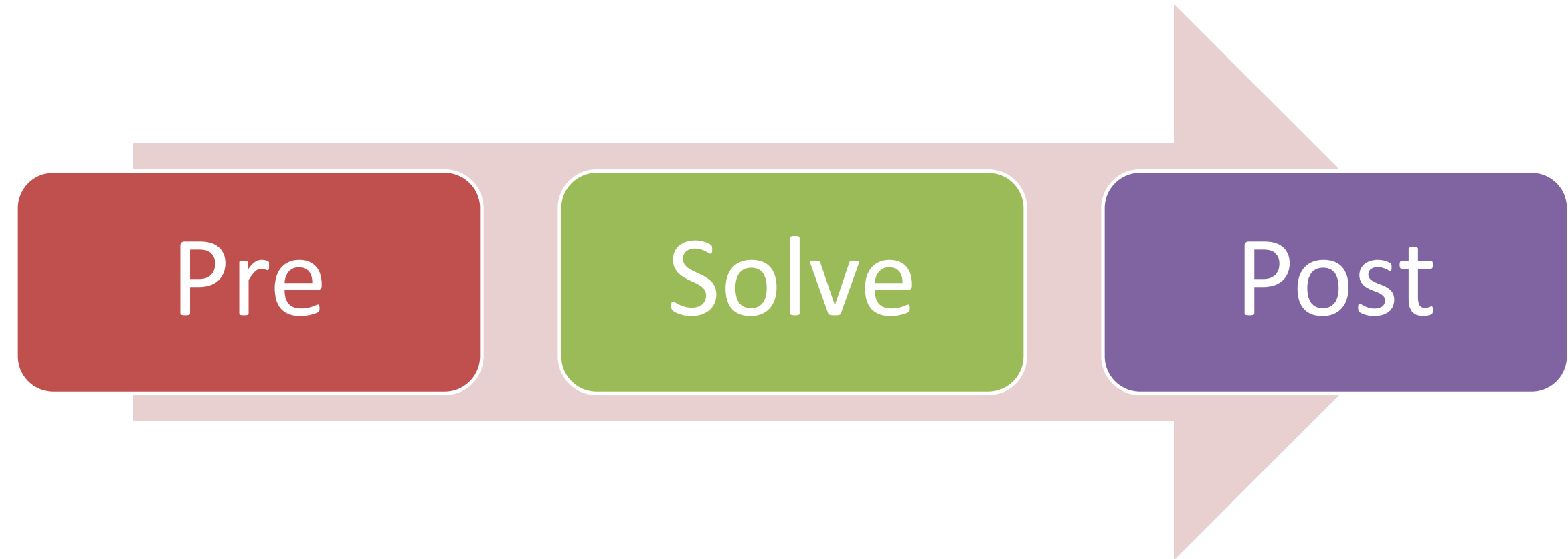


02 구성

절차



2. 클러스터 구성



Pre-processing (전처리 단계)	Solve (해석 단계)	Post-processing (후처리 단계)
<p>시뮬레이션 실행을 위한 입력 데이터를 준비하는 과정으로, 다음이 포함</p> <ul style="list-style-type: none"> • 기하 모델 생성 또는 불러오기 (CAD) • 메시(mesh) 생성 및 품질 확인 • 경계 조건 및 초기 조건 정의 • 재료 속성 정의 • 수치 해석 방법 선택 (예: FVM, FEM, FDM) • 솔버 설정 저장 	<p>지정된 수치 해석 방법을 기반으로 수치 방정식을 해석 (계산)하여 물리적 현상 (유동, 열, 구조 등)을 모사하는 핵심 단계</p>	<p>해석 결과 데이터를 시각화하고, 물리적 해석을 수행하여 의미 있는 정보를 도출하는 과정</p>

2. 클러스터 구성

Pre

- 메시 품질이 전체 해석 정확도에 큰 영향
- 너무 세밀한 메시: Solve 시간 급증, 너무 거친 메시: 정확도 손실
- CAD → Mesh로 변환 시 geometry 오류 (non-manifold, open surface 등) 에 유의

항목	설명
주 사용 자원	CPU single-thread, 메모리, 디스크 I/O
주요 소프트웨어	ANSYS Meshing, ICEM CFD, Pointwise, Gmsh 등
처리 시간	복잡한 형상, 정밀 메시일수록 지수적으로 증가
사용자 개입	직관적 UI 기반 수작업이 많음

- 자동 메시 생성 (Auto-meshing, Hex-dominant) 알고리즘 발전
- AI 기반 전처리 최적화 도입 (geometry cleaning, mesh adaptation)
- GPU 기반 전처리 툴 도입 (Ex: ANSYS Discovery)

2. 클러스터 구성

Solve

- 병렬화는 NUMA-aware, MPI rank mapping, I/O 최적화 필요
- Solver 안정성 확보 (시간 스텝, CFL 조건, 수렴 기준 설정 등)
- Check-pointing 및 장애 복구 전략 필요 (특히 장시간 해석 시)

항목	설명
주 사용 자원	CPU / GPU 병렬 계산, 대용량 메모리, MPI 통신
주요 솔버	Fluent, OpenFOAM, Abaqus, LS-DYNA, ANSYS Mechanical
처리 시간	수 시간 ~ 수 일 (복잡도, 도메인 크기, 병렬성에 따라 차이)
병렬화	HPC 환경에서 MPI + OpenMP, GPU Acceleration 필수

- GPU 기반 Solver 가속화 (CUDA, OpenACC, HIP) → 10배 이상 속도 향상 가능
- Low precision / Mixed precision Solver 도입 → HPL-AI와 유사
- AI + Physics-Informed Neural Networks (PINNs) 융합 Solver 연구

2. 클러스터 구성

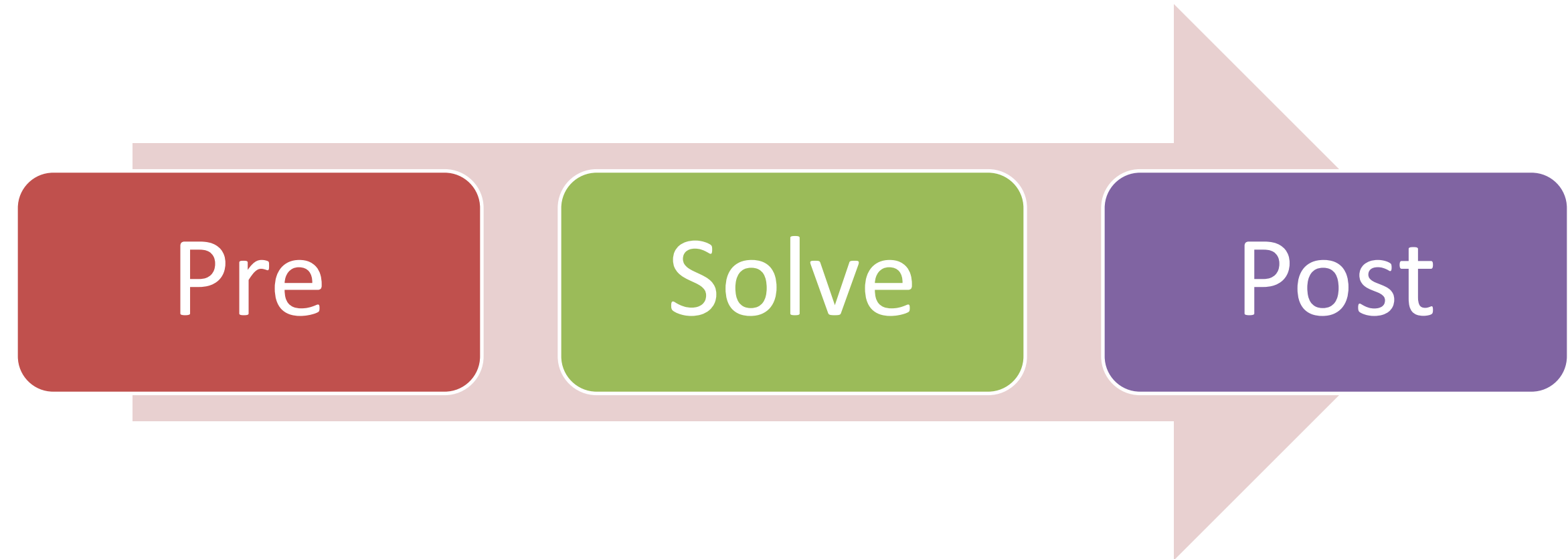
Post

- I/O 병목 → Lustre, BeeGFS 등 고성능 병렬 파일 시스템 필요
- 고해상도 시각화 시 VNC + GPU VDI 환경 필요
- 원격 접속 환경시 압축, down-sampling 기술 적용 필요

항목	설명
주 사용 자원	GPU (시각화), I/O bandwidth, 디스크 저장소
주요 툴	Paraview, Tecplot, ANSYS CFD-Post, Ensight
데이터 크기	수백 GB ~ 수 TB 이상 (Large scale CFD/FEA 시)
병렬화	일부 도구는 병렬 Post 지원 (ex: ParaView-MPI)

- Remote GPU Visualization (TurboVNC, NICE DCV) 증가
- Web 기반 Post (Ex: ParaviewWeb, Plotly Dash) 활용
- AI 기반 결과 분석 자동화 (특정 조건 이상 탐지, 특징 추출 등)

2. 클러스터 구성



서버 자원 배분을 각 단계별로 적절히 다르게 설정이 필요

- Pre: GUI + 메모리 많은 노드
 - Solve: HPC 병렬 노드, Infiniband, GPU
 - Post: GPU 시각화 노드 + 병렬 파일 시스템
-
- 워크플로우 전체 최적화는 단순히 Solve만 빠르게 하는 것보다, Pre → Solve → Post 전체 파이프라인을 고려해야 효과적
 - 대규모 프로젝트일수록 Pre/Post에서도 HPC 리소스를 적절히 분산/자동화하는 것이 중요

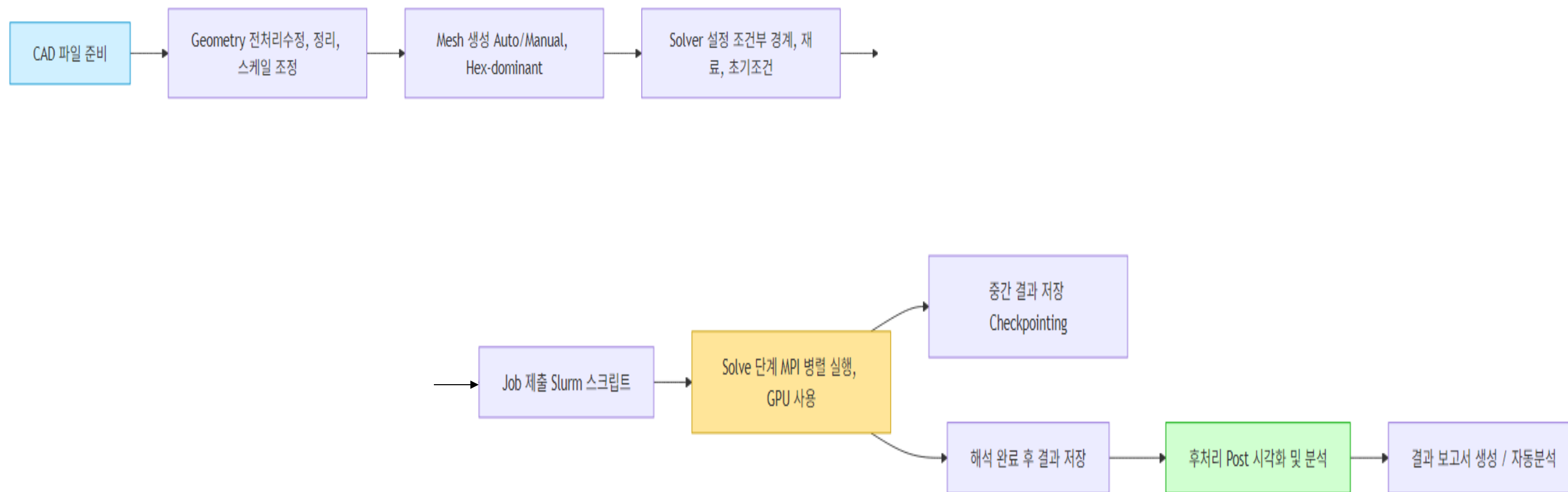
2. 클러스터 구성

단계	최적화 포인트	세부 전략	활용 기술/툴
Pre (전처리)	메시 품질/시간 최적화	- Geometry 오류 자동 탐지 및 수정- 메시 크기/형상 기반 자동 분할 전략 적용- Target QoI(Quality of Interest) 기반 메시 해상도 조정	ANSYS Discovery, SimScale AI Mesh, snappyHexMesh
	반복 작업 자동화	- CAD import → 메시 → 설정 자동화- 변수 스왑 자동 스크립팅	Workbench Journal, Python Script
	자원 매칭	- GUI 가속 가능한 GPU 노드 활용- RAM 용량이 큰 노드 선별	Slurm Partition 설정, VDI 환경
Solve (계산)	병렬 효율 극대화	- NUMA-aware MPI Rank Mapping- GPU-aware 스케줄링 및 Load Balancing- Dynamic Mesh Load Balancing	hwloc, OpenMPI, Slurm GRES
	중단 대비 전략	- Checkpoint 자동화- 장애 복구 자동 롤백	--checkpoint-interval, hpcctool, Revolve
	Mixed Precision 활용	- Solver 내부에서 double→mixed precision 전환- HPL-AI, PINN 적용	AMGX, TensorFlow PINN, OpenACC
	IO 병목 제거	- Compute/IO 분리 스케줄링- 병렬 파일 시스템에 직접 쓰기	Lustre, BeeGFS, ADIOS2
Post (후처리)	대용량 데이터 처리	- 해석 결과 압축 or 샘플링- Domain별 Subdomain Post- 결과 자동 필터링 및 추출	ParaView Batch, OpenFOAM FunctionObject
	시각화 환경 최적화	- TurboVNC + GPU + Remote Display- Web-based Visualization (ParaViewWeb)	NICE DCV, ParaviewWeb, Plotly Dash
	결과 해석 자동화	- 조건 이상 검출, 특징 추출 자동화- AI 기반 패턴 탐지, 유사 사례 비교	PyTorch, Scikit-learn, SHAP/LIME, Yolo+Post

