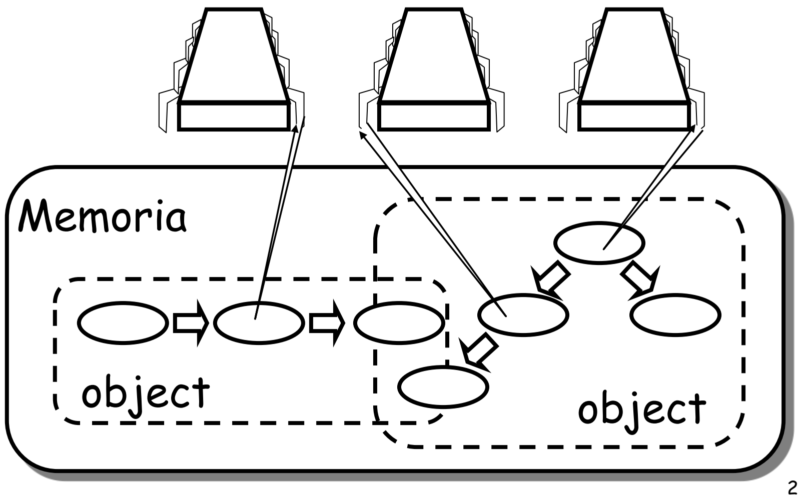
# Concurrent Objects

1

# La concorrenza



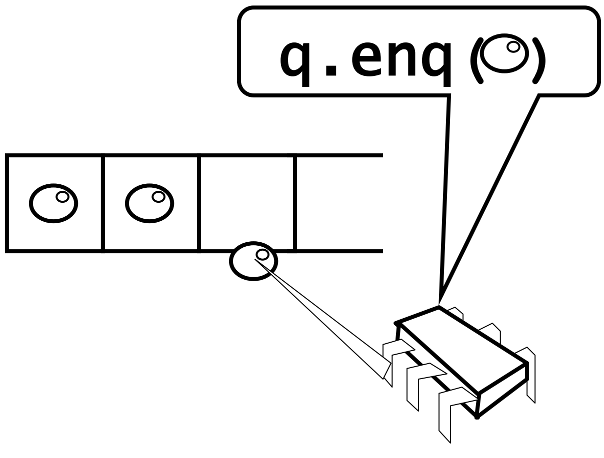

2

# Il problema che vogliamo affrontare

- Cosa e' un oggetto concorrente?
  - In quale modo lo **descriviamo?**
  - In quale modo lo **implementiamo**?
  - In quale modo dimostriamo **la correttezza**?

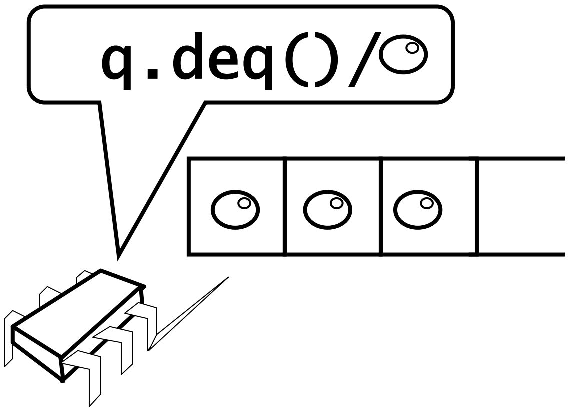
3

# FIFO Queue: Enqueue Method




4

# FIFO Queue: Dequeue Method



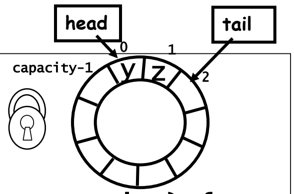

5

# Lock-Based Queue

```
class LockBasedQueue<T> {
  int head, tail;
  T[] items;
  Lock lock;
  public LockBasedQueue(int capacity) {
    head = 0; tail = 0;
    lock = new ReentrantLock();
    items = (T[]) new Object[capacity];
}
```

6

# Lock-Based Queue

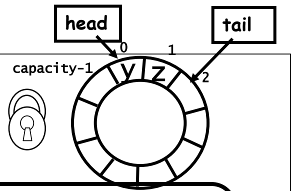



```
class LockBasedQueue<T> {
  int head, tail;
  T[] items;
  Lock lock;
  public LockBasedQueue(int capacity) {
    head = 0; tail = 0;
    lock = new ReentrantLock();
    items = (T[]) new Object[capacity];
}
```

Lock sull'oggetto



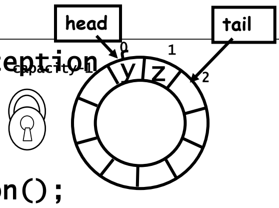
# Lock-Based Queue




```
class LockBasedQueue<T> {
  int head, tail;
  T[] items;
  Lock lock;
  public LockBasedQueue(int capacity) {
    head = 0; tail = 0;
    lock = new ReentrantLock();
    items = (T[]) new Object[capacity];
}
```

Condizione Iniziale
head = tail

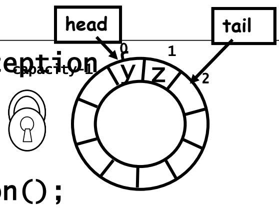
# Deq




```
public T deq() throws EmptyException {
  lock.lock();
  try {
    if (tail == head)
       throw new EmptyException();
    T x = items[head % items.length];
    head++;
    return x;
  } finally {
    lock.unlock();
  }
}
```

9

# Deq

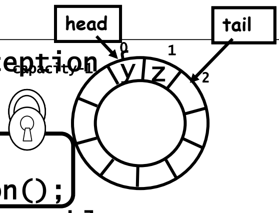



```
public T deq() throws EmptyException {
  lock.lock();
  try {
    if (tail == head)
       throw new EmptyException();
    T x = items[head % items.length];
    head++;
    return x;
  } finally {
    lock.unlock();
  }
}
```

Metodi sincronizzati

10

# Deq

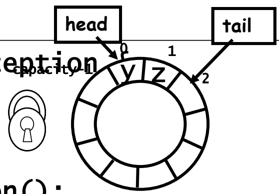



```
public T deq() throws EmptyException {
  lock.lock();
  try {
    if (tail == head)
       throw new EmptyException();
    T x = items[head % items.length];
    head++;
    return x;
  } finally {
    lock.unlock();
  }
}
```

If queue empty throw exception

11

# Deq

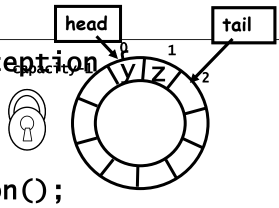



```
public T deq() throws EmptyException {
  lock.lock();
  try {
    if (tail == head)
       throw new EmptyException();
    T x = items[head % items.length];
    head++;
    return x;
  } finally {
    lock.unlock();
  }
}
```

Queue ! empty: Rimozione e aggiornamento

12

# Deq

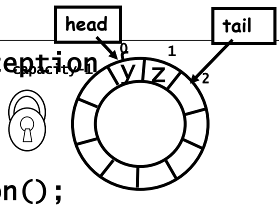



```
public T deq() throws EmptyException {
  lock.lock();
  try {
    if (tail == head)
       throw new EmptyException();
    T x = items[head % items.length];
    head++;
    return x;
  } finally {
    lock.unlock();
  }
}
```

Return risultato



# Deq




```
public T deq() throws EmptyException {
  lock.lock();
  try {
    if (tail == head)
       throw new EmptyException();
    T x = items[head % items.length];
    head++;
    return x;
  } finally {
    lock.unlock();
  }
}
```

Release lock !!

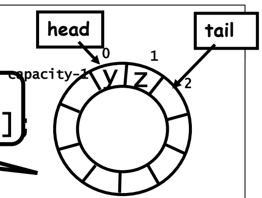
# Deq

```
public T deq() throws EmptyException {
  lock.lock();
  try {
    if (tail == head)
       throw new EmptyException();
    T x = items[head % items.length];
    head++;
    return x;
  } finally {
    lock.unlock();
  }
}
```

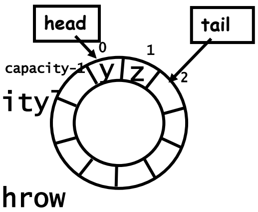
E' corretta? Le modifiche avvengono in mutua esclusione…

15

```
public class WaitFreeQueue {

  int head = 0, tail = 0;
  items = (T[]) new Object[capacity];

  public void enq(Item x) {
    if (tail-head == capacity) throw
         new FullException();
    items[tail % capacity] = x; tail++;
  }
  public Item deq() {
     if (tail == head) throw
         new EmptyException();
     Item item = items[head % capacity]; head++;
     return item;
}}
```




```
public class WaitFreeQueue {

  int head = 0, tail = 0;
  items = (T[]) new Object[capacity];

  public void enq(Item x) {
    if (tail-head == capacity) throw
         new FullException();
    items[tail % capacity] = x; tail++;
  }
  public Item deq() {
     if (tail == head) throw
         new EmptyException();
     Item item = items[head % capacity]; head++;
     return item;
}}
```




```
public class WaitFreeQueue {

  int head = 0, tail = 0;
  items = (T[]) new Object[capacity];

  public void enq(Item x) {
    if (tail-head == capacity) throw
         new FullException();
    items[tail % capacity] = x; tail++;
  }
  public Item deq() {
     if (tail == head) throw
         new EmptyExcept...
     Queue is update...ck!  ...++;
     return item;
}}
```

Le modifiche non sono mutuamente esclusive

18

# Progresso

- In un contesto concorrente si deve specificare sia le proprieta' di invarianza (safety) che le proprieta' di liveness
- Correttezza
  - Quando una implementazione e' corretta
  - Le condizioni che garantiscono il progresso

19

# Oggetti sequenziali

- Ogni astrazione ha un proprio ***state***
  - Le variabili di istanza ***fields***
  - Queue: vettore di items
- Ogni astrazione possiede dei metodi ***methods***
  - Descrivono come operare sullo state
  - Queue: enq e deq

20

# Specifica

- (Requires)
  - Prima di invocare un metodo l'oggetto e' nello stato corretto,
- (Effects)
  - Il metodo modifica lo stato correttamente oppure solleva una eccezione.

21

# Dequeue

- Requires:
  - Queue ! empty
- Effects:
  - Returns first item queue
- Effects:
  - Removes first item queue

22

# Sequenziale

- Le interazioni tra i metodi sono catturate tramite effetti laterali sullo stato degli oggetti
  - Invariante di rappresentazione a questo serve!!!
- Ogni metodo e' descritto singolarmente
- Refinement: possiamo aggiungere metodi senza modificare la descrizione dei vecchi metodi.

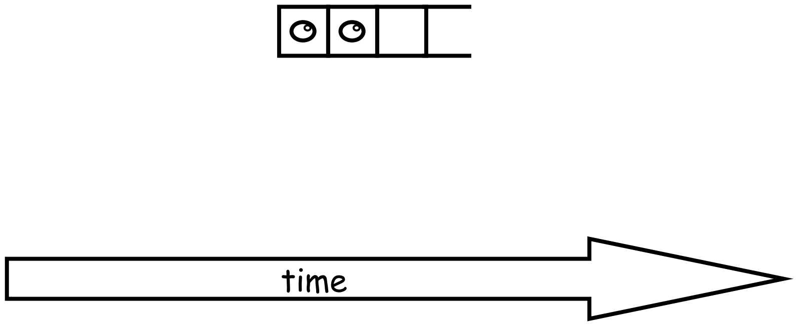
23

# Cosa cambia con la concorrenza?

- Metodi?
- La descrizione del metodo?
- Refinement?

