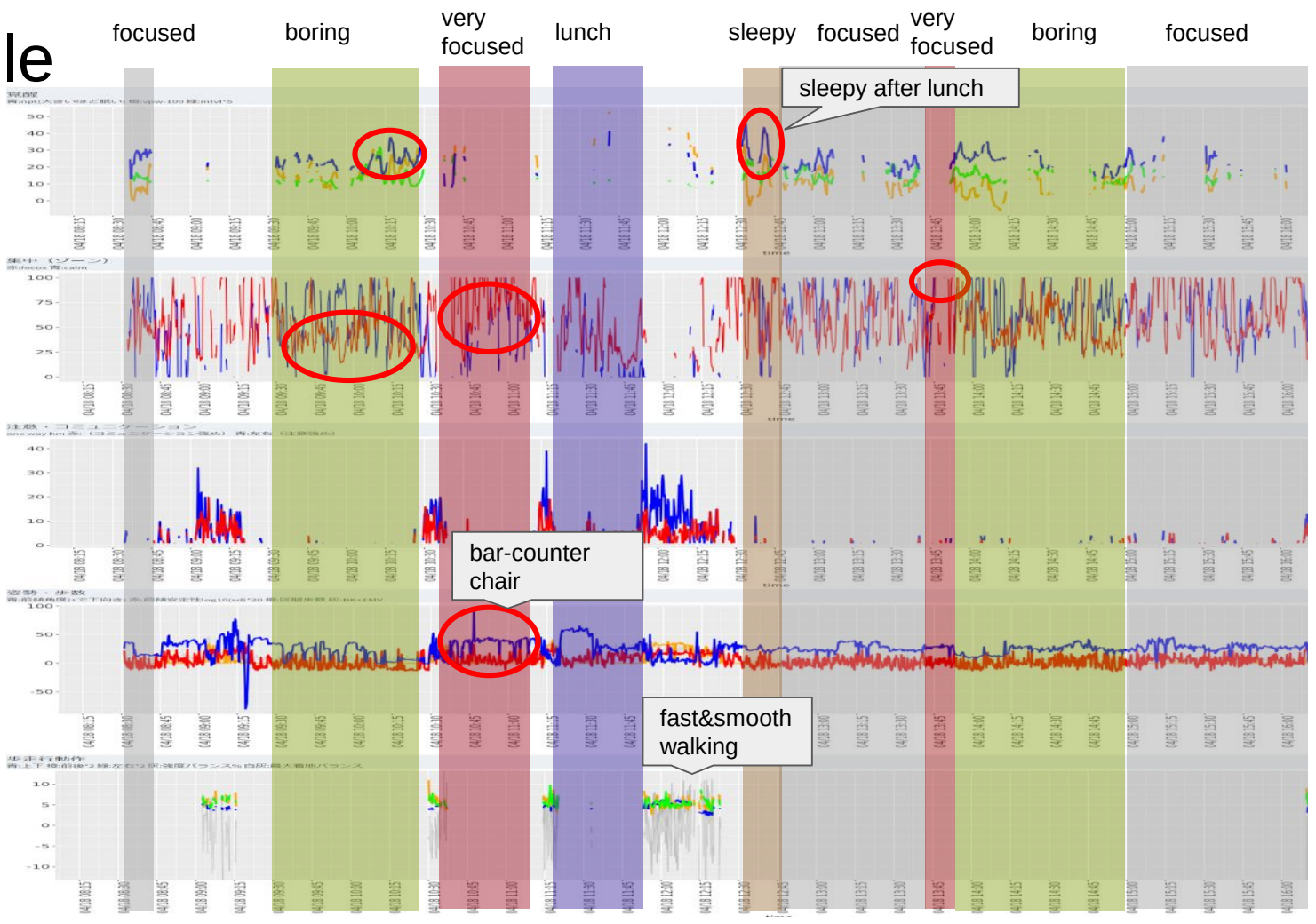
R&D中

Available data type (layer2)

			Still 着座・乗車時	Active 歩行・走行時
Unconscious 無意識		Awakeness 覚醒	○	
Conscious 意識	Closed クローズ	Focus 没入	○	
		Calm 落ちつき	○	
	Open オープン	Attendion / Communication 注意・コミュニケーション	○	○
Motional 動作	Posture 姿勢	Tilt 前傾角度	○	○
		Stability 姿勢安定性	○	○
	Gait 歩走行	Oscillation 振動幅(X/Y/Z)		○
		Cadence ケイデンス(ピッチ)		○
		Balance 左右バランス		○

