AWS IoT 개요 2

- AWS Greengrass -

부산대학교

김호원

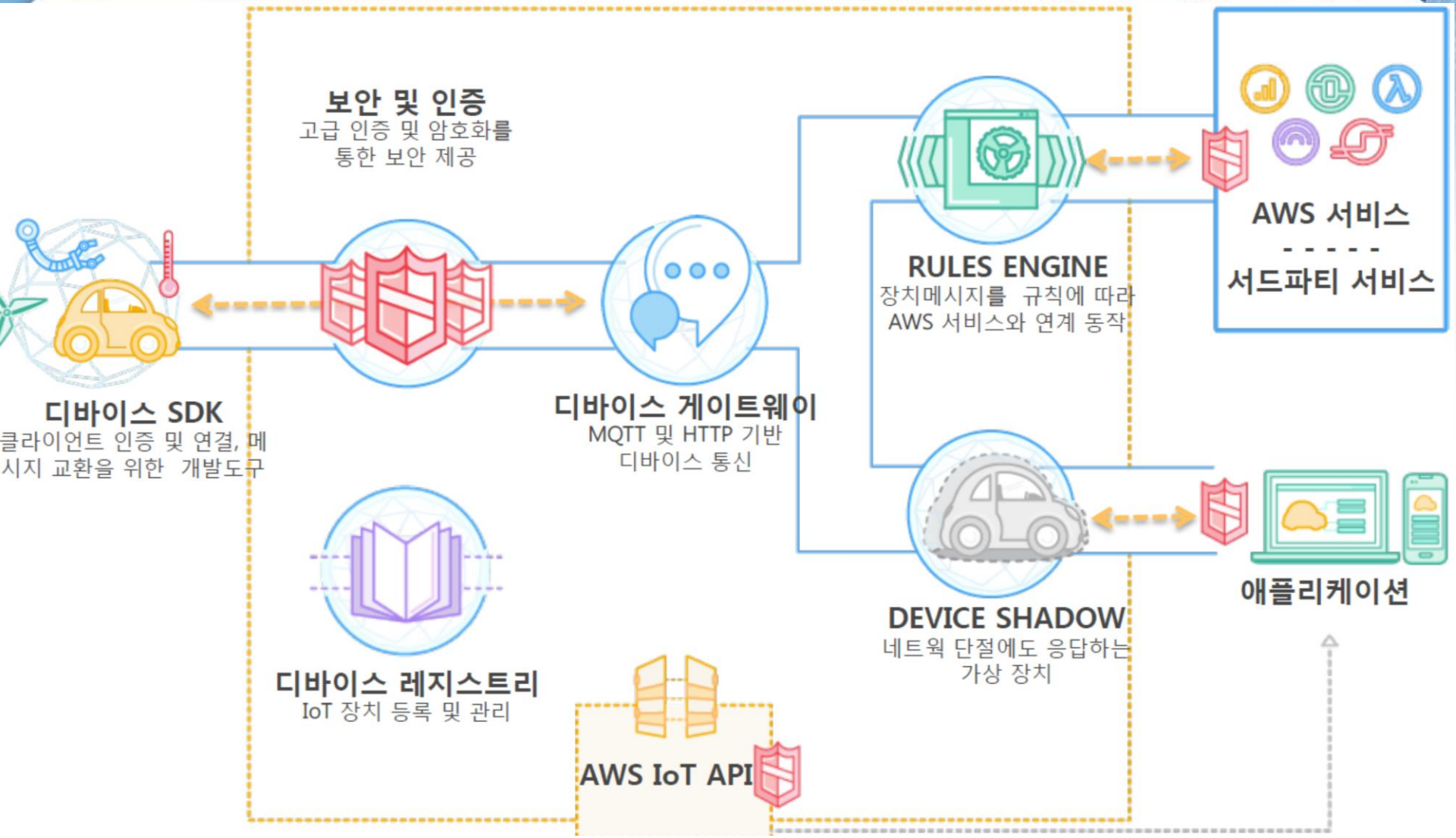
부산대
정보보호 및 IoT 연구실,
블록체인 보안 전문연구실,
사물인터넷연구센터