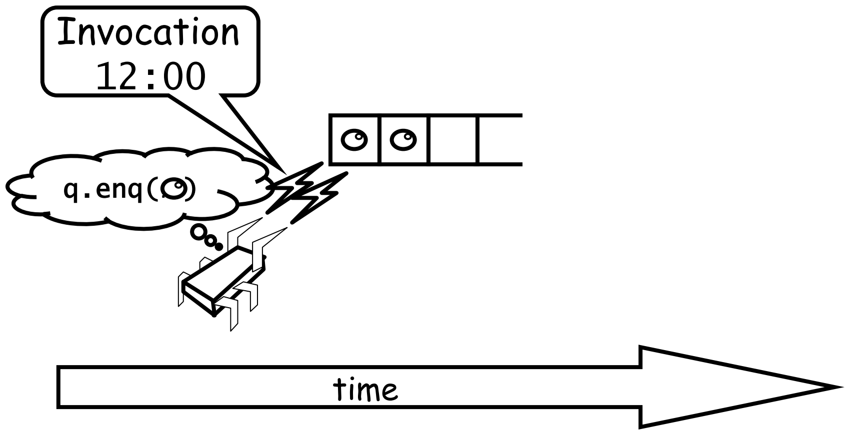
24

# “Methods Take Time”



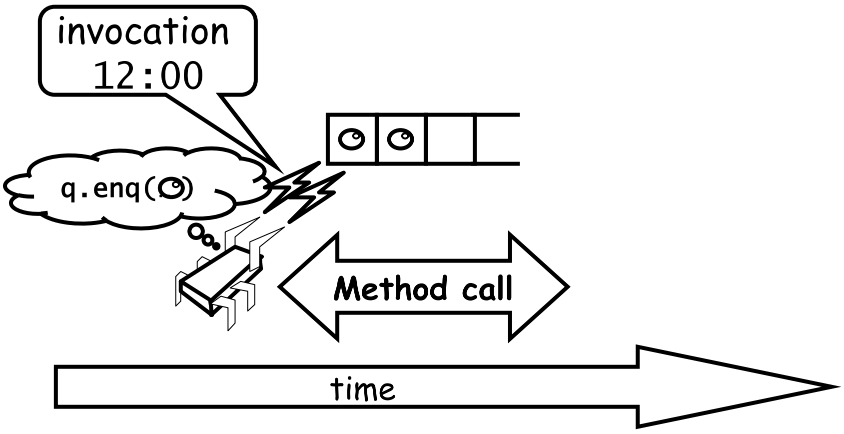

25

# Methods Take Time



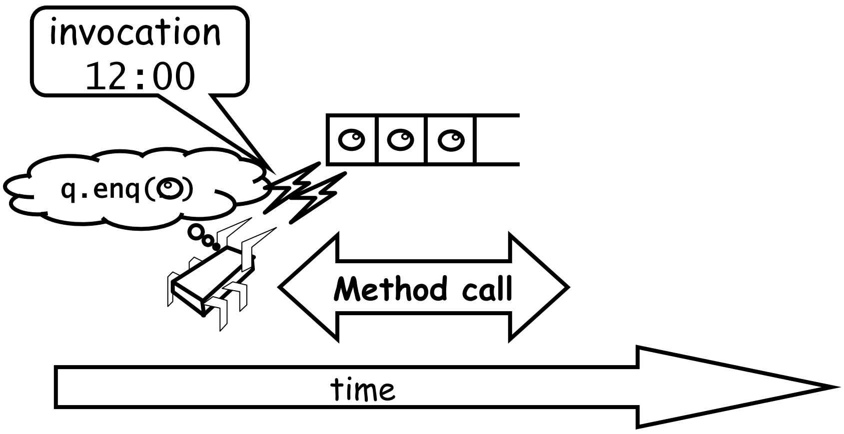

26

# Methods Take Time



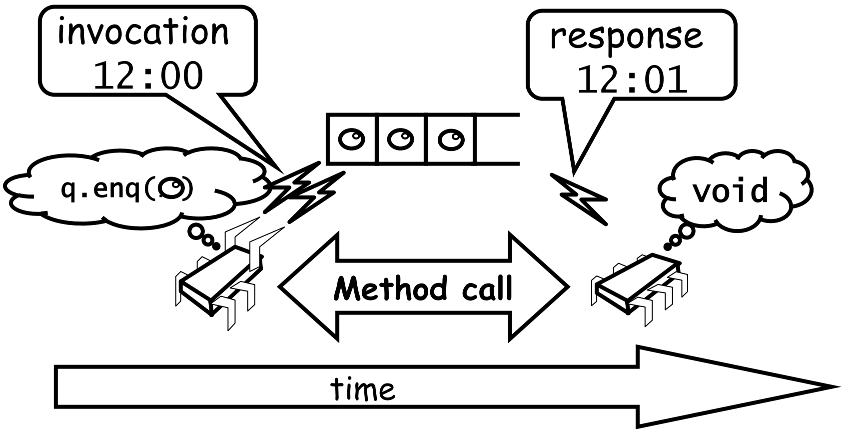

27

# Methods Take Time




28

# Methods Take Time



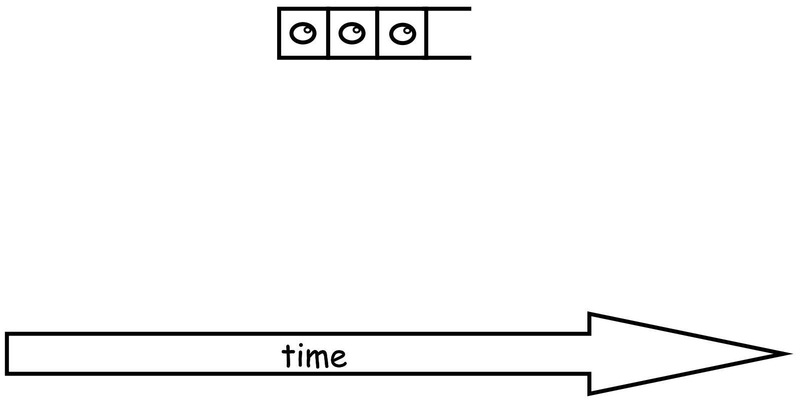

29

# Sequenziale vs Concorrente

- Sequenziale
  - Methods take time?
  - Quando mai!!!
    - Nell'astrazione che abbiamo utilizzato l'esecuzione dei metodi e' istantanea
- Concorrente
  - La chiamata di un metodo e' un intervallo.

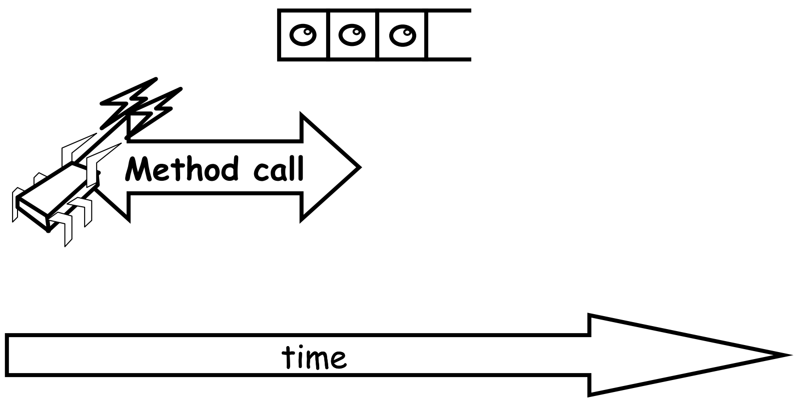
30

# Overlapping Time

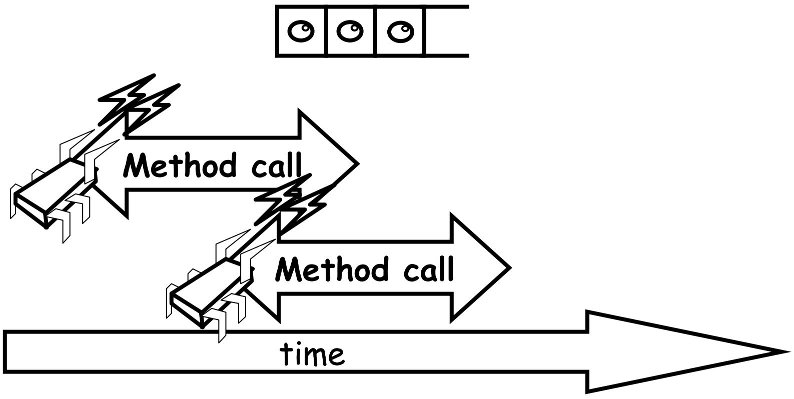





# Overlapping Time

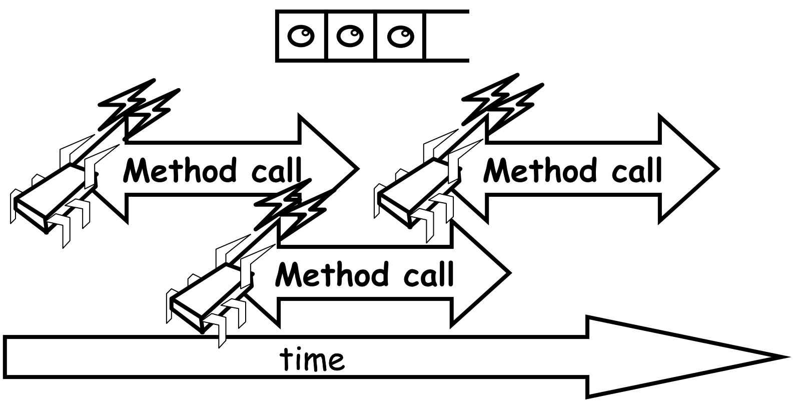


# Overlapping Time




33

# Overlapping Time




34

# Sequenziale vs Concorrente

- Sequenziale:
  - Stato degli oggetti e' significativo solamente ***tra*** le invocazioni dei metodi
    - Invariante di rappresentazione
- Concorrente
  - Dato che le chiamate si sovrappongono la stato di un oggetto potrebbe ***non essere consistente*** tra le invocazioni dei metodo

35

# Sequenziale vs Concorrente

- Sequenziale:
  - Ogni metodo e' descritto in isolamento
- Concorrente
  - Si devono comprendere ***tutte*** le possiibili interazioni con chiamate concorrenti
    - Cosa succede se due invocazioni di enq si sovrappongono?

36

# Sequenziale vs Concorrente

- Sequenziale:
  - Refinement
- Concorrente:
  - Ogni metodo puo' potenzialemnte interagire con tutti gli altri

37

# Sequenziale vs Concorrente

- Sequenziale:
  - Refinement
- Concorrente:
  - Ogni metodo puo'potenzialemnte interagire con tutti gli altri

Panic!

38

# La domanda

- Quale e' la nozione di correttezza nel caso concorrente?
  - FIFO implica ordine temporale stretto
  - Concorrenza implica un ordine temporale non determinato

39

```
public T deq() throws EmptyException {
  lock.lock();
  try {
    if (tail == head)
       throw new EmptyException();
    T x = items[head % items.length];
    head++;
    return x;
  } finally {
    lock.unlock();
  }
}
```

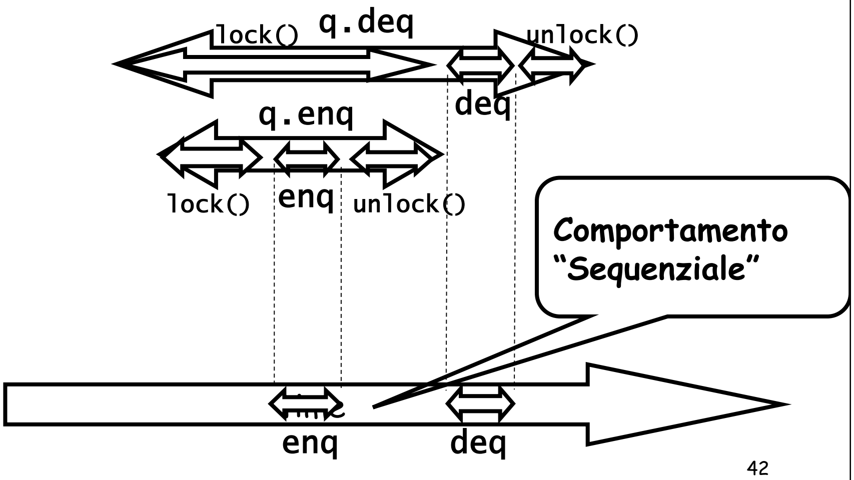
40

```
public T deq() throws EmptyException {
  lock.lock();
  try {
    if (tail == head)
       throw new EmptyException();
    T x = items[head % items.length];
    head++;
    return x;
  } finally {
    lock.unlock();
  }
}
```

Sezione critica

41

**Catturiamo la concorrenza mediante l'ordine Degli eventi**



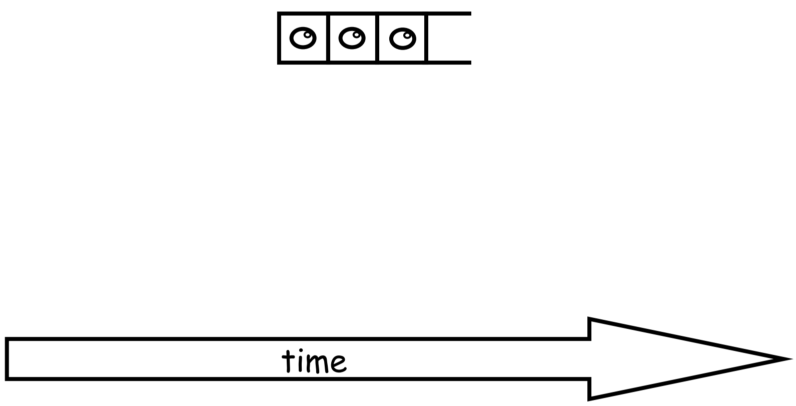

42

# Linearizability

- Un oggetto e' corretto se la sua proiezione sequenziale e' corretta
  - **Linearizable**

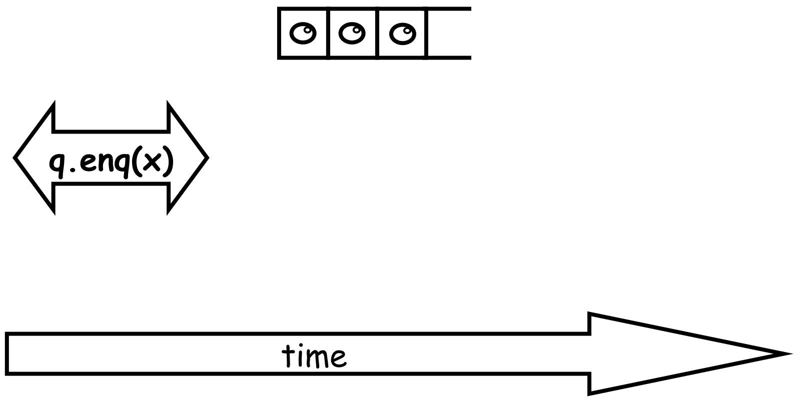
```
Un oggetto e'
linearizable: se tutte
le sue possibili
esecuzioni sono
linearizable
```