R&D中

Data sample



Awakeness
覚醒

Focus
没入

Calm
落着き

Attention / Communication
注意・コミュニケーション

Tilt
前傾角度

Stability
姿勢安定性

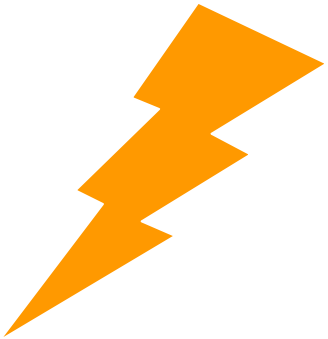
Cadence
ケイデンス(ピッチ)

Oscillation
振動幅(X/Y/Z)

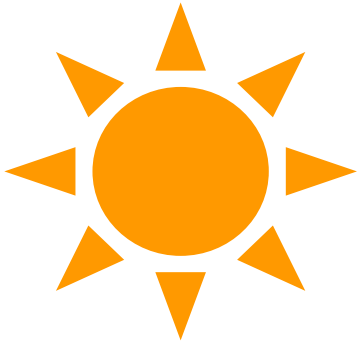
Balance
左右バランス

目指すもの

パフォーマンスのアップ



健康寿命のアップ



今後もハード・アプリの改良を
継続的に進める予定です

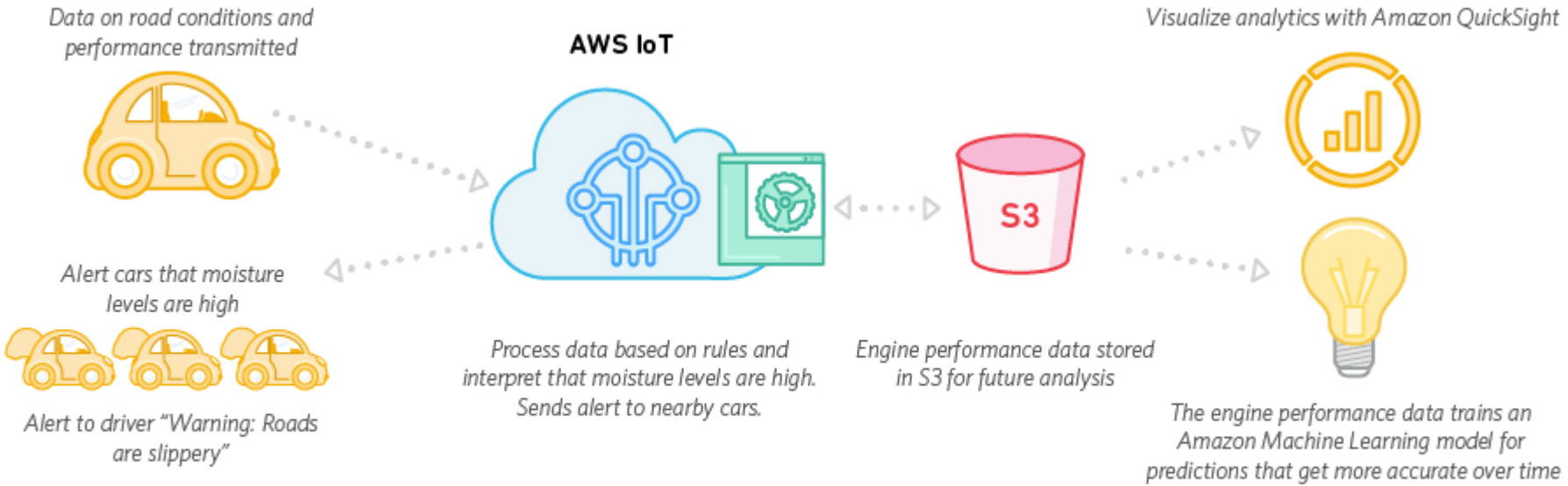
自己紹介

@komde (Twitter/Qiita) こもだ たいき
専攻は応用物理系
→東芝: 半導体デバイス設計にて多変量最適化（Cellの物理層）
→IBM: プロセス改革支援、ISO26262（機能安全）のソリューション開発
→JINS: 物流→情シス→経営企画→SCM&EC(＆MEME支援)→MEME次世代

