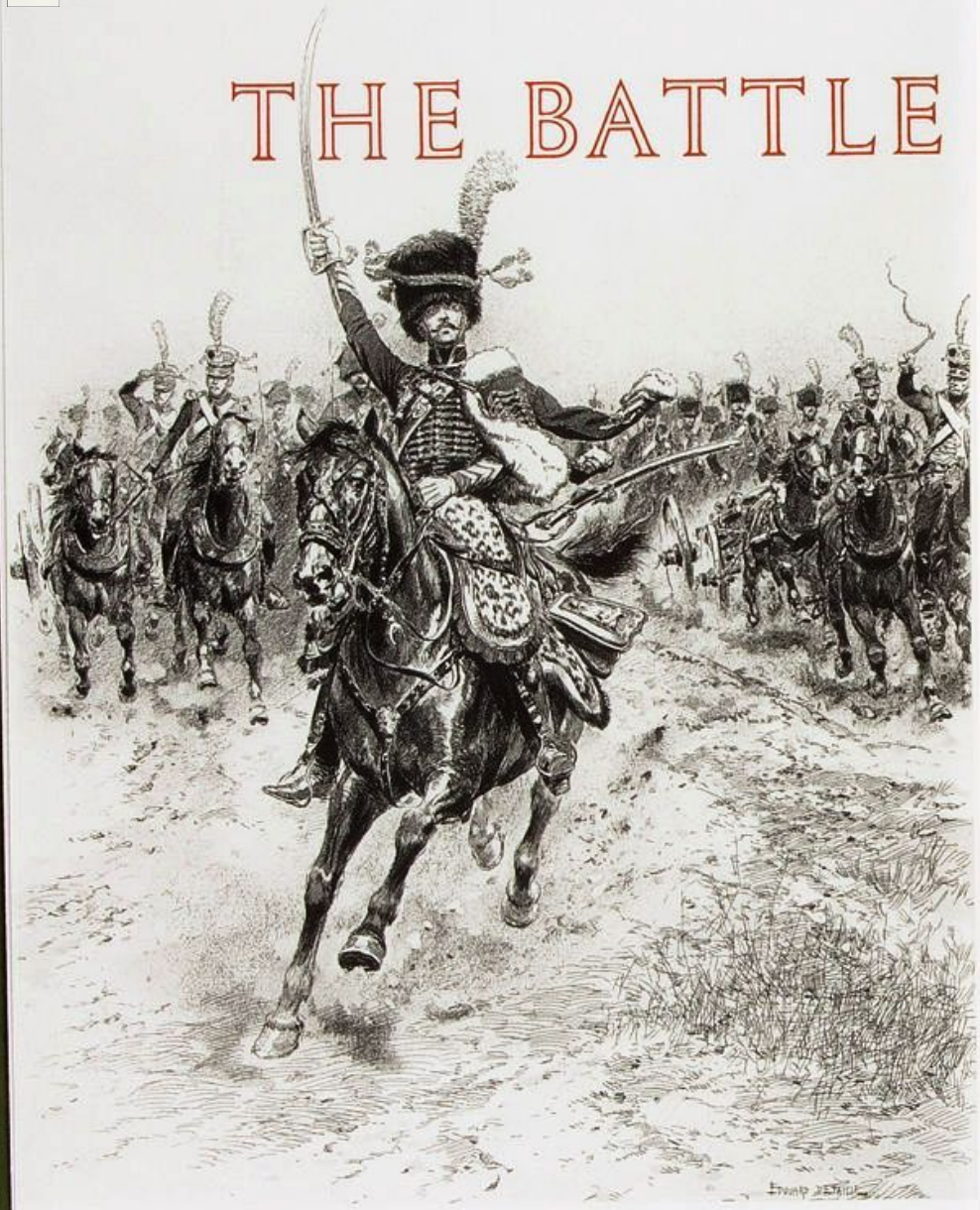
CONCORRENZA DI EVENTI



THE BATTLE OF WATERLOO

A ROMANTIC NARRATIVE BY

VICTOR HUGO from Les Misérables

with an Introduction by DREW MIDDLETON

Military Correspondent of The New York Times, and an

Epilogue, The Waterloo Dispatch, by REGINALD COLBY

The illustrations are reproduced from engravings by

EDOUARD DETAILLE

The Limited Editions Club · 1977

es. spin_10: interleaving

```
byte     n = 0;
active proctype P() {
     n = 1;
   printf("Processo P, n = %d\n", n);
}
active proctype Q() {
     n = 2;
   printf("Processo Q, n = %d\n", n);
}
```

