Performance (QoS) "improve response time"

Application layer

Application Program

IDL

stub

System layer

ORB/SOAP

ORB

Network layer

Translate CorFwd ()

Translate Bwd ()

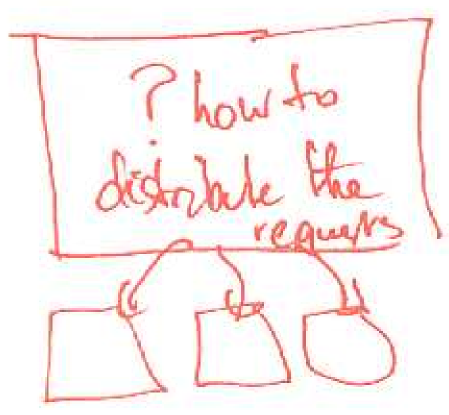
Marshalling ()

KERNEL (type engine)

WAY IDL/AP are designed? (e.g. changing the sets of refs into sets of structs)

Reduce the # invocations (independently on the application) ≡ CACHING

implementation dependant



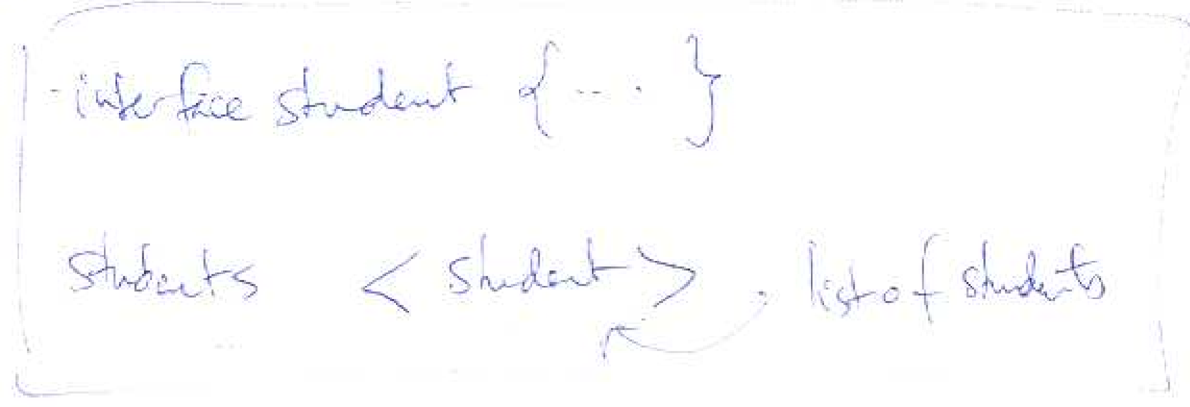
@ Server level (very high # of requests)

LOAD BALANCING

LAN (Local Area Network)

L5-1

JDL

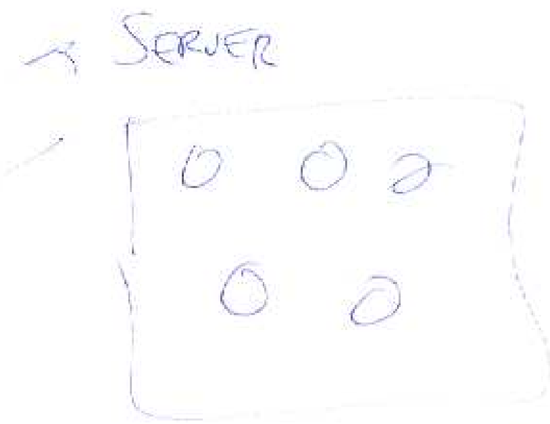


- interface student {... }

students < student > , list of students

(1) instance of students

<ref1, ref2, ... ref1000>



(2) on each element of the set, you apply a certain operation (e.g GetNAME(), GetMARK()).

(1) + (2) numbers of calls to server ≡ 1000 + 1 ≡ 1001

if N is the size of the set ⟹ N + 1 calls

(5-2)

A possible solution is to define Student as Struct (structure)

Struct Student { – }

Students <student>

How many calls to the server do we need to get all the information about students? only 1

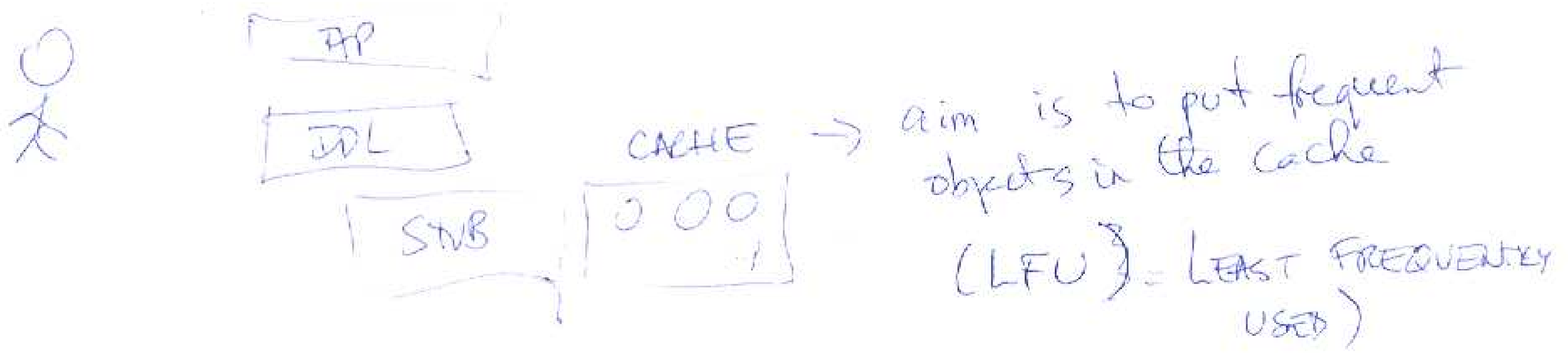
< [age; name], [ ], [ ] >

all values of the various attributes of the different elements of struct

LS3

CACHING

improve the locality of objects (not all objects, only those that are "critical". e.g. frequent objects, big objects, etc)



