# Massive Parallel Computing in Cloud and kakao.

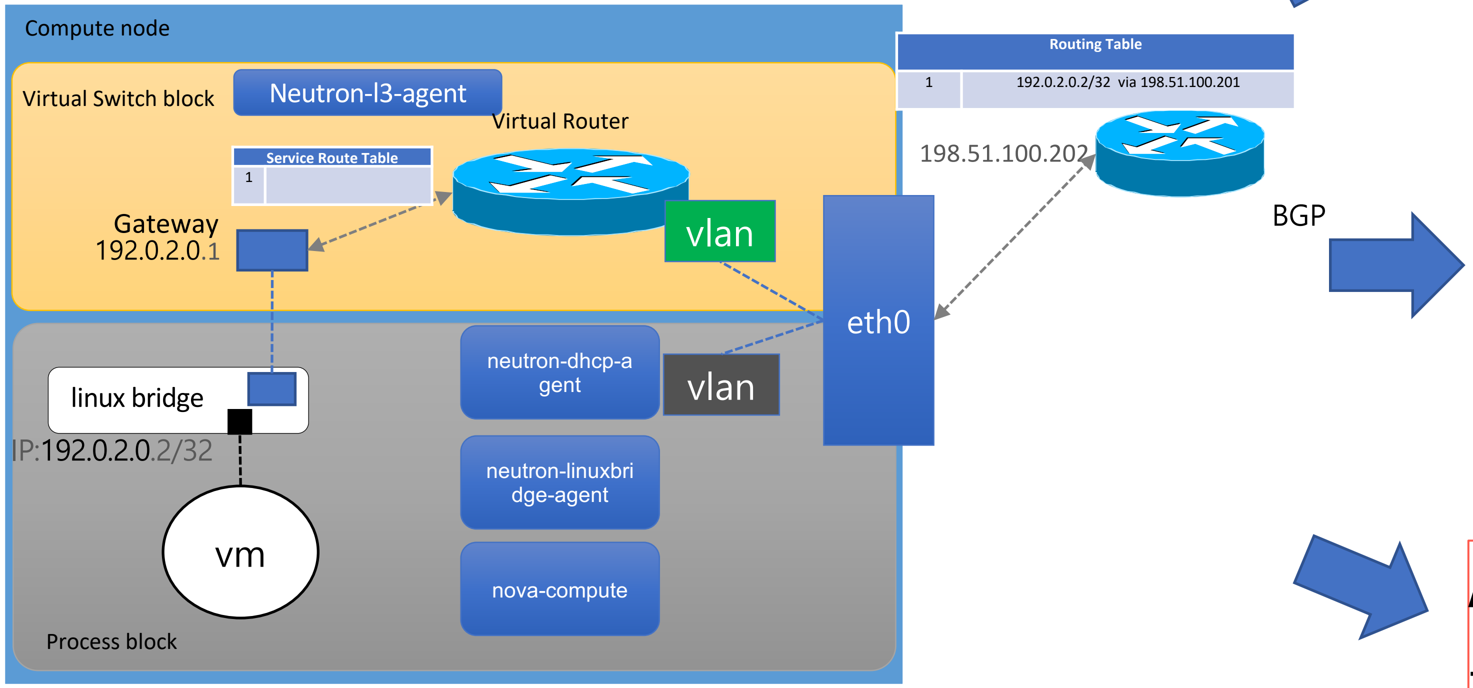
공용준 Andrew.Kong
카카오

# 00 Before, Massive Computing

# Improvement in kakao cloud

**/32bit subnet network**

**BGP + NAMESPACE**




**Cloud Load Balancer:**
+ ECMP + ARP Proxy

**Cloud Tenant Network:**
+ Multi Protocol Label Switching

**Accelerated Cloud:**
+ SR-IOV, + XDP/eBPF

# Improvement in kakao cloud

KVM

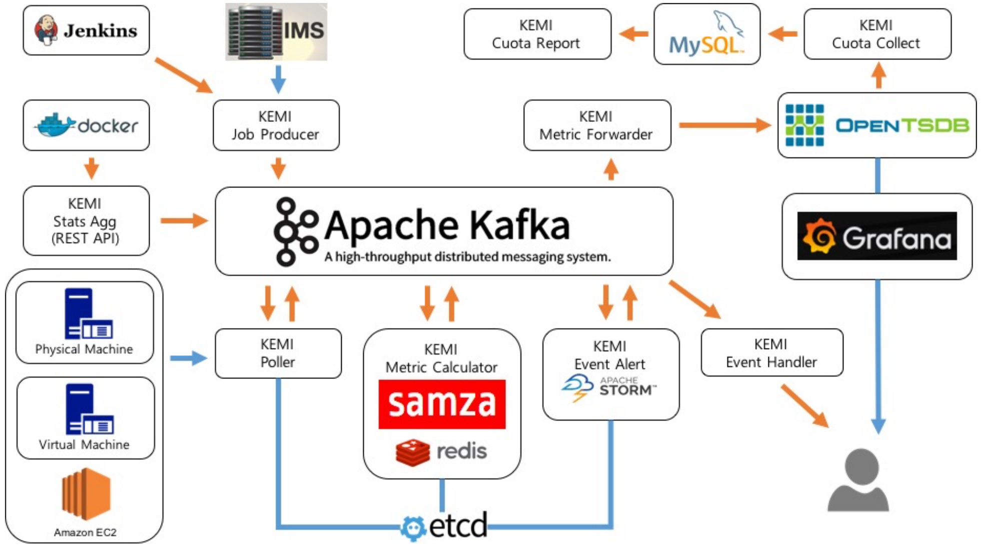
MESOSPHERE

DC/OS

# Improvement in kakao cloud

## KEMI:

### Kakao Event Metering monItoring




## KOCOON:

### KakaO COntainer based service mONitoring

KOCOON

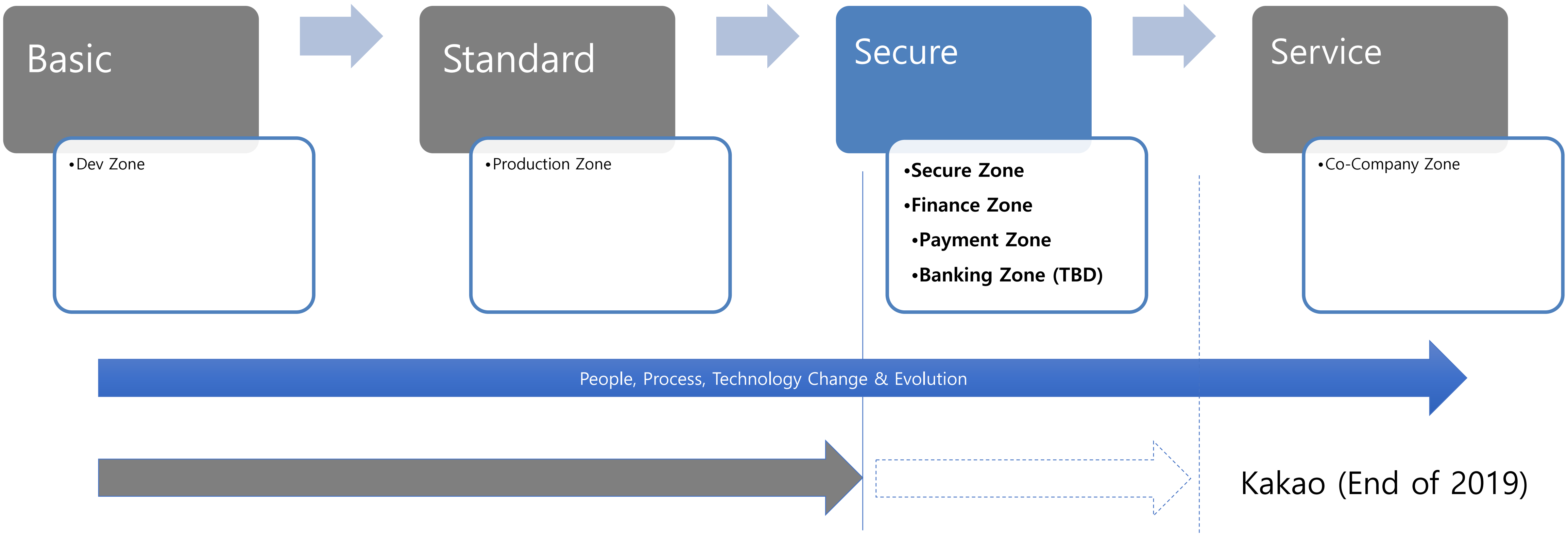
# Improvement in kakao cloud



| CMM0 | CMM1 | CMM2 | CMM3 | CMM4 |
|---|---|---|---|---|
| Legacy | self service Dev resource | limited Prod resources | Automated Cloud | Integrated Service Platform |
| output: cloudTF | output: KRANE (openstack cloud) | output: KEMI (MaaS) | output: DKOS (CaaS) | output: 9RUM (C.N.P) |

