연계 시뮬레이션의 종류 및 활용기술 비교

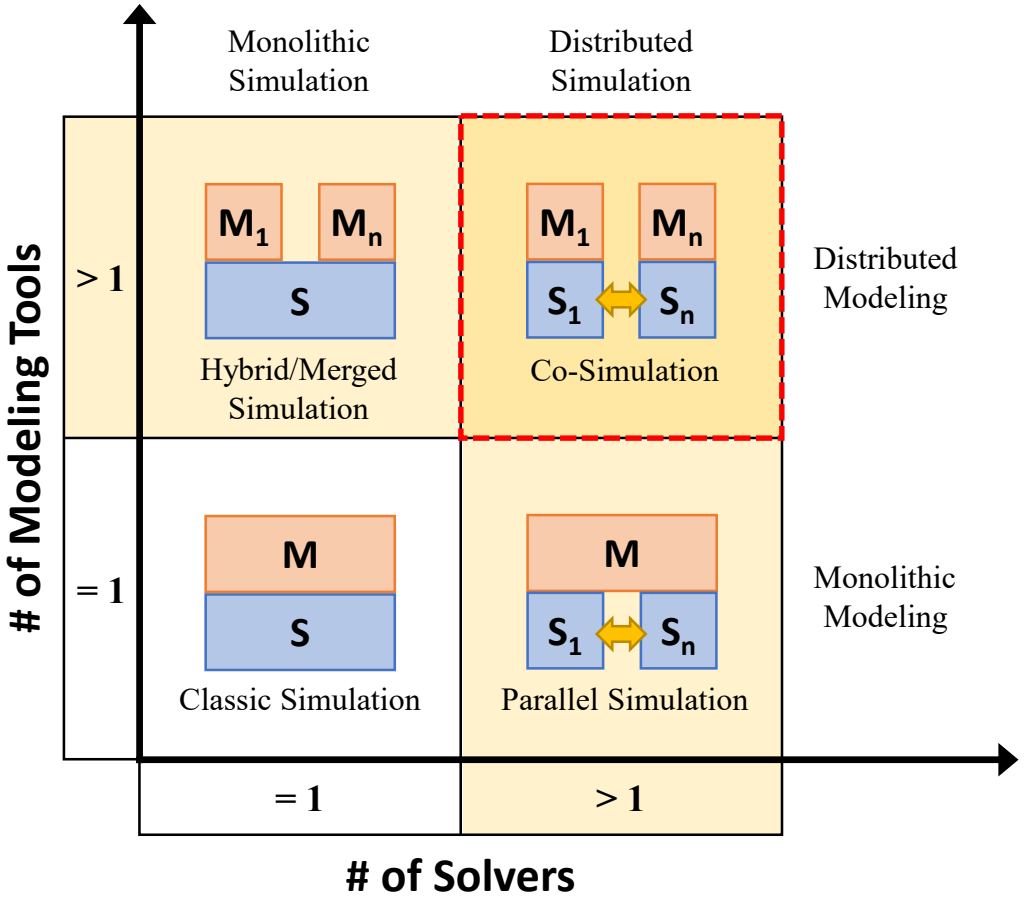
2022. 10. 19.