2018.11

AWS IoT 개요 2

I. AWS Greengrass 개요

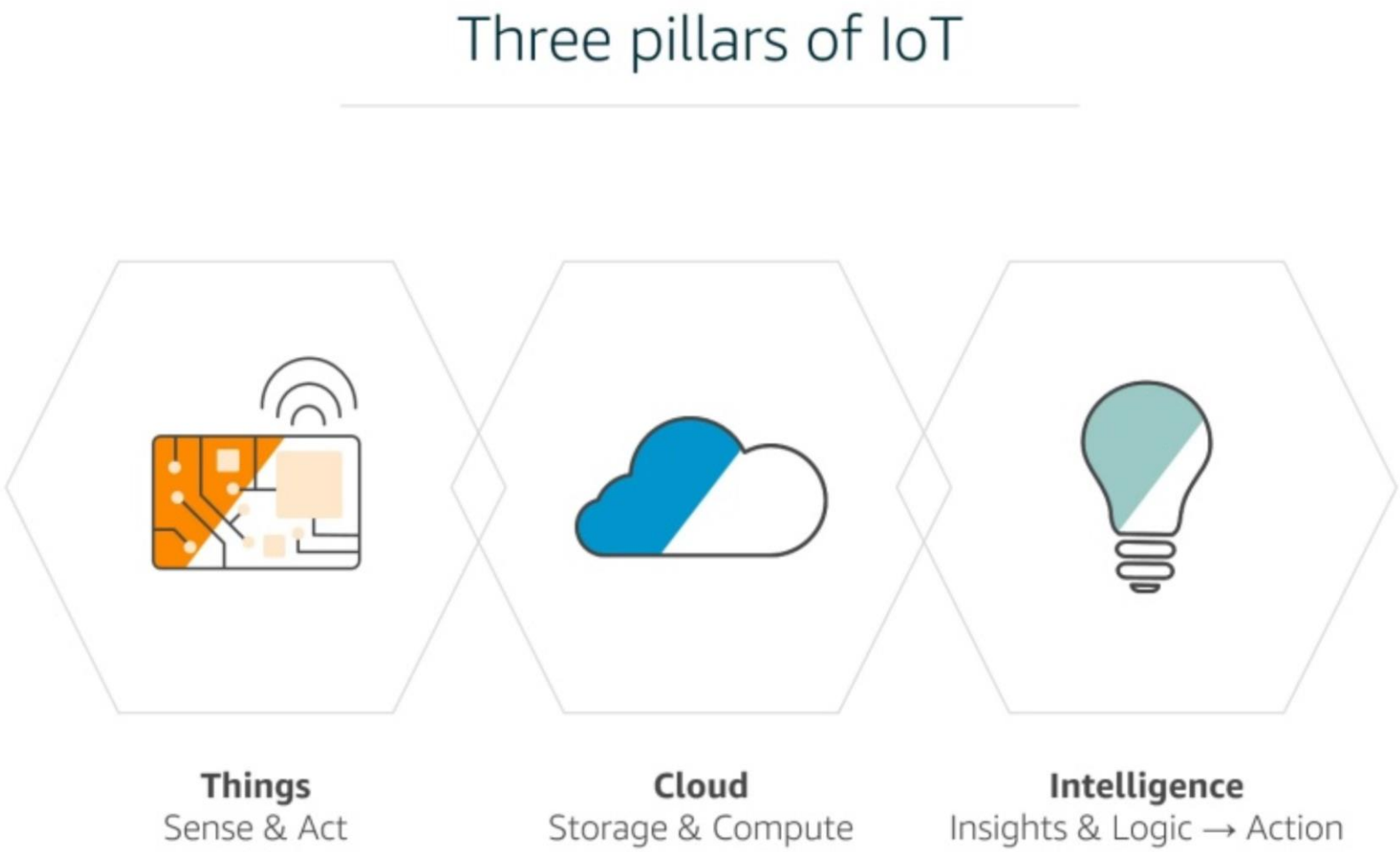
AWS IoT 개요



https://developer.amazon.com/blogs/post/Tx3828JHC709GZ9/Using-Alexa-Skills-Kit-and-AWS-IoT-to-Voice-Control-Connected-Devices

