소프트웨어 아키텍처 설계 및 평가

궁 상 환

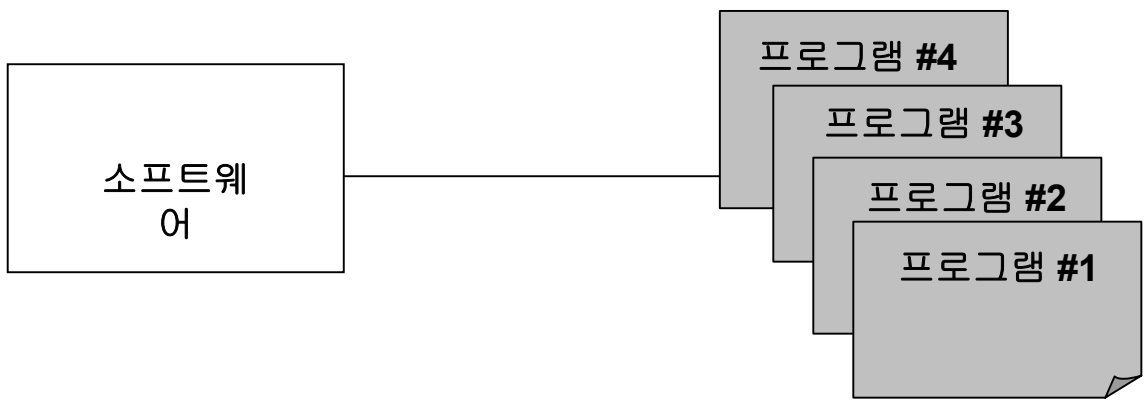
천안대학교

소프트웨어 아키텍처

개 요

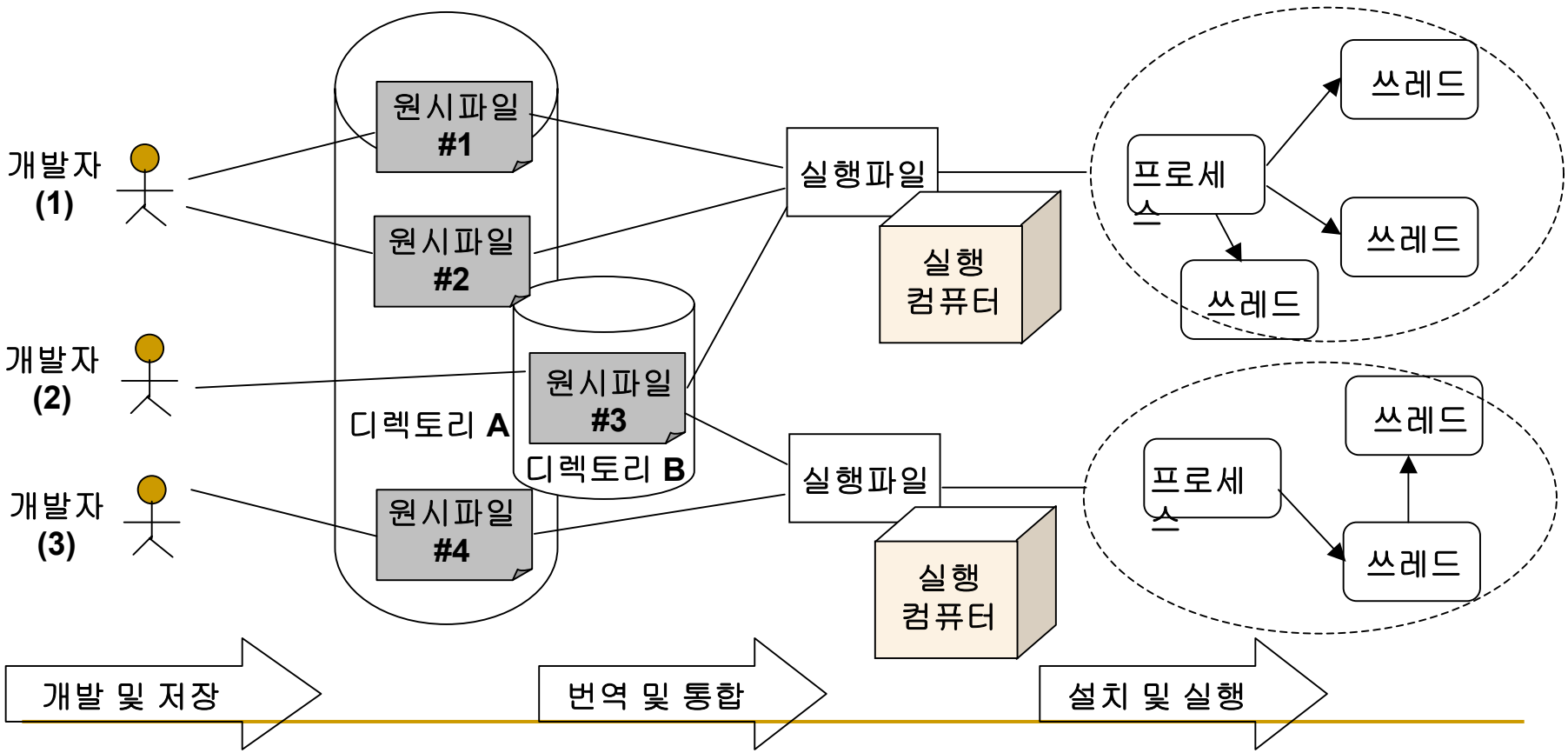
소프트웨어

- 협의적 의미
  - 프로그램의 집합
  - 프로그램은 소프트웨어의 가장 핵심적인 구성요소
  - 프로그램에는 프로그램 외에도 버퍼(메모리, 파일, DB)나 통신채널 등이 포함됨



소프트웨어의 모습

- 프로그램의 변화
  - 프로그램의 개발, 저장, 번역, 링크, 실행에 따라 모습이 변화

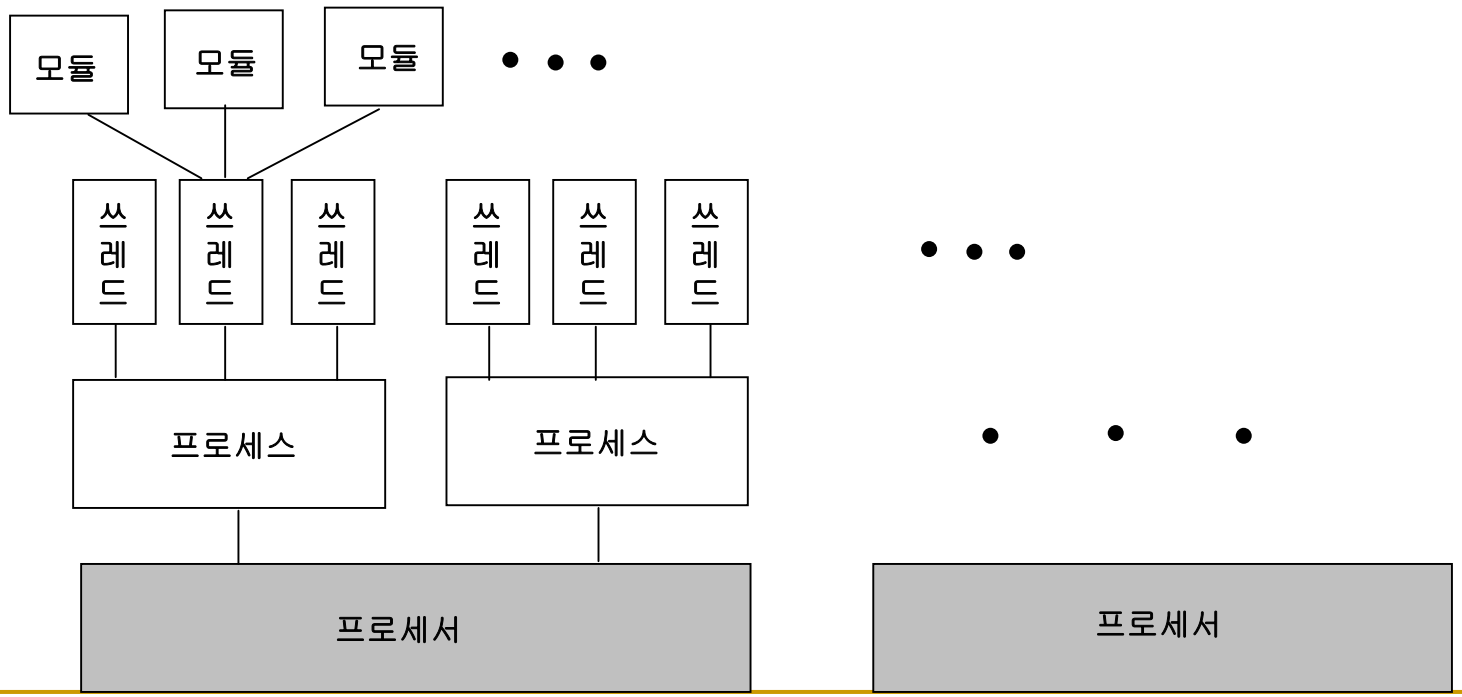


소프트웨어의 기본 구성요소

- 프로그램 모듈
- 버퍼 : 메모리, 디스크
- 상호작용

프로그램 모듈

- 모듈
  - 단위 : 객체나 함수, 기능블록, 서브시스템
- 프로세스/쓰레드
  - 실행 중인 프로그램, 실행 파일로부터 생성
  - 병렬적 처리

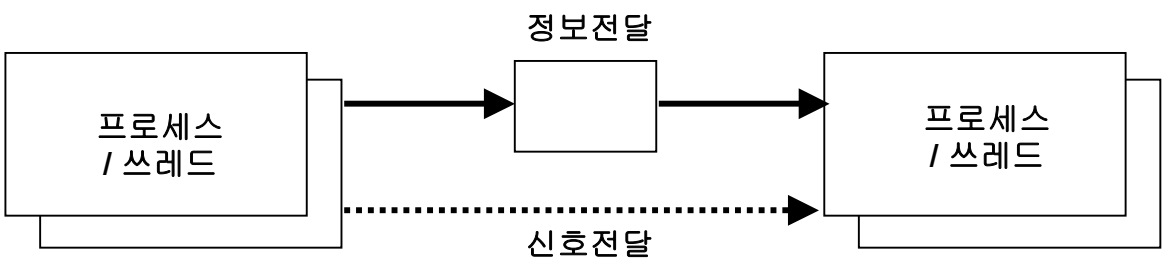


메모리 버퍼의 유형

- 공유 메모리(Shared Memory)
  - 복수의 프로세스가 정적인 정보를 공유
  - 예 : 전국 댐의 위험수위 표시
- 메시지 큐(Message Queue)
  - 복수의 프로세스간 계속적으로 전달되는 메시지
  - 예 : 주요 도로의 영상정보
- 소켓(Socket)
  - 메시지 큐와 동일한 Streaming 형태의 정보
  - 통신망 및 단일 시스템에서 복수의 프로세스나 쓰레드간 스트림 정보 전달

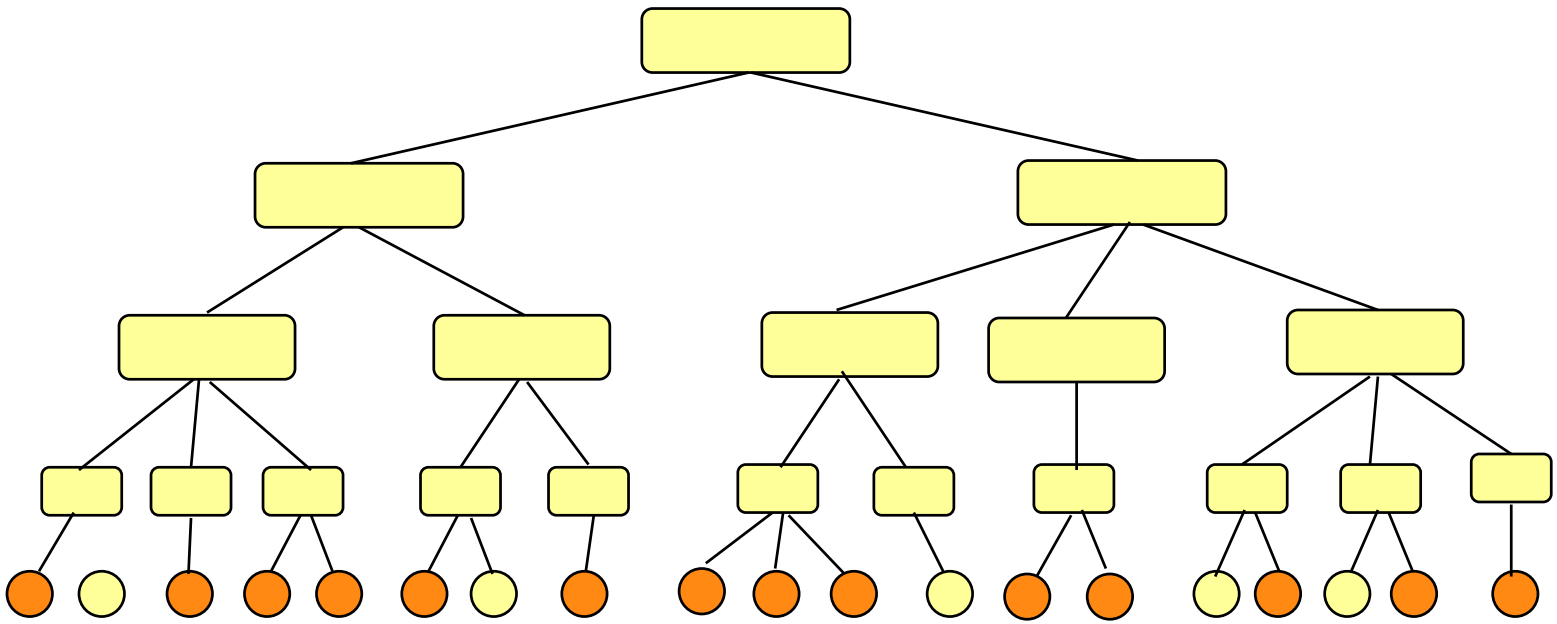
프로세스/쓰레드간 통신과 상호작용

- 필요성
  - 각각의 프로세스나 쓰레드는 동작 중에 다른 프로세스나 쓰레드의 활동을 알지 못함
  - 명시적인 통신 기능 필요
- Unix 환경
  - 정보전달 : pipe, shared memory
  - 신호전달 : signal 관련 함수(kill(), signal())
- Java 환경
  - 정보전달 : 복수의 쓰레드간 공통의 클래스 이용(Pipe 클래스도 일례)
  - 신호전달 : 신호전송 함수(wait(), notify() 등)



