GDC

AI-Driven QA: Simulating Massively Multiplayer Behavior for Debugging Games

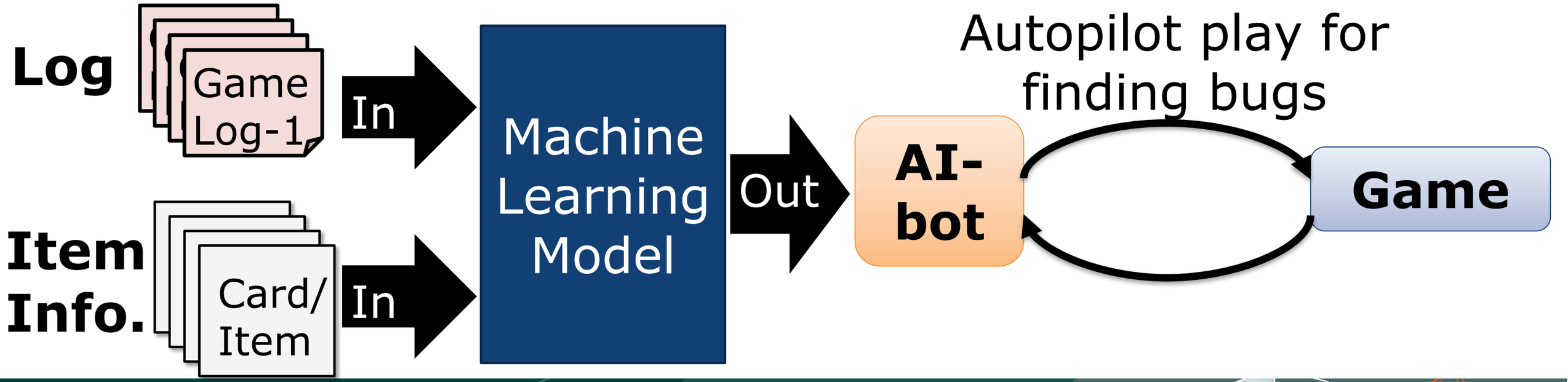
Shuichi Kurabayashi, Ph.D.
Cygames, Inc.
Keio University

GAME DEVELOPERS CONFERENCE® | MARCH 19–23, 2018 | EXPO: MARCH 21–23, 2018 #GDC18

UBM

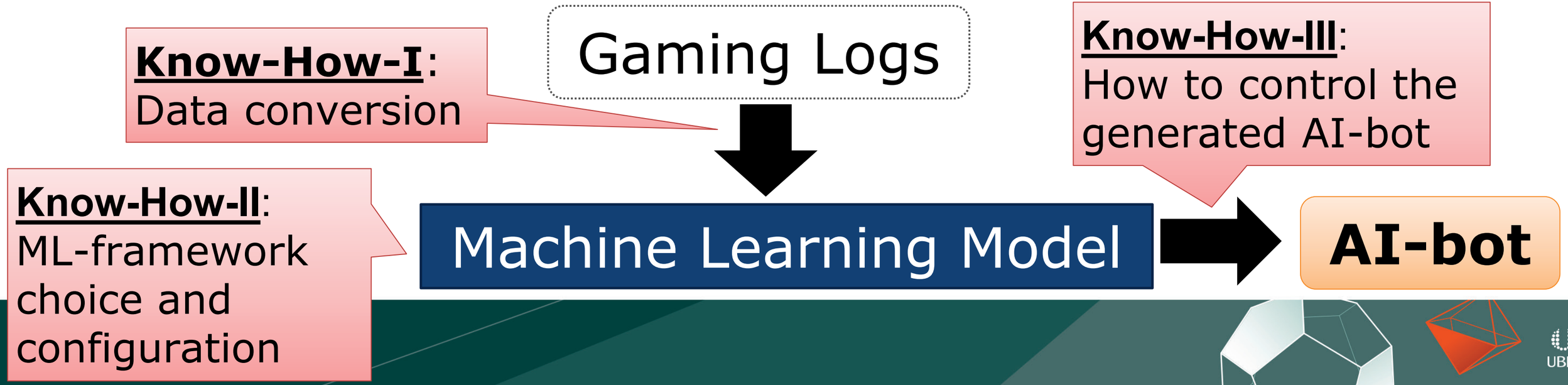
Summary

We disclose know-hows to develop an **AI-driven automatic quality assurance framework** that learns player behaviors from game logs to **simulate a wide range of playing styles**.



To build an AI-driven QA framework, we need:

- **Exploit the existing DNN software**, including ML-framework choice and neural network design.
- **Convert** a gaming log into a **"Machine-Learnable".**
- **Autopilot play** using massively many AI-bots.



Self Introduction

- Industrial Position: As a director of Cygames Research, I am leading AI and data-driven systems in Cygames, Inc.
- Academic Position: As a faculty, I am teaching database systems at the Graduate School of Keio University.

About Cygames Research

In-house research institute of Cygames. We are collaborating with many universities including UCSD, Keio, Universität Leipzig. We also welcome research partners from the industrial field!

https://research.cygames.co.jp

Agenda

Deep Neural Network corresponds to "**virtualized brain**" for thinking games

The **Headless** App Framework using **Linux containers** corresponds to "**virtualized body**" for playing games

Problem Definition

We need a method to check the integrity of a continuously evolving game.

Our target game genre: Online Collectible Card Game

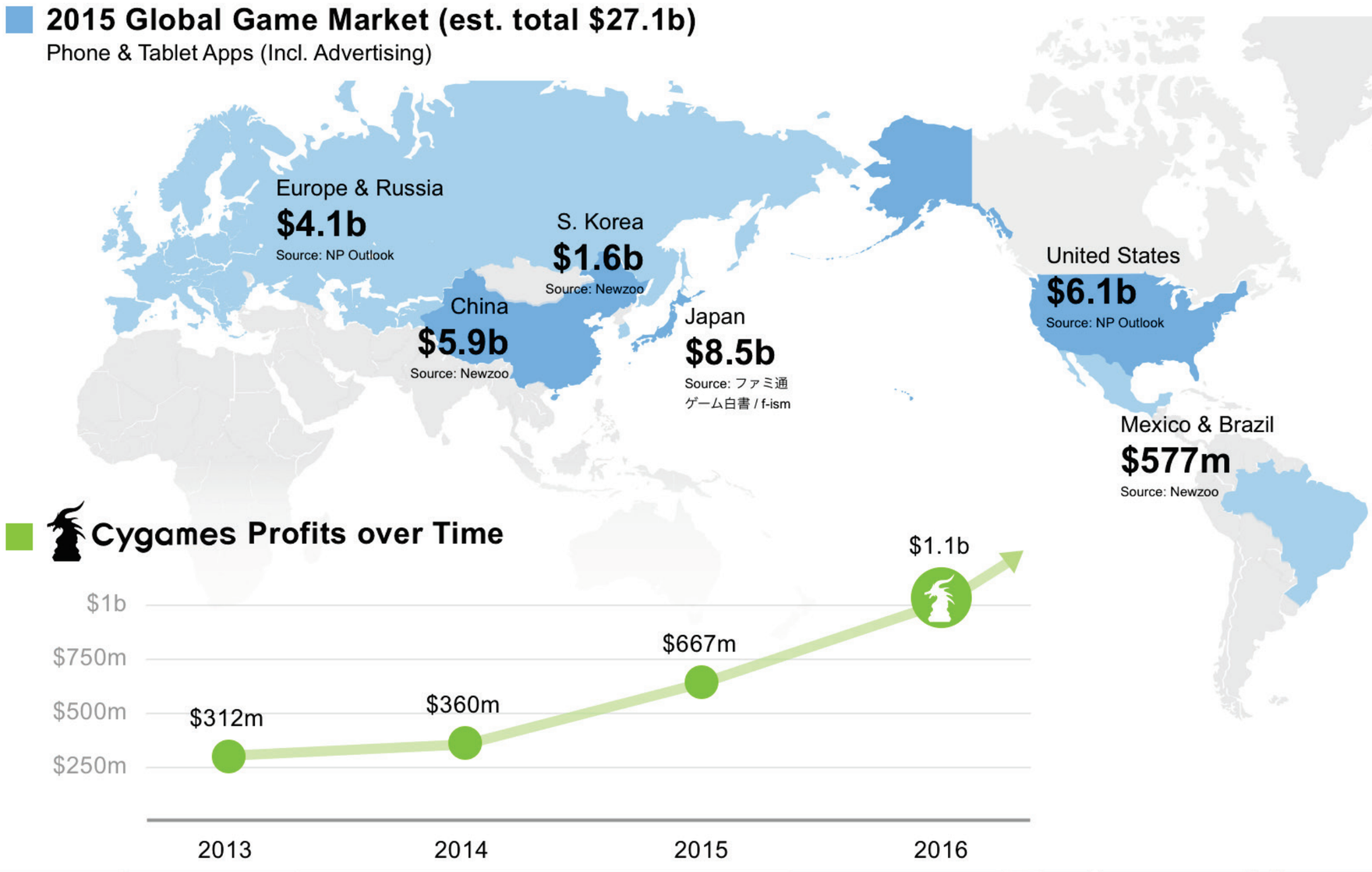
Millions of battles every day infinite ways to play

New card expansions every 3 months

The above screenshot is excerpted from
https://play.google.com/store/apps/details?id=com.cygames.Shadowverse

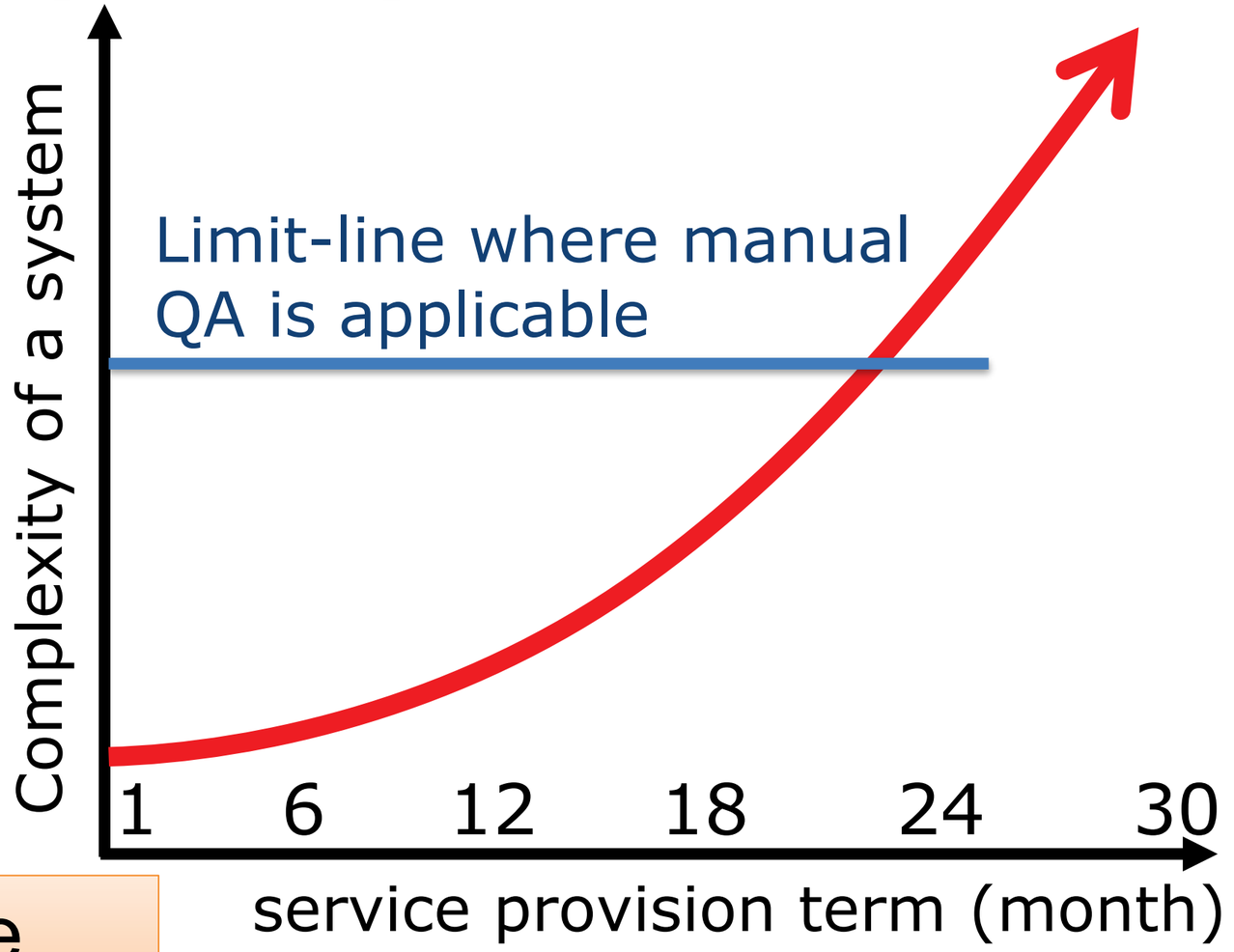
Introduction of Cygames, Inc.

