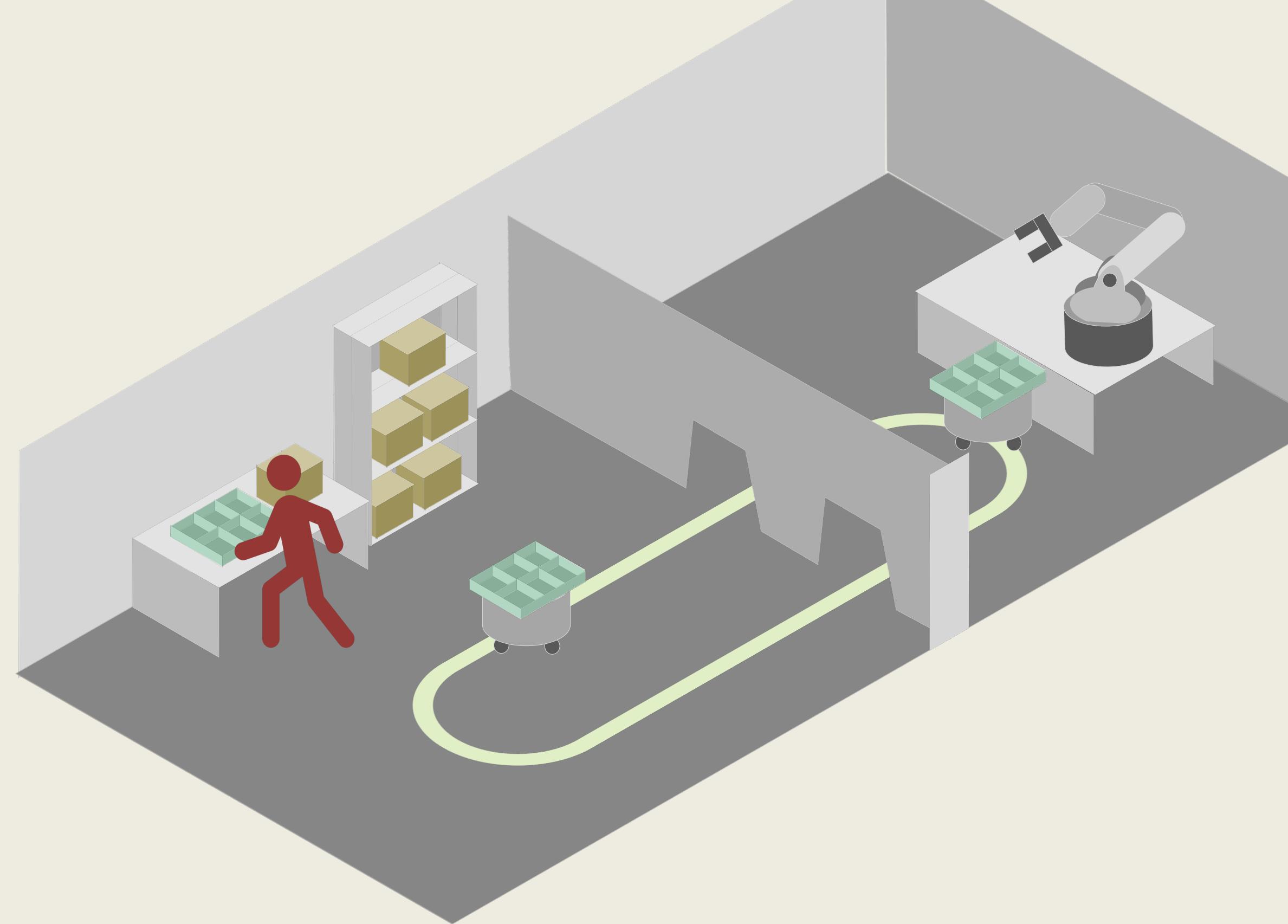


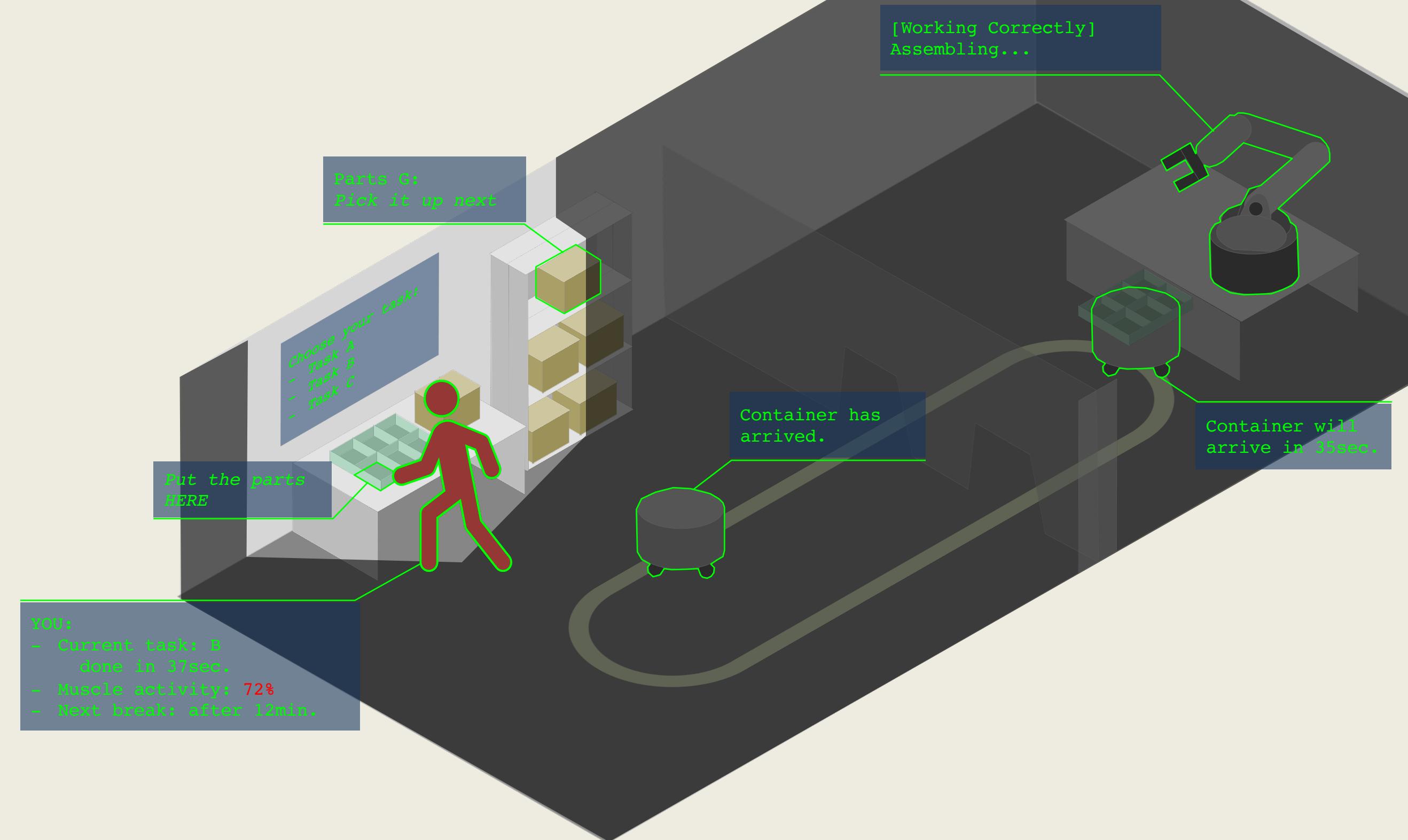