소프트웨어 재사용

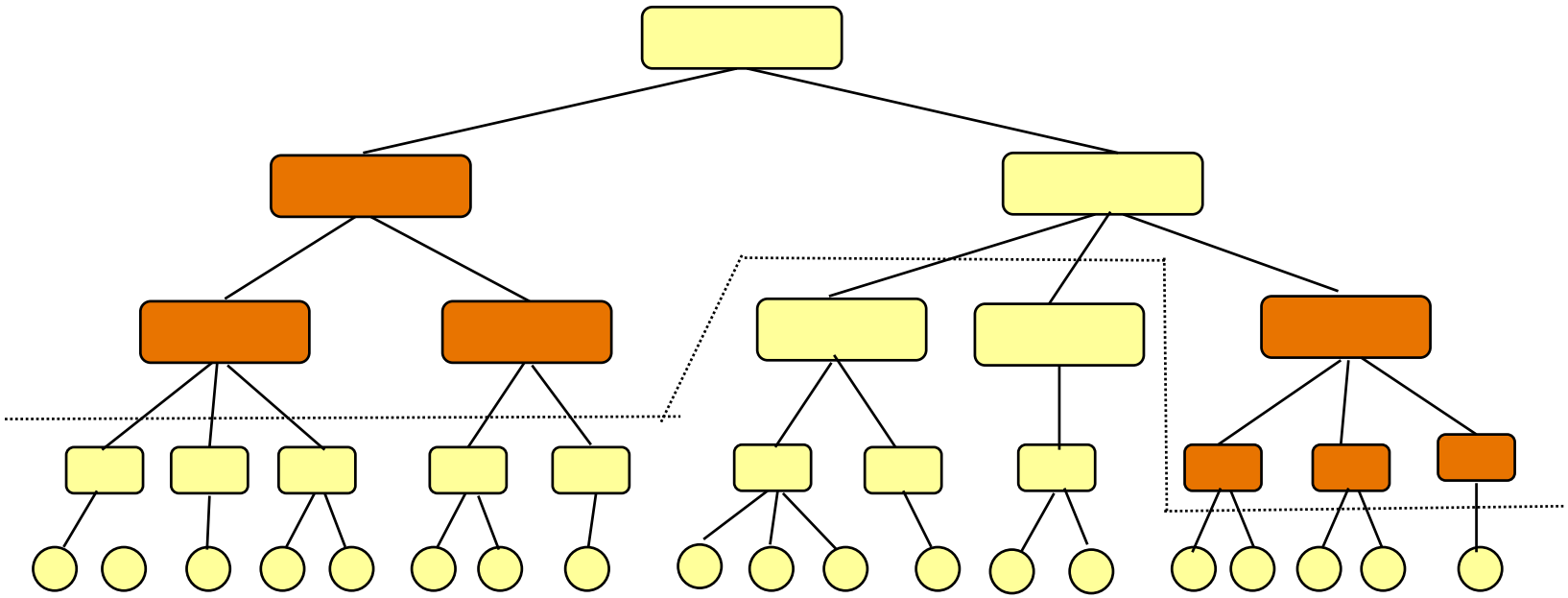
- 전통적인 재사용은 코드 모듈의 재사용에 초점을 맞춤



Library Reuse

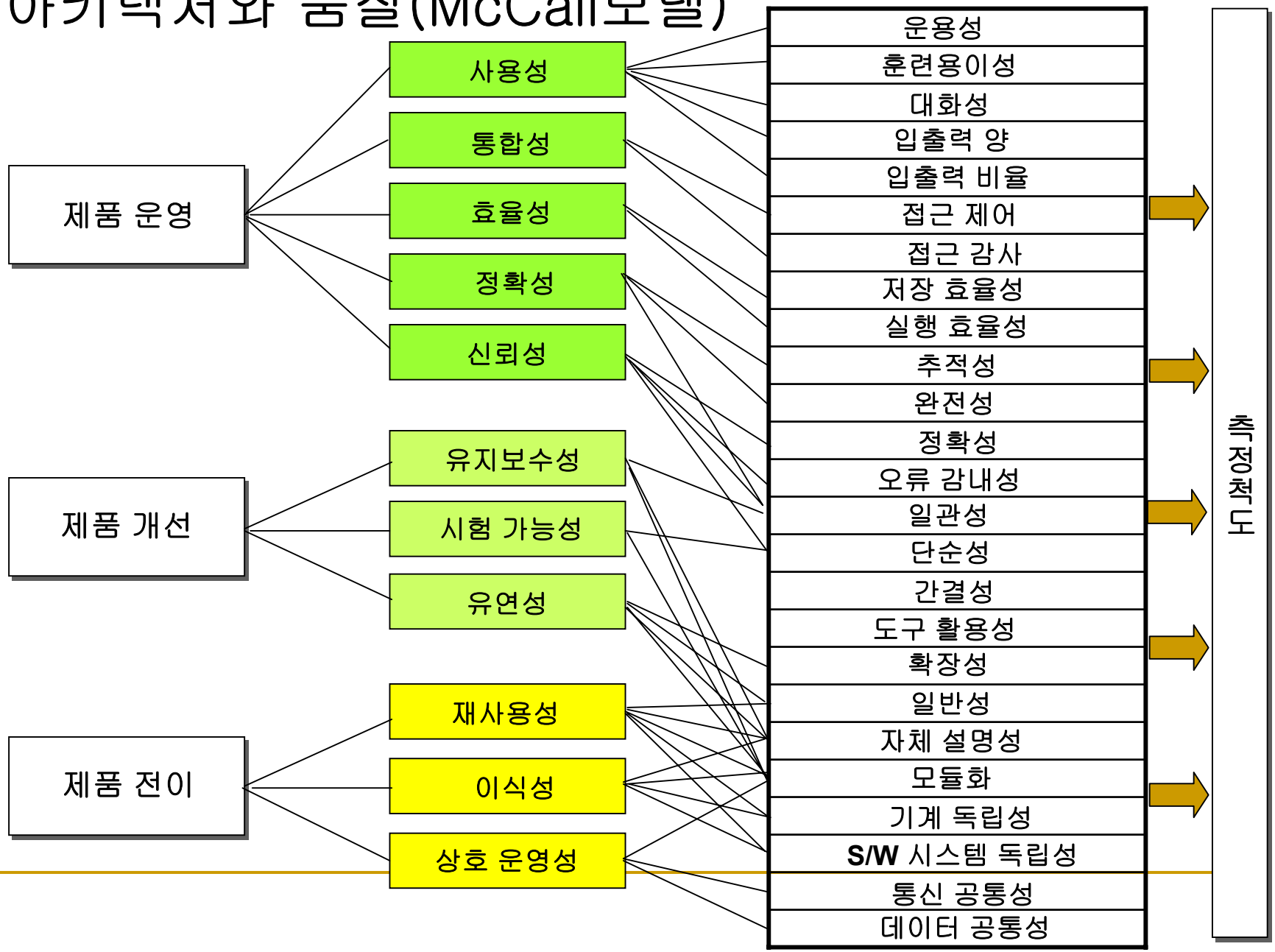
소프트웨어 재사용

- 아키텍처 재사용은 모듈의 집합인 컴포넌트와 설계의 재사용에 초점을 맞춤

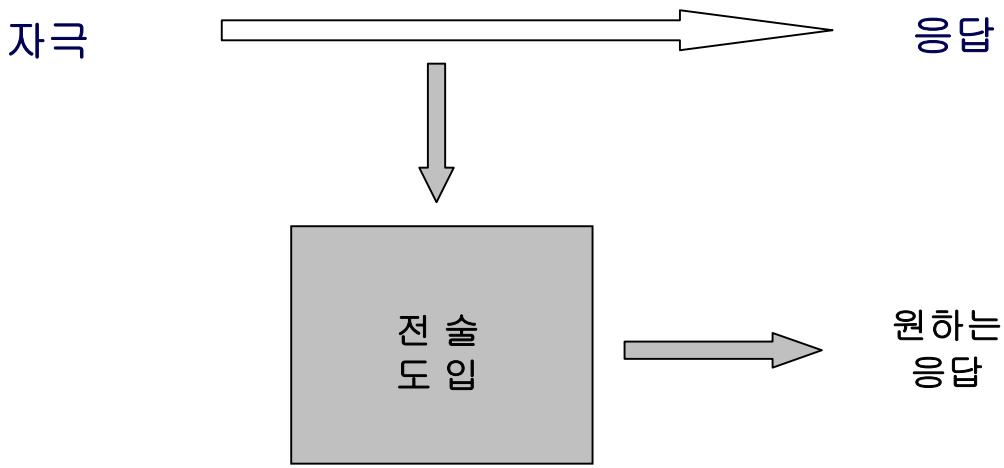


Architectural Styles/Patterns

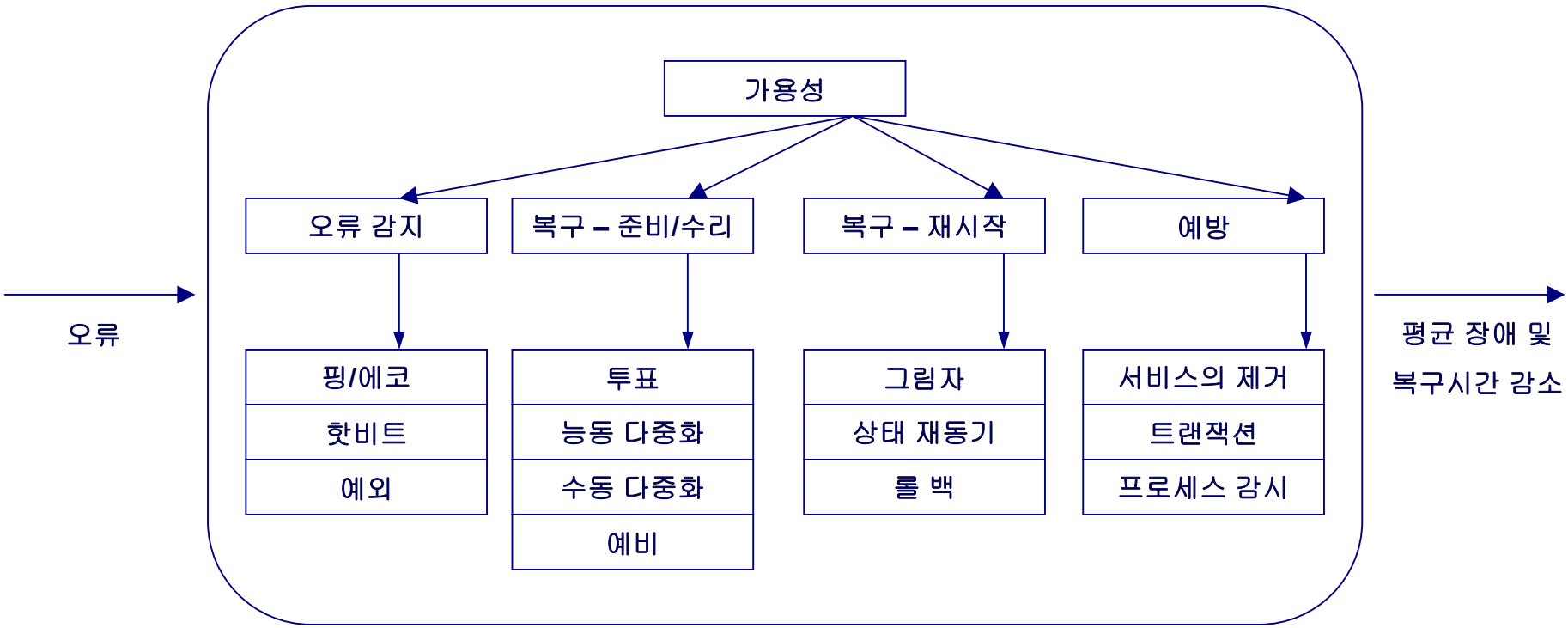
아키텍처와 품질(McCall모델)