Cygames is one of the premium mobile game developers in Japan



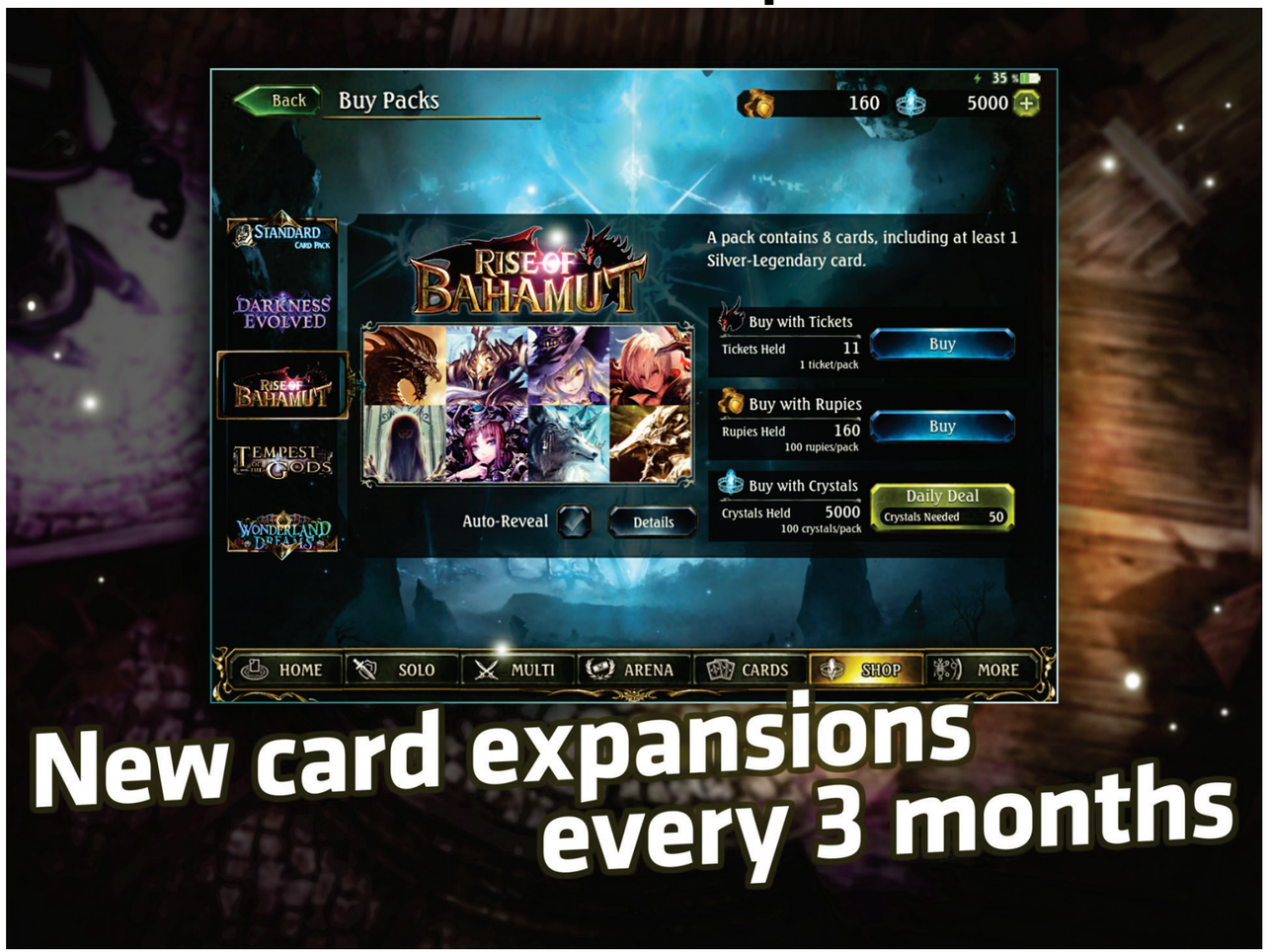
Complexity grows as a game grows

We are facing the situation that online games have been provided for several years. To keep the fun of games, we update games many times and add new items/cards/characters.



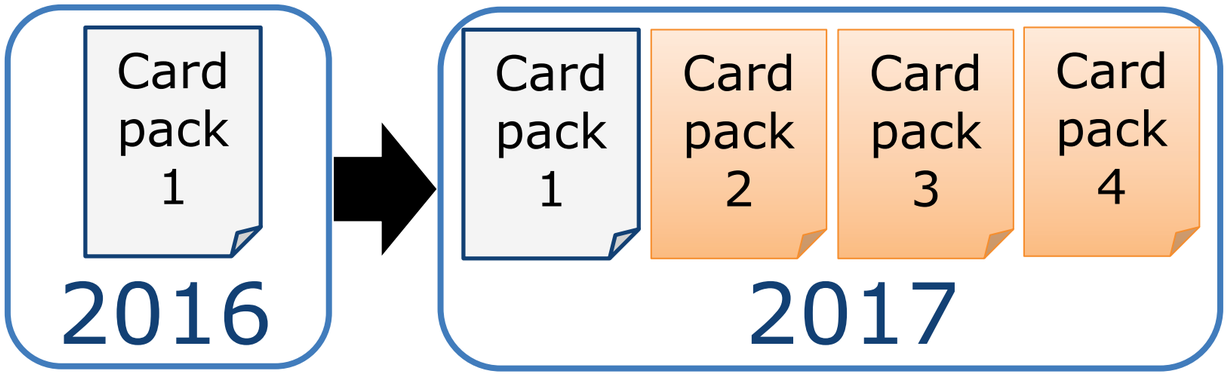
Game software is becoming more complex and difficult to be validated.

Incremental expansion is serious problem in QA



To extend the lifespan of a game title, it is **important to append new items constantly**.

However, it is very difficult to detect bugs when we introduce new card in an incremental way.



The above screenshot is excerpted from https://play.google.com/store/apps/details?id=com.cygames.Shadowverse

QA must grow as game growth

Combination Explosion in Games

Combination explosion is one of the most serious factors to increase the complexity of a game system. A combination of items highly expands the range of game strategies, but it also increases the verification cost drastically.

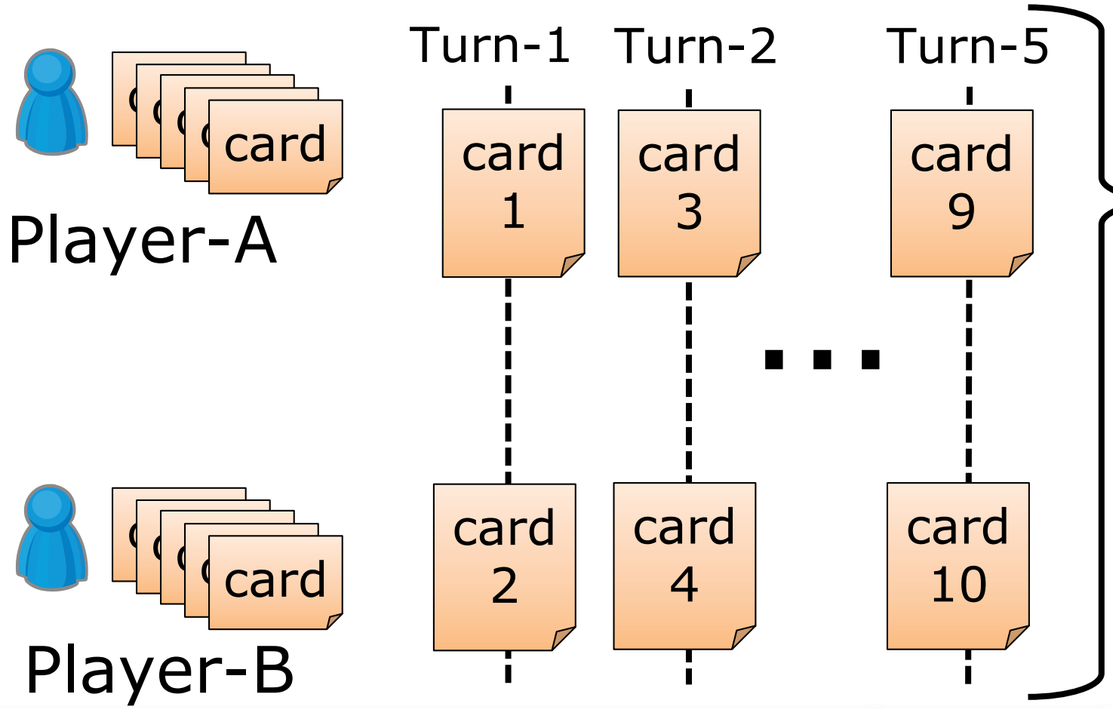
When the game consists of **1,000 types of card**, and its one play consists of **8 turns** in average, we have to verify **1,000^16 battle variations**.

items	Combinations when its play consists of 8 turns
100	100,000,000,000,000,000,000,000,000,000
1,000	10^48
10,000	10^64
100,000	10^80

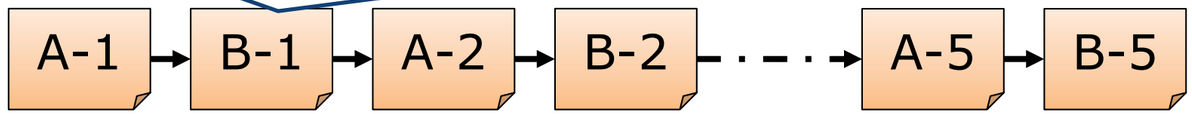
Even if the maximum number of items is limited to 100, it is very difficult to verify manually.

An Example of Combination Explosion in Games (1)

Consider a card game consisting of **ten types** of card (we assume the total amount of cards is infinite). **Two players** draw and play a card alternately in this game, and finish within **five turns**.



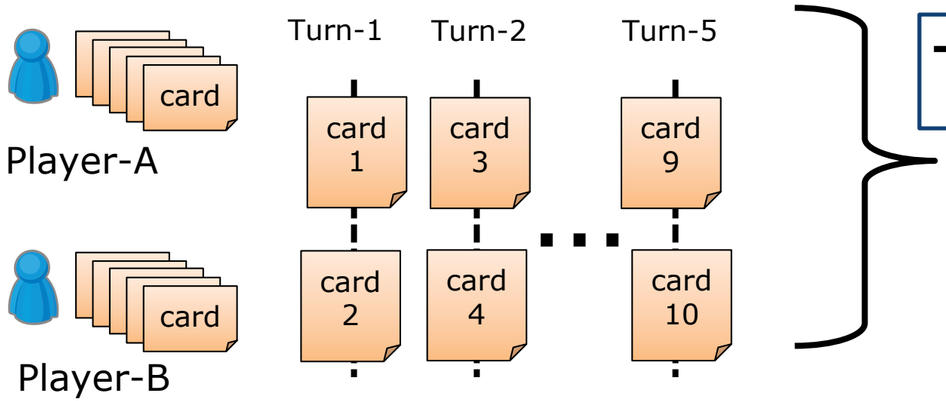
There is 10 possibilities for each card



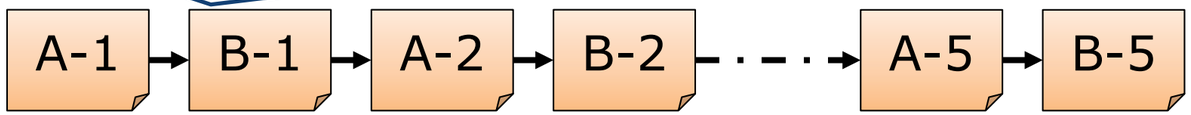
10^10 = 10,000,000,000
combination must be verified to guarantee that there is no bugs.

An Example of Combination Explosion in Games (2)

Consider a card game consisting of **a hundred types** of card. Two players draw and play a card alternately in this game, and finish within five turns.



There is **100 possibilities** for each card



100^10 = 100 quintillion combination must be verified to guarantee a good game balance. Manual QA is no longer applicable even if the number of card types is limited to 100.

Automation is required!

Man-Power-Driven QA

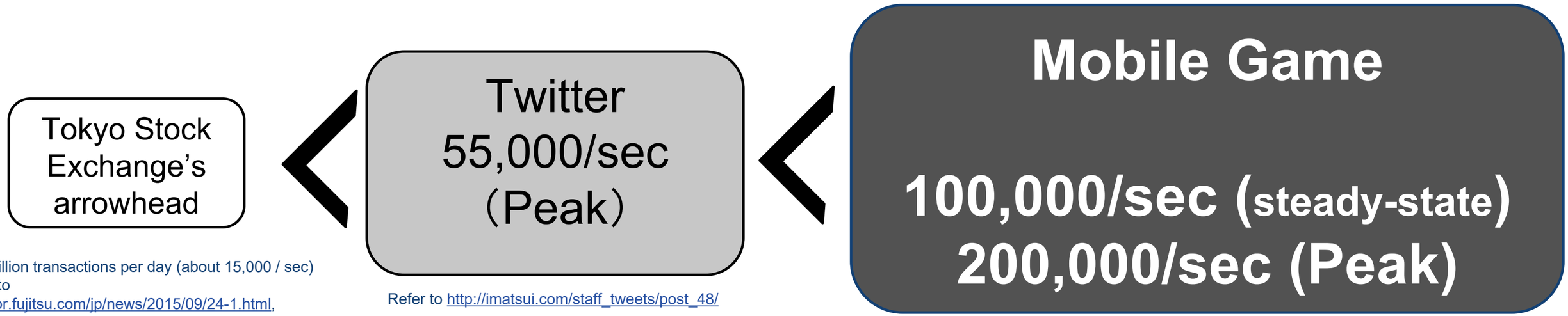
Automation!

AI-Driven QA in Cloud

Discovery and verification of bugs caused by combinations of items is very difficult with manual QA. Our mission is prioritizing the verification path by using AI that simulates humans' behavior.

Game logs are treasure for ML

Online game systems gather large-scale transactional logs, even in comparison with the conventional high-frequency trading system and SNS. Gaming industry could be a leader of ML utilization!



270 million transactions per day (about 15,000 / sec)
Refer to
http://pr.fujitsu.com/jp/news/2015/09/24-1.html,

Refer to http://imatsui.com/staff_tweets/post_48/

Our solution is “Similar-AI”

Similar-AI is an AI for simulating behaviors of ordinary people, rather than being stronger than people.

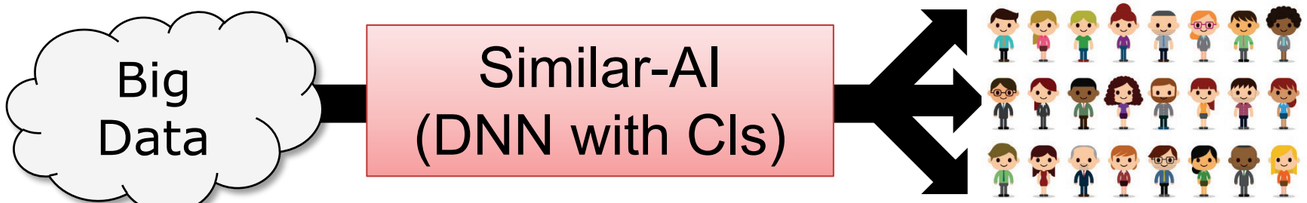
Differences between Similar-AI and conventional AI

AI bots, which are closely resembling humans, may discover bugs that users may encounter with high probability, from a vast amount of combinations.

Conventional AI (Strength and Accuracy)

A single AI that is stronger than humans (e.g. AlphaGo)

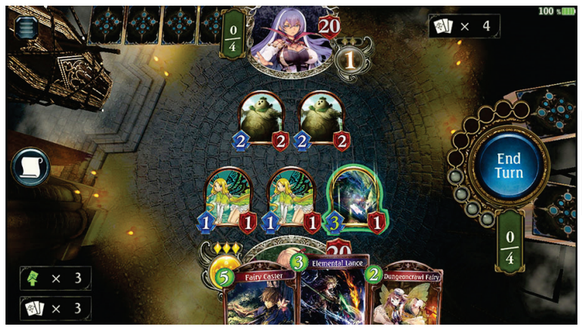
Similar-AI (Closeness with Humans)



Many "human-like AIs" including weak, middle-level, and strong users.

AI provides a good **approximate solution** to verifying combination explosion in games

A some kind of “extremely complex” function might be implemented



Game state

A human’s brain

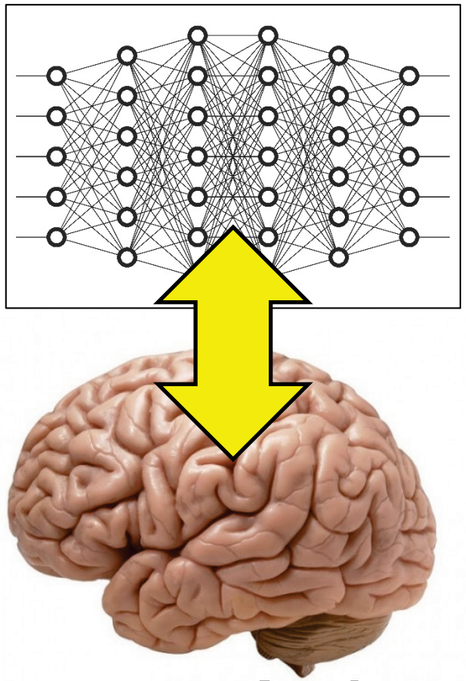
Next Action!