DhaibaConnect

産業技術総合研究所 人工知能研究センター
デジタルヒューマン研究チーム
遠藤 維

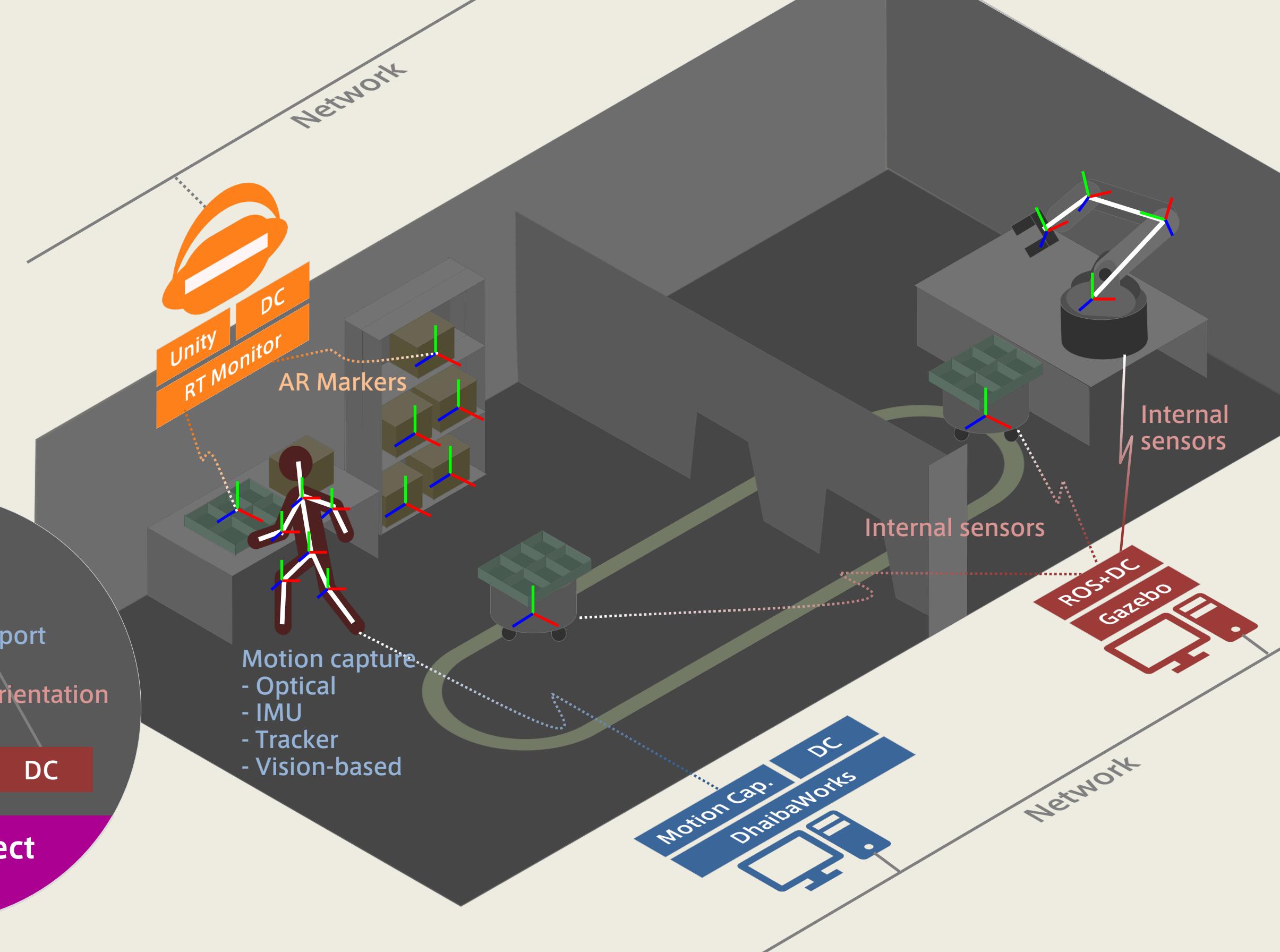
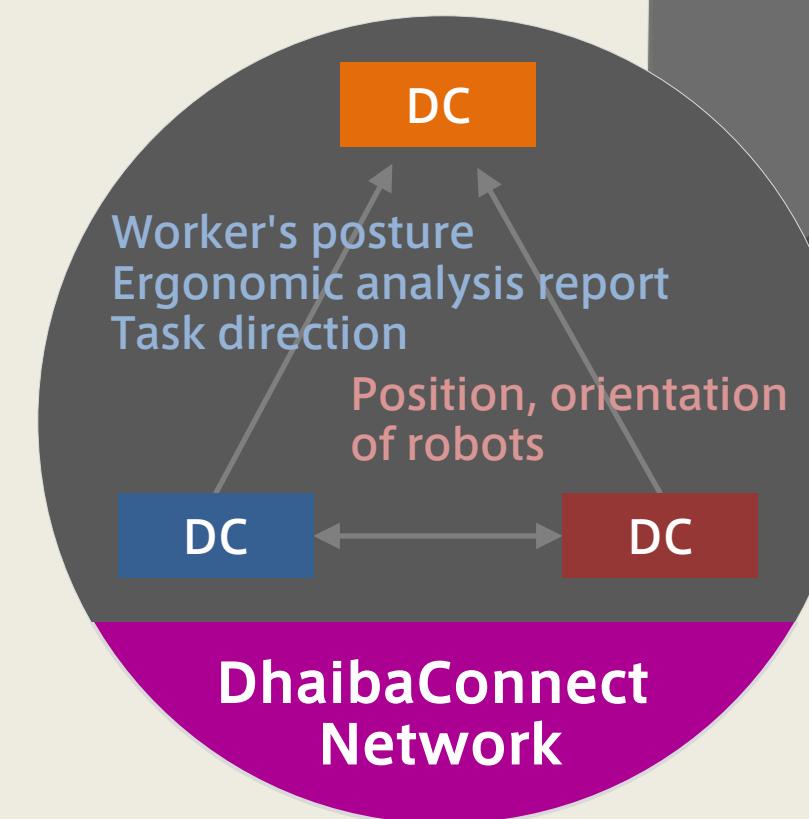
Physical World