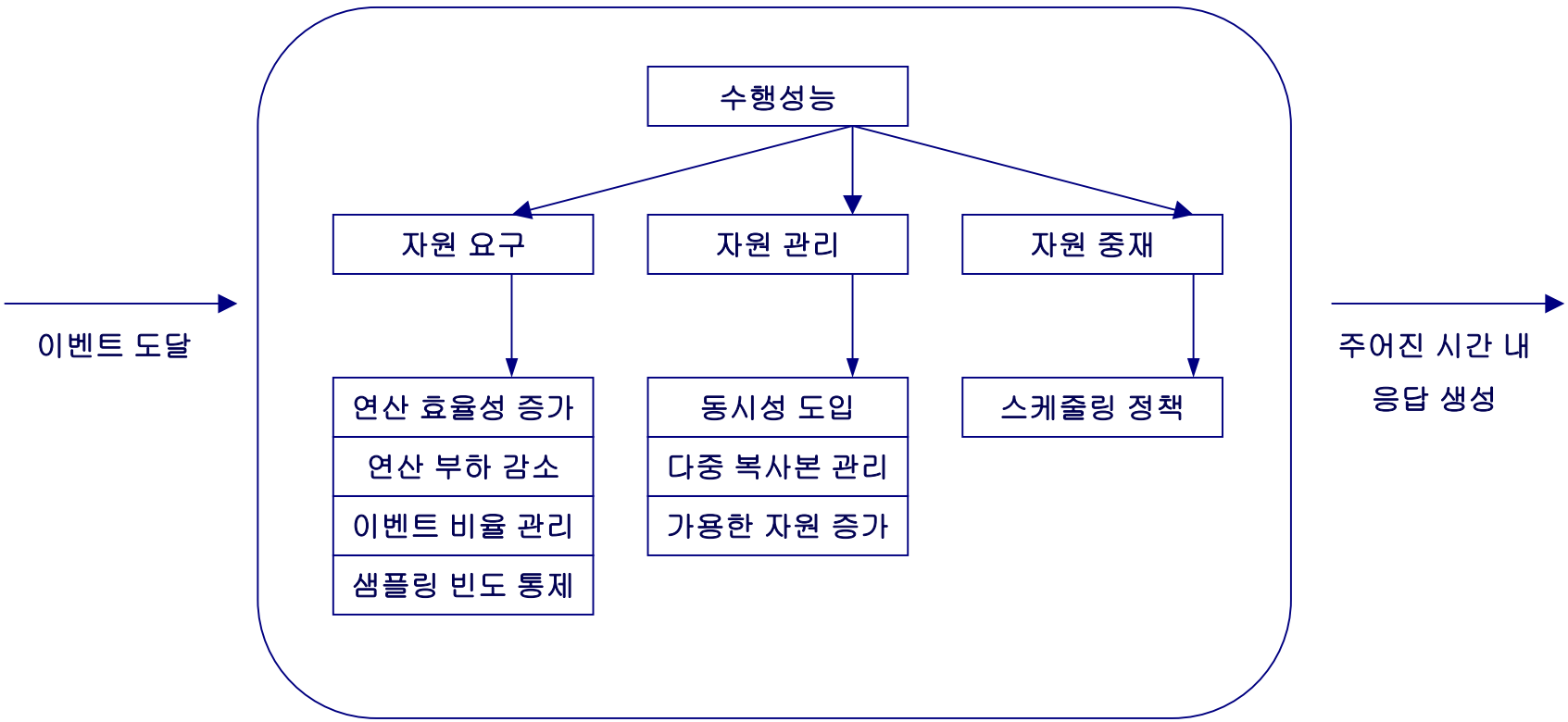
품질속성 달성방안



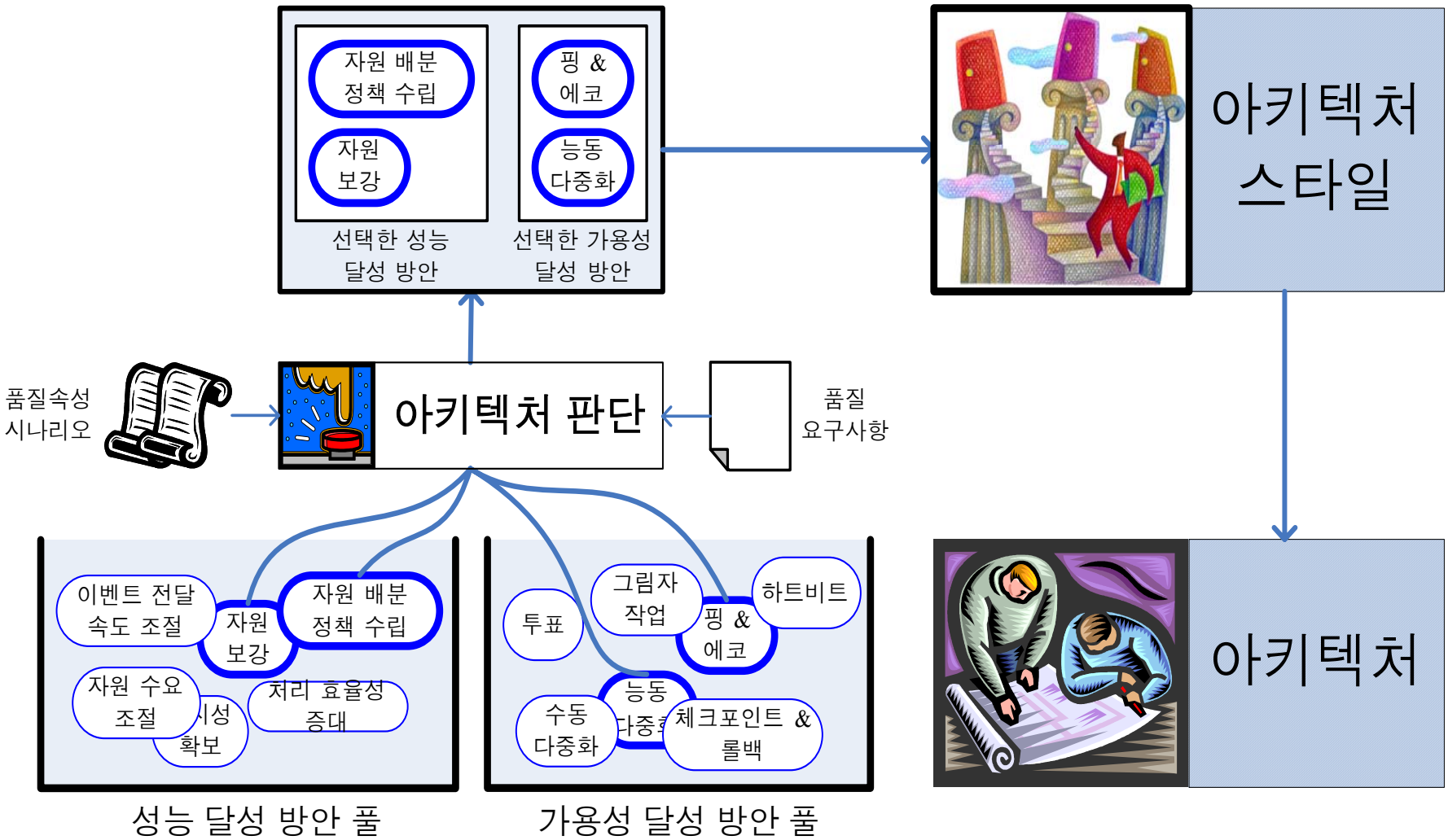
가용성 확보 전술


가용성
오류 감지
복구 – 준비/수리
복구 – 재시작
예방
핑/에코
핫비트
예외
투표
능동 다중화
수동 다중화
예비
그림자
상태 재동기
롤 백
서비스의 제거
트랜잭션
프로세스 감시
오류
평균 장애 및
복구시간 감소

성능 확보 전술


수행성능
자원 요구
자원 관리
자원 중재
연산 효율성 증가
연산 부하 감소
이벤트 비율 관리
샘플링 빈도 통제
동시성 도입
다중 복사본 관리
가용한 자원 증가
스케줄링 정책
이벤트 도달
주어진 시간 내
응답 생성

품질속성기반 아키텍처 설계



아키텍처 구조와 뷰타입

- 구조(structure)와 뷰타입(View Type)
  - 구조는 구성 요소의 집합 그 자체
    - 모듈 구조는 시스템 모듈의 집합과 이들의 조직관계
    - 아키텍처 문서화를 위한 **대표적인 구조**를 **뷰타입**이라고 함

주요 뷰타입

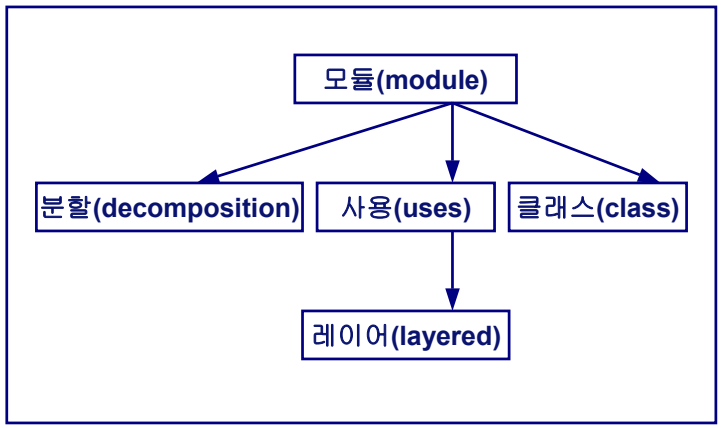
- 모듈(module) 뷰타입
  - 구성요소는 모듈이 됨 : 모듈은 구현의 단위 -> 시스템을 구현코드에 기반하여 바라 봄
  - 모듈에 할당된 기능이 중요
  - 프로그램의 동작관점은 미약

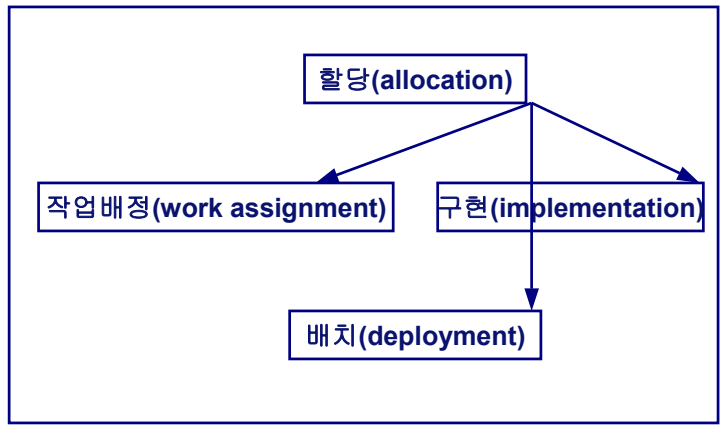
- 컴포넌트-커넥터(component-connector) 뷰타입
  - 컴포넌트는 실행 시간의 주요 연산 단위
  - 커넥터 = 컴포넌트 간의 통신 수단
  - 실행의 단위 컴포넌트, 공유자료, 병렬처리 등이 관심

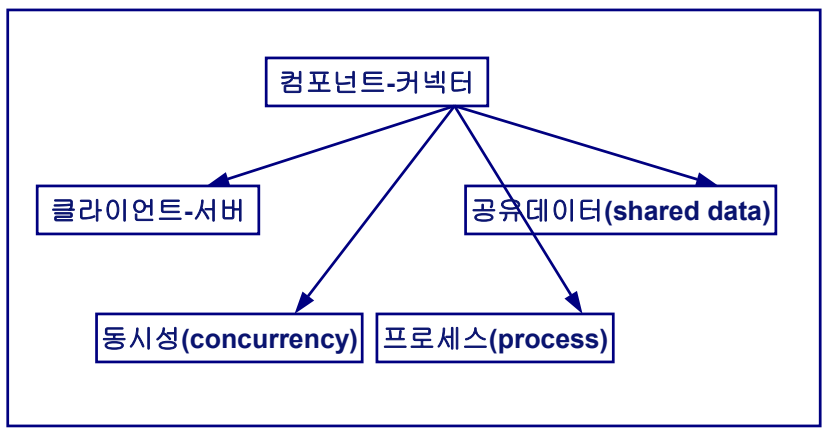
- 할당(allocation) 뷰타입
  - 소프트웨어 구성요소와 소프트웨어가 생성되고, 실행되는 외부 환경의 다른 요소들 사이의 관계
    - 소프트웨어 구성요소가 실행되는 프로세서(CPU)
    - 소프트웨어가 개발단계에서 저장되는 파일
    - 개발팀에의 소프트웨어 구성요소의 할당

뷰타입 요약

- 세 가지 구조 유형(다른 관점에서 바라 봄)





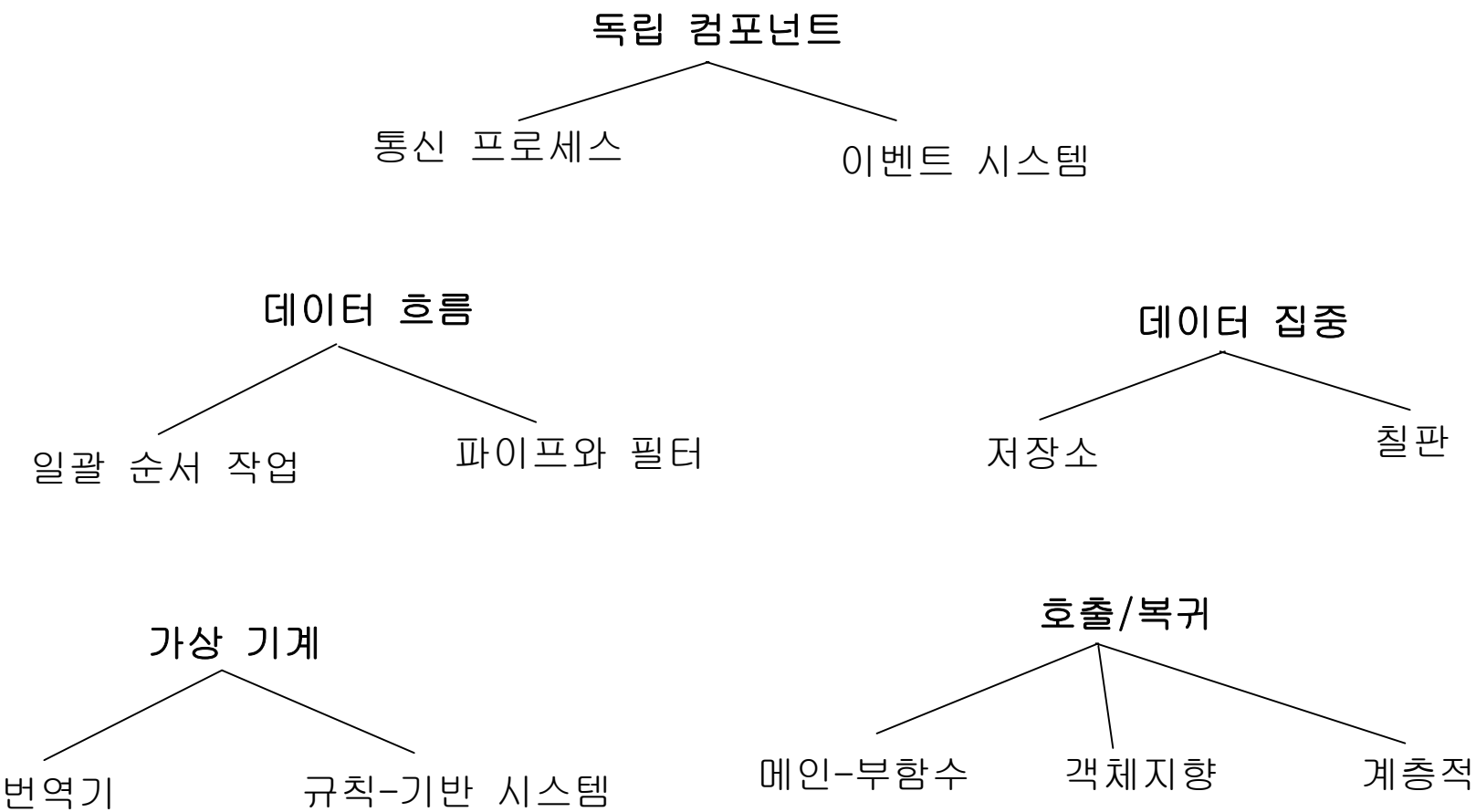


아키텍처 스타일

- 아키텍처 스타일/패턴
  - 많은 S/W 아키텍처 응용에서 많이 발견되는 공통된 스타일 또는 패턴
  - 패턴에 따라 대부분의 응용에 공통적으로 나타나는 유형과 특정한 응용을 중심으로 나타나는 패턴이 있음
  - 패턴의 요소가 보다 큰 골격을 위한 것과 세부적인 구성요소의 관계를 표현하는 것 등 다양한 레벨이 있음