Digitize the human's thinking by using Machine Learning (deep learning) technologies!

Deep Neural Network

Available as gaming logs

Game state



A human's brain

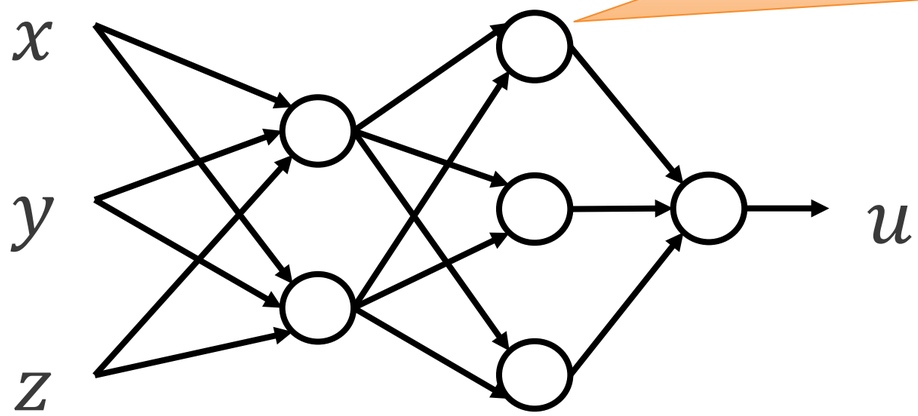
Available as gaming logs

Next Action!

Introduction to neural network

We can say NN is a method to generate an **"implicit" and "hidden" function** that associates the input data and the output data.

Each node has its own weight to calculate the output



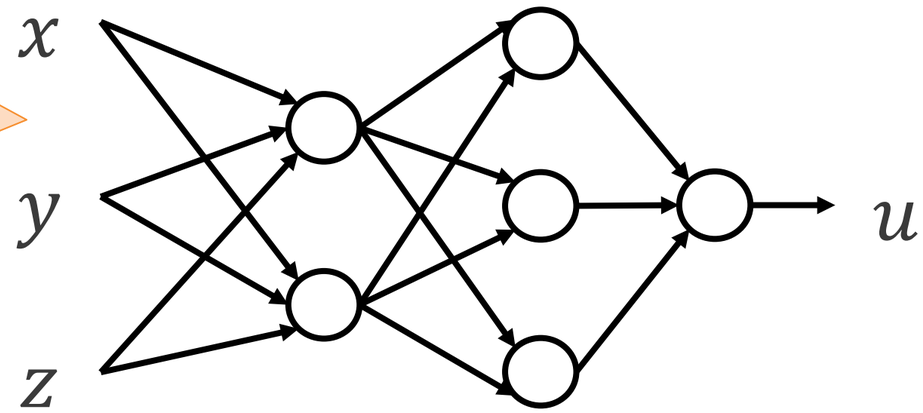
Input value

Output value

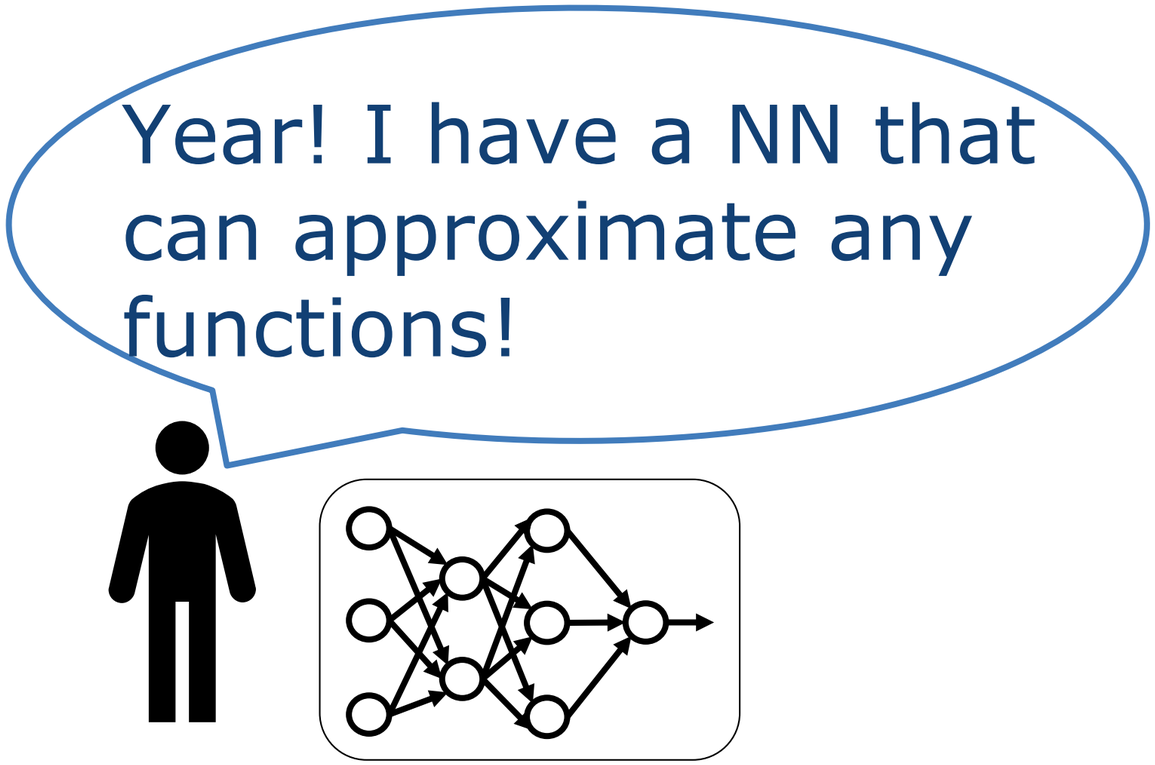
Multilayer Perceptron

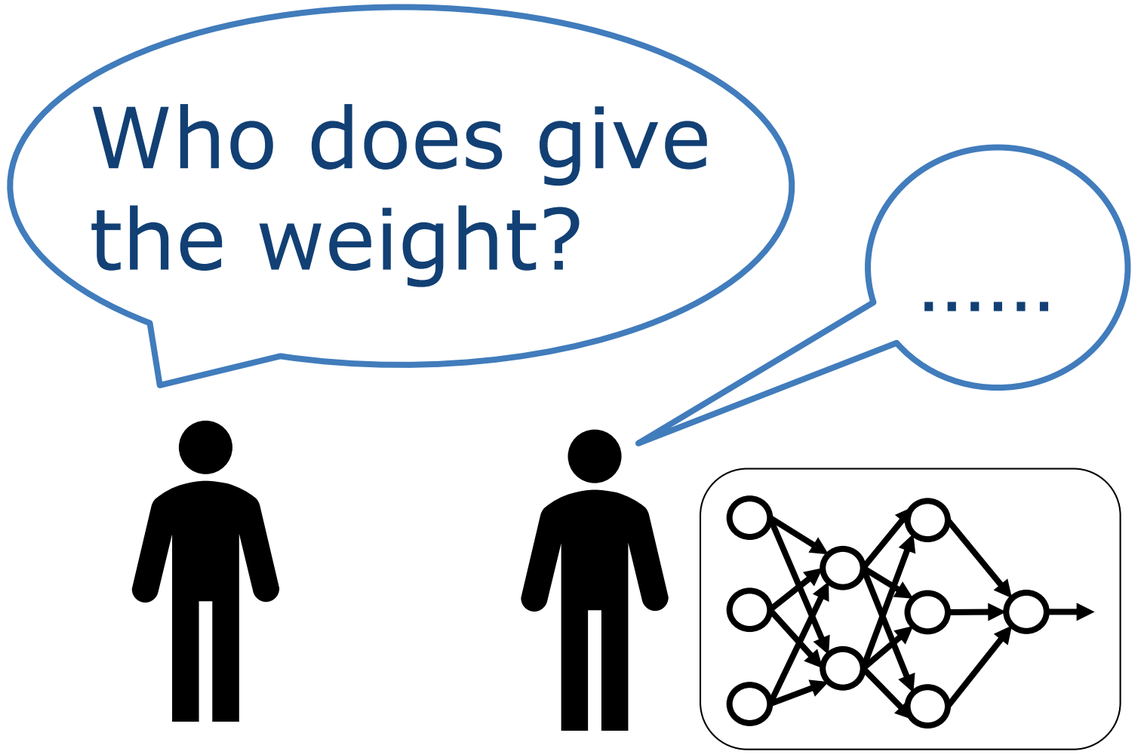
The most important feature of NN is that NN can represent an approximate function of any non-linear functions when NN has enough depth and dimensions.

This network structure (multilayer perceptron) is a universal structure for representing any functions.

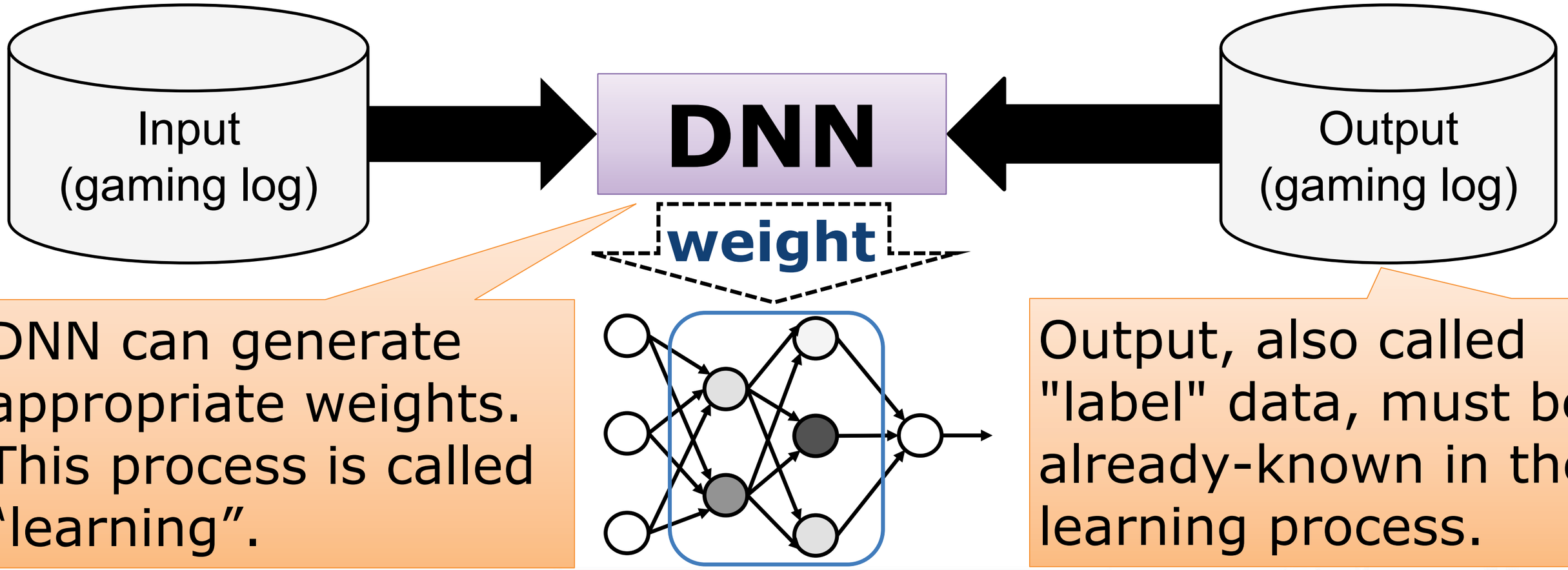


However, as we know “theoretical possibility” and “can do” is different.





Deep Neural Network provides methods to configure every weight automatically by exploiting big data



DNN can generate appropriate weights. This process is called “learning”.

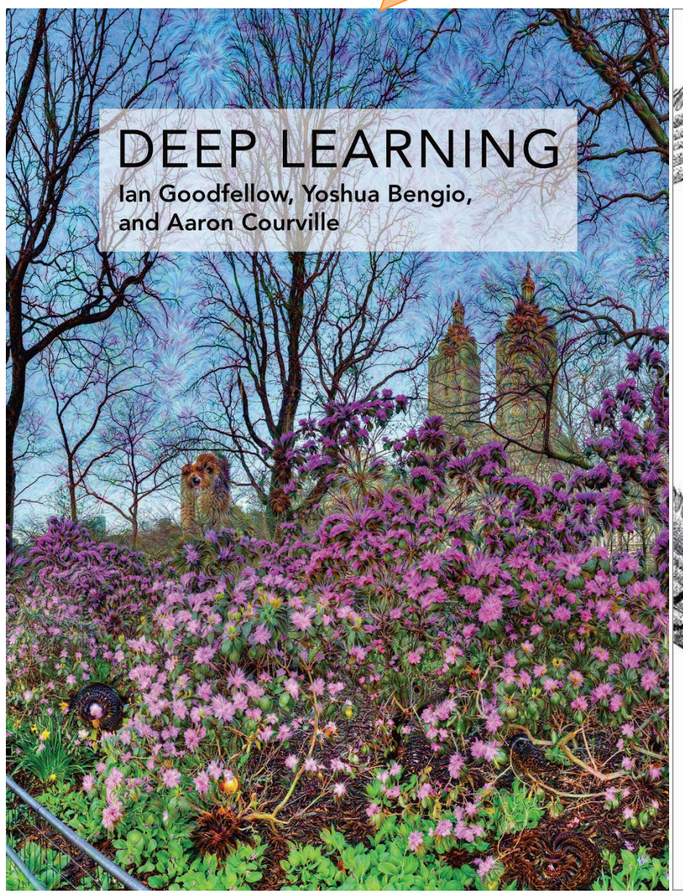
Output, also called "label" data, must be already-known in the learning process.