Augmented World



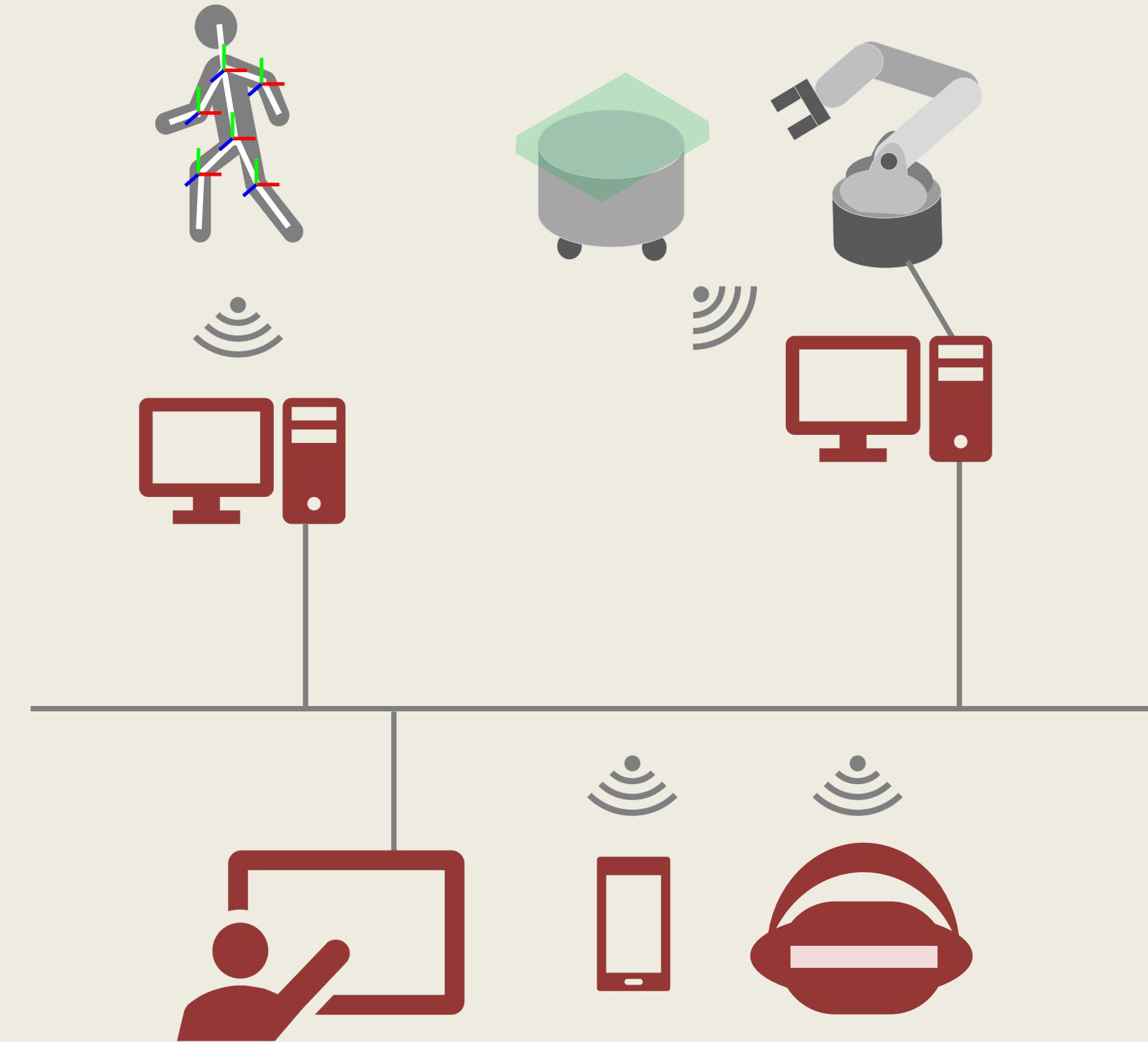
Interface of Cyber-Physical System



Concept of Cyber-Physical System

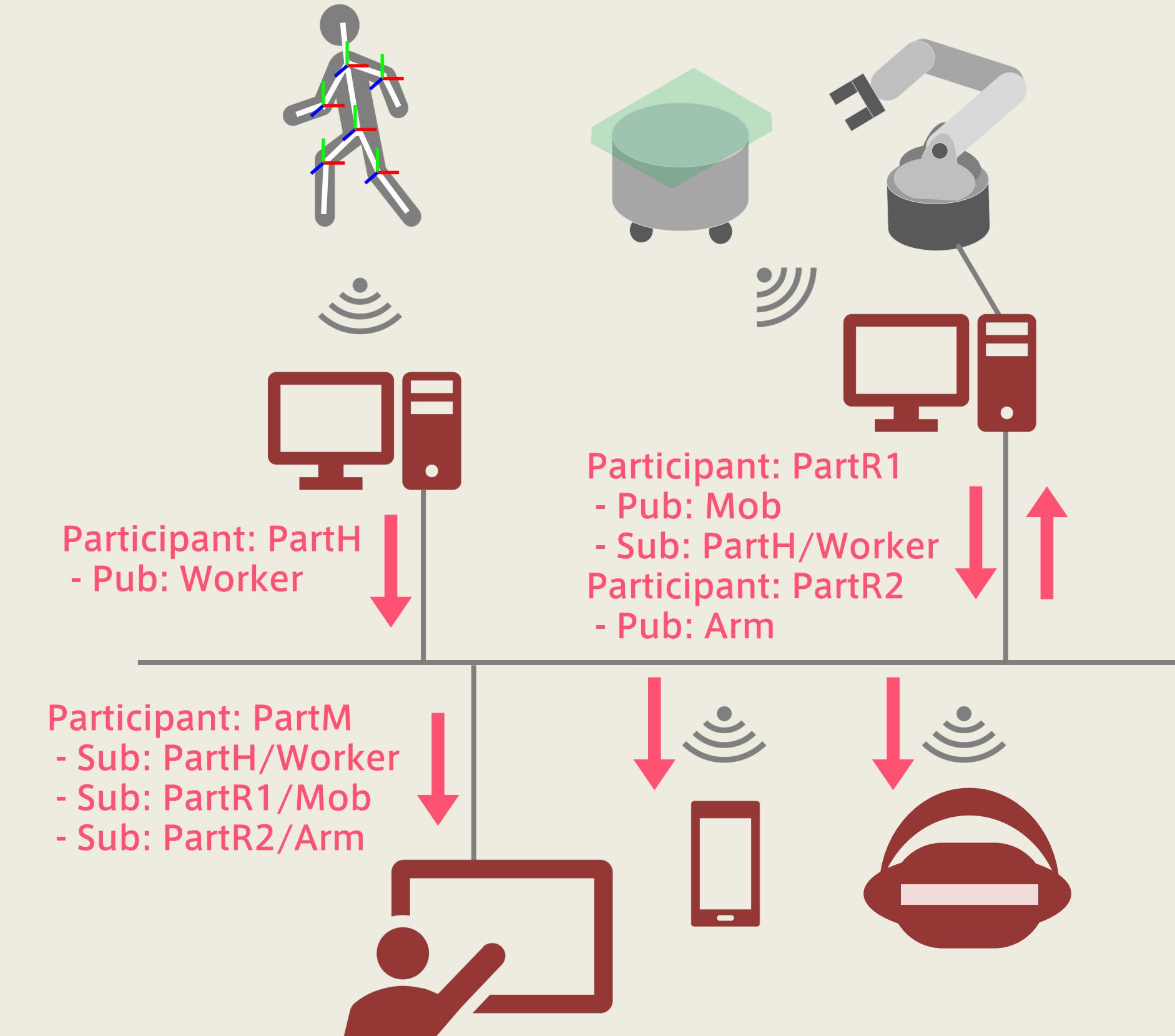
データ通信ミドルウェア

- － ユーザが定めた任意の形式のデータをネットワーク上のソフトウェア間で高速に通信
- － Data Distribution Service (DDS)規格に対応
 - 実装ライブラリとしてeProsimma Fast-RTPSを使用
- － DhaibaWorksのプラグインのほか、任意のソフトウェアでリンク可能なライブラリとして実装
 - Windows, macOS, Linux
 - モバイルOS上でもおそらく動作可能
- － 導入のメリット
 - 多様な計測システムを複数の計算機上に分散・同期
 - DhaibaWorksを移植できないモバイル端末等でも必要な情報をリアルタイムに共有



Data Distribution Service (DDS)