43

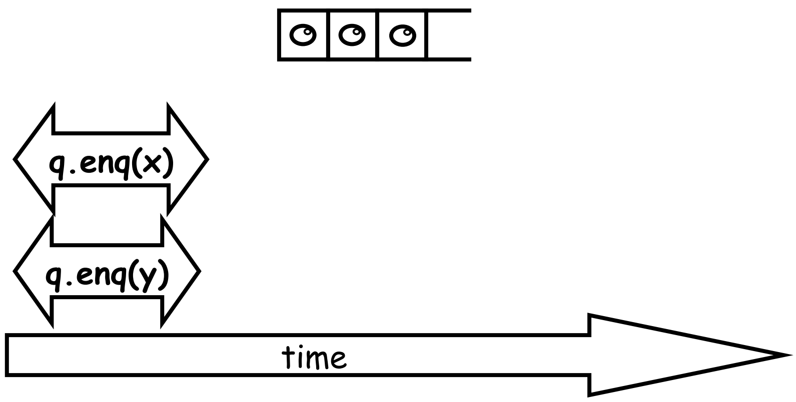



(6)

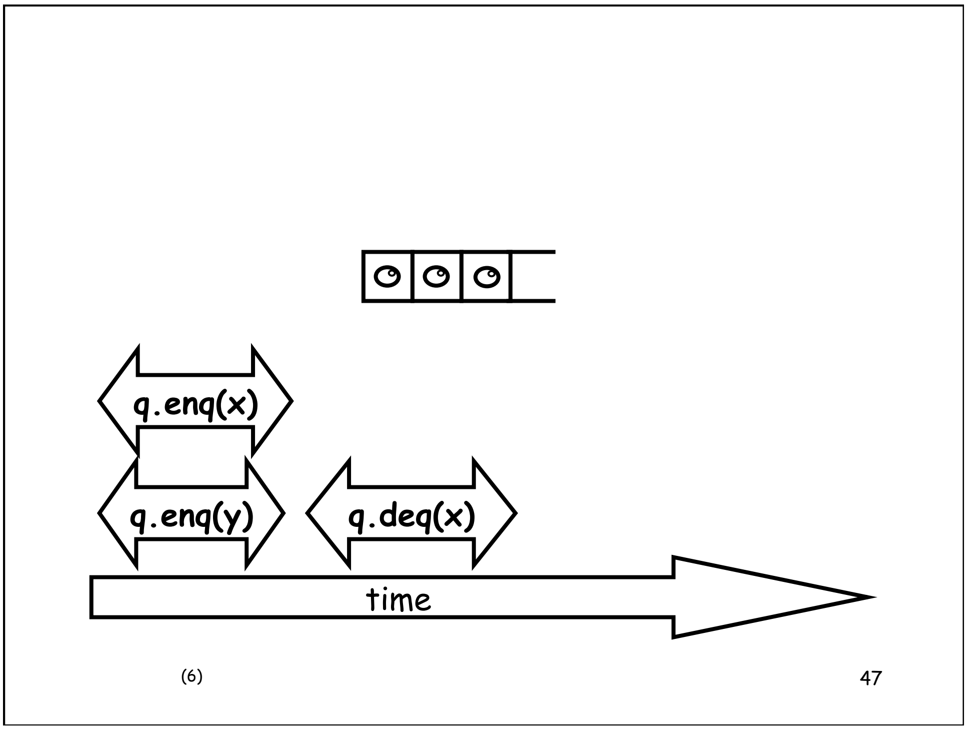
44

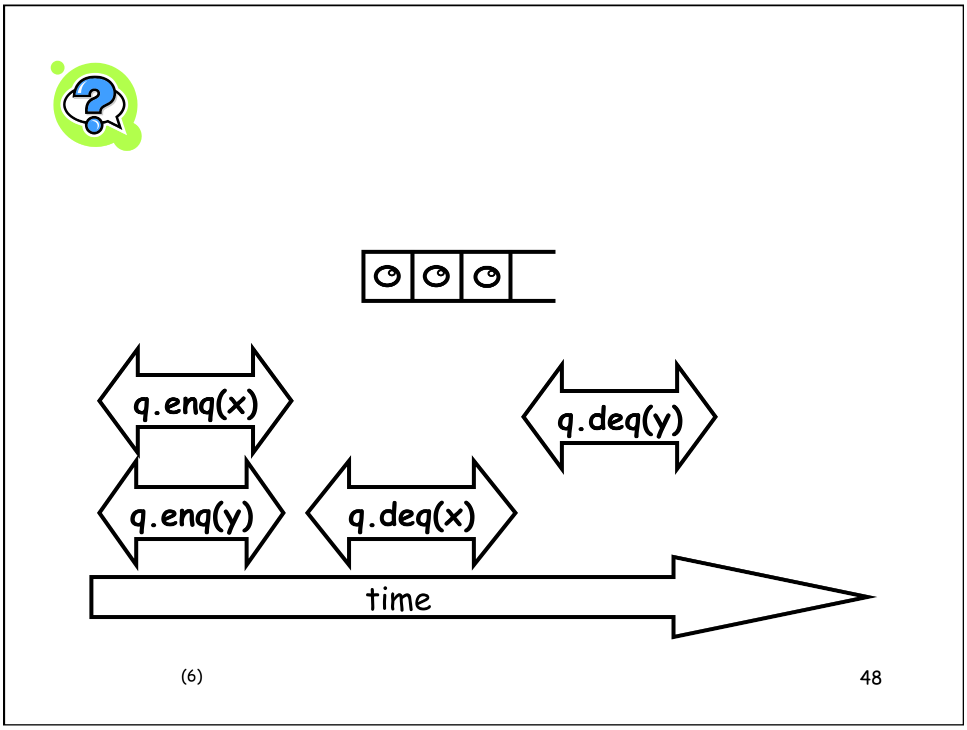


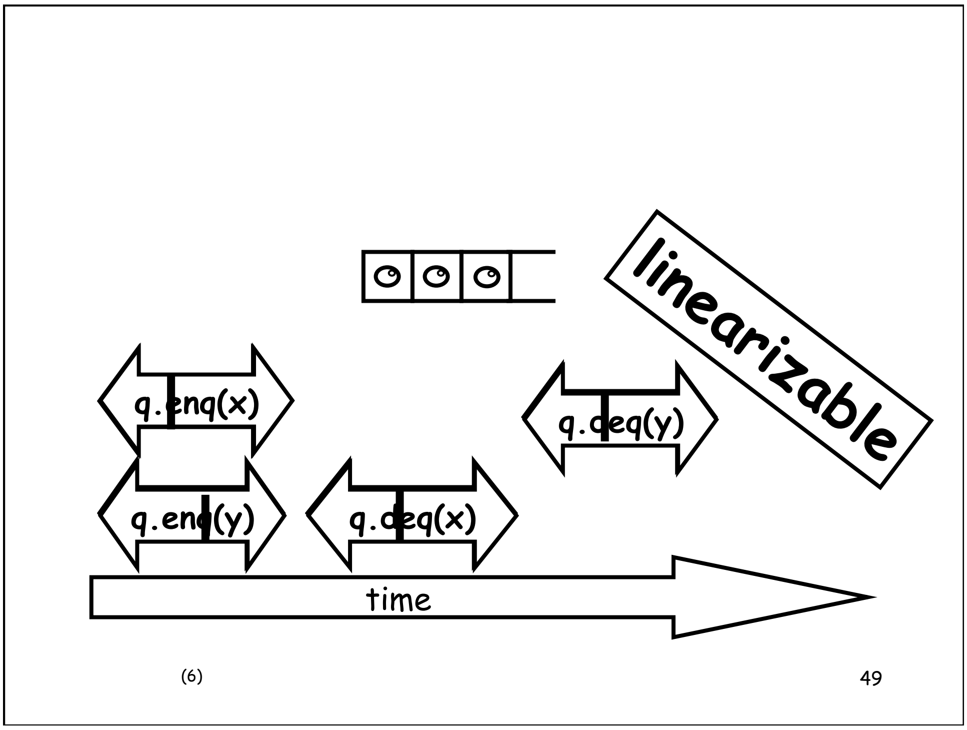

(6) 45



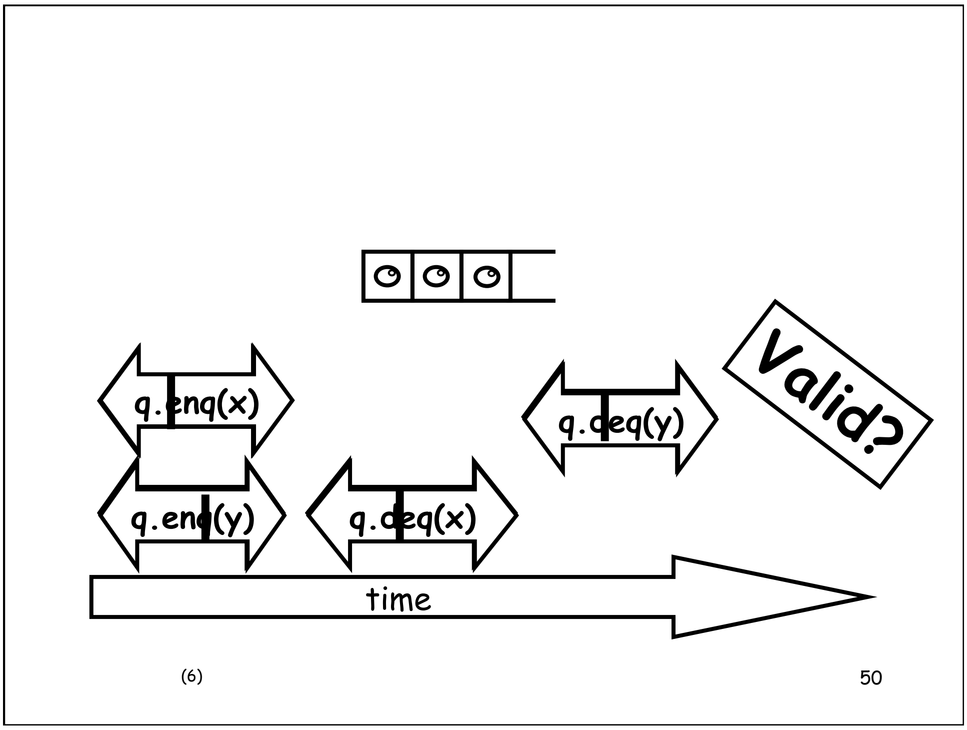

(6) 46


q.enq(x)
q.enq(y)
q.deq(x)
time
(6)
47

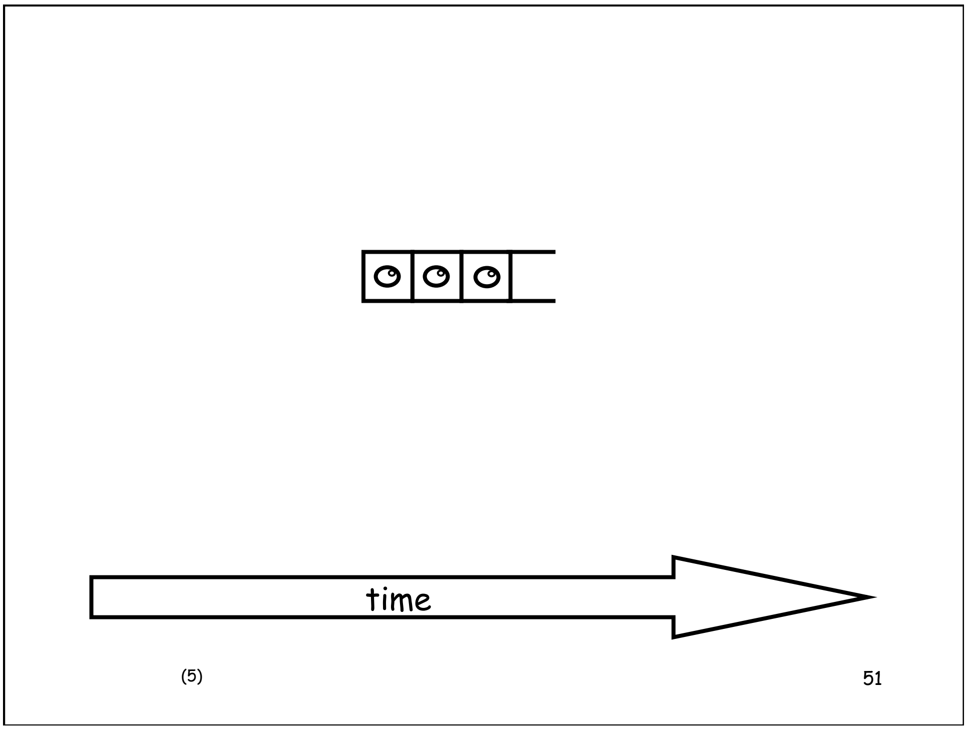

q.enq(x)
q.deq(y)
q.enq(y)
q.deq(x)
time
(6)
48

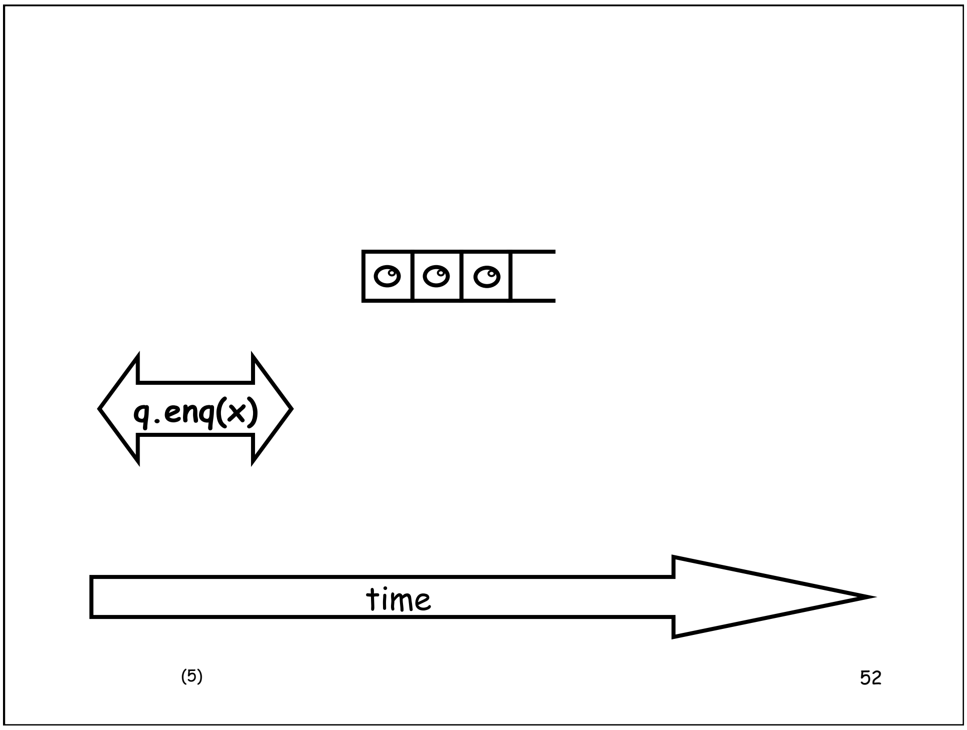


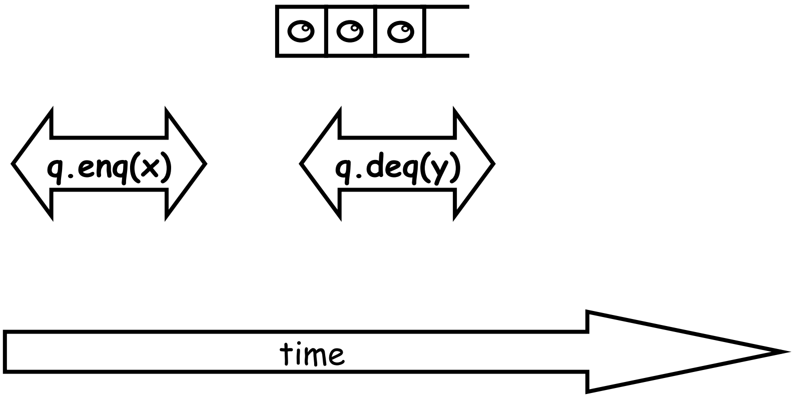
linearizable
q.enq(x)
q.deq(y)
q.enq(y)
q.deq(x)
time
(6)
49

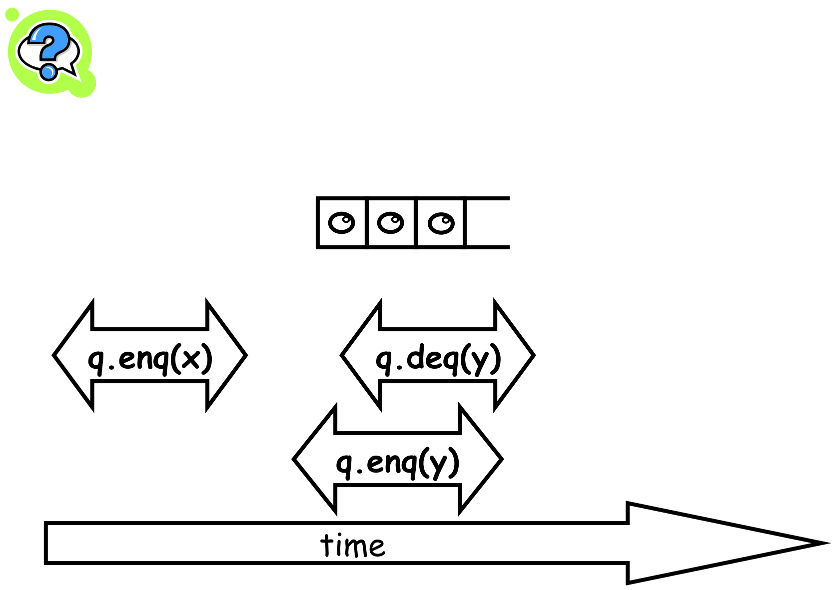
Valid?
q.enq(x)
q.deq(y)
q.enq(y)
q.deq(x)
time
(6)
50


