Issues of Choosing Paradigms

- Interactive? or
- Concurrent?

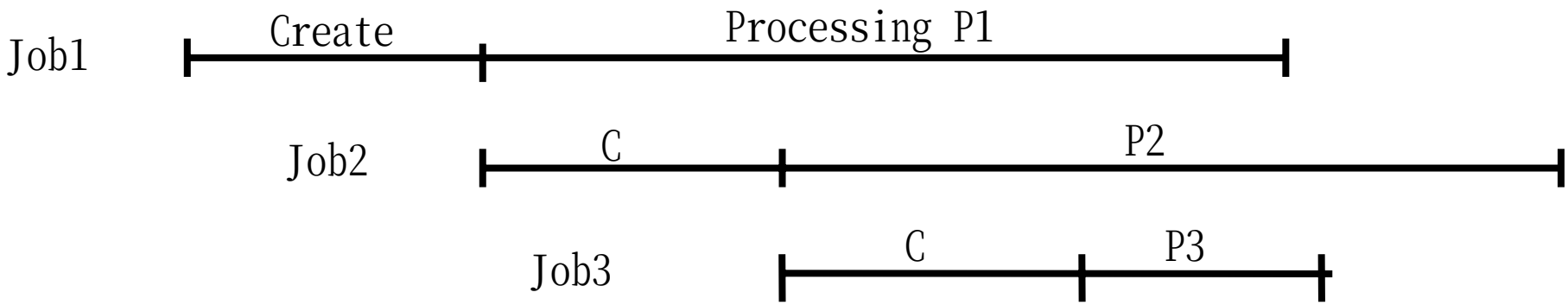
Factors for considerations:

- Level of concurrency

  the number of processes at a give time
- System Resources
- Cost of Concurrency
- Overhead and Delay
- Easy to program

P > C

Concurrency

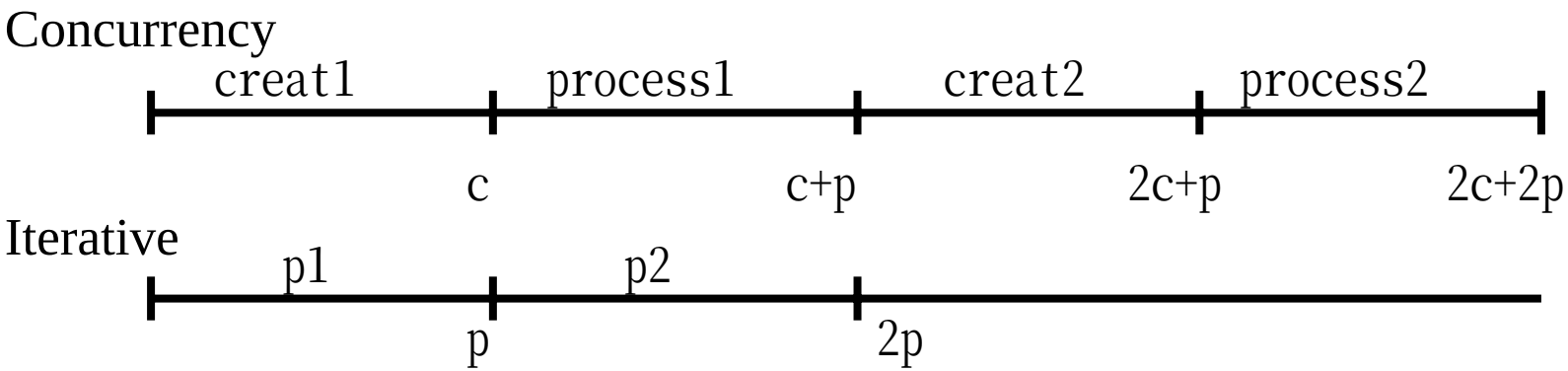


Interactive

Interactive paradigm will:

- cause unfair situation for some jobs
- Reduce Concurrency (increase delay)

$P \leq C$



- Concurrency will affect response time
- extra delays ⇒ requests get lost

  when request rate is > 1/c but ≤1/p

  ⇒ burnt at

But

- few servers operates close to its maximum throughput
- Rarely, P ≤ C

Suitable for small processing time problems.

Preallocation

Preallocate some salve processes.

- Each process iteratively receive requests
- Master process may die or become a slave (See Fig 15.2 and 15.3)

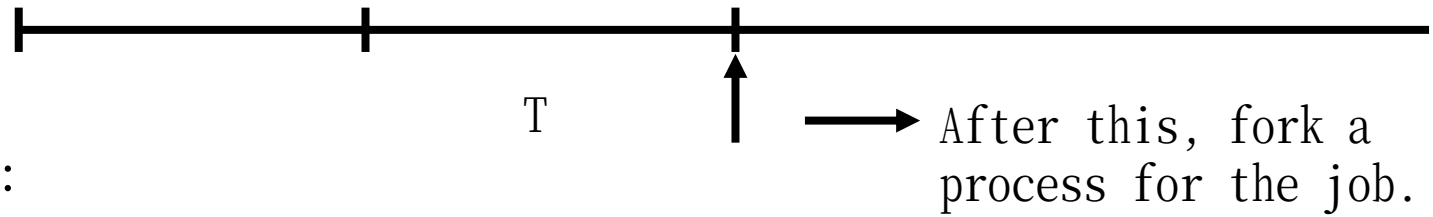
Advantages:

- No overhead for creating a process.
- More levels of concurrency
- Less loss $\Rightarrow$ UDP discards datagrams when queue is full. This method can catch it quickly. E.g. NFS
- Good for multiprocessing systems

Disadvantages:

- 佔 resource(So, better for small programs)
- Increase network or OS overhead

Delay Process Allocation



優點:

- if P ≤ T, just use one process.
  No overhead for C
- if P > T, overhead C is not significant anyway.

UNIX can do it! Exercise?!

Combine D.P.A. and P.A.P.

- no preallocation process first.
- Any delayed process will not die.
  Use maximum # M to control max levels

Select Problems

- Problems of select & poll:
  - A linear time operation.
  - Too slow for a large number of sockets.
- Solution:
  - In BSD Unix: kqueue
    - BSD Unix is heavily used by Yahoo!.
  - In Linux: epoll
    - Linux is heavily used by Google
  - Some fast Web servers using this. E.g.,
    - Zeus
    - thttpd (used by 無名小站?)