FUJITSU

The Future of HPC Systems: Towards Large-Scale Time-Slice Execution and Adaptive Accelerator Allocation

Eiji Yoshida

Computing Laboratory

Fujitsu Ltd.

Making HPC and AI more Efficient

FUJITSU

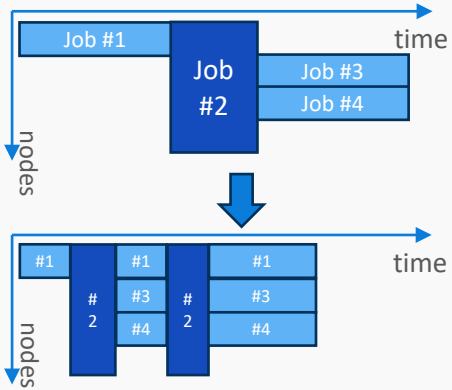
- **Better <u>interactivity</u> for new applications and users**
  - Instant execution is difficult on conventional HPC systems
  - New HPC users are not familiar with batch queue systems
    - Users access systems through interactive terminals like Jupyter Notebook

- **Maximize the <u>efficiency</u> of resource usage**
  - Accelerators are valuable resources
    - Supply shortage of GPUs
    - Increased cost of accelerator devices

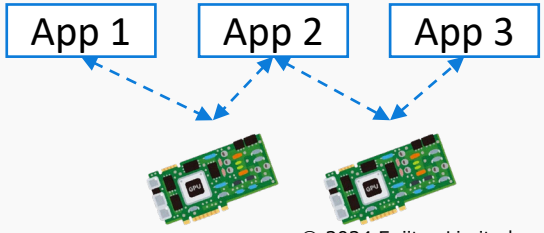
Our Technologies for new Requirements

FUJITSU

- **Scalable Fine-grained gang scheduling**
  - Scalable time-sliced parallel computing job execution
  - ➔ Providing better interactivity on HPC systems
  - ➔ Improving job execution efficiency



- **AI Computing Broker - Adaptive accelerator allocation**
  - Dynamic resource allocating mechanism for GPU applications
  - ➔ Application-aware efficient GPU usage within multiple jobs