好きな言語: R
好きなフレームワーク: Shiny
好きなAWSのサービス: Lambda

バイタル（生体）データって

IoTのイメージ



生体データを起点にするサービスの事例をあまり聞かない、、

なぜバイタルデータの中で心拍センサーは良く使われているのでしょうか？

1 意味云々よりデバイス・アルゴリズムとして枯れていてよく使われているから
2 心拍はフィジカルトレーニングに加えLF/HFから自律神経バランスが見える
3 周波数解析が通用する

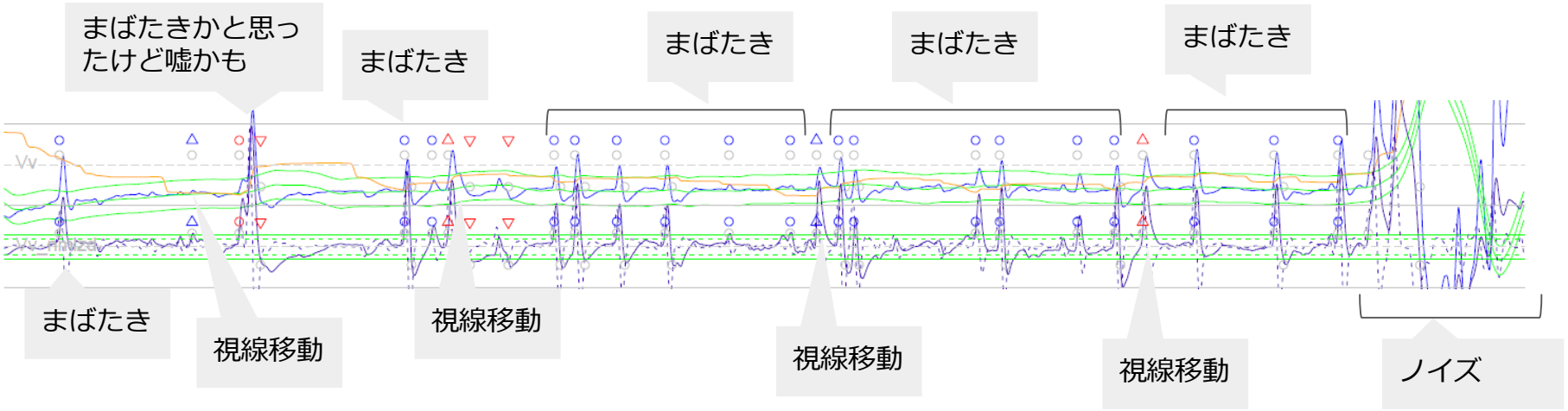
どれも正解ですが一番近いのは、、

意味云々よりデバイス・アルゴリズムとして枯れていてよく使われているから

バイタルデータは「面倒くさい」

データタイプ	意味の重要性	周期性	ウソ (ノイズ・空白)	教師データ	難易度	手法
モーション (加速度、角速度)	少	無し	無し	多	易	パターン認識(機械学習) 周波数解析 ルール
心拍、脈波、心電図	中間	有り	有り	多	中間	周波数解析→パターン認識 周波数解析→ルール
眼電位、脳波	多	無し	有り	少	難	多段ルール→統計/ML ※要は力業 （人間が中間層を指定できるDNN的な手法はアリだとは考えている）

眼電位の生データ



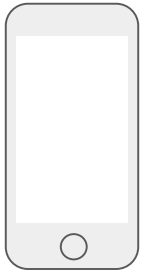
- どういうまばたき・視線移動をしているかは目で見て分かる
- 周期性が無くノイズ（ウソ）が多い
- 数秒のウィンドウから意味づけを得られることは難しく、ダイレクトなパターン認識は通用しない
- データ量が多い！

バイタルデータならではのレイヤーの深さと通信の制約

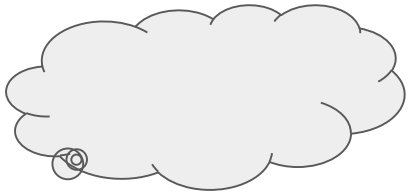
	データ量 bps	説明
生データ	25,000 bps ちょっとした音楽ストリーム	センサーが吐き出した、ノイズなどもそのまま含むデータ、周期データ
半生データ(まばたき等)	3200 bps BLE@iOSでストリームする限界	特徴点(眼電位でいうとピーク)の諸元を抜き出したデータ、イベントデータ
中間データ(区間区切り)	150 bps 間欠通信で許されるレベル	指標値に変換できるがそれ自身は意味の無いデータ、周期データ
指標値(覚醒・没入・姿勢)	50bps	意味のある（使える）データ

