Slide 9

TAS:
```
r = x;
x = c;
return(r)
```

Atomic swap:
```
r = x
x = r
return (r)
```

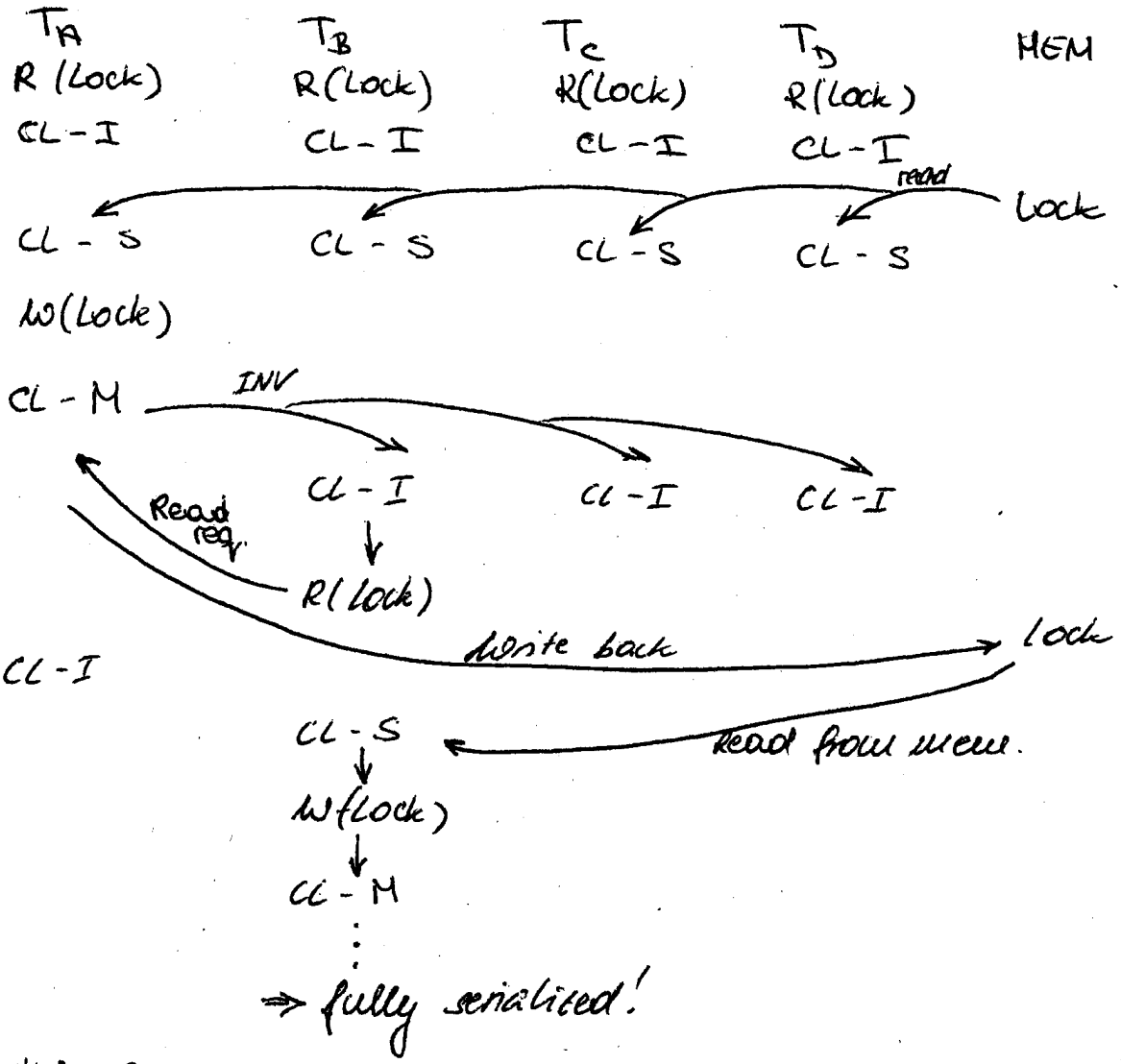
Fetch and Op:
```
r = x
y = OP(r)
x = y
return(r)
```

CAS:
```
r = x
if (r == c)
    x = s
return(r)
```

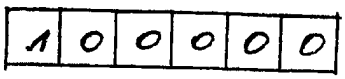
LL/SC:
```
r = LL(x)
:
if x unchanged, x = r; return true
else return failed
```

