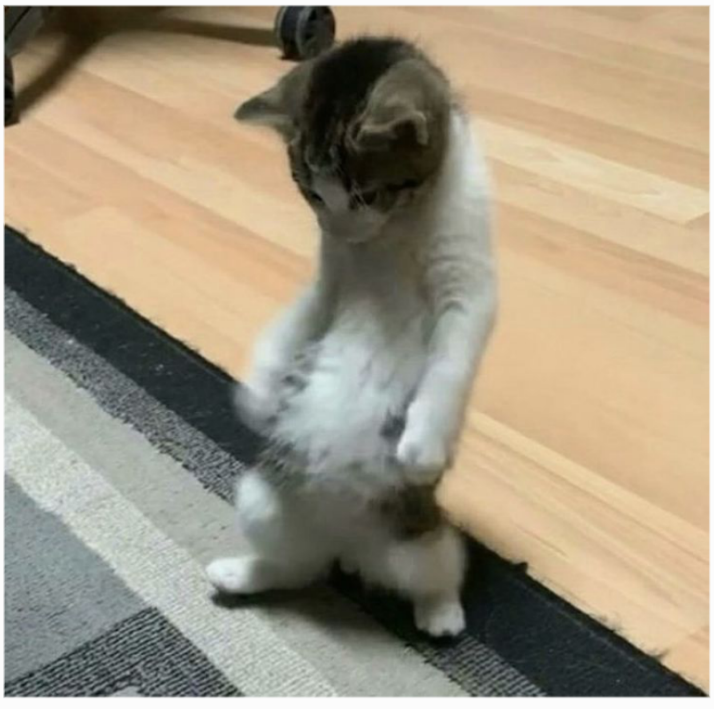
2. Processes



Process : running program

problem : "How to provide the illusion of Many CPU?"

Time sharing of the CPU

- allows users to run as many concurrent processes as they would like
- Potential cost : performance; CPU shared => run slowly

To implement virtualization of the CPU, OS need

- Mechanisms : low-level machinery
  - low-level methods or protocols that implement a needed piece of functionality.
  - answer to "how" question
  - ex) Context switch

- Policies : High-level intelligence
  - algorithms for making decision within the OS
  - answer to "why" question
  - ex) Scheduling policy

Space sharing

: resource is divided among those who wish to use it.
ex) disk space

Process's **Machine state**

- what a program can read or update when it is running.?
- At any given time, what parts of the machine are important to the execution of this program?

Component of machine state

- Memory : include instruction & data that program r/w
  - Address space : the memory that the process can address
- Registers : instructions explicitly read or update registers
  - Program counter, stack pointer, etc

Process API

- Create
- Destroy
- Wait : wait for a process to stop running
- Status

Process Creation

- In early OS, the loading process is done eagerly : all at once before running the program
- Modern OSes porform the process lazily : loading pieces of code or data only as they are needed during program execution (-> Paging, swapping)

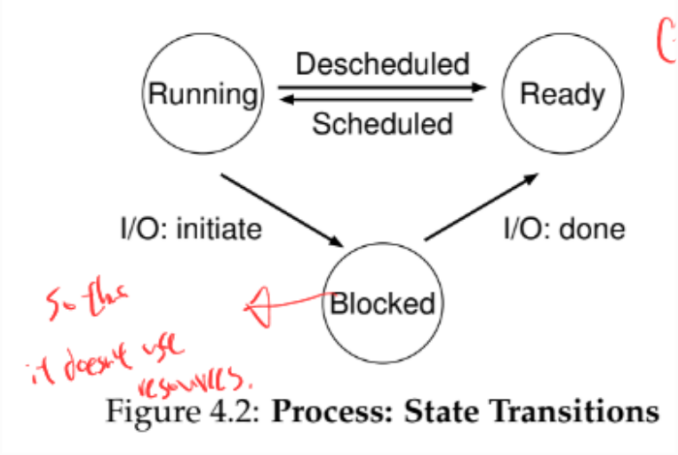


Figure 4.2: **Process: State Transitions**

Process States

- Blocked : not ready to run until some other event takes place
  - It is "that" process which is blocked, when it initiates I/O request
- Difference Between Ready vs. Blocked
  - Ready : don't use CPU resources (waiting in ready queue)
  - Blocked : dont; use CPU&memory resources

Process list

: data structure that save processes

A process consists of instructions, and each instruction can just do one of two things : use the CPU / issue an IO (and wait for it to complete)