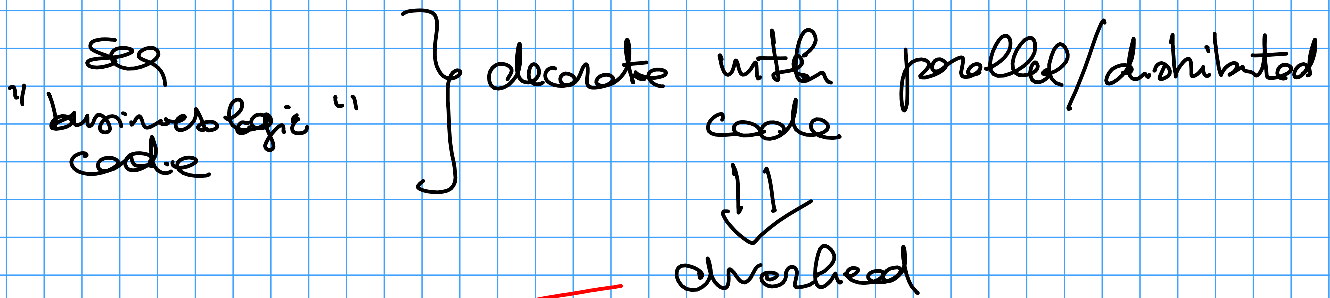
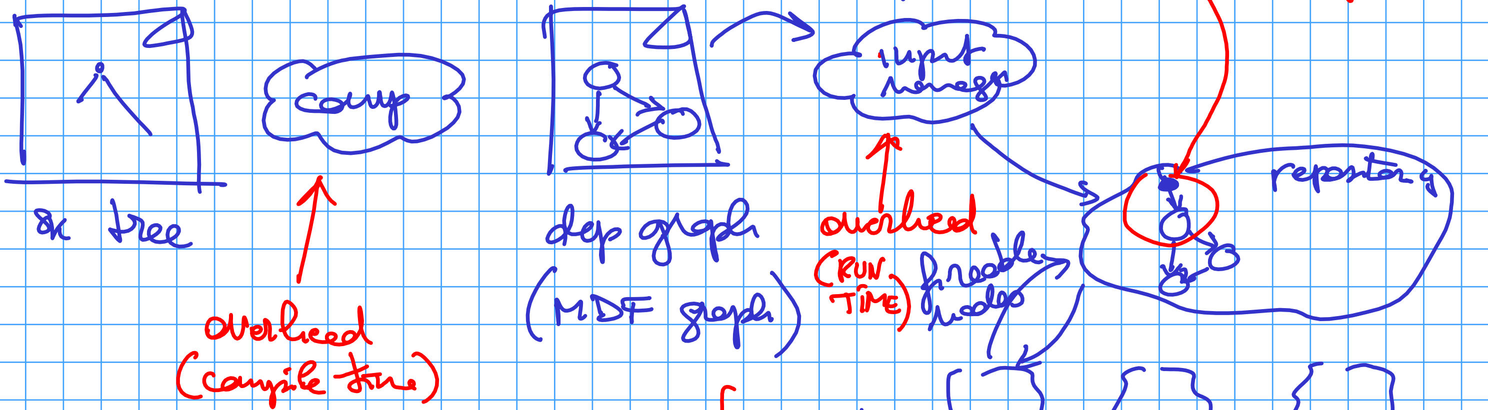
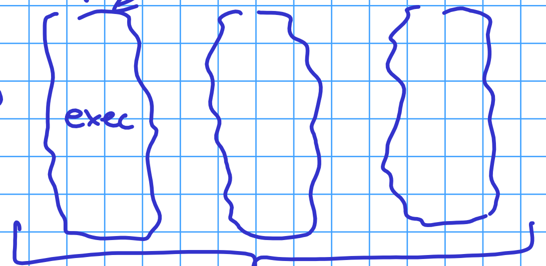
# DF implementation