time
(5)
51

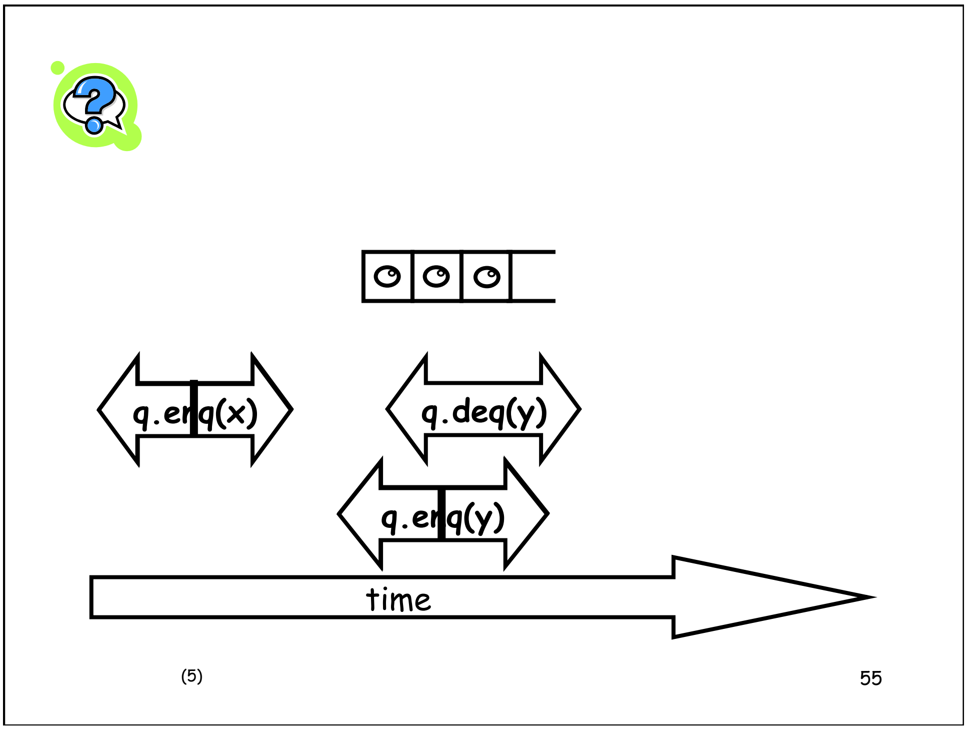

q.enq(x)
time
(5)
52

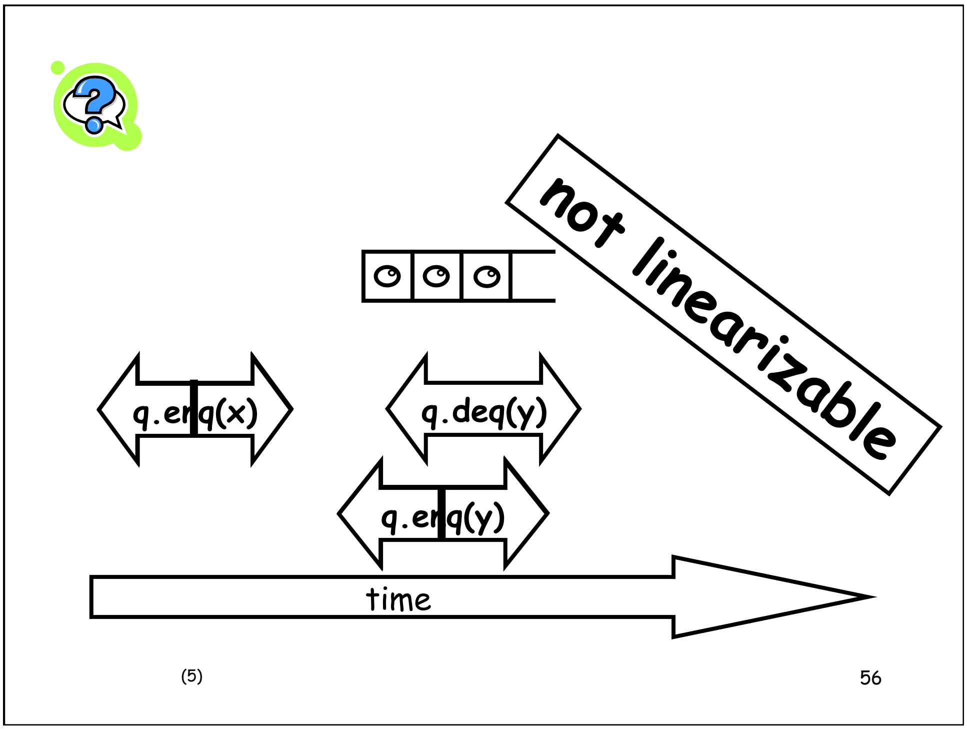


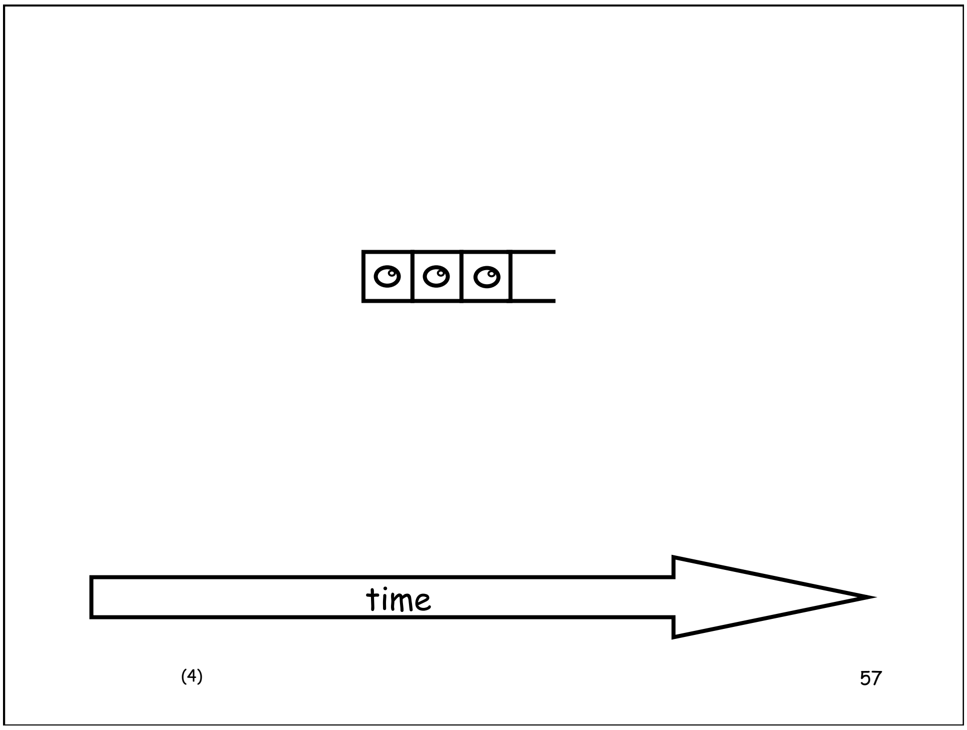

(5) 53

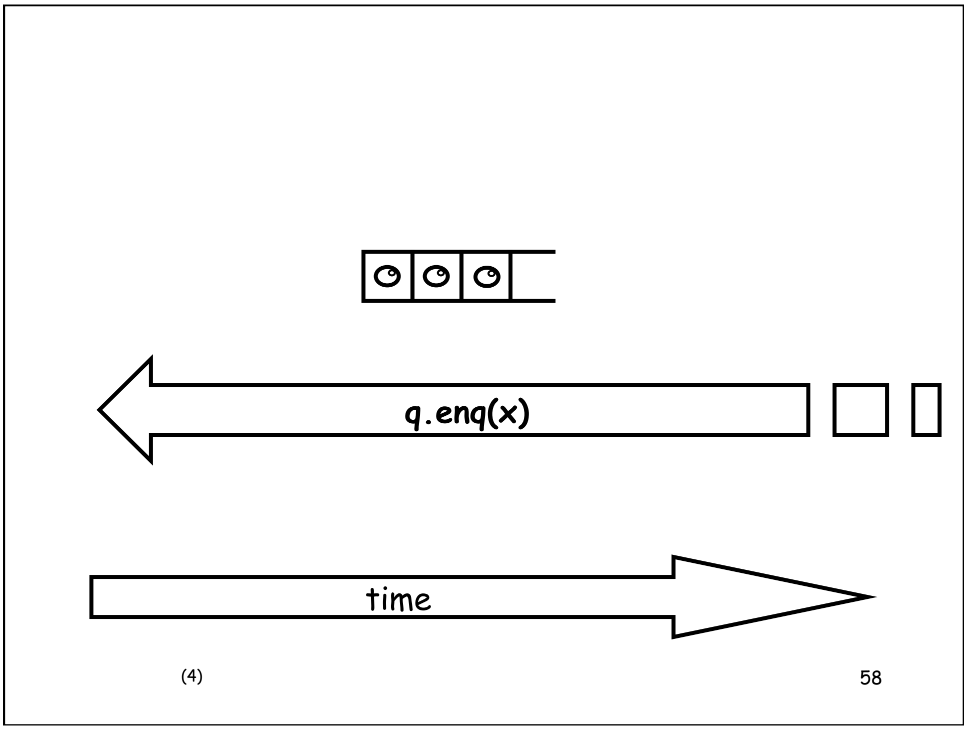


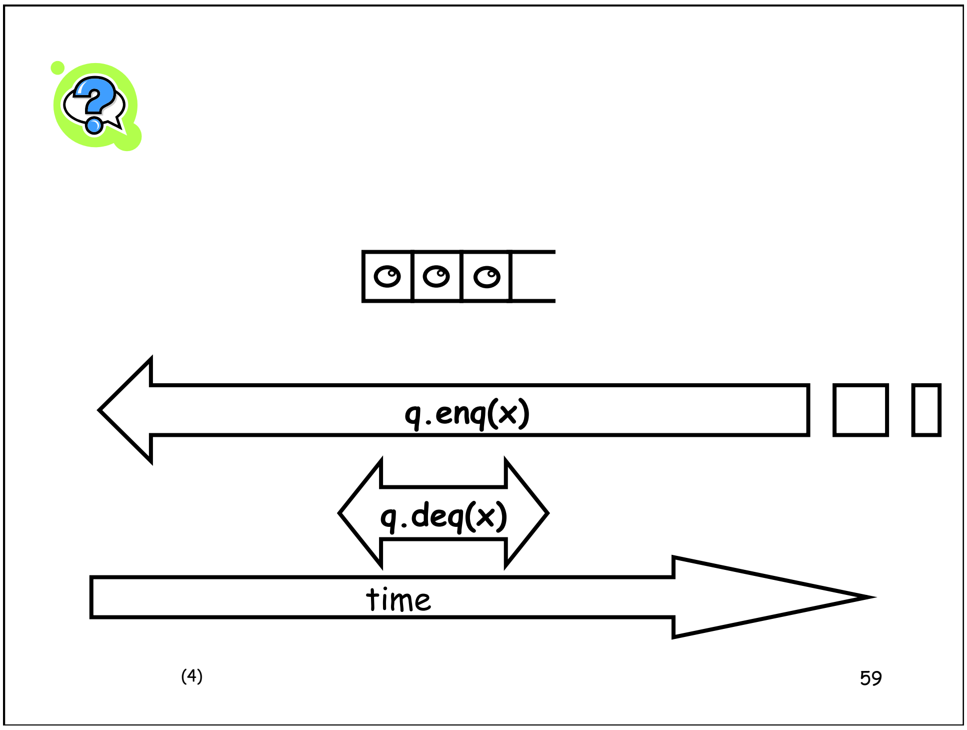

(5) 54

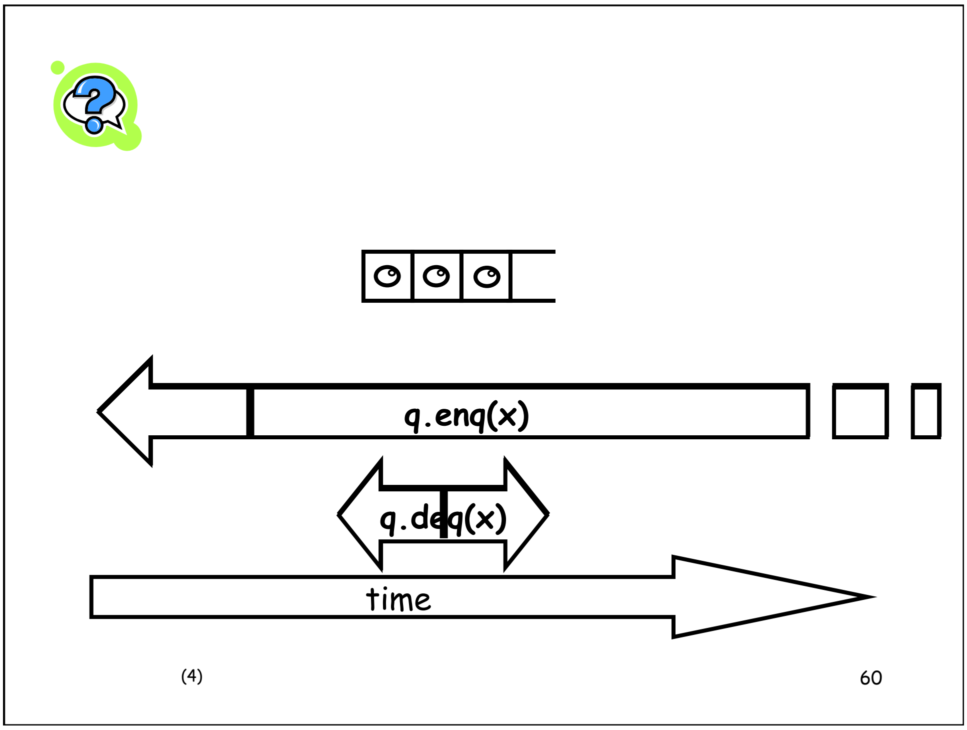

q.enq(x)
q.deq(y)
q.enq(y)
time
(5)
55

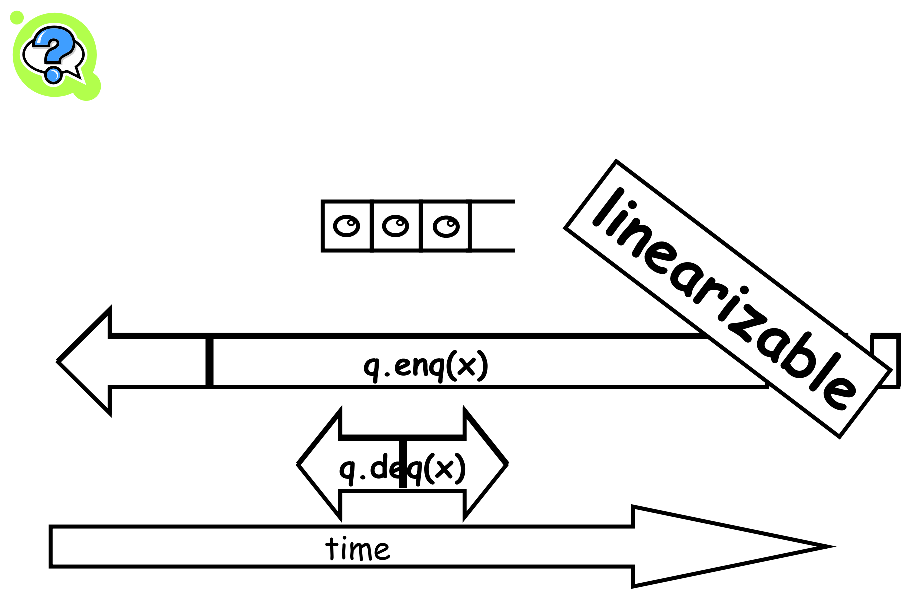

not linearizable
q.enq(x)
q.deq(y)
q.enq(y)
time
(5)
56


time
(4)
57

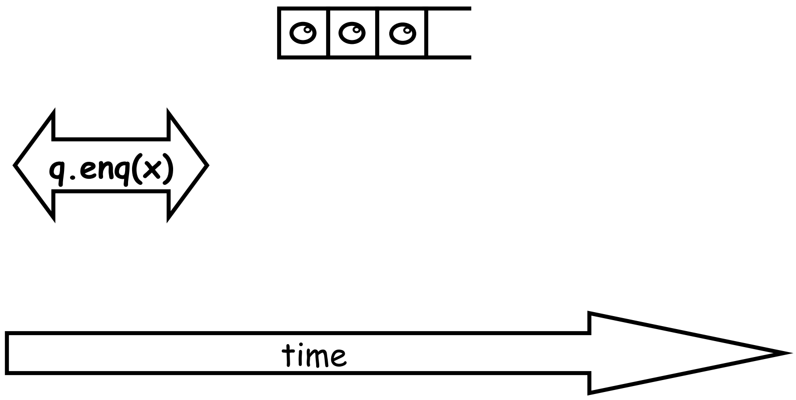

