

# Reliable Server Architecture

# 요소 (Elements)

- Node
  - Is a process with networking
- Actor
  - Is a thread (or running within a thread)
- Component
  - Is a functionality of an application

# Graph

- Dependency Graph
  - Actor A -> Actor B -> Node C
  - 만약 Actor B가 동작하지 않으면 (응답이 없으면) Actor A의 기능이 동작하지 않는다
    - Failure of Actor A caused by actor dependency
  - Component 수준에서 정의할 수 있다.

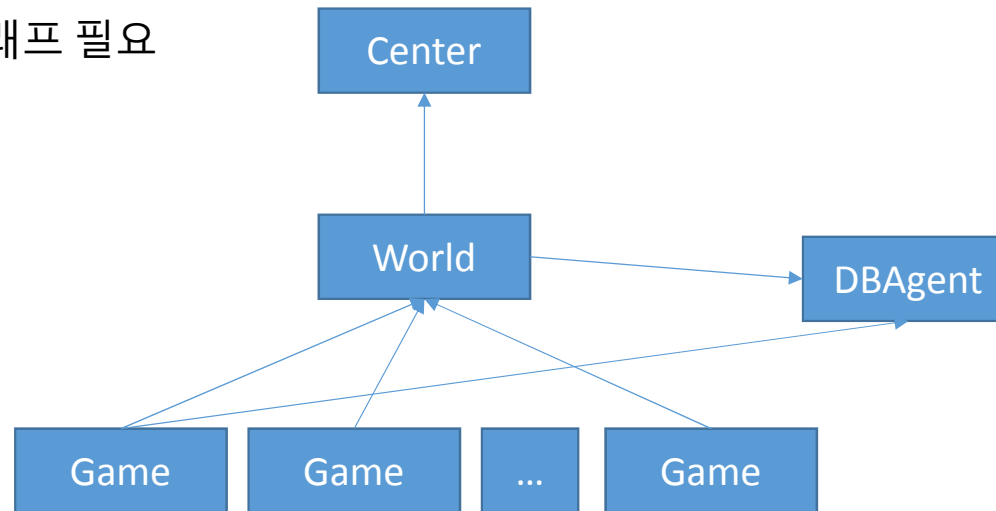
# 장애의 원인(Cause of Failures)

- Local
  - Node
    - System crash
    - Network failure
    - Too high utilization (Memory / CPU)
    - Called “System Environment Failure”
  - Actor
    - Application error
      - Exception (System, Application)
    - Logic error
    - Called “Component Failure”
- Dependency
  - Local failure of dependent Component / Actor / Node failure

# Dependency Graph

대략적인 구분

- 정확하게는 기능 (Actor) 단위 그래프 필요



# Component Dependency Graph

- 개념적인 구분
- 클래스나 함수 단위 구분이 아님
- 장애 전파와 복구를 위해 살핌
- Component의 구현 / 배치 모델과 장애 복구의 구조를 찾는 것
  - 최선의 배치와 최소의 장애와 최단의 복구 시간을 갖는 것

# Reliability (신뢰성)

- 사용자가 경험하는 서비스 불안정의 정도
  - 접속 오류
  - 단선
  - 긴 응답 시간
  - 클라이언트 Stability도 중요
    - 전체 서비스 품질에 포함

# Fault Tolerance 의 내용

## Fault Tolerance


- Detection
- Recovery
  - mask the error OR
  - fail predictably
- Designer
  - possible failure types?
  - recovery action (for the possible failure types)
- A fault classification:
  - transient (disappear)
  - intermittent (disappear and reappear)
  - permanent



# Failure Type (장애 타입)

- 탐지 / 복구
  - 발견해서 복구함
- 방지 / 감소
  - 이 쪽이 더 중요
  - 발생하지 않게 하거나 줄임
- 사실?
  - Dependency Graph의 구조에 따라 방지 / 감소 / 탐지 / 복구를 개선할 수 있다.

# 장애 타잎

 Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts
Omission failure <i>Receive omission</i> <i>Send omission</i>	A server fails to respond to incoming requests A server fails to receive incoming messages A server fails to send messages
Timing failure	A server's response lies outside the specified time interval
Response failure <i>Value failure</i> <i>State transition failure</i>	The server's response is incorrect The value of the response is wrong The server deviates from the correct flow of control
Arbitrary failure	A server may produce arbitrary responses at arbitrary times

# 장애 대상

- Crash
  - Node / Actor / Component Crash
    - Component Crash에 주로 관심
    - Node / Actor 크래시는 배치에 따른 Component Crash로 귀결됨
  - Network failure
    - 많은 구성 요소
    - 크래시와 구분이 거의 안 됨
- Timing Failure
  - 느린 응답

# 서비스 영향

- 사용자 관점
  - 사용 불가능한 Component 들
  - 다시 사용 가능하게 되는 데까지 걸리는 시간 / 노력
  - 불만족도
- Metrics
  - Availability / Reliability / Safety / Maintainability
  - MTBF, MTTR
  - 시스템 제공자의 관점