# A lot of progress in kakao cloud

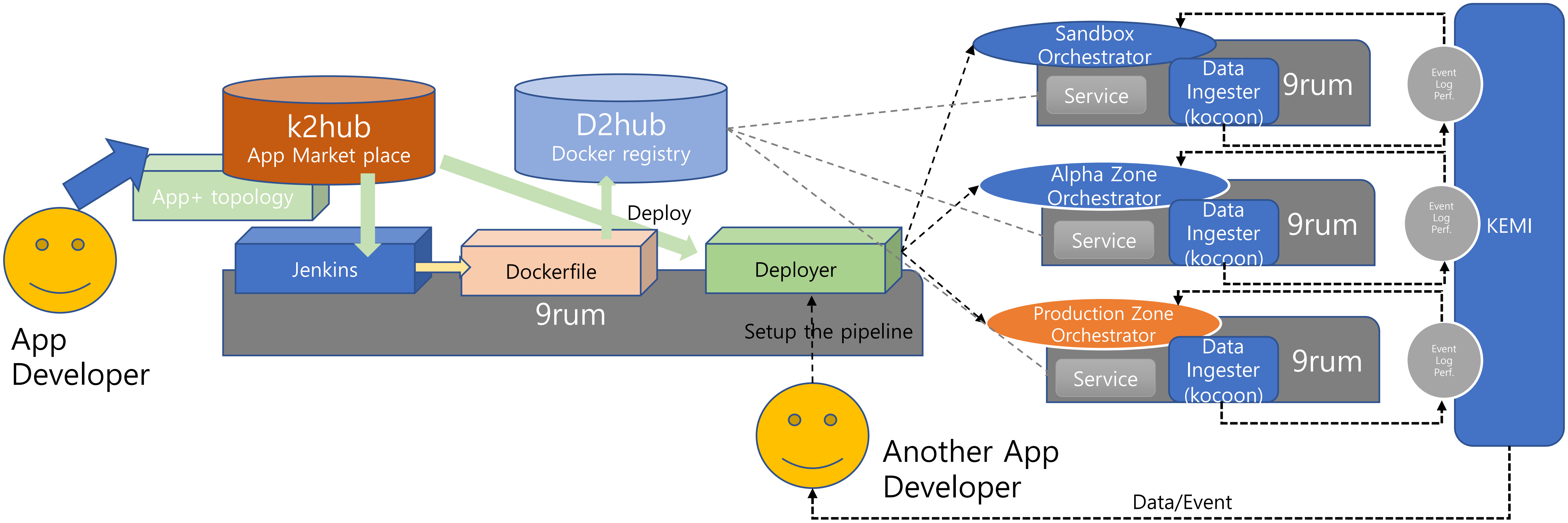




# A lot of progress in kakao cloud

# Changes in kakao  with cloud

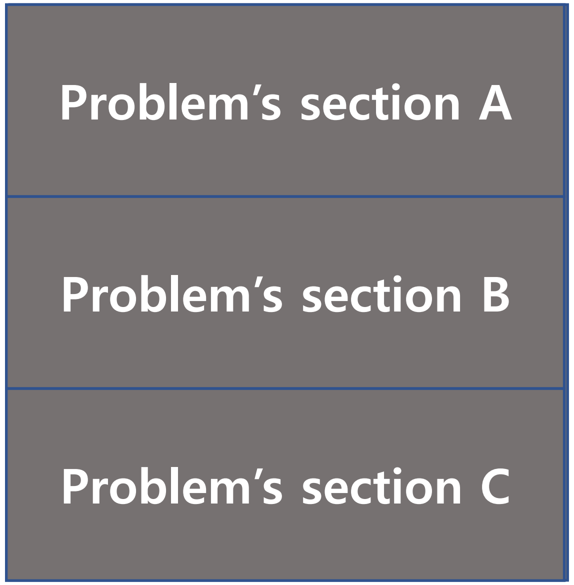




# 01 Massively Parallel Computing

# Massively Parallel Computing

## Massively Parallel Computing




Problem should be discrete
( linear / orthogonal ) .