q.enq(x)
time
(4)
58


q.enq(x)
q.deq(x)
time
(4)
59

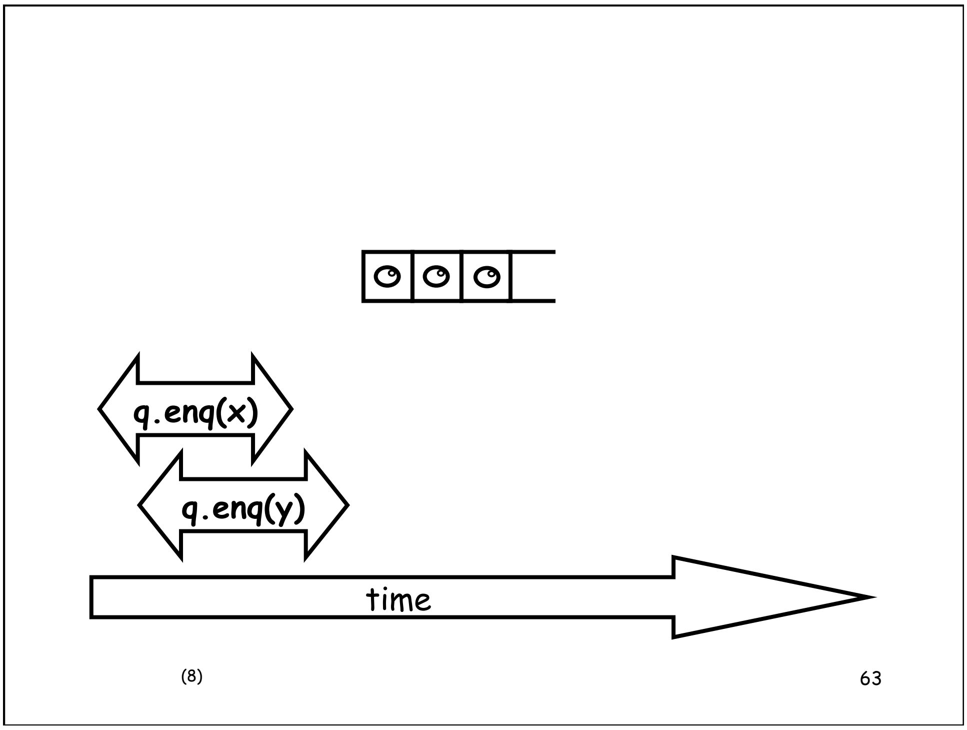

q.enq(x)
q.deq(x)
time
(4)
60

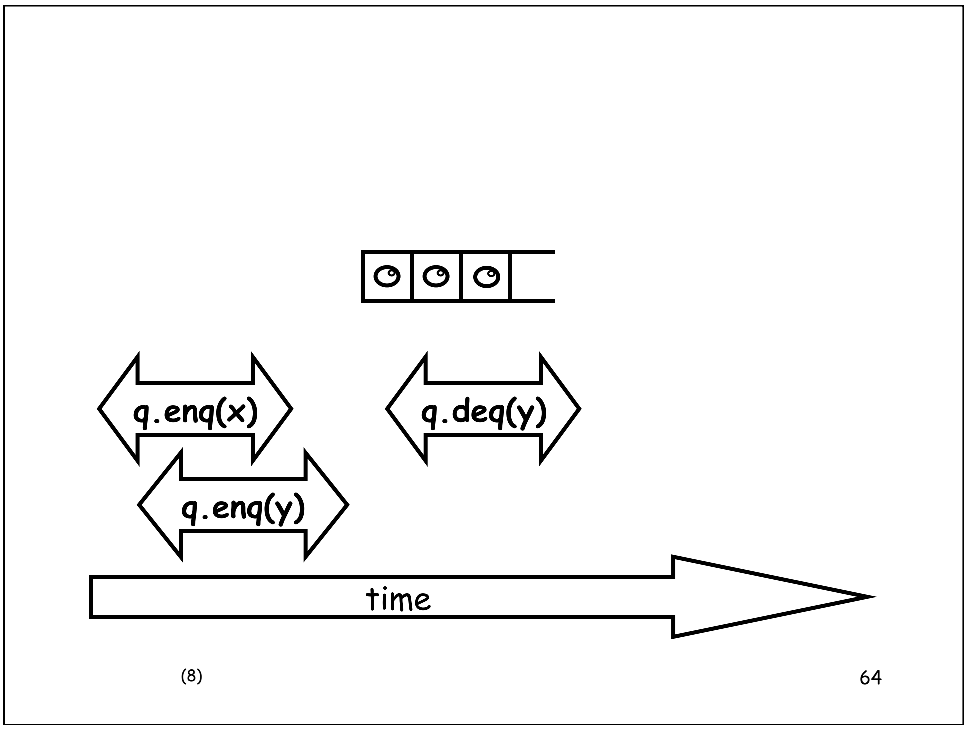


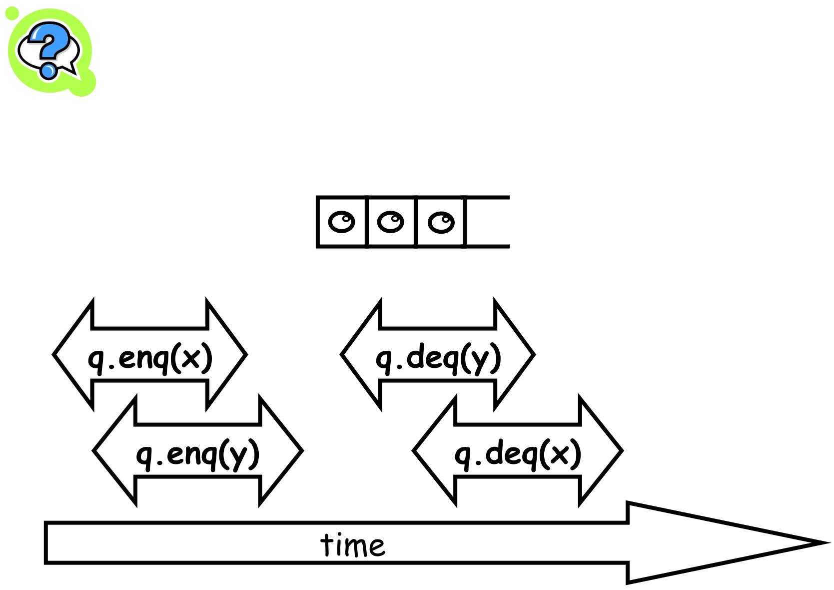

(4) 61




(8) 62


q.enq(x)
q.enq(y)
time
(8)
63

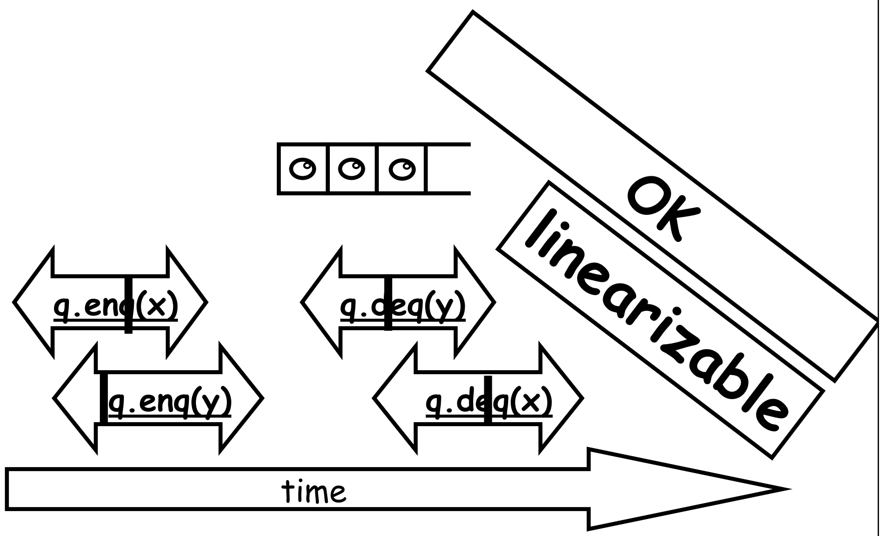

q.enq(x)
q.deq(y)
q.enq(y)
time
(8)
64



q.enq(x)
q.enq(y)
q.deq(y)
q.deq(x)
time


(8)

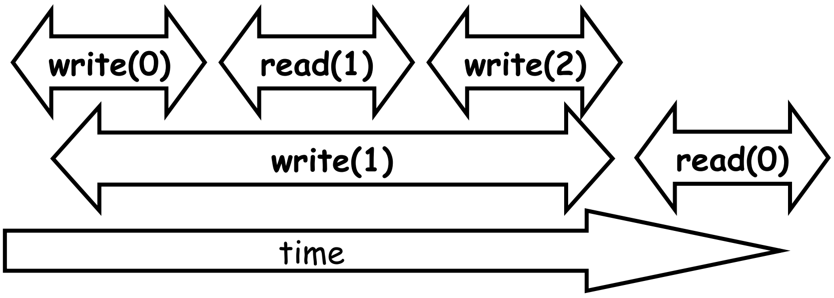
65



q.enq(x)
q.enq(y)
q.deq(y)