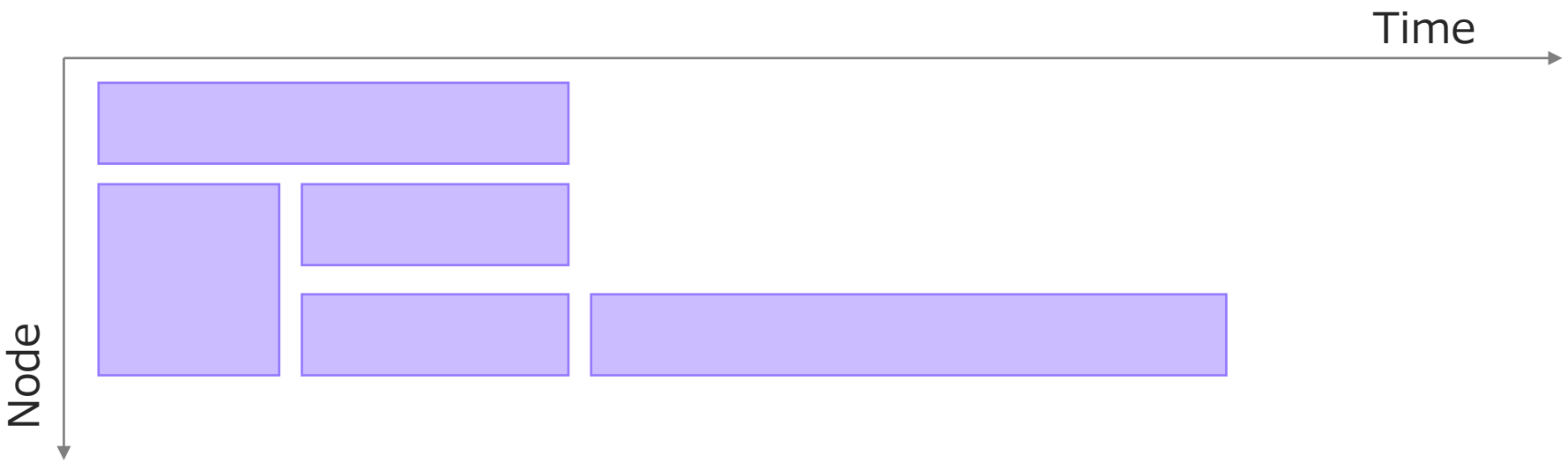


Scalable Fine-grained gang scheduling

Lack of Interactivity on HPC systems

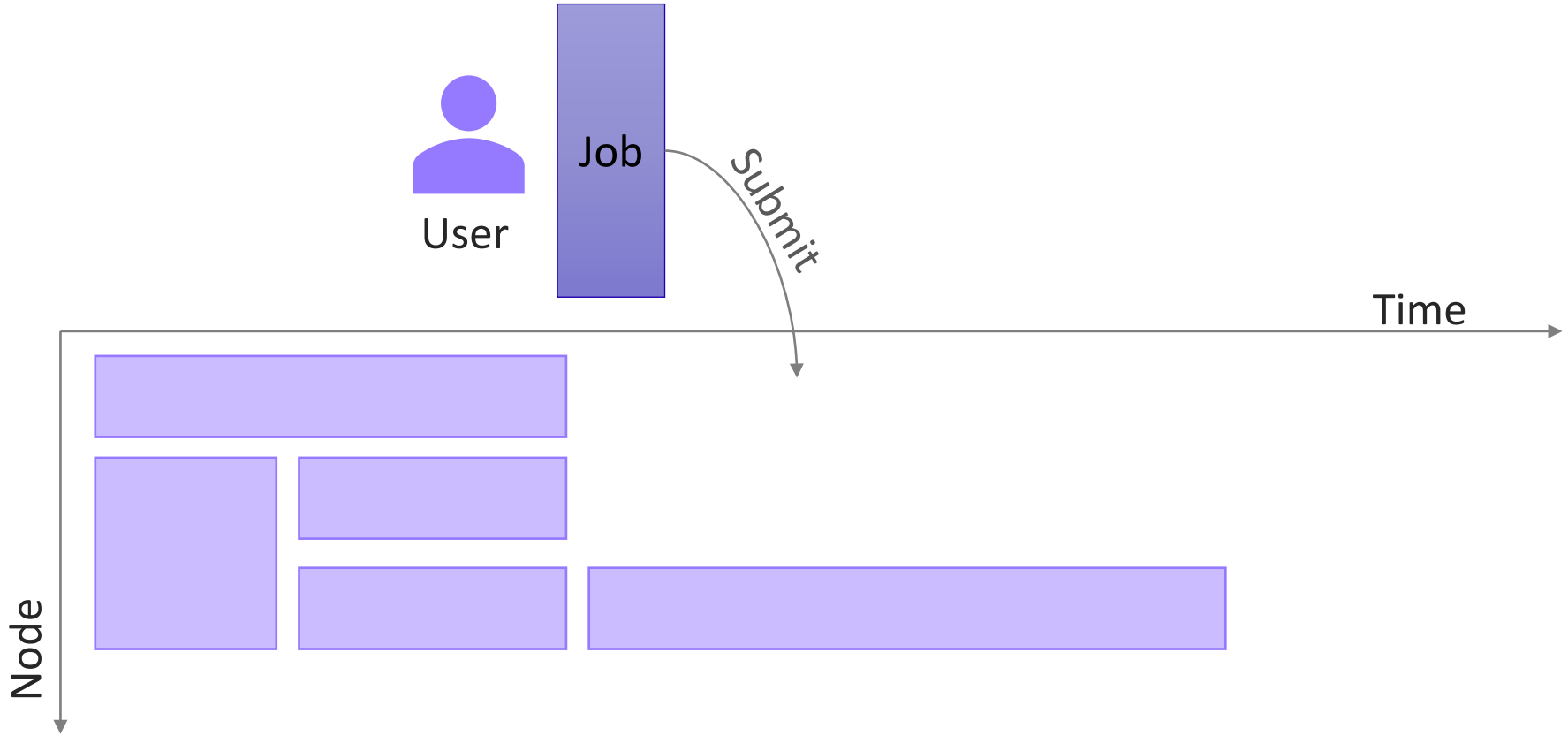
FUJITSU

- Long waiting time is required even for short jobs
  - The limitation of batch queueing systems

