**1. Linked list based queueing lock**

Lock: fetch-&-store --> join L

await predecessor to signal

Q&A -- conclusion: only join L needs mutual exclusion, not the curr->me

unlock: remove curr from L

signal successor

if next nil

comp-&-swap: atomicly check whether L->me and set L->nil

if failed, spin until next is not nil

Q&A -- Who is making decision? OS designer & hardware architecturer discuss it...(followed by an introduction of architecture history)

*adv: no contention(one got-it for every node)*

*disadv: if no fancy fetch-&-store, then slow*

**2. Barrier**

Centralized Barrier

count = N // init

atomic decrement(count)

if (count == 0)

count = N

else

spin while (count > 0)

Q&A -- fundamental? cache coherency

*Problem: Some processors might have arrived at the next barrier before last processor finnish the counting.*

*Improvement:*

else

spin while (count > 0)

while (count != N) //don't leave barrier before count reset

Exercise:

How to ensure that processors could leave the barrier before the count is set to N by another processor?

Next topic: sense reversing barrier