**instructions**

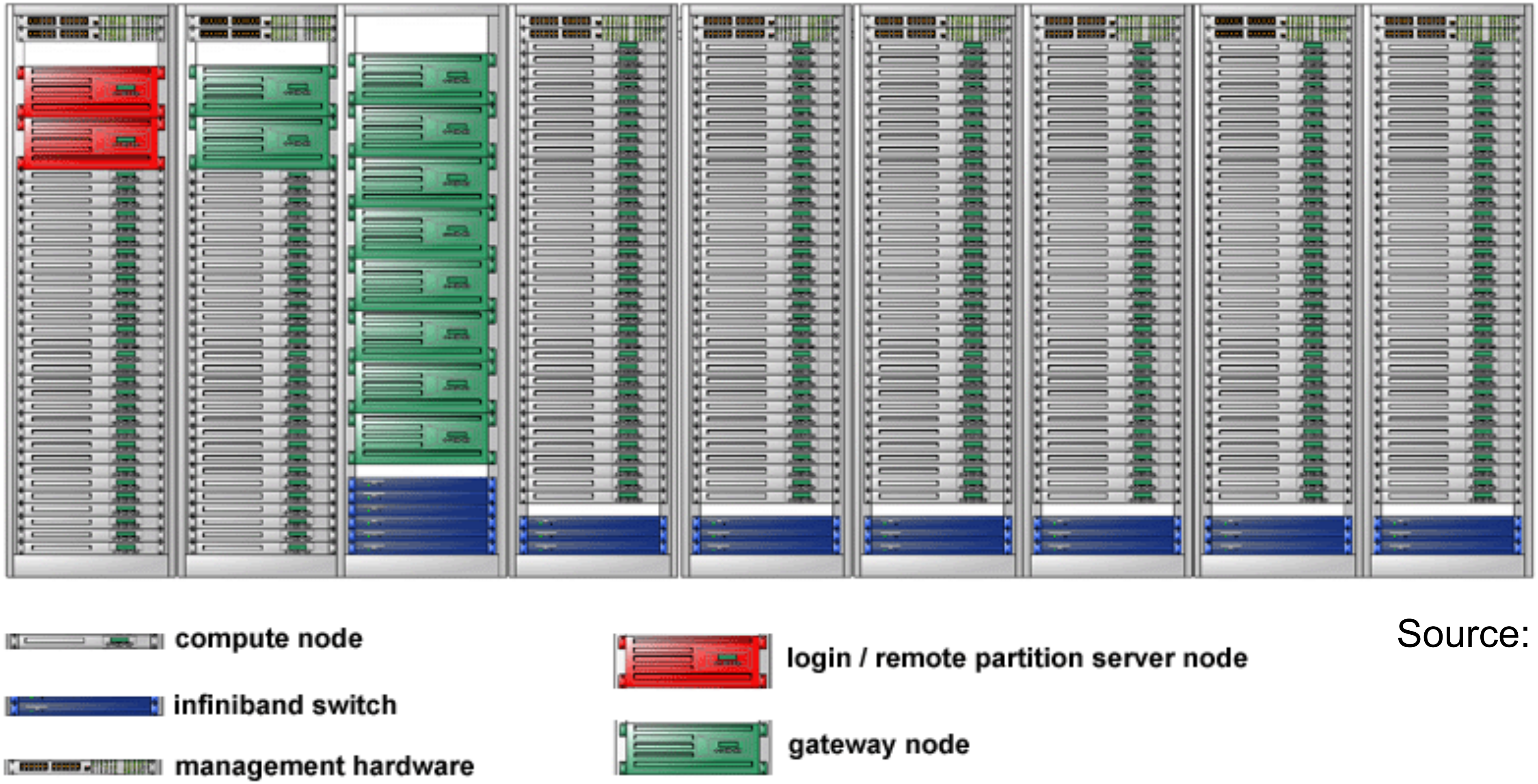


Processors could be multi processor
or multi machine connected by network

# Massively Parallel Computing



## Massively Parallel Computing

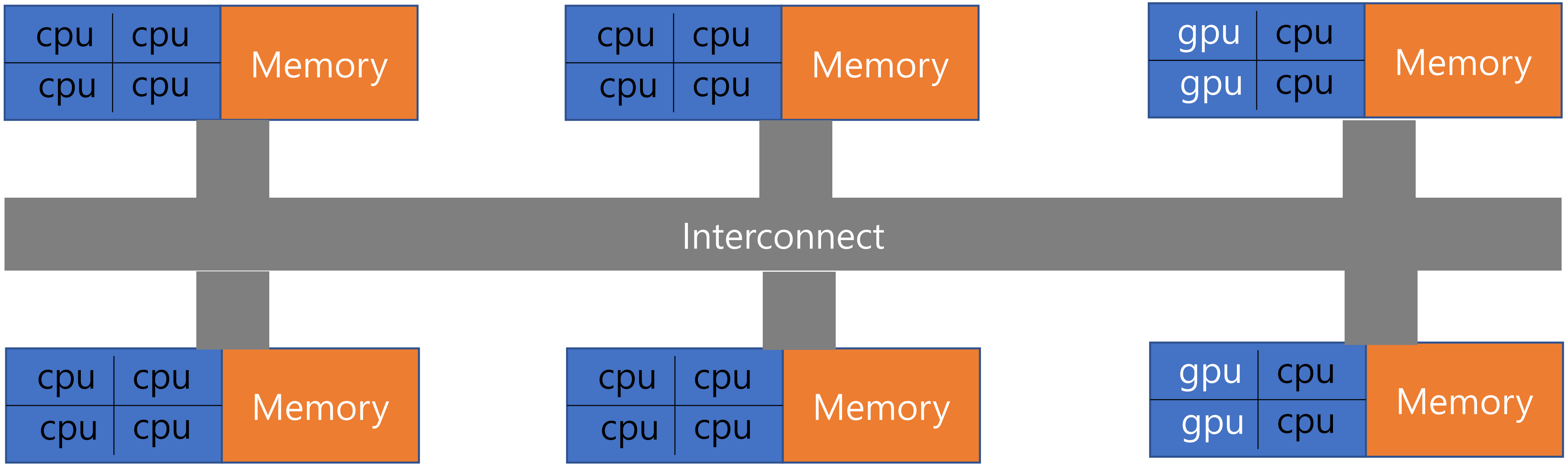



Source: LLNL

# Massively Parallel Computing

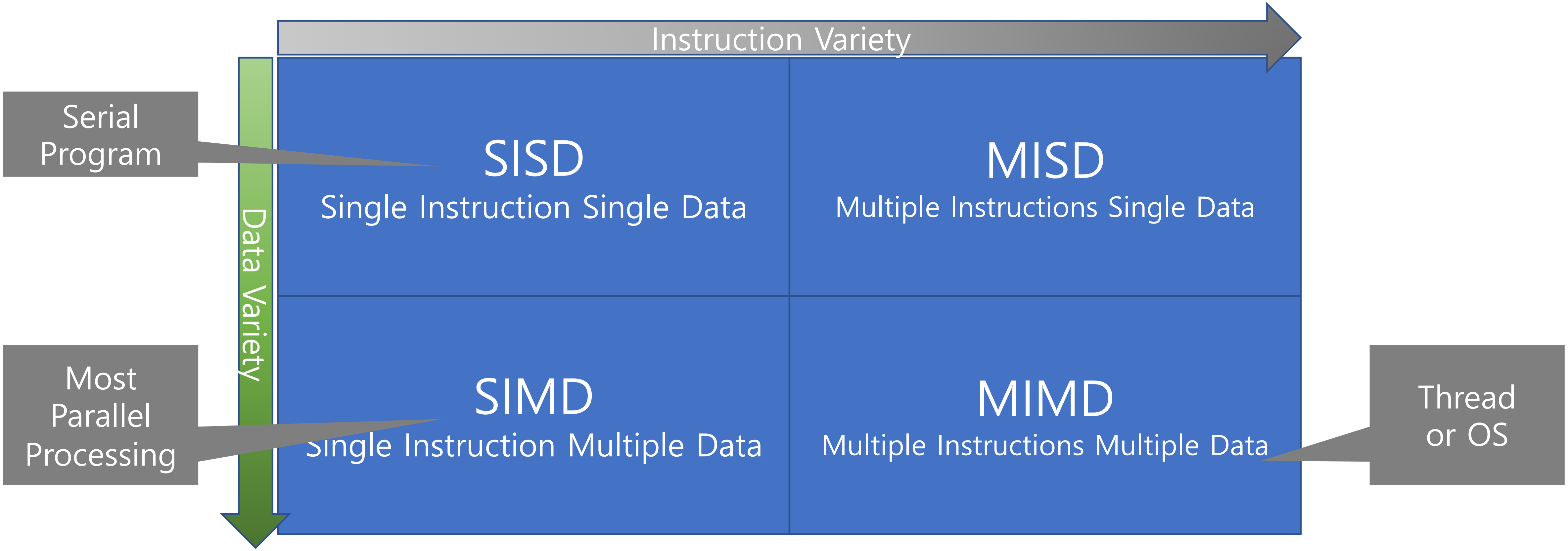


## Network model pic.



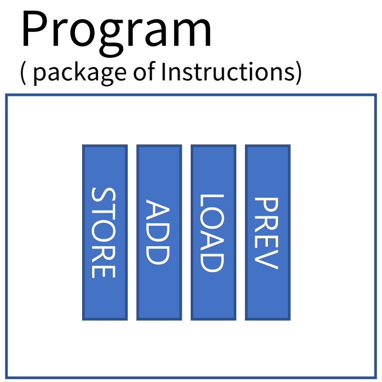
# Massively Parallel Computing type

## Flynn Classical Taxonomy



# Massively Parallel Computing type

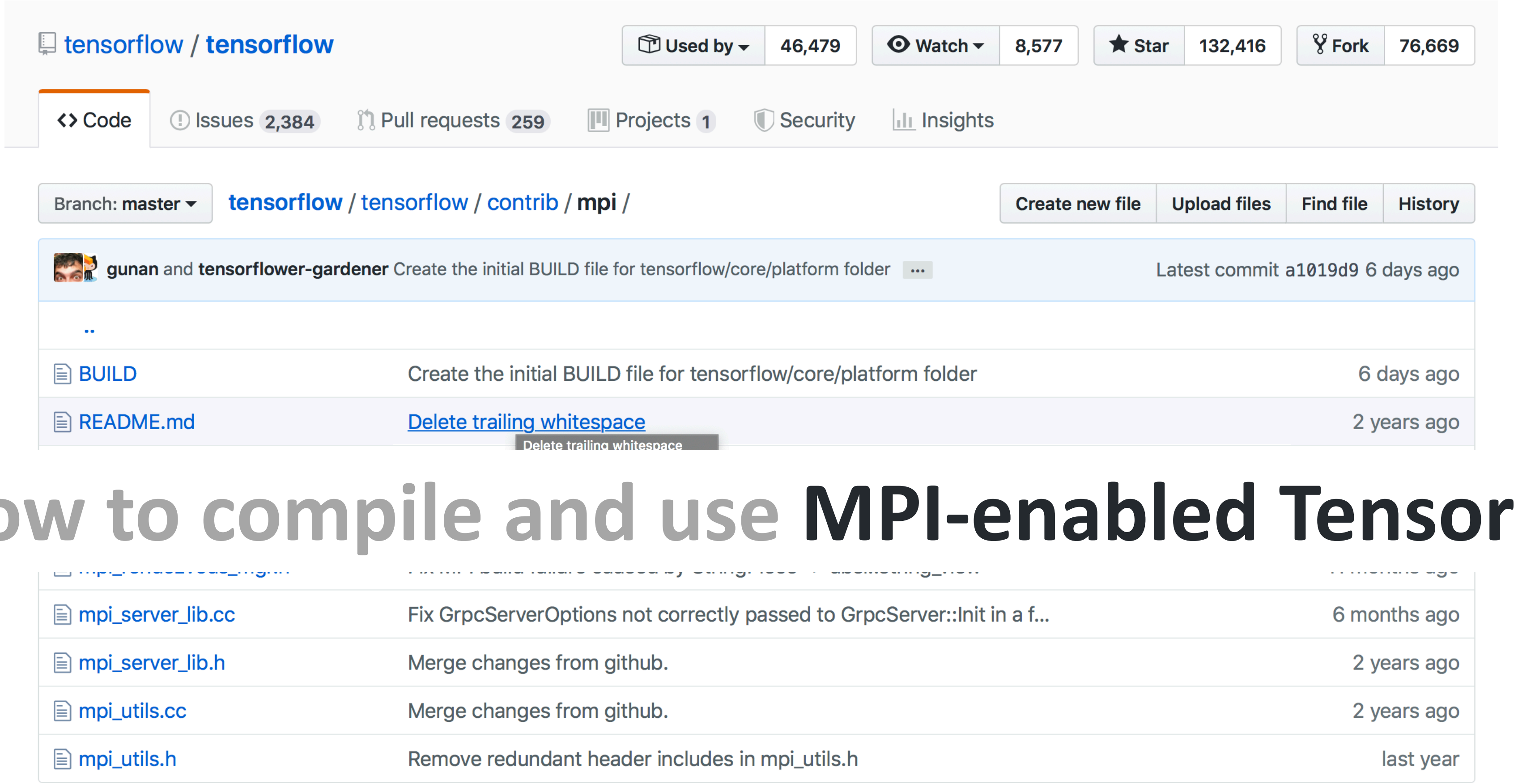
## Nowadays, Change Instruction to Program