q.deq(x)
time
OK
linearizable

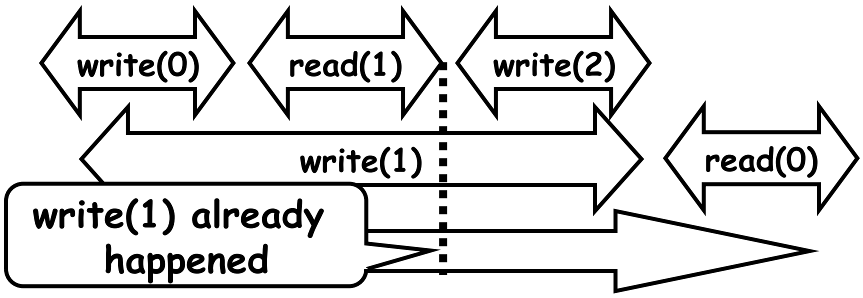

66

# Read/Write Register



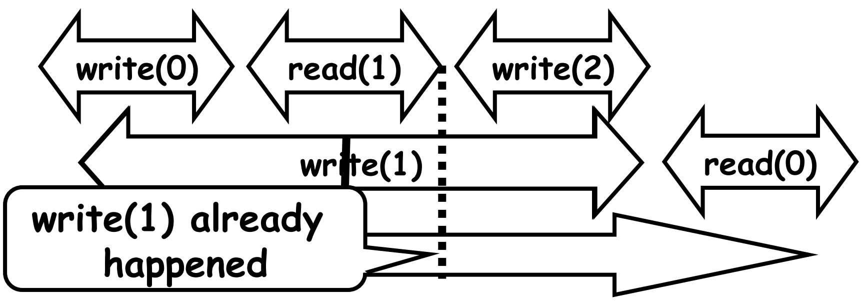

(4) 67

# Read/Write Register



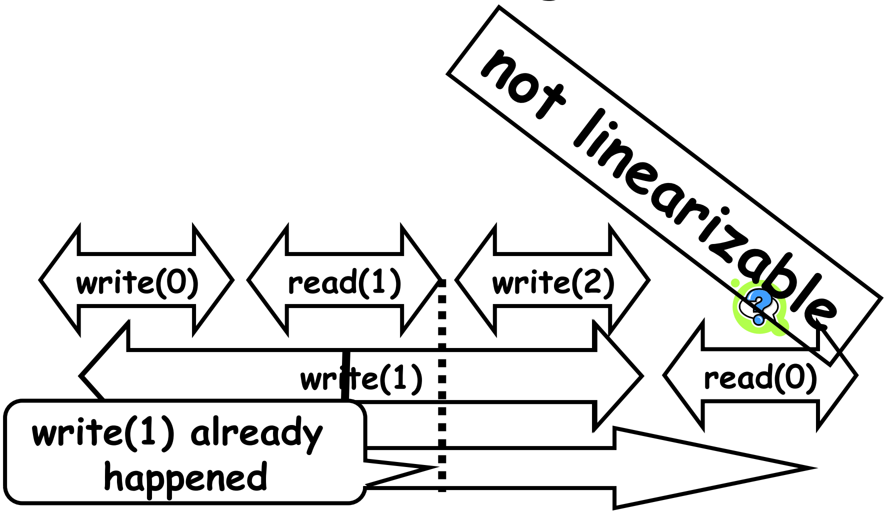

(4) 68

# Read/Write Register



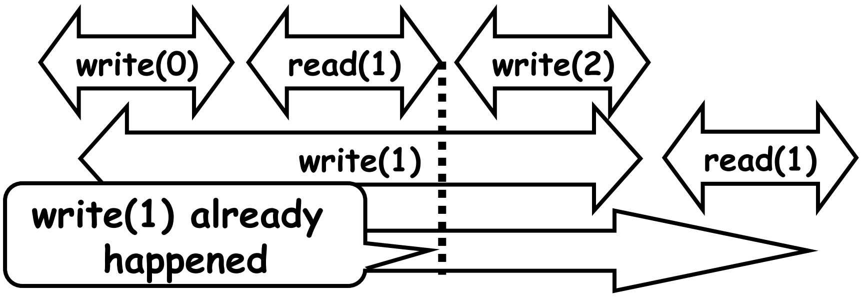

(4) 69

# Read/Write Register




(4) 70

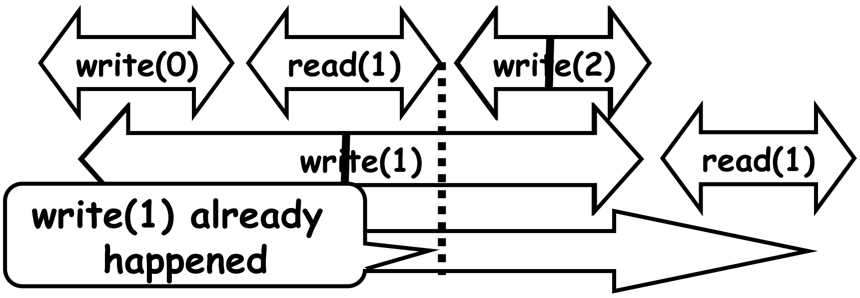
# Read/Write Register




(4)

71

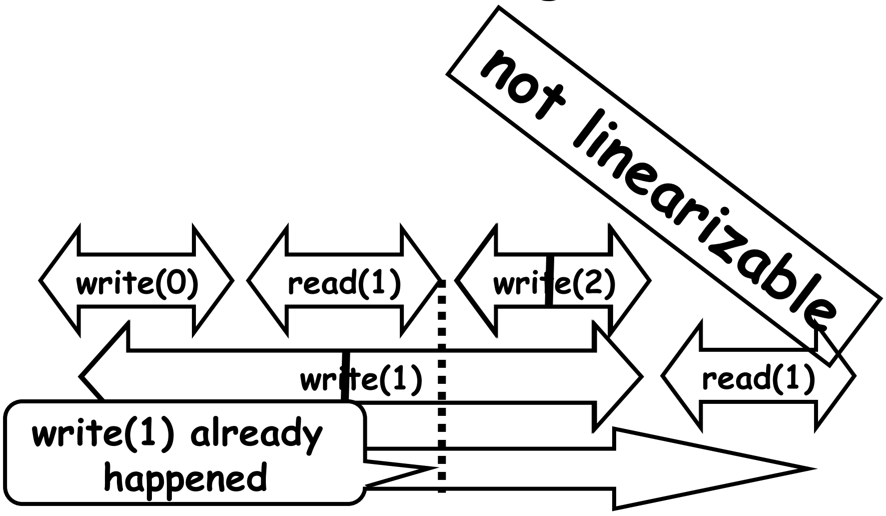
# Read/Write Register




(4)

