Esher armonia musica
contrappunto → Buck
2 temi: "La Bouree"

libro
Gödel — Fermata
Esher
Buck
titolo → "un eterna ghirlanda brillante"

atomicità → giovedì | venerdì
lab | concorrente

proprietà delle sezioni critiche

consenso mutua esclusione — race
assenza deadlock — bloccare
assenza delay — sempre la prima (tutti)
assenza starvation

software
complicate

hardware può darci una mano
+ veloci eff
errori

paradigmi fatti: apposta x prog. concorrente
Dijkstra — Dekker
tentativi 1 2 3 4
turni → ¬(¬ delay)
dico "mia" → mutua esclusione
aspettiamo → deadlock
crea → starvation

Dekker dichiaro di averne bisogno
se entrambi ne hanno bisogno
facciamo a turni

Dimostrazione x assurdo
x dimostrare che viene
rispettata la mutua esclusione
Dimostrazione x assurdo
x dimostrare che viene
rispettata assenza di deadlock

Dekker algoritmo 1975

1981 Peterson tricky
dichiara che ne ha bisogno
poi "fa un atto di cortesia"
rompere la simmetria

bello ok conciso ma basta
invertire due statement che salta tutto

Peterson - Generalizzazione a N

processi fanno a gara
loop esterno gara
loop interno partite

problema sol software → basate su
busy wait

Disabilitazione Interrupt

disable "disturbo"
enable

Funzionalità processore
mono processore ok
ma oggi usiamo multiprocessore
quindi non lo possiamo usare

Banca Saldo pericolosissimo
"mode utente"

"solo nel Kernel"
ordine spostata ready Queue
strutture dati Kernel
servizi basati su Kernel ma
su multiprocessore non funziona

/lecture_examples/2023 0929/
dekker_peterson_race
dekker1.c dekker2.c dekker3.c
dekker4.c
dekker.c
peterson.c peterson_multi.c
Test & Set
TS(x,y) = <y=x; x=1>

assegnamenti atomici + 1 istruzione
lock vp
globale locale

TS(lock, vp) = <vp = lock; lock = 1>

se lock = 0 posso entrare
in critical section

se lock = 1 rimetto lock a 1.
se uno non ha bisogno

Achille e la Tartaruga
P Q

non é vero che é sempre rispettata "no starvation"

Tartaruga rimane al palo

test & set fetch & set compare & swap etc

RISC istruzione non implementabile x mancanza di:

" load-link & store-conditional "

seconda annullata e bisogna rifare spinlock

Test & set scala tranquillamente + lock x diverse sez. critiche

c'é ancora busy wait

slide 93 Riassumendo

paradigmi concorrenti

+Facili x programmatore

implemen

---

teatralizzazione Test & Set

vp locale automatica

↳ nello stack ↗

TS(lock, vp) = <vp = lock; lock = 1>

P	lock	Q
vp = 01	0 1 1 0 1 0 1 0	vp = 1000
testi d'esame		

si può usare ... per fare C.S. ?

↓ un'altra istruzione atomica

http://en.wikipedia.org/wiki/Load-link/store-conditional

Semafori Semaphores

Americano: Traffic light

Dijkstra 1965

Dekker 1965

peterson 1981

olandese < Verhogen

(rosso ↓ verde) Proberen

un amico D: "l'olandese è un disturbo della gola"

costruisci semaforo, P(), V()

```
class Semaphore {
    semaphore (int v);
    void P(void);
    void V(void);
};
```

nP nV "successo"

init + nV − nP ≥ 0

valore del semaforo

nP ≤ init + nV

Semafori

```
Semaphore S(1);
process P {
    while (true) {
        S.P();
        CRITICAL SECTION
        S.V();
        non-critical section
    }
}
```

FIFO / fair → by def. of Semaphore

Documento:
"Decalogo di Prog. Concorrente"
principi di buon senso :D

Implementati in qualche modo
⇒ serve TS x implementare
tramite basso impl. alto liv.