2. 클러스터 구성

단계	최적화 포인트	세부 전략	활용 기술/툴
Workflow 전체	통합 파이프라인 자동화	- Pre→Solve→Post 순차 수행 자동화- 결과 폴더 자동 생성 및 관리	Snakemake, Airflow, bash pipeline
	자원 최적 분배	- 단계별 파티션 구분 (Pre용, Solve용, Post용)- Solve 집중 시간에 Pre/Post 비동기 병렬 처리	Slurm multi-step job script, Backfill 활용
	실시간 모니터링	- Job 상태/노드 부하 시각화- 실패 시 즉시 재시작 설정	Grafana, Slurm sreport/squeue, Slack/Telegram 알림봇

2. 클러스터 구성



2. 클러스터 구성

The screenshot displays the ANSYS Workbench interface for an "Unsaved Project - Workbench". The Project Schematic shows a workflow with components A through H:

- Component A:** Geometry (1 instance)
- Component B:** Mesh (1 instance)
- Component C:** Fluent (1 instance)
- Component D:** Results (1 instance)
- Component E:** Geometry (2 instances)
- Component F:** Mesh type 1 (2 instances)
- Component G:** Mesh type 2 (2 instances)
- Component H:** Mesh type 3 (2 instances)

Connections are shown between components: A to B, B to C, C to D, E to F, F to G, and G to H. The Properties of Schematic A2: Geometry panel is open on the right, showing various options for geometry and meshing.

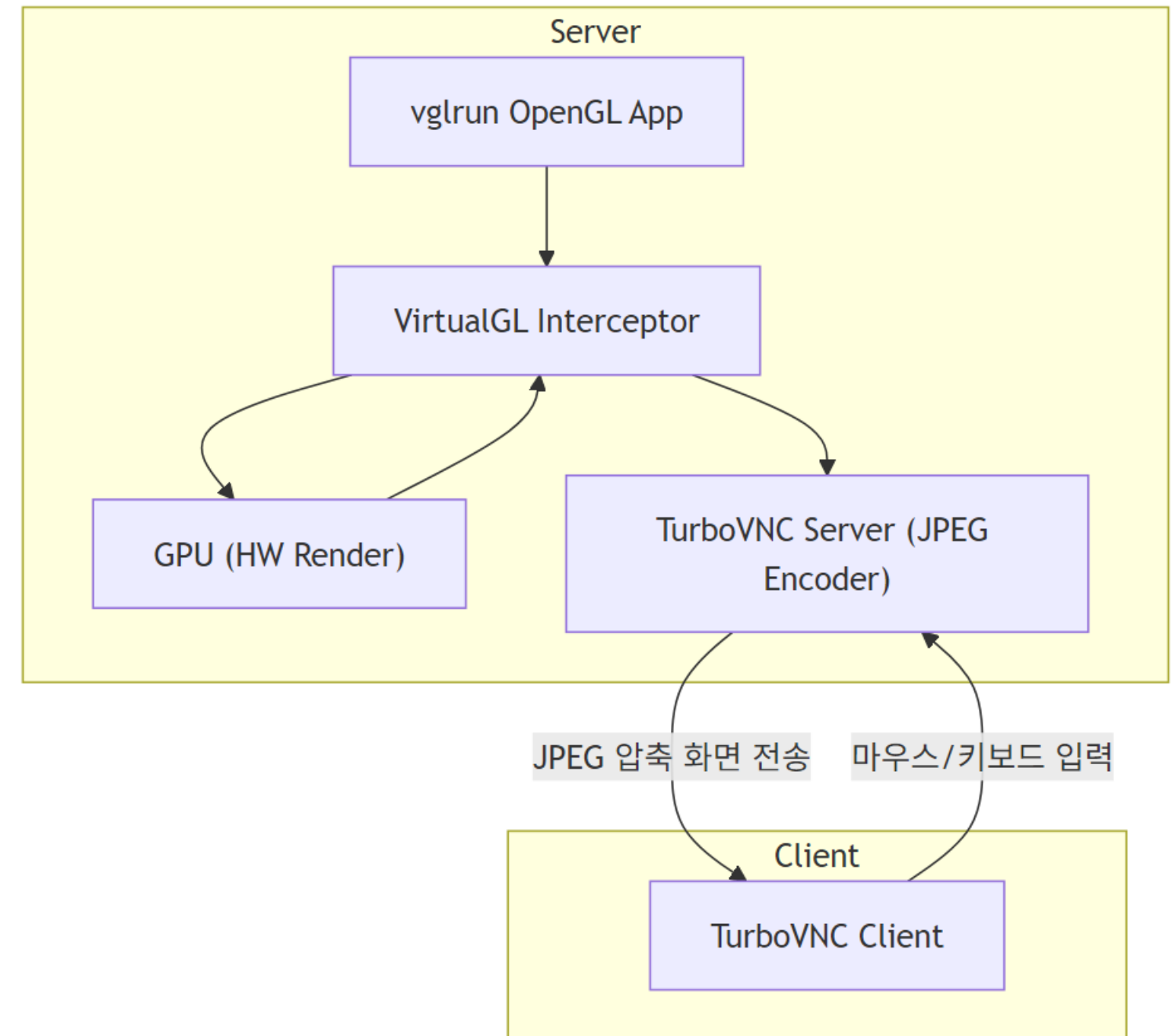
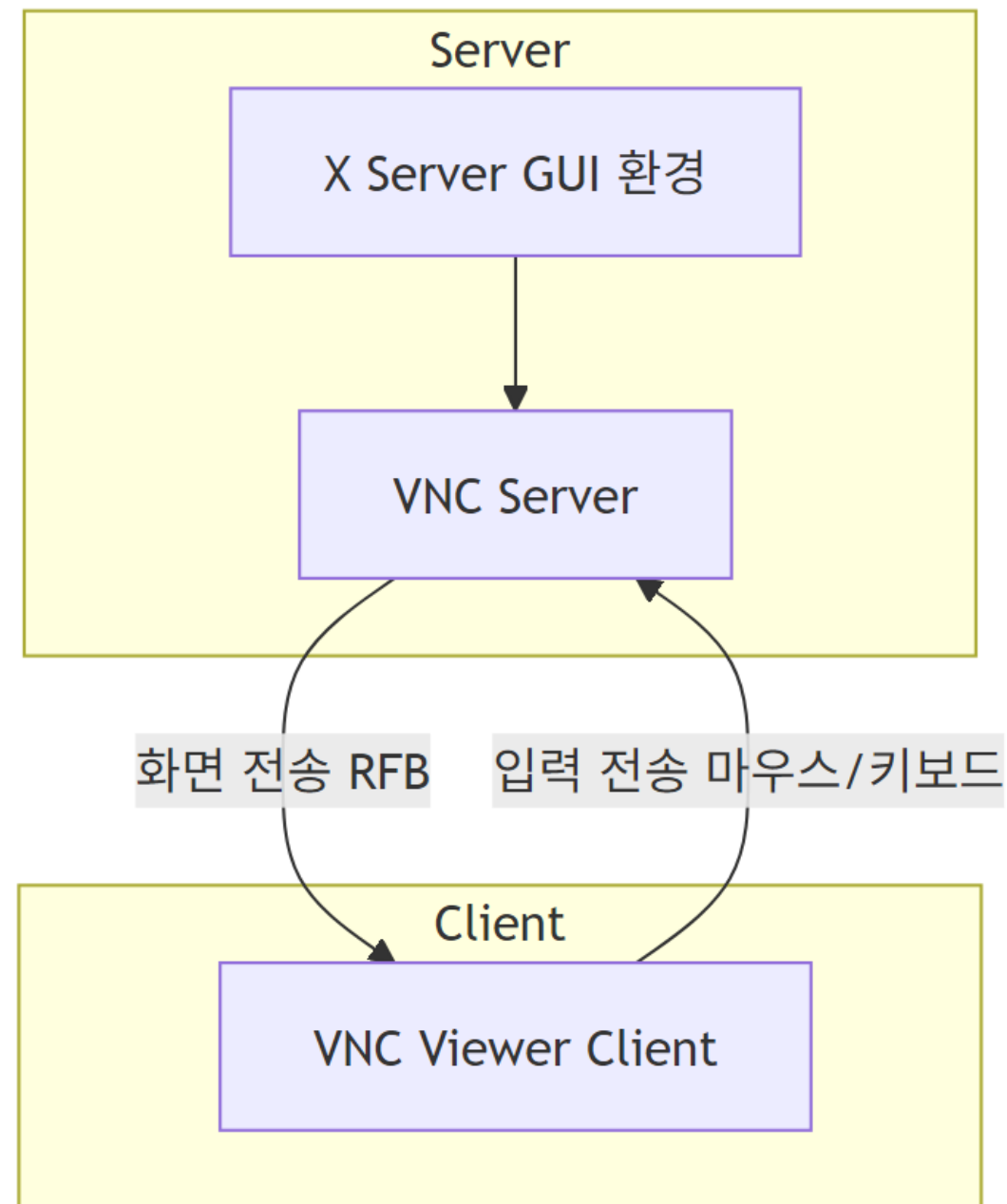
	A	B
1	Property	Value
2	General	
3	Component ID	Geometry
4	Directory Name	Geom
5	Notes	
6	Notes	
7	Used Licenses	
8	Last Update Used Licenses	
9	Basic Geometry Options	
10	Solid Bodies	<input checked="" type="checkbox"/>
11	Surface Bodies	<input checked="" type="checkbox"/>
12	Line Bodies	<input type="checkbox"/>
13	Parameters	Independent
14	Parameter Key	ANS;DS
15	Attributes	<input type="checkbox"/>
16	Named Selections	<input type="checkbox"/>
17	Material Properties	<input type="checkbox"/>
18	Advanced Geometry Options	
19	Analysis Type	3D
20	Use Associativity	<input checked="" type="checkbox"/>
21	Import Coordinate Systems	<input type="checkbox"/>
22	Import Work Points	<input type="checkbox"/>
23	Reader Mode Saves Updated File	<input type="checkbox"/>
24	Import Using Instances	<input checked="" type="checkbox"/>
25	Smart CAD Update	<input checked="" type="checkbox"/>
26	Compare Parts On Update	No
27	Enclosure and Symmetry Processing	<input checked="" type="checkbox"/>
28	Decompose Disjoint Geometry	<input checked="" type="checkbox"/>
29	Clean Geometry On Import	<input type="checkbox"/>
30	Stitch Surfaces On Import	None
31	Mixed Import Resolution	None
32	Import Facet Quality	Source

