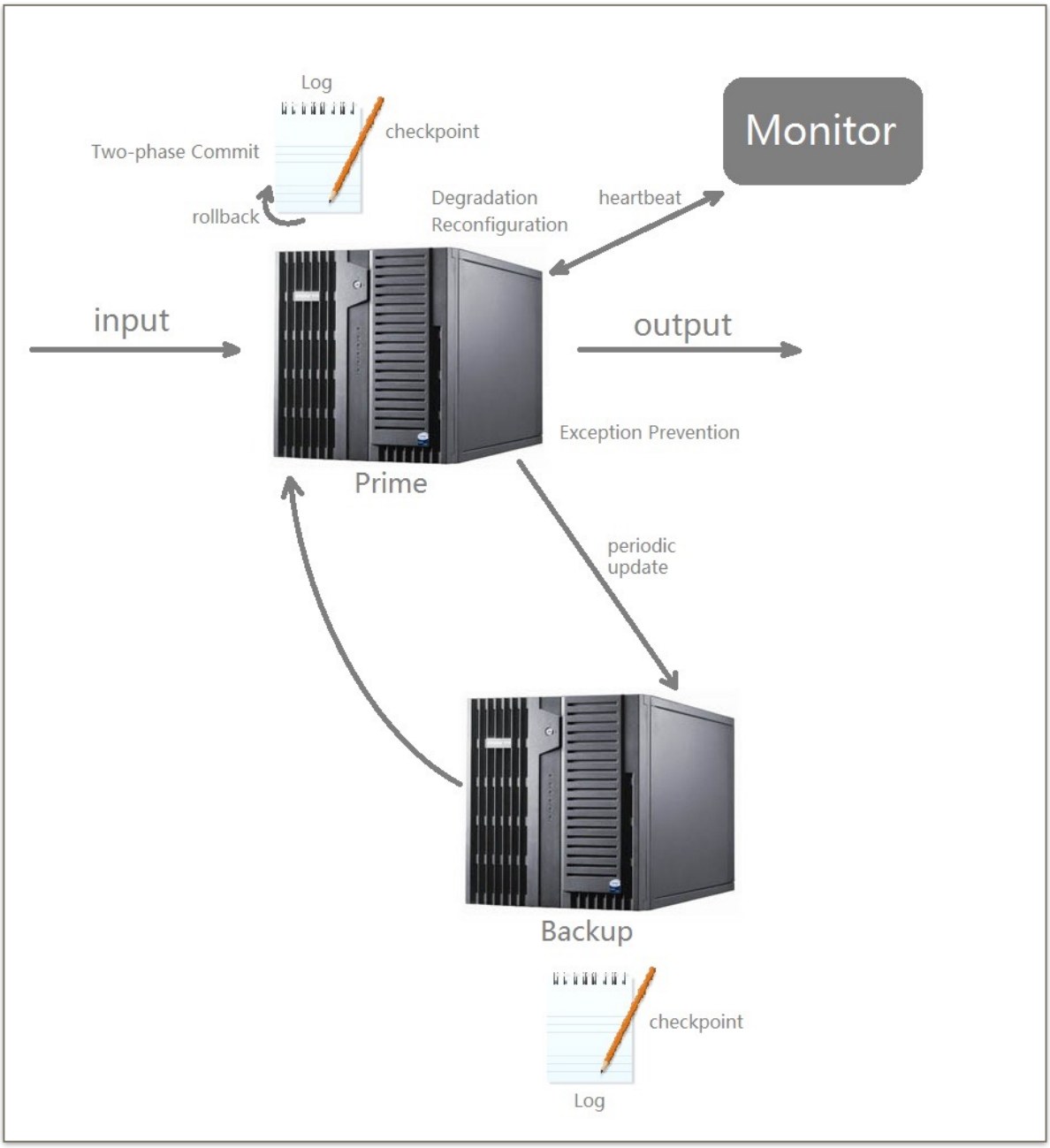
# assignment1 : Availability Design

3120102146 葛现隆

## Diagram




## Aim

working whole day &whole week, limited budget & high quality

## Tactic

Fault detection : Heartbeat
Recovery Preparation & repair : Passive red., Rollback, checkpoint, Degradation, Reconfiguration
Recovery Reintroduction : State resync.
Prevention : Transactions, Exception prevention

# Explain

Heartbeat

Since our server is running the whole day and whole week, we need a monitor to make sure it work all the time and work properly, and it can reduce overhead by piggybacking heartbeat messages on to other control messages being exchanged between the process being monitored and the distributed system controller.

Passive Redundancy

Since our budget is limited, it's more proper to apply passive redundancy than active.By periodic update, we can ensure the backup is sufficiently fresh as well as checkpoint and log. And upon failure, the standby components can take over from the primary.

Checkpoint & Rollback

Checkpoint alone with rollback is a very useful database recover tactic. Combining with passive redundancy, rollback can make sure the data is consistence and the not finished transaction is rolled back properly.

Degradation

In case of component failures, we can maintain the most critical system functions and drop less critical functions to recovery as well as maintain the server.

Reconfiguration

In our case, we need to server 1000 - 10000 customers per hour, that means we need to recovering from component failures by reassigning responsibilities to the resources left functioning, while maintaining as much functionality as possible.

State Resynchronisation

Since we use passive redundancy, we will periodically update the sate of the standby with the active, making sure consistence.

Transactions

Here we use two-phase commit to ensure the ACID properties of the asynchronous messages exchanged between distributed components. It's very nice to use transactions to fit rollback.

Exception prevention

Our server's function is very clear, and the possible exception is predictable,  so exception prevention can prevent system exception from occurring very efficiently.