For more detail of DNN

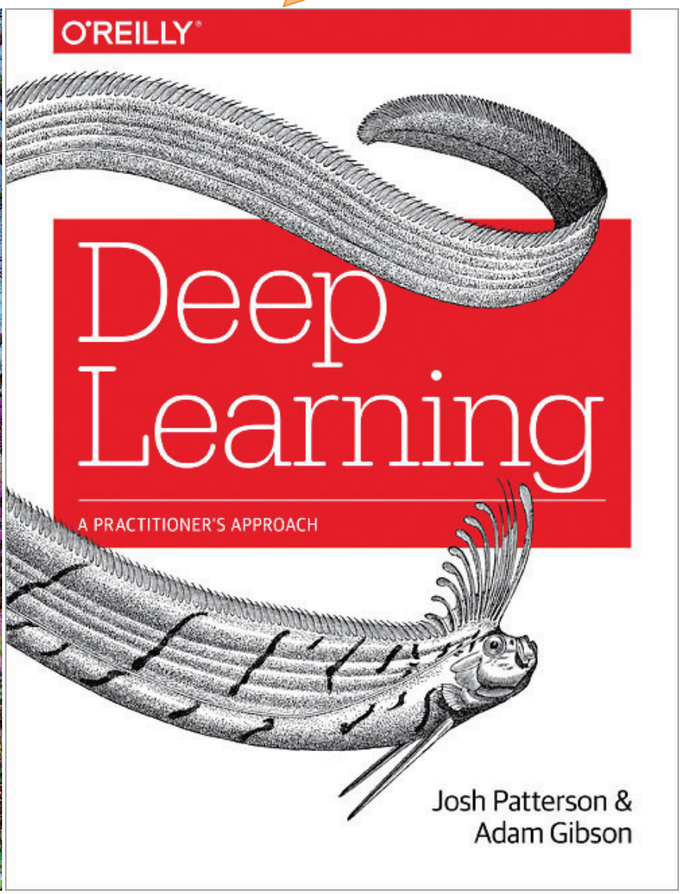
This talk intended to give you a very intuitive explanation of DNN. For deeper understanding of DNN, I recommend the following two books:

- “Deep Learning” by Ian Goodfellow, Yoshua Bengio and Aaron Courville, MIT Press.
- “Deep Learning - A Practitioner's Approach” by Adam Gibson and Josh Patterson, O’Reilly

theory



practice



Let's try your first DNN using Eclipse Deeplearning4j

As modern machine learning framework is highly matured, we can develop DNN apps **without deeply understanding its mathematics**.

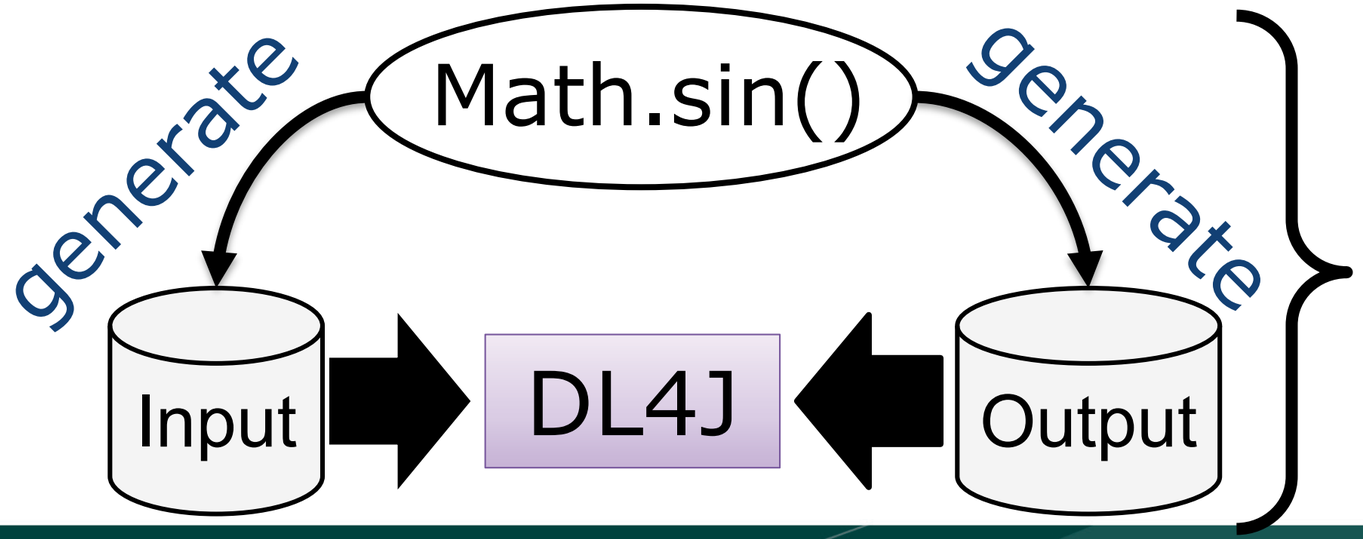
DL4J
deeplearning4j

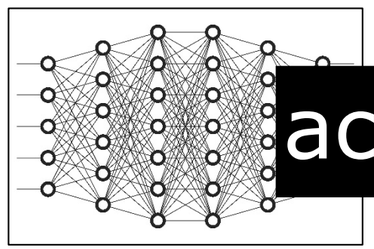
https://github.com/deeplearning4j/deeplearning4j

- ASF 2.0 Licensed. Java-based.
- High-level support of Parallelization including GPGPU
- Compatible with existing big data frameworks: Hadoop / YARN / Spark

Let's make DNN to learn sine function without math

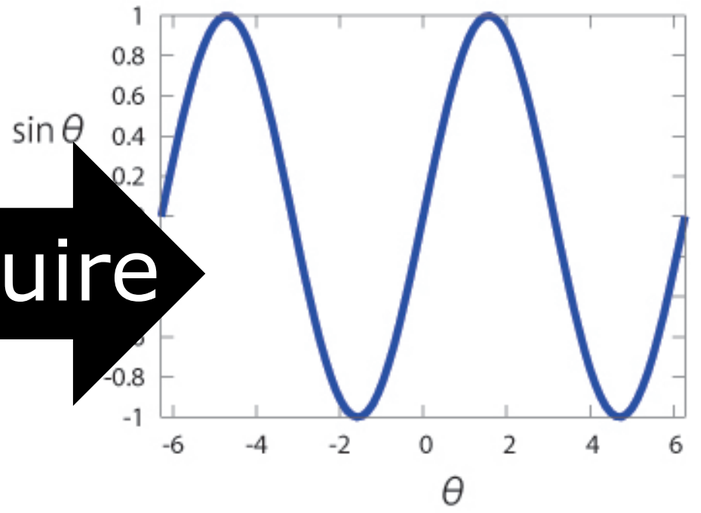
Basically, we need **a bunch of data** to make DNN learn something. But data preparation is awkward, so here we adopt **the sine function to generate big data virtually**. This example is based on the dl4j tutorial (https://github.com/deeplearning4j/dl4j-examples)





DNN

acquire



DNN Construction is very easy

Learning Params.

Network Structure

```
// Create the network
final int numHiddenNodes = 50;
final MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
    .seed(seed)
    .iterations(1) // Number of iterations per minibatch
    .optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)
    .learningRate(0.01) // Network learning rate
    .weightInit(WeightInit.XAVIER) // Gaussian distribution with mean 0, variance 2.0/(fanIn + fanOut)
    .updater(new Nesterovs(0.9)) // momentum for learning speed
    .list() // network layer list
    .layer(0, new DenseLayer.Builder().nIn(1).nOut(numHiddenNodes)
        .activation(Activation.TANH).build()) // hyperbolic tangent activation function
    .layer(1, new DenseLayer.Builder().nIn(numHiddenNodes).nOut(numHiddenNodes)
        .activation(Activation.TANH).build()) // hyperbolic tangent activation function
    .layer(2, new OutputLayer.Builder(LossFunctions.LossFunction.MSE) // MSE (mean squared error) loss function
        .activation(Activation.IDENTITY) // "identity" activation function is frequently a good choice, in conjunction
        with the MSE
        .nIn(numHiddenNodes).nOut(1).build())
    .pretrain(false).backprop(true).build();
final MultiLayerNetwork networkModel = new MultiLayerNetwork(conf);
networkModel.init();
```

This code is modified from https://github.com/deeplearning4j/dl4j-examples/blob/master/dl4j-examples/src/main/java/org/deeplearning4j/examples/feedforward/regression/RegressionMathFunctions.java

Important Parameters

```
// Create the network
final int numHiddenNodes = 50;
final MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
    .seed(seed)
    .iterations(1) // Number of iterations per minibatch
    .optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)
    .learningRate(0.01) // Network learning rate
    .weightInit(WeightInit.XAVIER) // Gaussian distribution with mean 0, variance 2.0/(fanIn + fanOut)
    .updater(new Nesterovs(0.9)) // momentum for learning speed
    .list() // network layer list
    .layer(0, new DenseLayer.Builder().nIn(1).nOut(numHiddenNodes)
        .activation(Activation.TANH).build()) // hyperbolic tangent activation function
    .layer(1, new DenseLayer.Builder().nIn(numHiddenNodes).nOut(numHiddenNodes)
        .activation(Activation.TANH).build()) // hyperbolic tangent activation function
    .layer(2, new OutputLayer.Builder(LossFunctions.LossFunction.MSE) // MSE (mean squared error) loss function
        .activation(Activation.IDENTITY) // "identity" activation function is frequently a good choice, in conjunction
        with the MSE
        .nIn(numHiddenNodes).nOut(1).build())
    .pretrain(false).backprop(true).build();
final MultiLayerNetwork networkModel = new MultiLayerNetwork(conf);
networkModel.init();
```

Important Parameters

- **Size of the hidden layer**: empirically-derived rules-of-thumb, usually between the size of the input and size of the output layers.
- **Learning rate**: If you increase the learning rate too much the result will diverge, and if you leave it too small you will lose convergence.
- **Optimization algorithm**: choose the appropriate one by referring the papers. Sebastian Ruder's survey paper summarized the latest optimization algorithms.

Sebastian Ruder, "An overview of gradient descent optimization algorithms," arXiv:1609.04747, 2017.

Network Structure (rarely changed)

```
// Create the network
final int numHiddenNodes = 50
final MultiLayerConfiguration
    .seed(seed)
    .iterations(1) // Number
    .optimizationAlgo(Optimiz
    .learningRate(0.01) // Ne
    .weightInit(WeightInit.XAVIER) // Gaussian distribution with mean 0, variance 2.0/(fanIn + fanOut)
    .updater(new Nesterovs(0.9)) // momentum for learning speed
    .list() // network layer list
    .layer(0, new DenseLayer.Builder().nIn(1).nOut(numHiddenNodes)
        .activation(Activation.TANH).build()) // hyperbolic tangent activation function
    .layer(1, new DenseLayer.Builder().nIn(numHiddenNodes).nOut(numHiddenNodes)
        .activation(Activation.TANH).build()) // hyperbolic tangent activation function
    .layer(2, new OutputLayer.Builder(LossFunctions.LossFunction.MSE) // MSE (mean squared error) loss function
        .activation(Activation.IDENTITY) // "identity" activation function is frequently a good choice, in conjunction
        with the MSE
        .nIn(numHiddenNodes).nOut(1).build())
    .pretrain(false).backprop(true).build();
final MultiLayerNetwork networkModel = new MultiLayerNetwork(conf);
networkModel.init();
```

Many network structures were proposed in both academia and industory. But **one hidden layer is sufficient** for the large majority of problems.

Generating the training data

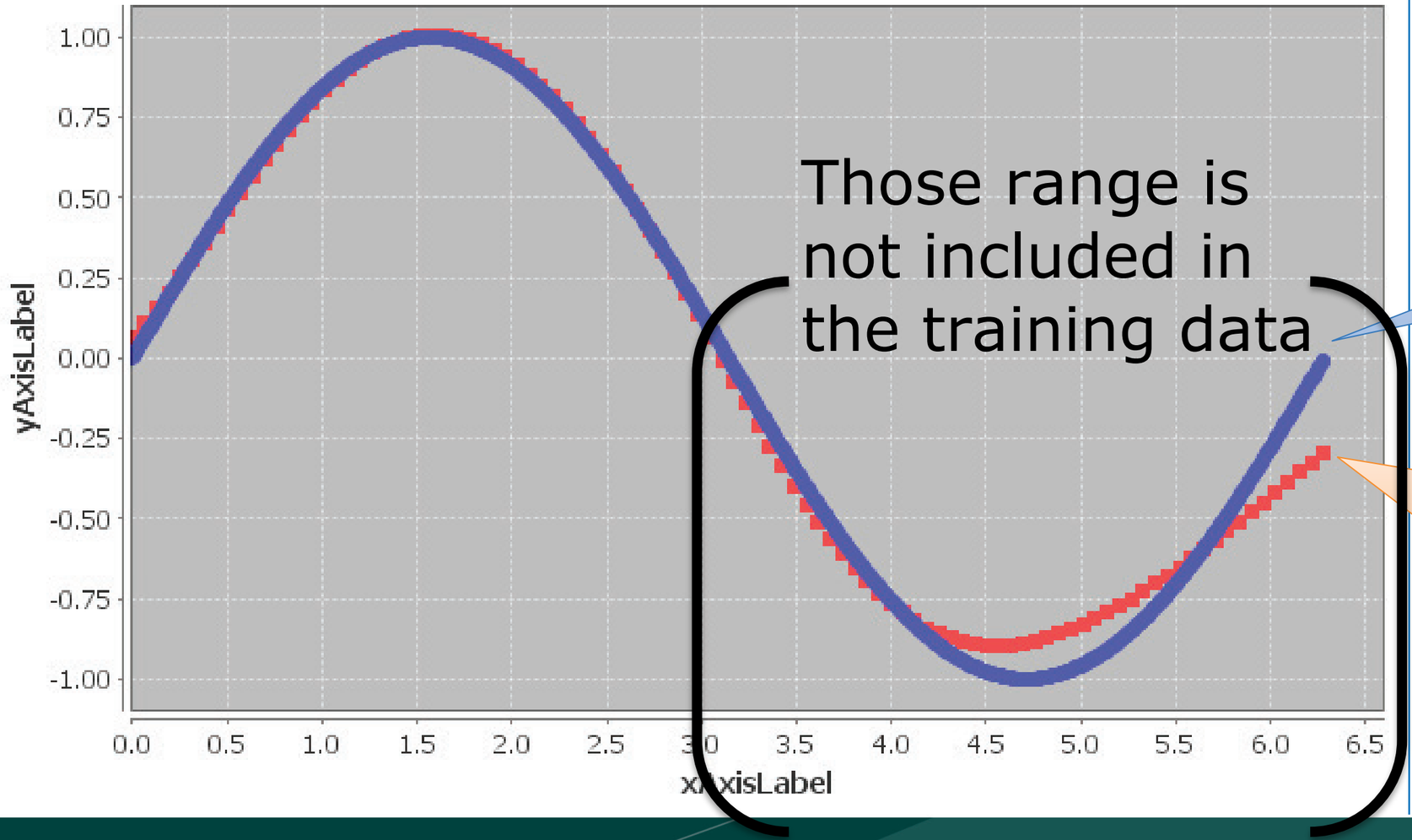
Nd4J.linspace returns evenly spaced numbers over a specified interval. This case returns 1000 values from -3.14 to +3.14.

```
// Generate the training data
final MathFunction sinMathFunction = new SinMathFunction();
final int nSamples = 1000; // Number of data points
final INDArray xSamples = Nd4j.linspace(-FastMath.PI, FastMath.PI, nSamples).reshape(nSamples, 1);
final int seed = 12345;
final int batchSize = 100;
// batchSize: i.e., each epoch has nSamples/batchSize parameter updates
final DataSetIterator dataSetIterator = getTrainingData(xSamples, sinMathFunction, batchSize, new Random(seed));
```

```
    private static DataSetIterator getTrainingData(final INDArray x, final MathFunction function, final int batchSize, final
    Random rng) {
        final INDArray y = function.getFunctionValues(x);
        final DataSet allData = new DataSet(x,y);

        final List<DataSet> list = allData.asList();
        Collections.shuffle(list, rng);
        return new ListDataSetIterator(list, batchSize);
    }
```

DNN learned sin() without any mathematical knowledge



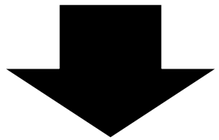
(blue) Math.sin() curve

(red) Predicted value by DNN

Which framework should we choose?