2. 클러스터 구성

솔루션	주요 영역	통합 여부	특징
ANSYS Workbench	멀티피직스	전체 통합	산업 표준, 모듈화, GUI 강력
COMSOL Multiphysics	멀티피직스	전체 통합	수식 기반 멀티피직스 커스터마이징 강점
Altair HyperWorks	구조 / 유동 / EM	대부분 통합	HyperMesh, OptiStruct, AcuSolve 등 연계
Autodesk Fusion 360	CAD + CAE	통합	소형 제품 설계 및 클라우드 해석에 적합
Simcenter (Siemens)	CFD / 구조 / NVH	전체 통합	NX와 연동, 고급 음향/진동 해석 가능
SolidWorks Simulation	CAD 기반 FEA	제한적 통합	설계-해석 연계 중심, 범용성 제한
OpenFOAM + ParaView + GMSH	오픈소스 CFD	× 별도 도구 통합 필요	유연하지만 통합 UI는 아님
LS-DYNA + LS-PrePost	충돌, 고속해석	부분 통합	전처리/후처리 가능, 워크플로우는 분리됨

2. 클러스터 구성

VNC (Virtual Network Computing)



2. 클러스터 구성

VNC (Virtual Network Computing)

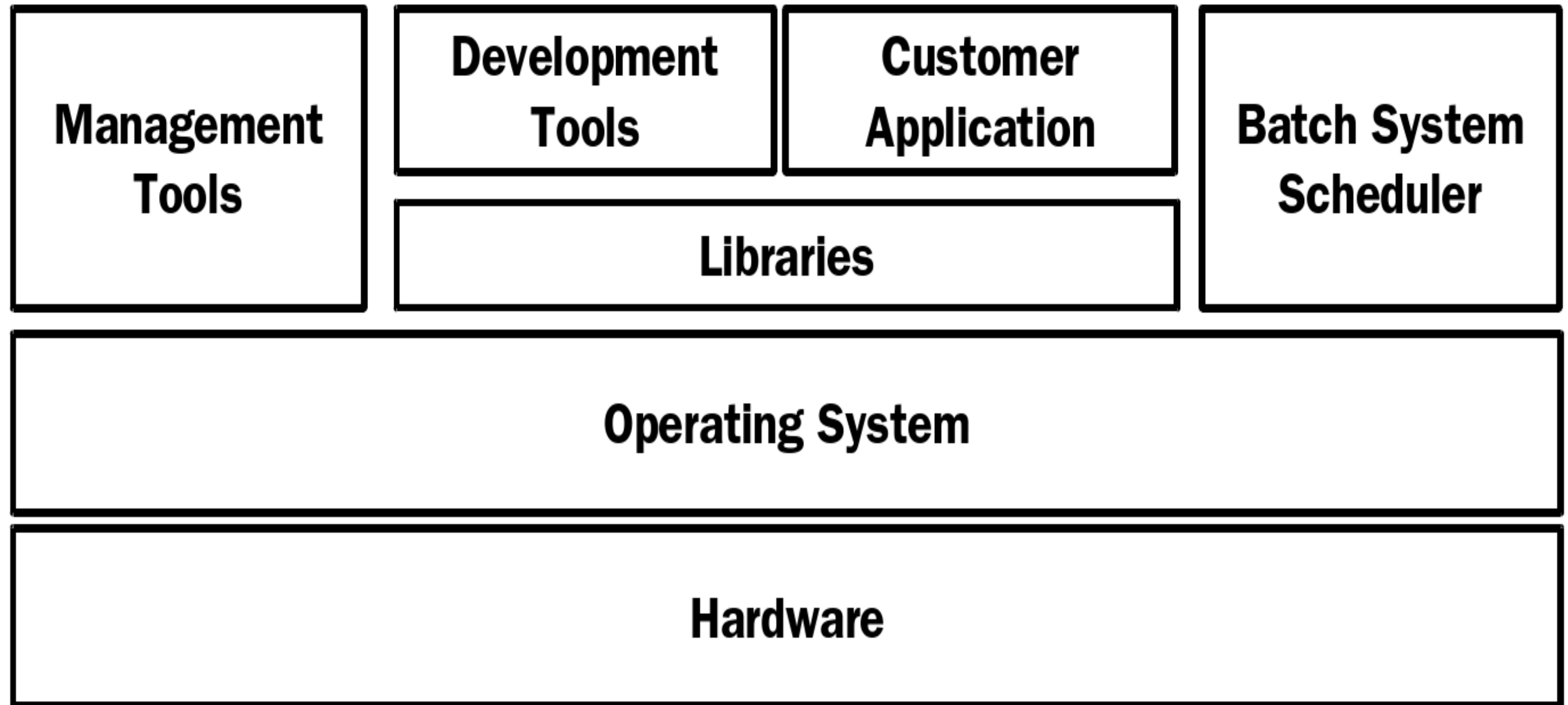
비교 항목	TigerVNC (또는 일반 VNC)	TurboVNC + VirtualGL
렌더링	CPU 기반 소프트웨어	GPU 하드웨어
압축 방식	Tight, RFB tile 방식	SIMD JPEG, 빠른 병렬 압축
응답성	일반 GUI용, 가벼운 작업	3D/영상/시뮬레이션용, 고성능
벤치마크	느리고 밀리초~초 단위 lag	50 MP/sec 이상 전송 속도 가능
사용처	데스크탑 원격 제어	CAD, Paraview, HPC, 시뮬레이션 원격

2. 클러스터 구성

RDP(Remote Desktop Protoco) Vs VNC (Virtual Network Computing)

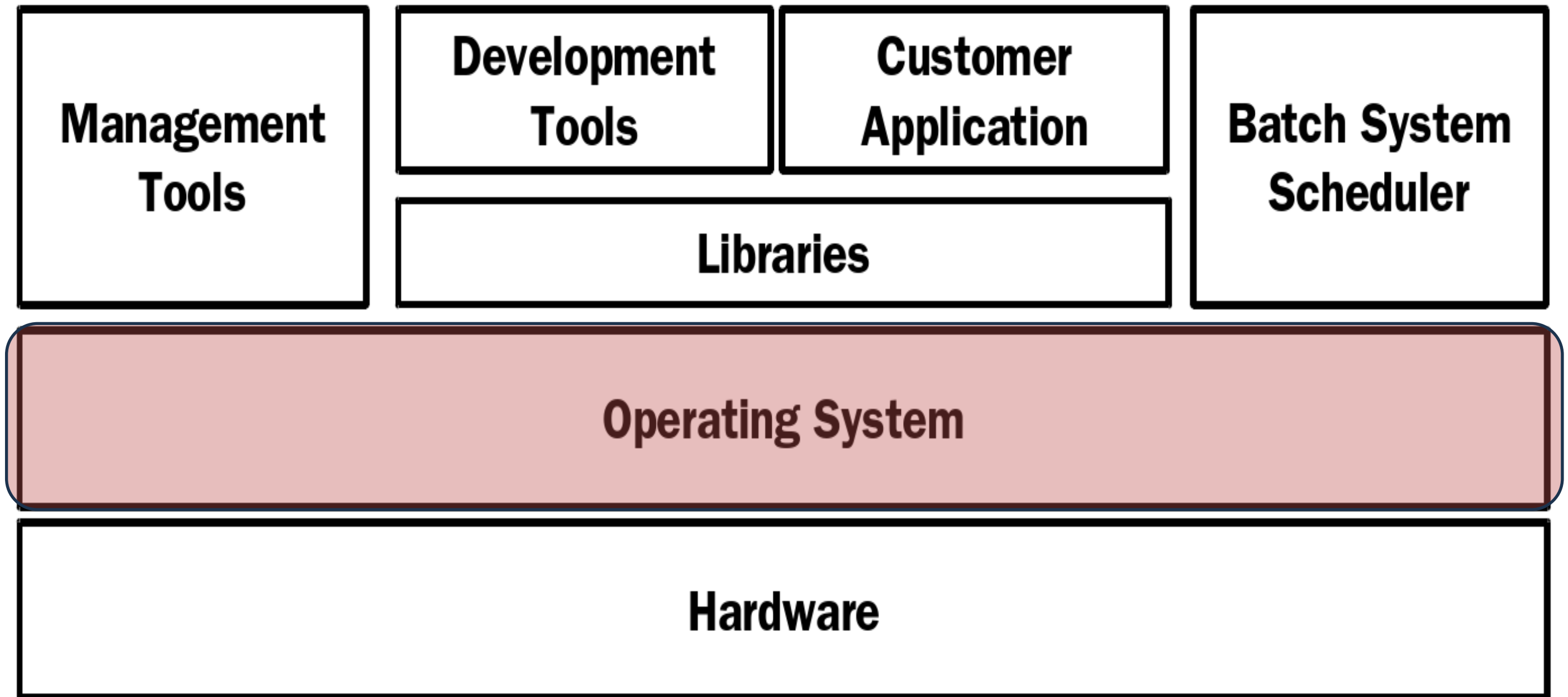
항목	RDP	TurboVNC + VirtualGL
OpenGL 지원	1.1 SW 에뮬레이션	GPU HW 가속 (NVIDIA, AMD 모두 지원)
렌더링 위치	서버 CPU 또는 GDI API	서버 GPU 직접 사용
적합한 앱	단순 GUI (Excel, notepad)	OpenGL 기반 GUI (StarCCM+, Ansys, Paraview 등)
창 깨짐 현상	발생 빈번	없음 (Native 렌더링)

2. 클러스터 구성



2. 클러스터 구성

Operation System



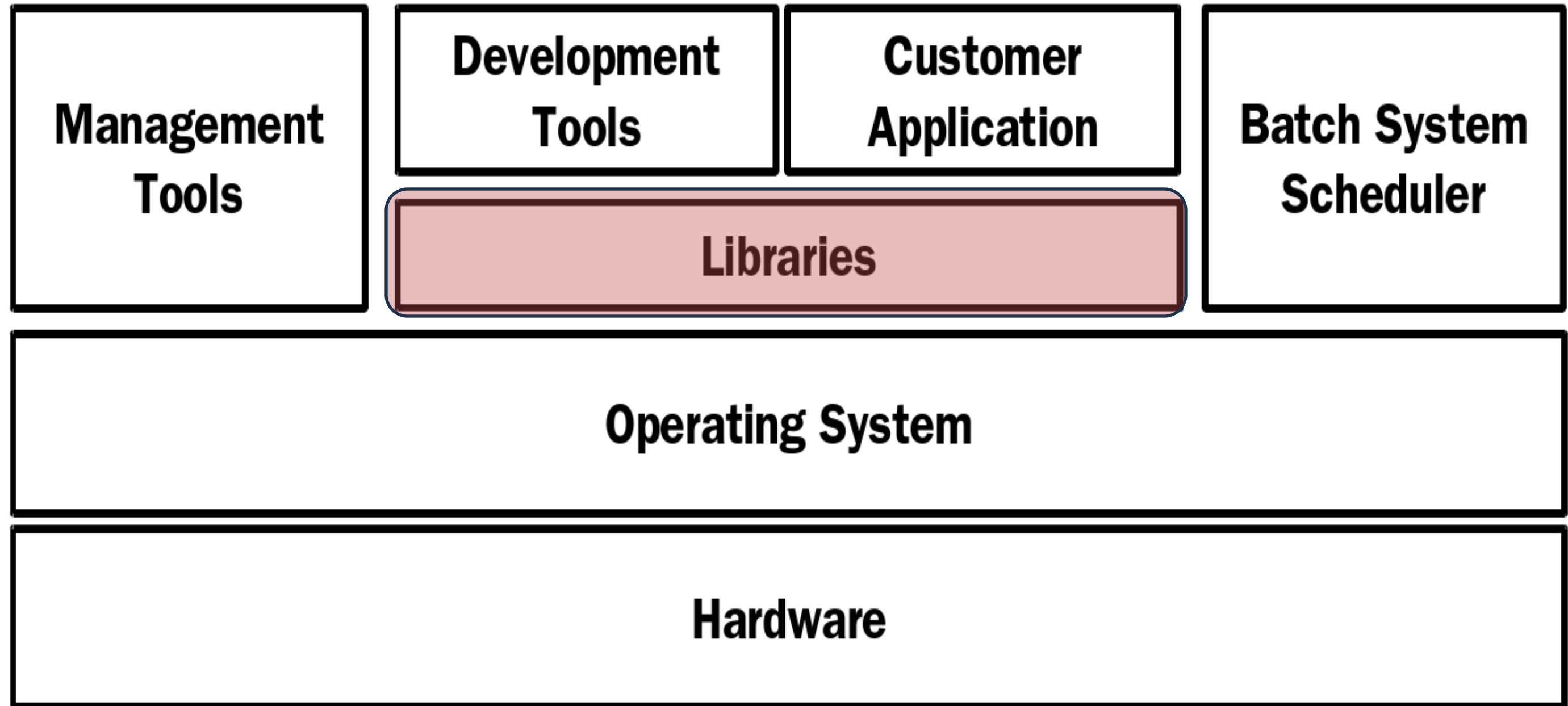
2. 클러스터 구성

Operation System



2. 클러스터 구성

Libraries



2. 클러스터 구성

MPI (Message Passing Interface)

Main Features

- MPI는 standard specification을 규정한 후에 구현이 이뤄지는 체계를 가짐
- MPI는 MPP(massively parallel processor)와 workstation cluster상에서의 고성능을 지원 하기 위하여 디자인 됨
- (주: PVM이 초기에는 workstation cluster를 지원하게 만들어졌다면, MPI는 주로MPP Vendor에 의하여 주도적으로 만들어진 스펙으로 워크스테이션 보다는 MPP에 중점적으로 그 스펙이 만들어졌다.)
- MPI는 널리 사용되고 있으며 공개버전과 Vendor에 의해 제공되는 버전이 존재하며 수 많은 MPI 홈페이지를 참조 할 수 있음
- MPI는 광범위한 업체 및 연구소, 다양한 구현자 및 사용자의 모임에서 만들어 졌음

Configuration

- MPP Vendor specific 구현, Workstation based 구현 등 다양한 구현이 존재하며, 그 구현 방식에 따라 천차만별이다. 최근 메시지 패싱 라이브러리 중 하나인 P4를 기반으로 구현된 MPICH가 널리 사용되고 있는 편이다.