TASK:
Reduce overheads



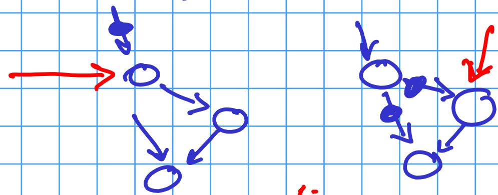

interpreters

ONCE & FORALL
- setup / termination

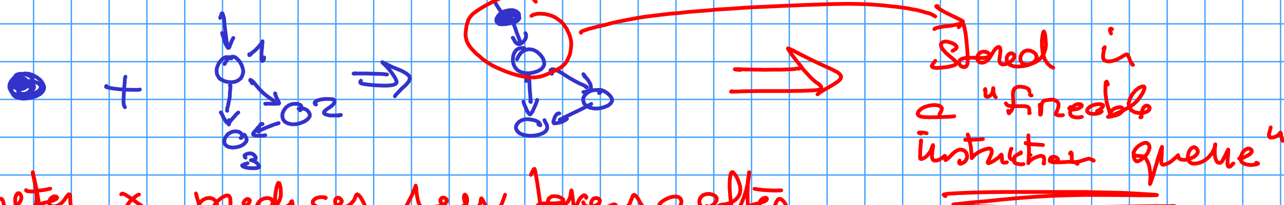
PER NODE computed
- fireable nodes moved to interpreters + code & + data
- synchronize accesses to the repository
- move results (tokens) back to the repository

main overhead introduced (RUN TIME)