AWS Greengrass

IoT의 주요 구성 요소: 센싱/저장/가공/분석

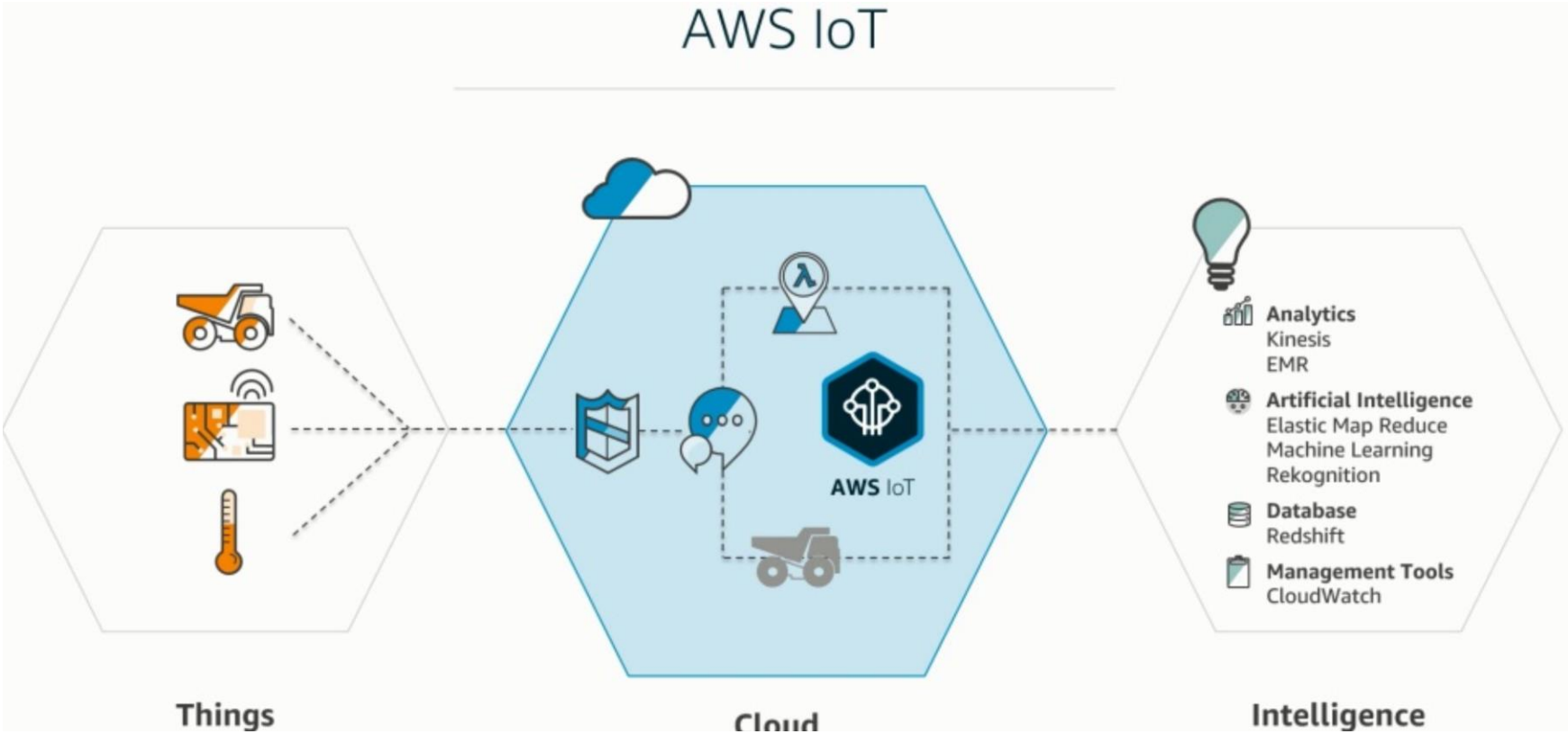


참고: AWS Greengrass Technical by Craig Williams

AWS Greengrass

IoT의 주요 구성 요소

– AWS IoT와 Edge computing



참고: AWS Greengrass Technical by Craig Williams

AWS Greengrass

AWS IoT도 Cloud Platform임

AWS IoT is a fully managed cloud platform that lets connected devices easily and securely interact with cloud applications and other devices.

1

Securely connect and manage any physical device across multiple networks and protocols

2

Extract and Filter data from your devices and take action with custom Rules

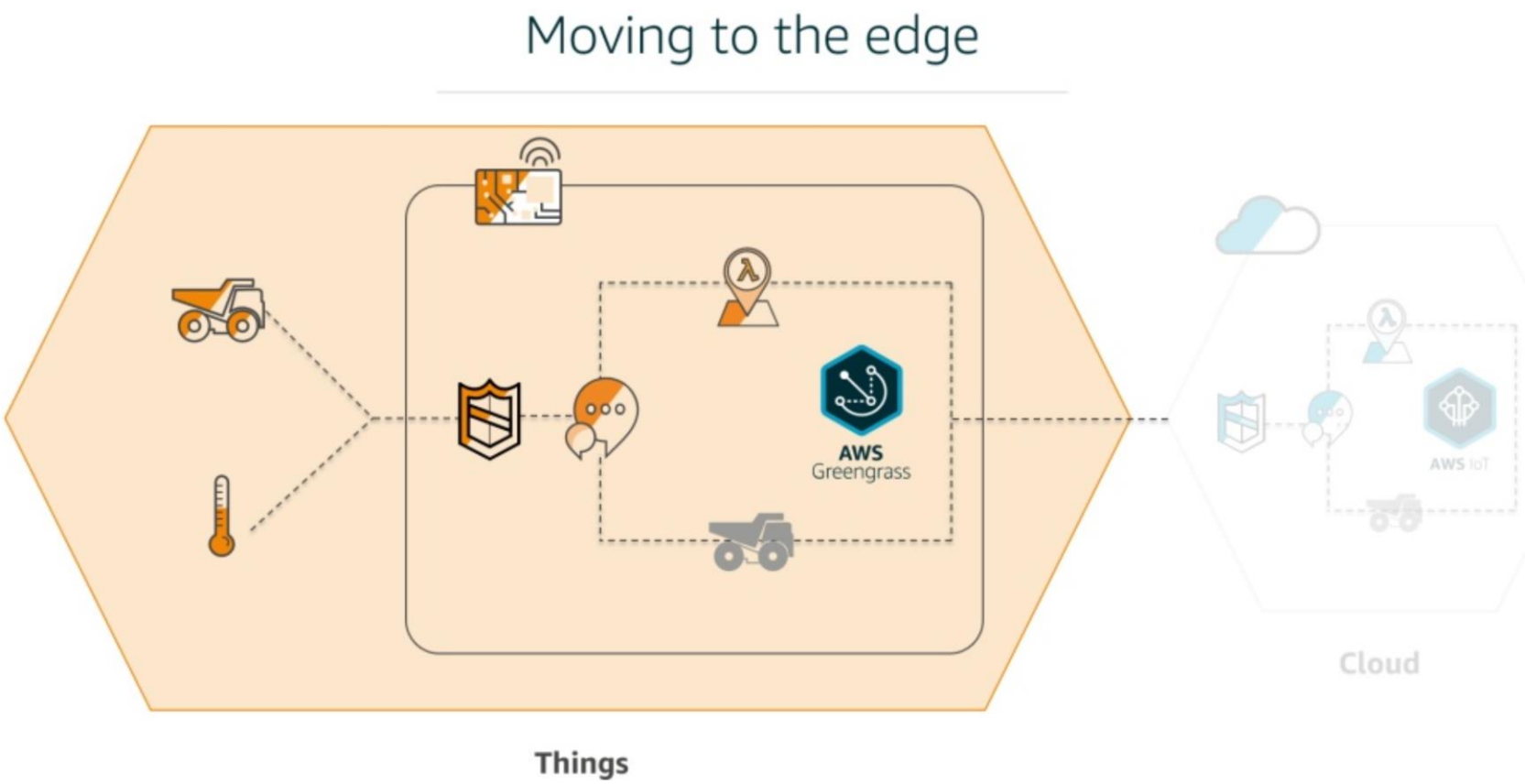
3

Create Web and Mobile Applications that Interact with Devices reliably at any time

