## introduction

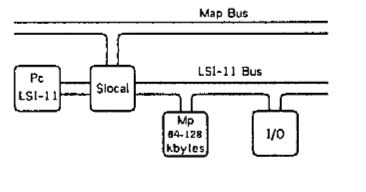
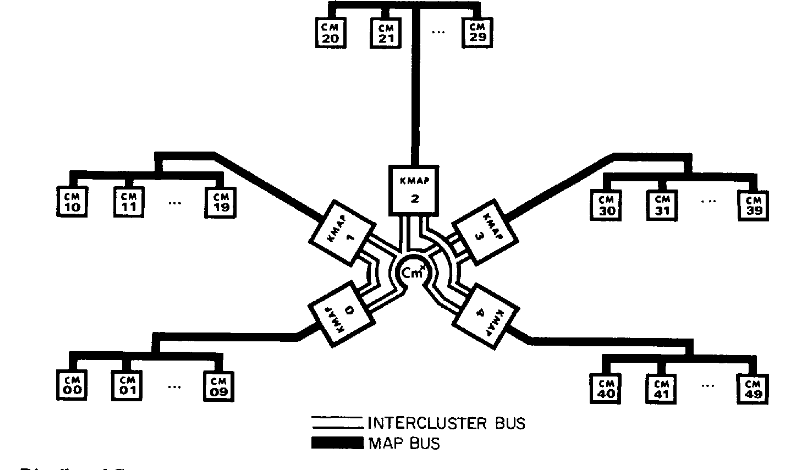
the mainpoint of this paper: problems of structures, rather than facilities

The distributed OS should have: modularity, robustness, performance.

1. physical arrangement: small local resources, large other resources
2. interprocessor communication facilities

system characteristics: 1) distributed control structure 2) parallelism of the task forces

## distributed structure



cm: computer modules, which are further divided into clusters

kmap: the communication controller

Slocal: switch, for addressing the cm.

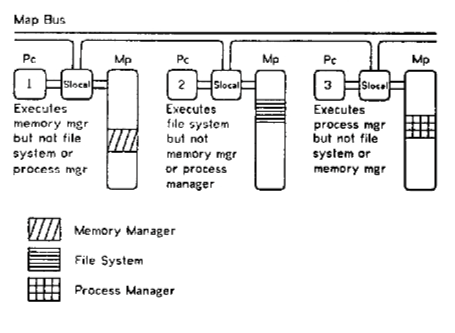
memory reference speed: local > nonlocal but within cluster > cross-cluster

higher \*local hit ratio\* -> higher efficiency

the failured approaches and their drawbacks:

1. one OS in one CM - the traditional structure makes the processing speed for many cms slow. And the reliability is suffered(not fault tolerent) e.g. Amazon reliable database system
2. OSs in all CMs - the local storage is not that large/ the hardware is too expensive for every machine to own. e.g. GPU
3. OS cache for some essential parts - hard to determine which parts of an OS is sufficient or not. e.g. special storage system S3, special database system RDS, special AI system GPU clusters.

the structure of medusa:



different parts of OS: utilities, are separated into different CMs. There is no certain guarantee for which CM contains which utility.

the invocation of functions across processors:

1. a process sends msg to a certain **pipe.**
2. destination process performs the operation and sends a return message to the return pipe indicated in the invocation message.

while executing, the control(usr program & other utilities) need to transfer to the destination processor.

the benefit for robustness and modularity: each processor as a utility, clear boundary; each module can be subtitied as it maintains its utility; the message function ensures that only the receiver will be influenced.

## the task force structure

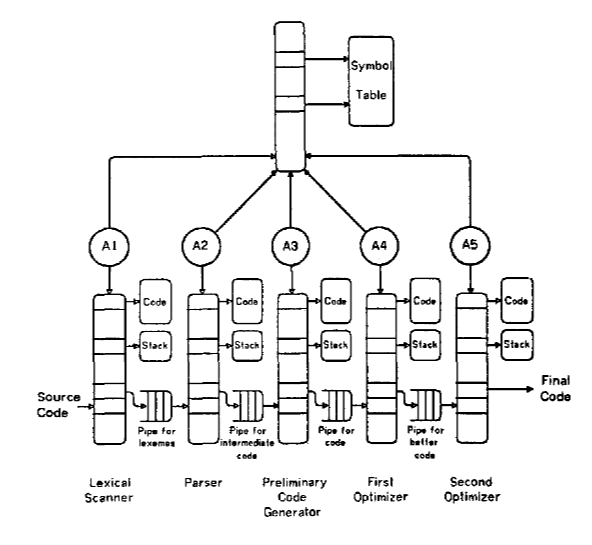
task force: a collection of concurrent activities that cooperate closely in the execution of a single logical task. With objects(shared descriptor list && private descriptor list) that may be manipulated by these activities.

\tip : activity - process, a utility can have multiple similar activities to ensure the reliability.