| | |
|---|---|
| SPSD | MPSD |
| SPMD | MPMD |

# Massively Parallel Computing

## BTW, WHY MPC and Cloud #1?



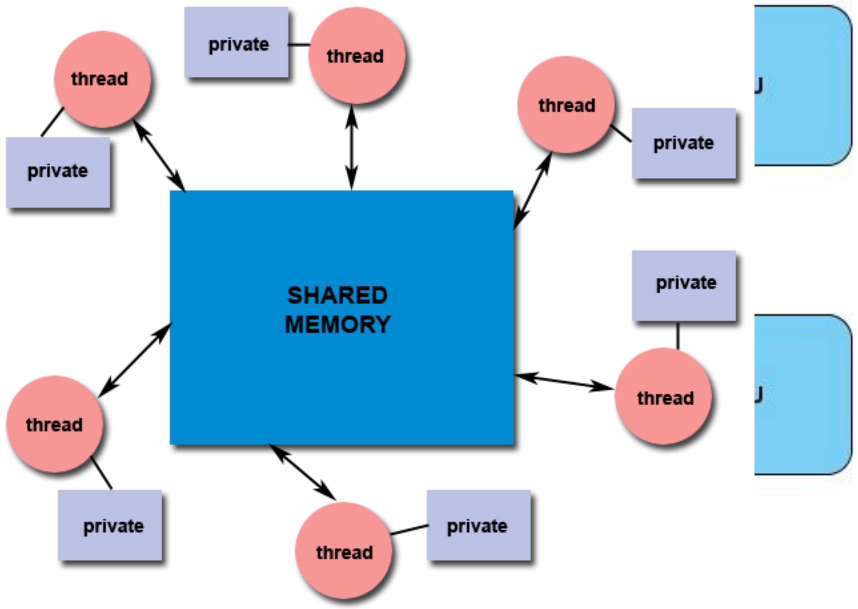

How to compile and use **MPI-enabled TensorFlow**

# 02 Generations of MPC programming

# Massively Parallel Computing type

**Parallel Programming model**

- Shared Memory Model
- Thread Model
  - POSIX thread
  - OpenMP
  - SHMEM



# Massively Parallel Computing type



## Parallel Programming model

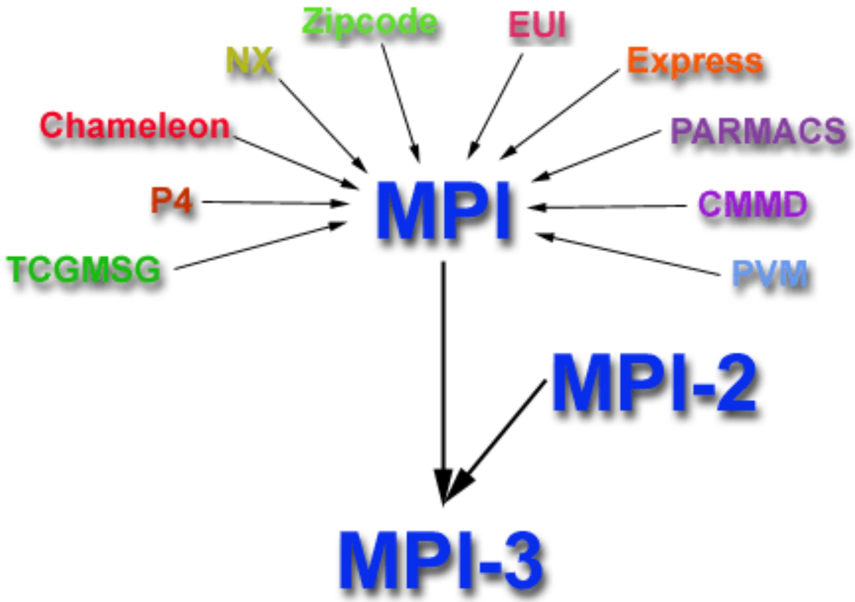
- Distributed Memory Model
  - MPI(Message Passing Interface)
  - 1994 MPI 1.0 Released
  - ssh ( or remote shell) based initialization

code

```
#include "mpi.h"
#include <stdio.h>
int main(int argc, char *argv[]) {
    int numtasks, rank, len, rc;
    char hostname[MPI_MAX_PROCESSOR_NAME];

    // initialize MPI
    MPI_Init(&argc,&argv);
   // get number of tasks
   MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
   // get my rank
   MPI_Comm_rank(MPI_COMM_WORLD,&rank);
   // this one is obvious
   MPI_Get_processor_name(hostname, &len);
   printf ("Number of tasks= %d My rank= %d Running
on %s₩n", numtasks,rank,hostname);
  // do some work with message passing
   MPI_Finalize(); }
```