AWS Greengrass

AWS Greengrass

– AWS Cloud 기술의 edge computing화 기술



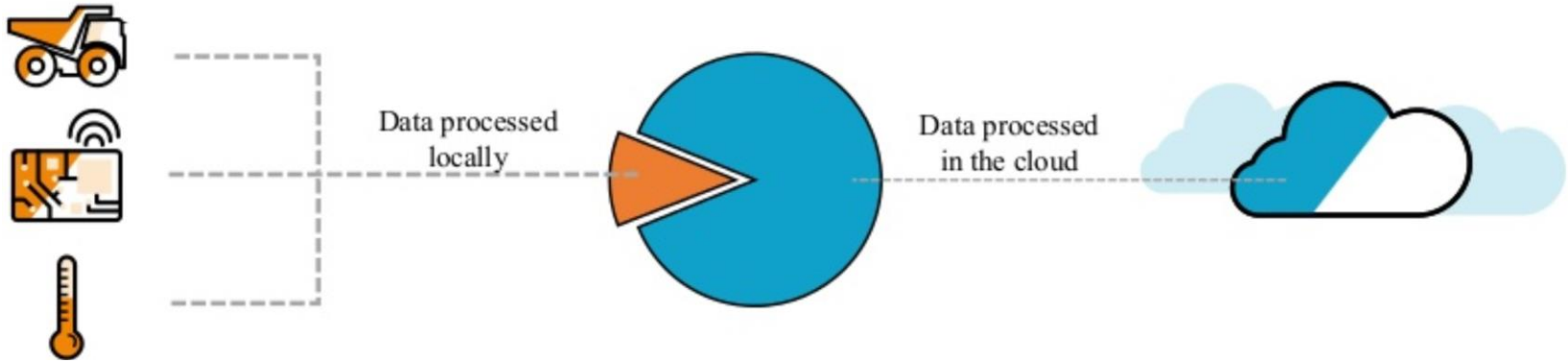
AWS Greengrass

AWS Greengrass

AWS Greengrass

Moving to the edge

AWS Greengrass extends AWS onto your devices, so they can act locally on the data they generate, while still taking advantage of the cloud.



AWS Greengrass

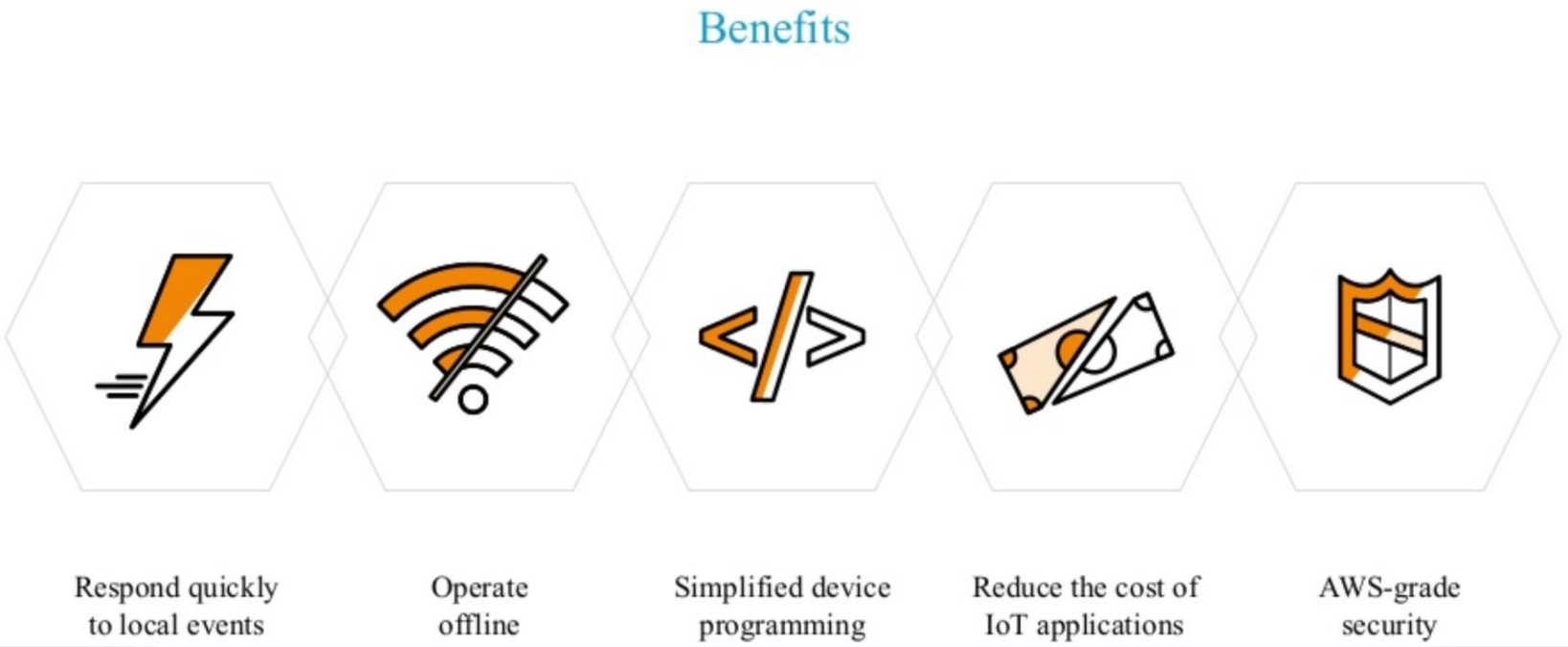
AWS Greengrass

– In Greengrass, Local Lambda, Local Device Shadows, Local Broker, Local Security can be used

AWS Greengrass

AWS Greengrass

- IoT 디바이스와의 원활한 통신(local event에대한 빠른 응답)
- Cloud/서버와 통신이 불안정한 상황에서도 서비스 가능
- 쉬운 디바이스/게이트웨이 프로그래밍
- IoT 응용 서비스 코스트 절감
- 높은 보안성 제공