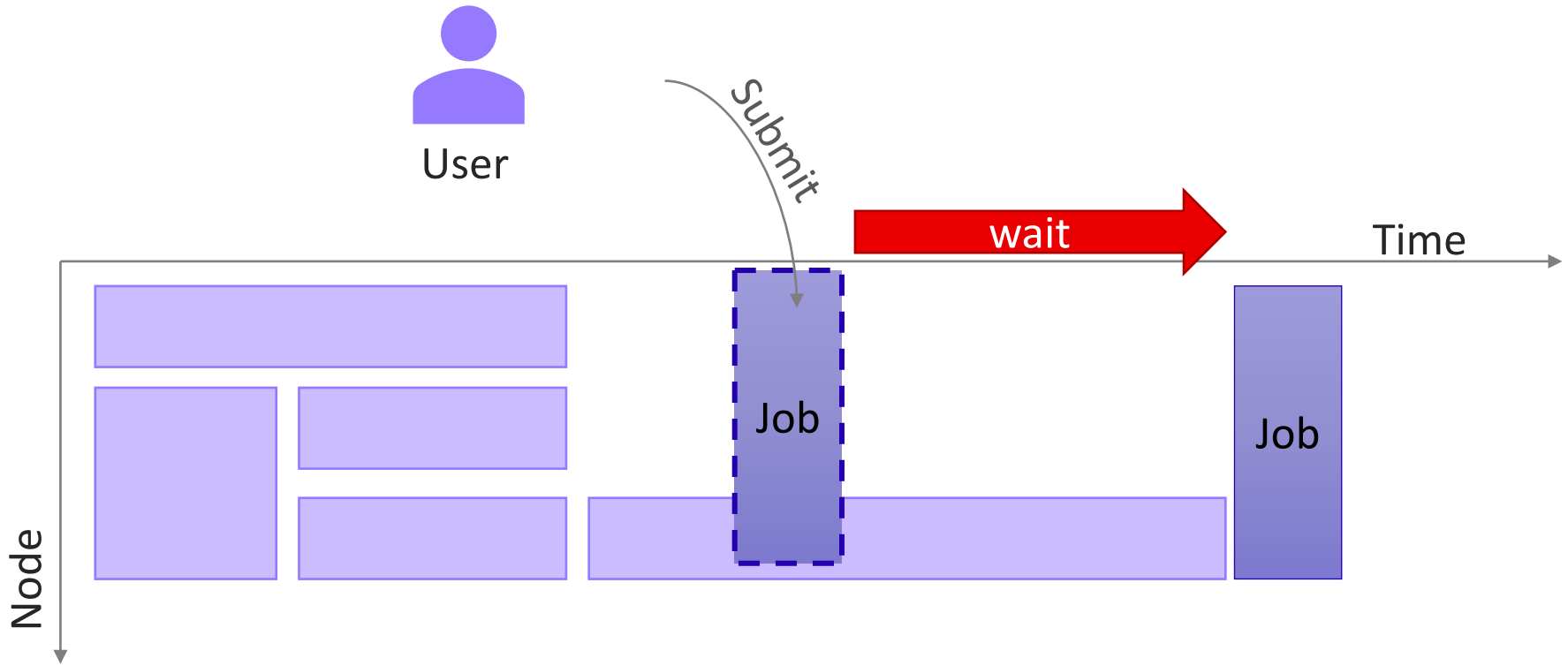


Lack of Interactivity on HPC systems

FUJITSU

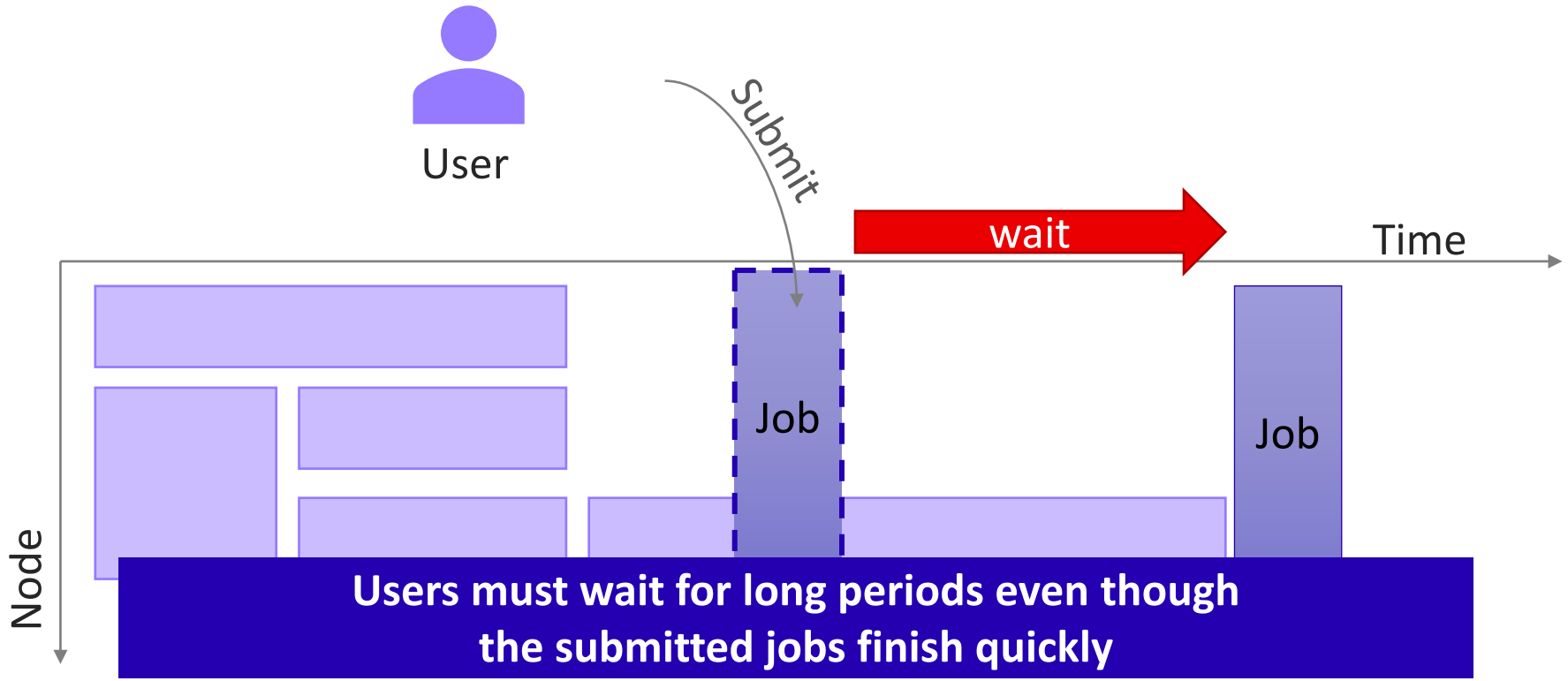


Lack of Interactivity on HPC systems



Lack of Interactivity on HPC systems

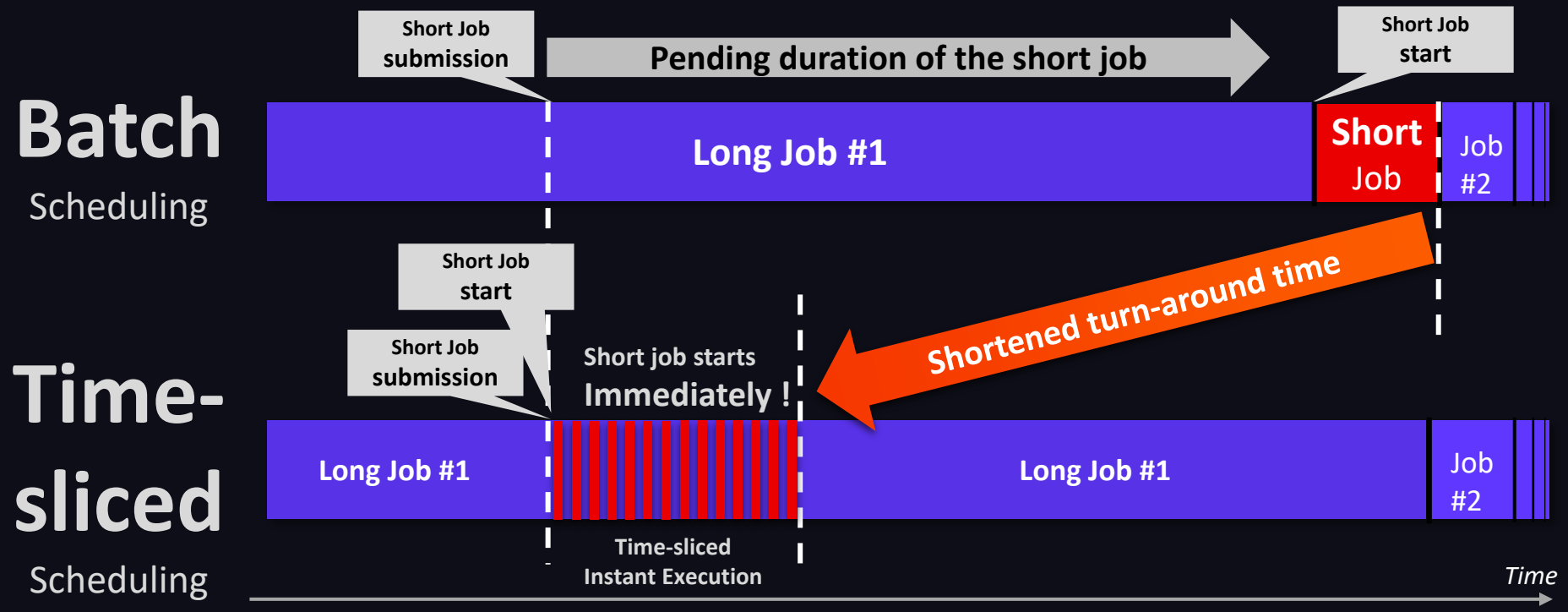
FUJITSU



Users must wait for long periods even though the submitted jobs finish quickly

Transition from Batch Scheduling

FUJITSU

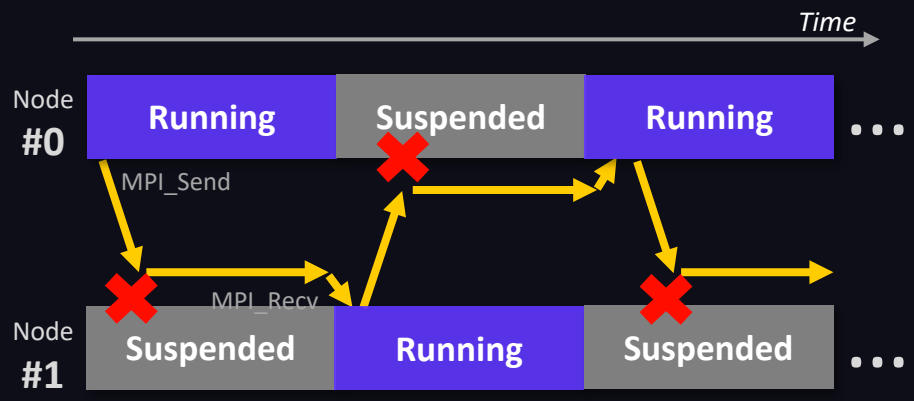


Requirements of Parallel Applications

FUJITSU

Unsynchronized

Scheduling



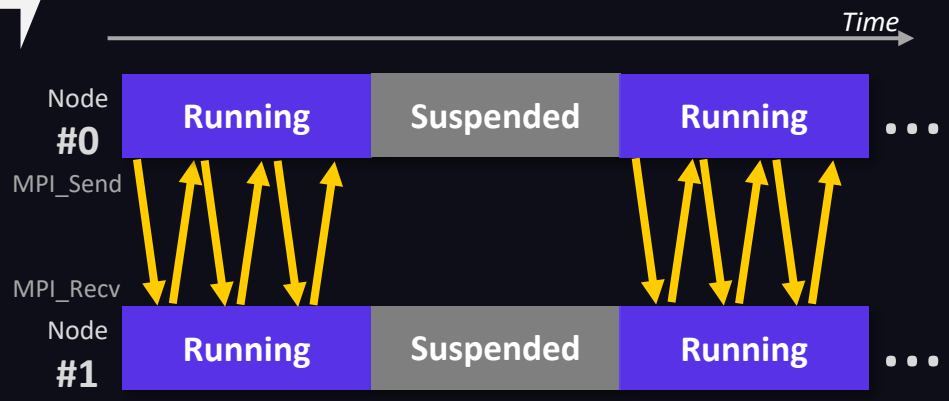
Poor Performance

Only one message can be sent per time-slice (the worst case)