- **Apache MXNet**
- **Caffe**
- **Chainer**
- **Deeplearning4j**
- **Keras**
- **MS Cognitive Toolkit**
- **PyTorch**
- **TensorFlow**

There are considerable amount of frameworks because the fundamental mathematical model of DNN is simple.



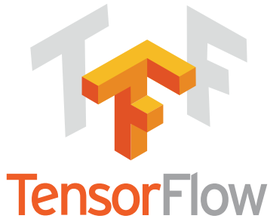
We have to choose an appropriate framework.

Which framework should we choose when...?

1. You are a researcher on DL itself
   → TensorFlow and PyTorch
2. You are students who are learning DL
   → Keras, and then TensorFlow
3. You work on Natural Language Processing
   → PyTorch
4. You work in industry
   → TensorFlow
5. You have to get results within this week
   → Keras on TensorFlow on Google Cloud ML Engine

TensorFlow and PyTouch

Both frameworks can utilize GPU for accelerated learning. The most significant difference is their execution model: “Define-and-Run” adopted by TensorFlow and “Define-by-Run” adopted by PyTorch.



- GPGPU
- Python
- Very popular
- Many add-ons
- **Define-and-Run**

- GPGPU
- Python
- Very popular
- Many add-ons
- **Define-by-Run**

Define-and-Run and Define-by-Run

Under the define-and-run framework, you would define computational graph structure and then executes the graph by giving the data, whereas the define-by-run framework generates the graph structure fully at runtime.

- **Define-and-Run**
- Good at **parallelization**
- Good at **distribution**

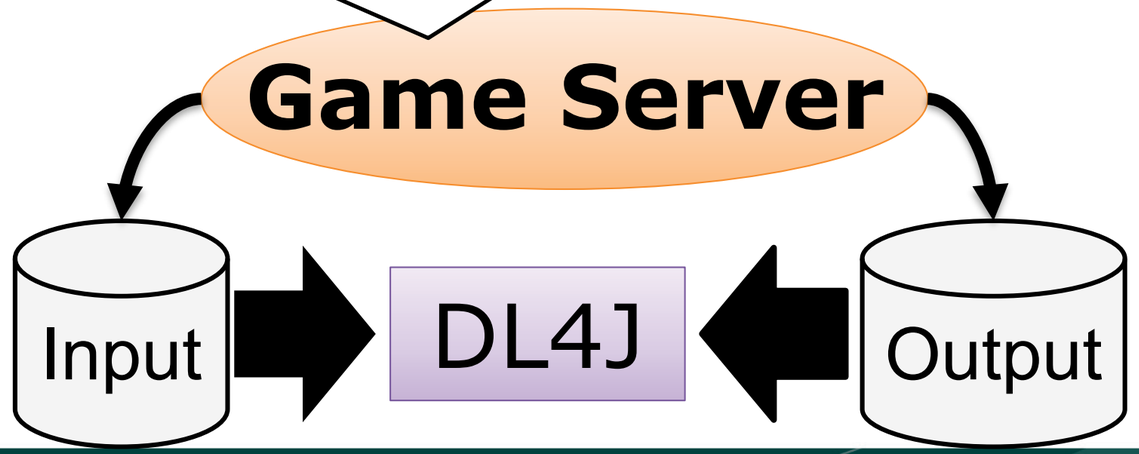
- Difficult to support dynamic graphs

- **Define-by-Run**
- Good at fast iteration
- Good at dealing with NLP and video sequence by **supporting dynamic graphs**

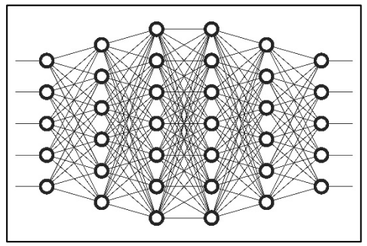
Next Step: Applying DNN to Gaming Log

We saw that 3-layered DNN works very fine. Next step is to make DNN acquire the humans' behavior from the gaming log. The key technique is data conversion.

We use the game server instead of Math.sin()

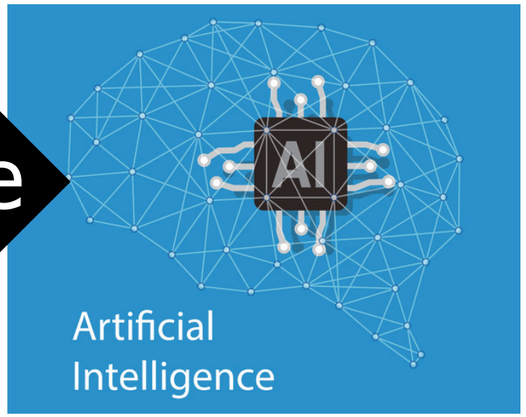


DNN learns relationships between the game state and the next state.



DNN

acquire



Data Conversion Case Study: Online Collectible Card Game

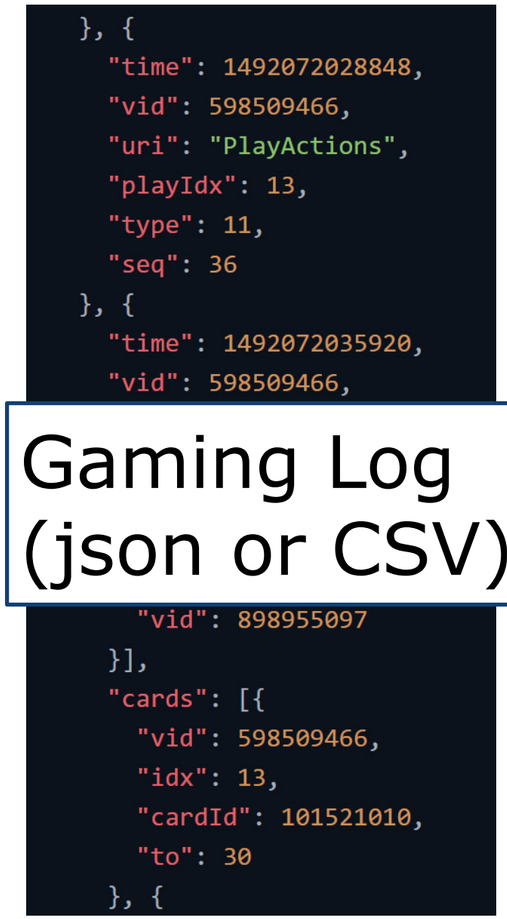
Opponent player

A stage where several cards are being played.

Player and his/her deck.

The above screenshot is excerpted from
https://play.google.com/store/apps/details?id=com.cygames.Shadowverse

Typical Gaming Log cannot be learned directly



Gaming Log (json or CSV)

Player-A's action-A1

Player-B's action-B1

Player-A's action-A2

Player-B's action-B2

⋮

Player-B's action-B*n*

DNN cannot learn such sequential logs directly

Converting the gaming log into tensor data model

Player-A's action-A1

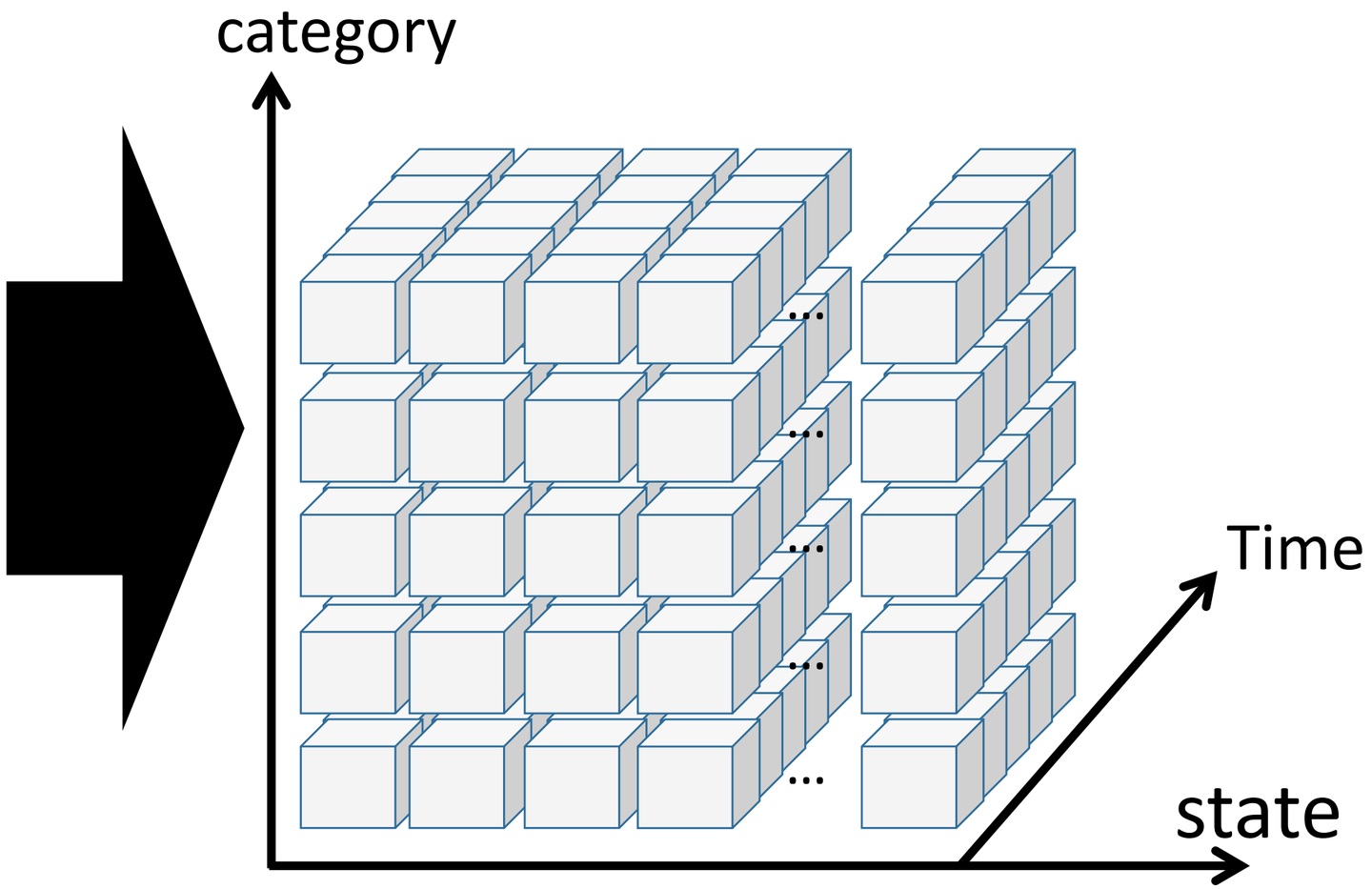
Player-B's action-B1

Player-A's action-A2

Player-B's action-B2

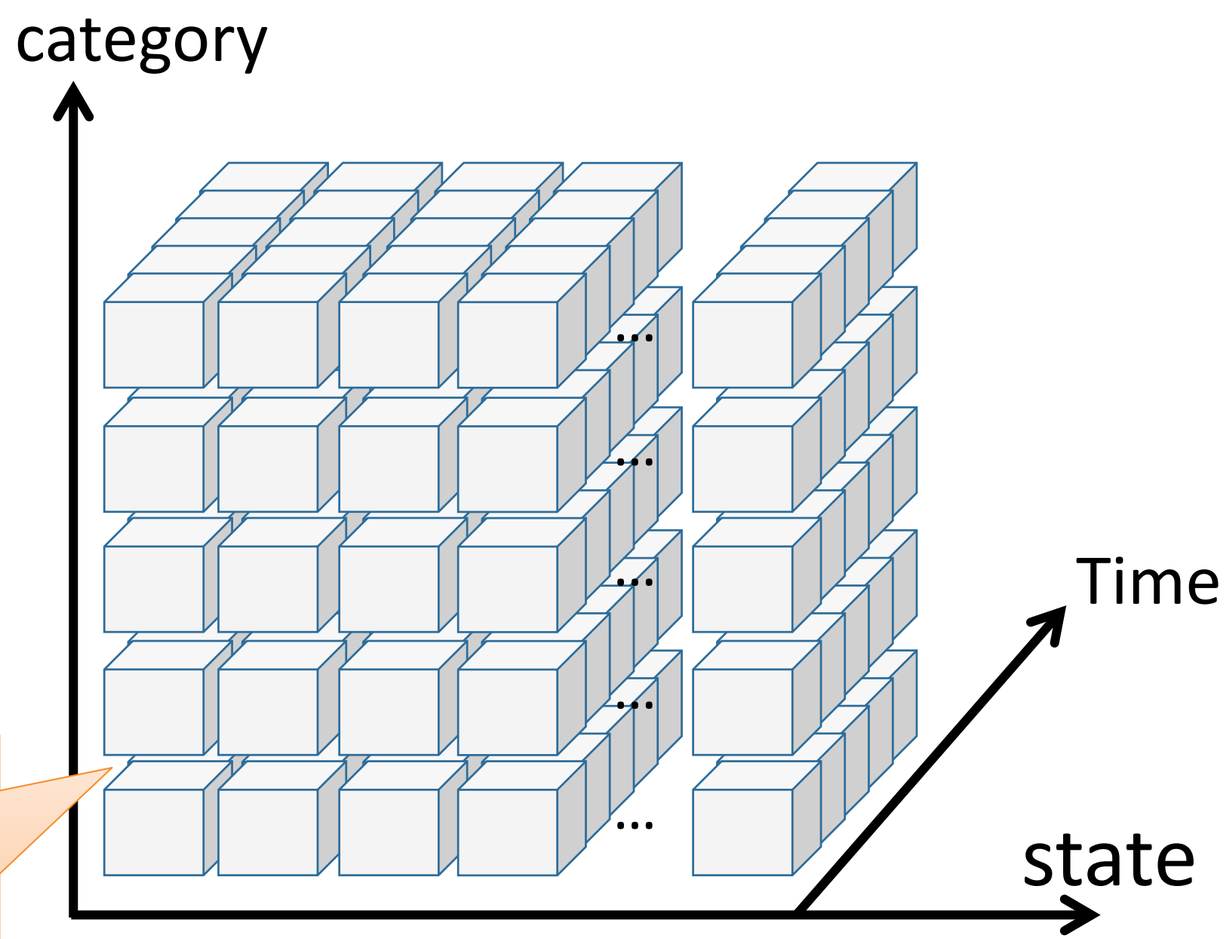
⋮

Player-B's action-B*n*



A tensor is n-dimensional array, intuitively.

- float v // scala
- float[] v // vector
- float[][] v // matrix
- float[][][] v // tensor
- float[][][][] v // tensor
- float[][][][][] v // tensor



Each block represents a matrix. Thus tensor looks like a matrix of matrix.