絶対的な制約

BLE



3G/4G/WiFi



アルゴリズム配置の制約

プロセッサ	FLOPS	消費電力	
Cortex-M0 / MSP	FP/DSP が無い！DIVIDEできない！	0.02m~1mW	軽めのMoving-X
Cortex-M4~	20～100 MFlops	0.5m~100mW	軽いML, FFT系
Cortex-A	5～100 GFlops	0.1~2W	ML, 軽いxNN
Xeon E	0.2～1 TFlops	50~200W	xNN
Tesla	1～10 TFlops	100~400W	重いxNN

←ラズパイでもココ

- 現行品でバッテリーは500mWh
- Moving-X(MA:移動平均など) 、分布(正規分布、ガンマ分布)をよく使用するが、意味づけまでエッジで行おうとすると高負荷になりすぎる
- sin(atan(x,y)) 程度の式でも重く、代理式（2次関数）に置き換えが必要

バイタルデータを使ったウェアラブル端末の三重苦

- レイヤー毎の多段処理がまだまだ必要
- エッジでデータのエントロピーを上げておかないとBLEの通信帯域の上限に近づき使い勝手が悪くなる
- エッジのプロセッサに意味づけまで行う処理能力は期待できない

→上記の制約から大体のエッジとクラウドの役割分担は決まる

→アルゴリズム上、様々な中間データをJOINしたくなるが、最終的にはレイヤー間が疎結合になるように心の底で留意しておく

JINS MEME（将来版）の処理フロー

メガネ（エッジ）

短周期（数十秒～数分）の特徴抽出と後段で利用可能なsummerizeまでを非力なプロセッサでできる範囲で頑張る

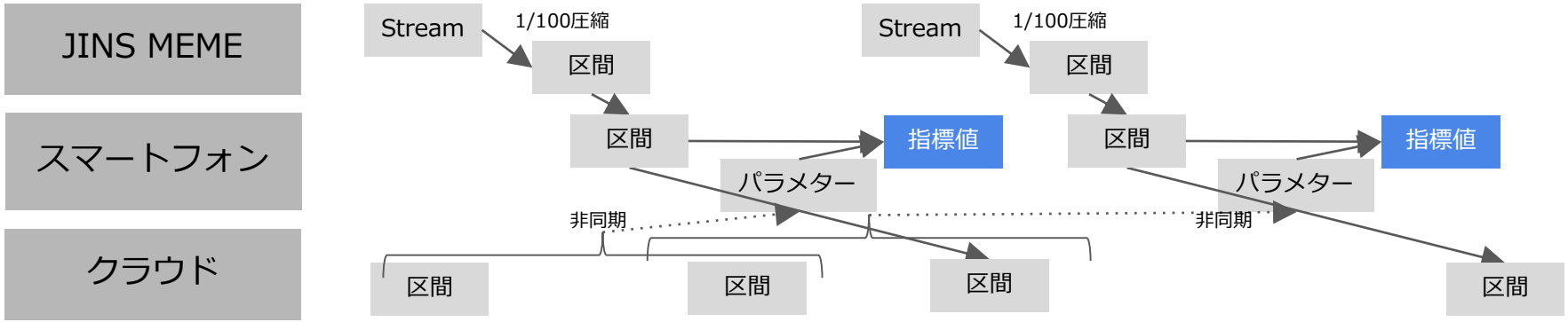
スマホ

短周期の特徴データとクラウドの長周期パラメターを利用して意味づけデータ(layer2)を生成しサービスを実行する

ネットワークの寸断がサービス全体の単一障害点にならないように「非同期」でクラウド連携

クラウド

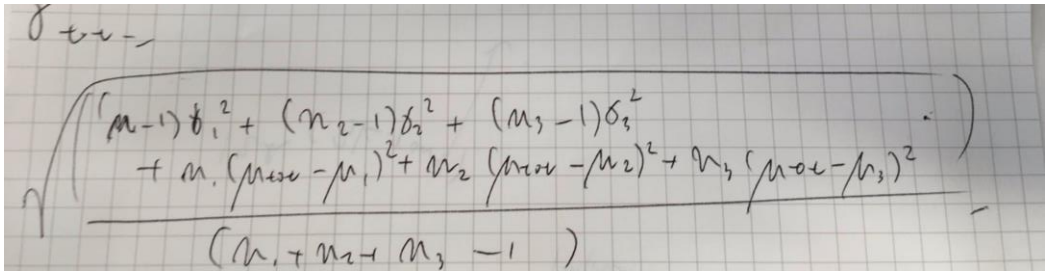
リアルタイム型のサービスのための長周期（数時間～数週間）のパラメター算出と、バッチ型の分析サービスを担当