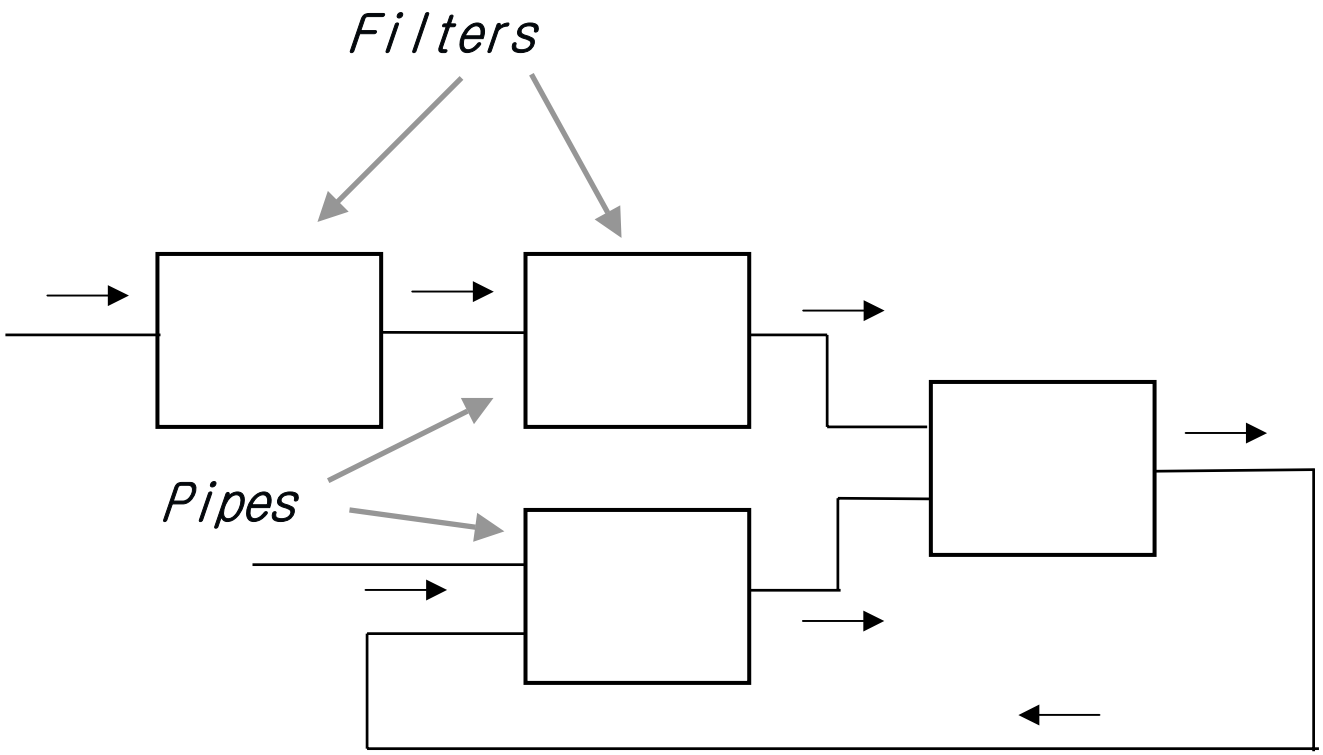
- 아키텍처 스타일은 다음을 정의
  - 구성 요소들
  - 구성 요소들의 상호 관계와 구성방식/배치
  - 구성 요소들의 정확한 의미와 제약사항
  - 구성 방식에 맞춰 구성요소들이 상호작용하는 메커니즘

아키텍처 스타일 패밀리


독립 컴포넌트
통신 프로세스
이벤트 시스템
데이터 흐름
일괄 순서 작업
파이프와 필터
데이터 집중
저장소
칠판
가상 기계
번역기
규칙-기반 시스템
호출/복귀
메인-부함수
객체지향
계층적

데이터 흐름(Data Flow)System

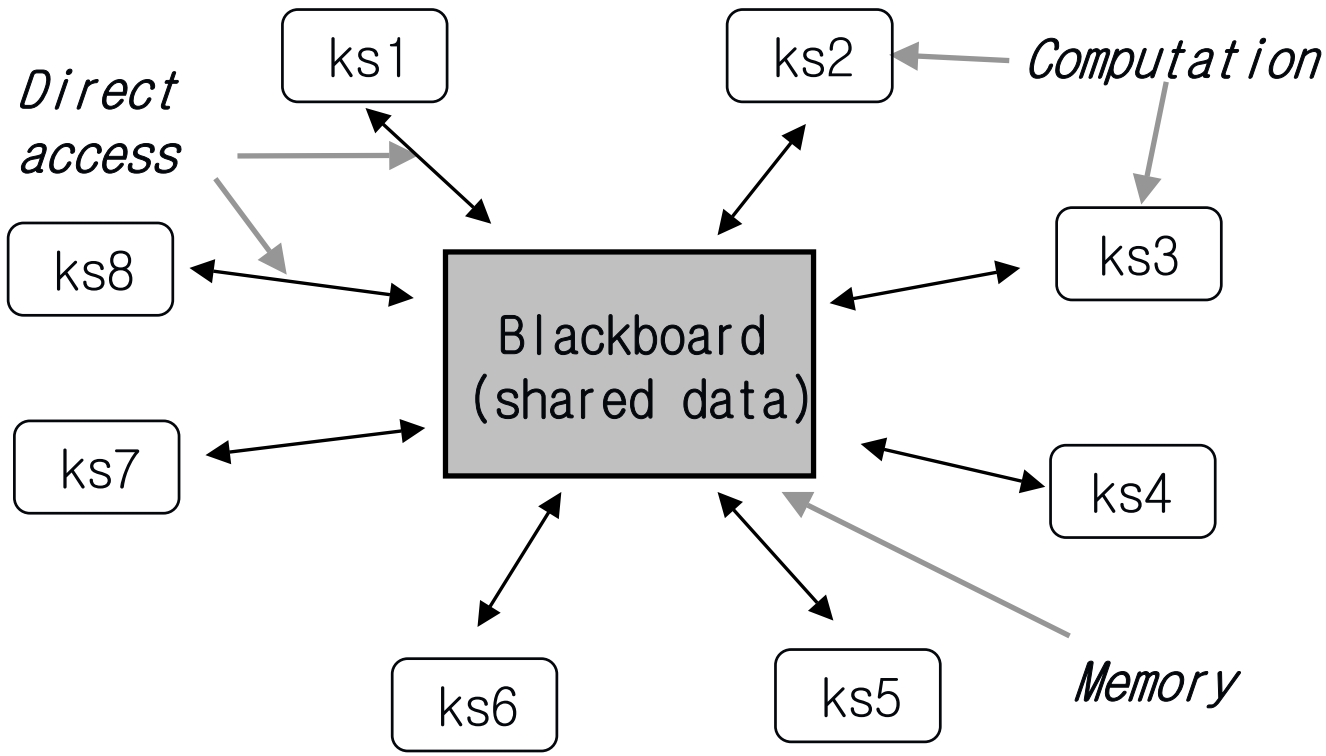
- 파이프와 필터(Pipes and filters)



데이터 집중(Data Centered) System

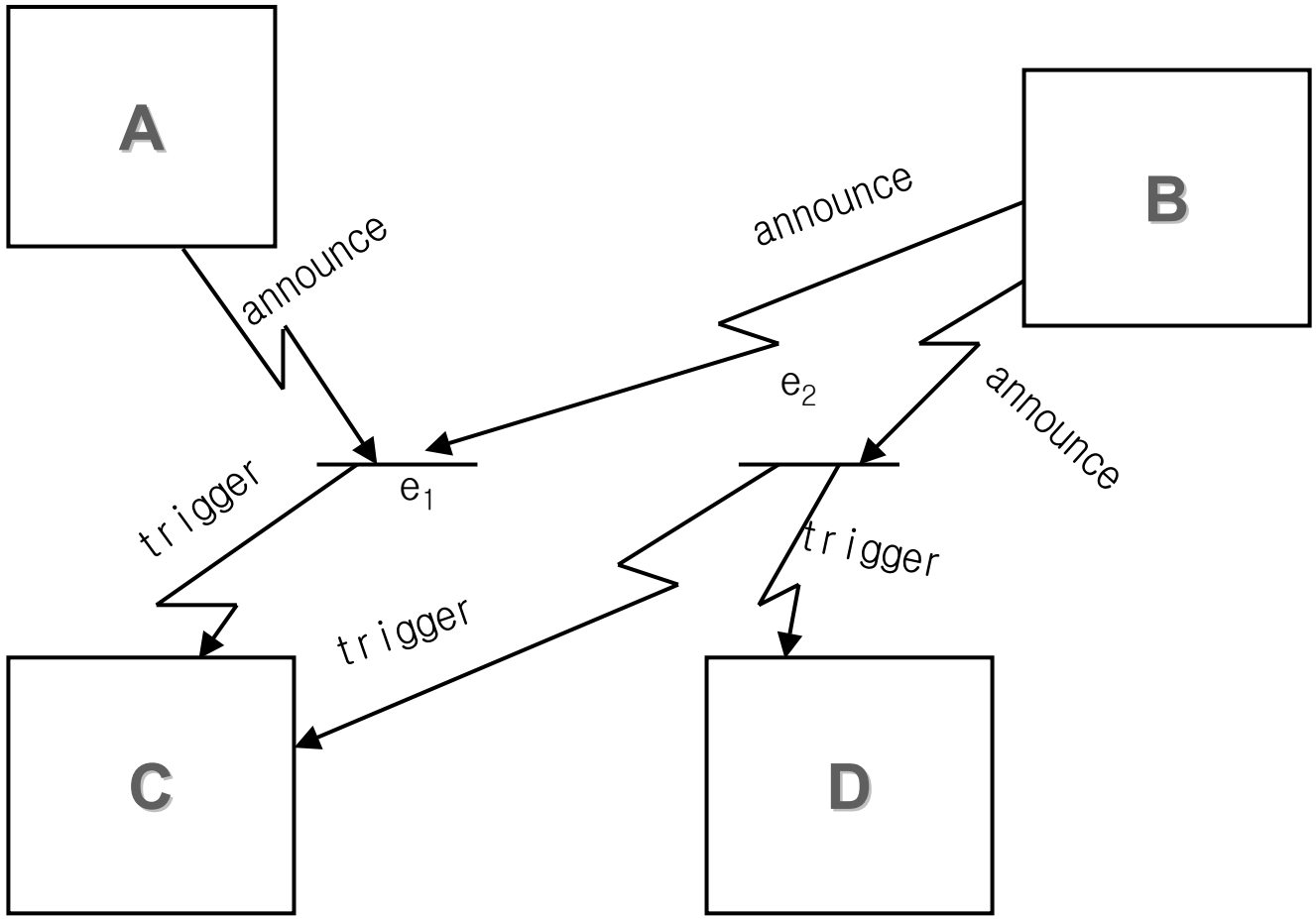
- Blackboard

ks : knowledge source



독립적인 컴포넌트

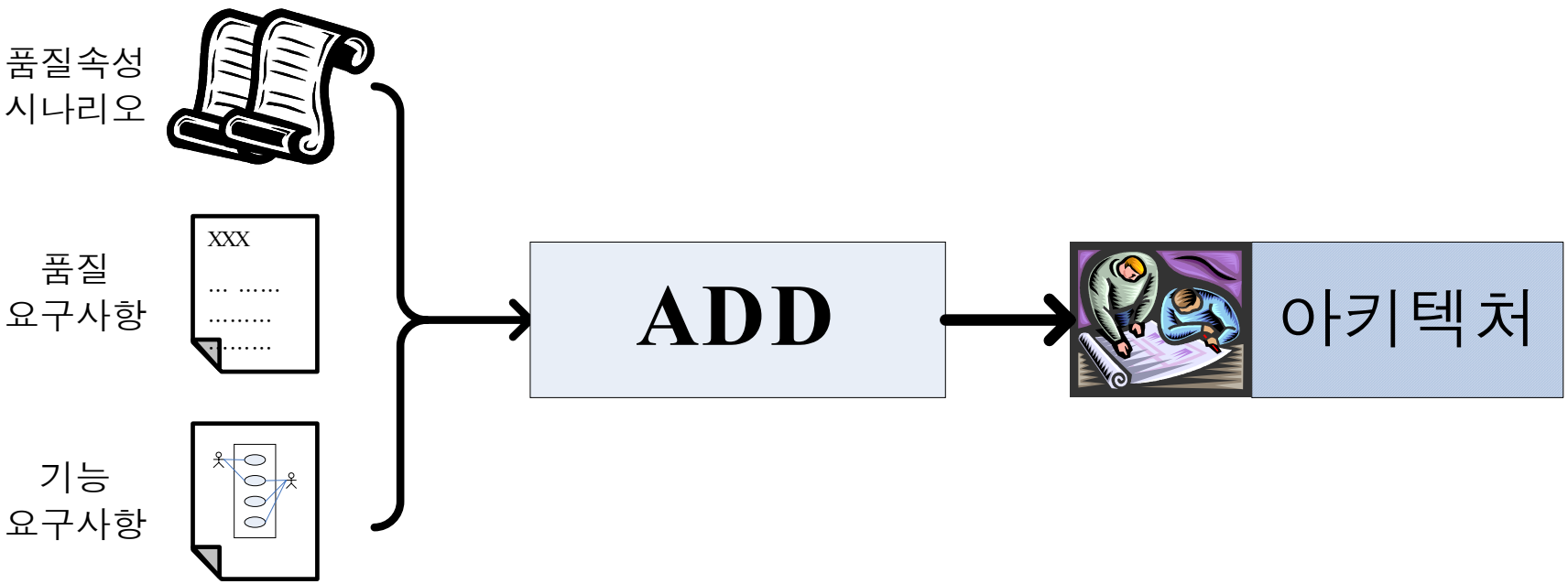
- Event based, Implicit Invocation



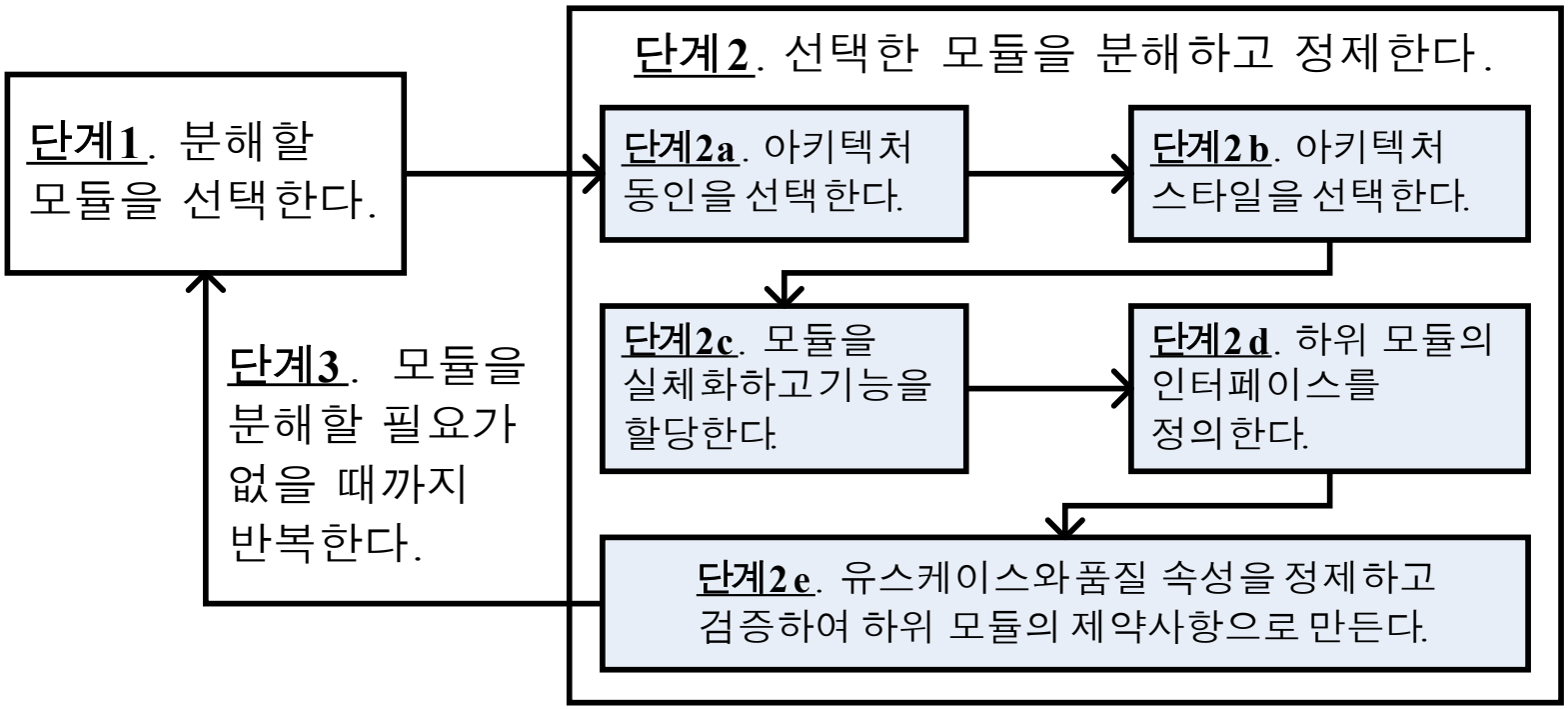
아키텍처 설계

ADD 정의

- ADD(Attribute-Driven Design)란?
  - 미국 SEI(Software Engineering Institute)에서 연구 개발한 소프트웨어 아키텍처 설계 방법