Convert the gaming log into the tensor data model

Turn-1

	$card_1$	$card_2$	…	$card_n$
stage	1	0	…	1
deck	0	1	…	1

One matrix represents the battle stage status at a specific timepoint. “1” means the card exists, and “0” means the card does not exist.

action

Turn-2

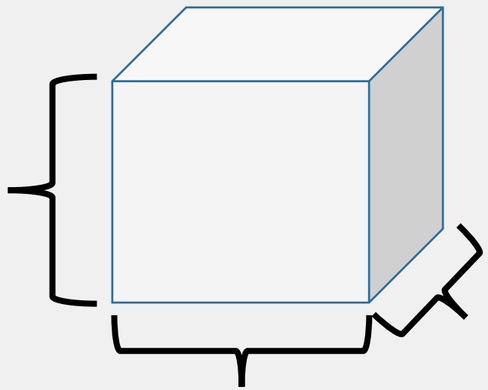
	$card_1$	$card_2$	…	$card_n$
stage	1	1	…	1
deck	0	0	…	1

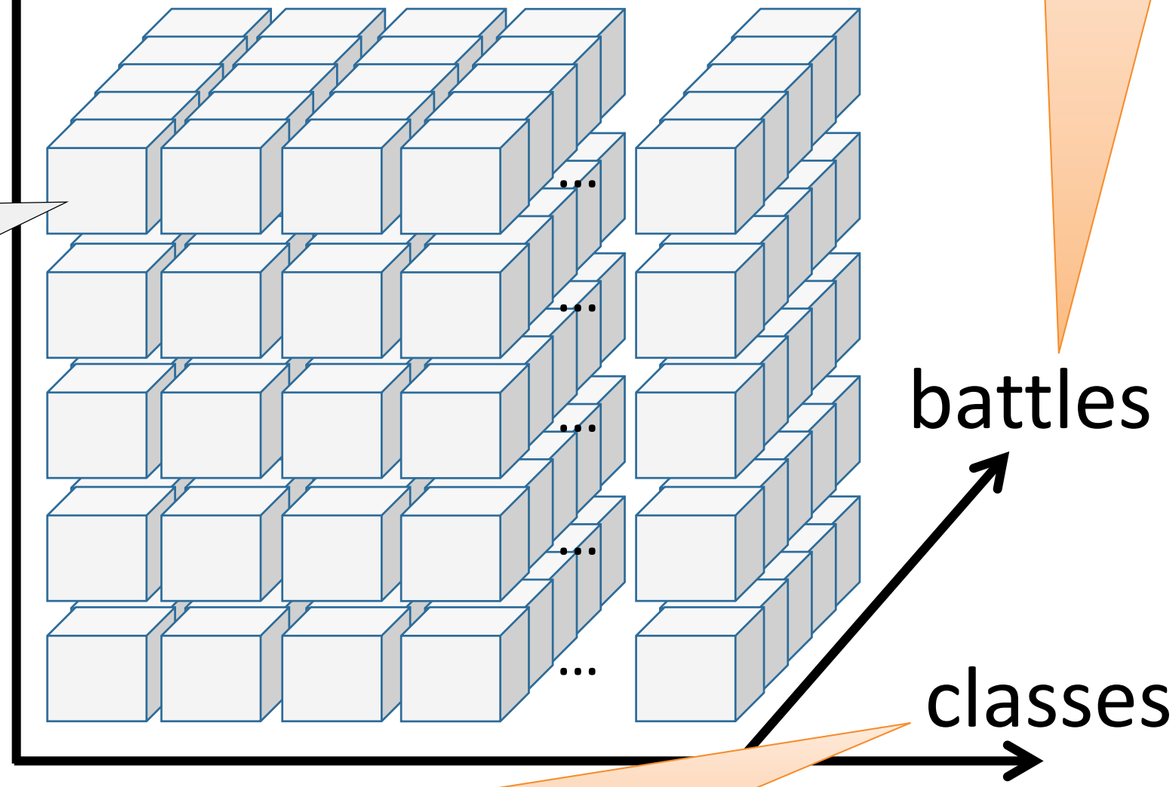
When a user plays “card2”, it moves from the deck to the stage.

The gaming log of online collectible card video game could be transformed 2 by n (n= the number of cards) by k (k= the number of turns) tensor data model.

Tensor Data Model for representing Gaming Log

A cube corresponds to one battle





Now, the gaming long can be learned by DNN!

Picking up the most important axis from the game system

In typical online collectible card games, users can choose his/her hero (avatar), and the types of hero, called "class" highly affect the strategy.

Thus, we adopt the class as an **independent axis of the tensor data model** for classifying the gaming log.

The above screenshot is excerpted from https://shadowverse-portal.com/

Advanced Techniques for Converting Complex Gaming Log such as Real-Time Games rather than Turn-Based

- The simple tensor data model has a difficulty to represent a **sequential relationship between actions**.

- We introduce a concept of **"N-gram" for representing connection strength between every action**. This n-gram is a well-known method in the field of natural language processing (NLP)

N-gram: Representing "connection strength" as a vector

Input String

A B C D A B

Extracting Grams

A B C D A B

Slide window on each iteration

B C D A B

A C D A B

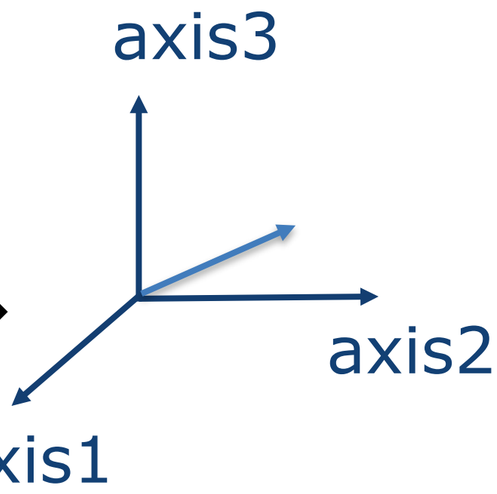
A B D A B

A B C A B

Count-up Grams

Gram	Freq.
AB	2
BC	1
CD	1
DA	1

Create N-gram Vector



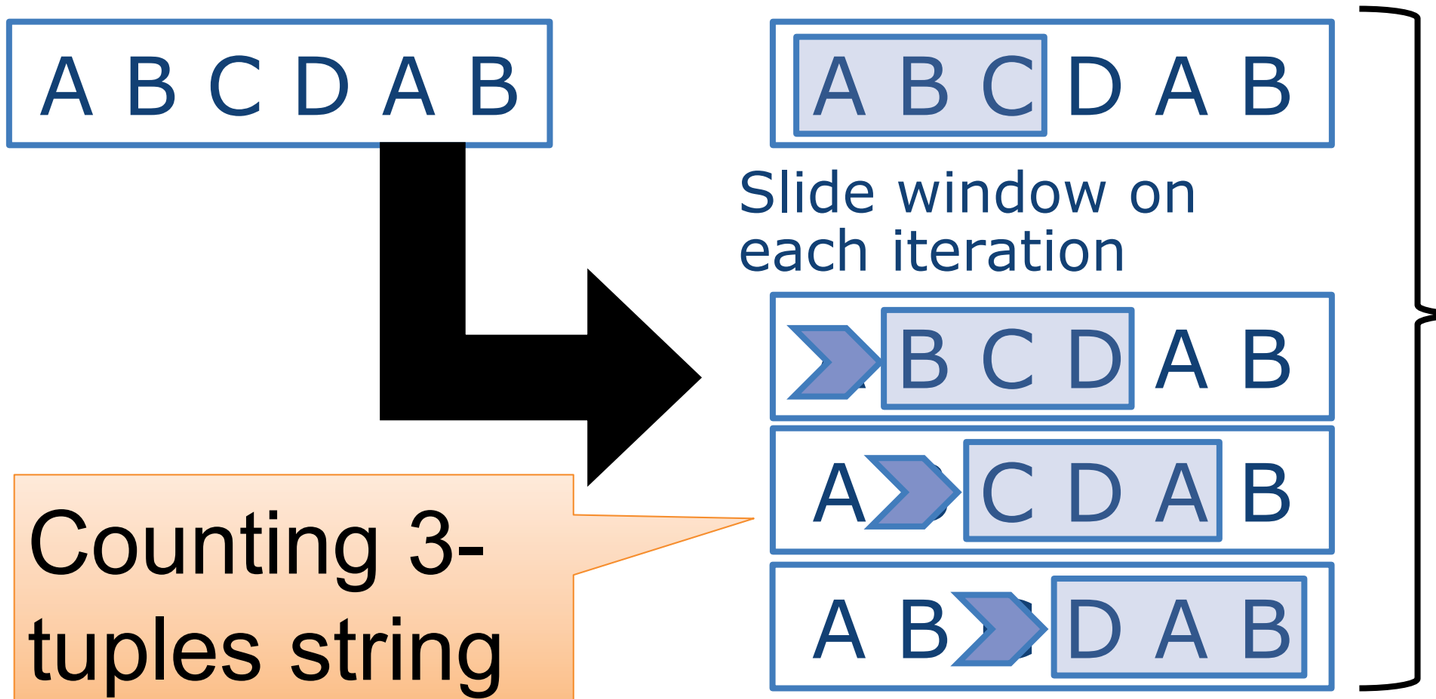
Connection strength of each element is measured as frequency of them.

tri-gram: Representing 3 factor connection strength

Input String

Extracting Grams

Count-up Grams



Gram	Freq.
ABC	1
BCD	1
CDA	1
DAB	1

Counting 3-tuples string

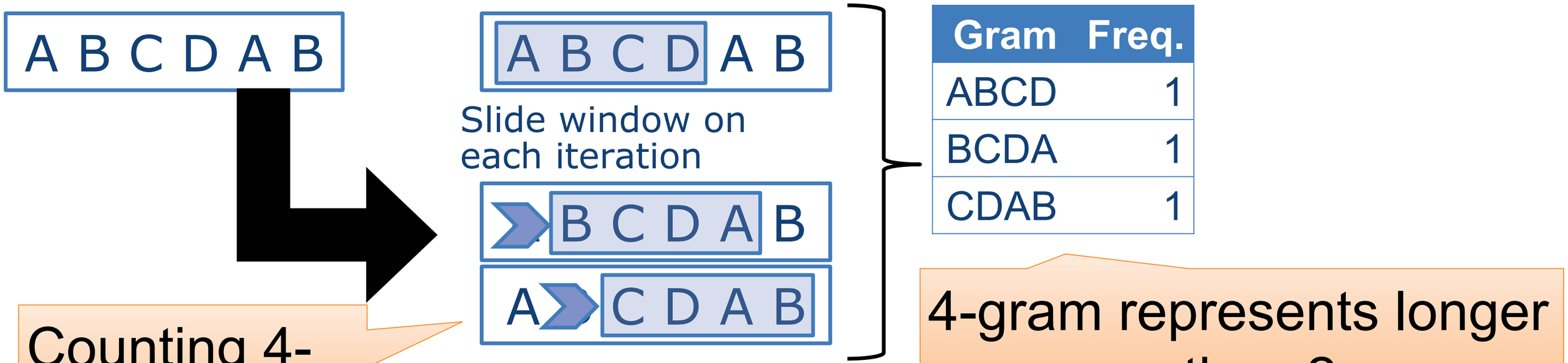
3-gram represents longer sequence than 2-gram

4-gram: Representing 4 factor connection strength

Input String

Extracting Grams

Count-up Grams



Gram	Freq.
ABCD	1
BCDA	1
CDAB	1

Counting 4-tuples string

4-gram represents longer sequence than 2-gram

Summary of Data Conversion

- Sequential gaming log can be converted to the tensor data model **without losing any game semantics**.
- We need some scalable data processing framework such as **Apache Hadoop** and **Apache Spark** in order to convert all data.
- Tensor data is very sparse but deeplearning4j's **ND4J** classes provide efficient method to store them.

The Headless App Framework using Linux containers

- We have developed the DNN module, the data conversion module. Now, we can predict user's behavior by using DNN.

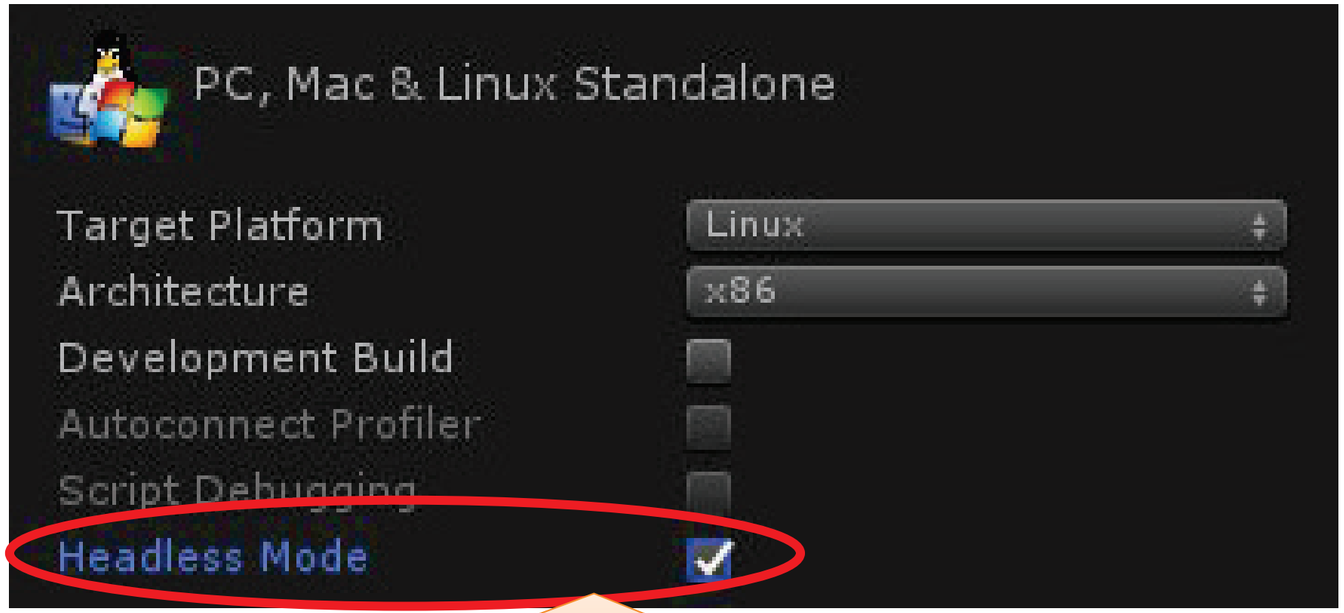
- Our aim is to detect "bugs" of actual app binary, thus, the AI-bot must play an actual game.

- We utilized a headless mode of the existing game engines and Linux containers for wrapping it.

Headless Game App

- We deployed a **headless game app**, which **shares the same code with production game**, into AWS EC2 infrastructure.
- Modern game engines such as Unity support “headless mode” for running on the server-side.