リアルタイム＆ステートレス化のための差分更新アルゴリズム

スケーラビリティを保つため、特定区間を抜き出しサマるというバッチ的ロジックは使用せず、逐次処理化

- 長周期の平均値は移動加重平均を使用、合計値・n数で分解して持つ
- 標準偏差の差分処理は統計の式を素直に使う



ベイズ更新とかも差分アルゴリズムには向いてます

サーバー側アーキテクチャへの落とし込み

サーバーレスアーキテクチャを採用した経緯

初期の要件は「ログを安く蓄積。でも必要になったときにはとりだしやすいように。」

開発当初Amazon DynamoDBは存在したものの、 Amazon Kinesis、AWS Lambdaいずれも存在しない時代

・**初期モデル(2015年2月)**
Amazon EC2（ログブローカ）※→ Amazon DynamoDB
→ Amazon S3
※リアルタイムログを収集する可能性も視野に、スマホ間のサーバの通信プロトコルはMQTTを採用

・**第二世代(AWS Lambda リリース前の過渡期)**
Amazon Kinesis→AWS Elastic Beanstalk※→Amazon DynamoDB→AWS Elastic Beanstalk※→ Amazon S3
※re:Invent でLambdaが発表されていたので、後で移植できるよう、node.js で開発しEBで稼働

・**第三世代(2015年11月, JINS MEMEサービスイン)**
Kinesis → AWS Lambda → Amazon DynamoDB → AWS Lambda → Amazon S3

アーキテクチャとして考えたこと

λ（ラムダ）アーキテクチャ

事業がどうなるかわからないからランニングコストは抑えたい、一方でリアルタイム処理、バッチ処理の両方が必要になる可能性も高そうだったので、リアルタイムとバッチのハイブリッドとなるλアーキテクチャを意識した設計に。ログは重要な資産として捨てたくないので安価なS3に保管。