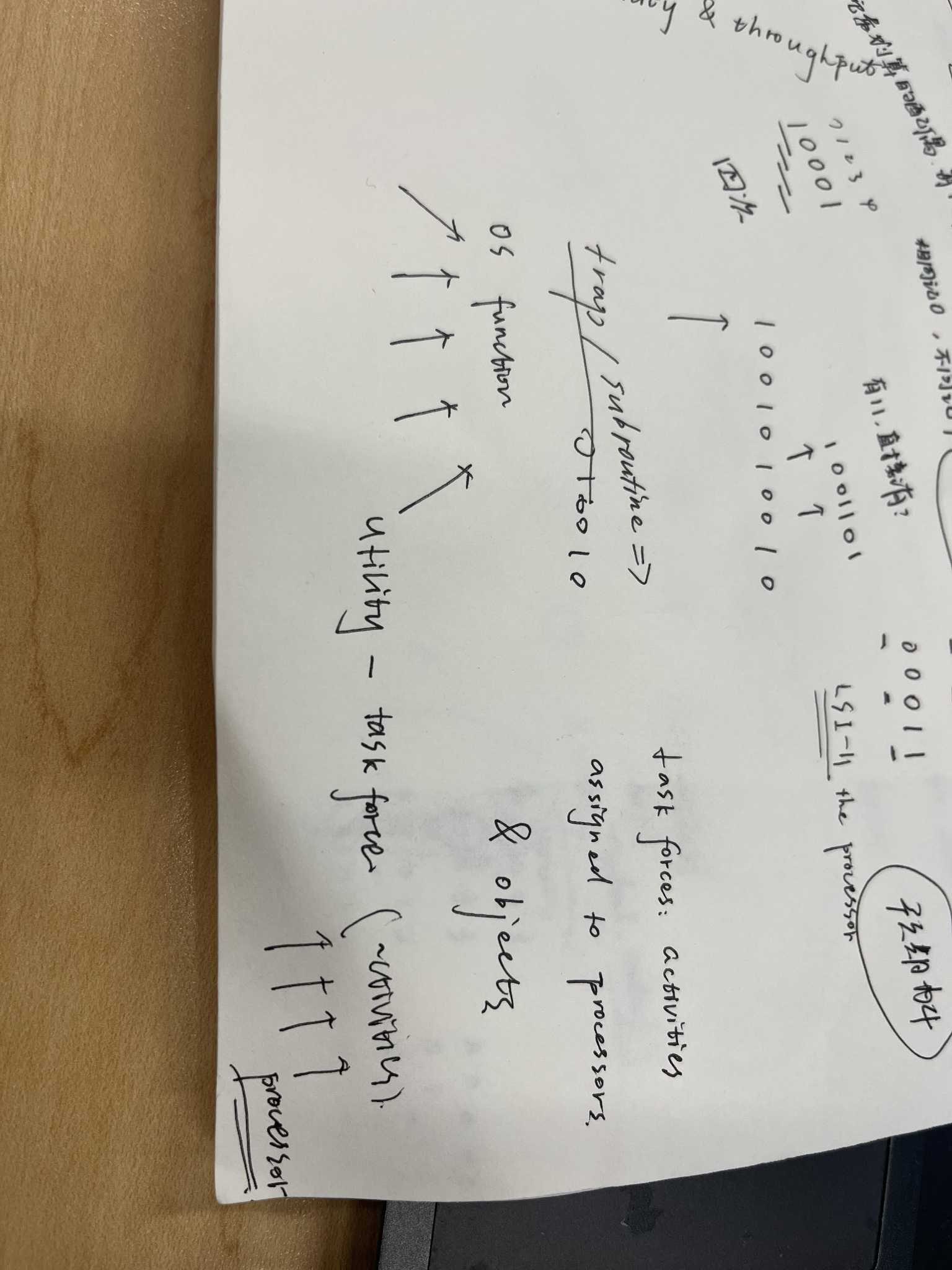
So when a fail happenes, we can just replace the activity in the UDL without restarting it.

objects are accessed by \*descriptors\*, which are stored in the descriptor lists.

* page objects
* pipes and semaphores(stored in kmap);
* file control blocks, task forces, descriptor lists(implemented by OS utility)



## overview of the current system



## pipes and utility communication

pipes: uninterpreted strings of bytes.

Maintaining the byte counts for messages for integrity.

identity of sender is sent to receiver.

waiting for the communication pipe -> processor context swaps -> pause time

## utility priviledges

Each utility activity executes with a status bit.

the descriptor lists:

1. activity - PDL
2. task force - SDL
3. processor - UDL(utility)
4. XDL(external descriptor list): presenting the absolute system name of a descriptor list to its Kmap. -> right amplification

trade off: the complexity and the distributed.

## the internal strucutre of utilities

deadlock problem: comes from the circular dependencies

solution: provide service resources dynamically from a large service pool.

the conditions for deadlock free:

1. functions are divided into service classes
2. each utility can provide services in one service classes
3. the number of utility is dynamically allocated.

(this part read further)

## exception reporting

exceptions: internal && external

buddy: solution to the exceptions remotely, can handle for all kinds of exceptions

exception classes -> each own a handler

tha activity who explicitly uncovers the exception: internal exception; for other activities who share the objects: external exception.

\tip - spin wait

spin: continuously checking

wait: wait for a mutex