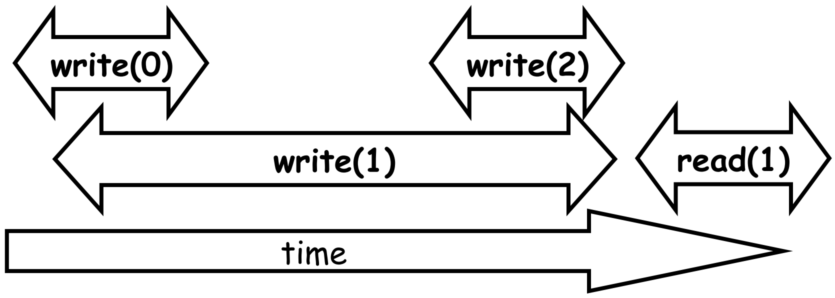
72

# Read/Write Register



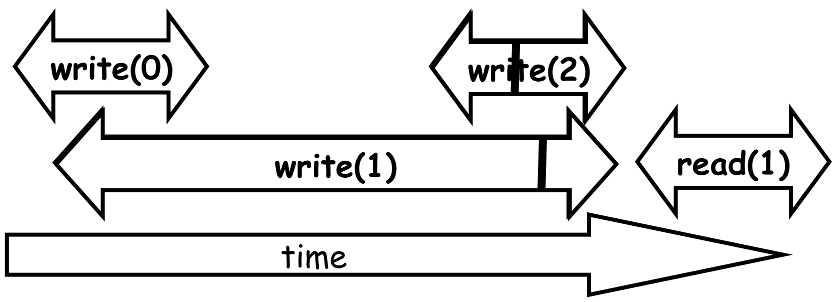

(4) 73

# Read/Write Register



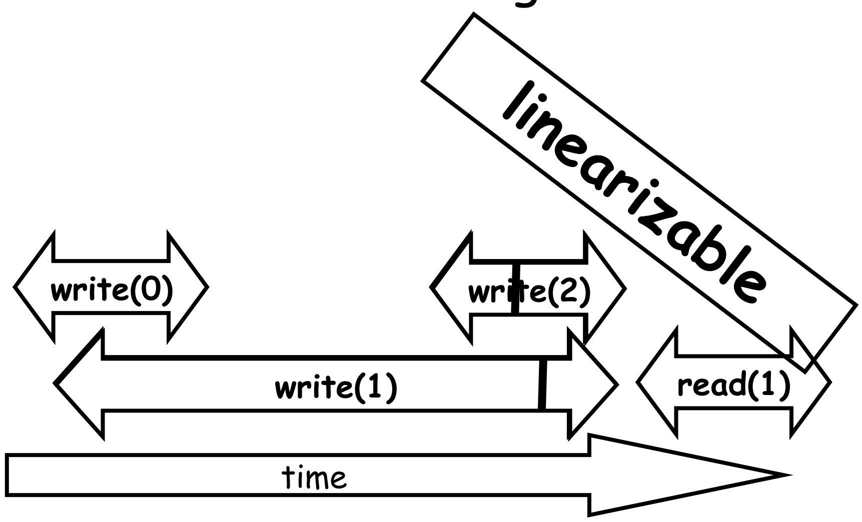

(4) 74

# Read/Write Register



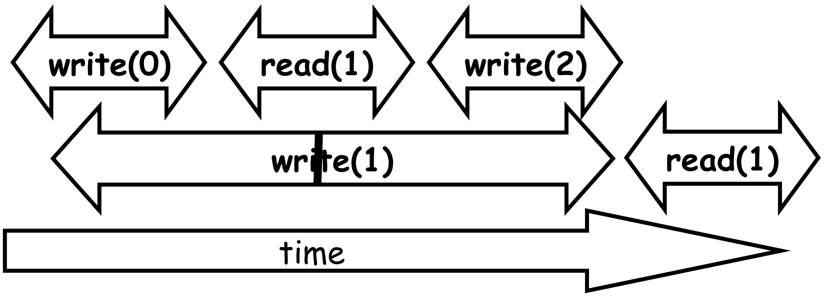

(4) 75

# Read/Write Register



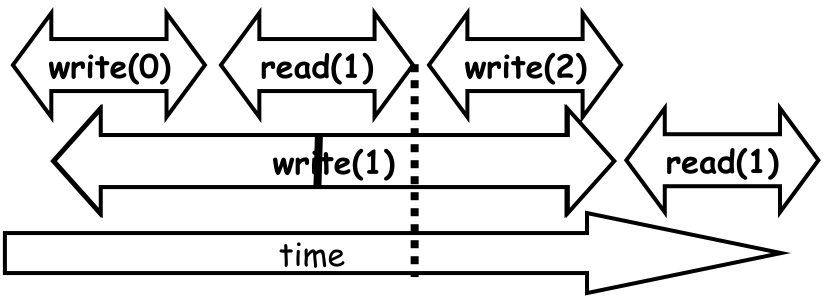

(4) 76

# Read/Write Register



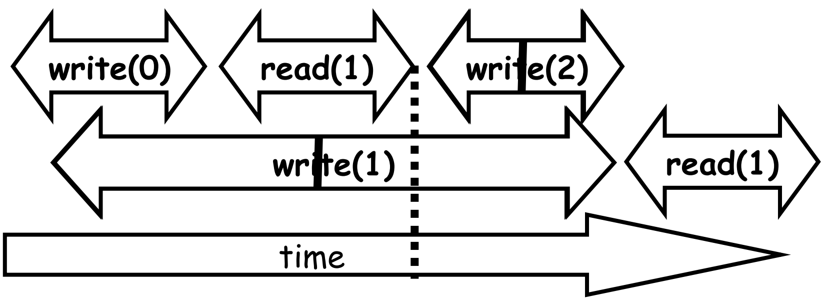

(2) 77

# Read/Write Register



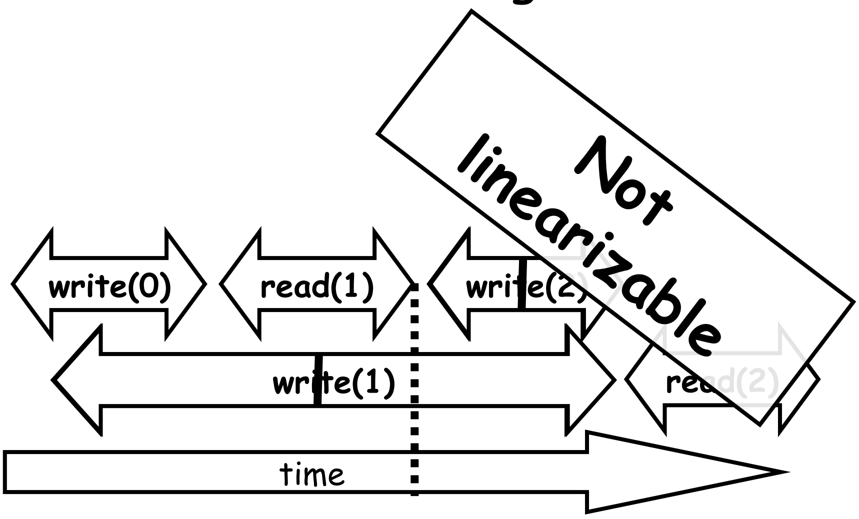

(2) 78

# Read/Write Register




(2) 79

# Read/Write Register




(2) 80

# Esecuzione

- Problema
  - Possiamo specificare i punti di linearizzazione senza dover necessariamente parlare di esecuzione?
- Non e' sempre possibile
  - In alcuni casi i punti di linearizzazione dipendono strettamente dalla particolare esecuzione:
    - Esempio dei registri visto in precedenza

81

# Modello formale

- Definizione precisa dei comportamenti
  - Eliminano i punti di ambiguita'
- Permette di avere tecniche di verifica
  - Formali (esempio model checking)
  - Ma anche informali
    - Ragionare prima di programmare !!!

82

# Tecnica dello split degli eventi

- Chiamata (quasi la precondizione)
  - Nome del metodo & args
  - `q.enq(x)`
- Risultato (quasi la postcondizione)
  - Risultato oppure exception
  - `q.enq(x)` returns `void`
  - `q.deq()` returns `x`
  - `q.deq()` throws `empty`

83

# Notazione

**A q.enq(x)**

(4)

84

# Notazione

**A q.enq(x)**

**thread**

(4) 85

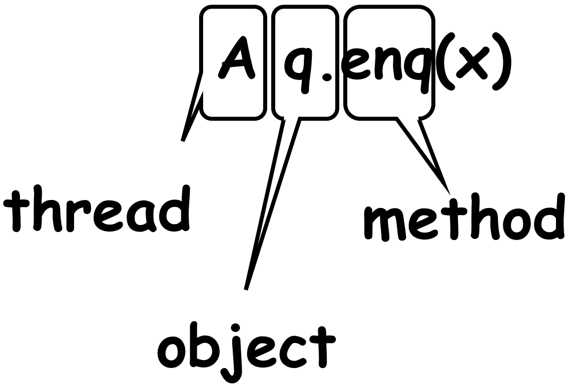
# Notazione

**A q.enq(x)**

**thread** **method**

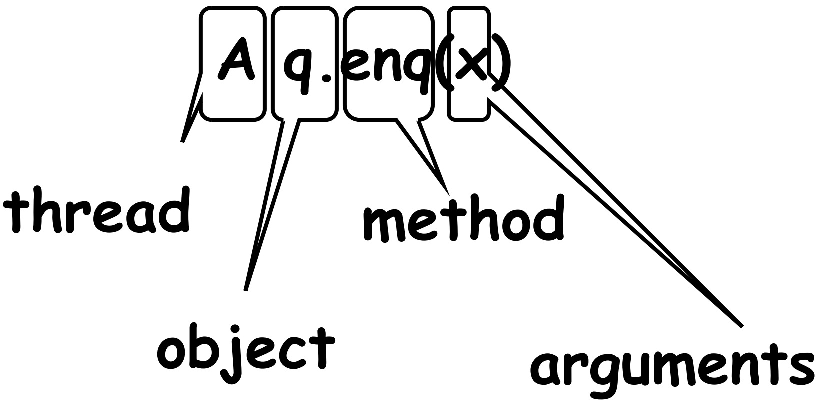
(4) 86

# Notazione




(4) 87

# Notazione




(4) 88

## Risposta: notazione

**A q: void**

(2) 89

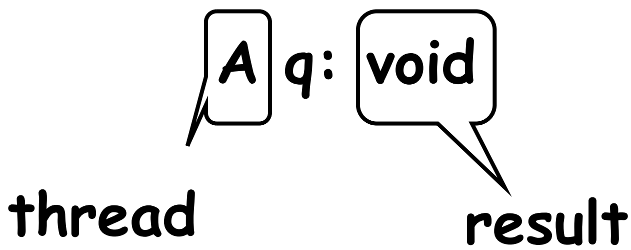
## Notazione

**A q: void**

**thread**

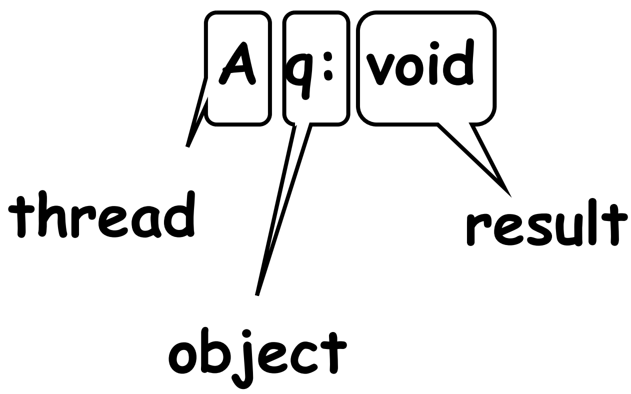
(2) 90

# Notazione

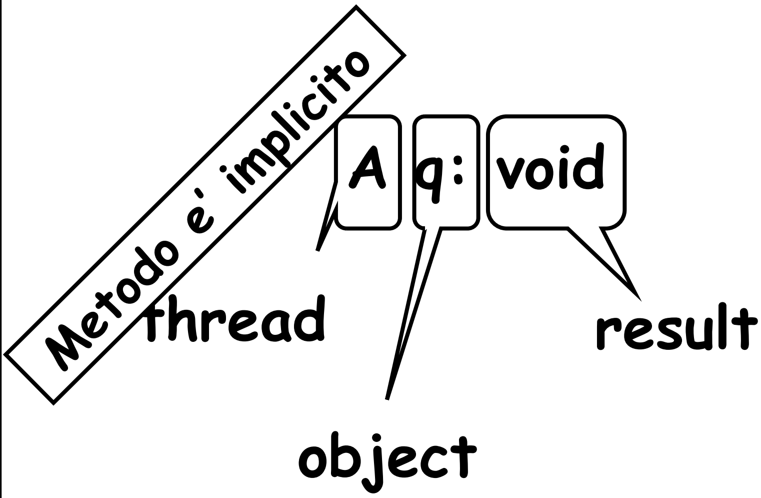





# Notazione

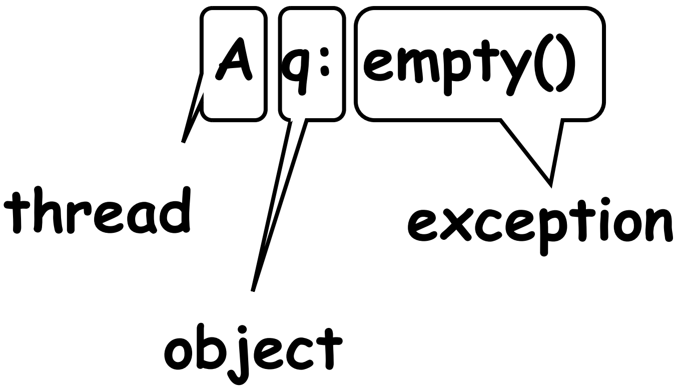


# Notazione




(2) 93

# Notazione




(2) 94

## Descrizione di una esecuzione
## ***History***

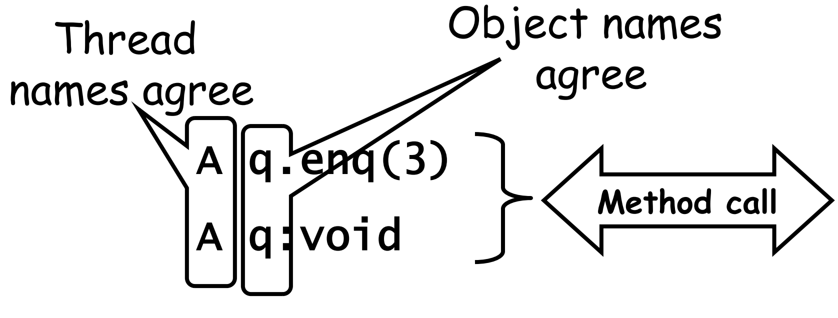
```
        A q.enq(3)
        A q:void
        A q.enq(5)
H =     B p.enq(4)
        B p:void
        B q.deq()
        B q:3
```