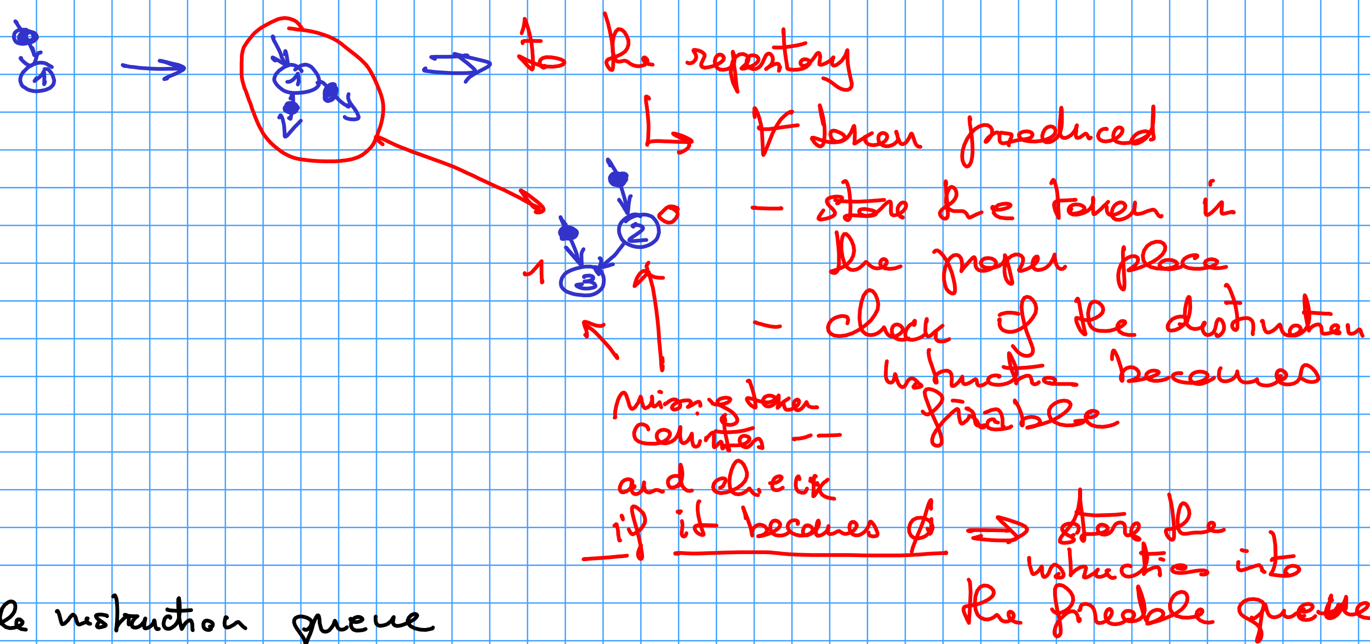
find fireable instructions / node



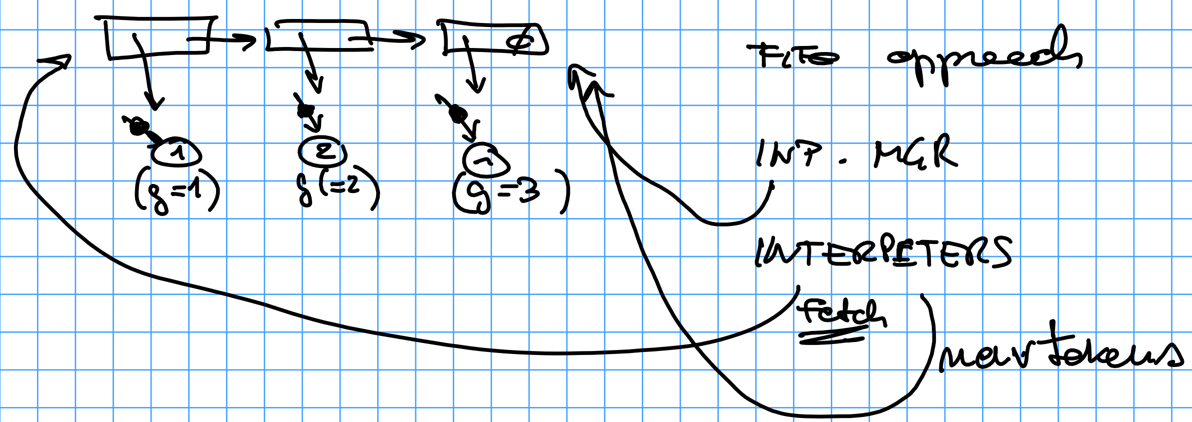
1) input data set available (INPUT MANAGER)