2. 클러스터 구성

PVM (Parallel Virtual Machine)

Main Features

- H/W architecture, data format, computing speed, machine's load, network traffic load 가 각각 틀린 컴퓨터에서 수행 가능하며, 현재 Win32에서도 수행 가능
- 특정 OS나 H/W의 특별한 Feature를 사용하지 않고 구현하여 C Source형태로 배포 되므로 새로운 기종에 포팅 가능
- Run-time에 최대(4096-1)개(1개는 daemon)의 host를 VM에 추가 할 수 있으며 이론적으로 한 Host당 218-1개의 태스크를 수행 할 수 있음
- Host나 network의 failure 발생시 자동 복구하지 않으나, 복구하는 프로그램을 작성 할 수 있는 인터페이스 제공(polling/notification primitives)

Configuration

- PVM 시스템은 각 시스템에서 수행되는 daemon들의 연결로 구성되며, 사용자 task는 PVM Library 루틴을 사용하여 daemon에 접속하여 VM을 구성 할 수 있음

2. 클러스터 구성

MPI 병렬 프로그램

Node :

정의

- Node는 물리적으로 하나의 서버, 머신, 호스트입니다.
→ 예: 1대의 물리 서버, 1대의 가상머신(VM), 클라우드 인스턴스 한 대

특징

- 하나의 Node 안에는 여러 CPU 소켓, 코어, 메모리, GPU가 있을 수 있음
- 여러 Node를 묶어 하나의 HPC 클러스터를 만듦

Rack :

정의

- Rank는 MPI 프로그램에서 실행되는 하나의 프로세스(process)를 나타냅니다.
→ 즉, 동일한 프로그램을 실행하는 각각의 독립적인 프로세스
- Rank는 전 세계적으로 고유한 ID 번호(0 ~ N-1) 를 갖습니다.
→ MPI_Comm_rank 로 확인 가능

특징

- Rank는 mpirun 또는 mpiexec 할 때 -np 옵션으로 몇 개 띄울지 결정
예: mpirun -np 8 ./myapp → Rank 0~7
- 하나의 Node 안에 여러 Rank가 있을 수 있음
예: Node 1대에 Rank 0~3 (4개 프로세스)
- MPI 프로그램에서 모든 통신은 “Rank 간 통신”으로 정의됨
→ 누구(송신 Rank)가 누구(수신 Rank)에 뭉 보내는지

2. 클러스터 구성

MPI 병렬 프로그램

단계	설명
1. 초기화	MPI_Init : MPI 환경을 초기화
2. 프로세스 정보 얻기	MPI_Comm_size (전체 프로세스 수), MPI_Comm_rank (각 프로세스의 ID)
3. 통신 작업	MPI_Send, MPI_Recv 등
4. 종료	MPI_Finalize : MPI 환경 종료

가장 간단한 Hello World 예제

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    int world_size, world_rank;

    // 1. MPI 초기화
    MPI_Init(&argc, &argv);

    // 2. 전체 프로세스 개수
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // 3. 내 rank 얻기
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // 4. 출력
    printf("Hello world from rank %d out of %d processors\n", world_rank,
world_size);

    // 5. MPI 종료
    MPI_Finalize();

    return 0;
}
```

2. 클러스터 구성

MPI 병렬 프로그램

단계	설명
1. 초기화	MPI_Init : MPI 환경을 초기화
2. 프로세스 정보 얻기	MPI_Comm_size (전체 프로세스 수), MPI_Comm_rank (각 프로세스의 ID)
3. 통신 작업	MPI_Send, MPI_Recv 등
4. 종료	MPI_Finalize : MPI 환경 종료

통신 예제 (Send/Recv) : Rank 0 → Rank 1로 값 42를 전달

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    int rank, size;
    MPI_Status status;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int number;

    if (rank == 0) {
        number = 42;
        MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
        printf("Rank 0 sent number %d to Rank 1\n", number);
    } else if (rank == 1) {
        MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
        &status);
        printf("Rank 1 received number %d from Rank 0\n", number);
    }

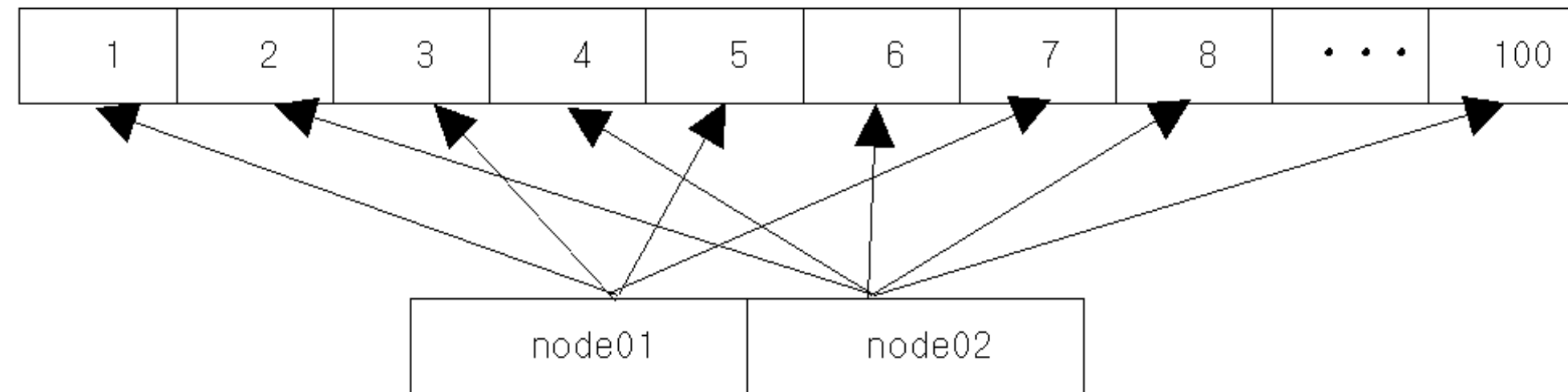
    MPI_Finalize();
    return 0;
}
```

2. 클러스터 구성

병렬 프로그램

SUM

다음 프로그램은 1부터 100까지 더하는 프로그램으로 Broadcast 연산과 Reduce 연산을 하도록 되어 있고 프로그램 흐름은 다음과 같다.



위 그림과 같이 node01 은 (1,3 5,7, 9..... 99)까지 합을 node02 는 (2,4,6,8,10 ... 100)가지의 합을 구하게 되어 이를 합하게 되는 구조이다.

2. 클러스터 구성

----- sum.c Source -----

```
#include <stdio.h>
#include <sys/time.h>
#include "mpi.h"
```

```
static struct timeval time_value1;
static struct timeval time_value2;
```

```
#define START_TIMER gettimeofday (&time_value1, (struct timezone *) 0)
#define STOP_TIMER gettimeofday (&time_value2, (struct timezone *) 0)
#define ELAPSED_TIME ((double) ((time_value2.tv_usec - time_value1.tv_usec)*0.001 +  
((time_value2.tv_sec - time_value1.tv_sec) * 1000.0 )))
```

2. 클러스터 구성

```
int main (int argc, char *argv[])  
{  
    int done = 0;  
    int number = 0;  
    int sum;  
    int mysum = 0;  
    int iproc;  
    int nproc;  
  
    int i;
```

MPI_Init (&argc, &argv);

MPI_Comm_size (MPI_COMM_WORLD, &nproc);

MPI_Comm_rank (MPI_COMM_WORLD, &iproc);

2. 클러스터 구성

```
while (!done)
{
    if (iproc == 0)
    {
        if (number == 0) number = 100;
        else number = 0;

        START_TIMER;

    }
}
```

MPI_Bcast (&number, 1, MPI_INT, 0, MPI_COMM_WORLD);

2. 클러스터 구성

```
if (number == 0) done = 1;
else
{
    for (i = iproc + 1; i <= number; i+= nproc) mysum += i;
    MPI_Reduce (&mysum, &sum, 1, MPI_INTEGER, MPI_SUM, 0, MPI_COMM_WORLD);
    if (iproc == 0)
    {
        STOP_TIMER;
        printf ("sum is %d\n", sum);
        printf ("elapsed time is %.3fms\n", ELAPSED_TIME);
    }
}
}
MPI_Finalize ();
}
```

2. 클러스터 구성

실제 이 프로그램을 실행을 하여 보면 다음과 같다.

1 CPU 사용 시

```
[baron@node01 TEST]$ mpirun -np 1 SUM  
sum is 5050  
elapsed time is 0.061ms
```

2 CPU 사용 시

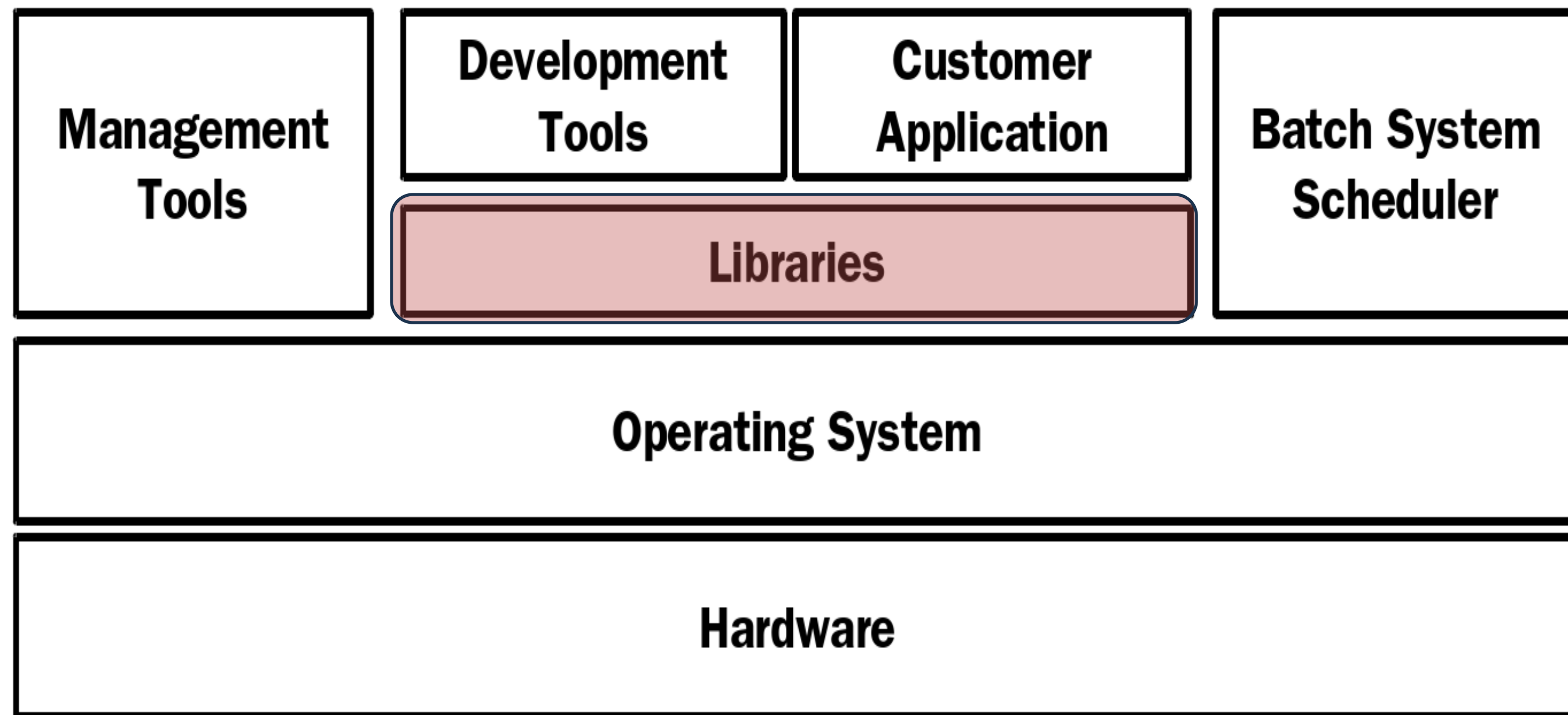
```
[baron@node01 TEST]$ mpirun -np 2 SUM  
sum is 5050  
elapsed time is 3.547ms
```

문제 :

병렬 처리에도 불구하고 이 결과가 나오는 이유는 무엇인가?