Unity Build Option



Making the app “headless” is straightforward

System Architecture

DNN server is executed on GPGPU-enhanced AWS instance

node.js is suitable because it adopts single thread asynchronous I/O model.

Deep Neural Network Server
(prediction engine)

DL4J server (GPGPU)

Headless Game App
Management Server

node.js

express

REST API

WebSocket

Container pool

docker

Autopiloting by node.js

The number of computing nodes can be increased as much as needed because the system was designed as a shared-nothing system.

Headless unity app instances are under control of Docker

Headless Unity

Headless Unity

...

Headless Unity

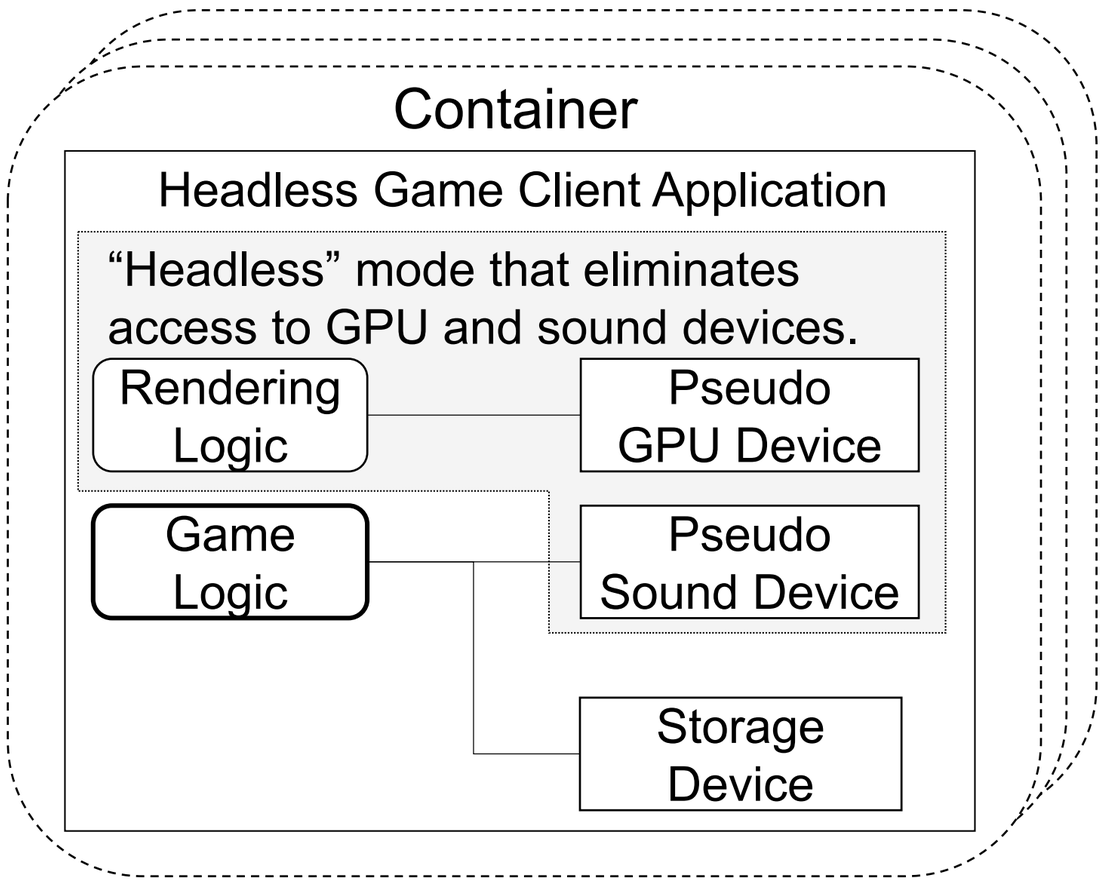
Headless apps are managed by Docker the virtualization environment.

Headless + Containerization

Making game apps able to run transparently in cloud servers requires the game app to support two functionalities:

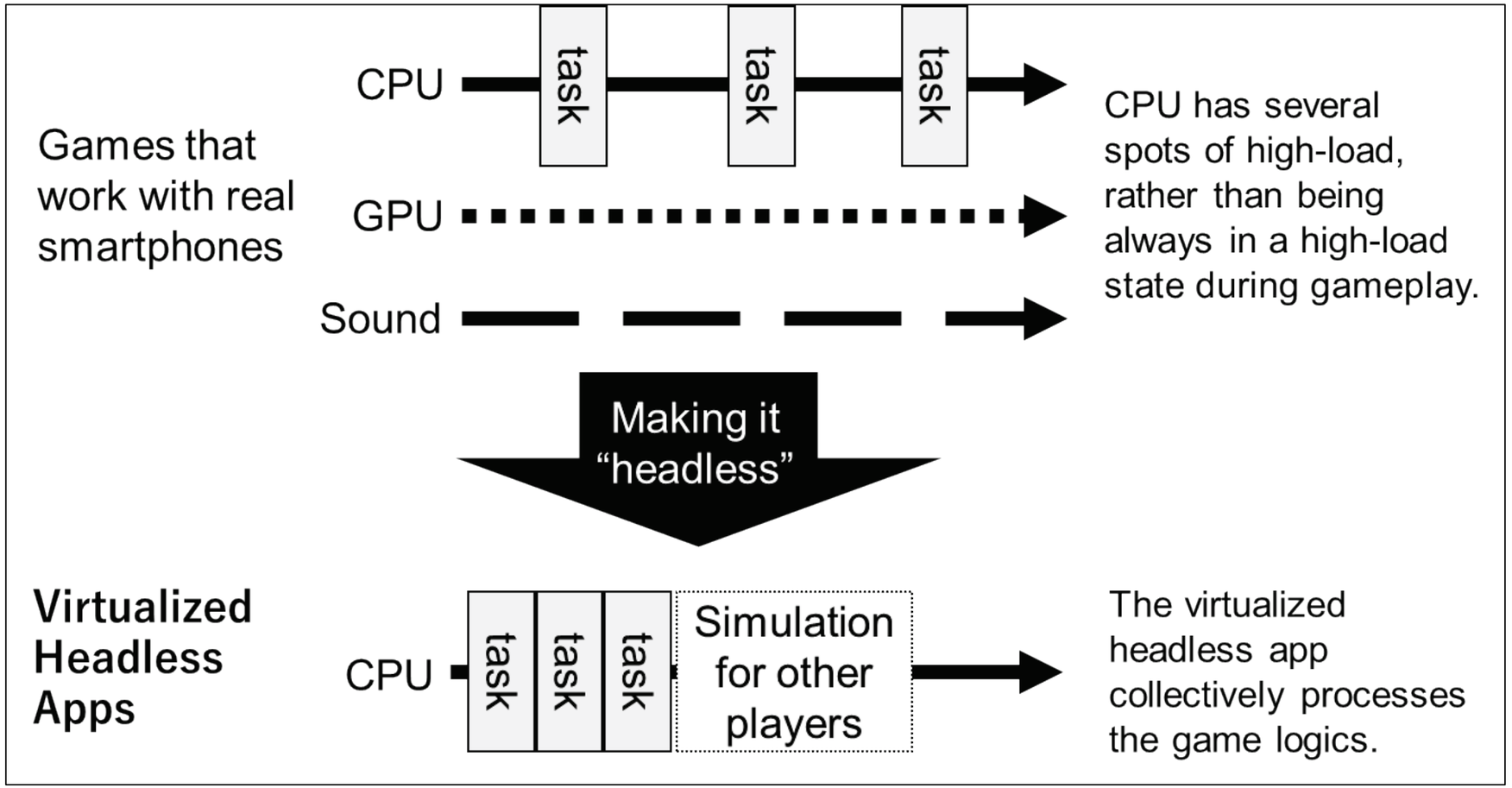
(1) a headless mode that eliminates GPU and sound access

(2) app isolation for simultaneously running multiple instances of apps in a single server node.



A headless mode for eliminating GPU and sound

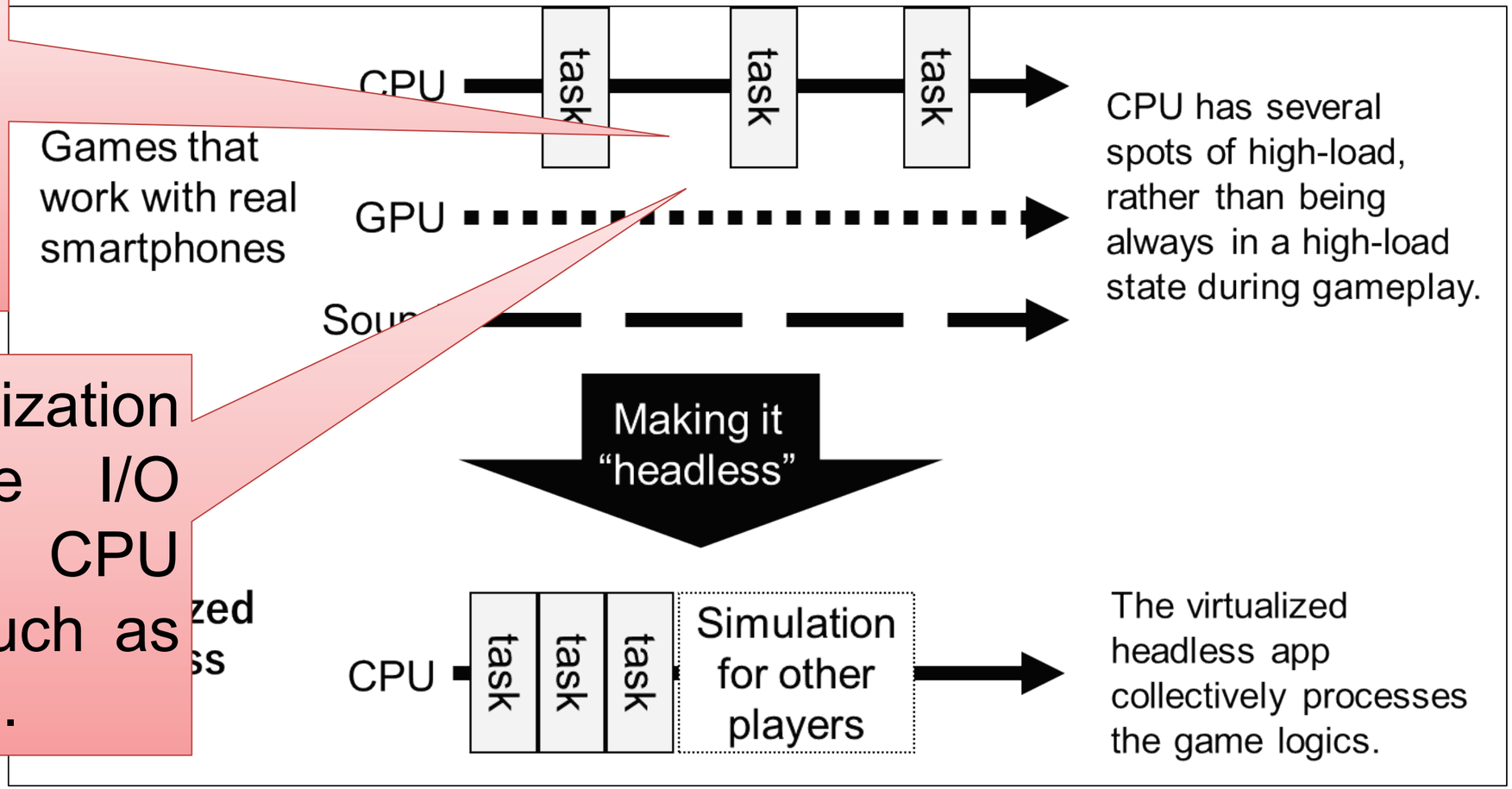
A frame rate is too slow to maximize the server utilization, because animation and sound playing processes are designed to be viewed by a user, who is living in a real world slower than the CPU.



A headless mode for eliminating GPU and sound

The game system must insert several waits to align the timing to the graphics and sound.

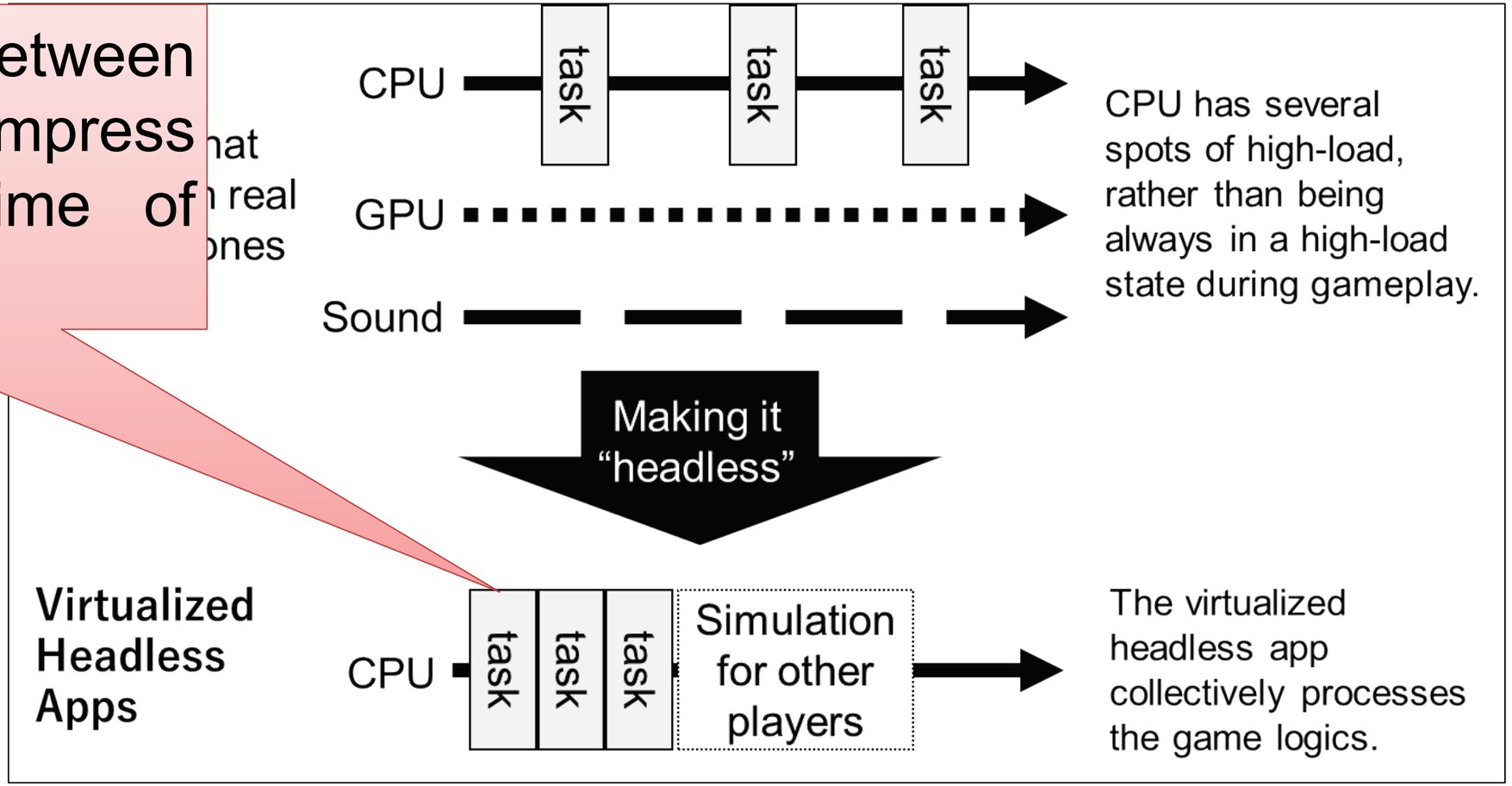
There are synchronization costs related to the I/O processing between a CPU and external devices such as GPU and sound devices.