AWS Greengrass

GGC(Greengrass Core)

- The runtime responsible for Lambda execution, messaging, device shadows, security, and for interacting directly with the cloud
- Required spec
  - Min Single-Core 1GHz Processor
  - Min 128MB RAM
  - X86 & ARM
  - Linux Kernel 4.4 wit OverlayFS enabled(Ubuntu or Amazon)
- Greengrass core takes advantage of your device's compute, memory, storage, and peripherals
- Any device that uses the IoT Device SDK can be configured to interact with Greengrass Core via the local network
  - AWS IoT Device SDK supports C, C++, Python 2.7, JavaScript
  - SQLite 3+

AWS Greengrass

GGC(Greengrass Core)

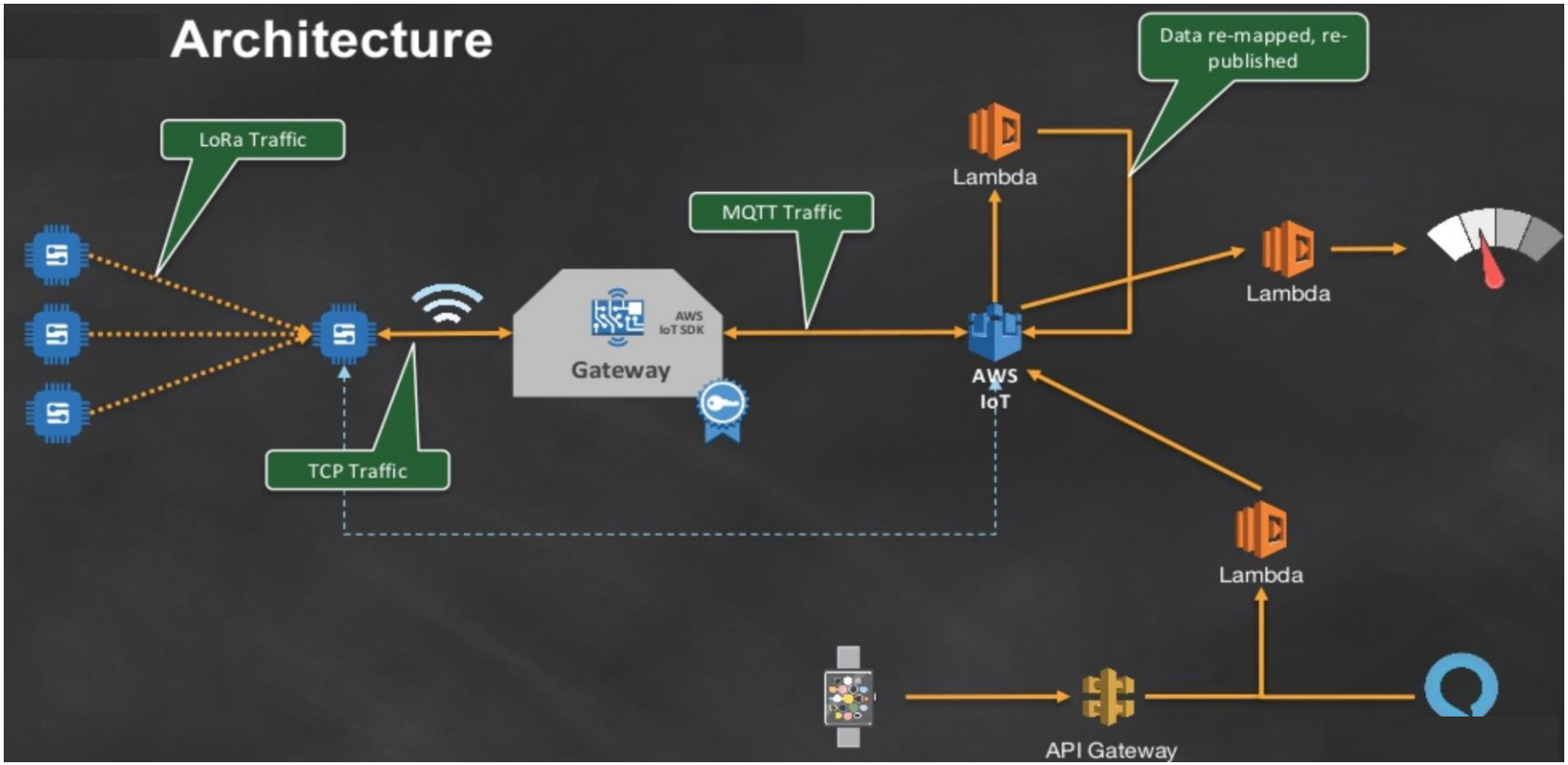
- Device work together locally
  - A Greengrass Group is a set of Cores and other devices configured to communicate with one another
- Devices work together with the cloud
  - Greengrass works with AWS IoT to maintain long-lived connections and process data via the rules engine
  - Your Lambda functions can also interact directly with other AWS services

AWS Greengrass

AWS Greengrass 응용

AWS Greengrass

AWS Greengrass 사용 환경 구조



II. AWS Greengrass 참고

AWS Lambda

Local Lambda

- Lambdas are event-driven compute functions
  - With Greengrass you can write Lambda functions in the cloud and deploy them locally
- Greengrass runs Lambdas written in Python 2.7
  - Invoke Lambda functions with messaging and shadow updates
- Local Lambda - What you can do
  - Command and control
  - Offline operation
  - Data filtering & aggregation
  - Iterative learning