2. 클러스터 구성

Development Tools



2. 클러스터 구성

Development Tools

“The Free Lunch is Over”

2. 클러스터 구성

Development Tools

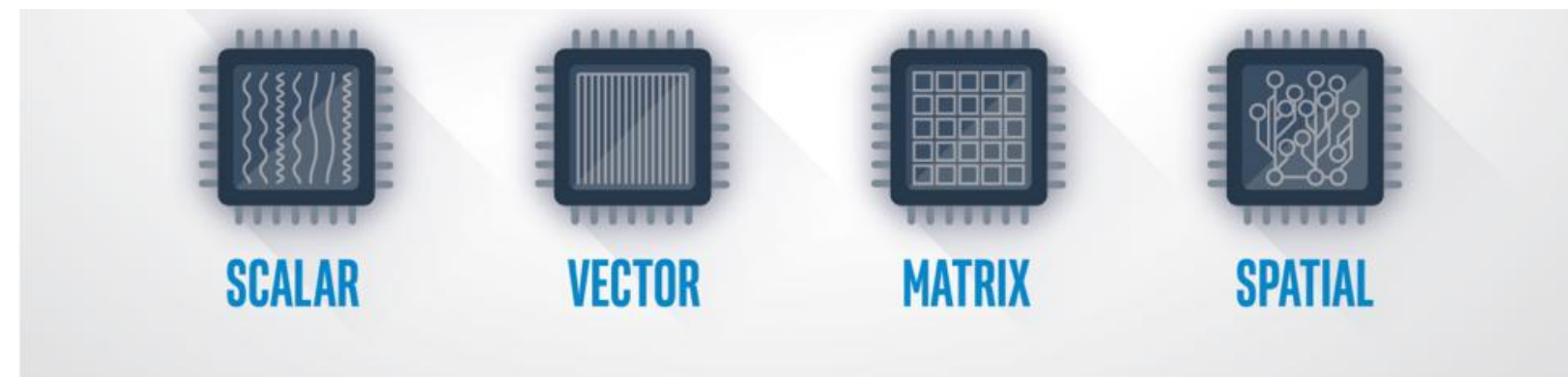
코드에 대한 호환성

코드에 대한 재 활용성

2. 클러스터 구성

Development Tools

Intel oneAPI



2. 클러스터 구성

Development Tools



Intel® One API Base Toolkit

개발자로 하여금 CPU, GPU와 FPGA 전반에 걸쳐 성능과 데이터 중심의 응용프로그램을 구축하고, 테스트, 배포할 수 있게 지원하는 이 기본 키트

- Intel® oneAPI Data Analytics Library
- Intel® oneAPI Deep Neural Networks Library
- Intel® oneAPI DPC++/C++ Compiler
- Intel® oneAPI DPC++ Library
- Intel® oneAPI Math Kernel Library
- Intel® oneAPI Threading Building Blocks
- Intel® oneAPI Video Processing Library
- Intel® Advisor
- Intel® Distribution for GDB*
- Intel® Distribution for Python*
- Intel® DPC++ Compatibility Tool
- Intel® FPGA Add-on for oneAPI Base Toolkit
- Intel® Integrated Performance Primitives
- Intel® VTune™ Profiler



Internet of Things

네트워크 에지(Network Edge)에서 실행되는 효율적이고 안정적인 고성능 솔루션을 구축

- Intel® oneAPI DPC++/C++ Compiler
- Intel® C++ Compiler Classic
- Intel® Inspector
- Eclipse* IDE
- IoT Connection Tools
- Linux* Kernel Build Tools



High-Performance Computing

확장이 가능한 빠른 C++, Fortran, OpenMP와 MPI 응용프로그램을 제공

- Intel® oneAPI DPC++/C++ Compiler
- Intel® oneAPI DPC++ Library
- Intel® Cluster Checker
- Intel® Fortran Compiler (Beta)
- Intel® Fortran Compiler Classic
- Intel® Inspector
- Intel® MPI Library
- Intel® Trace Analyzer and Collector



Rendering

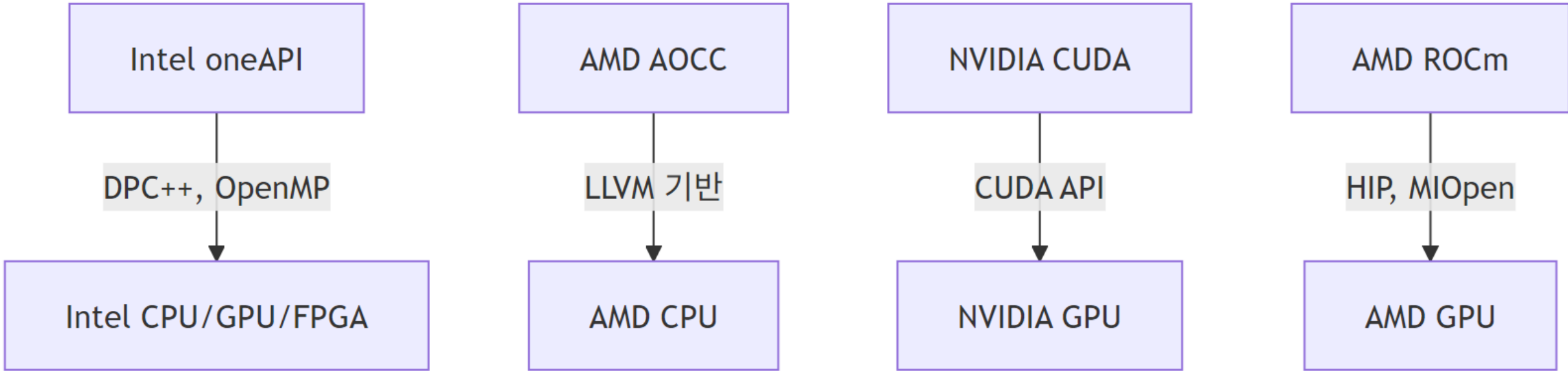
고 충실도의 고성능 시각화 응용프로그램

- Intel® Embree
- Intel® Open Volume Kernel Library
- Intel® Open Image Denoise
- Intel® OpenSWR
- Intel® OSPRay
- Intel® OSPRay Studio
- Intel® OSPRay for Hydra*

2. 클러스터 구성

항목 개발사	Intel oneAPI Intel	AMD AOCC AMD	NVIDIA CUDA NVIDIA	AMD ROCm AMD
주요 구성 목적	이기종 컴퓨팅 플랫폼 (CPU/GPU/FPGA)	CPU 최적화 컴파일러 (LLVM 기반)	NVIDIA GPU 병렬 프로그래밍	AMD GPU 병렬 컴퓨팅 플랫폼
타겟 아키텍처	Intel CPU, GPU, FPGA (Xe)	AMD Zen 계열 CPU	NVIDIA GPU (Ampere, Hopper 등)	AMD GPU (CDNA, MI200/300 계열 등)
주요 언어 및 API	DPC++ (SYCL), C/C++, Fortran, OpenMP	C/C++, Fortran, OpenMP	CUDA C/C++, CUDA Fortran, Python	HIP (CUDA 호환), OpenCL, OpenMP
주요 구성 요소	DPC++, oneMKL, oneDNN, VTune, Advisor	AOCC 컴파일러, AOCL 라이브러리	cuBLAS, cuDNN, TensorRT, Nsight	HIP, MIOpen, rocBLAS, rocFFT, rocProfiler
지원 OS	Linux, Windows	Linux (RHEL/CentOS/Ubuntu)	Linux, Windows, WSL2	Linux 전용 (Ubuntu, RHEL 등)
CUDA 코드 이식성	dpct로 CUDA → DPC++ 변환	없음	자체 CUDA 코드	hipify로 CUDA → HIP 자동 변환
라이브러리 생태계	oneAPI Toolkit (oneMKL, oneDNN 등)	AOCL (BLIS, FFT, Libm 등)	CUDA-X, cuDNN, cuBLAS, NCCL	ROCm libs (MIOpen, rocBLAS, RCCL 등)
벤치마크 최적화 대상	HPC, AI, 금융모델링, CFD	HPC, LLVM 기반 최적화	AI, HPC, 그래픽, 물리 엔진	AI inference, HPC (PyTorch, TensorFlow)
주요 활용 분야	슈퍼컴퓨팅, AI 추론, DPC++ 병렬화	AMD CPU 기반 HPC 워크로드	AI 학습/추론, GPU 병렬처리	GPU 기반 AI 추론 및 HPC 워크로드
분석 및 디버깅 도구	Intel VTune, Advisor, Inspector	GDB, LLVM Profiler	Nsight Compute, Nsight Systems	rocprof, rocTracer
대표 워크로드	GROMACS, OpenFOAM, TensorFlow (SYCL)	SPEC CPU, Fluent, OpenFOAM (CPU용)	PyTorch, TensorFlow, cuDNN 기반 모델	PyTorch, TensorFlow (with HIP)
오픈소스 여부	부분 오픈소스 (DPC++ 등)	LLVM 기반 오픈소스 AOCC	대부분 폐쇄형 (일부 라이브러리는 공개됨)	대부분 오픈소스
2025 최신 버전 기준	oneAPI 2025.1	AOCC 4.x	CUDA Toolkit 12.x	ROCm 6.x

2. 클러스터 구성



용어	분류	목적 및 역할	대상 아키텍처	사용 주체 (툴킷)	예시 용도
DPC++ (Data Parallel C++)	언어 (SYCL 확장)	CPU/GPU 병렬화	Intel CPU/GPU, 일부 NVIDIA/AMD	Intel oneAPI	AI inference, 과학 계산, 이기종 클러스터
OpenMP	병렬 API	멀티코어 병렬처리	CPU, 일부 GPU(OpenMP 5.0 이상)	GCC, Intel, AOCC 등	멀티스레딩 루프 병렬화 (e.g., #pragma)
LLVM 기반 Low Level Virtual Machine	컴파일러 프레임워크	컴파일러 백엔드	거의 모든 ISA	AOCC, Clang, oneAPI 등	AOCC, Clang, Flang 컴파일러 기반
CUDA API	프로그래밍 API	NVIDIA GPU 병렬처리	NVIDIA GPU 전용	NVIDIA CUDA Toolkit	PyTorch, TensorFlow GPU backend
HIP Heterogeneous-Compute Interface for Portability	프로그래밍 API	CUDA ↔ AMD 변환	AMD GPU 주 대상	AMD ROCm HIP	CUDA 코드 호환 실행 (e.g., hipLaunchKernel)
MIOpen	딥러닝 라이브러리	AI 연산 최적화	AMD GPU	AMD ROCm + PyTorch 등	cuDNN 대체 (Convolution, BN, Pooling 등)

2. 클러스터 구성

```
// 목적:  $Y[i] = a * X[i] + Y[i]$   
// 입력: float *X, *Y; float a;  
// 출력: 계산 후 Y[i]
```

2. 클러스터 구성

```
#include <stdio.h>
#include <stdlib.h>

void daxpy(int n, float a, float *x, float *y) {
    for (int i = 0; i < n; ++i) {
        y[i] = a * x[i] + y[i];
    }
}

int main() {
    int n = 1000000;
    float a = 2.0f;

    float *x = (float *)malloc(n * sizeof(float));
    float *y = (float *)malloc(n * sizeof(float));

    // 초기화
    for (int i = 0; i < n; ++i) {
        x[i] = 1.0f;
        y[i] = 2.0f;
    }

    daxpy(n, a, x, y);

    // 결과 확인 (일부만 출력)
    for (int i = 0; i < 5; ++i) {
        printf("y[%d] = %f\n", i, y[i]);
    }

    free(x);
    free(y);
    return 0;
}
```