**Sequenza di chiamate/ risposte dei metodi**

95

## Definizione

- Chiamata & risposta *match* se




(1)

96

# Object Projections

H =
```
A q.enq(3)
A q:void
B p.enq(4)
B p:void
B q.deq()
B q:3
```

97

# Object Projections

H|q =
```
A q.enq(3)
A q:void

B q.deq()
B q:3
```

98

## Thread Projections

H =

```
A q.enq(3)
A q:void
B p.enq(4)
B p:void
B q.deq()
B q:3
```

99

## Thread Projections

H|B =

```
B p.enq(4)
B p:void
B q.deq()
B q:3
```

100

## Pending

```
      A q.enq(3)
      A q:void
      A q.enq(5)
H =   B p.enq(4)
      B p:void
      B q.deq()
      B q:3
```

***pending* non ha matching sulla risposta**

101

```
      A q.enq(3)
      A q:void
      A q.enq(5)
H =   B p.enq(4)
      B p:void
      B q.deq()
      B q:3
```

**Potrebbe non avere avuto effetto**

102

```
       A q.enq(3)
       A q:void
       A q.enq(5)
H =    B p.enq(4)
       B p:void
       B q.deq()
       B q:3
```

**Pending: elimare**



# Completamento

```
                A q.enq(3)
                A q:void

Complete(H) =   B p.enq(4)
                B p:void
                B q.deq()
                B q:3
```

# History: sequenziale

```
A q.enq(3)
A q:void
B p.enq(4)
B p:void
B q.deq()
B q:3
A q:enq(5)
```

(4) 105

**match**

```
A q.enq(3)
A q:void
B p.enq(4)
B p:void
B q.deq()
B q:3
A q:enq(5)
```

(4) 106

```
A q.enq(3)
A q:void
B p.enq(4)
B p:void
B q.deq()
B q:3
A q:enq(5)
```

match

match

(4) 107

```
A q.enq(3)
A q:void
B p.enq(4)
B p:void
B q.deq()
B q:3
A q:enq(5)
```

match

match

match

(4) 108

```
A q.enq(3)
A q:void
B p.enq(4)
B p:void
B q.deq()
B q:3
A q:enq(5)
```

match

match

match

pending finale
OK

(4)

109

# Storie sequenziali

```
A q.enq(3)
A q:void
B p.enq(4)
B p:void
B q.deq()
B q:3
A q:enq(5)
```

match

match

match

pending finale
OK

Non si sovrappongono
Chiamate di thread
differenti

(4)

110

# History: buona formazione

H=
```
A q.enq(3)
B p.enq(4)
B p:void
B q.deq()
A q:void
B q:3
```

111

**Proiezione sequenziale**

H=
```
A q.enq(3)
B p.enq(4)
B p:void
B q.deq()
A q:void
B q:3
```

H|B=
```
B p.enq(4)
B p:void
B q.deq()
B q:3
```

112

H|B=
B p.enq(4)
B p:void
B q.deq()
B q:3

H=
A q.enq(3)
B p.enq(4)
B p:void
B q.deq()
A q:void
B q:3

H|A=
A q.enq(3)
A q:void

113

# Equivalenza

**Nessuna differenza osservabile dai thread**

H|A = G|A
H|B = G|B

H=
A q.enq(3)
B p.enq(4)
B p:void
B q.deq()
A q:void
B q:3

G=
A q.enq(3)
A q:void
B p.enq(4)
B p:void
B q.deq()
B q:3

114

# Specifica sequenziale

- Una specifica sequenziale descrive quando una storia e' legale nell'assunzione
  - Single-thread, single-object
- Tecniche:
  - Pre & post
  - Numerose tecniche

115

# Legal Histories

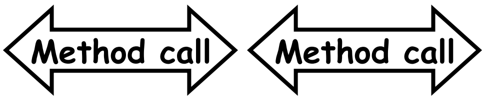
- Una storia (multi-object) H e' legale se
  - Per ogni oggetto **x**
  - **H|x** una specifica sequenziale di **x**

116

# Precedenza

```
A q.enq(3)
B p.enq(4)
B p.void
A q:void
B q.deq()
B q:3
```

**Una chiamata precede un'altra se l'evento di risposta precece l'evento di chiamata**




(1) 117

# Non-Precedenza

```
A q.enq(3)
B p.enq(4)
B p.void
B q.deq()
A q:void
B q:3
```

**Overlap!!**

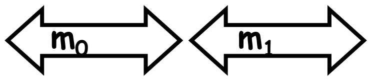

(1) 118

# Notazione

- History **H**
- Esecuzione dei metodi $\mathbf{m_0}$ e $\mathbf{m_1}$ in **H**

• Diciamo che $m_0 \rightarrow_H m_1$, se
  - $\mathbf{m_0}$ precede $\mathbf{m_1}$

• La relazione $\mathbf{m_0} \rightarrow_H \mathbf{m_1}$




  - Ordine parziale
  - Ordine totale se **H** e' sequenziale

119

# Linearizability

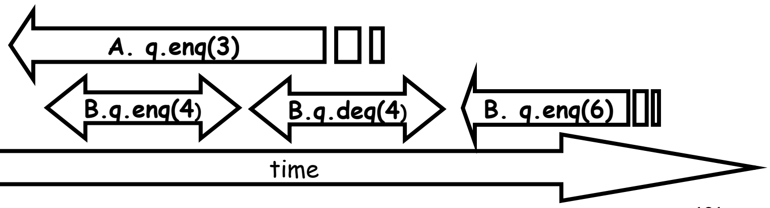
• Una history H e' ***linearizable*** se puo' essere estesa a ***G***:
  - Aggiungendo risposte alle chiamate pending
  - Scartando chiamate pending

• In modo tale che G sia equivalente a
  - History **S sequenziale e legale**
  - $\rightarrow_G \subset \rightarrow_S$
    - **S** rispetta l'ordine di ***G***

120

# Esempio

```
A q.enq(3)
B q.enq(4)
B q:void
B q.deq()
B q:4
B q:enq(6)
```

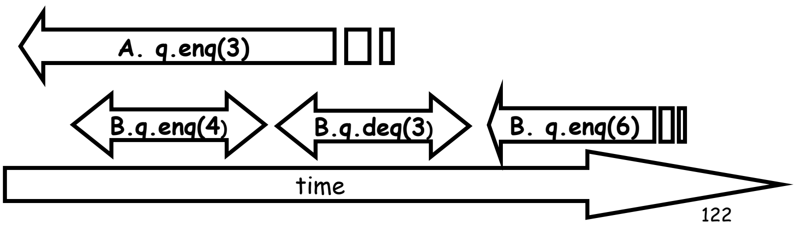



121

# Esempio

```
A q.enq(3)
B q.enq(4)
B q:void
B q.deq()
B q:4
B q:enq(6)
```

**Pending**
**Da completare**

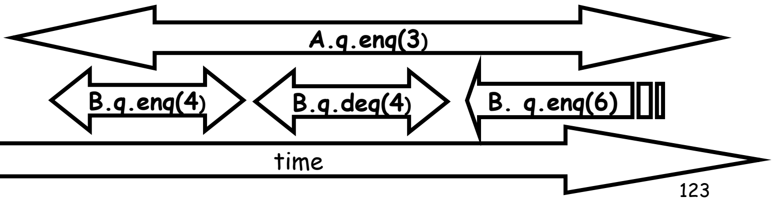



122

# esempio

```
A q.enq(3)
B q.enq(4)
B q:void
B q.deq()
B q:4
B q:enq(6)
A q:void
```

**pendingn**

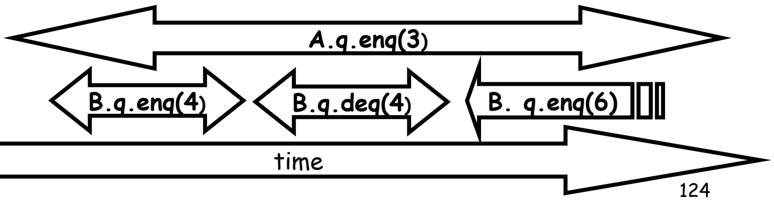



123

# esempio

```
A q.enq(3)
B q.enq(4)
B q:void
B q.deq()
B q:4
B q:enq(6)
A q:void
```

**eliminare**

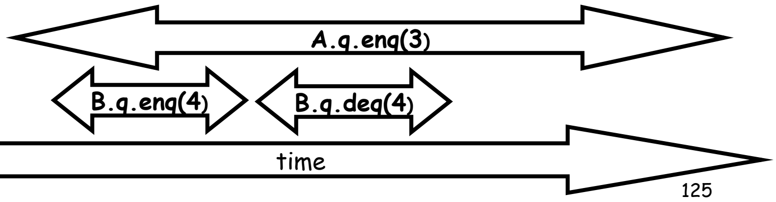



124

# esempio

```
A q.enq(3)
B q.enq(4)
B q:void
B q.deq()
B q:4

A q:void
```

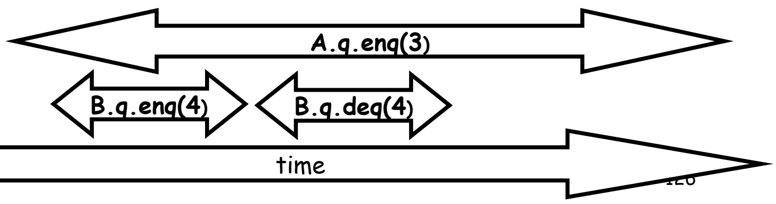
**eliminare**




125

# esempio

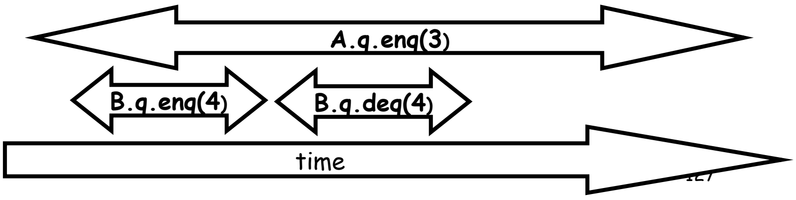
```
A q.enq(3)
B q.enq(4)
B q:void
B q.deq()
B q:4
A q:void
```




126

A q.enq(3)
B q.enq(4)
B q:void
B q.deq()
B q:4
A q:void

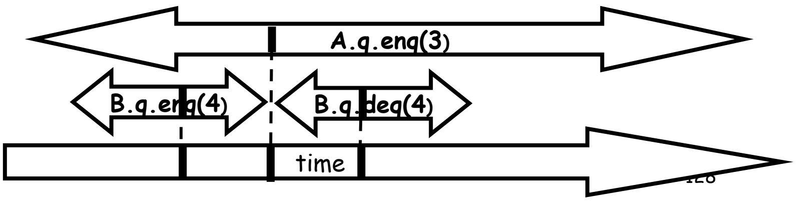
B q.enq(4)
B q:void
A q.enq(3)
A q:void
B q.deq()
B q:4




A q.enq(3)
B q.enq(4)
B q:void
B q.deq()
B q:4
A q:void

**Storia sequenziale equivalente**

B q.enq(4)
B q:void
A q.enq(3)
A q:void
B q.deq()
B q:4



# Discussione

- Domanda: la nozione di linearizability implica situazioni in cui i metodi vengono sospesi?
- Risposta: NO!!.
- Linearizability e' *non-blocking*

129

Consideriamo l'invocazione del metodo

```
A q.inv(...)
```

Assumiamo che sia pending in H, allora esiste una risposta

```
A q:res(...)
```

Tale che

```
H + A q:res(...)
```

e' linearizable

130

# Come si dimostra?

- Prendiamo una linearizzazione S diH
- Se S contiene
  - A `q.inv(…)` e la sua risposyta allora abbiamo fatto!! ,
  - Altrimenti, costruiamo
  - `S + A q.inv(…) + A q:res(…)`

131

# Composability Theorem

- H e' linearizable se e solo se
  - Per ogni oggetto x
  - H|x e' linearizable

132

## Perche e' importante?

- Modularita'
- Tecnica di prova per gli oggetti

133