Fault-Tolerance Techniques for Mobile Agent Systems

Introduction

- Mobile agent has been proposed in different application domains:
  - E-commerce
  - Mobile Computing
- It is important to have:
  - Fault-detection
  - Recovery

Mobile Agent Execution Model

- A mobile agent executes on a sequence of machines.
- A **place** P_i provides a logical execution environment for the agent.
- Executing an agent at a place is called a **stage** S_i of the agent execution.

Mobile Agent Failure Model

- We can classify failures into 3 classes:
  - Agent failure
  - Place failure
  - Machine failure
- We assume that *agent failure* and *place failure* will not happen.

Mobile Agent Failure Model

- When a machine failure happens, all agents executing will be terminates.
- When an agent wants to travel to a failed host, an exception will be raised.
- We assume that the agent will be terminated in this case.

Problems of failures

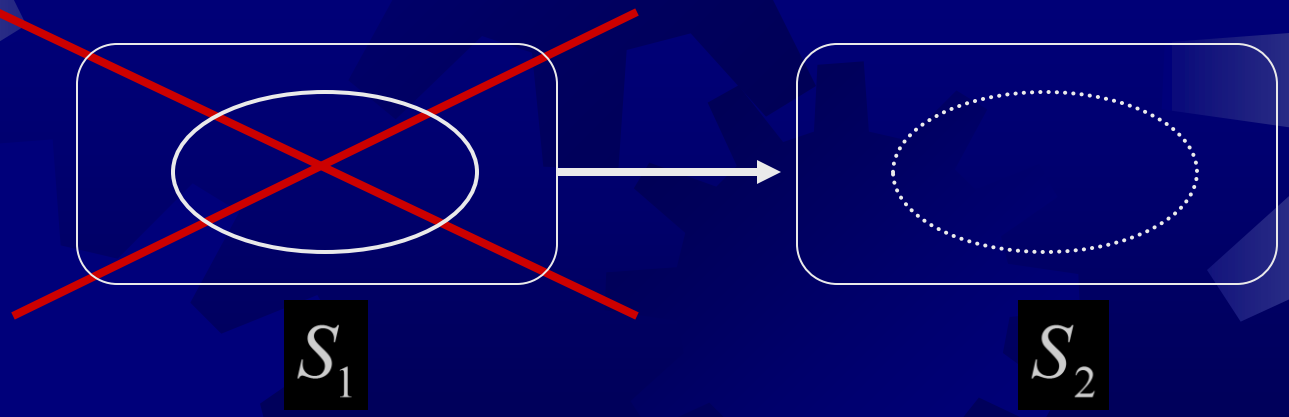
- Agent travels in the network.
- It is difficult to estimate the running time of an agent.
- Two problems :
  - Agent owner believes that the agent has been lost, but, in fact, it is not.
  - Agent owner waits for the agent to finish its execution, but the agent is actually terminated abnormally.

Concerns in Protocol Design

- Blocking-free
  - Assume that we have a prefect failure detection mechanism.
  - Suppose we have checkpoint every agent at every host.
  - If we have detected a host fails, we restart that failed host.

Concerns in Protocol Design

- This kind of recovery is prone to blocking.
- While the recovery is taking place, the execution is blocked until the recovery finishes.



Concerns in Protocol Design

- Exactly-once
  - Suppose an agent is trapped inside a very busy network.
  - If the owner launches another agent, we will have 2 instances in the network.
  - It will double the effect done by a single agent if the actions are not idempotent (non-intrusive).

Server Failure Detection

- A server fault-tolerance mechanism is two-folded.
  - Agent have to stop traveling to failed server.
  - There should be global daemons detecting failures.
  - Once failure is detected, recovery should take place.