# 방향

- Component를 세분화하고
- Component의 의존 관계를 줄여
- Reliability를 올린다

# Redundancy (중복)

- 동일 Component를 여러 개 둠
- Replication
  - 데이터 복제
  - 동일 Component를 여러 개 만드는 방법
  - 지연을 고려해야 함

# 의존 구조 (Dependency Topology)

- Flat (평면)
  - 대등한 Component들의 평면 배치
  - 메시 구조와 유사
- Hierarchical (계층)
  - 트리 형태
  - 간결한 구조이나 단일 지점 장애 발생 가능성이 높아짐

# 게임 서버의 주요 Component들

- 사용자 인증 / 권한 검사 (User Authentication / Authorization)
- 사용자 위치 (User Location)
- 길드 / 파티 관리 (Global Container)
- 스케줄링 (Global Scheduler)
  - 미션맵
  - 이벤트
- 인스턴스 (Instance)
  - 진입 / 진출
  - 생성 / 소멸
  - 게임 플레이
- 데이터베이스 (Persistence)
  - 조회 / 저장



# 게임 플레이

- 사용자가 가장 오래 머무르는 곳
- 게임의 가치가 발현되는 곳
- 의존하는 컴포넌트들
  - 인스턴스 (관리)
  - 데이터베이스 (Persistence 제공 컴포넌트들)
  - 클라이언트
  - Global Container
  - Global Scheduler
- 방향
  - Persistence 컴포넌트를 로컬화
    - 분산되고 장애 복구가 가능한 데이터베이스 사용
  - 클라이언트 장애 시 동일 상태 자동 복구
    - 긴 진입 과정을 자동화
    - 게임의 이전 상태 복구
      - 네트워크 장애 또는 클라이언트 크래시 일 경우
  - Keep playing!