- Publish/Subscribe型のデータ通信規格
 - Participant : DDSへの参加者。通常はソフトウェアごとに設定
 - トピック : 送受信されるデータ構造の1かたまり
 - Publish : トピックをネットワーク上に「発行」
 - Subscribe : 発行されているトピックを「購読」
- 通信プロトコル : RTPS
 - Real-Time Publish-Subscribe Wire Protocol
 - » ROS2でも採用されている
- トピック名を指定することで、自動的に発行している計算機と接続し、購読できる
 - TCP/IPと違いサーバが不要：負荷を分散できる
 - UDPと違い発行者が購読者を把握できる：帯域を節約可能
 - DhaibaConnectではネットワーク上に存在するトピックリストを取得することも可能



DhaibaWorksからPub

— リンク構造 (Armature)

- 光学式、IMU、デプスセンサ、イメージセンサ

— ランドマーク点群 (FeaturePoints)

- 光学式、デプスセンサ、トラッカー、ARマーカ

— メッシュ

- 製品モデルの形状

— カメラ

- ユーザの視野

— 任意のスコア

- 人間工学指標解析結果

その他のシステムからPub

— リンク構造

- ロボットアームの位置姿勢

— ランドマーク点群

- モビリティロボットの位置姿勢

- 人流解析

— メッシュ・点群

- 環境の三次元計測点群・メッシュ

— カメラ

- HMDの視野

eProsima Fast RTPS

- eProsima_FastRTPS-1.7.2-Linux.tar.gzを解凍
- requiredcomponents/eProsima_FastCDR-1.0.8-Linux.tar.gzを解凍
- ターミナルから入力：
 - cd <eProsima_FastCDR-1.0.8-Linuxフォルダのディレクトリ>
 - ./configure --prefix=/usr/local
 - make
 - sudo make install
- cd <eProsima_FastRTPS-1.7.2-Linuxフォルダのディレクトリ>
- ./configure --prefix=/usr/local
 - make
 - sudo make install

