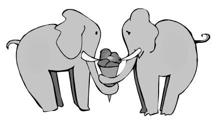
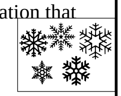
§4 Managing Dynamic Shared State

1. Consistency-throughput trade-off
2. Centralized information repositories
3. Frequent state regeneration
4. Dead reckoning



Dynamic Shared State

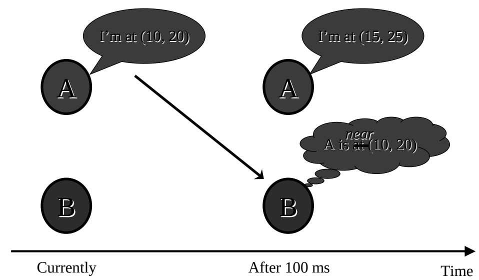


- Dynamic shared state constitutes the changing information that multiple hosts must maintain
  - participants, their locations and behaviours
  - environment itself, all objects, weather, natural laws,...
- In a highly dynamic environment, almost all information about the world may change ⇒ needs to be shared
- Accuracy is fundamental to creating realistic environments
- Makes an environment available to multiple users
  - without dynamic shared state, each user works independently (and alone)

Maintaining Dynamic Shared State

- Building an NVE = the problem of managing the dynamic shared state
- Trade-offs between the available resources and the desired realism of the VE experience
- Three basic approaches to maintain dynamic shared state:
  - shared repositories
  - frequent broadcast
  - state prediction

Example of Dynamic Shared State



§4.1 Consistency-Throughput Trade-off

- The fundamental rule about NVE shared state:

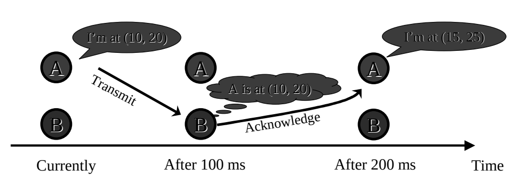
It is impossible to allow dynamic shared state to change frequently and guarantee that all hosts simultaneously access identical versions of that state.

⇒The NVE can either be
- a *dynamic world* in which information changes frequently, or
- a *consistent world* in which all hosts maintain identical information

but it cannot support both.

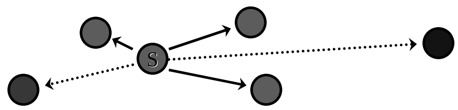
Reasoning Behind the Trade-off 1 (2)

- To guarantee *absolute consistency* among the hosts, the data source must wait until everybody has received the information before it may proceed
  - delay from original message transmission, acknowledgements, possible retransmissions
- The source can generate updates only at a limited rate
- Time for the communication protocol to reliably disseminate the state updates to the remote hosts



Reasoning Behind the Trade-off 2 (2)

- There is a delay before the state change is received by other hosts
- If the shared state is updated often, it might be updated while the previous update messages are still on the way
- Whilst some hosts see new values, others may still see older ones
- Because of the inherent transmission delay, one cannot update the shared state frequently and still ensure that all remote hosts have already received all previous state updates



Design Implications

- Available network bandwidth must be allocated between
  - messages for updating the dynamic shared state and
  - messages for maintaining a consistent view of that dynamic shared state

  among participants in the NVE.

Absolute consistency ←→ High update rate

The trade-off spectrum

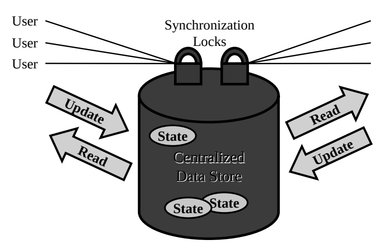
Trade-off Spectrum

Absolute consistency ←→ High update rate

Centralized information repositories — Frequent state regeneration — Dead reckoning

§4.2 Centralized Information Repositories

- Ensure that all hosts have identical information



File Repository

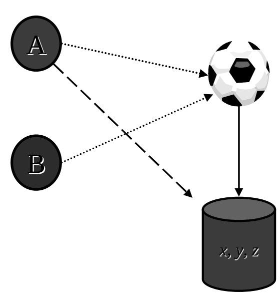
- A directory contains files that hold the shared state
  - a file for each user
- Read the shared states to generate view:

```
for all files in the directory
    open the file in read-only mode
    read the user state information from the file
    close the file
draw the scene from the local user's point of view
```

- Update the shared state:

```
open the user file in write-only mode
write the new state information to the file
close the file
```

Problem: Who's Got the Ball Now?



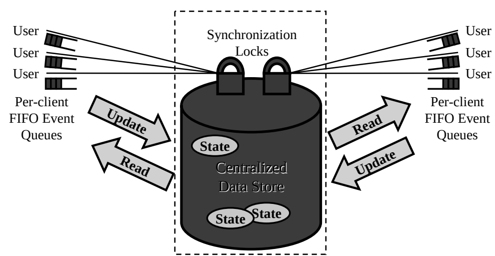
Repository in Server Memory

- Server process simulates a distributed file system
- NVE client can
  - query the server for any of the shared state
  - initiate a write to any of the shared state
- Each host maintains a TCP/IP connection to the server process
- Clearly faster than a file repository
  - the current state is in memory
  - the client does not perform explicit open and close operations
  - the client does not need to request locks when writing data
  - the server may support batched operations

Repository in Server Memory (cont'd)

- New problems
  - if the server crashes, the shared state is lost
  - resources to maintain persistent TCP/IP connections
- Benefits of a server repository
  - simplicity
  - reasonable performance

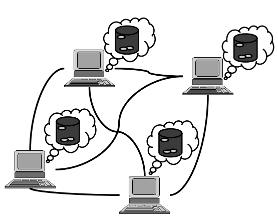
'Eventual' Consistency



Pull and Push

- The clients 'pull' information when they need it
  - make a request whenever data access is needed
  - problem: unnecessary delays, if the state data has not changed
- The server can 'push' the information to the clients whenever the state is updated
  - clients can maintain a local cache
  - problem: excessive traffic, if the clients are interested only a small subset of the overall data

Virtual Repositories



- Distributed consistency protocol
  - hosts exchange messages directly
  - ensure that all hosts receive updates
  - determine a common global ordering for updates
- No central host
- Every host has an identical view
- All state information is accessed from local caches, which behave like a central repository

Virtual Repositories (cont'd)

- Advantages of distribution
  - eliminates the performance bottleneck
  - eliminates the bandwidth bottleneck
  - permits better fault tolerance
- A client do not need to monitor all shared state with absolute consistency
  - area-of-interest management
  - varying consistency requirements

Centralized Repositories: Advantages and Drawbacks

- Provide an easy programming model
- Generally guarantee information consistency
- No notion of data 'ownership'
  - host is able to update any piece of shared state
- Data access and update have unpredictable response times
- Communications overhead
  - acknowledgements, retransmissions