variabili "miracolose"

parametri Formali
f(g(x))

appelli

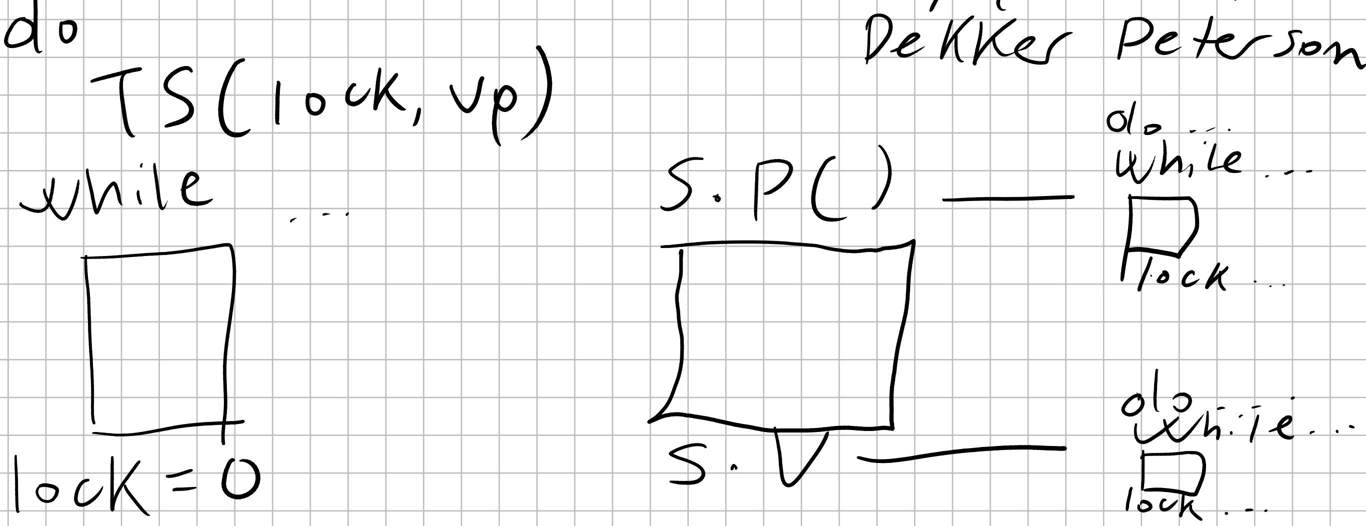
Funzione semaforo (P o V)

[enter (S)]

// codice funzione

...

[exit (S)] → usando TS, spin-lock, ... Dekker Peterson



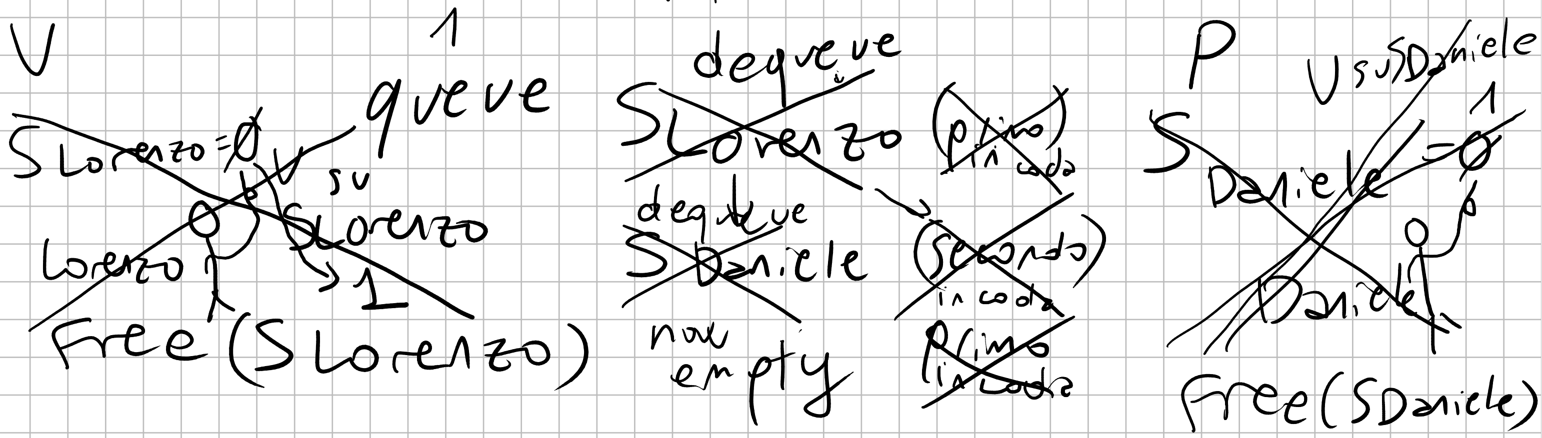
busy wait limitato alle funzioni di P e di V. → p. 106