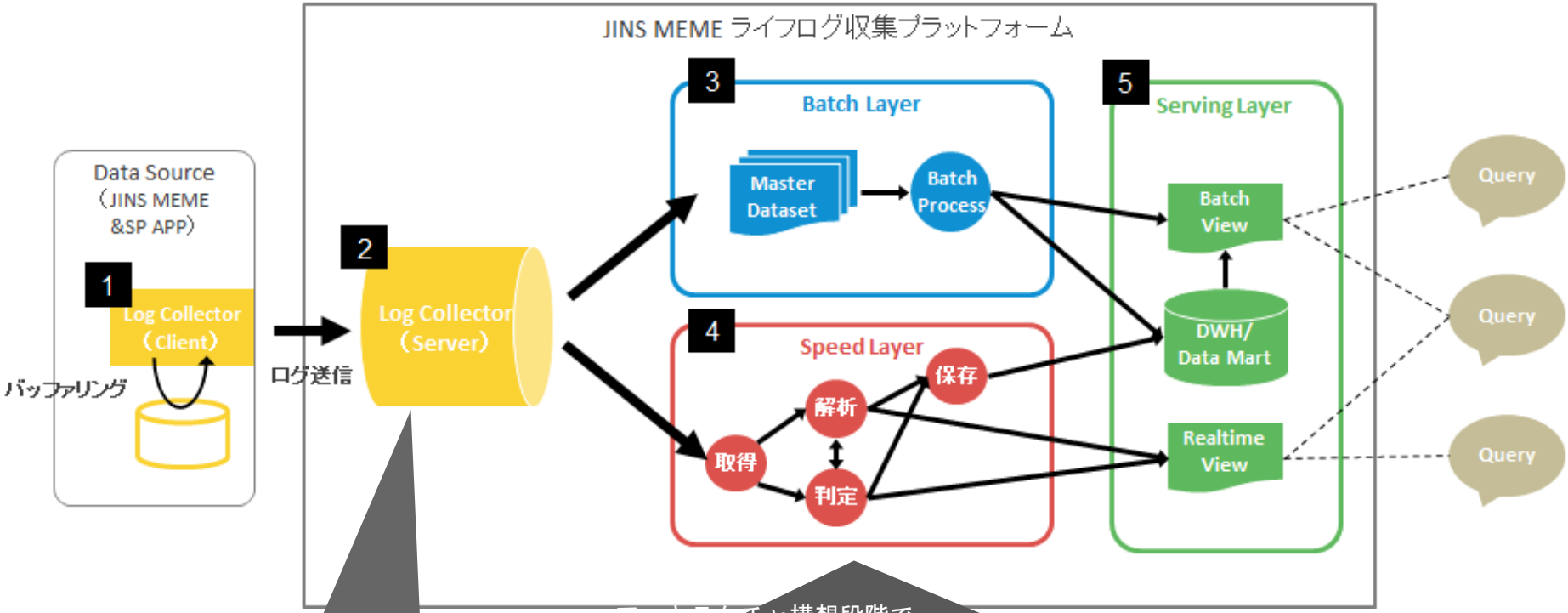
マイクロサービスアーキテクチャ

サービスの統廃合や作り変えを視野にいれ、最初からマイクロサービスアーキテクチャを採用。JINS MEMEで構築したOAuth2.0 ProviderがOpenID Connect Providerに正常進化し、JINSデジタルサービスの認証基盤に。

データレイク（社内分析用）

S3に蓄積されたログの取り出し。データサイエンティストに親しみのあるSQL（Presto/EMR）でデータの抽出が可能。使い勝手よりも、ランニング安く＆変化に強くすることをコンセプトに開発。