Shadows

- Shadows ?
  - JSON documents that represent state of your devices and Lambdas
  - Define them however is logical to you : a car, an engine, a fleet
  - Sync to the cloud or keep them local

- Shadows - What you can do
  - Device state (current and desired)
  - Granular device state (only synched to the cloud for debug)
  - Dynamic configuration (e.g., numeric factors of an ML model)

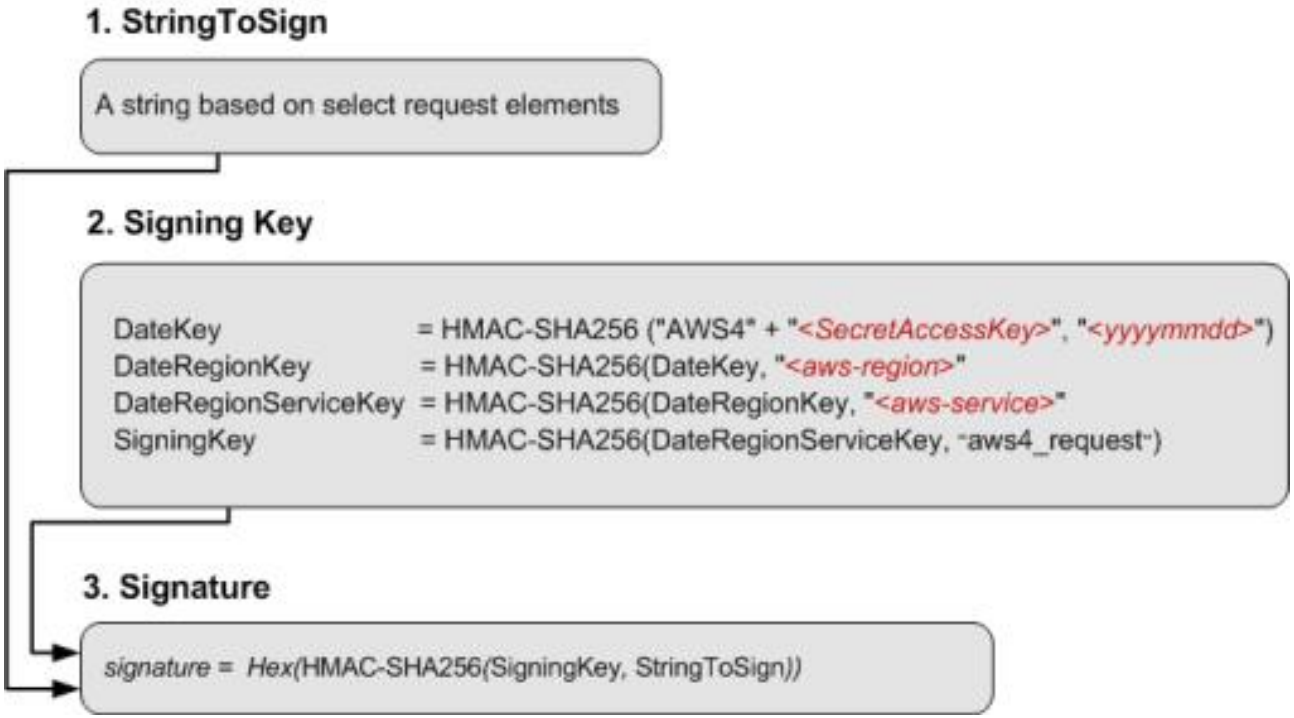
Messaging, Security

- Local MQTT Pub/Sub messaging
  - Define subscriptions between publishers and subscribers
  - Apply MQTT topic filters

- Security
  - Mutual authentication, both locally and also with the Cloud
  - Certificate on your devices can be associated to SigV4 credentials in the cloud
  - You can directly call any AWS service from Greengrass
  - Local GGC has its own root certificate
  - Each device certificate is signed by the GGC's root certificate
  - Greengrass core now uses the local CA and device certificate for authentication

Messaging, Security

SigV4 (Signature version 4)



Signature Version 4 is the process to add authentication information to AWS requests sent by HTTP.

For security, most requests to AWS must be signed with an access key, which consists of an **access key ID and secret access key.** These two keys are commonly referred to as your security credentials