allocazione dinamica

<u>teatralizzazione</u> impl. semafori tramite BinSem

puntatore laser o mouse

mutex ~~1~~ ~~0~~ ~~1~~ ~~0~~ ~~1~~ ~~0~~ value ~~0~~ ~~1~~ 0
~~1~~ 0 1 0 1 0
1



Implementazione Sem Binari tramite Generali

2 sem. Generali

$S0 = V$

$S1 = 1 - V$

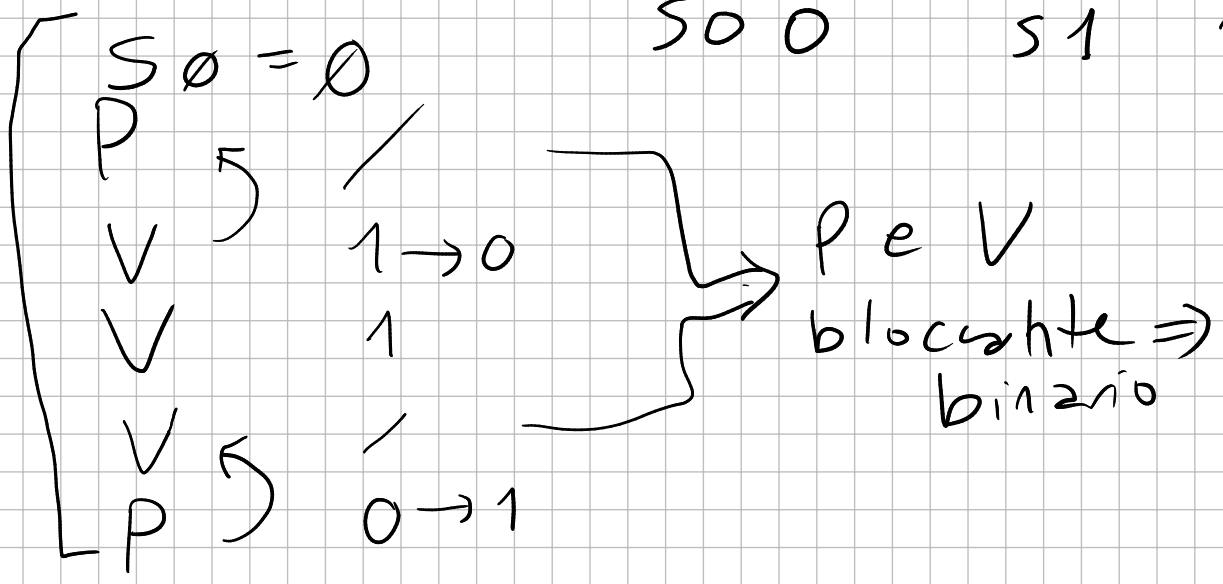
inizializzazione

```
P(){
    S0.P()
    S1.V()
}
```

```
Q(){
    S1.P()
    S0.V()
}
```

supponendo $V = 0$

S0 0 S1 1



Stesso potere espressivo

venerdì prob. campione

x prog. utente

---

gruppi