유병현

KAERI 한국원자력연구원
Korea Atomic Energy Research Institute

목차

- 시뮬레이션의 구분



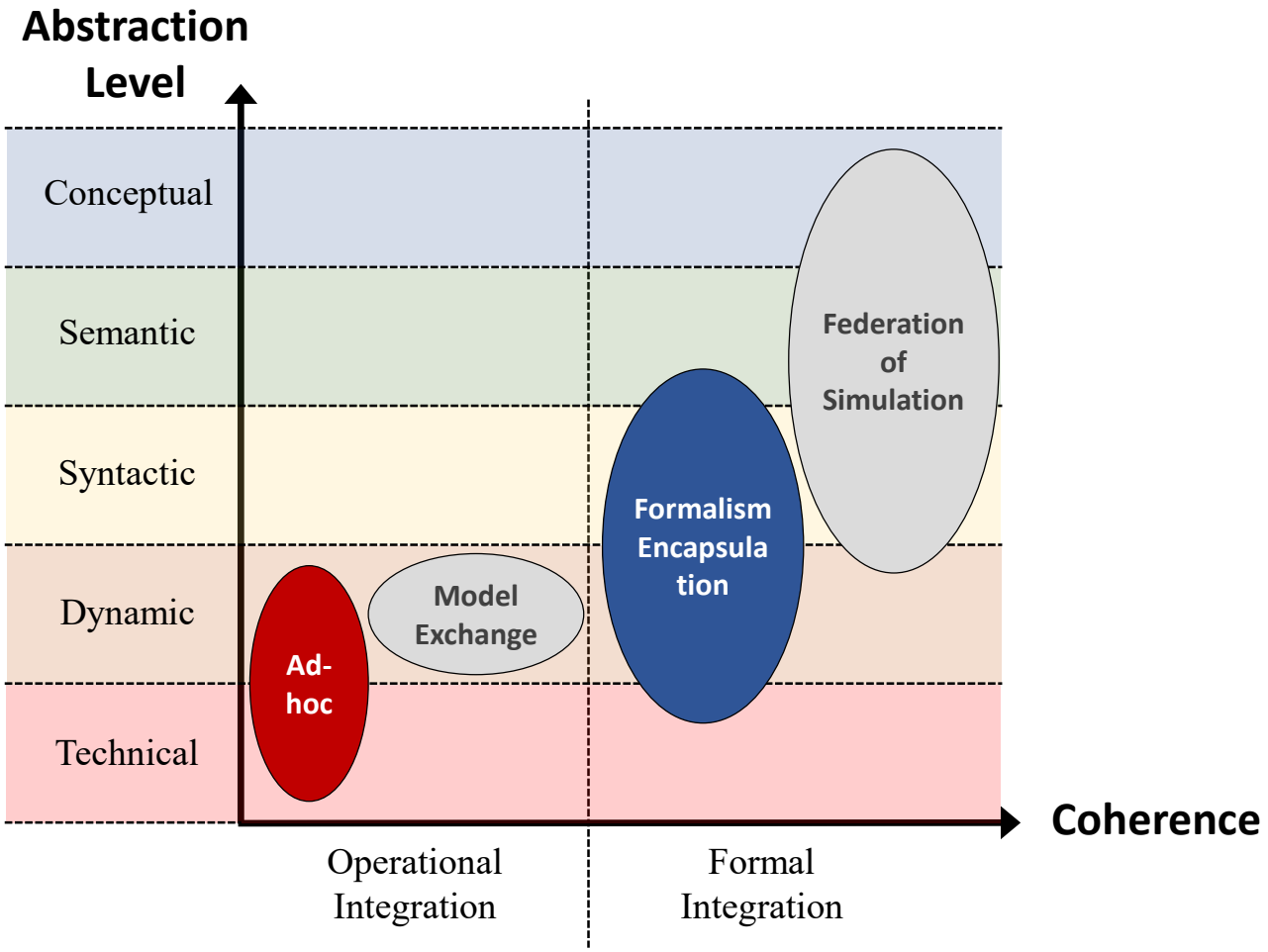
Steinbrink, C., et al., doi: 10.1007/978-3-319-64635-0_13 (2017)

▪ 연계 시뮬레이션의 5가지 추상화 단계

Abs. layer	Description	Issues
Conceptual	Highest level where the models are considered as black boxes and the level concerns the co-simulation framework representation.	Generic structure of the framework Meta-Modeling of the components
Semantic	The level concerns the signification and the role of the co-simulation framework with respect to the open questions of the investigated system and studied phenomenon.	Signification of individual models Interaction graph among the models Signification of each interaction
Syntactic	The level concerns the formalization of the co-simulation framework.	Formalization of individual models in the respectiv e domains Specification and handling the difference between a formalism to another one
Dynamic	The level concerns the execution of the co-simulation framework, the synchronization techniques and harmonization of different models of computation.	Order of execution and causality of models Harmo nization of different models of computation Resolution for potential conflict in simultaneity of actions.
Technical	The level concerns the implementation details and evaluation of simulation.	Distributed or centralized implementation Robustness of the simulation Reliability and efficiency of the simulation