Synchronized

Scheduling

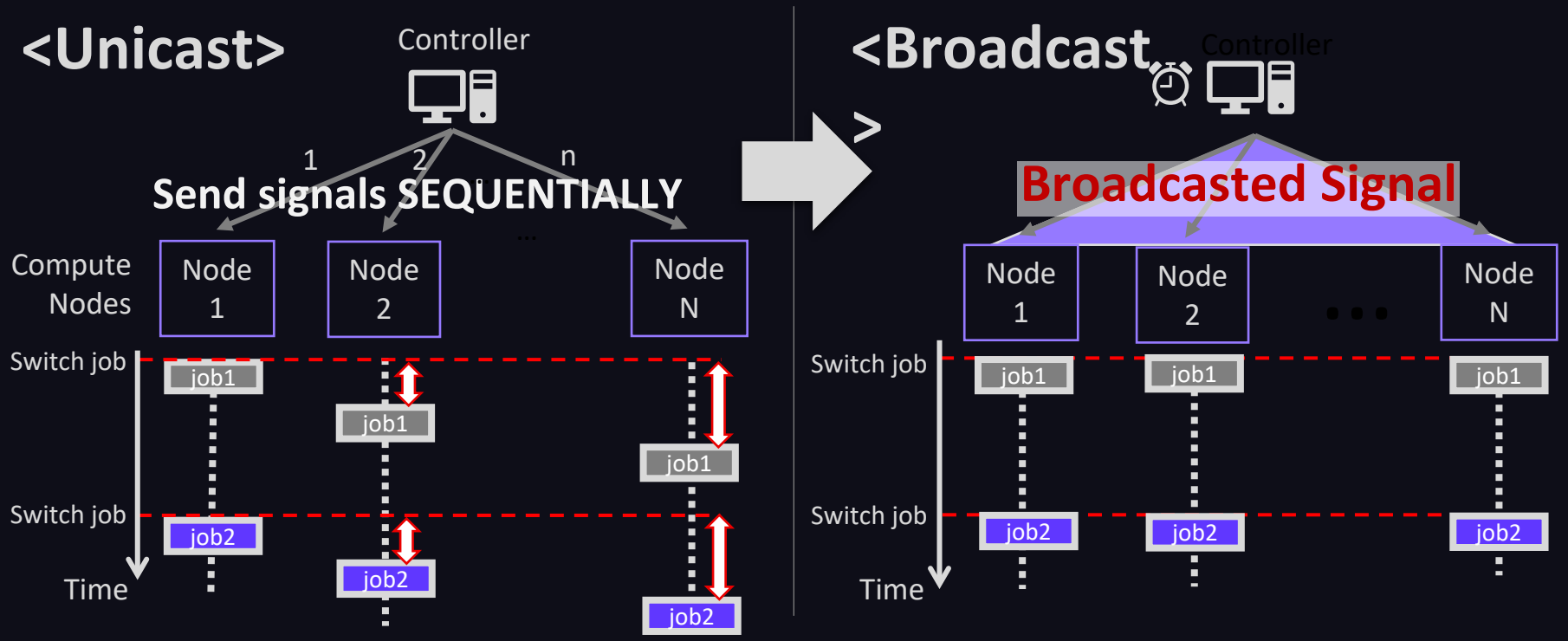
Also Known as Gang Scheduling



The Same Performance as usual

Multiple messages can be sent in a single time-slice

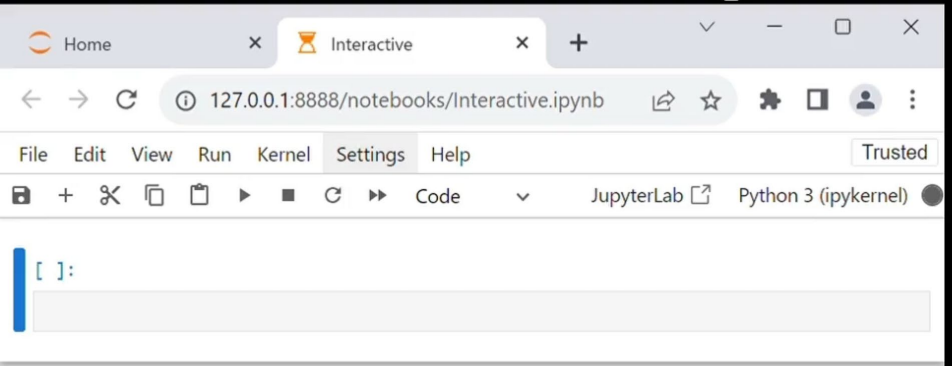
Scalable Synchronization with Broadcast Control Signals



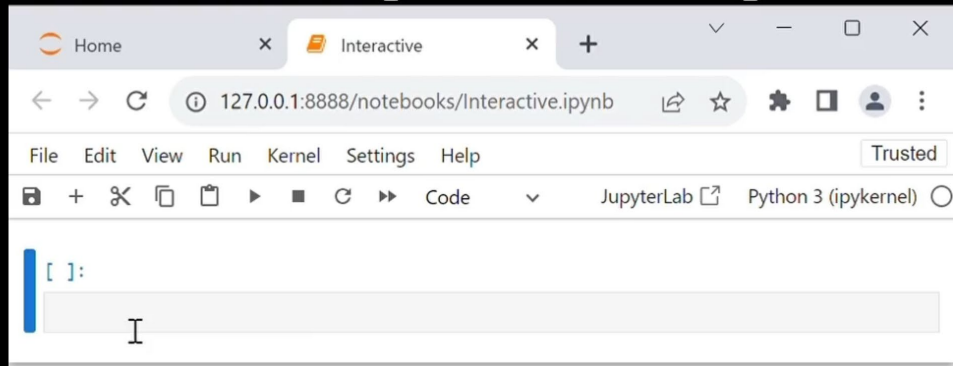
Jitter caused by sequential unicast communications

Good synchronization with a single broadcast communication

Batch scheduling



Gang scheduling



This is a demonstration of an interactive Jupyter Notebook that uses a parallel cluster as its backend processing system.