2. 클러스터 구성

DPC++ (SYCL) 변경 포인트

```
buffer<float> x_buf(x, range<1>(N));
```

```
buffer<float> y_buf(y, range<1>(N));
```

```
q.submit([&](handler &h) {
```

```
    auto x_acc = x_buf.get_access<access::mode::read>(h);
```

```
    auto y_acc = y_buf.get_access<access::mode::read_write>(h);
```

```
    h.parallel_for(range<1>(N), [=](id<1> i) {
```

```
        y_acc[i] = a * x_acc[i] + y_acc[i];
```

```
    });
```

```
});
```

2. 클러스터 구성

AOCC (OpenMP) 변경 포인트

```
#include <omp.h> // 반드시 포함
```

```
void daxpy(int n, float a, float *x, float *y) {  
    #pragma omp parallel for // 병렬화 지시문  
    for (int i = 0; i < n; ++i) {  
        y[i] = a * x[i] + y[i];  
    }  
}
```

```
void daxpy(int n, float a, float *x, float *y) {  
    for (int i = 0; i < n; ++i) {  
        y[i] = a * x[i] + y[i];  
    }  
}
```

2. 클러스터 구성

CUDA 변경 포인트

```
__global__ void daxpy(float *x, float *y, float a, int N) {  
    int i = blockIdx.x * blockDim.x + threadIdx.x;  
    if (i < N)  
        y[i] = a * x[i] + y[i];  
}
```

// 호출부

```
daxpy<<<numBlocks, blockSize>>>(d_x, d_y, a, N);
```

2. 클러스터 구성

HIP (ROCm) 변경 포인트

```
__global__ void daxpy(float *x, float *y, float a, int N) {  
    int i = blockIdx.x * blockDim.x + threadIdx.x;  
    if (i < N)  
        y[i] = a * x[i] + y[i];  
}
```

// 호출부

```
hipLaunchKernelGGL(daxpy, dim3(numBlocks), dim3(blockSize), 0, 0, d_x,  
d_y, a, N);
```

2. 클러스터 구성

변경 포인트 요약

변환 대상	기본 C 코드	DPC++	OpenMP	CUDA / HIP
for 루프	for (i = 0; i < N...)	parallel_for(range<1>(N), ...)	#pragma omp parallel for	__global__ kernel + index 계산
데이터 구조	float *x, *y	buffer<float>	동일	float *d_x, *d_y (GPU 메모리)
실행 위치	CPU 단일 스레드	Intel GPU / CPU 등	CPU 멀티코어	NVIDIA/AMD GPU

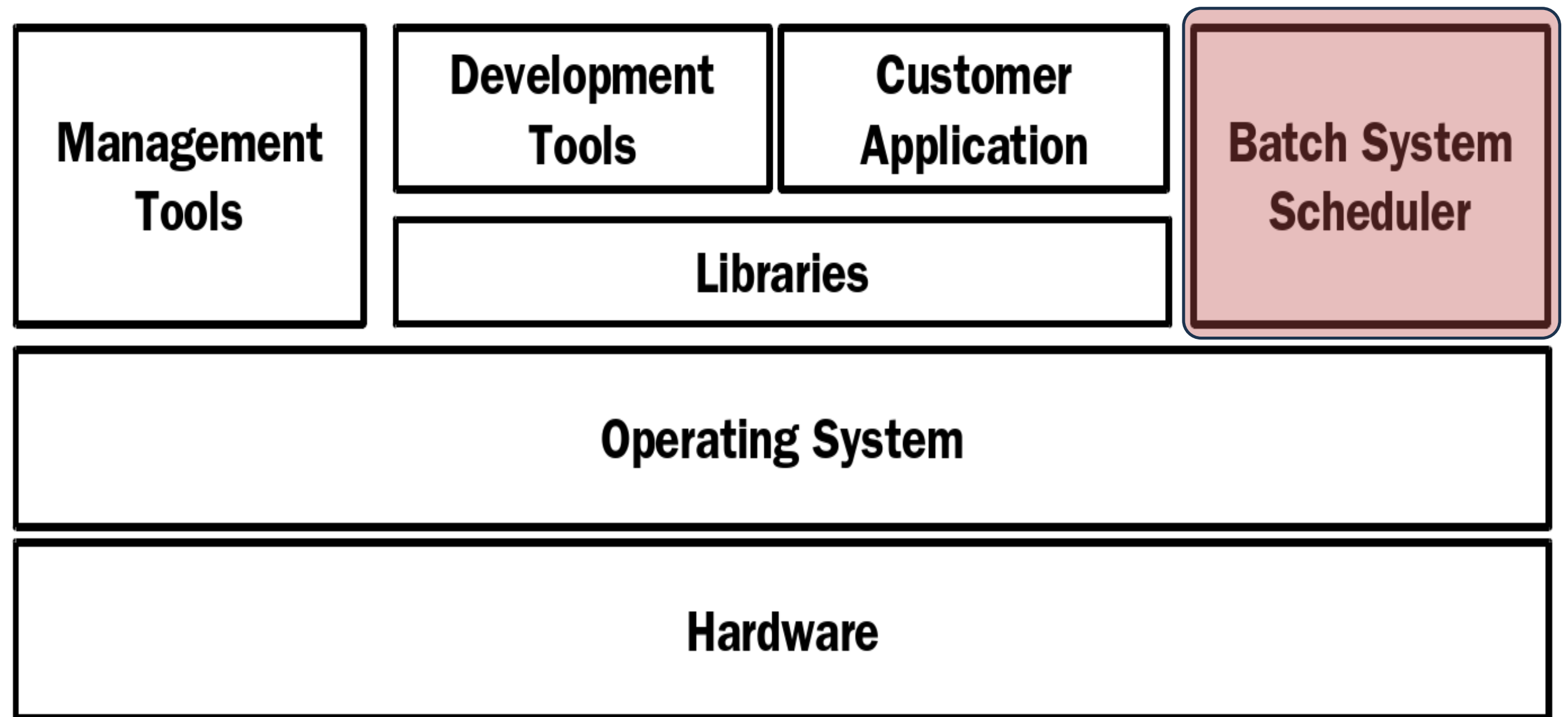
2. 클러스터 구성

성능 비교 요약 (예시)

Toolkit	N = 10M	N = 100M	N = 1B
CUDA (NVIDIA)	0.06s	0.48s	4.2s
ROCm (AMD)	0.08s	0.52s	4.5s
AOCC (OpenMP)	0.18s	1.6s	15.0s
DPC++ (oneAPI)	0.25s	2.3s	22.1s

2. 클러스터 구성

Batch System – Scheduler



2. 클러스터 구성

Batch System - Scheduler

- Batch System - Scheduler**는 HPC(고성능 컴퓨팅), 클러스터, 데이터 센터에서 컴퓨팅 자원을 효율적으로 분배하고 관리하기 위한 핵심 구성요소입니다.

구성 요소	설명
Batch System	사용자의 작업(Job)을 대기열에 넣고, 실행 조건이 충족되면 순서대로 실행하는 시스템
Scheduler	각 Job의 우선순위, 자원 사용량, 정책 등을 고려해 어떤 Job을 언제 어디서 실행할지 결정하는 소프트웨어 컴포넌트

2. 클러스터 구성

Batch System - Scheduler

항목	스케줄러 없음	스케줄러 있음
작업 제출 방식	사용자가 직접 ssh 접속 후 실행	sbatch, qsub, bsub 등 명령어로 제출
자원 충돌	여러 작업이 CPU/GPU를 동시에 점유 가능	자원 충돌 없이 순차적으로 실행됨
자원 할당 최적화	불균형 사용 (어떤 노드는 과부하, 어떤 노드는 idle)	전체 노드 자원 균형 있게 분배
대기열 관리	없음 - 먼저 온 사람 마음대로	대기열 기반 - 공정한 우선순위 적용
실패/재시작 관리	수동 재실행 필요	자동 재시작 및 체크포인트 지원 가능
병렬 작업 관리	수동 스크립트 조정 필요	MPI, OpenMP, Array Job 등 병렬 작업 자동 분산
모니터링/로그 관리	개별 사용자 관리	중앙 집중식 모니터링, 로깅 지원

2. 클러스터 구성

Batch System - Scheduler



- 2020년 6월에
- SGE(Son of Grid Engine) 및 Torque 작업 스케줄러 사용을 중단한다고 발표
- ParallelCluster 버전 2.0.0 - 2.7.0에는 클러스터 구성 옵션(Slurm 및 AWS Batch 외에)을 통해 사용할 수 있는 작업 스케줄러로 SGE 및 Torque가 포함되었습니다. SGE 및 Torque는 커뮤니티에서 만들고 유지 관리하는 오픈 소스 스케줄러이지만 일정 기간 동안(2016-SGE, 2018-Torque) 활발하게 개발되지 않았습니다.

2. 클러스터 구성

Batch System – Scheduler 용어 정리

- 자원 (Resource)

자원은 일반적으로 CPU, 메모리 디스크 네트워크 등 시스템에서 사용자가 이용하는 각각의 구성 요소들을 포함하는 개념

하나의 메인보드(Main Board) (Node) 에 존재하는 요소들을 노드(Node) 라는 단위로 명명

- 작업 (Job) & 테스크 (Task)

작업은 사용자가 자원 할당을 요청하는 것으로 하나 또는 여러 노드를 할당 받을 수 있음 작업에는 자원에 대한 요청 정보와 해당 자원에서 실행할 어플리케이션 정보를 포함

- 스케줄러(Scheduler) & 메타 스케줄러(Meta Scheduler)

스케줄링을 담당하는 프로그램을 스케줄러라 하는데 RJMS의 하위 시스템 중 하나

Moab, Maui, GRMS, GridWay 등 몇몇 스케줄러는 클러스터가 아닌 그리드 시스템에 대 하여 스케줄링 기능을 하는데 이러한 스케줄러를 메타 스케줄러 (Meta Scheduler) 라 한다

2. 클러스터 구성

Batch System – Scheduler 용어 정리

- RJMS(Resource and Job Management System)

스케줄러는 컴퓨팅 자원을 사용자의 작업에 연결해주는 역할을 수행하고 자원과 작업을 매칭(matching)하는 스케줄링 기능으로 구분

JMS(Job Management System), RMS(Resource Management System), LRMS(Local Resource Management System), 스케줄러(Scheduler), Queuing System, 배치 시스템 (Batch System), 배치 스케줄러(Batch Scheduler), Workload Management System(Platform)

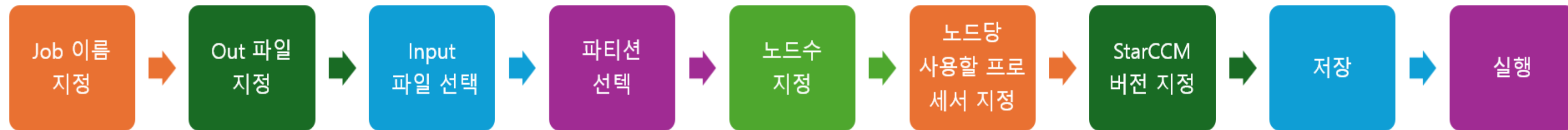
- 그리드(grid) & 클러스터 (Cluster)

그리드 컴퓨팅은 (Grid Computing) 그리드 컴퓨팅 은 분산 병렬 컴퓨팅의 한 분야로서 원거리 통신망 , (WAN: Wide Area Network) 으로 연결된 서로 다른 종류의 (heterogeneous) 컴퓨터들을 묶어 가상의 대용량 고성능 컴퓨터를 구성하여 고도의 연산 작업 혹은 대용량 처리 를 수행하는 것을 일컫는다