ADD 단계

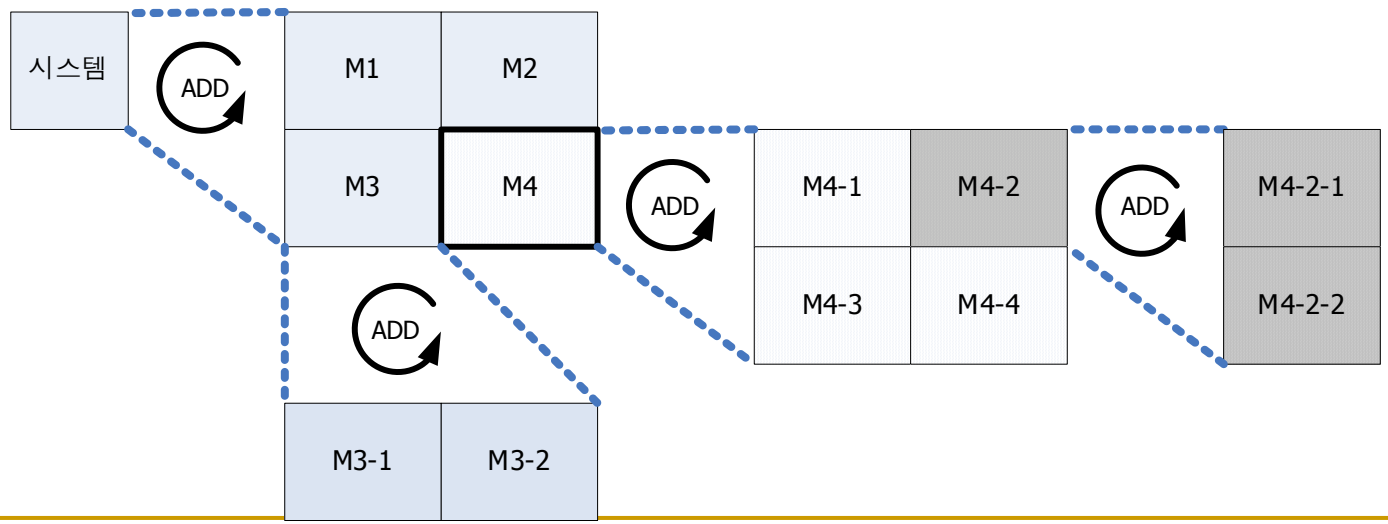

단계1. 분해할 모듈을 선택한다.
단계2. 선택한 모듈을 분해하고 정제한다.
단계2a. 아키텍처 동인을 선택한다.
단계2b. 아키텍처 스타일을 선택한다.
단계2c. 모듈을 실체화하고 기능을 할당한다.
단계2d. 하위 모듈의 인터페이스를 정의한다.
단계2e. 유스케이스와 품질 속성을 정제하고 검증하여 하위 모듈의 제약사항으로 만든다.
단계3. 모듈을 분해할 필요가 없을 때까지 반복한다.

ADD 단계

- 단계1. 분해할 모듈을 선택한다.
  - 설계 대상이 될 모듈을 선택하고 이 모듈의 제약조건, 품질 요구사항, 기능 요구사항도 정의한다.
  - 예제 시스템 : 기상정보수집 모듈 선택

ADD 단계

- 모듈 분해 과정
  - ADD를 제일 처음 시작할 때 대상 모듈은 보통 전체 시스템이다. 전체 시스템을 적절한 수준의 서브시스템으로 분해하여 ADD를 실시한다.
  - 다음 반복에서는 서브시스템을 다시 적절한 수준의 모듈로 나눠 ADD를 실시한다.
  - 이 작업을 반복한다.



ADD 단계

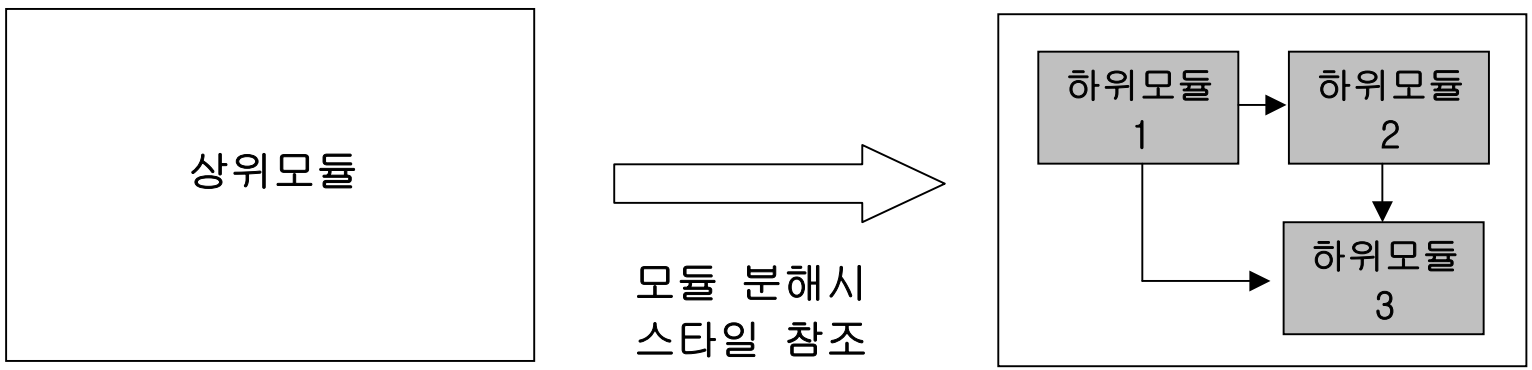
- 단계2. 선택한 모듈을 분해하고 정제한다.
  - 단계2a. 아키텍처 동인의 결정
    - 아키텍처가 다뤄야 할 기능 요구사항과 품질 요구사항을 결정
    - 많은 요구사항 가운데 제일 중요한 요구사항들을 뽑아서 아키텍처 동인으로 선택
    - 예제 적용 : 아키텍처 동인 결정
      - 기능 : 전국의 지역 센터로부터 전송되는 기상정보는 중앙센터의 DB에 수집이 되어야 한다.
      - 비기능 : 전송되는 기상정보는 실시간으로 수집이 가능하여야 한다. (성능)

ADD 단계

- 단계2b. 아키텍처 스타일을 선택한다.
  - 품질속성을 고려하여 적절한 아키텍처 스타일을 결정(성능, 변경가능성, 경제성 등)
  - 다양한 관점(View: 뷰)의 고려
    - 논리적 관점 : 기능의 분할, 재사용, 확장성
    - 동시성 관점 : 병행성, 성능
    - 배치 관점 : 설치 컴퓨터

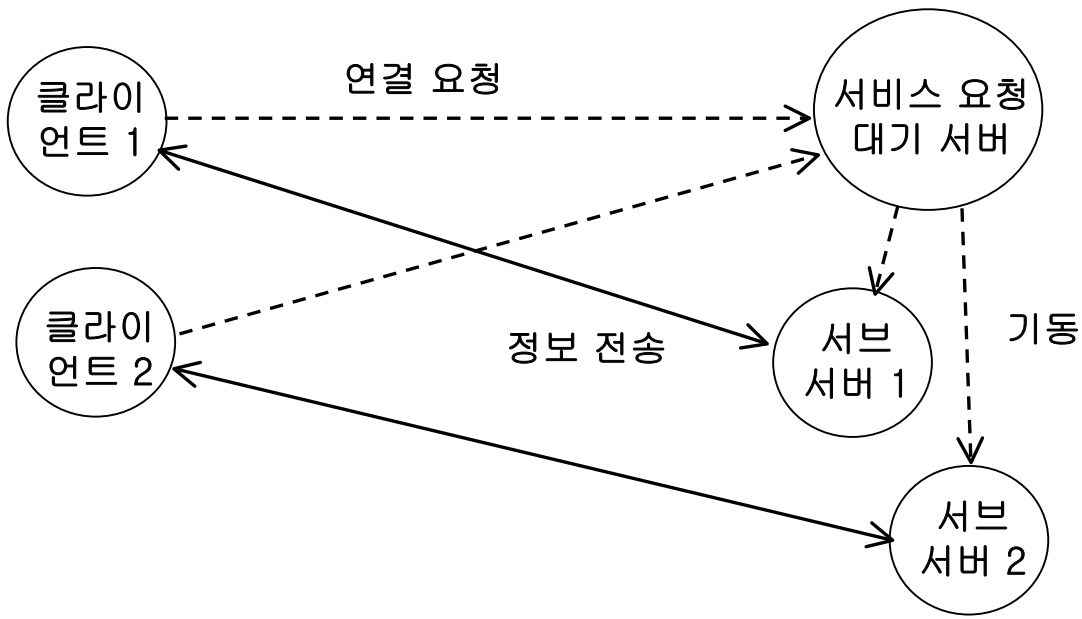
ADD 단계

- 단계2b. 아키텍처 스타일을 선택한다.
  - 스타일을 토대로 하위 모듈과 그들 사이에 관계 결정



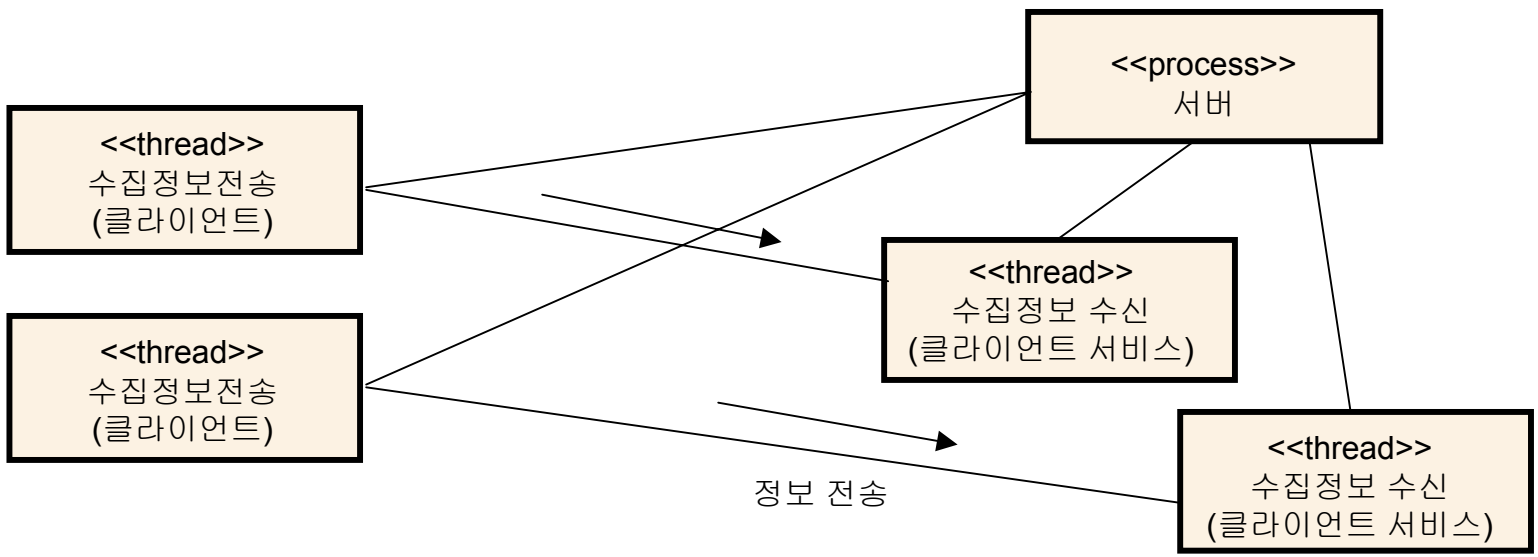
ADD 단계

- 단계2b. 아키텍처 스타일을 선택한다.
  - Communicating Process 스타일 : 실시간 성능 고려



ADD 단계

- 단계2c. 모듈을 실체화하고 기능을 할당한다.
  - 선택한 아키텍처 스타일이 정의한 모듈들로 목표 시스템의 기능들을 매핑
  - 예제 적용
    - 아키텍처 스타일 적용 : Communicating Process
    - 클라이언트 서비스 : 수집 기능할당
    - 클라이언트 : 전송 기능 할당