- $ **spin -p -g** filename.pml

1	2	3	4	5	6
n = 1	n = 1	n = 1	n = 2	n = 2	n = 2
printf(P)	n = 2	n = 2	printf(Q)	n = 1	n = 1
n = 2	printf(P)	printf(Q)	n = 1	printf(Q)	printf(P)
printf(Q)	printf(Q)	printf(P)	printf(P)	printf(P)	printf(Q)

simulazione interattiva

```
byte     n = 0;
active proctype P() {
     n = 1;
   printf("Processo P, n = %d\n", n);
}
active proctype Q() {
     n = 2;
   printf("Processo Q, n = %d\n", n);
}
```

- in casi reali la simulazione random esplora *fra* tutte le possibili computazioni e quindi può risultare inutile o non praticabile
- simulazione interattiva:
  chiede ogni volta all'utente che scelta fare
- `$ spin -i filename.pml`

es. spin_11: interferenze

```
byte    n = 0;

active proctype P() {
  byte temp;
  temp = n + 1;
  n = temp;
  printf("Processo P, n = %d\n", n)
}

active proctype Q() {
  byte temp;
  temp = n + 1;
  n = temp;
  printf("Processo Q, n = %d\n", n)
}
```

Process	Statement	n	P:temp	Q:temp	Output
P	temp = n + 1	0	0	0	
Q	temp = n + 1	0	1	0	
P	n = temp	0	1	1	
Q	n = temp	1	1	1	
P	printf(P)	1	1	1	
Q	printf(Q)	1	1	1	P, n = 1
					Q, n = 1

es. spin_12: insiemi di processi identici

```
byte     n = 0;

active [2]  proctype P() {
  byte temp;
  temp = n + 1;
  n = temp;
  printf("Processo P%d, n = %d\n", _pid, n);
}
```

es. spin_13: operatore `run` e processo `init`

```
byte n;
```

non c'è **active**

```
proctype P(byte id ; byte incr) {
  byte temp;
  temp = n + incr;
  n = temp;
  printf("Processo P%d, n = %d\n", id, n)
}

init {
  n = 1;
  atomic {
    run P(1, 10);
    run P(2, 15)
  }
}
```

es. spin_14: verifica con asserzione

```
byte     n = 0;
proctype P() {
     byte temp, i;
     for (i: 1..5) {
          temp = n;
          n = temp + 1
     }
}
init {
     atomic {
          run P();
          run P()
     }
     (_nr_pr == 1); // forza init ad attendere gli altri due
     printf("Il valore è %d\n", n);
     assert (n > 2) // per cercare il caso peggiore
}
```

es. spin_15: d_step { }

```
byte     n = 0;
active proctype P() {
  byte temp;
  d_step {
  temp = n + 1;
  n = temp;
  }
  printf("Processo P, n = %d\n", n)
}
active proctype Q() {
  byte temp;
  d_step {
  temp = n + 1;
  n = temp;
  }
  printf("Processo Q, n = %d\n", n)
}
```

Process	Statement	n	P:temp	Q:temp	Output
P	temp = n+1; n = temp	0	0	0	
Q	temp = n+1; n = temp	1	1	0	
P	printf("P")	2	1	2	
Q	printf("Q")	2	1	2	P, n = 2 Q, n = 2

es. spin_16: sezione critica

```
bool wantP = false, wantQ = false;
active proctype P() {
    do
     :: printf("Sezione NON Critica P\n");
        wantP = true;
        printf("Sezione Critica P\n");
        wantP = false
    od
}
active proctype Q() {
  do
  ::  printf("Sezione NON Critica Q\n");
      wantQ = true;
      printf("Sezione Critica Q\n");
      wantQ = false
  od
}
```

es. spin_17: sezione critica

```
bool wantP = false, wantQ = false;
byte critica = 0;
active proctype P() {
    do
    :: printf("Sezione NON Critica P\n");
       wantP = true;
       critica++;
       printf("Sezione Critica P\n");
       assert (critica <= 1);
       critica--;
       wantP = false
    od
}
active proctype Q() {
    do
    :: printf("Sezione NON Critica Q\n");
       wantQ = true;
       critica++;
       printf("Sezione Critica Q\n");
       assert (critica <= 1);
       critica--;
       wantQ = false
    od
}
```

