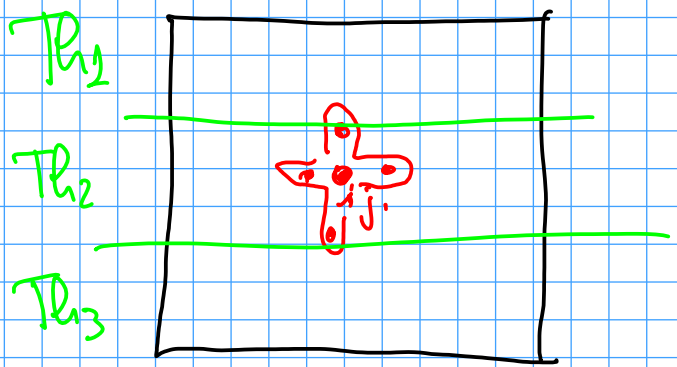
Synchronization



A stencil (R)

∀ iteration ↓

B def (W)

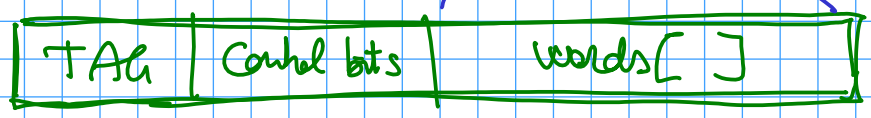
Stencil computations ⇓ owner computes rule

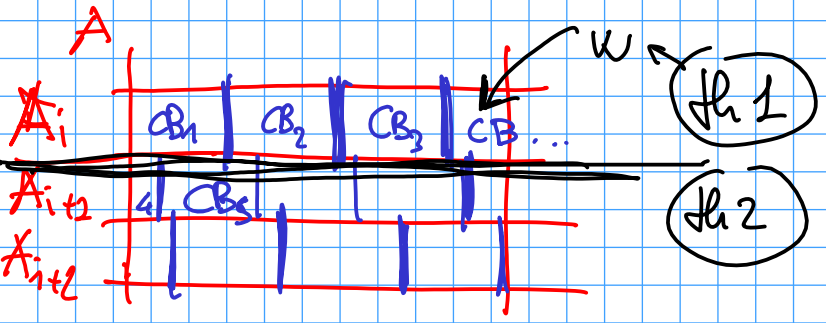
```
Thj : for(int i = initrow[j];
          i< endrow[j]; i++)
      {
        "mutex" →  X[][] = .. X[i][j] ...
      }
```

lock [N][N]

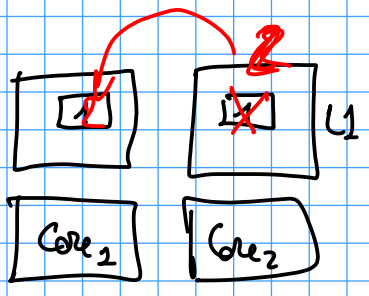
→ try to avoid any kind of implicit synchronizations

cache line



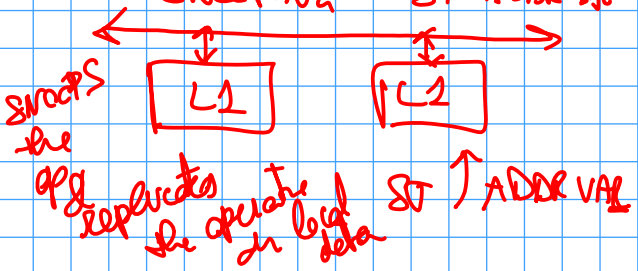


L1 ≠ L1



cache coherence protocol

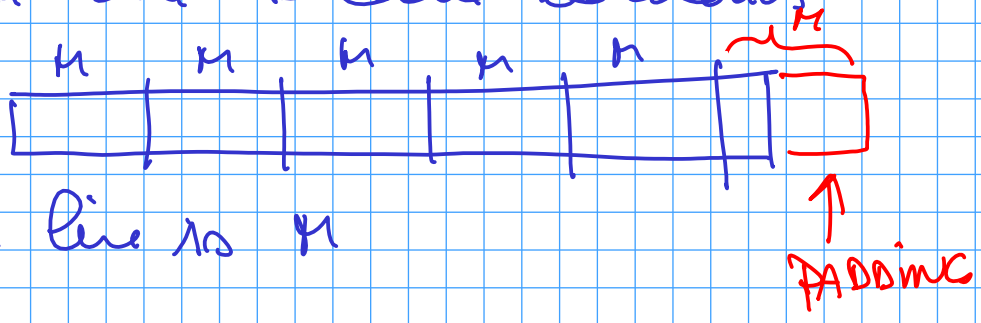
directory based



⇒ PADDING TECHNIQUES

↓ align data to cache boundaries

A[N][N]



size of a cache line is M

$A[N][\lceil \frac{N}{M} \rceil M]$

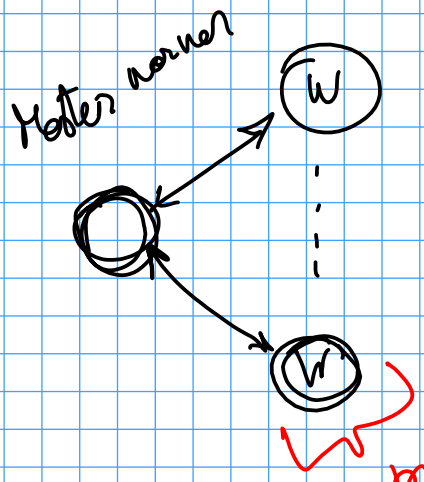
CLIENT SERVER

Server ::