TSX: (RTM - restricted TM)
```
xBEGIN (jump addr if fail)
    : (xABORT)
xEND
```

slide 12



slide 21



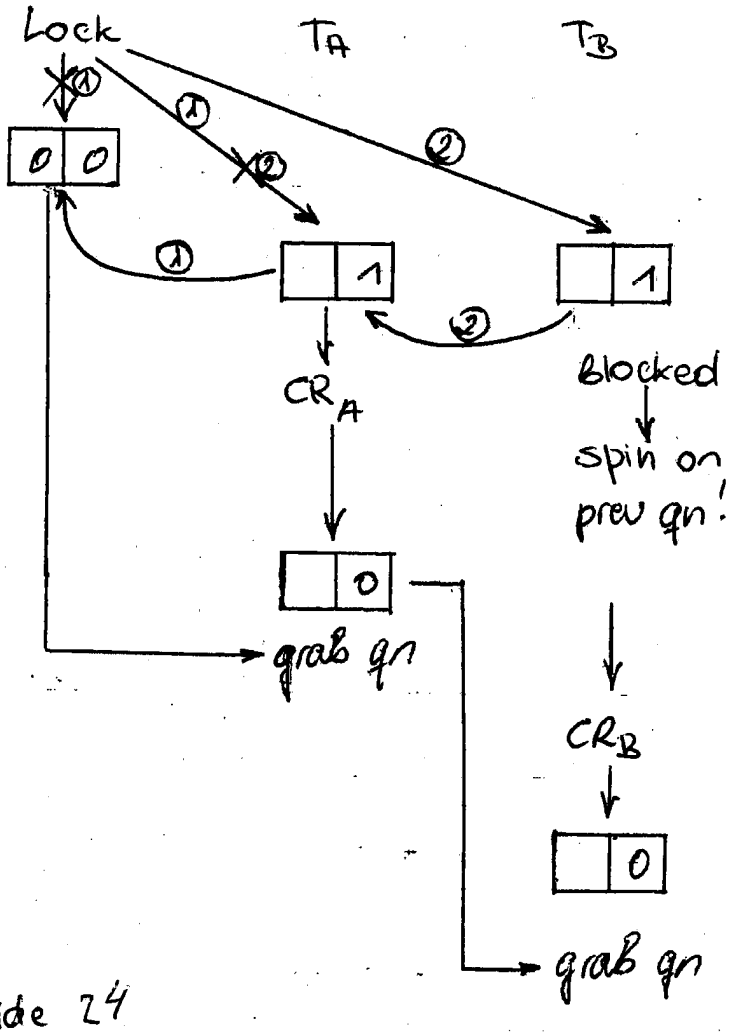
tail = 0;

T_A index = 0;
arr[0] = 1
enter CR

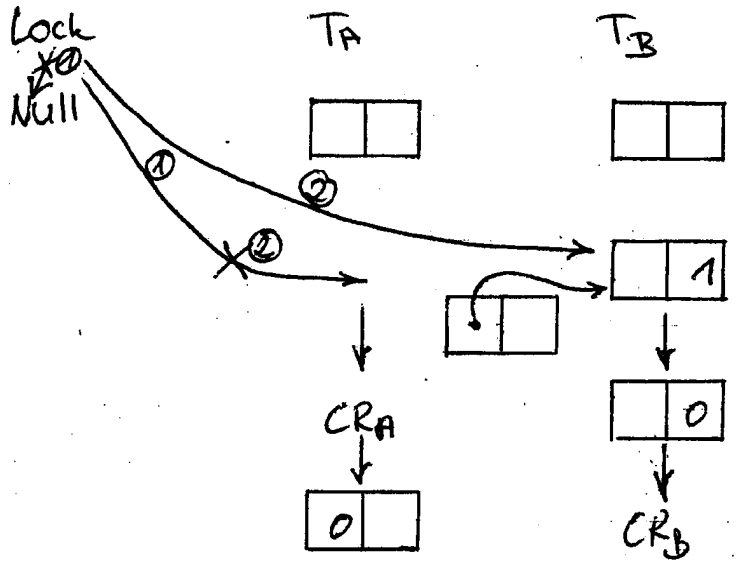
T_B index = 1
arr[1] = 0 ↻ spin

T_A arr[0] = 0
arr[1] = 1
⇓
T_B enter!