Server Failure Detection

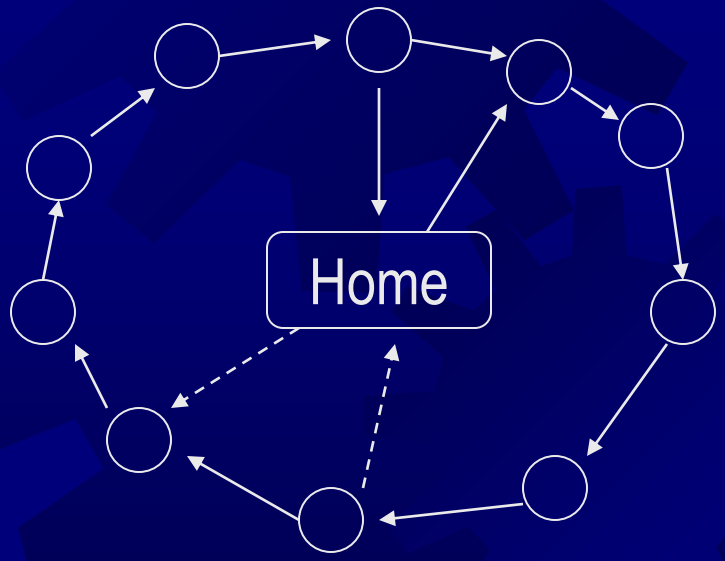
- A simple server fault-tolerance mechanism:
  - When an agent finishes computation, it checks if the next server is available or not.
  - If yes, it travels to that server.
  - If no, it waits at its resident server until the next host is available.

Server Failure Detection

- The way to detect server failure depends on what agent platform is using. E.g. RMI and RPC.
- We run a daemon global to all the servers. This daemon can detect and recover failed servers.
- However, the daemon is a **single point of failure**. We should introduce multiple instances of this monitor daemon to ease the problem.

Experiment

- We have set up an experiment on server failure detection.
- The network:



Experiment

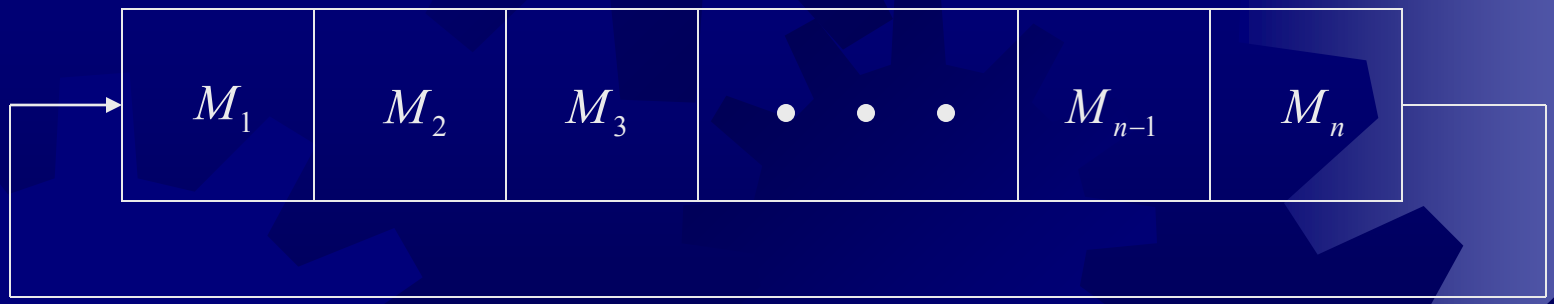
- To introduce failures to the server, we have a daemon running along with every server.
- The job of the daemon is to kill the servers randomly.
- We have set the probability to be 0.1 per 2 minutes.

Experiment

- We have 2 kinds of agents:
  - One can detect the availability of the next server.
  - Another one cannot.
- The former will wait for recovery.
- The latter will travel to failed servers and being terminated.

Experiment

- We have a global daemon.
- It detects and recovers server failures.
- It detects the servers failures by following a cyclic server list.