Batch scheduling

Home | Interactive

127.0.0.1:8888/notebooks/Interactive.ipynb

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[*]:

```
import ipyparallel as ipp
rc = ipp.Client(profile='example-profile')
```

[ ]:

```
%%px
from mpi4py import MPI
import random
import time
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
print(f"rank = {rank}")
```

Gang scheduling

Home | Interactive

127.0.0.1:8888/notebooks/Interactive.ipynb

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[4]:

```
import ipyparallel as ipp
rc = ipp.Client(profile='example-profile')
```

[ ]:

```
%%px
from mpi4py import MPI
import random
import time
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
print(f"rank = {rank}")
```

Preparing the parallel environment

Writing the parallel code to execute

Batch scheduling

Home | Interactive

127.0.0.1:8888/notebooks/Interactive.ipynb

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[*]:

```
import ipyparallel as ipp
rc = ipp.Client(profile='example-profile')
```

[*]:

```
%%px
from mpi4py import MPI
import random
import time
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
print(f"rank = {rank}")
```

[ ]:

Gang scheduling

Home | Interactive

127.0.0.1:8888/notebooks/Interactive.ipynb

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[4]:

```
import ipyparallel as ipp
rc = ipp.Client(profile='example-profile')
```

[5]:

```
%%px
from mpi4py import MPI
import random
import time
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
print(f"rank = {rank}")
```

%px: 100% 8/8 [00:01<00:00, 6.60tasks/s]

```
[stdout:3] rank = 3
[stdout:5] rank = 5
[stdout:2] rank = 2
[stdout:4] rank = 4
[stdout:6] rank = 6
[stdout:7] rank = 7
[stdout:0] rank = 0
[stdout:1] rank = 1
```

We can see the result immediately on gang scheduling system, but nothing appears on the batch scheduling system

Batch scheduling

Home | Interactive

127.0.0.1:8888/notebooks/Interactive.ipynb

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[*]:

```
import ipyparallel as ipp
rc = ipp.Client(profile='example-profile')
```

[*]:

```
%%px
from mpi4py import MPI
import random
import time
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
print(f"rank = {rank}")
```

[ ]:

Gang scheduling

Home | Interactive

127.0.0.1:8888/notebooks/Interactive.ipynb

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[4]:

```
import ipyparallel as ipp
rc = ipp.Client(profile='example-profile')
```

[5]:

```
%%px
from mpi4py import MPI
import random
import time
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
print(f"rank = {rank}")
```

%px: 100% 8/8 [00:01<00:00, 6.60tasks/s]

```
[stdout:3] rank = 3
[stdout:5] rank = 5
[stdout:2] rank = 2
[stdout:4] rank = 4
[stdout:6] rank = 6
[stdout:7] rank = 7
[stdout:0] rank = 0
```

Providing interactive parallel programming environment and running large-scale real-time applications immediately

Improved Interactivity and Throughput

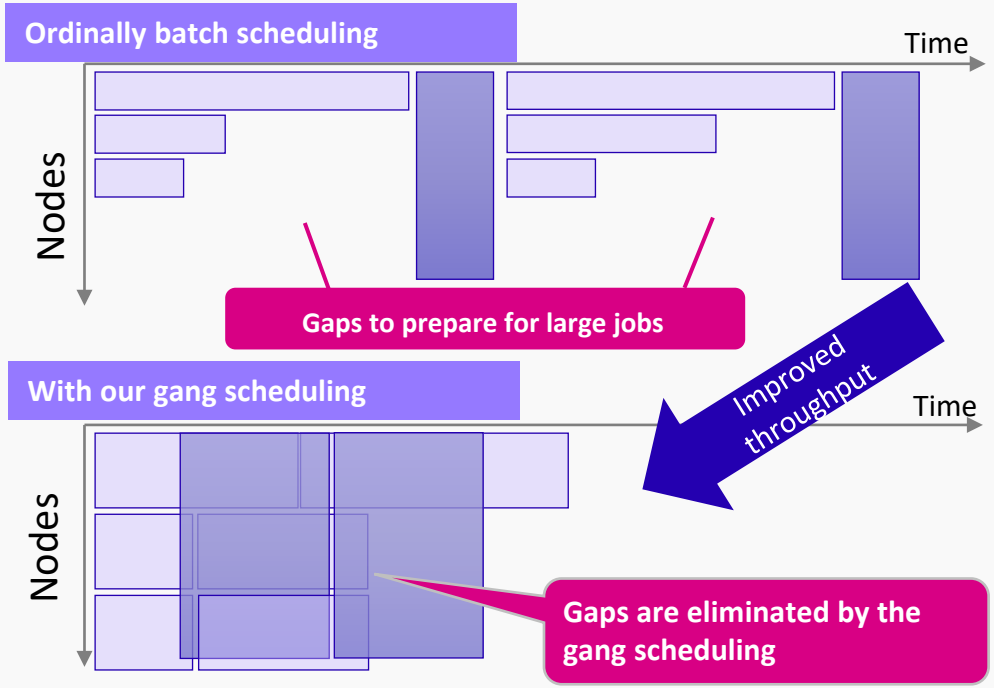
FUJITSU

- We have already deployed the mechanism on the actual system

- Successfully eliminated gaps caused by large jobs

- We confirmed
  - Shortened waiting time
  - Job throughput acceleration

on our on-premise cluster



AI Computing Broker

AI Computing Broker

FUJITSU

Realize more efficient use of GPU in AI learning process

Challenges in using GPU

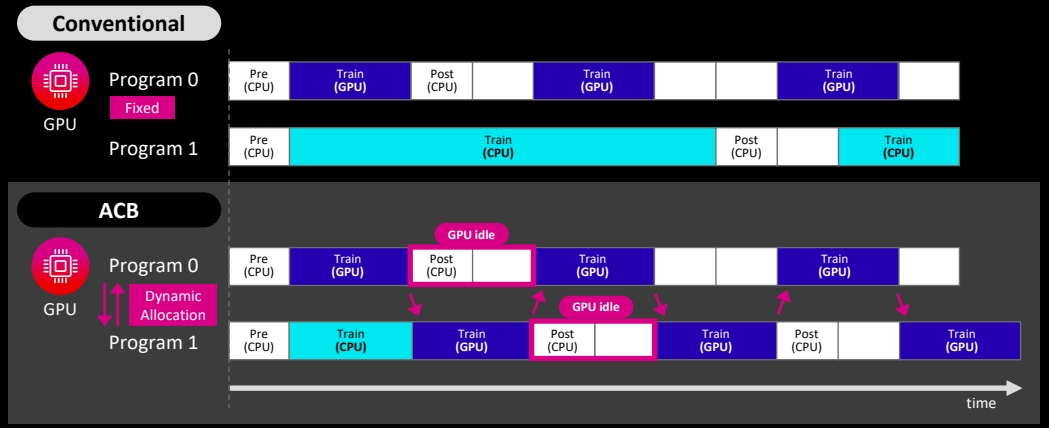
- Since the GPU is allocated on a per-program basis, even if the GPU idle time is large, other programs cannot use the GPU until the program ends.
- When virtualizing the GPU to share it among multiple programs, the GPU memory is divided for each program, reducing the capacity per program.