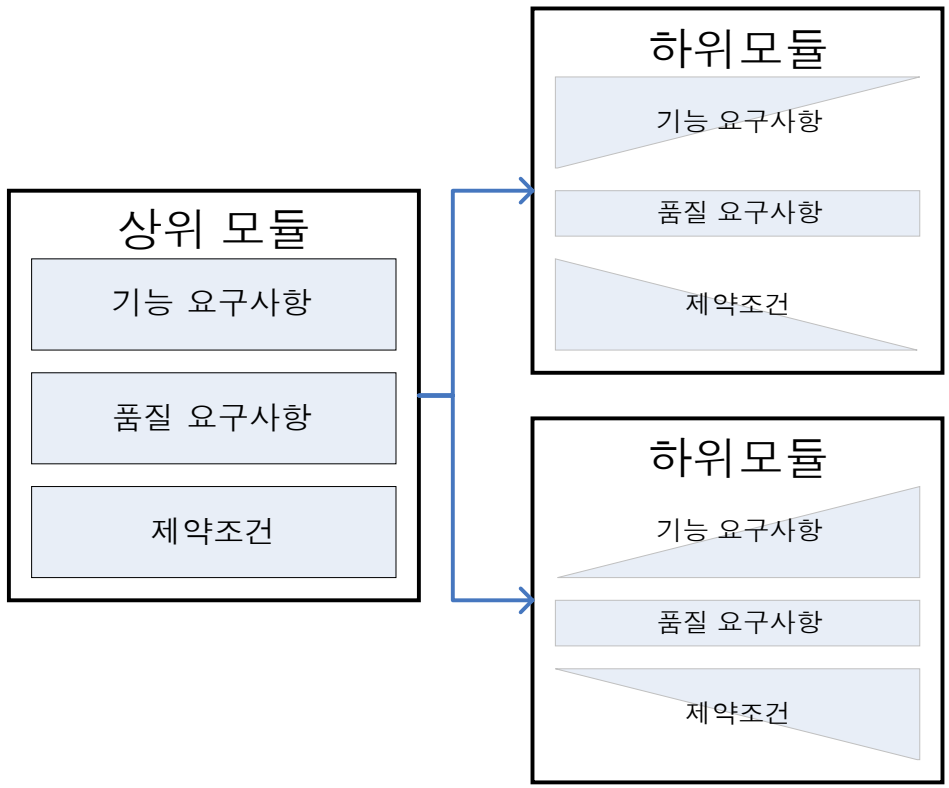


ADD 단계

- 단계2d. 하위 모듈의 인터페이스를 정의한다.
  - 각 하위 모듈이 제공하는 서비스와 제공받아야 하는 서비스를 정의한다.
  - 예제 응용
    - 모듈간 인터페이스 : 일관된 형태의 메시지 객체 정의
    - 메시지 유형 : 측정 메시지, 확인 메시지, 통제 메시지 등

ADD 단계

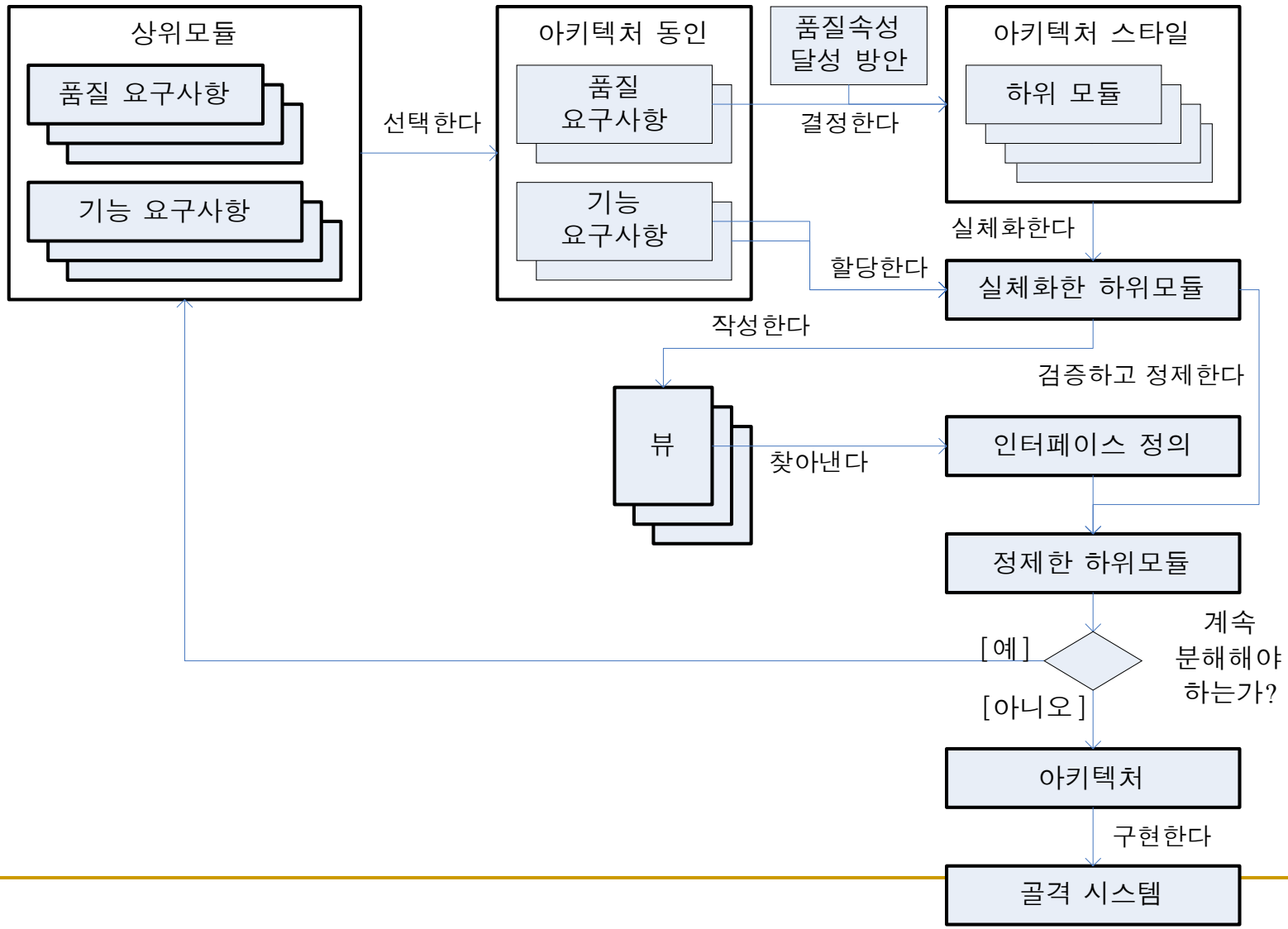
- 단계2e. 유스케이스와 품질요소를 정제하고 검증해서 하위 모듈의 제약사항으로 바꾼다.
  - 예제 시스템 : 기상정보 전송 성능, 수집성능 등



ADD 단계

- 단계3. 모듈을 분해할 필요가 없을 때까지 반복한다.
  - 모듈을 더 분해해야 하는지 판단하고 더 분해해야 한다면 단계2에서 정제가 끝난 하위 모듈이 단계1의 상위 모듈이 되어 ADD 단계를 반복한다.

ADD 산출물 흐름


상위모듈
품질 요구사항
기능 요구사항
선택한다
아키텍처 동인
품질
요구사항
기능
요구사항
품질속성
달성 방안
결정한다
아키텍처 스타일
하위 모듈
실체화한다
할당한다
실체화한 하위모듈
작성한다
검증하고 정제한다
뷰
찾아낸다
인터페이스 정의
정제한 하위모듈
[예]
계속
분해해야
하는가?
[아니오]
아키텍처
구현한다
골격 시스템

아키텍처 평가

아키텍처 평가 시기

- 초기(Early)
  - 아키텍처가 완전히 정의될 때까지 기다릴 필요 없이, 아키텍처를 정의하는 과정에서 평가가 가능
  - Discovery Review : very early mini-evaluation
    - 요구사항이 고정되기 전에 수행
    - 산출물로는 좀 더 견고한 요구사항 명세와 요구사항을 만족
    - 시키기 위한 초기 아키텍처 접근법들
- 말기(Late)
  - 아키텍처가 고정되고 구현이 끝난 다음에 평가
  - 제품을 구매하였거나 조직의 자산(기존 Legacy) 시스템을 사용할 때 이런 평가 방법을 사용
- 단, 평가를 할 수 있을 만큼의 충분한 아키텍처가 존재해야 한다.

아키텍처 평가자

- 평가 팀
  - 평가를 수행하고 분석을 수행하는 팀
- 이해관계자
  - 아키텍처와 아키텍처로부터 만들어질 시스템에 관심을 가지고 있는 이해관계 당사자들
    - 아키텍처가 해결해야 하는 요구사항을 제시하는 사람들
  - 아키텍트/컴포넌트 설계자, 프로젝트 관리자, 고객, 스폰서 등
  - 의사결정권자(project decision maker)도 이해관계자가 됨

아키텍처 평가의 대상

- 단일 아키텍처
  - 제시된 하나의 아키텍처에 대한 적절성 검증
- 경쟁관계에 있는 여러 아키텍처 대안
  - 관심(평가) 영역을 정해놓고 이 영역에 있어서, 대안 아키텍처의 장,단점을 조사
  - 관리자는 어떤 아키텍처를 선택 혹은 개선할 것인지, 새로운 아키텍처를 설계할 것인지 결정

아키텍처 접근법(Architectural Approach)

- 품질속성을 달성하기 위한 아키텍처 모델/스타일
- 아키텍처 스타일
  - 구성
    - 컴포넌트 타입 : 데이터저장소, 프로세스, 객체
    - 콘넥터 타입과 상호작용 방식 : 파이프, 이벤트
    - 컴포넌트의 배치
    - 배치와 행위의 제약사항
    - 비용과 이득의 명시
- 아키텍처 접근법
  - 아키텍처 스타일이 특정한 응용에서 활용 될 때
  - 시스템의 구조 외에도, 확장이나, 변경, 타 시스템과의 통합에 관한 방안 제시

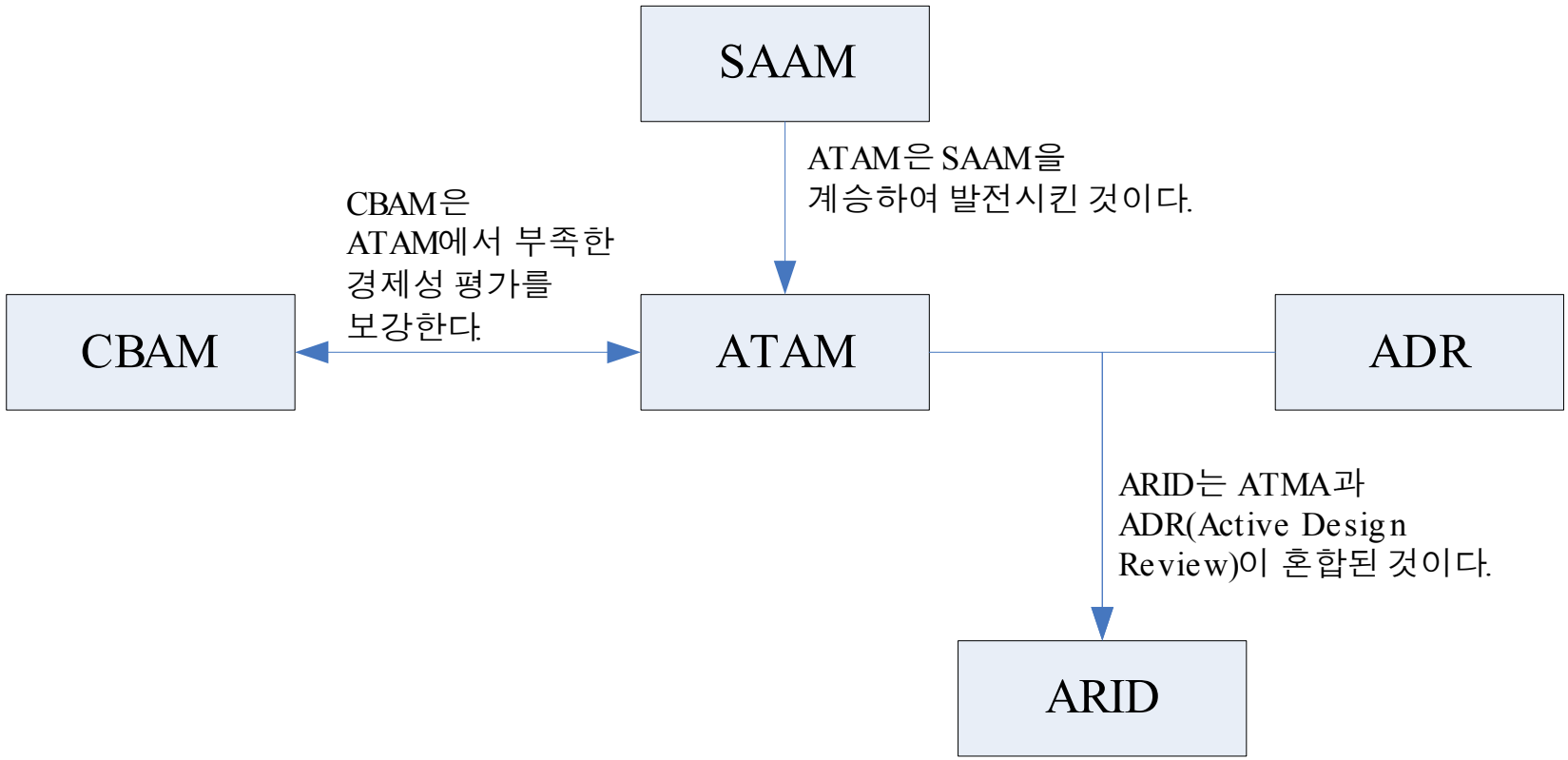
아키텍처 평가 산출물

- 우선순위가 정해진 품질속성 요구사항
- 사용된 아키텍처 접근법 카탈로그(목록)
  - 아키텍트가 채택한 설계전략이나 접근법들의 목록
- 아키텍처 접근법과 품질속성 간의 매핑
  - 아키텍처 접근법이 어떻게 품질속성 요구사항을 달성할 수
  - 있는지 (혹은 달성할 수 없는지) 보여주는 문서
- 품질속성과 아키텍처 접근법을 찾기 위해 사용된 질문
  - 미래에 아키텍처가 발전할 때 재활용

- 위험요소(Risks)와 비위험요소(Non-Risks)
  - 위험요소 : 문제의 소지를 가지고 있는 아키텍처 결정사항
  - 비위험요소 : 대체적으로 아키텍처 설계에서 수행되어 온 사례를 가정으로 만들어진 좋은 결정사항