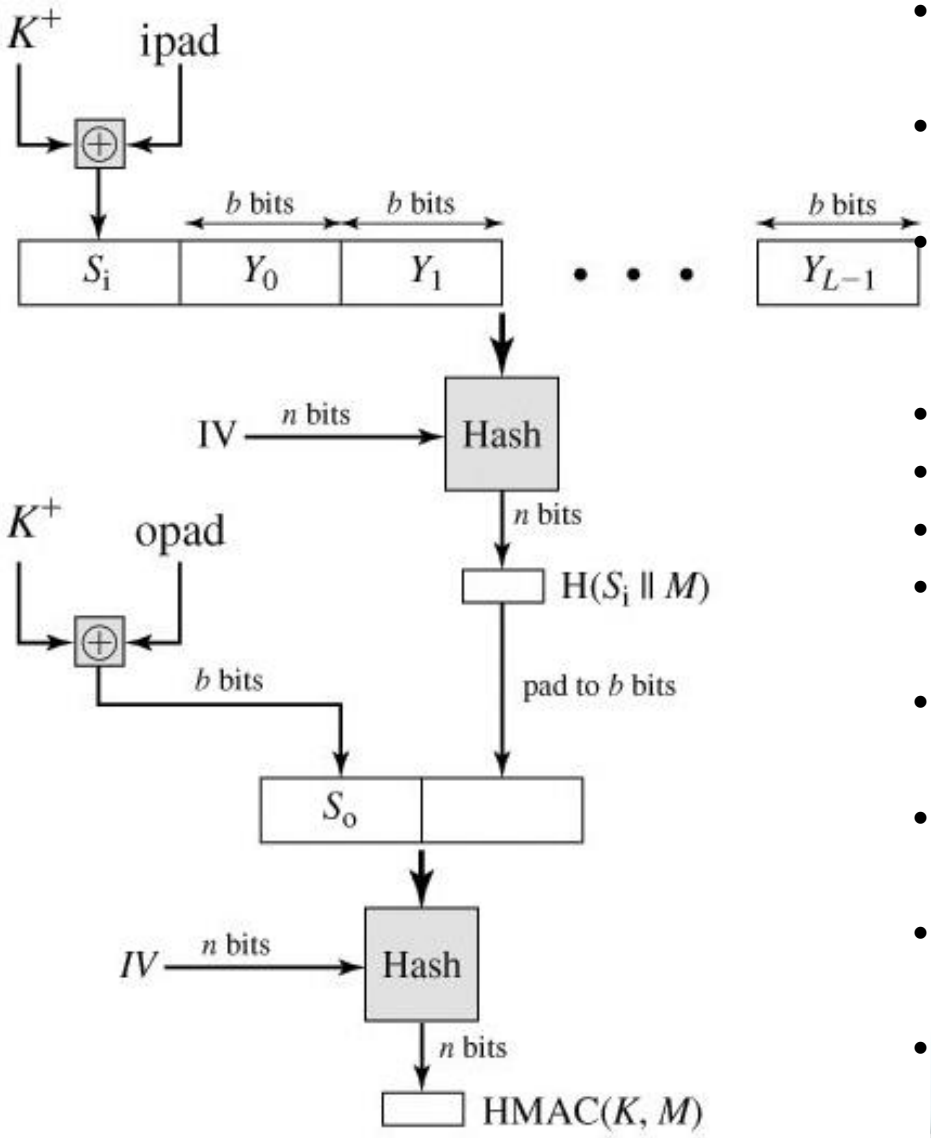
HMAC-SHA256

- specified as Internet standard RFC 2104
- uses hash function on the message:

$$HMAC_K = Hash[(K^+ \text{ XOR } opad) \;||\; Hash[(K^+ \text{ XOR } ipad)||M)]]$$

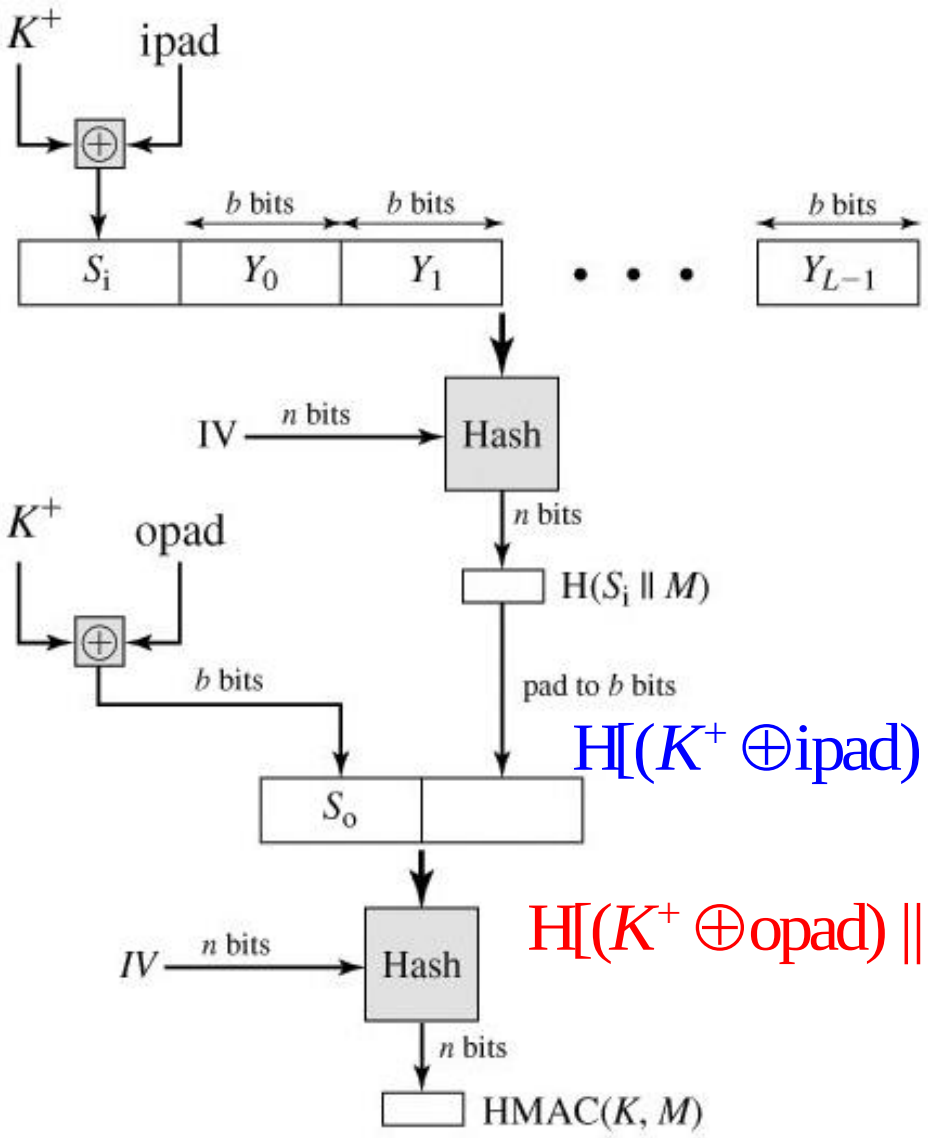
- where K^+ is the key padded out to size
- and opad, ipad are specified padding constants
- overhead is just 3 more hash calculations than the message needs alone
- any hash function can be used
  - eg. MD5, SHA-1, RIPEMD-160, Whirlpool