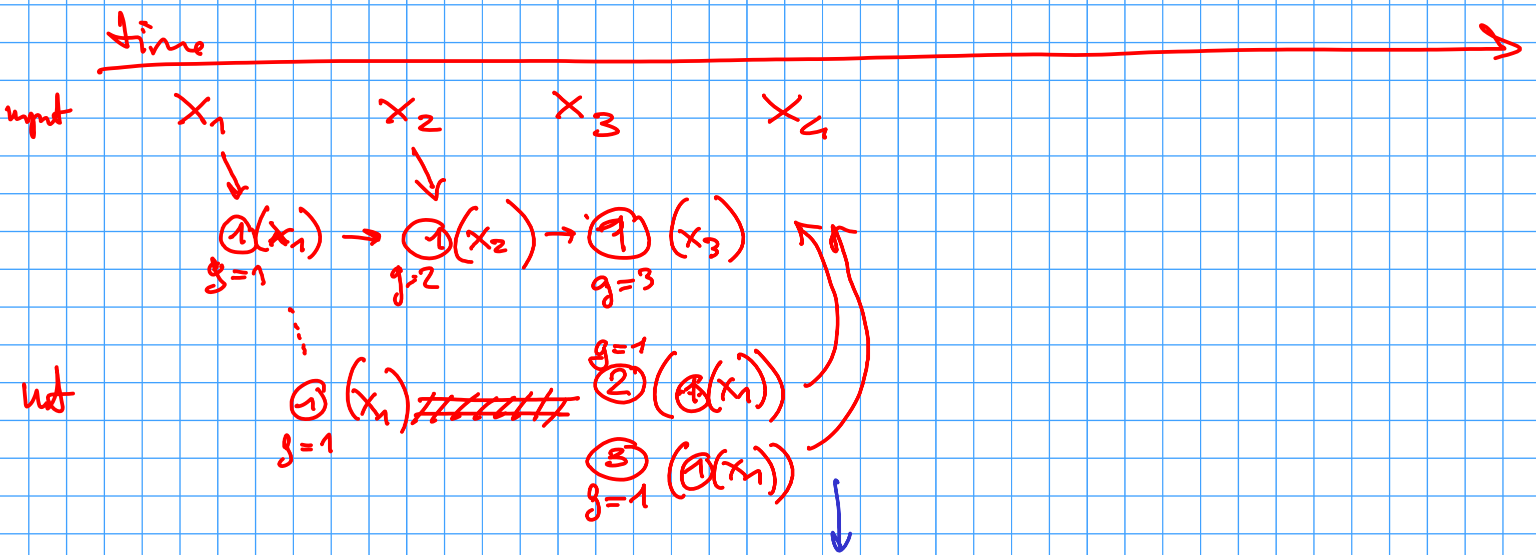


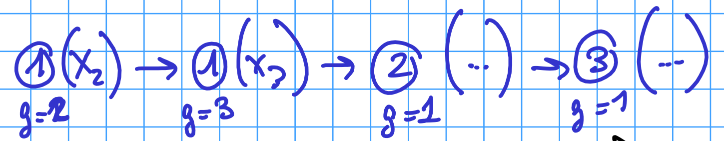

2) interpreter x produces new tokens after computing some instruction



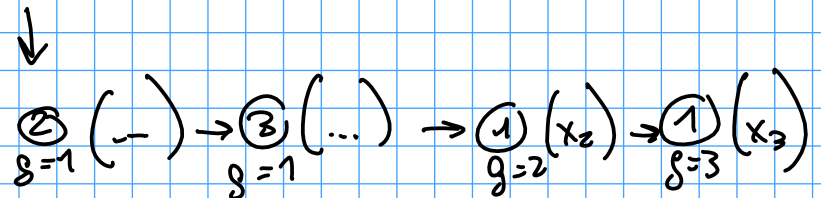

fireable instruction queue

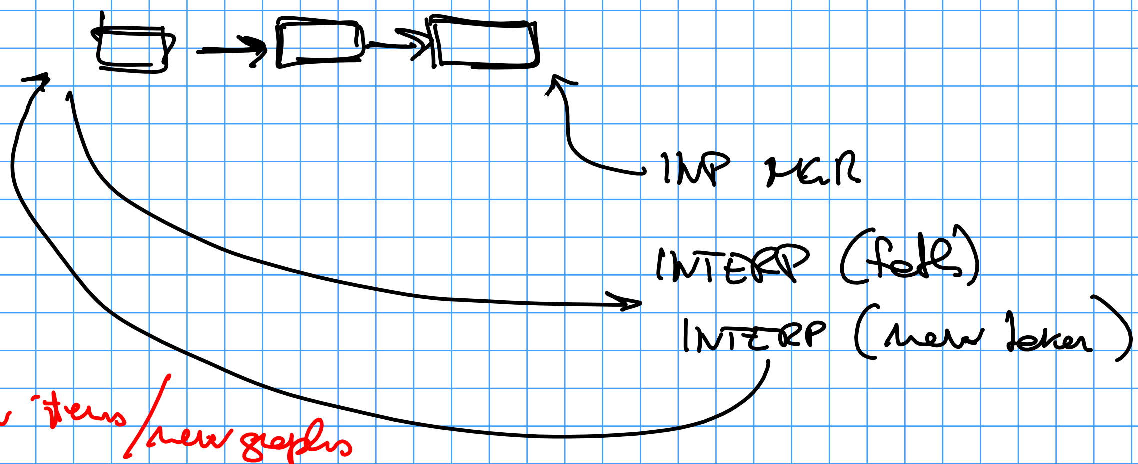


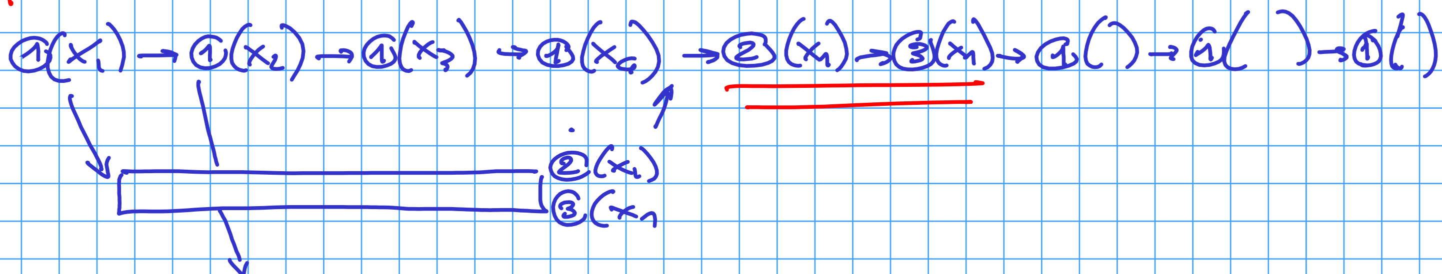


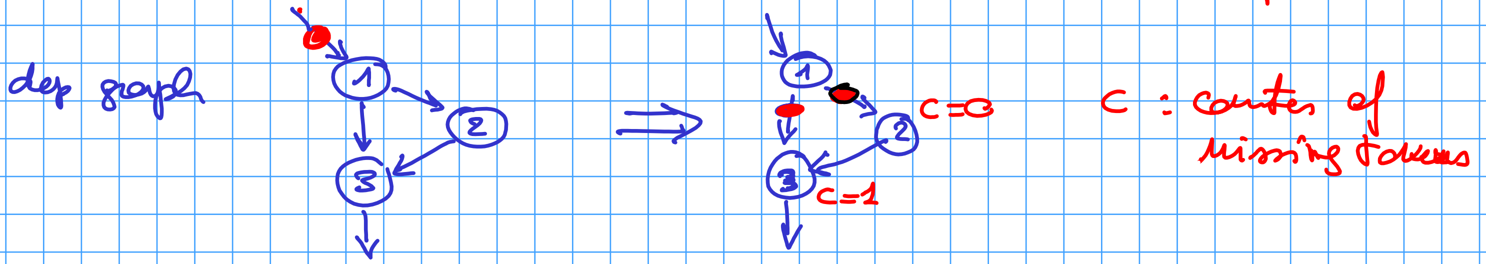