copy to every cluster nodes

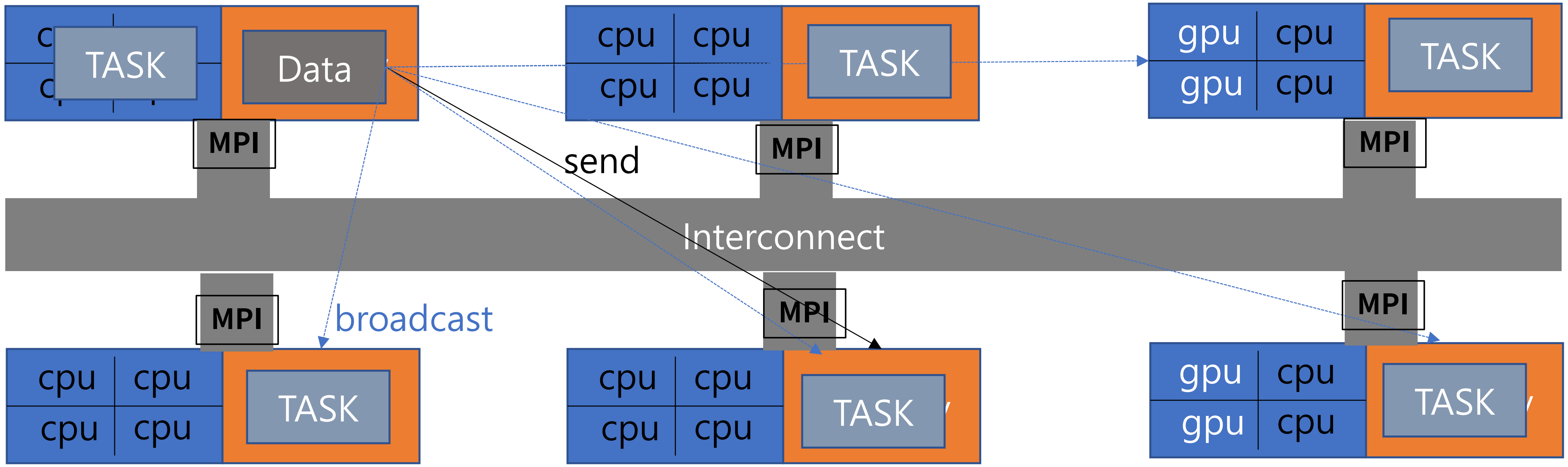



execution

```
mpirun –np 16 –hostfile hosts a.out
```

# MPC generation 1

## Parallel Programming model : MPI



# Massively Parallel Computing type

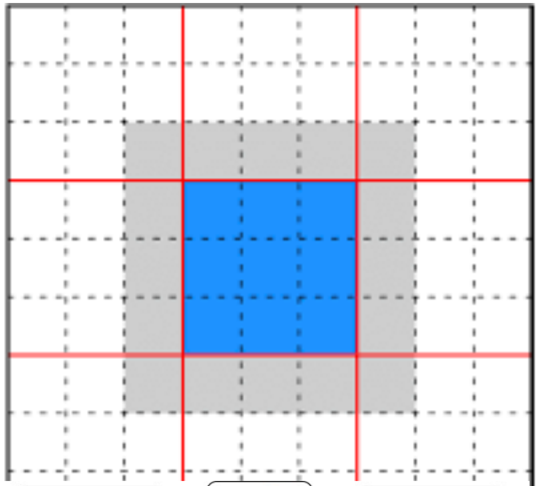
**Designing Parallel Programming :**

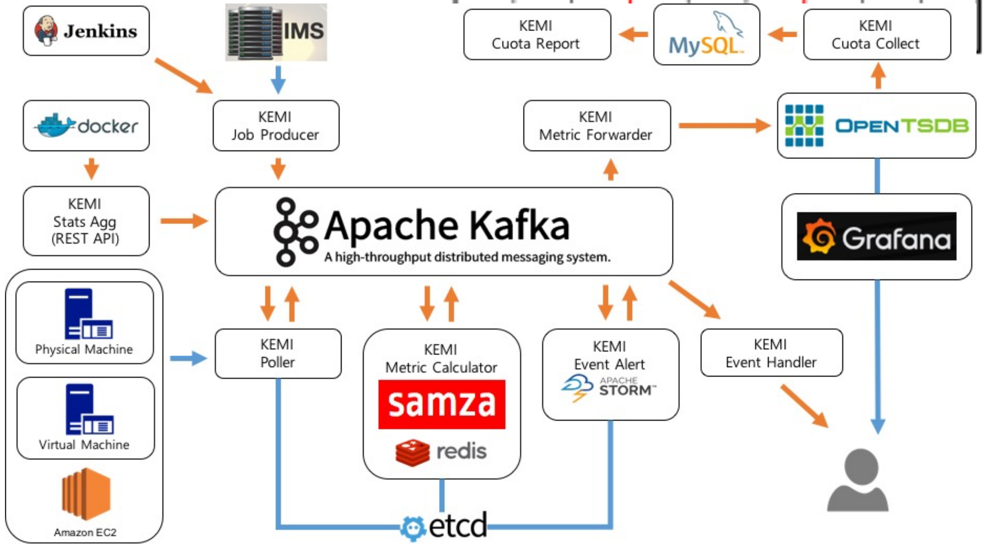
Domain Decomposition

- Data is decomposed
- Simple problem with Big Memory

Functional Decomposition

- Instruction is decomposed
  - Signal Processing
  - Workflow
  - Data pipe lining





# MPC Considering factors

**Communication overhead**

latency and bandwidth

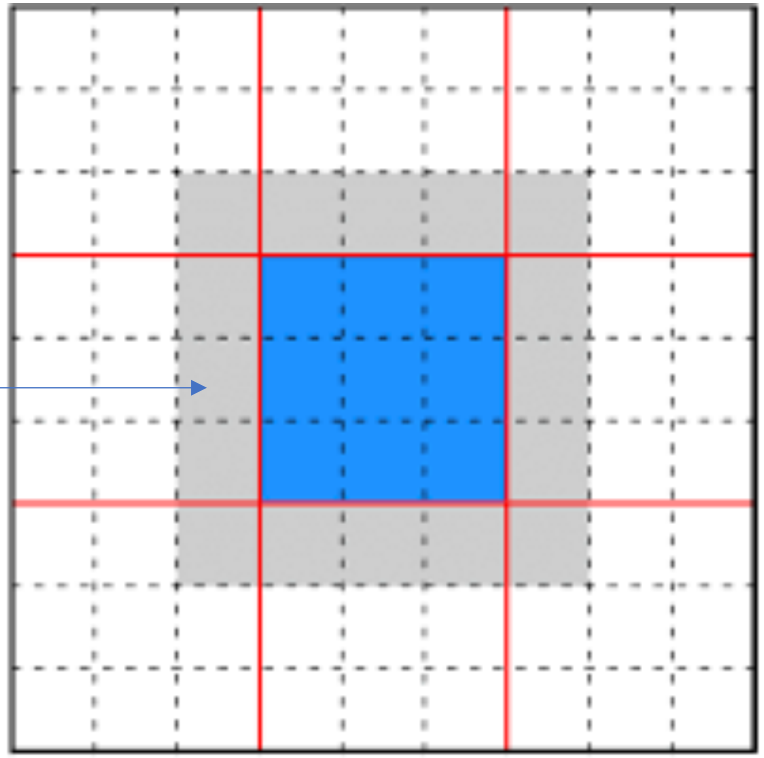
Visibility of communications

Synchronous vs. asynchronous communications

**Data Dependencies**

A(J) = A(J-1) + A(J+1)

data(domain)
overlapping



**Data sharing**

(Data/program) Send or Broadcast

Shared FileSystem

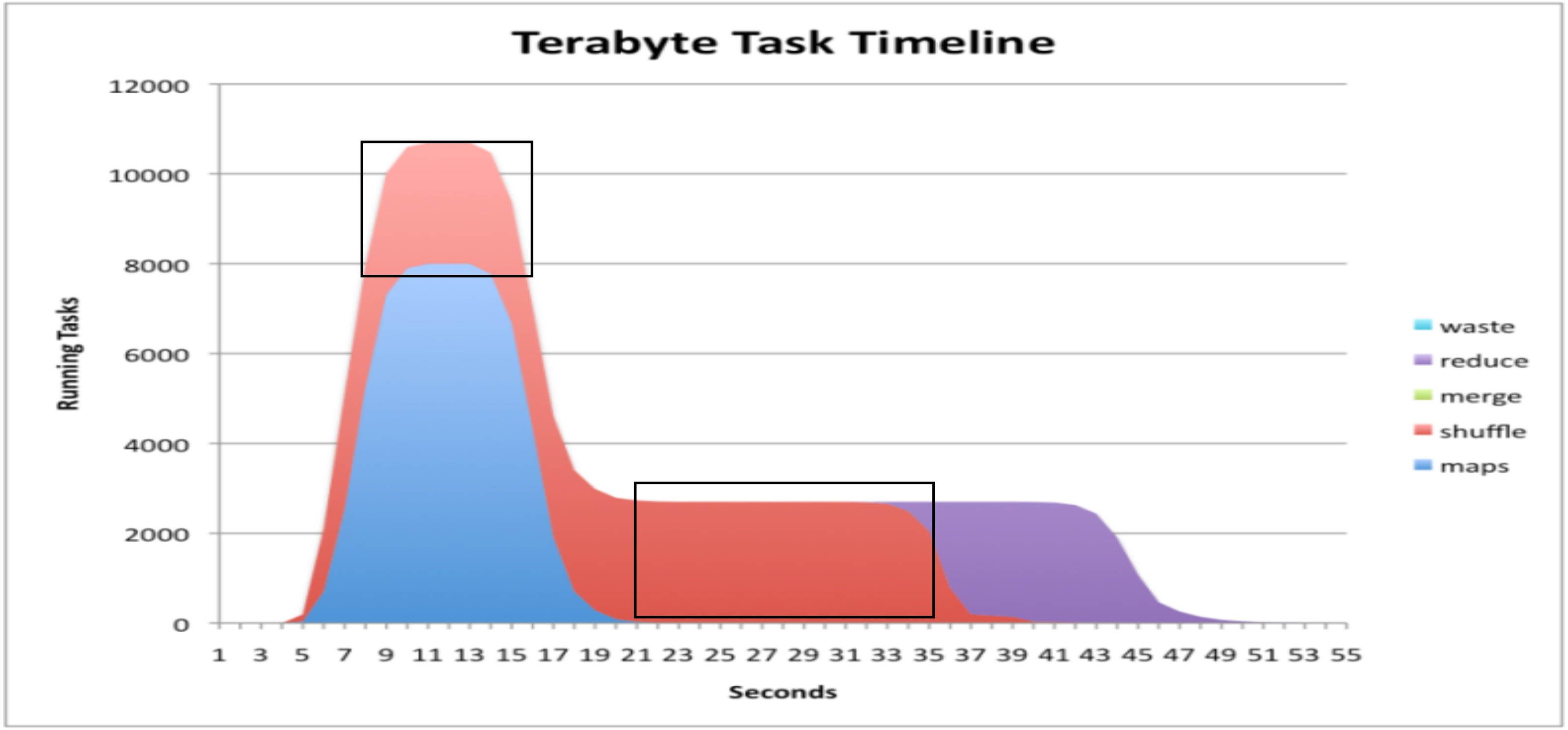
# MPC generation 2

**Google said they found solution:**

Google File System and

MapReduce Framework (J Dean, SIGMOD, 2004)

- Share Nothing, Zero Down Time Computation Framework
- But, They do have issues with data communications