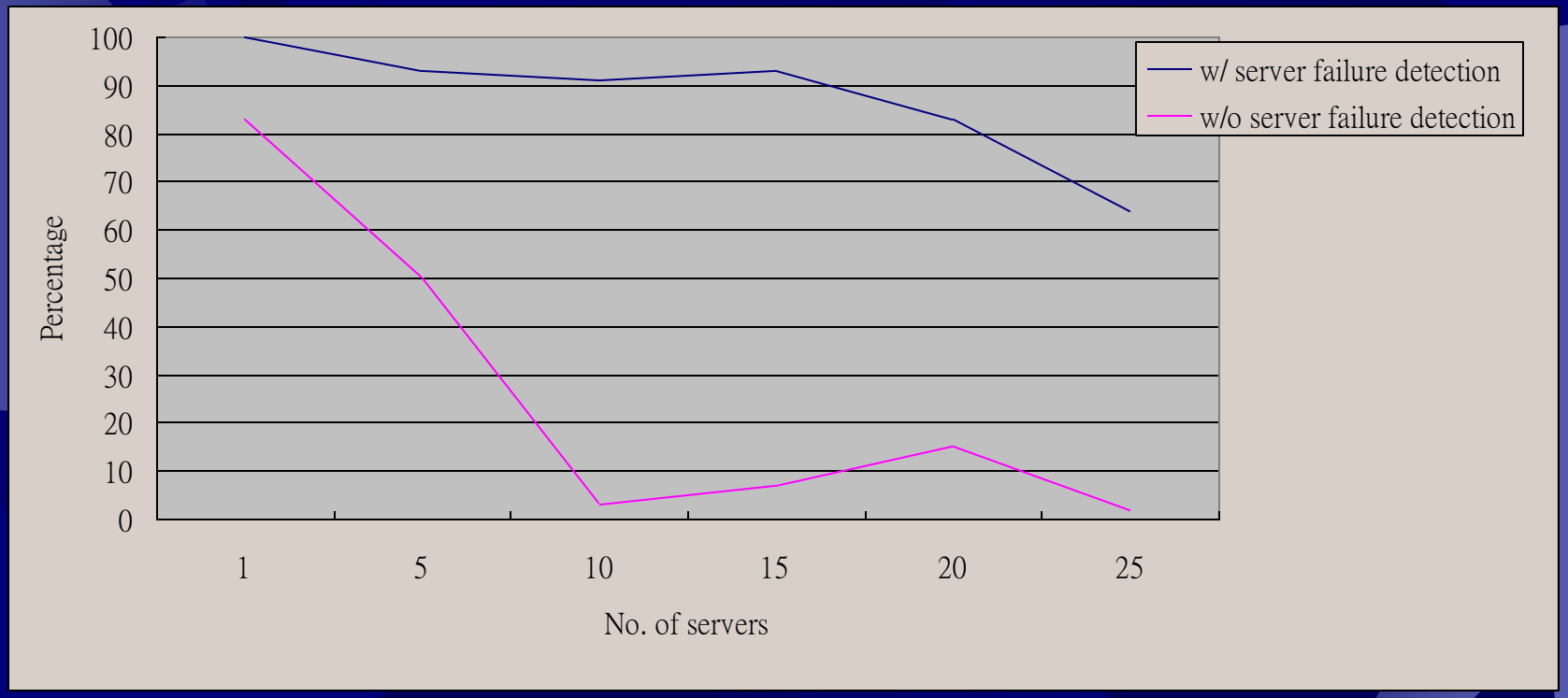


Experiment

- Estimation of the time between a server fails and it is recovered:
  - Let *p* be the probability that a server fails.
  - Let τ be the time needed to perfore the recovery process.
  - Let *n* be the number of servers.
  - The worst time *T* = $np\tau$

Experiment

- Result



Experiment

- Agents are still losing because the resident servers of the agents die while the agents are waiting.
- The time that the agent is waiting is linearly proportional to the number of servers.
- Therefore, the curve is dropping more or less in a linear manner.

Agent Failure Detection

- Pull approach
  - Pull information out of the agent periodically.
  - The owner queries the agent.
- Use **agent proxy**.
- Defect:
  - If agent is on the way traveling to a server, it cannot respond.

Agent Failure Detection

- Push approach
  - Agent pushes information to the owner.
  - Agent sends heartbeat messages to the owner periodically.
- Better than *pull approach*
  - No need to know where the agent is.