CMAKEのインストール

- ターミナルから入力：
 - sudo apt install cmake
 - sudo apt install cmake-curses-gui

DhaibaConnectN

- DhaibaConnect r190811.tar.gzを解凍
- ターミナルから入力：
 - cd <DhaibaConnectN/buildフォルダのディレクトリ>
 - cmake ..
 - ccmake .. **fastRtpsを/usr/localにインストールした場合は不要**
» DDS_INC_DIRとDDS_LIB_DIRを設定
 - make
 - sudo make install

DhaibaConnectNSample

- DhaibaConnectNと同様にビルドする
 - sudo make installは不要

```
int main()↓
{↓
    // Obtains an unique instance of DhaibaConnect::Manager.↓
    Manager* manager = Manager::instance();↓
    ↓
    // Initializes the manager.↓
    // The first parameter is a participant name.↓
    manager->initialize("DhaibaConectNSample");↓
    ↓
    // Creates and registers two publishers.↓
    // The parameters of Manager::createPublisher() are↓
    // topicName, topicTypeName, isUniversalTopic, requiresHighReliability↓
    // respectively.↓
    // When isUniversalTopic is true, the topic name is set to topicName.↓
    // Else, it is set to <Participant Name>/<topicName>.↓
    // requiresHighReliability should be set to true, if the publisher↓
    // sends the data only once according to subscriber's request.↓
    PublisherInfo* pubArm = manager->createPublisher(↓
        "SampleArmature.Armature", "dhc::Armature", false, true);↓
    PublisherInfo* pubLink = manager->createPublisher(↓
        "SampleArmature.LinkState", "dhc::LinkState", false, false);↓
    ↓
```

```
// Connections::connect() can connect a signal object with a slot.↓
// PublisherInfo::matched is a signal where the type is↓
// Signal<void(PublisherInfo*, MatchingInfo*)>.↓
// Any function or class method which has the same parameters as the signal has↓
// can be the slot.↓
// All slots connected to the signal is invoked when the signal is emitted.↓
//↓
// In this example, the publisher pubArm send the dhc::Armature data↓
// only when a new subscriber for the "DhaibaConectNSample/SampleArmature.Armature"↓
// is found.↓
// Twj0 for each link is 4x4 matrix which represents the local coordinate system↓
// of the relative joint in initial state.↓
Connections::connect(&pubArm->matched,↓
{[&](PublisherInfo* pub, MatchingInfo* info){↓
    dhc::Armature data; data.links().resize(2);↓
    data.links()[0].linkName() = "Link0";↓
    SetIdentityMatrix(data.links()[0].Twj0().value());↓
    data.links()[0].tailPosition0().value()[0] = 1000;↓
    ↓
    data.links()[1].linkName() = "Link1";↓
    data.links()[1].parentLinkName() = "Link0";↓
    SetIdentityMatrix(data.links()[1].Twj0().value());↓
    data.links()[1].Twj0().value()[12] = 1000;↓
    data.links()[1].tailPosition0().value()[0] = 2000;↓
    pub->write(&data);↓
}});↓
```



```
double r(0.0);↓
    ↓
    // On the other hand, the publisher pubLink continuously sends current joint ↓
    // rotation/translation as the array of 4x4 matrix R.↓
    // Current local coordinate system Twj can be recursively calculated as follows:↓
    // Twj = Twp * (Twp0.inv() * Twj0) * R↓
    // where Twp0/Twp is the initial/current local coordinate system for the joint of
    // the parent link.↓
    std::thread th([&](){↓
        {↓
            while(1){↓
                dhc::LinkState_data;↓
                data.value().resize(2);↓
                SetIdentityMatrix(data.value()[0].value());↓
                SetIdentityMatrix(data.value()[1].value());↓
                data.value()[1].value()[12] = 100.0 * sin(r);↓
                r += 0.1;↓
                pubLink->write(&data);↓
                this_thread::sleep_for(chrono::milliseconds(1000/30));↓
            }↓
        });↓
        th.detach();↓
    }
```