Features of AI Computing Broker

- GPU scheduling specialized for deep learning
- Control of GPU allocation at the processing unit level within the program
- Improves GPU utilization rate, reducing overall processing time
- More processes can be executed with the same GPU resources
- Unlike sharing of GPUs through virtualization technology, the program can fully utilize the GPU-equipped memory

Technology of AI Computing Broker

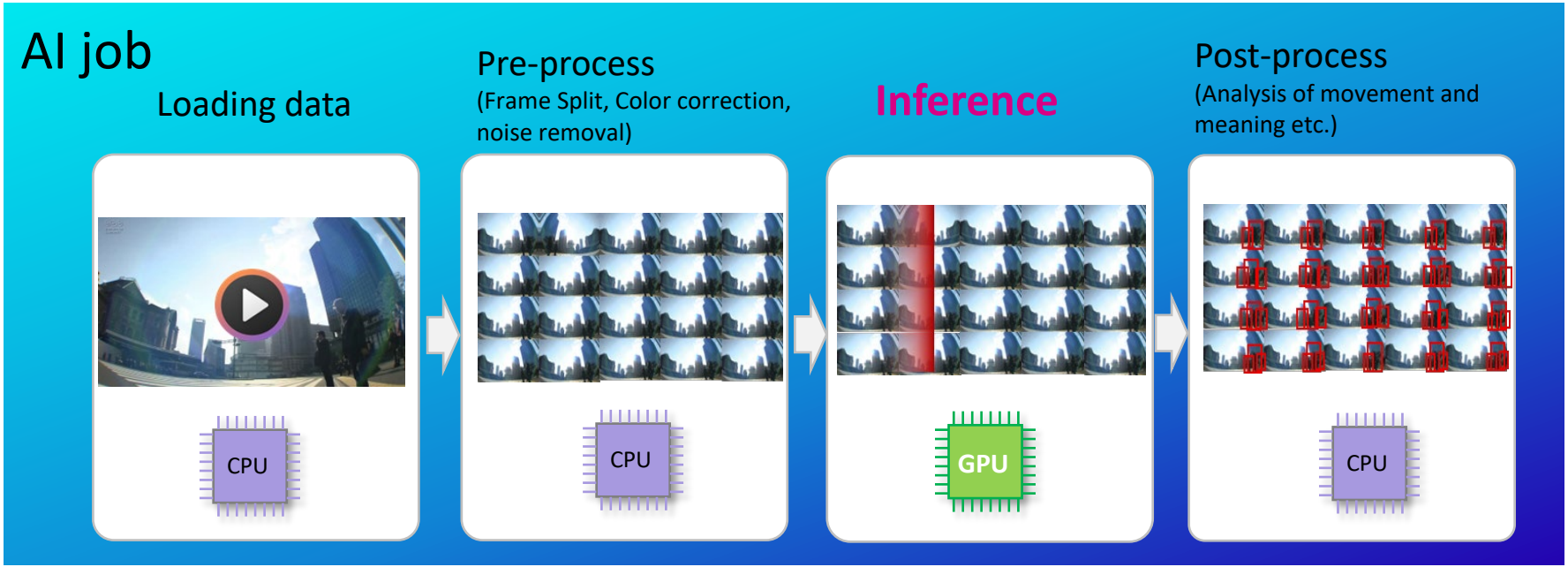
Analyze AI processing content, dynamically allocate GPU to processes that require GPU