FIFO policy + priority on the graph id

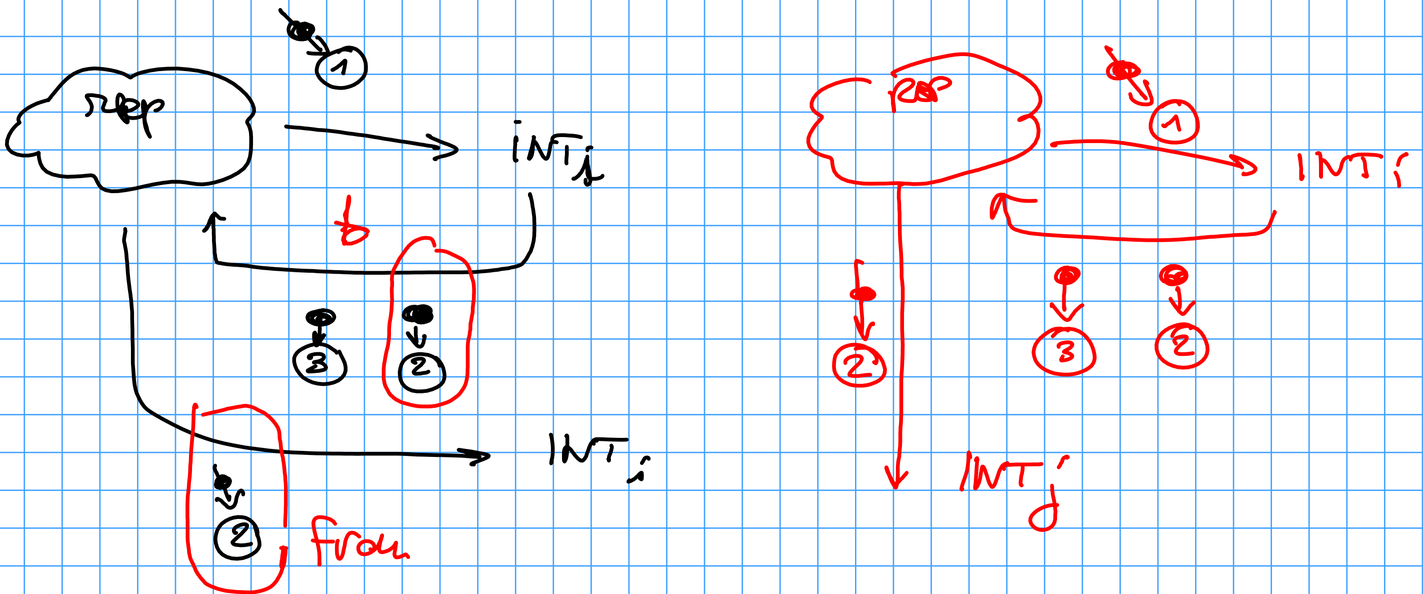




source overhead: moving data & code to/from interpreters




Reduce overhead ⇒ avoid communications

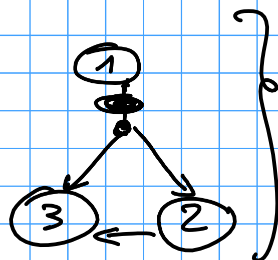
$INT_i$ sends token data to REP & keeps a copy in a local cache

REP sends ② (ref to the locally cached data) instead of sending copy of the whole data

further optim.:

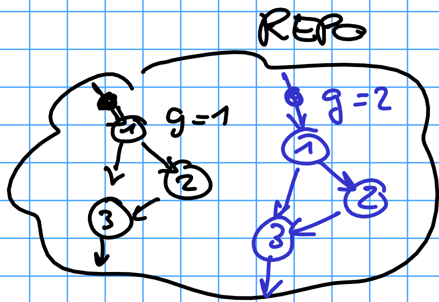
REP sends ① → ② ↓ ③ to $INT_i$

$INT_i$ computes ①, sends ③ to REP & then computes ②

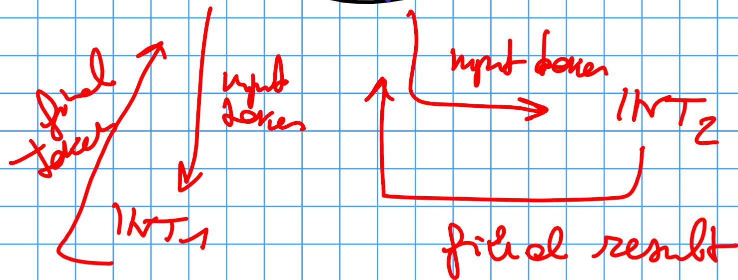



different kind of graphs with different needs as far as caching is concerned

similarities in directory based cache coherence protocols
& manager in the repository
+ interpreters talking each other