→ aim is to put frequent objects in the cache (LFU = LEAST FREQUENTLY USED)

When the cache is full
- eviction: which object to evict to allow a new object to be included
- Consistency

↳ increase HIT RATE ≡ high probability to find the object requested in the cache

(L5-4)

LOAD BALANCING is about IMPROVING THE <u>RESPONSE TIME</u>
(Performance)

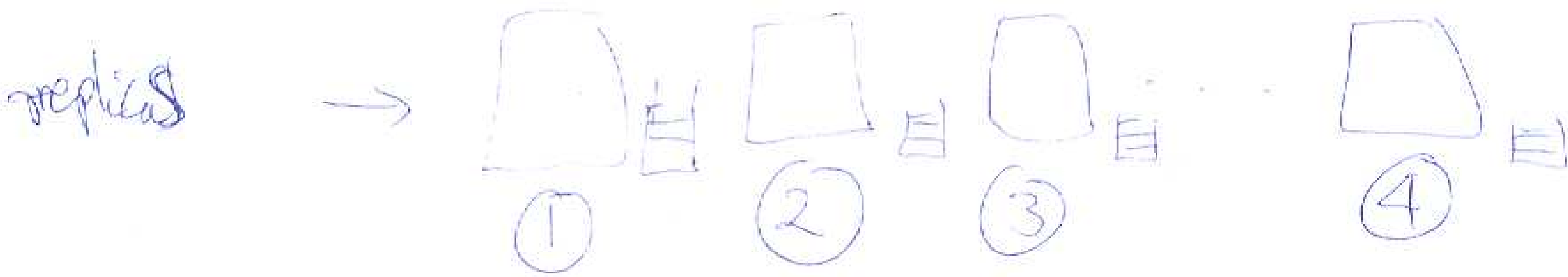
by <u>replicating</u> the Server

REPLICATION is also about replicating the server, but
the focus is in improving AVAILABILITY

if you want BOTH (performance + availability) ⟹
Sacrifice the degree of performance (ie
<u>not</u> the best response time) and availability
(<u>not</u> very high ≃ 60% available service)

(L5-5)

LOAD BALANCING (@ SERVER SIDE)



DISPATCHER
(Issues)

- LOAD INDEX (computes the load at each server)
- Task assignment (making a decision about which server to assign the task)
- Task migration

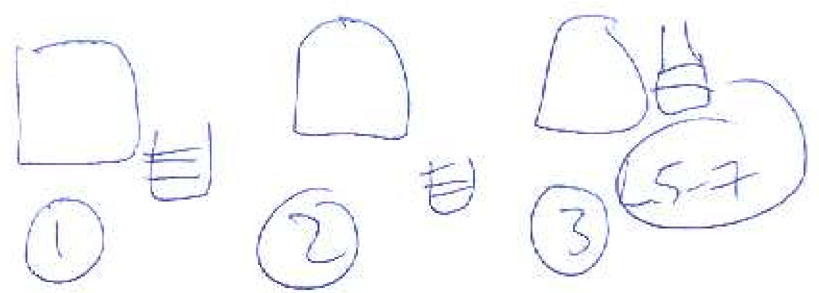
L5-6

when you know the load INDEX, NORMALLY SPEAKING the task assignment policy is straightforward (i.e. because you assign tasks to the least load first (LLF) server as this server has the smallest # of tasks in it QUEUE)

TRUE : when the task size distribution is EXPONENTIAL $\Rightarrow$ LLF is the best strategy

$P(X > x) \propto e^{-\lambda x}$

$x \uparrow$ $P(\text{find a big task}) \simeq 0$



FALSE :

the task distribution is **not** exponential

(PARETO Distribution)

$P(X > x) \not\to 0$

probability that a task goes beyond a certain threshold in **not** nil

HIGH variability in the task size (e.g HTTP traffic)

LLF (is based on the shortest queue)

is **NOT** optimal

(L5-8)

Slow Down (SD)

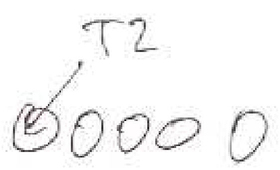
because you have high variability in the task size distribution (ie it is not exponential)

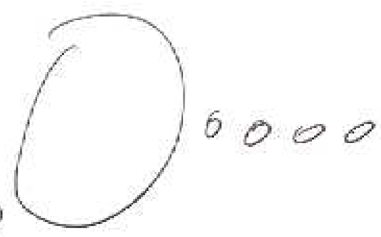
$P(X > x_0)$ = probability that the task size is bigger than x_0

(HTTP traffic) Pareto $P(X > x_0)$ when $x_0 \uparrow$ (high) $\neq 0$

exponential $P(X > x_0)$ when $x_0 \uparrow$ (high) $\simeq 0$

80% of tasks are small





in order to compare T1 and T2, you need to use a different metric (than waiting time)

Aim

$$\frac{\text{Waiting Time (T1)}}{\text{Task Size (T1)}} = \frac{\text{Waiting Time (T2)}}{\text{Task Size (T2)}}$$

→ fairness

fairness ≡ treat the tasks in the same way i.e. the ratio of waiting time by the task size is similar for ALL tasks

T1: Task size = 100, Waiting Time = 10

T2: Task size = 10, Waiting Time = 1