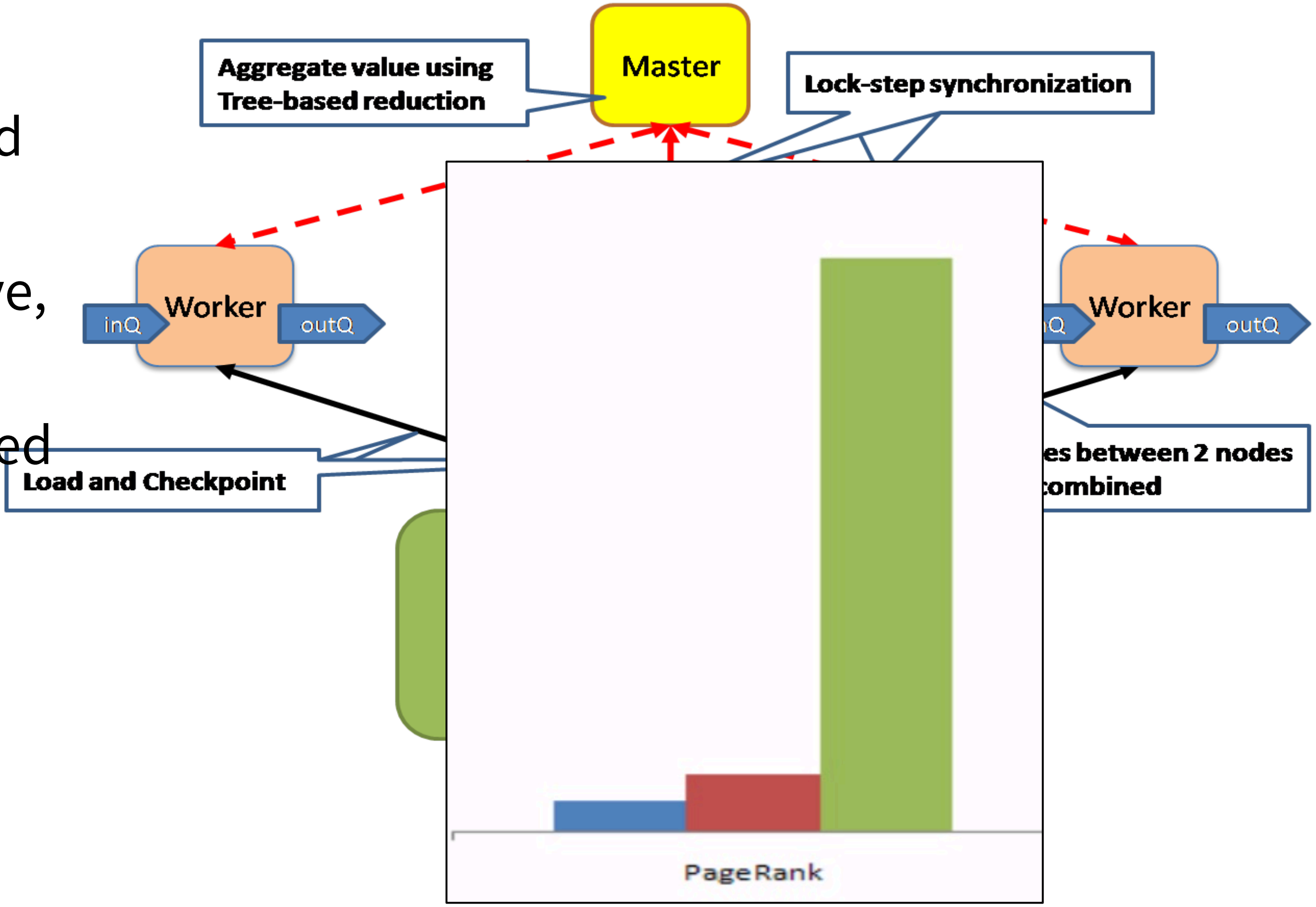
# MPC generation 3

## Google said they found another solution:

**Pregel**(SIGMOD, 2010)**:**

- BSP(Bulk Synchronous Processing) based Computational framework
- MapReduce is well suited for non-iterative, data parallelized problem
- communication is only done by predefined graph connections

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 1:10 | 0 | 1 | 2 | 3 | 4 |
| 1 | 0:100 | 0 | 1 | 2 | 3 | 4 |
| 2 | 0:102 | 0 | 1 | 2 | 3 | 4 |
| 3 | 0:101 | 0 | 1 | 2 | 3 | 4 |
| 4 | 0:14 | 0 | 1 | 2 | 3 | 4 |
| 5 | 0:12 | 0 | 1 | 2 | 3 | 4 |
| 6 | 0:11 | 0 | 1 | 2 | 3 | 4 |
| 7 | 0:100 | 0 | 1 | 2 | 3 | 4 |
| 8 | 0:13 | 0 | 1 | 2 | 3 | 4 |
| 9 | 2:9 | 0 | 1 | 2 | 3 | 4 |



# 03 Cloud Native way of Tackling MPC

# Cloud Native way of tackling technical issues

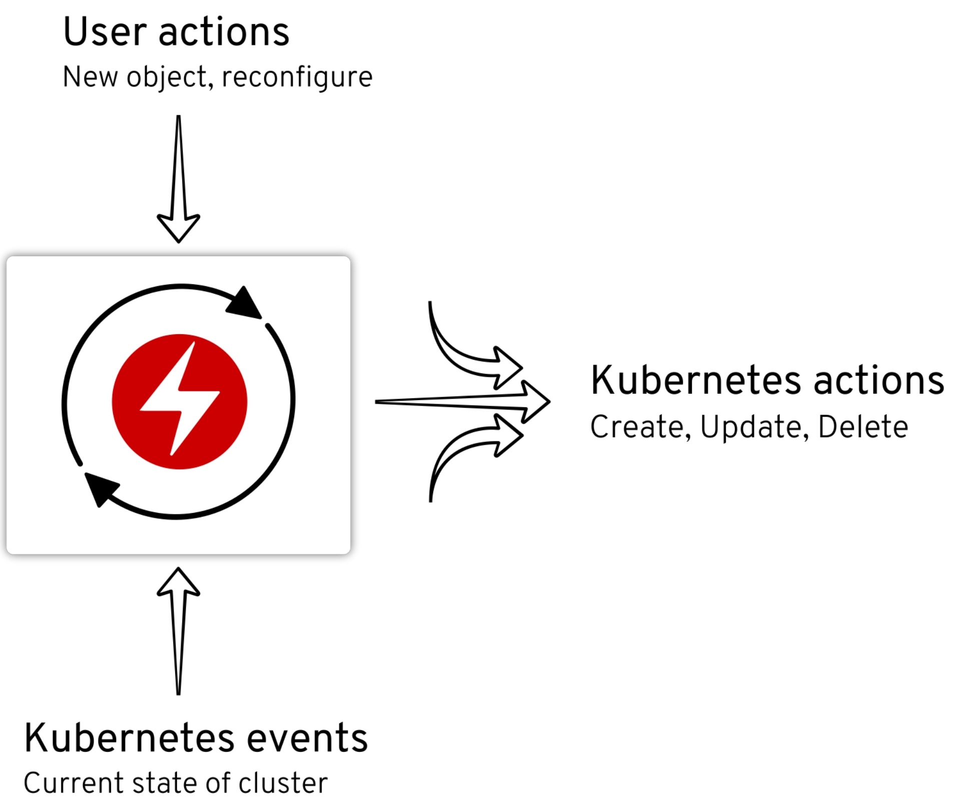


- **Cloud Native means it uses CONTAINER / CONTAINER ORCHESTRATOR**
- **Cloud Native means 'Automation'**
- **Network Architecture or Communication Model Setup**
  - Create Cluster Automatically
  - Create Communication Model Automatically
- **Job Fault tolerance**
  - Massive and Log running Job should have a way to handling 'Failures'
  - Like MapReduce's 'Speculative Execution'