# Flat 하게 만들기

- 복제를 통한 분산
  - 사용자 인증 / 권한 부여
  - 사용자 위치
  - 인스턴스 정보
  - 인스턴스 진입 / 진출

# 다중화된 DB

- NuoDB / VoltDB
- 분산 DB들
- NoSQLs

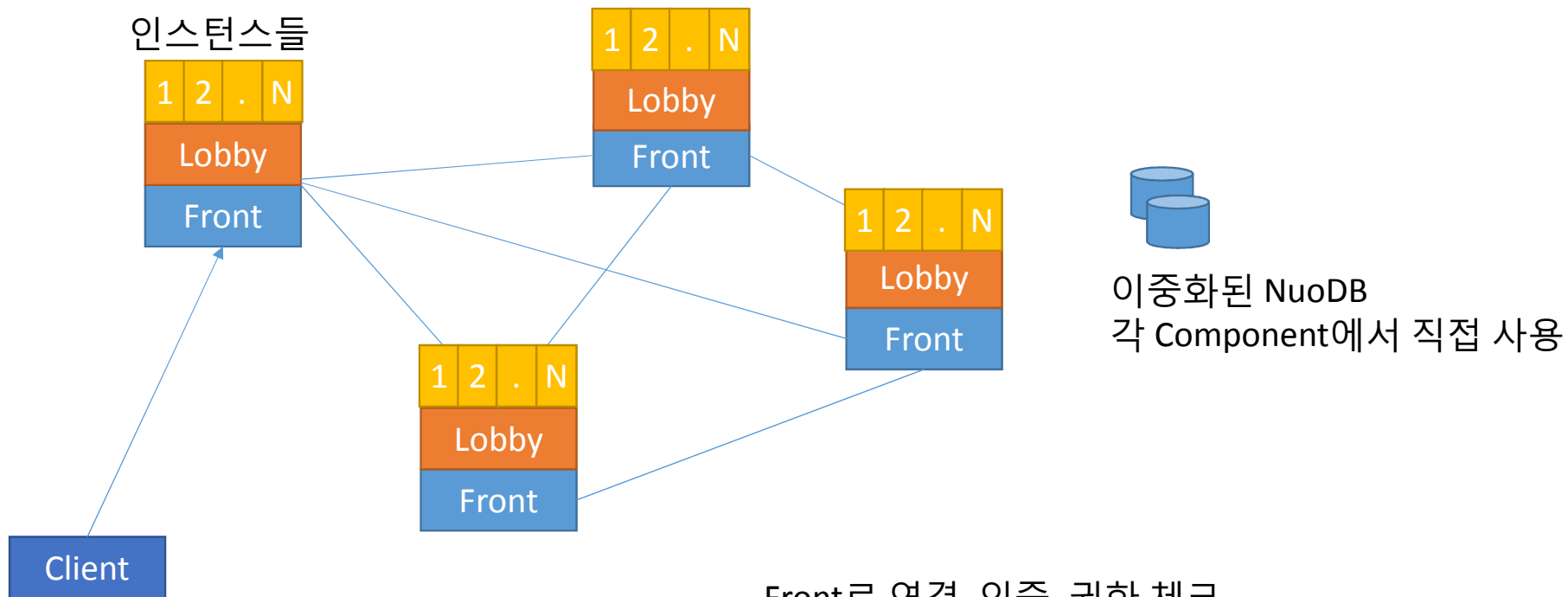


- 쿼리 로컬에서 처리
  - DBAgent와 같은 단일지점 장애가 있을 수 있는 컴포넌트를 제거

# Global Container / Scheduler / \*

- 장애 시에도 동작하도록 코드 구현
- 응답이 없으면 사용 안 해도 무관하도록
  - “응답이 없습니다” 메시지로 충분하도록
  - 나중에 시도하면 되도록 기능 설계

# 제안 구성



Front로 연결. 인증. 권한 체크.  
Front의 사용자 정보 Replication (다른 노드 동기화)  
Lobby에서 인스턴스 관리  
Lobby 인스턴스 정보 / 상태 / 사용자 정보 동기화

# Replication

- Lobby만 살펴봄
- Instance 생성
  - 전체 Lobby의 실행 상태를 보고 타겟 Lobby를 선정하고 요청
  - 응답이 오면 생성된 상태
  - 진입 진행
- Instance 진입
  - 예약 후 진입
  - 예약에서 실패하면 실패
  - 진입 후 클라이언트에서 연결 변경 (다른 노드일 경우)
  - 클라이언트에서 연결 변경 후 체크인 (진입 확인)
- Instance 진출
  - 바로 나가고 전체에 동기화
- Instance 소멸
  - 소멸 중 상태로 변경
  - 소멸 중 상태 동기화
  - 일정 시간 후 소멸
  - 소멸 중에도 진입이 될 수 있음

# 장애 복구

- Front
  - 시작 시 하나의 다른 Front에 사용자 정보 요청
  - 전체 Front에 갱신된 age보다 최신의 정보 요청
- Lobby
  - Front와 동일. Lobby에 대해 진행
- Instance
  - 재진입으로 복구
- 장애 복구는 새로운 노드 추가와 동일

# 장애 모니터링과 Replication

- 주기적인 heartbeat
- Available / Unstable / Unavailable / Crash
- Unavailable 상태 진입 시 해당 Lobby의 인스턴스 정보 제거
- Unstable에서 Available로 가면 마지막 시간에서 추가 동기화



# 확장성 고려

- Node 가 많아지면 동기화 트래픽이 많아짐
- 특수한 Front의 추가로 가능
  - Bridge와 같은 기능
  - 두 개 그룹 간의 중계 역할만 수행

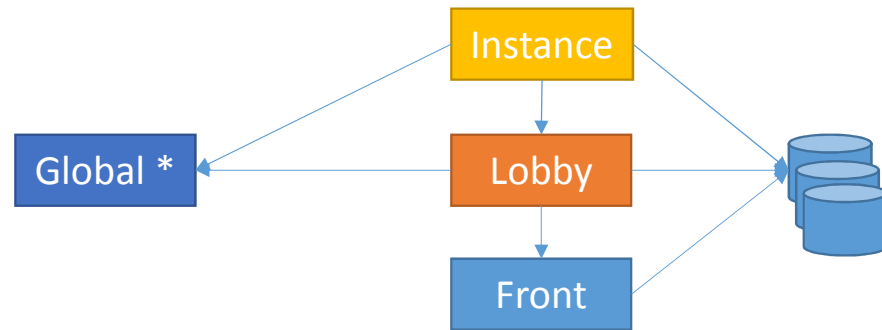
# Global 컴포넌트의 이동

- Global 컴포넌트를 갖는 노드의 장애시
  - 다른 노드 선정
  - 실행
  - 복구 (동기화 통해)
  - 전체 통지

# 장애 지점

- DB 노드 전부 실패
- Global 컴포넌트
  - 일부 기능 장애
- 이외에 게임 단위별 장애
  - 빠른 복원으로 해결

# 의존 그래프



단일 장애 지점을 제거

# 정리

- Replication으로 중앙 서비스 제거
  - Center
  - 대규모 Lobby
- DB
  - 분산 DB 사용 (NuoDB가 적절한 비용의 대안이 될 수 있음)
  - 로컬에서 쿼리 (DBAgent와 같은 Cache / Agent 서버를 두지 않음)
- 재진입
  - 빠른 재진입 흐름을 미리 고려 및 구현
- 이를 통해 Flat (평면) 구조의 신뢰성 있는 서비스 구조 마련

# 자료

- <https://www.cs.helsinki.fi/u/jakangas/Teaching/DistSys/DistSys-08f-5.pdf>
  - Fault Tolerance (장애 탐지와 복구) 일반에 대한 개요
- <http://www.distributed-systems.net/courses/ds/ds-slides/chp08.pdf>
  - 위 소개의 자세한 버전