初期構想段階のアーキテクチャ

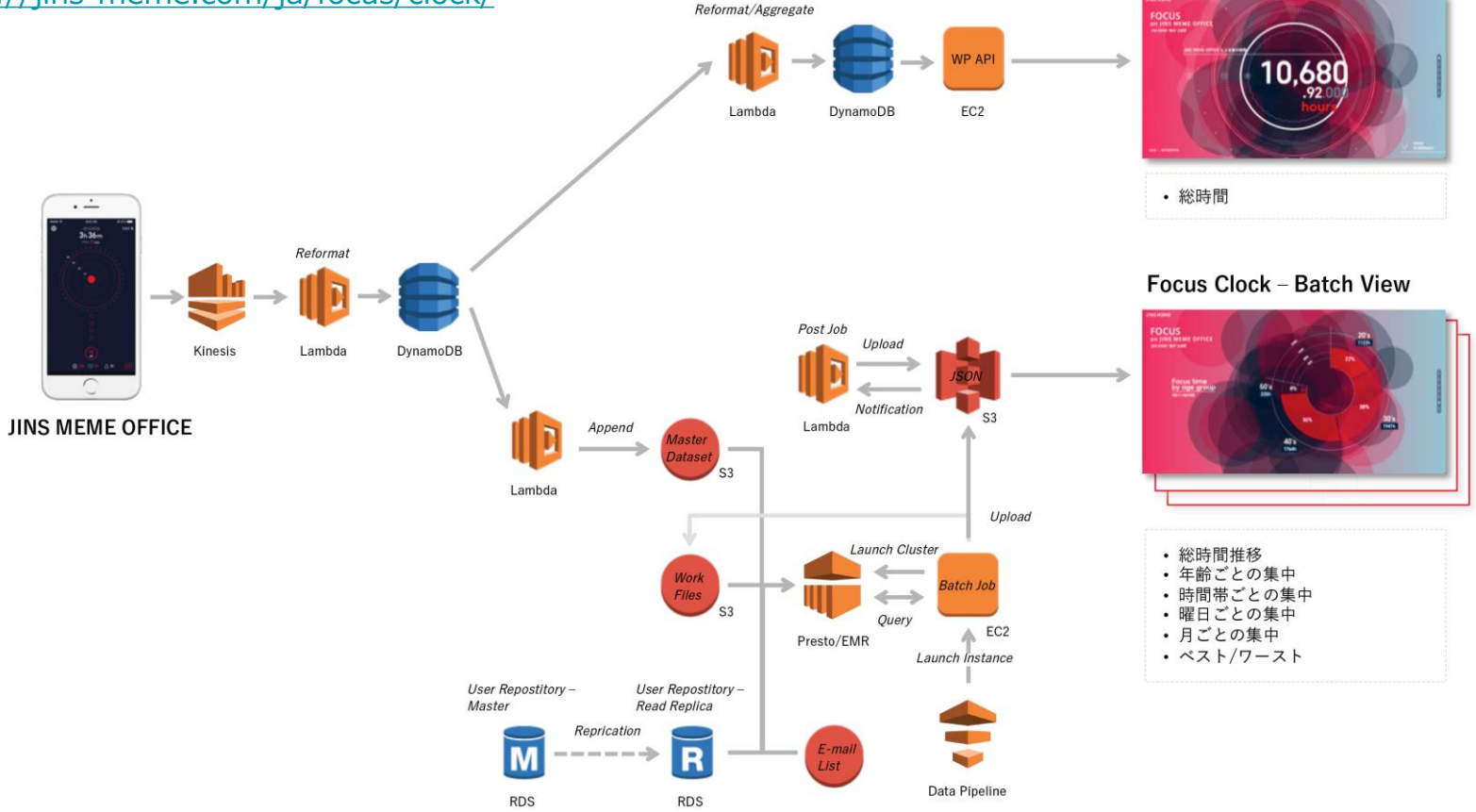


初期はMQTT Brokerを構築。
Amazon Kinesis、AWS Lambdaに順次入替えした

・アーキテクチャ構想段階で、
（１）安くデータを蓄積・保存、（２）Batchデータ取り出し、（３）リアルタイムデータ処理、を担保
・後はその時時点で最適なプラットフォーム・技術で各サブシステムを構築
・サブシステムはそれぞれ独立したサービス（マイクロサービス）として実装されているので、よりよい要素技術・プラットフォームが登場したら入れ替えていくことを意識

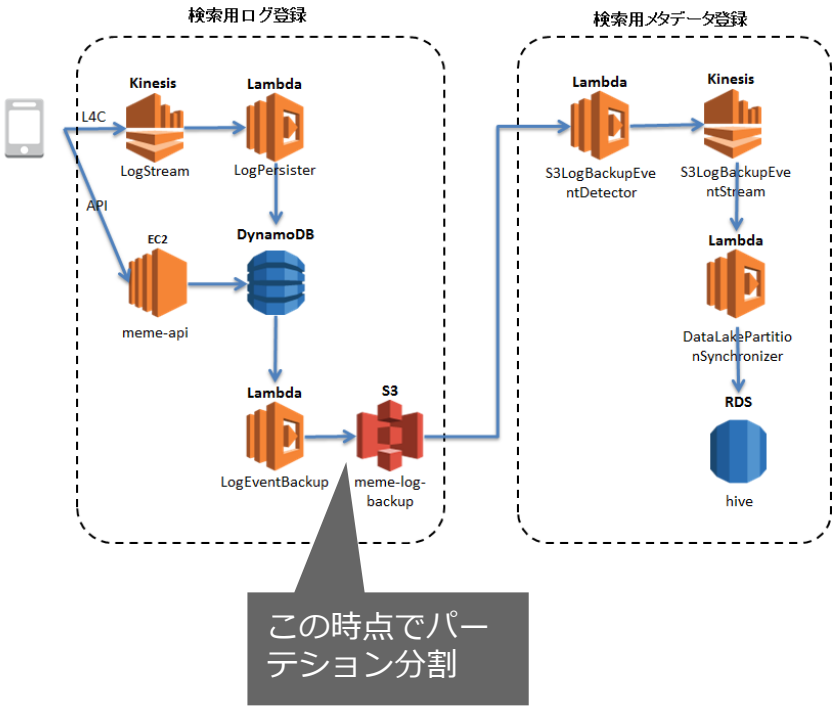
最新サービスのアーキテクチャ (Focus Clock)