# Network Architecture/Communication Model Setup



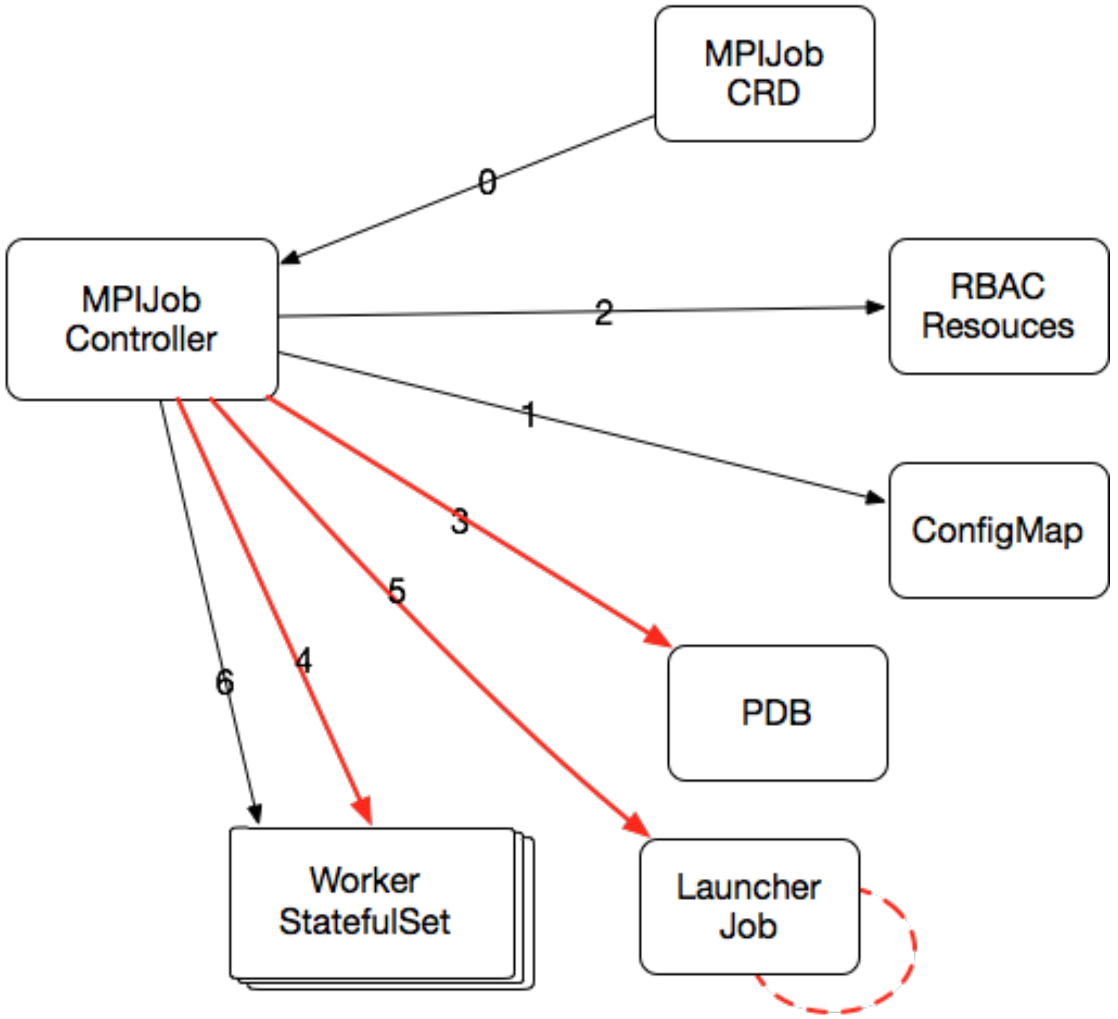
- **Using k8s's operator**
- **Operator is a representation of its component ( Deployment , configmap, statefulset ..etc )**
- **Operator allows develop built-in plugin for your purpose**



# Network Architecture/Communication Model Setup



- MPI operator
  - Create MPI Cluster over k8s
  - setup ssh  between the pod
    - create RSA key , save it config map, send to pod etc..
  - copy MPI program to the pod



# Job Fault tolerance

- Most notorious thing MPI
  - if one of the process down, all system goes down
  - so MPI developer always considering remedy for this like restart file , history file,
  - this creates the complexity
- In a Cloud Native world, you can use container level check pointing
  - CRIU (Checkpoint/Restart In User space www.criu.org)
  - CRIU writes container status to share file system
  - with this you can restart the MPI pod

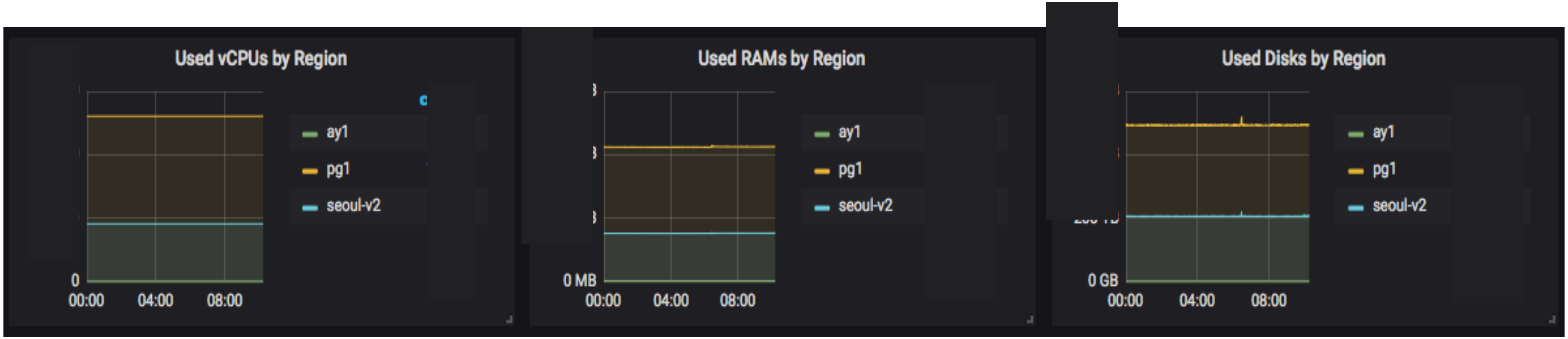
# 04 Application Case with Cloud Native MPC in kakao

# Massively Parallel Computing

## BTW, WHY MPC and Cloud #2?

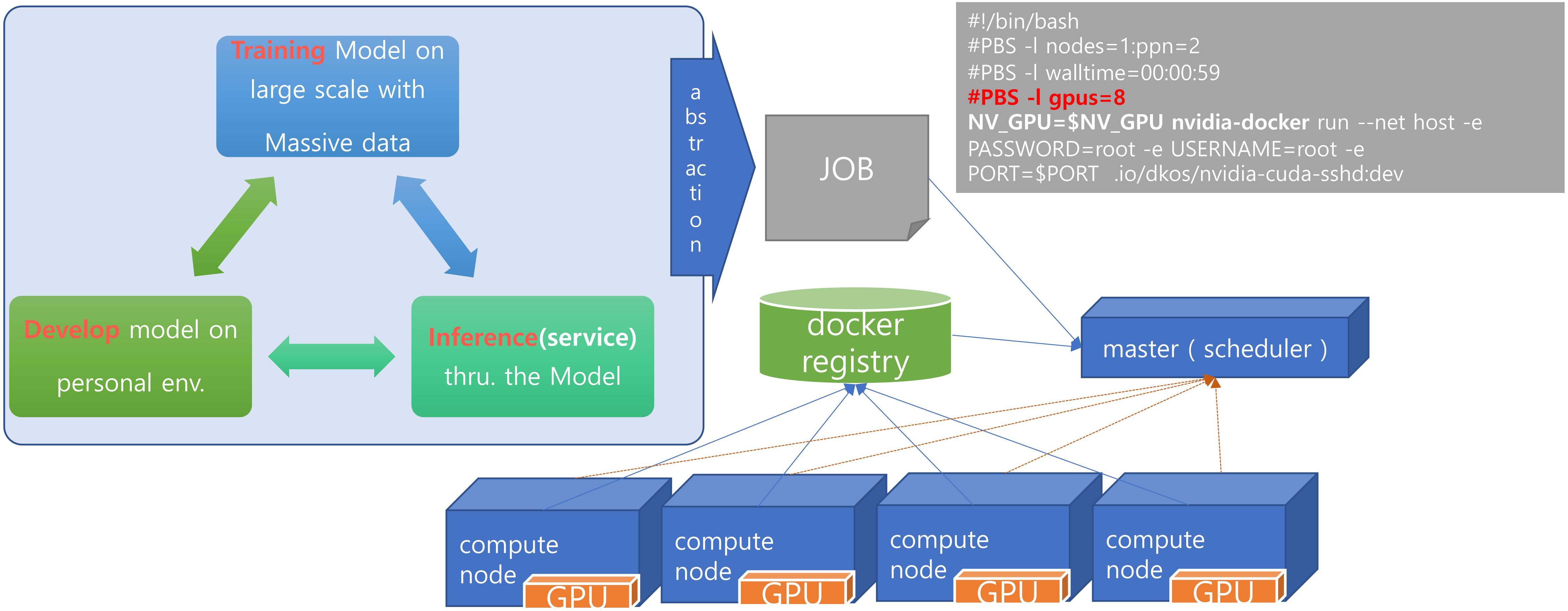
**#Core 000000, Memory 000TB, Disk 000TB**