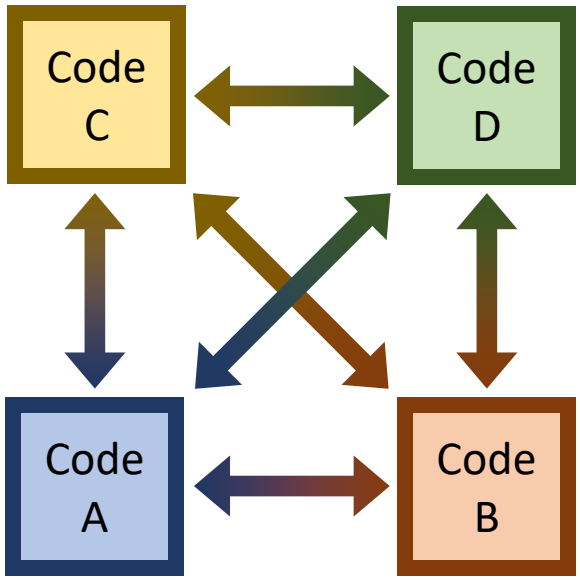
Nguyen, V.H. et al., doi: 10.3390/en10121977, 2017

- 연계 방식에 따른 구분



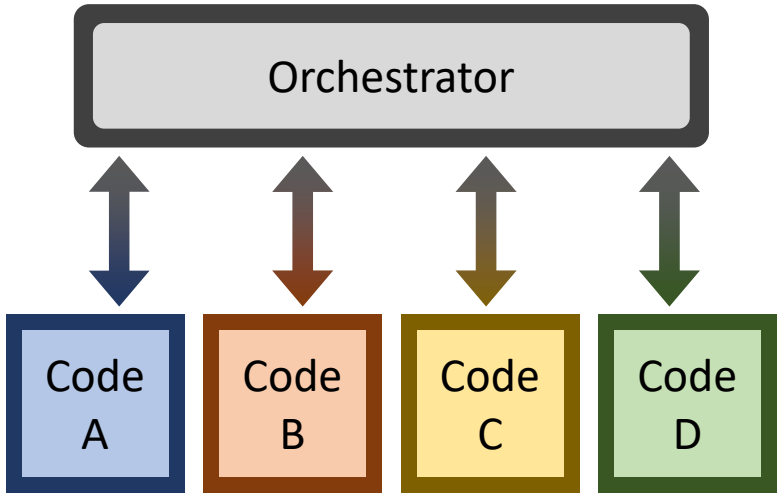
Nguyen, V.H. et al., doi: 10.3390/en10121977, 2017

▪ 연계 방식에 따른 구분



Ad-hoc or **Code-to-Code**
(Operational Integration)

- 작은 규모의 연계 방식 구현에 유리
- 규모 확장에 따른 복잡도 증가



Orchestrator
(Formal Integration)

- 일반화된 연계 방식 구현에 유리
- 작은 규모의 경우 불필요한 작업

- **Engineering/Research 분야의 연계 시뮬레이션**
  - 연계 소프트웨어 구조 설계
    - ✓ Semantic – Syntactic – Dynamic – Technical
  - Technical 구현 단계에서의 주요 요구사항
    - ✓ 안전성
      - 안정적인 계산 흐름의 진행
    - ✓ 신뢰성
      - 개별 코드의 계산 정확도, 수치적 재현성 등
    - ✓ 효율성
      - 데이터 교환 방식의 성능, 데이터 동기화 알고리즘

▪ Engineering/Research 분야의 연계 시뮬레이션

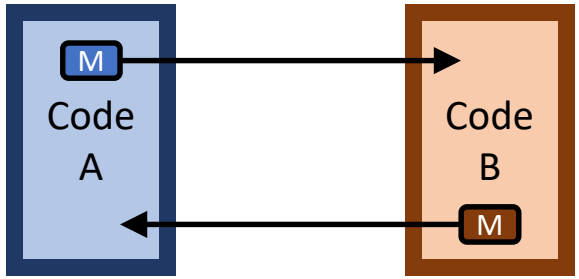
- 하드웨어 제약사항
  - ✓ 운영체제 (개별 해석코드의 제한사항)
    - 이종 운영체제 간 데이터 교환 필요 여부
  - ✓ 대규모 계산 vs. 소규모 계산
    - 클러스터 방식 or 단일 workstation 수준

- 소프트웨어 제약사항
  - ✓ 하드웨어 제약사항에 따른 기술 스택 선택 (통신 라이브러리 등)
    - ex) 클러스터: MPI, PC: Shared Memory

