3 processes:

P1:

```
____

if (c) send(p2,m);
while(1) {
  receive(p3,m);
   ____
  send(p2, m);
}
```

P2:

```
____

while(1){
  receive(p1,m);
  ____
  send(p3,m);
}
```

P3:

```
____

while(1) {
  receive(P2,m);
   __
  send(p1,m);
}
```

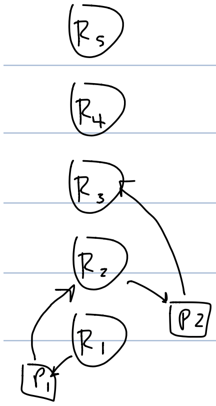
(1) deadlock prevention

a. mutual exclusion must be enforced

b. no-preemption must be enforced

c. hold-and-wait must be enforced

d. circular wait must hold



deadlock prevention schemes

(1) wait-die   priority

(2) wound-wait P_i, P_j

$$\underset{5}{\overset{\parallel}{P(P_i)}} \underset{<}{>} \underset{10}{\overset{\parallel}{P(P_j)}}$$

$P_1 \rightarrow P_2 \rightarrow P_3$

wait-die ↗older waits | wound-wait → newer waits

time stamp

$\underset{TS(5)}{\overset{\parallel}{P_i}}, \underset{TS(10)}{\overset{\parallel}{P_j}}$ P_i, P_j

(2) deadlock avoidance

safe state1

req. → ↓

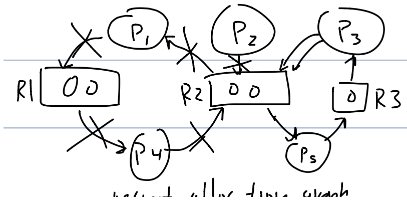
safe state2

12 tape drives

process	max needs	allocated	still needed
P_0	10	5	10-5=5
P_1	4	2	4-2=2
P_2	9	~~2~~ 3	9-2=~~7~~6
12-9=~~3~~2		~~9~~ 10	

(3) deadlock detection

(graph reduction method)



request allocation graph

request allocation graph

(4) deadlock recovery

- priority of the process

  real-time, iteractive, batch

- cost of restarting process

  Shell, internet browser, text editors, finite element software

- current state of the process

Skip Chp 6

memory management (chapter 7)

source code (in memory) $\xrightarrow[\text{compilation}]{\text{compiler/assembler}}$ object module (in memory) $\xrightarrow[\text{linking}]{\text{linkage editor}}$ load modules (on disk) $\xrightarrow[\text{loading}]{\text{loader}}$ load module (in memory)

logical address space

physical address space

logical-to-physical address binding/assignment

(1) earliest moment
static binding

  low kernel code, real-time, embedded

(2) compilation

(3) linking time

dynamic binding (relocatable)

relocatable

absolute: R1, immediate operands