Maekawa's Algo.

- Each site's request set is constructed so that
  - Intersection of request set for any pair of sites is not null
  - Each site is in its own request set
  - The request set size is K for any site.
  - Each site is contained in K sets (K = sqrt(N))
- To request
  - Site Si sends REQ(i) to all sites in its request set.
  - On receiving the request, Sj will send REPLY(j) if it hasn't sent a reply to anyone since it got the last release. Otherwise hold.
- To Execute CS
  - When you get all Replies
- To Release CS
  - Send Release(i) to all sites in request set.
  - When Sj gets release message, it sends reply to next waiting request.

- Need 3*sqrt(N) messages, 2*T synch. delay.
- Problem – deadlock can occur
  - Imagine a situation with three sites each requesting CS.
- Solution – prioritize request using timestamps and do some extra processing.
  - Basically, eliminate circular wait. Site will send a failure message if it can’t honor your request.
  - If a site is locked, but receives a request from a site with higher priority, it “inquires” from the locking site to see if the lock can be released.
  - Message traffic now 5*sqrt(N)

Token Based

- Suzuki Kasami Broadcast Algorithm:
  - Basically, need a token to get into CS. Site possesing the token can get into CS repeatedly. RN is an array of integers denoting the largest number in request sequence from a site. The token itself has an array LN containing sequence number of most recently executed request and a queue Q of requesting sites.
- Request
  - If requesting site does not have token, it increments RNi[i] and sends REQ(i, RNi[i]) to everyone else. When Sj receives this, it updates RNj[i]. If it has idle token it sends it to Si
- CS is executed when token is received
- Release
  - Set LN[i] to RNi[i]. If RNi[j] = LN[j]+1, then Sj is appended to token Q
  - If token queue is nonempty, delete top entry and send token to that site. This makes it "*non-symmetric*"
- Messages is 0 or N, Snych. delay is 0 or T.

Raymond's Tree Based Algo.

- The site with the token is the root of a tree. Each node has a variable called holder pointing to parent. Each node also has a r-q that contains requests for tokens from children.
- Request
  - To request, send request to parent if your r_q is empty and add yourself to the r_q
  - When you get a request, add to r_q and forward to parent if you have not sent a previous request.
  - When root site gets request, it sends token to requesting site and sets holder to point to that site.
  - When site gets a token, it deletes top entry from r_q, sends token and points holder. If r_q is nonempty, it sends request to holder.
- Execute
  - When get the token and your request at top of r_q
- Release
  - If r_q is nonempty, delete top entry , send token,point holder. If r_q still nonempty, send request to holder.
- Message complexity is O(logN), Synch. Delay is (T log N) /2

- **Do Section 6.14**