AI can play “no-wait game”

By eliminating wait between every frame, we can compress the total execution time of single play.

AI can play such extremely fast-speed game because AI can recognize each frame status.



Performance Study

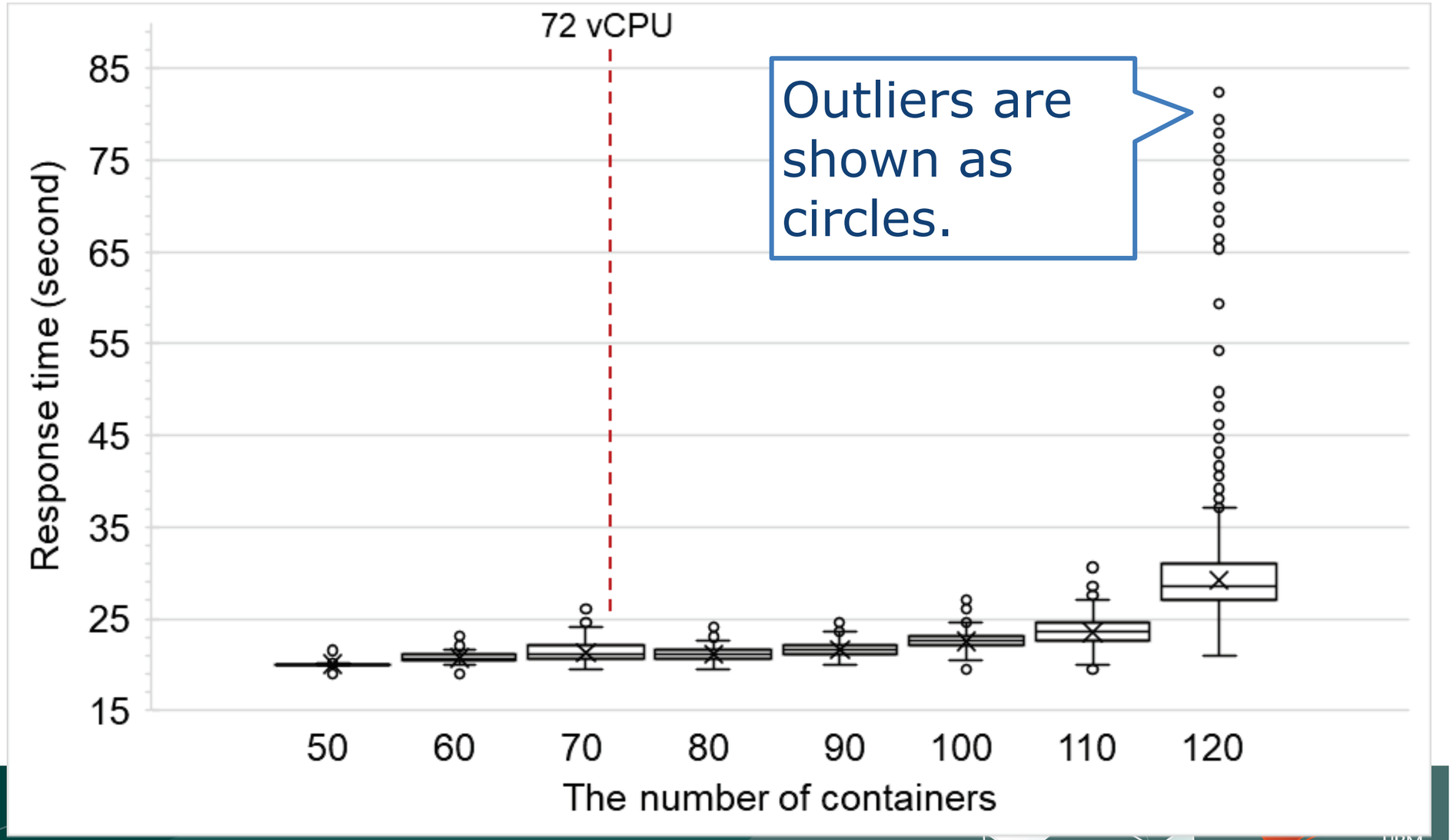
- Experiments were conducted to evaluate the real-world scalability and response speed of the implemented system.
- The aim was to clarify the tradeoff relationship between the parallelism and response speed of the headless app under containers running in a single server.
- The host is Amazon EC2 C5 instance c5.18xlarge equipped with 72 logical CPU cores and 144 GiB memory. It equips two processors (Intel Xeon Platinum 8124M @ 3.00 GHz), and each CPU provides 18 cores. Owing to Intel's hyper-threading technology, each core can execute two logical threads in parallel.

How to measure the performance

- A set of parameters was prepared for simulating a battle scene in the game that takes about 30 s when launched on a real smartphone that renders the graphics.
- An experimental program was implemented that sent the simulation parameters over 60 min and measured the response time of each simulation.
- Eight experiments were conducted with 50, 60, 70, 80, 90, 100, 110, and 120 containers.

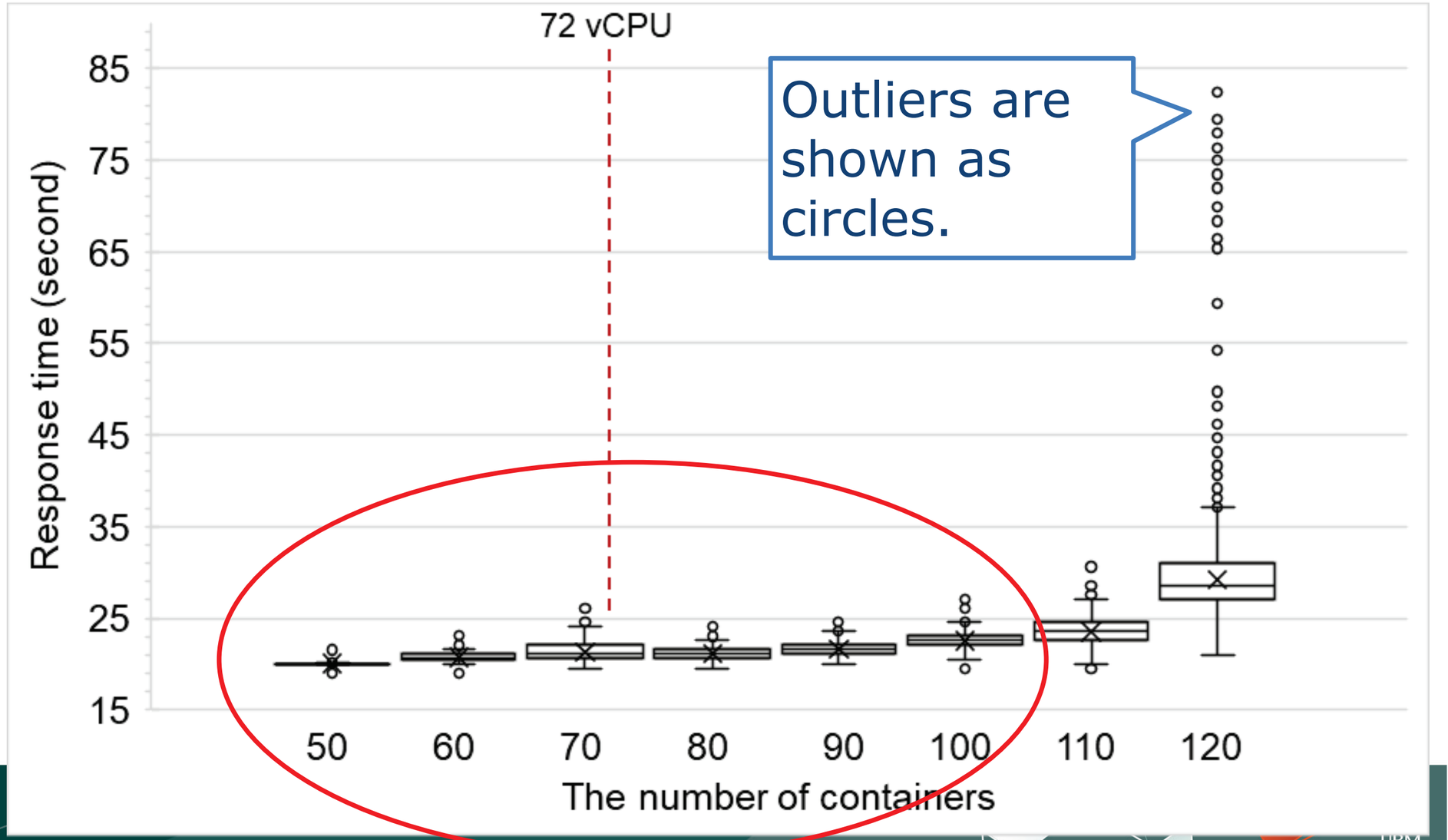
Performance Study: Parallelism in a single server

This chart shows the eight response time tendencies of the implemented system hosting the headless game app with respect to the number of containers.



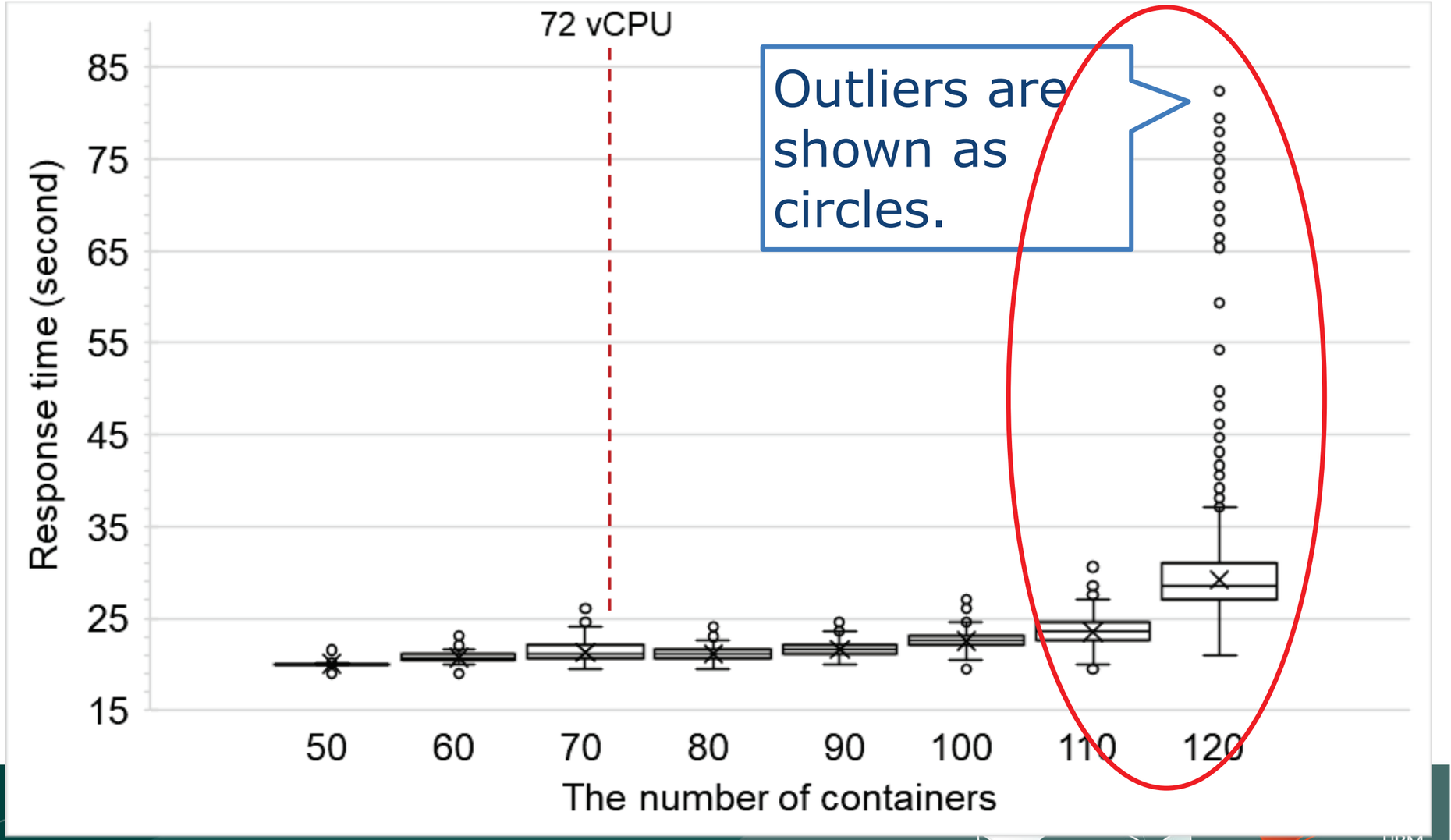
Performance Study: Parallelism in a single server

Up to 100 containers, the proposed system achieved a stable response speed.



Performance Study: Parallelism in a single server

With 120 containers, 667 outliers were generated that occupied 4.6% of the 14,306 simulation processes in total. Thus, setting up 100 containers is suitable for practical application.



Analysis

- The experiment gave evidence that **the system can activate containers 1.5 times the number of CPU cores** to achieve good response and good throughput performances.

- It showed that the system effectively uses a many-core server by parallelizing the existing game app in headless mode.

Summary

- In our game services, over multi-billion data gathers in a quarter. By utilizing the existing data processing frameworks and DNN framework, we could utilize those data for achieving automatic QA.
- Core technique is the design of the tensor data model. We had to carefully design data conversion model from gaming log to the tensor.
- This automatic QA is fully-automated, thus it is highly effective to find bugs by running it 24 hours 365 days.

Future Work

- We plan to extend Similar-AI to support “simulation of user's behavior” for verifying how users deal new items when we add those new elements in a game.

- Because Similar-AI is something like “mirror” of real users, we can use it as a strong market investigation and survey tool.

Tips to Success in Developing AI-driven QA

- Design tensor data model carefully.
  - This is human's responsibility.
- Use well-established network model.
  - Stand on the shoulders of giants.
- Update frequently.
  - Fresh data is required to simulate users.

Conclusion Remarks

- DNN frameworks are highly matured, and now we can develop an AI-driven QA framework, by implementing data conversion method from a noisy gaming log into a tensor data model.

- The existing game engines support “headless” mode for executing game apps in the server-side for scalable and automatic bug detection.

- The industry daily obtains a bunch of data from players, which are valuable to realize AI-based automatic QA.