One of the causes of low GPU utilization

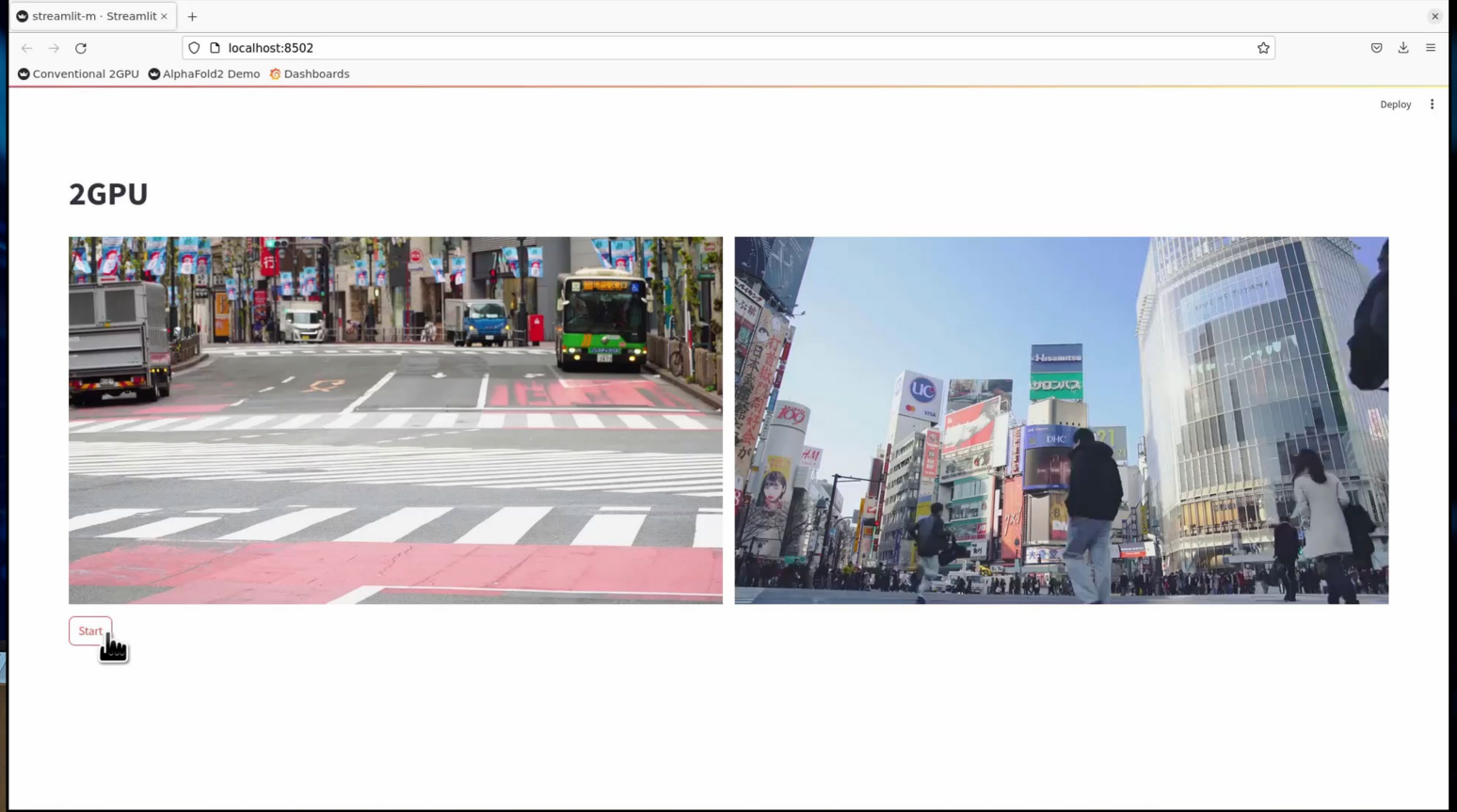
FUJITSU

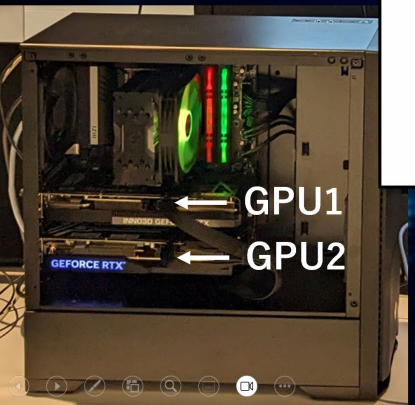
AI jobs do not always require dedicated GPUs



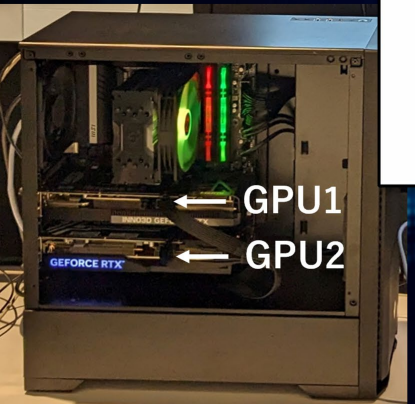
ACB detects GPU utilization phases and dynamically allocates GPU(s)

2 GPU


streamlit-m · Streamlit
localhost:8502
Conventional 2GPU
AlphaFold2 Demo
Dashboards
Deploy
2GPU
Start

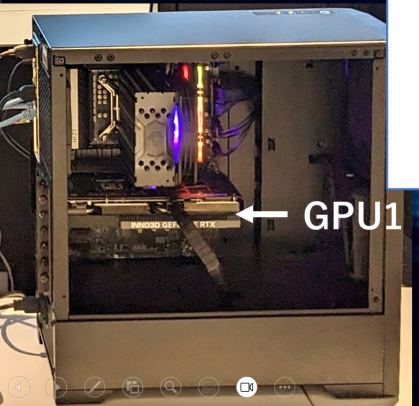

GPU1
GPU2
GEFORCE RTX

2 GPU


GPU1
GPU2
GEFORCE RTX


streamlit-m · Streamlit
localhost:8502
Conventional 2GPU
AlphaFold2 Demo
Dashboards
RUNNING...
Stop
Deploy
2GPU
20.33 (fps)
20.35 (fps)
Start

1 GPU
(w/o ACB)



streamlit-m · Streamlit × | streamlit-m · Streamlit ×

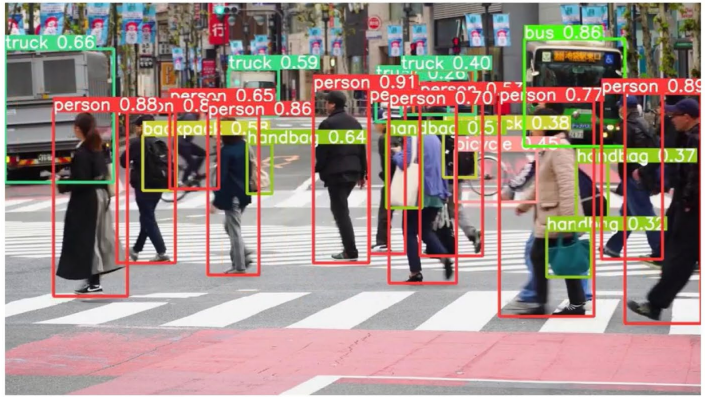
localhost:8501 100%

Conventional 1GPU | AI Computing Broke... | AlphaFold2 Demo | Dashboards

RUNNING... Stop Deploy

1GPU

20.10 (fps)

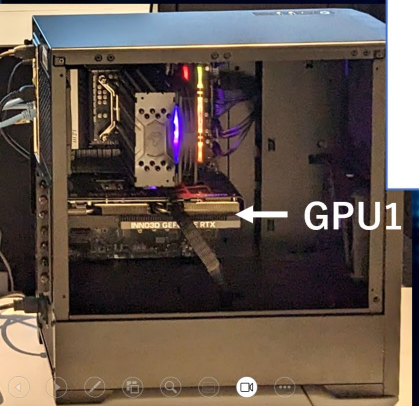


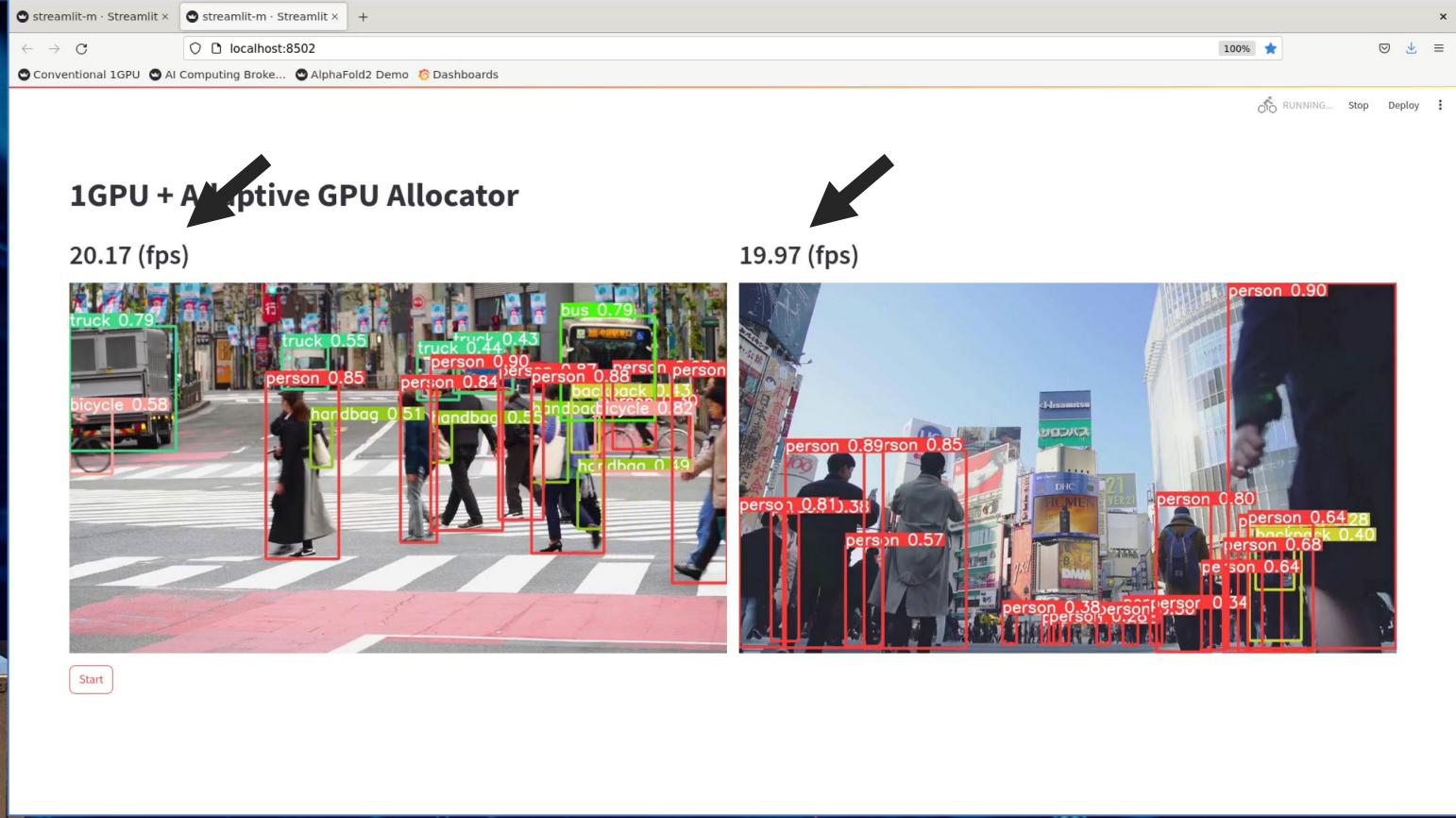
7.56 (fps)