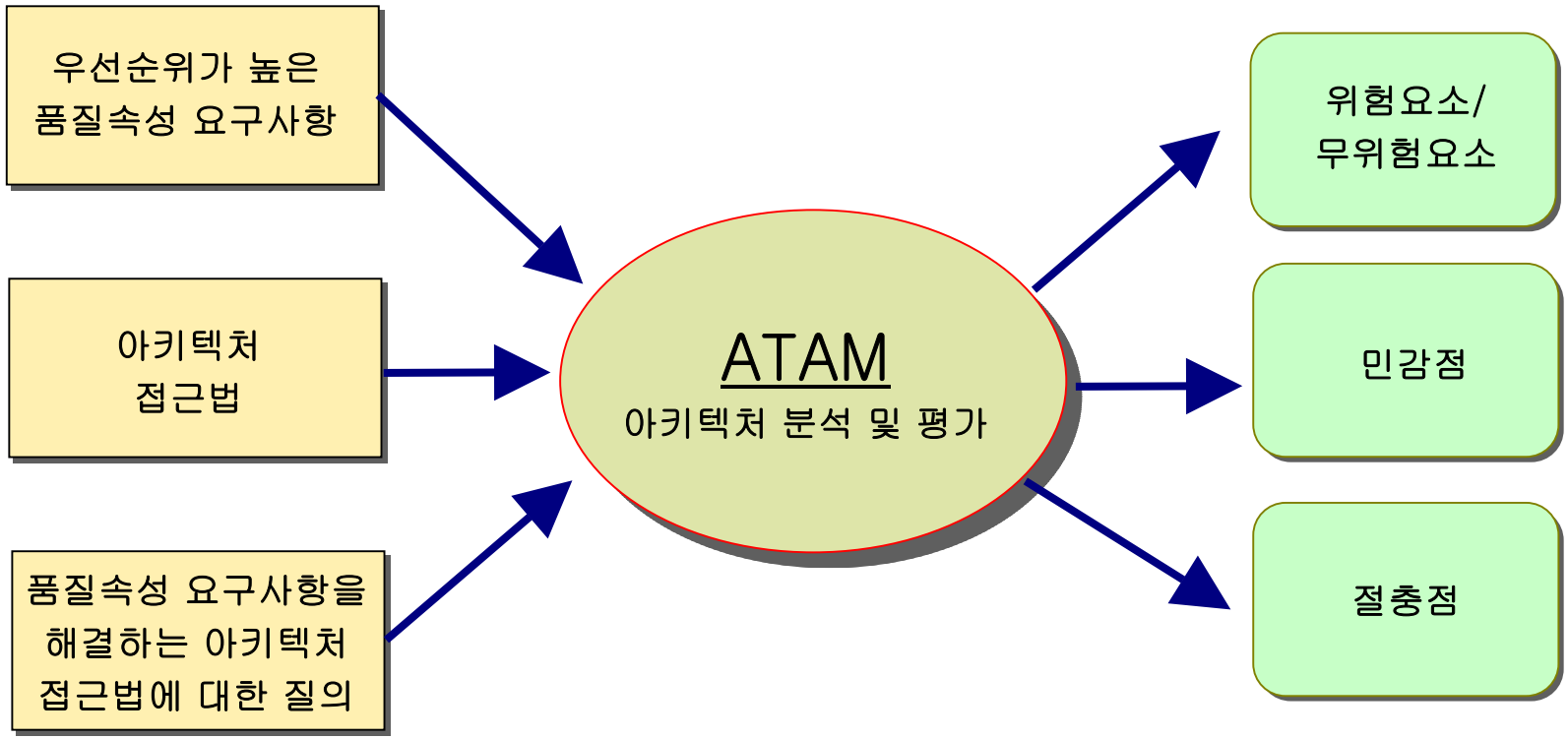
- 민감점(Sensitivity points)과 절충점(Tradeoff points)
  - 민감점 : 특정한 품질속성을 달성하는데 있어서 중요한 요소로 작용하는 속성
    예) 중요한 메시지의 처리 지연
  - 절충점 : 특정 품질속성에 영향을 끼침과 동시에 다른 품질속성에는 민감요소로 작용하는 컴포넌트 속성
    예) 암호화 수준의 변경은 보안과 성능에 영향을 미침. 암호화 수준을 높이면 보안성은 좋아 지지만, 성능은 떨어짐

평가방법과 유형

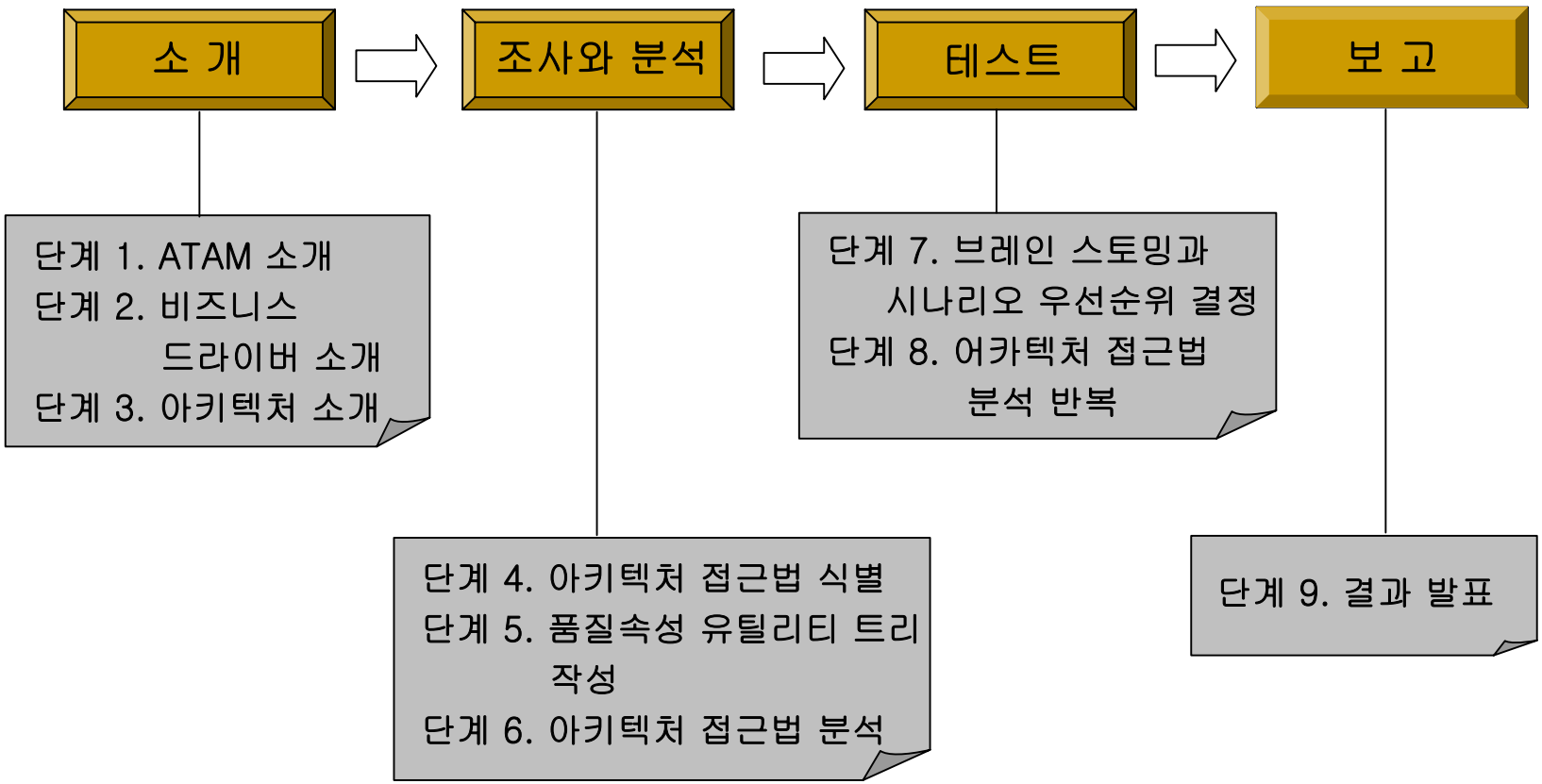


- ATAM : Architecture Tradeoff Analysis Method
- SAAM : Software Architecture Analysis Method
- ARID : Active Reviews for Intermediate Designs
- ARD : Active Design Review
- CBAM : Cost Benefit Analysis Method

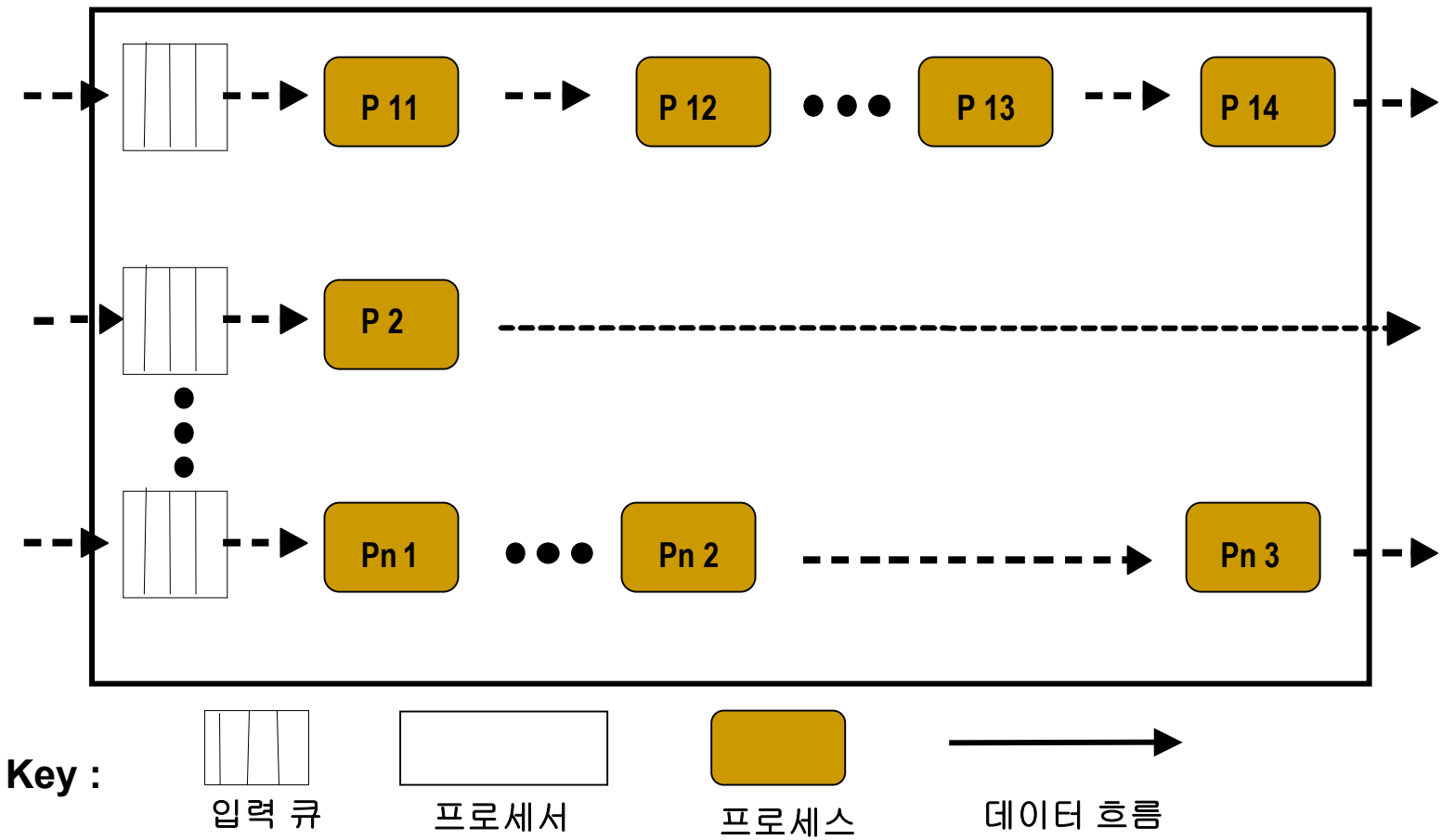
ATAM


우선순위가 높은
품질속성 요구사항
아키텍처
접근법
품질속성 요구사항을
해결하는 아키텍처
접근법에 대한 질의
ATAM
아키텍처 분석 및 평가
위험요소/
무위험요소
민감점
절충점

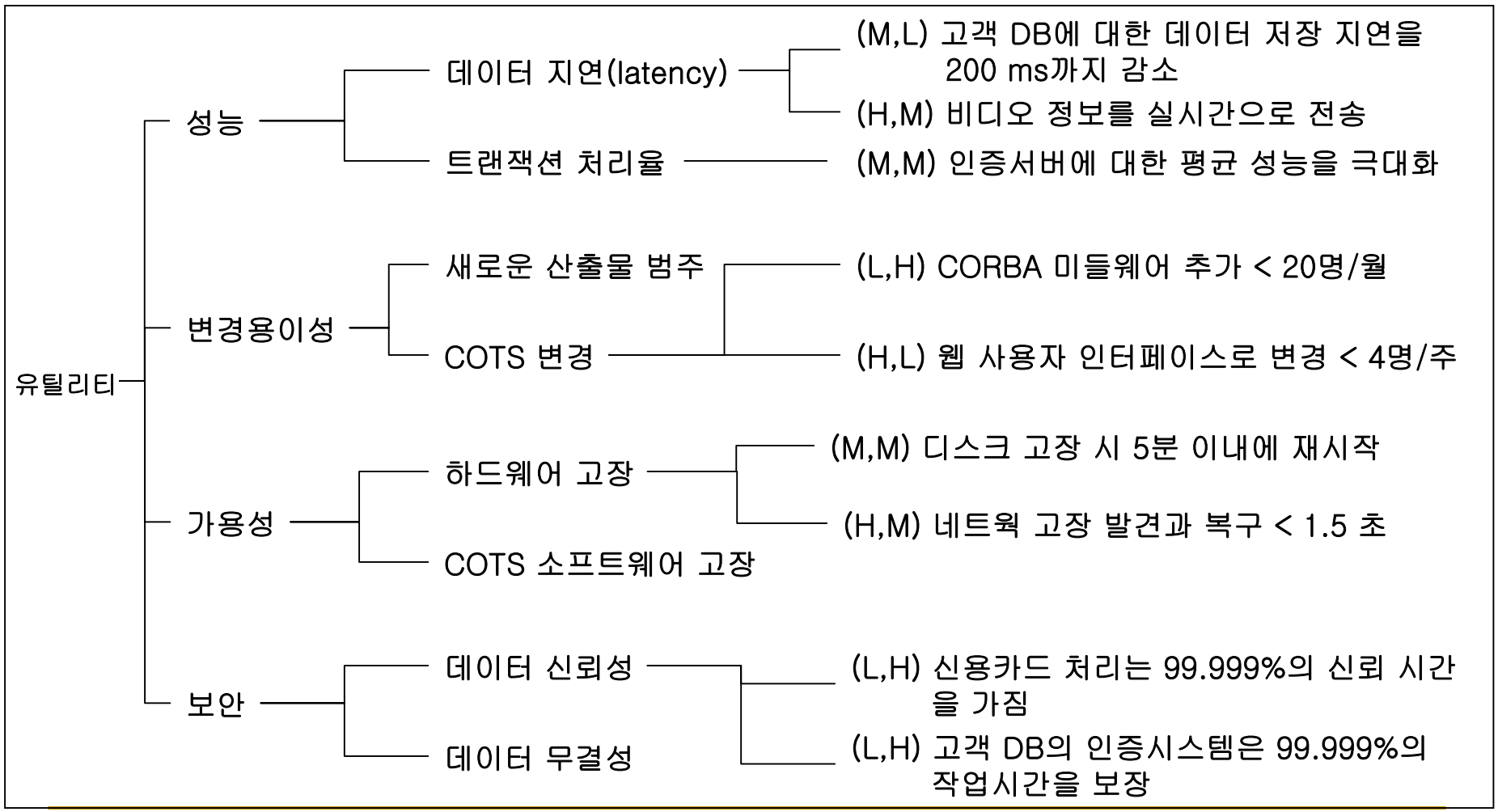
ATAM의 평가 단계(steps)