⇒ schedule the black graph to $INT_1$
& the blue one to $INT_2$




⇒ no data parallelism
stream parallelism only

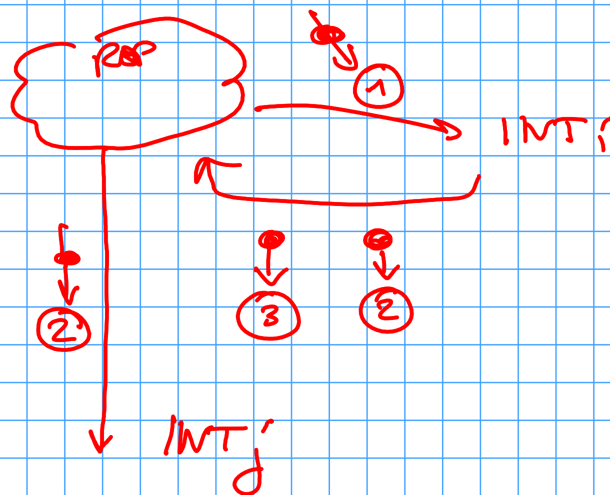
⇓

good throughput
bad latency

Reduce overhead ⇒ do apply better scheduling

⇓

AFFINITY SCHEDULING

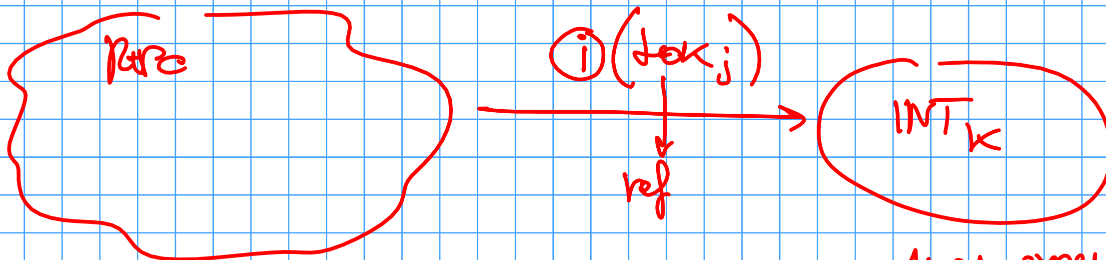


# AFFINITY SCHEDULING

INT : get token$_i$ computes produces token$_j$ and keeps token$_i$ & token$_j$ in a local "cache" managed with LRU policy
(sw cache)

REPO : try to schedule the computation
graphnode ( param$_1$ ... param$_n$ )
token$_i$ token$_j$
where he knows token$_x$ needed by the computation have been produced

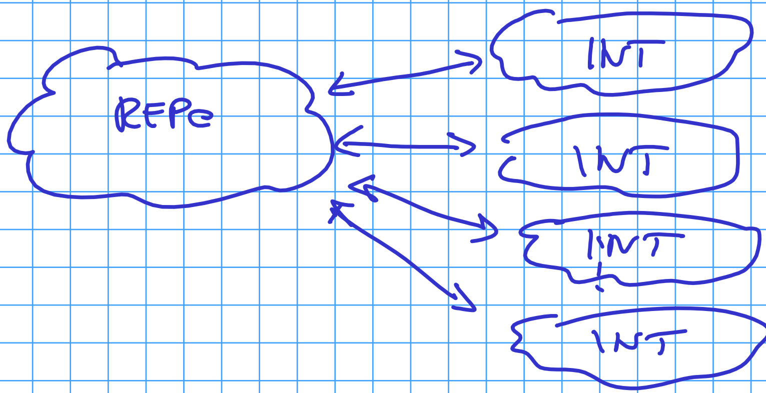
knowing that INT$_k$ produced tok$_j$

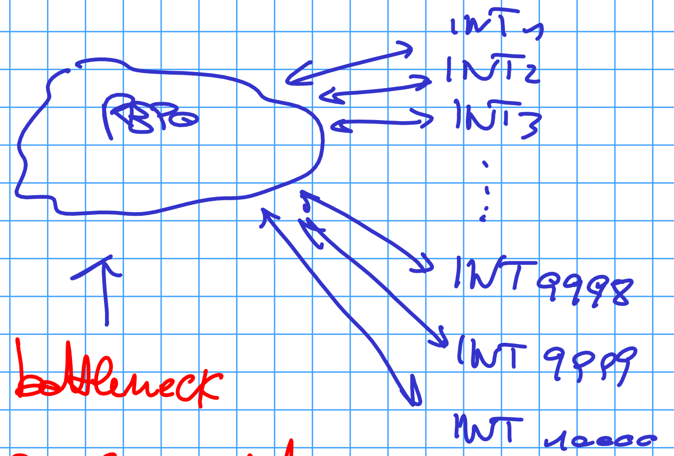



may experiment a token cache "fault"
then ? → see (*) below

⟹ when INT$_k$ produces token$_j$
① stores token$_j$ in the local cache
② deploys back token$_j$ to the REPO

then ? (*) raise a fault
- asking the token$_j$ to the REPO
- wait
- when it comes : compute