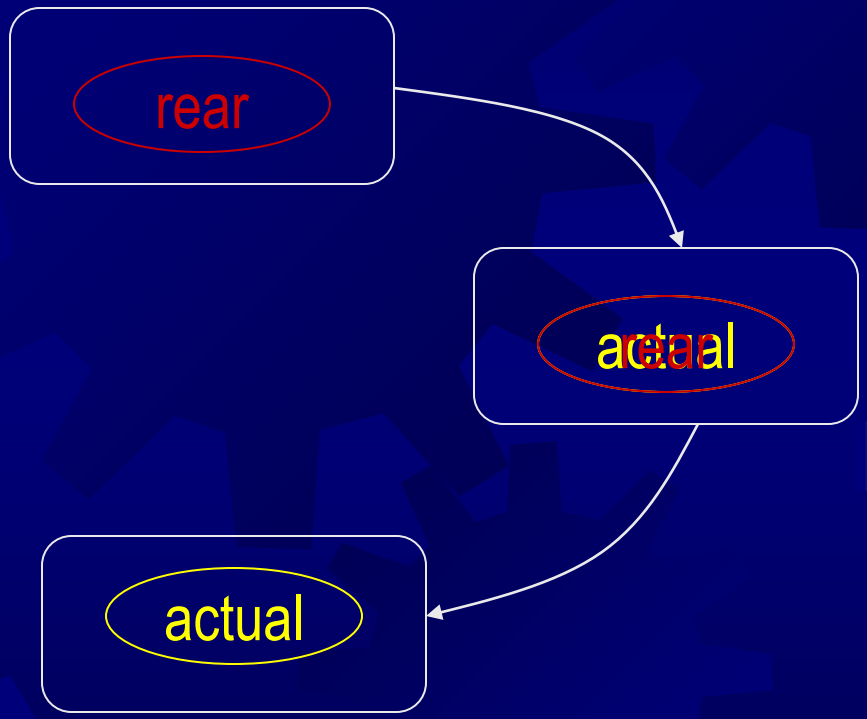
Agent Failure Detection

- Defects of the above 2 approaches
  - Centralized.
  - Depends on status of the network.
  - Produce a lot of traffic on the network.

Agent Failure Detection

- Cooperative Agent Approach
  - 2 agents are sent at one time.
  - One is called **actual agent**.
  - Another one is called **rear guard**.
  - Rear guard always lags the actual agent.

Agent Failure Detection



Agent Failure Detection

- How does it work:
  - When the actual agent arrives at a server, it sends message to the rear guard
    - I am in XXX
  - When the actual agent leaves a server, it sends message to the rear guard
    - I am leaving XXX
  - The rear guard will then travel to XXX.

Agent Failure Detection

- How to detect and recover the agent:
  - Assumption (1)
    - Checkpoint of actual agent.
    - It is for the use of recovering actual agent.
  - Assumption (2)
    - Agent will not be lost while traveling
    - This eliminates the possibility that rear guard cannot receive I am in XXX message.

Agent Failure Detection

- Case 1
  - Rear guard cannot receive the message I am leaving XXX within a timeout period.
  - This implies the agent crushes.
  - The rear guard can use the checkpointed actual agent to continue execution.