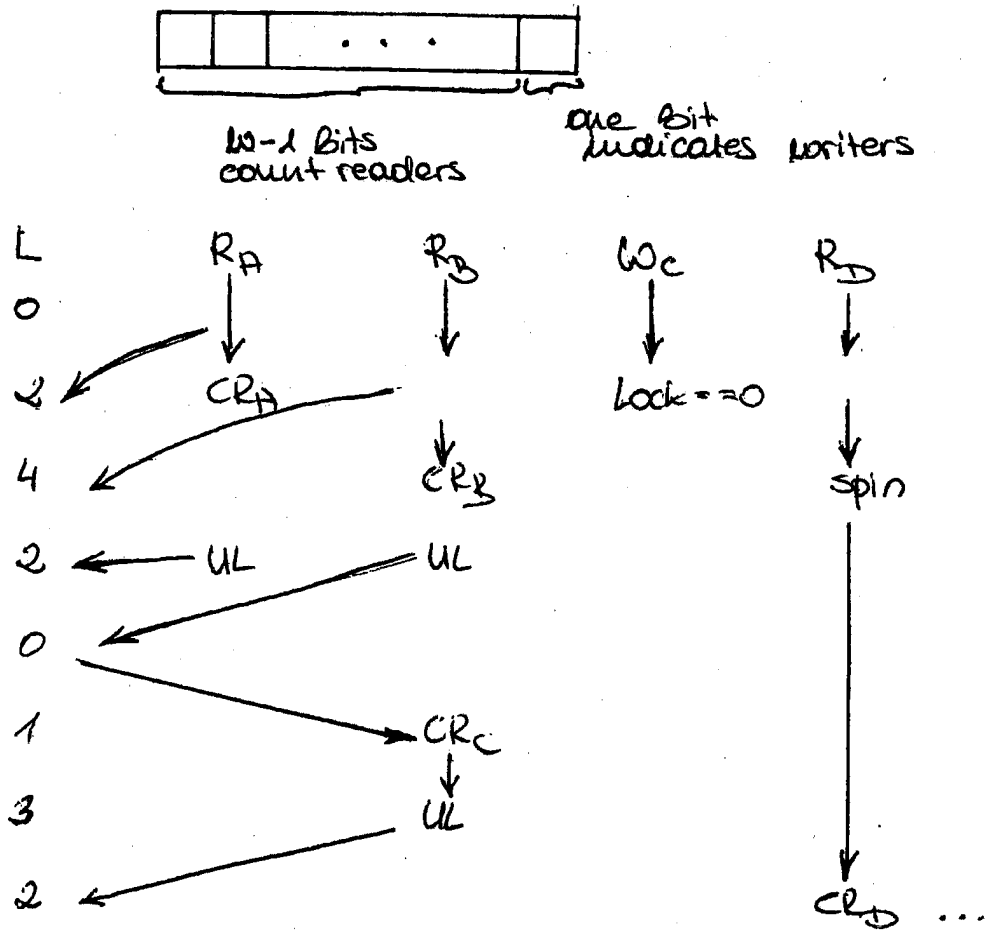
Slide 23



Slide 24



Slide 29



Slide 32

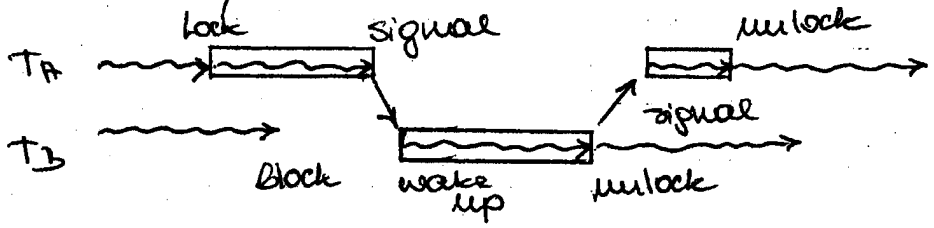
1 - Mutual exclusion
2 - Hold resource, request another
3 - Circular dependency
4 - No preemption

Slide 32

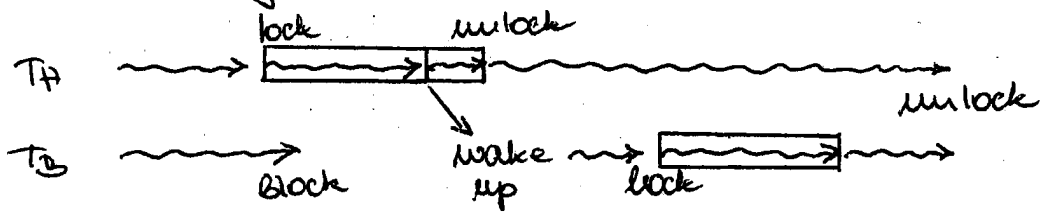
- remove any of two necessary conditions
- e.g. "circular dependency" by same order of lock acquisition!
- ...

Slide 38

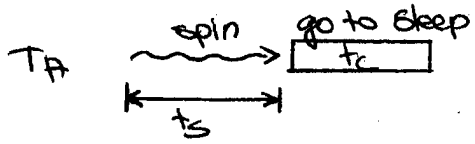
Hoare style



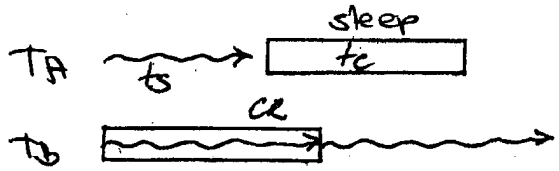
Mesa style



Slide 39



t_s - time to spin
t_c - context switch overhead



Bad! would have been better to keep spinning!

→ optimal spin time depends on how long the other thread remains in CR!

Slide 40

- assume other thread spends time t_x in CR!
- three cases: t = overhead!
  - sleep immediately:
    $t = t_c$
  - sleep after t_c:
    if $(t_x \leq t_c)$: $t = t_x$
    if $(t_x > t_c)$: $t = 2t_c$
  - sleep never:
    $t = t_x$

→ optimal:
if $(t_x \leq t_c)$: $t = t_x$ - spin
if $(t_x > t_c)$: $t = t_c$ - sleep immediately

⇒ "sleep after t_c" is 2-competitive!

Slide 53

	add	remove	contains
add	C	C	missed update
remove	C	C	C
contains	missed update	C	no conflict