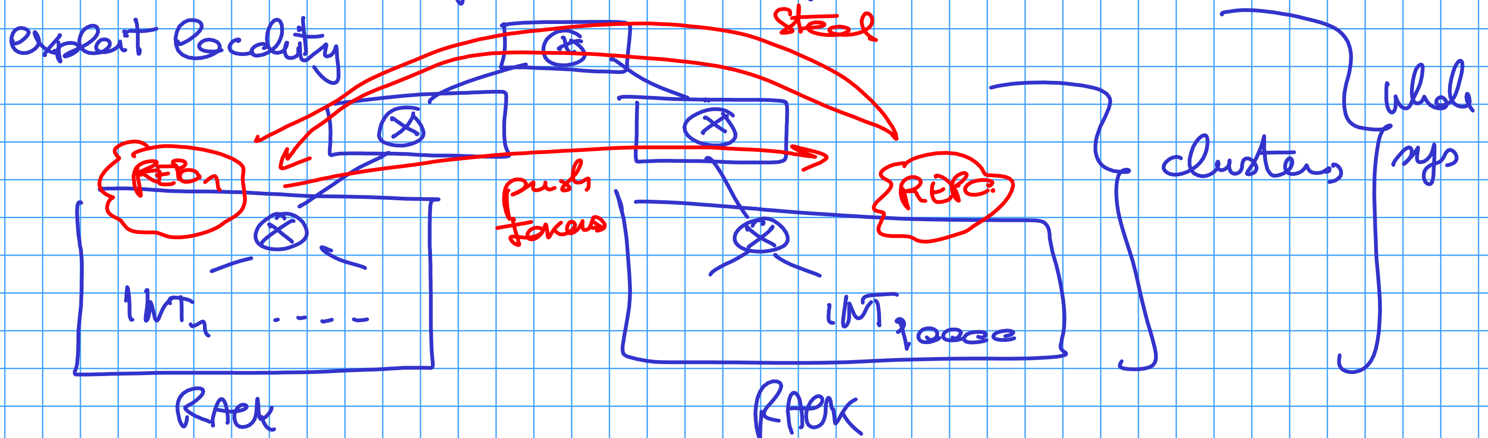



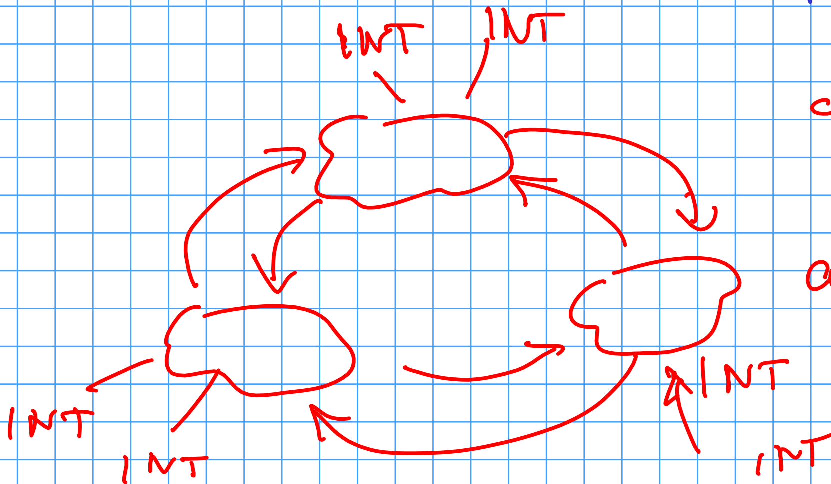

single point of failure

Reduce overhead ?

How ?

⇒ parallel repository




clockwise ring to push tokens

anticlockwise to fetch freeable instructions