- 정책적 제약사항
  - ✓ 연계 시뮬레이션을 구성하는 개별 해석코드의 소스코드 접근 제한 여부

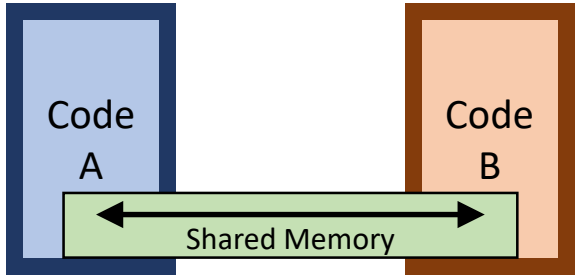
▪ 프로세스 간 통신 (Inter-Process Communication, IPC)

- Message
  - ✓ 데이터를 명시적으로 전송
  - ✓ 라이브러리 다수 존재
    - MPI, ZeroMQ, Kafka 등

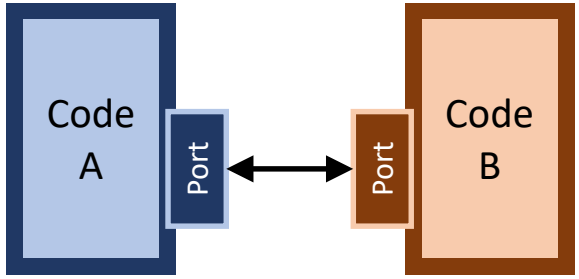


- Shared Memory
  - ✓ 두 프로세스의 공용 메모리 공간을 생성
  - ✓ 개발자가 상황에 맞게 직접 구현

  * Static/dynamic linking 을 통한 컴파일 단계에서의 Shared Memory 개념이 아님



- Socket
  - ✓ 네트워크 통신을 위해 Port를 열어 전송
  - ✓ 운영체제별 통신 구현체 사용



▪ IPC 방식 별 특징 비교

	원격 통신	동기화 기능	속도
Message	O	O	느림?
Socket	O	X	느림
Shared Memory	X	X	빠름

- 하드웨어 제약사항에 따른 IPC 기술 선택 시나리오
  - ✓ ex1) 단일PC 연계: Message, Socket, Shared Memory
  - ✓ ex2) 클러스터 연계: Message, Socket
  - ✓ ex3) 이종 운영체제간 통신이 필요한 연계: Socket