https://jins-meme.com/ja/focus/clock/

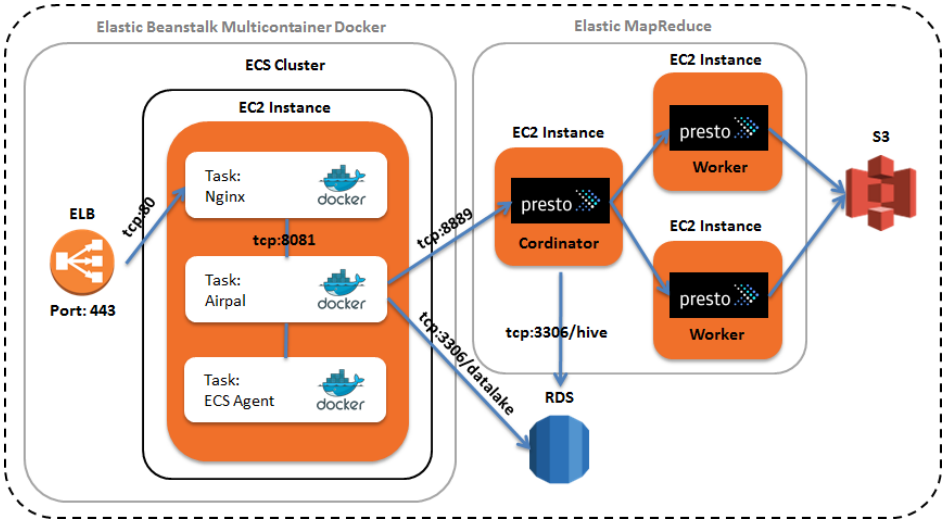


データレイクのアーキテクチャ

データ登録



データ検索



開発プロセス

開発スピード重視で2-Phaseでツールを使い分けています

プロトタイピング（試行錯誤）：

・なんだかんだSQLは使いやすいので Amazon RDS にデータ(raw/ref/art)をストア
・Shiny(R言語のパッケージ)を使用し、Webで半リアルタイムに演算結果を確認することでロジックを高速習熟
・実験的に作成したアルゴリズムの習熟度が上がったら、逐次処理化・疎結合化・ステートレス化までをR/Shinyで検証し切る

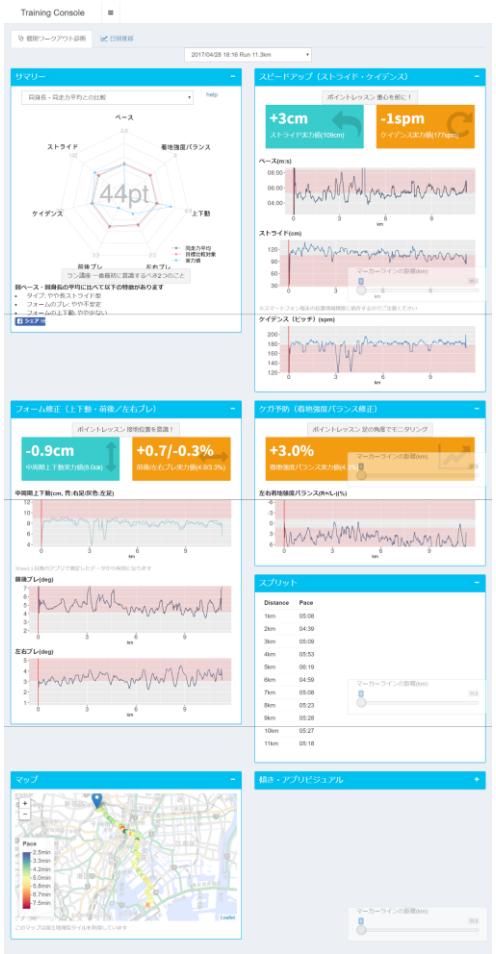
ステージング・本番：

・Shiny版と途中段階の演算結果が同じことをアプリ・サーバーのレイヤー毎に確かめながらリリースまで進める

Shinyデモ

作りこみサンプル

https://jinsmeme.shinyapps.io/training_console_sample/



ウェアラブルに足を突っ込んでみての雑感

分業が進みすぎていてインテグレーションが大変ですが、最新のツールを使用し理解できる範囲を拡げることでインテグレーションポイントが減りました

- ハード部品： メカエンジニア
- センサー回路： エレキエンジニア
- ファームウェア： 組み込みエンジニア
- センサーデータ・アルゴリズム： 統計エンジニア、データアナリスト
- アプリ： アプリ開発者
- クラウド： SIer

積極的に最新ツールを使いインテグレーション能力をつけておかないと、どこかから（主にAmazonさん）新サービスが出てきた時に仕事がなくなる、みたいなことが起きるかもしれません、、、

まとめ

- バイタルデータ処理は通信帯域やバッテリーの制約が厳しく、制約に合わせてアルゴリズム配置が決まっていく
- サービスリリース後も大幅な処理の変更が続いていくため、フレキシブルな処理構成をとれるラムダアーキテクチャ・マイクロサービスアーキテクチャは現在取りうるアーキテクチャとしては非常に効果的であった
- ツールは常に進化しており、新しいものを導入し続けることで開発やオペレーションのスリム化が実現可能に
- Enjoy serverless-life!