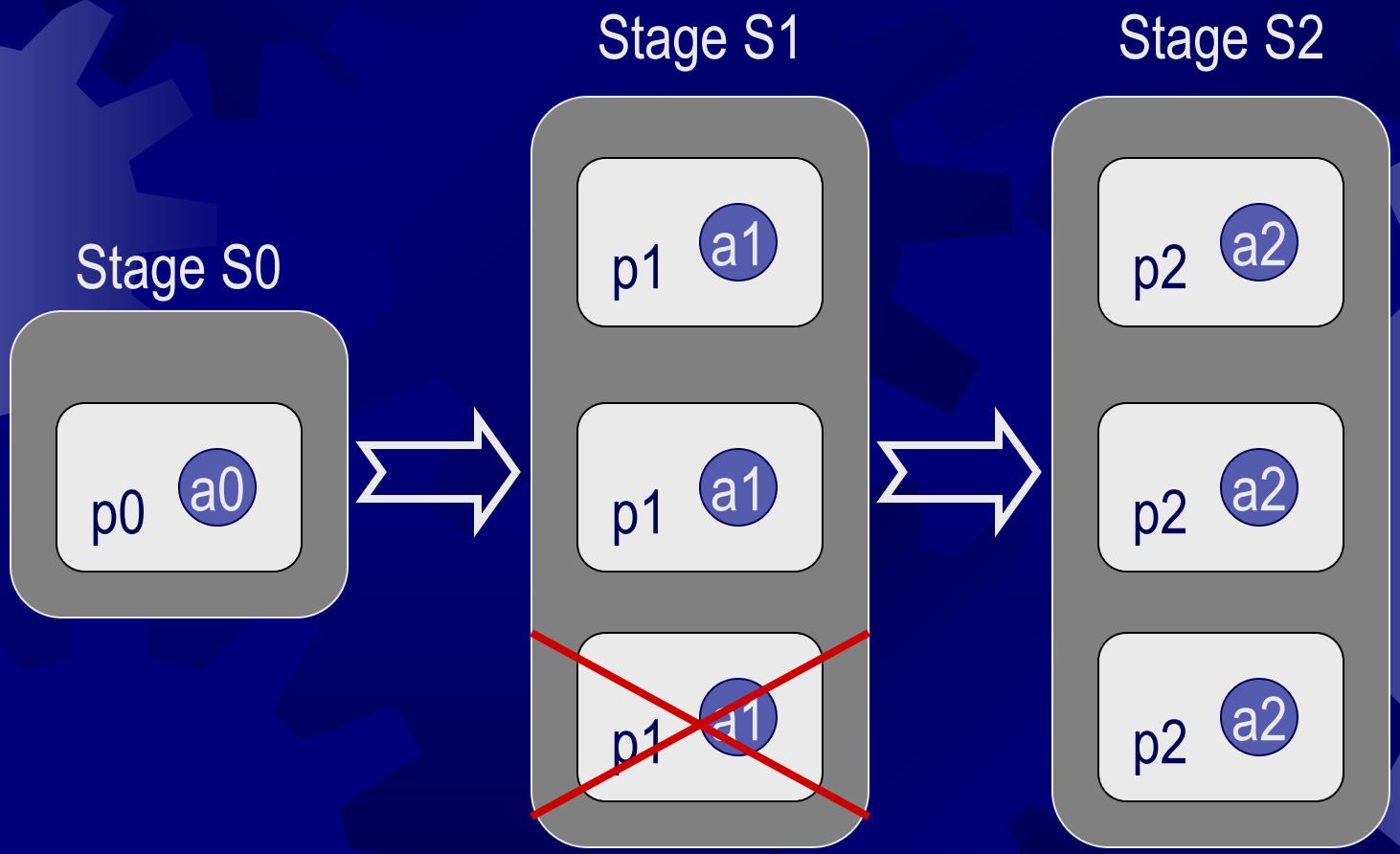
Agent Failure Detection

- Case 2
  - Actual agent cannot send I am in XXX to rear guard.
  - This implies the rear guard crushes.
  - Actual agent can transmit a rear guard to its previous server.