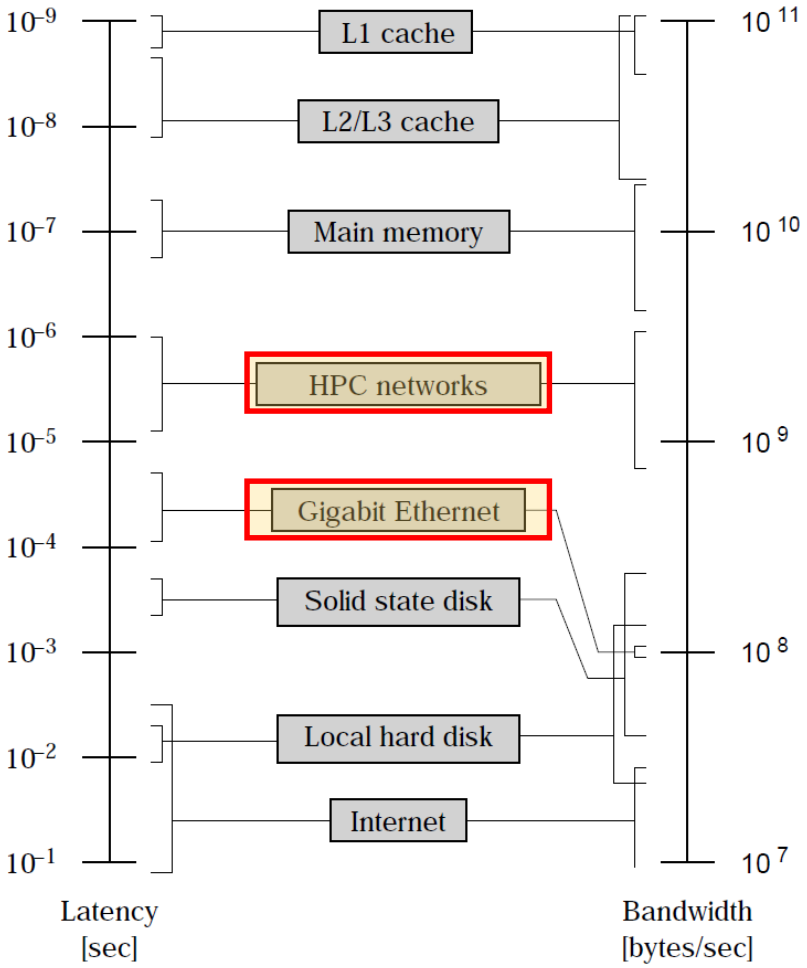
▪ 고성능 클러스터의 통신 기능 활용

- 네트워크 카드
  - ✓ (Gigabit) Ethernet

    일반적인 Socket, TCP/IP 등을 활용한 데이터 전송 구현 시

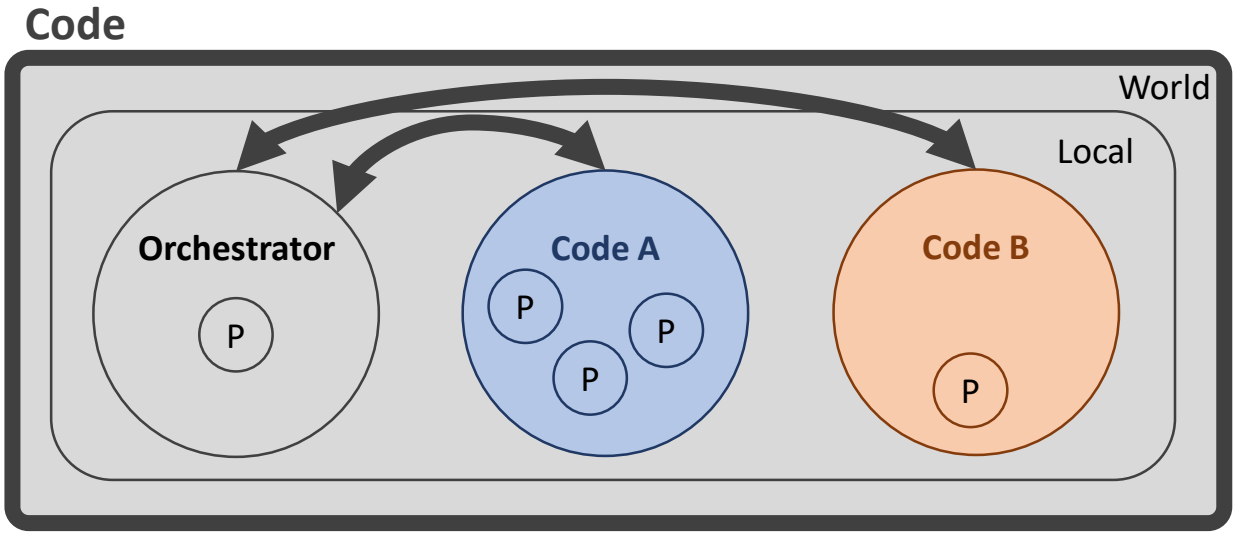
- HPC Networks
  - ✓ Infiniband, Omni-Path