HMAC-SHA256



- H: embedded hash function (e.g., MD5, SHA-1, RIPEMD-160)
- IV: initial value input to hash function
- M: message input to MHAC (including the padding specified in the embedded hash function)
- Yi : ith block of M (0<=i<= L-1)
- L : number of blocks in M
- b: number of bits in a block
- n : length of hash code produced by embedded hash function
- K : secret key recommended is (>= n);
- K^+ : K padded with zeros on the left so that the result is b bits in length
- ipad : 00110110 (0x36 in HEX) repeated b/8 times
- opad : 01011100 0x5C in HEX) repeated b/8 times

HMAC-SHA256



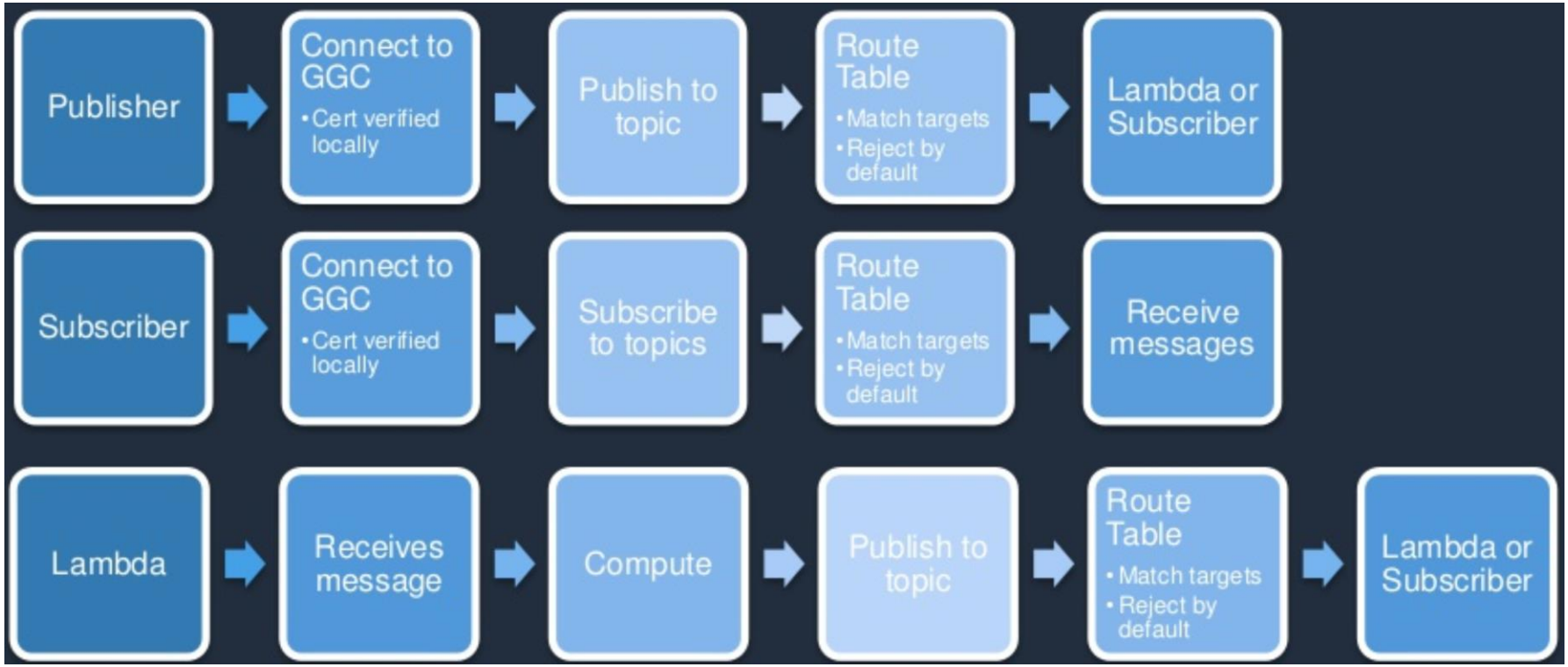
1. Append zeros to the left end of K to create a b-bit string K^+
2. XOR (bitwise exclusive-OR) K^+ with ipad to produce the b-bit block Si.
3. Append M to S_i.
4. Apply H to the stream generated in step 3.
5. XOR K^+ with opad to produce the b-bit block S_0
6. Append the hash result from step 4 to S_0
7. Apply H to the stream generated in step 6 and output the result.

$$H[(K^+ \oplus \text{ipad}) \| M]$$

$$H[(K^+ \oplus \text{opad}) \| H[(K^+ \oplus \text{ipad}) \| M]]$$

Broker specifics

Broker



Use Case - example

Gas Turbine Filter Maintenance

- Each turbine consists of multiple devices/sensors
- Each turbine has a single Greengrass core
- Lambda functions have enough logic to perform edge machine learning
- Models and Lambda changes can be centrally updated
- Lambda filters data and using it's edge ML can determine when to request maintenance/replacement of filters
- Only required data is sent up to AWS IoT once an hour

감사합니다

Q & A

부산대학교 전기컴퓨터공학부 김호원
부산대학교 사물인터넷 연구센터장
howonkim@pusan.ac.kr