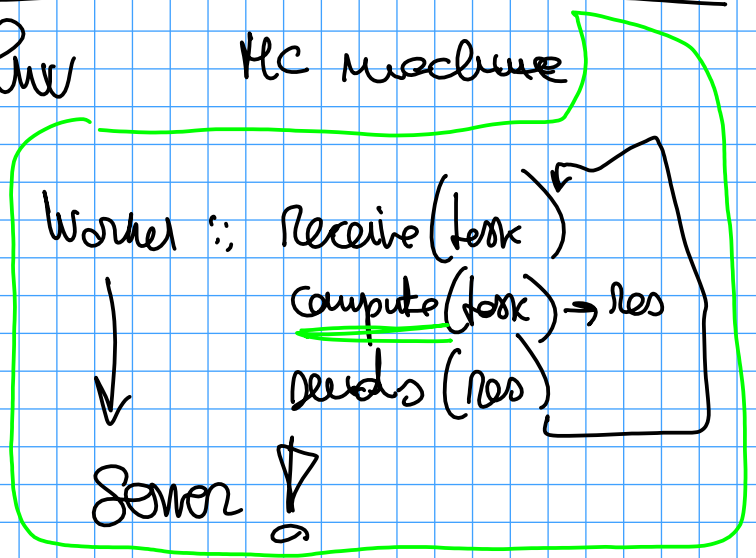
socket create
bind socket
(listen)
accept
↳ socket
read req
write answers
close

MT Server ::

socket create
bind socket
(listen)
accept
↳ socket
fork
pthread_create
process
read
writes
closes
exits
pid



CoW/NoW
(MC mode)



processes on a MC machine → MT server

4 cores → ALTERNATIVE 1
placing 4 "workers" on 4 diff ports

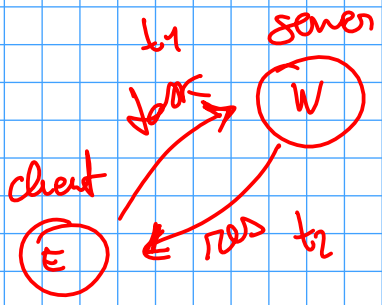
ALTERNATIVE 2 (the good one)
place a server (MT!)
where you use a thread pool with 4 th

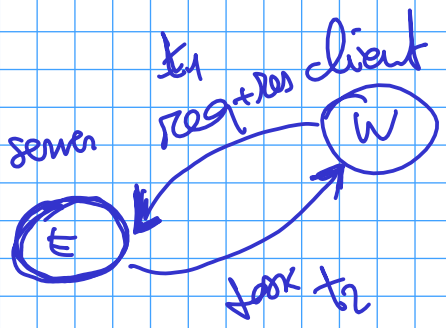
ALTERNATIVE 3 (the best one)

Worker Server :: socket / bind / listen
prepare the server socket

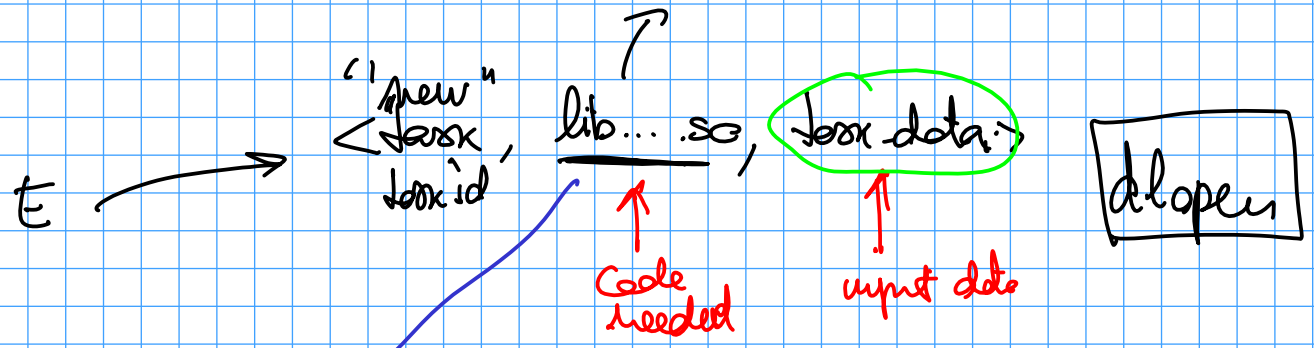
for(...) pthread_create(...)
prepares the thread pool

while(...){
  S = accept(...);
  activate 1 thread from the pool
}





Tout compute (Tin t)



Subject to optimization
→ first time → send code
→ i ... n → send a ref to code

W : load .so library
invoke compute() with the input data
res =
send back(res)