    Vendor 가 제공한 통신 라이브러리 및 MPI 라이브러리 활용 시

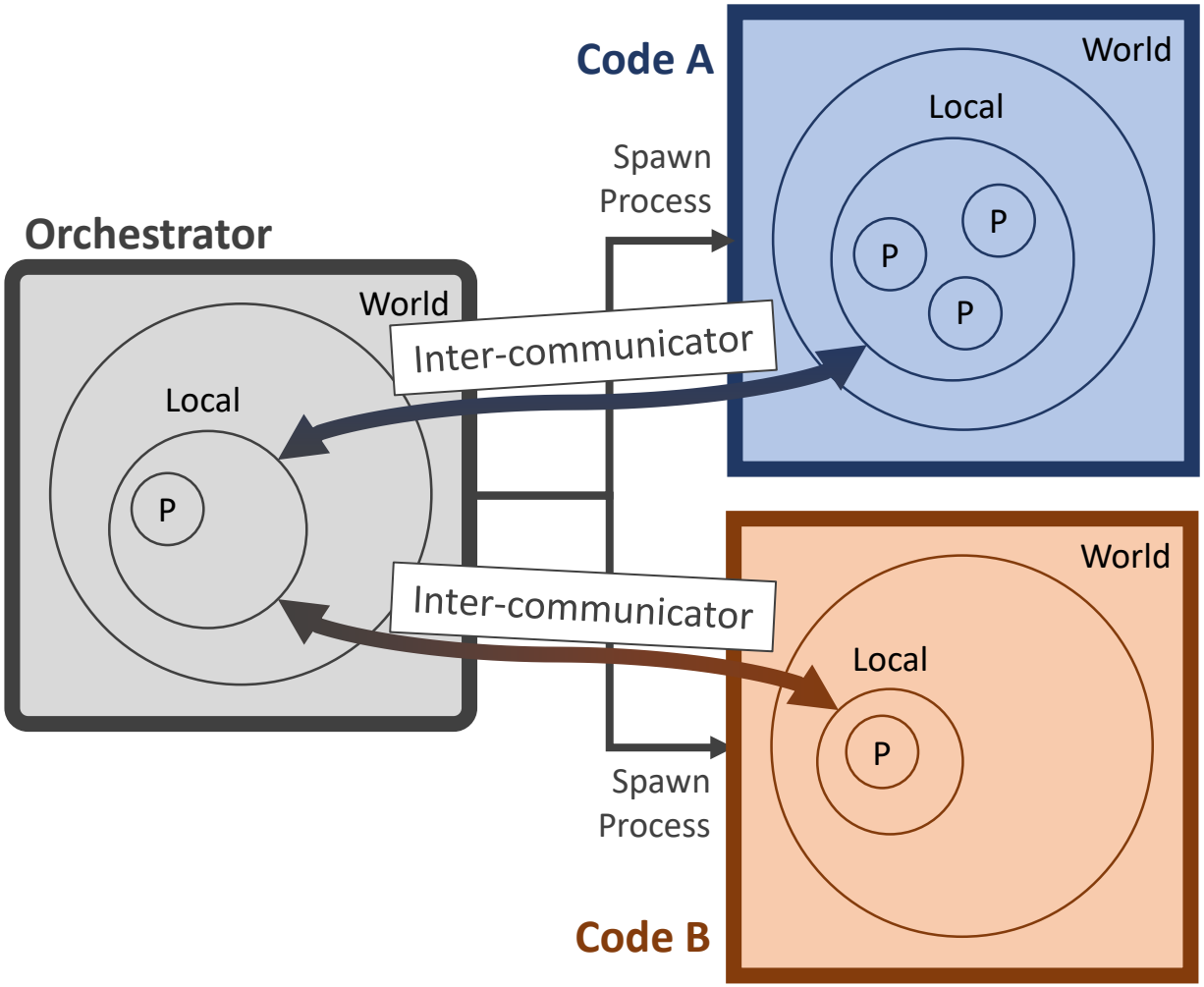


Introduction to High Performance Computing for Scientists and Engineers, doi: 10.55551855048 (2010)

- MPI를 활용한 프로세스 간 통신 (Inter-Communicator)



- MPI를 활용한 프로세스 간 통신 (Inter-Communicator)



▪ 개별 해석코드의 수치적 재현성 확보

- 해석코드의 V&V 가 수행되는 환경과 다를 경우
  - ✓ 운영체제 종류/버전, 컴파일러 종류/버전, 활용 라이브러리 버전에 따라 부동소수점 연산이 달라지는 특성으로 인해 수치적 재현이 어려움

- 해결 방법
  1. 해석코드의 계산 환경을 연계 환경으로 통합
     - 연계 환경에 맞춰 V&V 수행
     - 수치적 재현성 포기
  2. 해석코드별 운영체제를 유지하여 네트워크 통신을 통해 연계
     - 해석 코드별 전용 계산 노드 확보 후 연계
     - 통일되지 않은 환경 간 네트워크 통신 수행 (Imbalance)
  3. 해석코드별 운영체제를 유지하여 네트워크 통신을 통해 연계
     - 해석 코드의 환경을 모사하여 동일 환경에서 운용
     - 가상화에 따른 연산자원의 불필요한 낭비 문제 (Overhead)