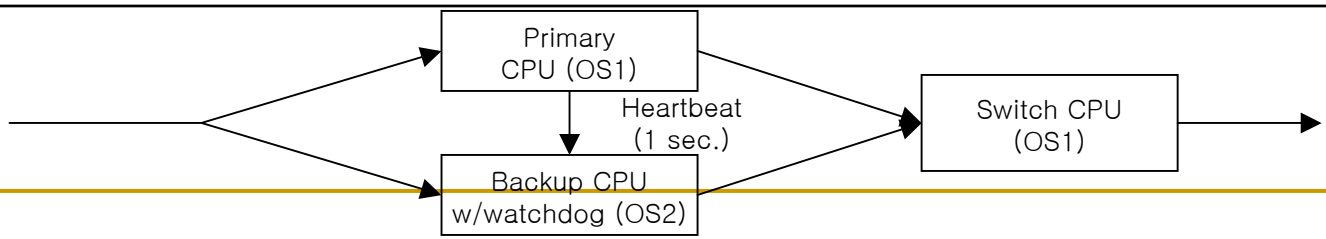
- 아키텍처 접근법 식별



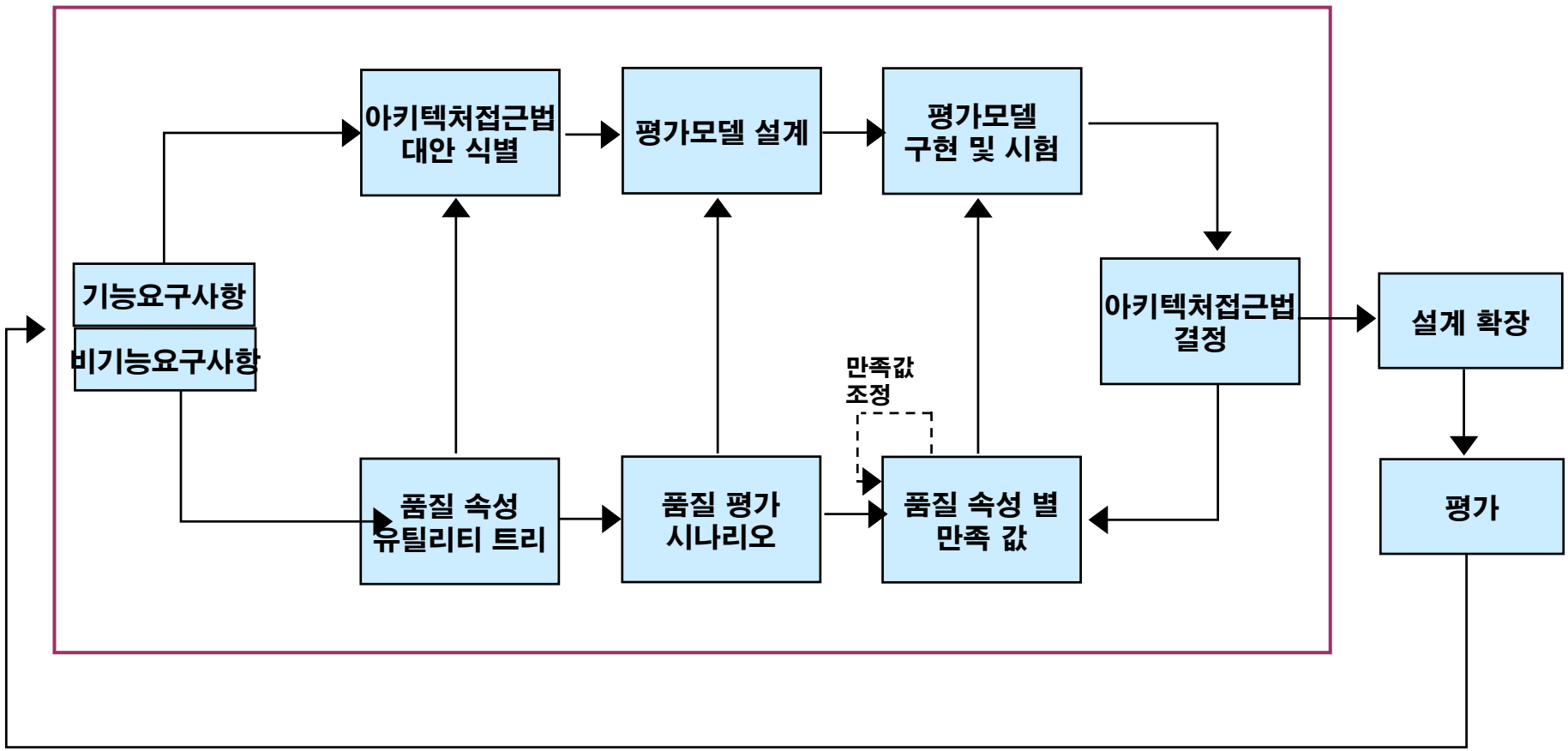
품질속성 유틸리티 트리(예)



아키텍처 접근법 분석서 양식(예)

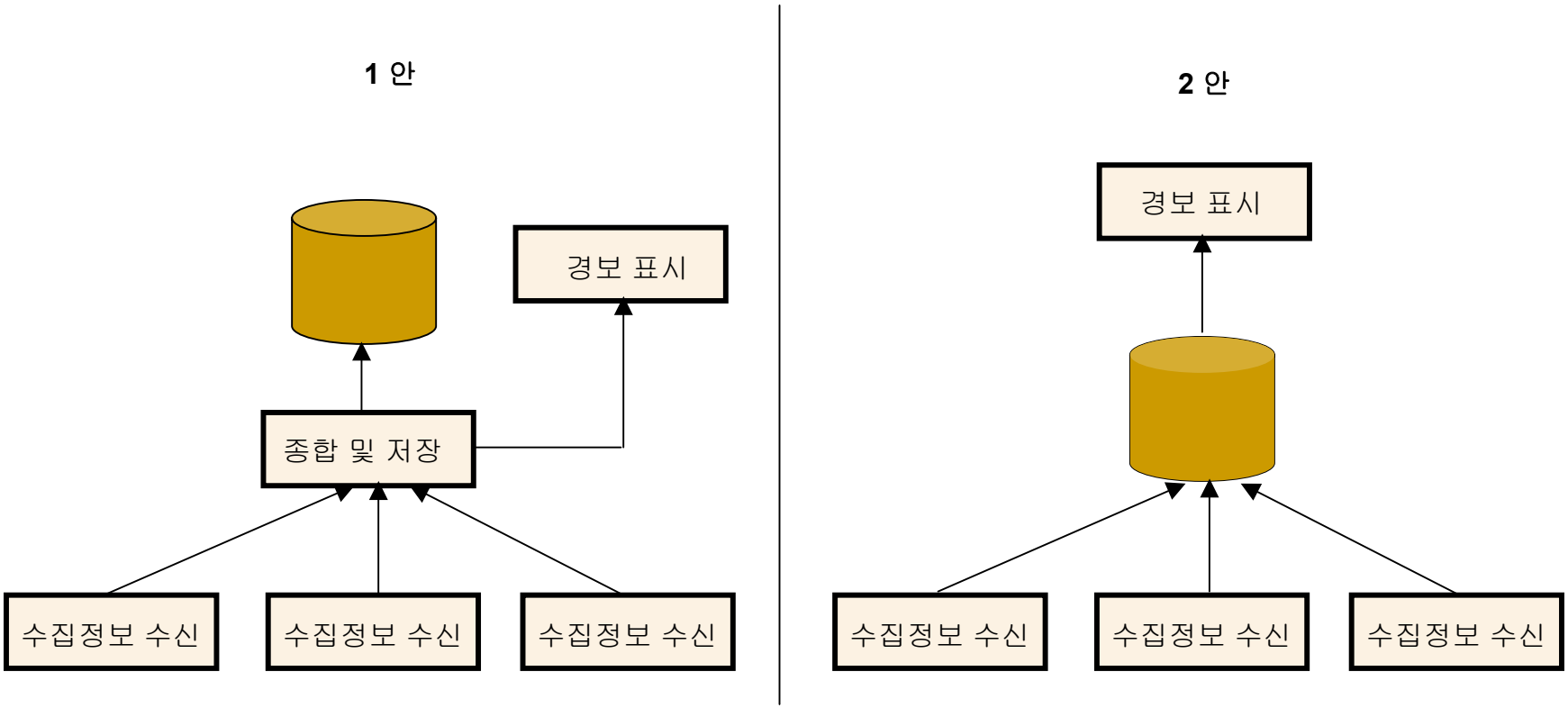
아키텍처 접근법 분석				
시나리오 #: A12	**시나리오:** 주 CPU의 하트웨어 고장의 발견과 복구			
속성	가용성			
환경	정상적인 운영			
자극(Stimulus)	CPU 고장 등 시스템 고장			
반응(Response)	0.999999 스위치 가용성			
아키텍처 결정	**민감점**	**절충점**	**위험요소**	**비위험요소**
Backup CPUs	S1		R1	
Backup data channel	S2	T1	R2	
Watchdog	S3			N1
Heartbeat	S4			N2
Failover routing	S5			N3
추론(Reasoning)	▪ 다른 하드웨어와 운영체제에서 동일한 작업환경을 유지하는 것은 위험요소가 될 수 있음(Risk R1 참조) ▪ 장애시 최악의 시스템간 교체(rollover)는 4초이내에 달성된다. ▪ Heartbeat과 Watchdog의 비율에 기반하여 2초 이내에 고장의 감지를 보장 ▪ Watchdog은 단순하고 신뢰성 있음 ▪ 가용성 요구사항은 백업 데이터 채널의 부족으로 인해 위험요소가 될 수 있다. (Risk R2 참조)			
아키텍처 다이어그램				

아키텍처 설계 및 평가 통합 모델(1)

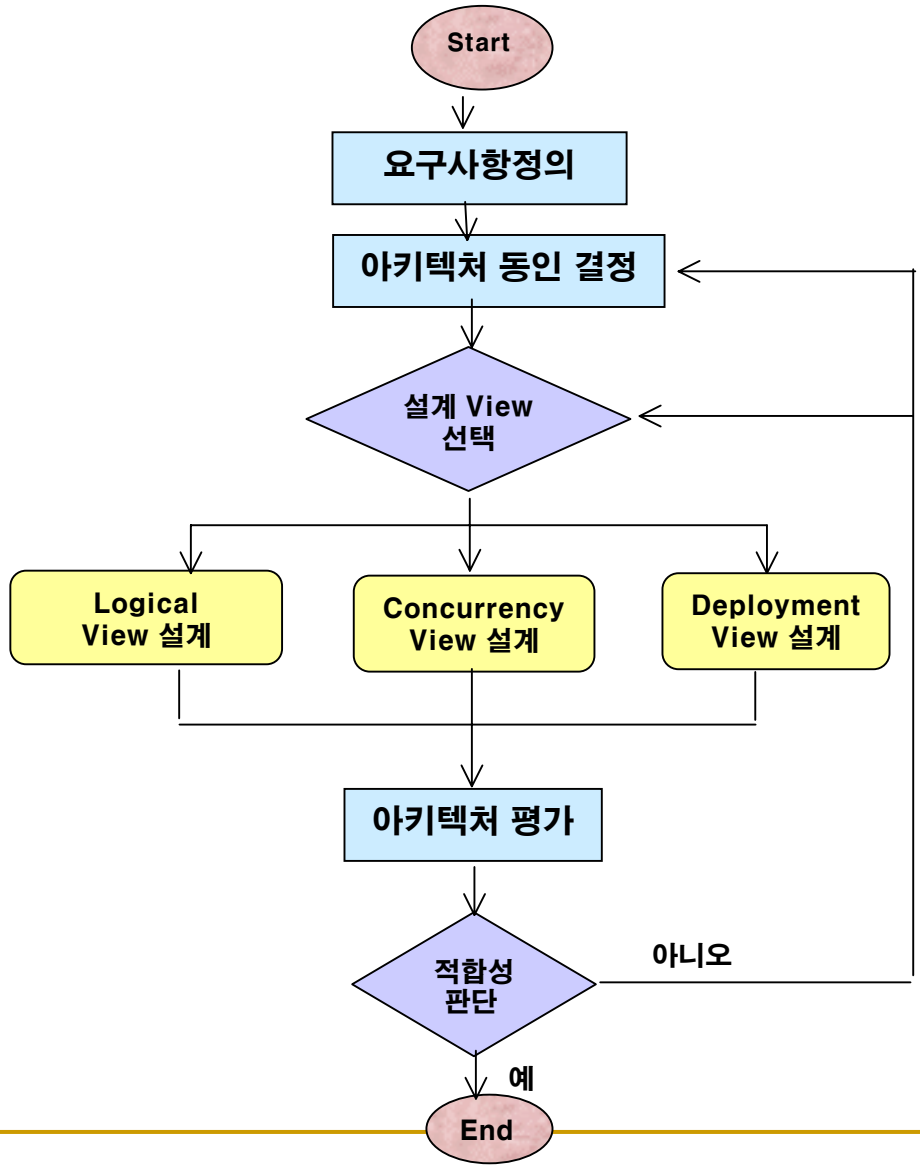


아키텍처 설계 및 평가 통합 모델

- 대안의 평가 및 접근법 선택



아키텍처 설계 및 평가 통합 모델



정 리

- 소프트웨어 요구의 변화
  - 규모의 공룡화
  - 소프트웨어 품질의 중요성 제고
- 방법론의 활용 필요
  - 절차와 방법, 도구의 이용 및 확산 필요
  - 특히, 정보의 공유와 동의하는 문화의 도입 필요
- 세미나에서는
  - 소프트웨어, 설계, 아키텍처, 품질의 정의 소개
  - 설계와 평가 방법론 소개