Start

1 GPU + ACB

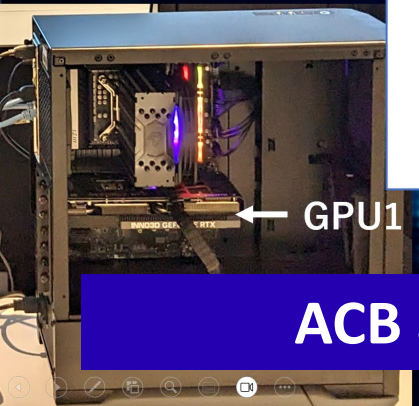
with ACB

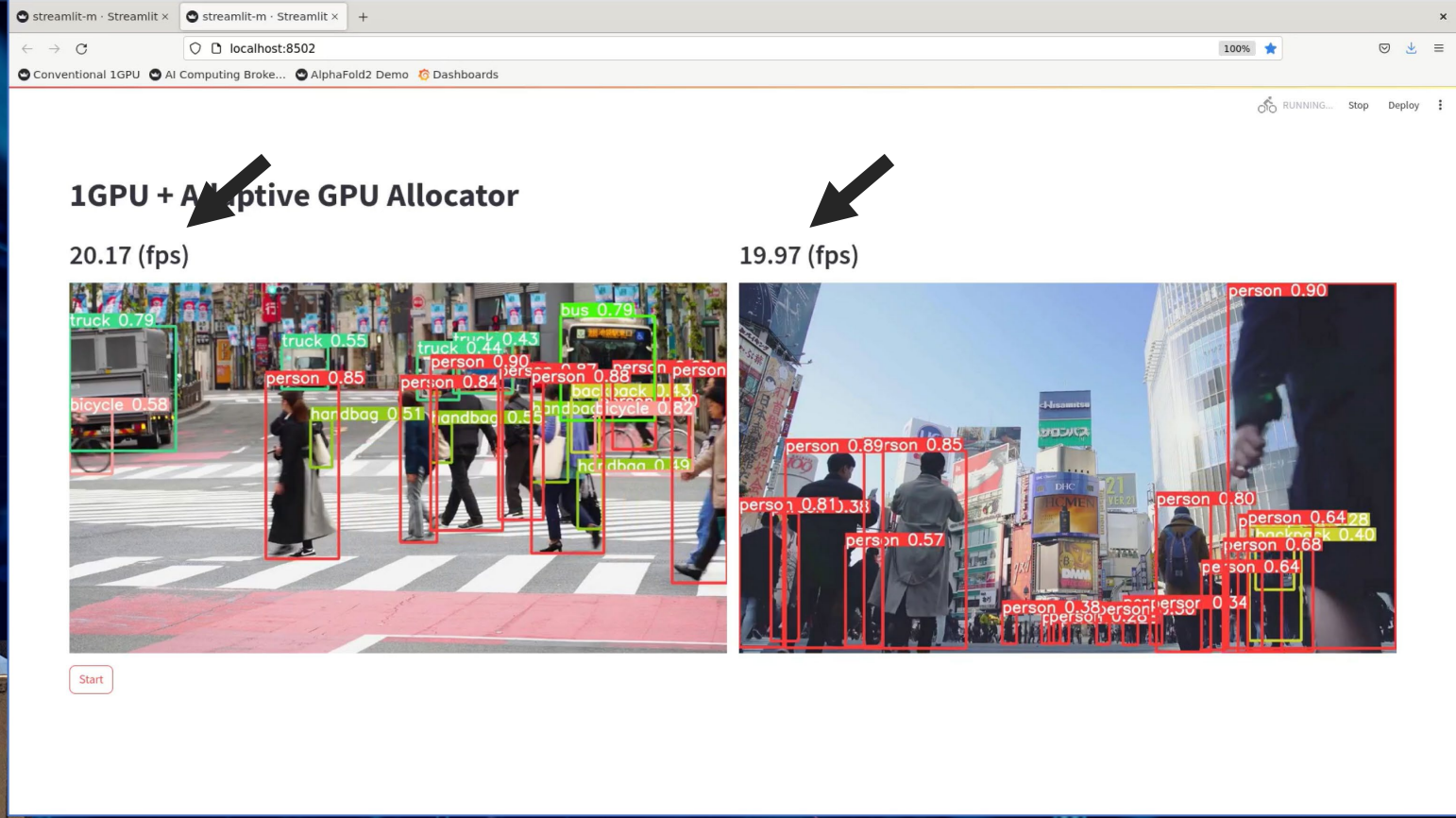




1 GPU + ACB

with ACB





ACB achieved 2-GPU performance using just 1 GPU

Conclusion

Conclusion

FUJITSU

- We developed two technologies to improve **interactivity** and **efficiency** on HPC systems
  - Scalable Fine-grained gang scheduling
  - AI Computing Broker - Adaptive accelerator allocation

- We have already started deploying to real systems.

- **If you're interested, please reach out at Booth #H30**.

Thank you