03 연계 기술 비교

▪ 가상머신 기술

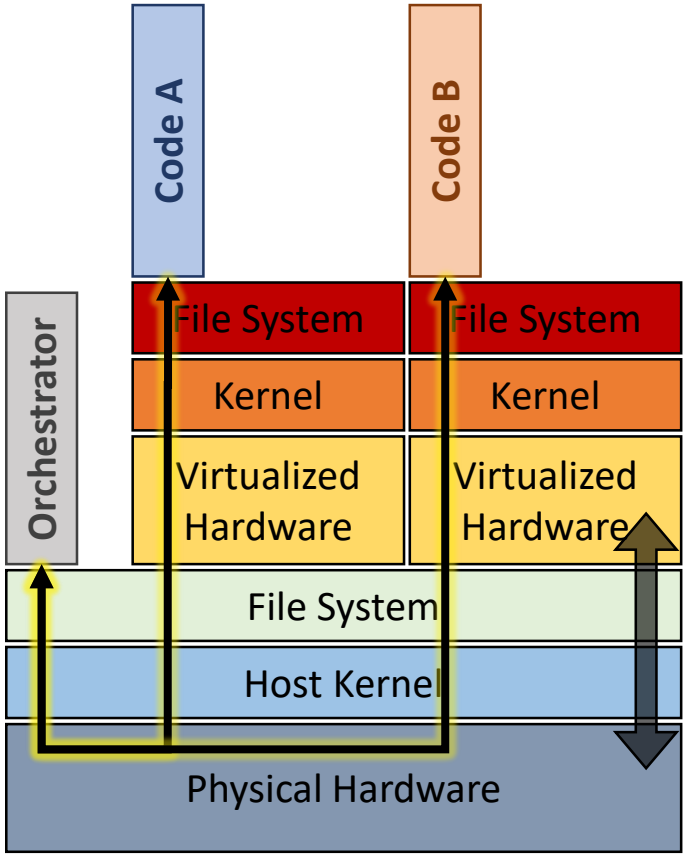
- 가상 컴퓨터 환경을 재현하는 기술
  - ✓ Host와 완전히 독립된 영역 (개별 해석코드의 V&V 머신과 동일한 환경)
  - ✓ 컴퓨터 한 대에 여러 가상 머신을 생성하여 운영 가능
- 완벽히 독립된 환경을 만들 수 있으나, 추가적인 성능 오버헤드가 생김

▪ 가상화 컨테이너 기술

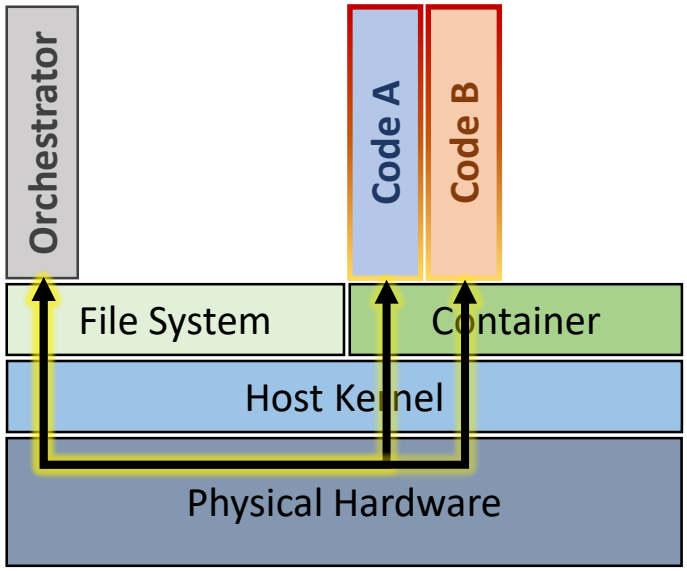
- 프로세스를 격리하는 기술
  - ✓ 가상화된 머신이 아닌 가상화된 환경을 격리하는 기술
  - ✓ 수치적 재현성은 소프트웨어 환경의 영향을 받으므로 프로세스 및 환경 격리로 충분함
- 독립된 환경 확보 및 성능 오버헤드 최소화 가능

* Linux 머신에서 Windows 를 가상화 할 경우 성능 오버헤드 존재

- 가상화 기술 비교



Virtual Machine Architecture



Container Architecture

04 요약

▪ 통합해석체계 개발 시 고려사항

- 연계 시뮬레이션 구성 방식
  - ✓ Code-to-Code
  - ✓ Centralized Orchestration

- 목적에 따른 연계 시스템 구성

  ex) 소규모 연산의 가벼운 Simulator

  대규모 연산의 Virtual Reactor

- 시스템 구성에 따른 활용 기술 선택
  - ✓ 통신: Message, Shared memory, Socket
  - ✓ 가상화: Virtual machine, Container

더 나은 세상을 위한 원자력기술

세계가 따라 배우는 원자력연구원

감사합니다

KAERI 한국원자력연구원
Korea Atomic Energy Research Institute