클러스터 (cluster)는 여러 대의 컴퓨터들이 연결되어 하나의 시스템처럼 동작하는 컴퓨터들의 집합을 말한다 클러스터의

2. 클러스터 구성

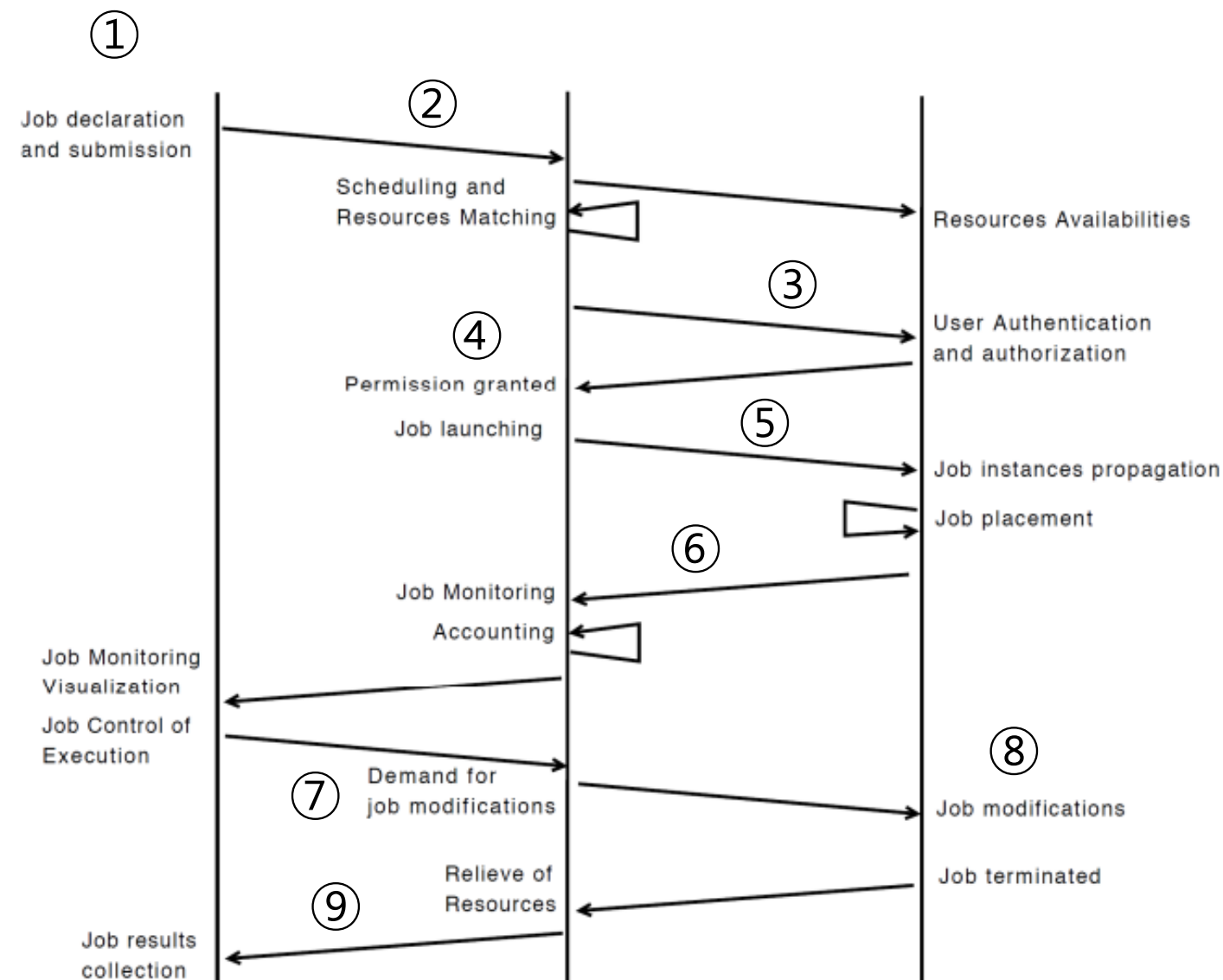
작업 제출 실행 모니터링 과정



- ①작업을 할 컴퓨팅 그룹 결정
- ②Input 파일을 '기존 컴퓨팅 노드의 스토리지' 또는 '신규 컴퓨팅 노드의 스토리' 중에 선택
- ③작업을 실행할 컴퓨팅 그룹 '파티션'을 선택
- ④StarCCM 버전 선택
- ⑤실행

2. 클러스터 구성

Batch System – Scheduler 흐름



1. Job Declaration and Submission

사용자가 작업(Job)을 제출
→ RJMS가 작업을 받아들임

2. Scheduling and Resources Matching

RJMS가 현재 클러스터 자원 상태를 조회
→ Resource Availability 정보를 통해 어느 노드에 작업을 배치할 수 있는지 판단

3. User Authentication and Authorization

제출된 작업의 사용자 인증 및 권한 확인

4. Permission Granted

인증이 성공하면 작업 실행 허가

5. Job Launching

작업을 실제로 실행하기 위한 초기화 과정
→ Job instances propagation: 여러 노드에 작업 복제 전파
→ Job Placement: 각 노드에 작업을 실제로 배치

6. Job Monitoring / Accounting

작업이 실행되는 동안 모니터링 및 사용 자원 기록(회계) 수행
사용자도 작업 상태를 시각적으로 확인 가능

7. Demand for Job Modifications

사용자가 작업 중간에 수정 요청 가능 (우선순위 변경, 취소, 리소스 재조정 등)

8. Job Modifications / Job Terminated

자원 관리자가 요청을 반영하여 Job을 수정하거나 종료
→ 자원 반환 처리 (Relieve of Resources)

9. Job Results Collection

작업 결과를 사용자에게 반환

2. 클러스터 구성

작업 제출 실행 모니터링 과정

1. 사용자가 작업을 기술한 스크립트를 작성하고 서버에 제출을 한다.
2. 서버는 자원의 사용 가능 여부에 대한 정보를 확인한다.
3. 새로운 작업 요청에 대한 스케줄링을 할 때 작업이 필요로 하는 자원을 매칭하여 자원을 작업에 할당한다.
4. 계산 노드에 대하여 사용자 인증 및 권한을 확인한다.
5. 계산 노드에 대한 사용자 접근이 승인되면 서버에 통보한다.
6. 자원이 확보된 이후에 서버에서 사용자의 작업을 론칭(launching) 한다.
7. 계산 노드로 작업이 전파되고 생성된 태스크에 대한 배치가 이루어진다.
8. 작업이 실행된 이후 서버에서 작업을 모니터링한다.
9. 사용자의 작업에 대한 어카운팅(accounting) 이 함께 이루어진다.
10. 사용자는 서버를 통해 실행되고 있는 작업에 대하여 모니터링 할 수 있다.
11. 작업이 실행되는 중에 사용자가 작업에 대한 변경을 원할 경우 서버에 작업에 대한 변경을 요청한다
12. 서버가 사용자의 요청을 계산 노드에 전달하면 실행중인 작업에 대한 변경된 설정이 반영된다.
13. 이후 작업이 종료되면 서버가 해당 이벤트를 받게 되고 해당 작업에 할당되었던 자원을 해제한다 .

2. 클러스터 구성

Batch System – Scheduler 용어 정리

사용자의 작업과 사용 가능한 자원 간의 매칭을 담당하는 스케줄링 기능이 RJMS의 가장 중요한 역할 중 하나

구성 요소	설명
Resource Manager	노드, CPU, GPU, 메모리 등 자원의 상태를 추적하고 할당하는 역할 (예: slurmd, pbs_mom)
Job Manager	사용자로부터 제출된 작업을 수신하고, 스케줄러에 전달하며 상태를 추적
Scheduler	제출된 작업의 우선순위, 자원 요청, 정책 등을 기반으로 실행 순서 및 노드 할당 결정
Accounting/Policy Engine	사용자, 그룹, 프로젝트별 자원 사용량을 추적하고 제한 및 우선순위 적용

2. 클러스터 구성

Batch System – Scheduler 용어 정리

	기본 기능	고급 기능
자원 관리	<ul style="list-style-type: none"> • 자원 구성 구조(Hierarchy), 파티션(Partitions) • 작업 론칭(Job Launching) • 작업전파(Job Propagation) • 작업 실행 관리(Job Execution Control) • 작업 배치(Task Placement) 토폴로지 (Topology), 바인딩(Binding) 	<ul style="list-style-type: none"> • 고가용성 (High Availability) • 에너지효율(Energy Efficiency) • 토폴로지 인지 배치 (Topology aware placement)
작업 관리	<ul style="list-style-type: none"> • 작업 선언 (Job Declaration) 종류(type), 특성(Characteristics) • 작업 관리 (Job Control) • 모니터링 (Monitoring) 보고 (Reporting), 가시화 (Visualization) 	<ul style="list-style-type: none"> • 인증 (Authentication) 제한(limitations), 보안(security) • QoS Checkpoint, accounting • Interfacing MPI libs, APIs, debuggers
스케줄링	<ul style="list-style-type: none"> • 스케줄링 알고리즘(Scheduling Algorithm) • 큐 관리 (Queues Management) 	<ul style="list-style-type: none"> • Advanced Reservation • Application Licenses

2. 클러스터 구성

Batch System – Scheduler – 이기종 자원 관리의 변화와 복잡성 증가

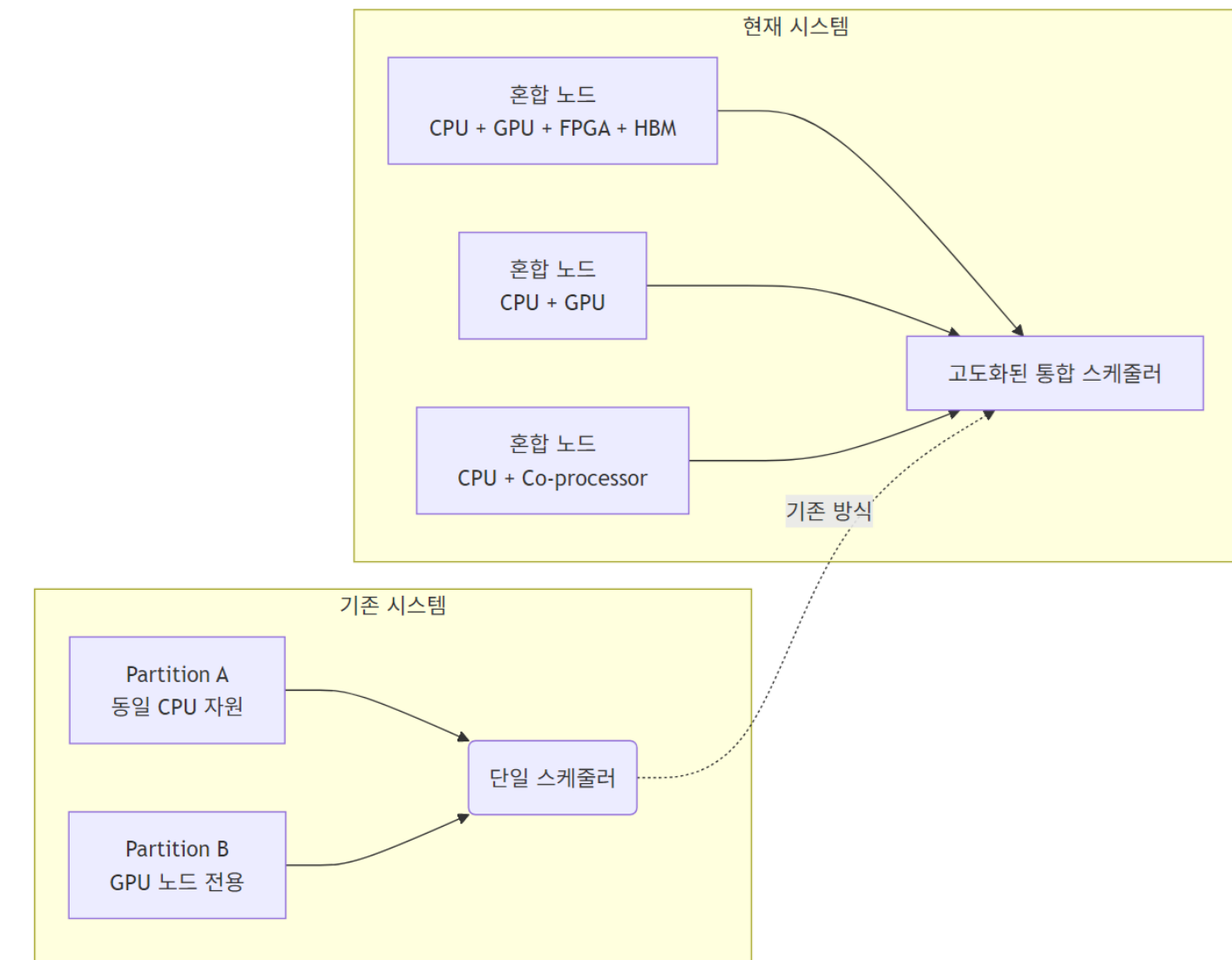
- 기존 시스템에는 이기종 형태의 자원에 대해서는 파티션(Partition) 으로 구분 지어 각각의 파티션에 대해 동일한 자원의 형태를 포함하도록 구성하거나 여러 클러스터로 구분한 후 클러스터를 단일 스케줄러가 관리하는 방식으로 운영
- 오늘날의 하드웨어 구조는 자원 자체를 파티션으로 구분하는 것이 불가능하고 가속기(Accelerator or Co-processor) 형태의 이기종 자원들을 통합 관리해야 하는 어려움으로 인해 자원 을 관리하는 구성을 보다 복잡하게 한다.