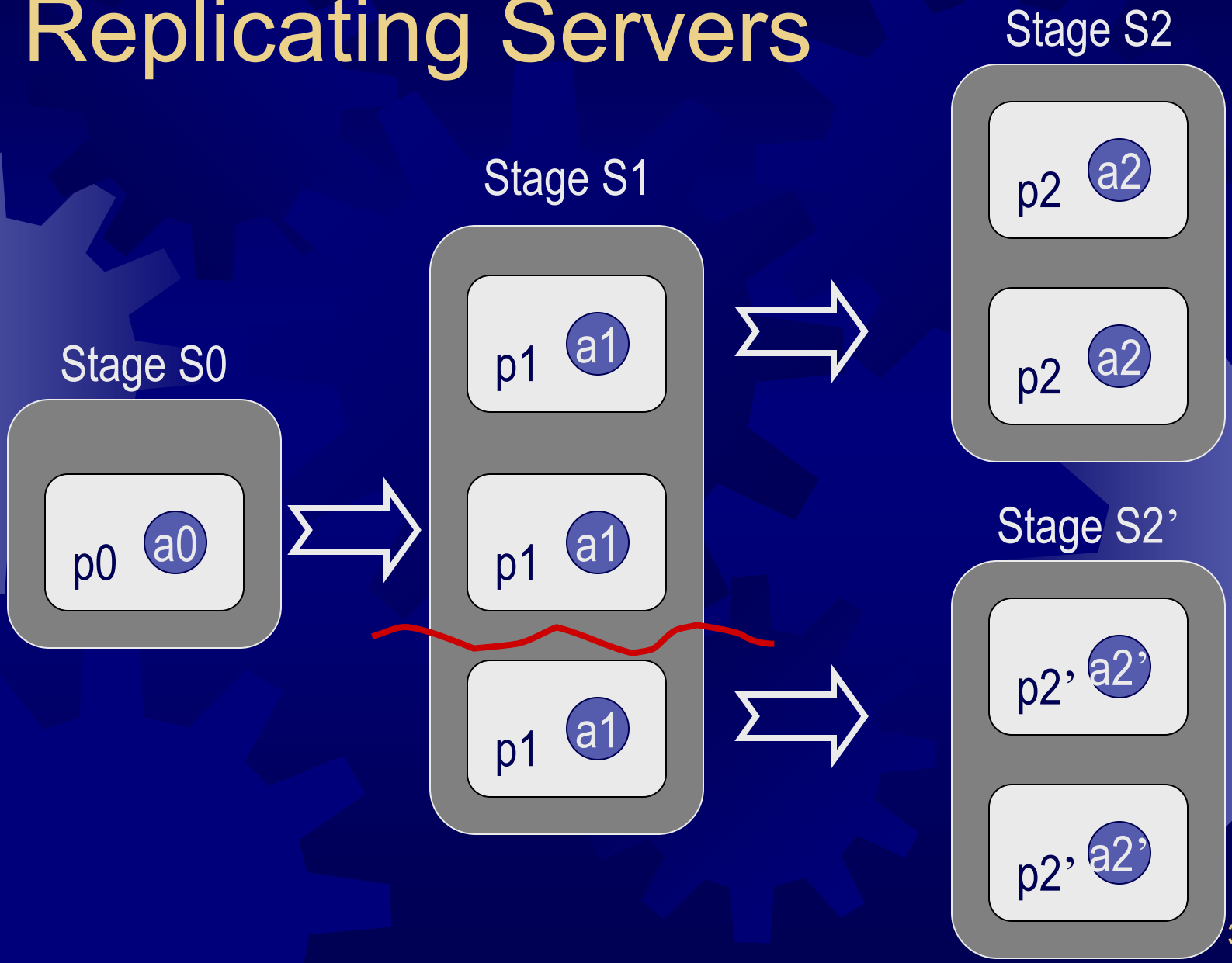
Agent Failure Detection

- Advantage
  - Decentralized
  - Probability of both rear guard and actual agent die are very small.
  - Small amount of messages comparing to periodic messages.

Replicating Servers



Replicating Servers



Checkpointing and Rollback

- Not all data can be checkpointing easily.
- Two types of agent data
  - Strongly reversible objects
  - Weakly reversible objects

Checkpointing and Rollback

- Strongly reversible objects
  - They can be compensated by means of an image of the objects.
  - E.g. Information retrieving agent.

Checkpointing and Rollback

- Weakly reversible objects
  - They may be different from the original data after compensations.
  - E.g. Electronic money.

Conclusion and Future Work

- We will continue to focus on agent failure detection.
- The above failure detection schemes do not satisfy the **exactly-once** and **blocking-free** requirements. Efforts are still needed.