**What if we can utilize all together?**



# MPC with cloud Phase 1

- **Simple integration with job scheduler and container**



# MPC with cloud Phase 1

- **The issues with Phase 1**
  - **Cannot effectively use resource for each process ( training/inference/Develop)**
  - **Cannot elastically scale in/out the process**
    - worker is statically assigned, process is also statically assigned
  - **Cannot do the parallel computing with containers**
    - container is just one of method for copying method

# MPC with cloud Phase 2

- **Using DKOSv3 (Container orchestrator as a service with k8s/dcos) in kakao, we can use k8s and MPI operator very easily**
- **setup YAML for computing and running**

```
❯ kubectl get crd
NAME                              AGE
ciliumendpoints.cilium.io         132d
ciliumnetworkpolicies.cilium.io   132d
mpijobs.kubeflow.org              13m
```

```
apiVersion: kubeflow.org/v1alpha1
kind: MPIJob
metadata:
  name: mpijob
spec:
  replicas: 2
  processingResourceType: cpu
  template:
    spec:
      containers:
```

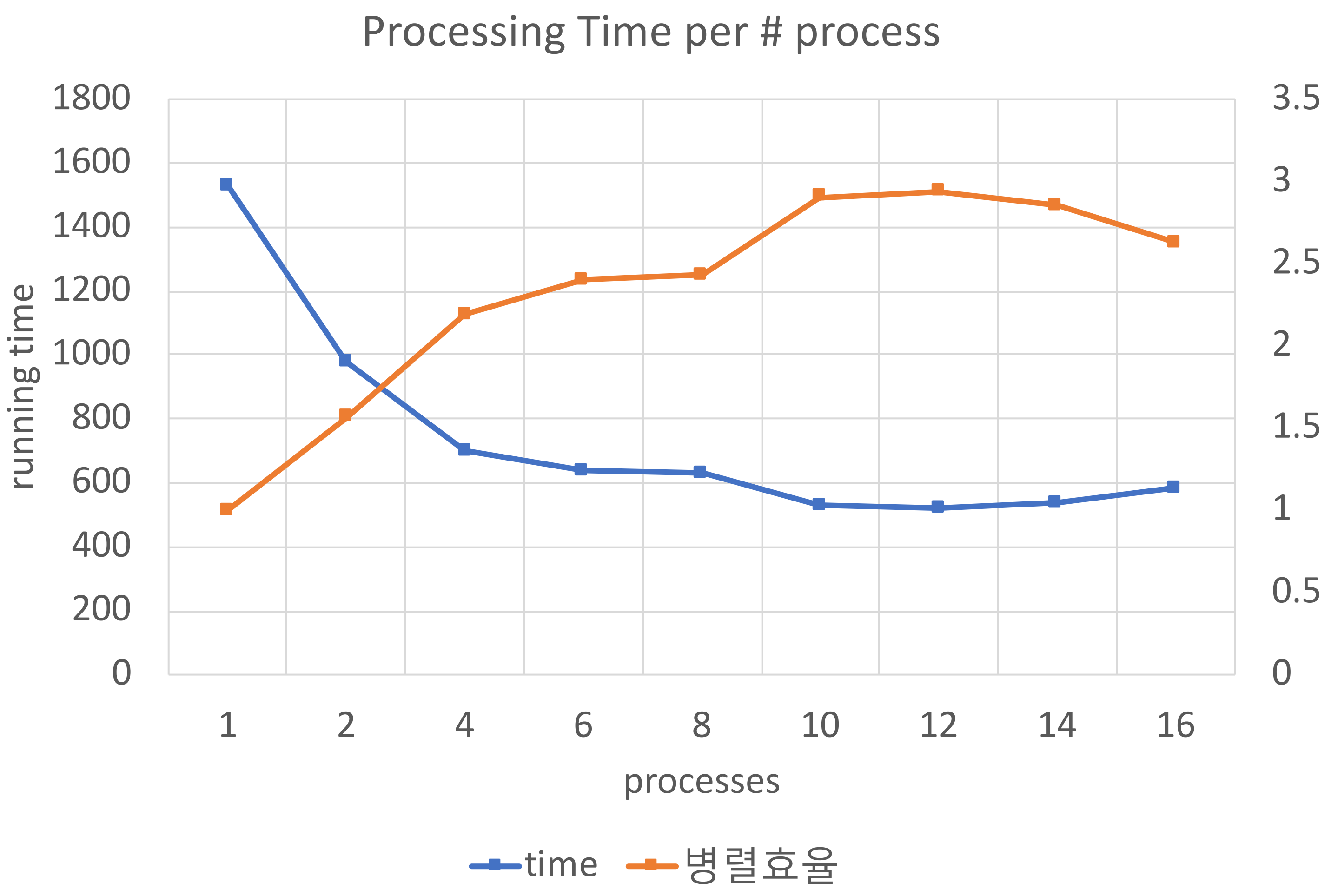
**kubectl create –f sample.yaml**

```
          - -c
          - 'mpirun -bind-to none -map-by slot --allow-run-as-root -x NCCL_DEBUG=INFO
-x LD_LIBRARY_PATH bash -c "ldconfig /usr/local/cuda/lib64/stubs ;  python
tensorflow_mnist.py"'
        resources:
          limits:
            cpu: 4
        volumeMounts:
        - mountPath: /examples/MNIST-data-0
          name: datavolume
        - mountPath: /examples/MNIST-data-1
```

**sample.yaml**

# MPC with cloud Phase 2

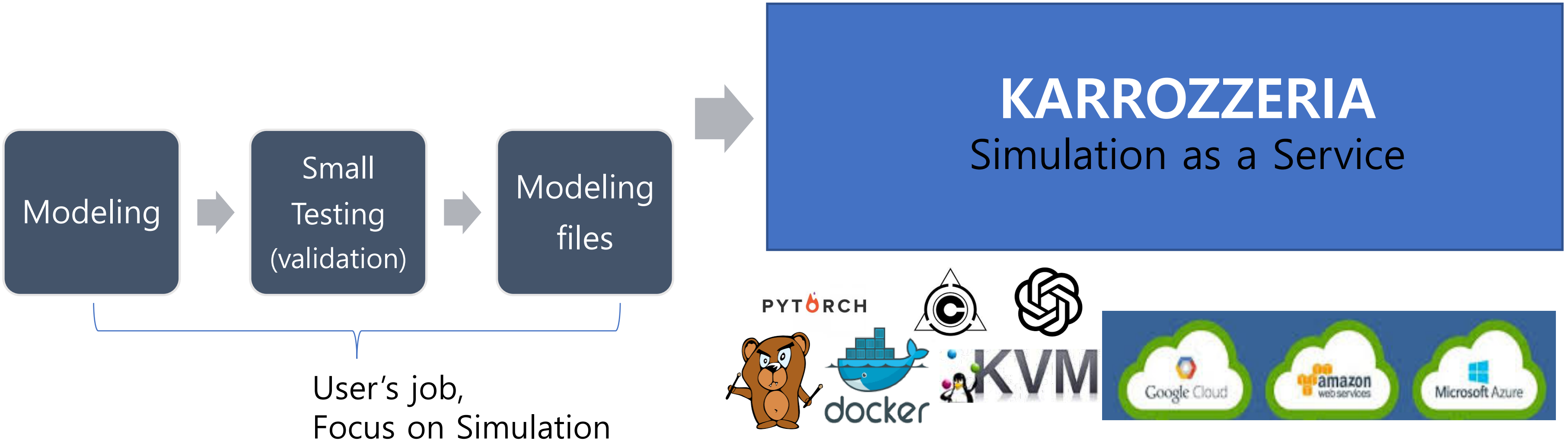
- **Then the result comes out**



# MPC with cloud Phase 3

- **Simulation as a Service**
- **Put the model binary(**AI/ML/MPC···**), we give you the data and simulator**

```
curl -X POST "http://karrozzeria:5000/1.0/simluation/tests"  -d
"{"runtimePath": ”tensorflow", ”model_data": ”car_street"}"
```



# MPC with cloud Phase 3 Issues

- **Programming Model issue**
  - Domain Decomposition overlapping and reinforcement Learning multiagent
- **InterConnect issue**
  - Container uses tunnel network for the inter pod networking (CNI )
  - Latency and throughput always performance bottle neck
  - kakao uses SR-IOV and VxLAN offloading for this purpose.
- **File( Program, Data )Sharing issues**
  - need API operated distributed file system
  - kakao has object file system ( kage/tenth) and develop CSI plugin for this

- Q &A