◆ 과거의 구조

- 자원을 **유형별로 파티션(Partition)** 구분
- 각 파티션은 동일한 CPU/메모리 구성
- **여러 클러스터 → 하나의 스케줄러로 통합 관리**

◆ 현재의 변화

- 가속기(Accelerator, GPU 등)가 필수 요소로 통합
- 파티션으로 단순 구분이 **불가능**
- CPU/GPU/FPGA/메모리 혼합 → 자원 관리 복잡성 증가
- **동적 자원 할당과 스케줄링 전략의 고도화 필요**



2. 클러스터 구성

Batch System – Scheduler

정책	설명
FIFO	작업들을 도착한 순서대로
Backfill	가용한 충분한 자원을 할당 받지 못해 기다리는 작업에 의해 생성된 홀 (hole)을 늦게 제출된 작업이 할당 받아 실행하는 방식
Time Sharing	여러 작업들이 동일한 자원을 할당 받은 경우 OS의 context switching 같은 방식에 의해 일정 간격의 시간을 분배 받아 작업을 실행하는 방식
Fair share	여러 작업들이 동일한 자원을 이용할 경우 많이 실행된 작업의 우선순위를 조절하여 자원을 많이 받지 못한 작업이 더 많이 실행되어 공정하게 운영하는 방식
Preemption	우선순위가 낮은 작업들을 멈추고 우선순위가 높은 작업들이 실행되도록 하는 방식
Gang Scheduling	여러 작업들이 동일한 자원을 할당 받을 때 특정한 시간 간격으로 서로 돌아 가면서 자원을 이용하는 방식

2. 클러스터 구성

Batch System – Scheduler

항목	Torque	OAR	Platform LSF	Condor	Slurm	UGE
프로그래밍 언어	C	Perl, Ruby, Ocaml	C	C++	C	C
인증	SSH, Munge	SSH	SSH	GSI, SSL, Kerberos, Password	Munge, Kerberos	PAM 기반 인증
암호화	SSH 기반	SSH 기반	SSH 기반	Triple DES, BLOWFISH	Munge/Kerberos 기반 암호화	PAM 기반 암호화
무결성	기본 제공	기본 제공	기본 제공	MD5 제공	기본 제공	Integrity 제공
전역 파일 시스템	Any, NFS	None, NFS	NFS	None, NFS, AFS	NFS, Lustre, HDFS, AFS	대부분 포함 (NFS, Lustre 등)
계산노드 내 이기종 구성	yes	yes	yes	yes	yes	yes
작업 우선순위	yes	yes	yes	yes	yes	yes
그룹 우선순위	yes	yes	yes	yes	yes	yes
큐 형태	programmable	programmable	programmable	fairshare, programmable	multifactor, fairshare	배치, 양방향, 병렬, 조합
SMP 인지	yes	yes	yes	basic	yes	yes
최대 실행 노드	tested	tested 80K	tested 10,000	tested 120K	tens of thousands	tens of thousands
최대 작업 제출	tested	tested > 20K	tested 100K	tested 100K	tested 100K	> 300K
CPU scavenging	yes	yes	yes	yes	no	yes
병렬 작업	yes	yes	yes	MPI, OpenMP, PVM	yes	yes
체크포인팅	yes (BLCR)	yes (BLCR)	yes	yes	yes (BLCR)	yes (사용자/커널)

2. 클러스터 구성

Batch System – Scheduler

항목	Torque	OAR	Platform LSF	Condor	Slurm	UGE
프로그래밍 언어	C	Perl, Ruby, Ocaml	C	C++	C	C
인증	SSH, Munge	SSH	SSH	GSI, SSL, Kerberos, Password	Munge, Kerberos	PAM 기반 인증
암호화	SSH 기반	SSH 기반	SSH 기반	Triple DES, BLOWFISH	Munge/Kerberos 기반 암호화	PAM 기반 암호화
무결성	기본 제공	기본 제공	기본 제공	MD5 제공	기본 제공	Integrity 제공
전역 파일 시스템	Any, NFS	None, NFS	NFS	None, NFS, AFS	NFS, Lustre, HDFS, AFS	대부분 포함 (NFS, Lustre 등)
계산노드 내 이기종 구성	yes	yes	yes	yes	yes	yes
작업 우선순위	yes	yes	yes	yes	yes	yes
그룹 우선순위	yes	yes	yes	yes	yes	yes
큐 형태	programmable	programmable	programmable	fairshare, programmable	multifactor, fairshare	배치, 양방향, 병렬, 조합
SMP 인지	yes	yes	yes	basic	yes	yes
최대 실행 노드	tested	tested 80K	tested 10,000	tested 120K	tens of thousands	tens of thousands
최대 작업 제출	tested	tested > 20K	tested 100K	tested 100K	tested 100K	> 300K
CPU scavenging	yes	yes	yes	yes	no	yes
병렬 작업	yes	yes	yes	MPI, OpenMP, PVM	yes	yes
체크포인팅	yes (BLCR)	yes (BLCR)	yes	yes	yes (BLCR)	yes (사용자/커널)

2. 클러스터 구성

Batch System – Scheduler 비교

Common User Commands					
	PBS/Torque	Slurm	LSF	SGE	LoadLeveler
Job submission	qsub [script file]	sbatch [script file]	bsub [script file]	qsub [script file]	lsubmit [script file]
Job deletion	qsub [job id]	scancel [job id]	bkill [job id]	qdel [job id]	llcancel [job id]
Job status	qstat [job id]	squeue [job id]	bjobs [job id]	qstat -u * [-j job id]	llq -u [username]
Job status (by user)	qstat -u [user_name]	squeue -u [user_name]	bjobs -u [user_name]	qstat [-u user_name]	llq -u [username]
Job hold	qhold [job id]	scontrol hold [job id]	bstop [job id]	qhold [job_id]	llhold -r [job id]
job release	qrls [job id]	scontrol release [job id]	bresume [job id]	qrls [job id]	llhold -r [job id]
Queue list	qstat -Q	squeue	bqueues	qconf -sql	llclass
Node list	pbsnodes -l	sinfo -N OR scontrol show nodes	bhosts	qhost	llstatus -L machine
Cluster status	qstat -a	sinfo	bqueues	qhost -q	llstatus -L cluster
GUI	xpbsmon	sview	xlsf OR xlsbatch	qmon	xload

2. 클러스터 구성

Batch System – Scheduler 비교

Environment Variables					
	PBS/Torque	Slurm	LSF	SGE	LoadLeveler
Job ID	\$PBS_JOBID	\$SLURM_JOBID	\$LSB_JOBID	\$JOB_ID	\$LOAD_STEP_ID
Submit Directory	\$PBS_O_WORKDIR	\$SLURM_SUBMIT_DIR	\$LSB_SUBCWD	\$SGE_O_WORKDIR	\$LOADL_STEP_INITDIR
Submit Host	\$PBS_O_HOST	\$SLURM_SUBMIT_HOST	\$LSB_SUB_HOST	\$SGE_O_HOST	
Node list	\$PBS_NODEFILE	\$SLURM_JOB_NODELIST	\$LSB_HOSTS/LSB_MCPU_HOST	\$PE_HOSTFILE	\$LOADL_PROCESSOR_LIST
job Array Index	\$PBS_ARRAYID	\$SLURM_ARRAY_TASK_ID	\$LSB_JOBINDEX	\$SGE_TASK_ID	

2. 클러스터 구성

Batch System – Scheduler 비교

Job Specification					
	PBS/Torque	Slurm	LSF	SGE	LoadLeveler
Script directive	#PBS	\$SBATCH	\$BSUB	#\$	#@
Queue	-q [queue]	-q [queue]	-q [queue]	-q [queue]	class=[queue]
Node Count	-l nodes=[count]	-N [min[-max]]	-n [count]	N/A	node=[count]
CPU Count	-l ppn=[count] OR -l mppwidth=[PE_count]	-n [count]	-n [count]	-pe [PE][count]	
Wall Clock Limit	-l walltime=[hh:mm:ss]	-t [min] OR -t [days-hh:mm:ss]	-W [hh:mm:ss]	-l h_rt=[seconds]	wall_clock_limit=[hh:mm:ss]
Standard Output File	-o [file name]	-o [file name]	-o [file name]	-o [file name]	output=[file name]
Standard Error File	-e [file name]	-e [file name]	-e [file name]	-e [file name]	error=[file name]
Combine stdout/err	-j oe (both to stdout) OR -j eo (both to stderr)	(use -o without -e)	(use -o without -e)	-j yes	
Copy Environment	-V	--export=[ALL [NONE] variables]		-V	environment=COPY_ALL

2. 클러스터 구성

Batch System – Scheduler 비교

Job Specification					
	PBS/Torque	Slurm	LSF	SGE	LoadLeveler
Event Notification	-m abe	--mail-type=[events]	-B or -N	-m abe	notification=start error complete never always
Email Address	-M [address]	--mail-user=[address]	-u [address]	-M [address]	notify_user=[address]
Job Name	-N [name]	--job-name=[name]	-J [name]	-N [name]	job_name=[name]
Job Restart	-r [y/n]	--requeue OR --no-requeue (NOTE : configurable default)	-r	-r [yes no]	restart=[yes no]
Job Type					
Working Directory	N/A	--workdir=[dir_name]	(submission directory)	-wd [directory]	initialdir=[directory]
Resource Sharing	-l naccesspolicy=singlejob	--exclusive OR --shared	-x	-l exclusive	node_usage=not_shared
Memory Size	-l mem=[MB]	--mem=[mem][M G T] OR --mem-per-cpu=[mem][M G T]	-M [MB]	-l mem_free=[memory][K M G]	requirements=(Memory >= [MB])

2. 클러스터 구성

Batch System – Scheduler 비교

Job Specification					
	PBS/Torque	Slurm	LSF	SGE	LoadLeveler
Account to charge	-W group_list=[account]	--account=[account]	-P [account]	-A [account]	
Tasks Per Node	-l mppnppn [Pes_per_node]	--tasks-per-node=[count]		(Fixed allocation_rule in PE)	tasks_per_node=[count]
CPUs Per Task		--cpus-per-node=[count]			
Job dependency	-d [job id]	--depend=[start:job_id]	-w [done exit finish]	-hold_jid [job_id job_name]	
Job project		--wckey=[name]	-P [name]	-P [name]	
Job host preference		--nodelist=[nodes] AND/OR --exclude=[nodes]	-m [nodes]	-q [queue]@[node] OR -q [queue]@@[hostgroup]	
Quality Of Service (QOS)	-l qos=[name]	--qos=[name]			
Job Arrays	-t [array_apec]	--array=[array_spec] (Slurm version 2.6+)	J "name[array_spec]"	-t [array_spec]	
Generic Resources	-l other=[resource_spec]	--gres=[resource_spec]		-l [resource]=[value]	
Licenses		--licenses=[license_spec]	-R "rusage[license_spec]"	-l [license]=[count]	
Begin Time	-A "YYYY-MM-DD HH:MM:SS"	--begin=YYYY-MM-DD[THH:MM[:SS]]	-b [[year:][month:]day:]hour:minute	-a [YYMMDDhhmm]	

2. 클러스터 구성

```
#!/bin/bash
#SBATCH --job-name=CFD_Workflow
#SBATCH --output=workflow_%j.out
#SBATCH --error=workflow_%j.err
#SBATCH --partition=prepost
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
#SBATCH --time=00:30:00
#SBATCH --mail-user=your_email@example.com
#SBATCH --mail-type=END,FAIL
```

```
echo "[STEP 1] Preprocessing (Mesh Generation)"
module load ansys/2024r1
fluent 3d -g -t8 -i pre.jou
```

```
# Step 2: Submit solve as dependent job
JOB_SOLVE=$(sbatch --dependency=afterok:$SLURM_JOB_ID solve.slurm | awk '{print $4}')
echo "[STEP 2] Solve job submitted with ID $JOB_SOLVE"
```

```
# Step 3: Submit post-processing job
sbatch --dependency=afterok:$JOB_SOLVE post.slurm
```

2. 클러스터 구성

```
#!/bin/bash
#SBATCH --job-name=CFD_Solve
#SBATCH --output=solve_%j.out
#SBATCH --partition=compute
#SBATCH --ntasks=64
#SBATCH --cpus-per-task=1
#SBATCH --gres=gpu:4
#SBATCH --time=04:00:00

echo "[STEP 2] Running Solver"
mpirun -np 64 fluent 3d -g -t64 -i solve.jou
```

2. 클러스터 구성

```
#!/bin/bash
#SBATCH --job-name=CFD_Post
#SBATCH --output=post_%j.out
#SBATCH --partition=prepost
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=16
#SBATCH --time=00:45:00
```

```
echo "[STEP 3] Post-processing"
paraview --script=auto_post.py
```

2. 클러스터 구성

감사합니다