es. spin_18: sincronizzazione busy-waiting

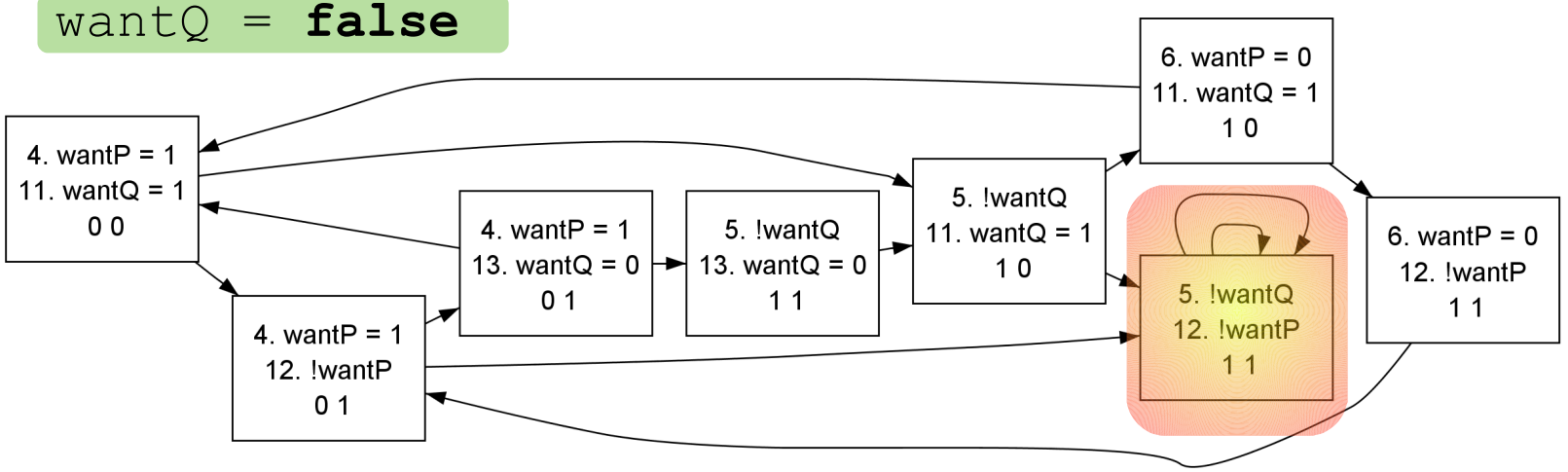
```
bool wantP = false, wantQ = false;
active proctype P() {
    do
    :: printf("Non critical section P\n");
       wantP = true;
       do
        :: !wantQ -> break;
        :: else -> skip      // busy waiting: posso eliminare
       od;
       printf("Critical section P\n");
       wantP = false
    od
}
active proctype Q() {
    do
    :: printf("Non critical section Q\n");
       wantQ = true;
       do
        :: !wantP -> break;
        :: else -> skip      // busy waiting: posso eliminare
       od;
       printf("Critical section Q\n");
       wantQ = false
     od
}
```

es. spin_19: sincronizzazione blocking

```
bool wantP = false, wantQ = false;
active proctype P() {
    do
    :: printf("Sezione NON Critica P\n");
       wantP = true;
       !wantQ;          // blocking waiting
       printf("Sezione Critica P\n");
       wantP = false
    od
}
active proctype Q() {
    do
    :: printf("Sezione NON Critica Q\n");
       wantQ = true;
       !wantP;          // blocking waiting
       printf("Sezione Critica Q\n");
       wantQ = false
     od
}
```

es. spin_19: deadlock

```
1  bool wantP = false, wantQ = false;
2
3  active proctype P() {
4   do :: wantP = true;
5         !wantQ;
6         wantP = false
7   od
8  }
9
10 active proctype Q() {
11  do :: wantQ = true;
12        !wantP;
13        wantQ = false
14  od
15 }
```



es. spin_20: atomizzazione

```
bool wantP = false, wantQ = false;
active proctype P() {
  do :: printf("Sezione NON Critica P\n");
        atomic {
           !wantQ;
           wantP = true
        }
        printf("Sezione Critica P\n");
        wantP = false
  od
}
active proctype Q() {
  do ::  printf("Sezione NON Critica Q\n");
        atomic {
           !wantP;
           wantQ = true
        }
        printf("Sezione Critica Q\n");
        wantQ = false
  od
}
```

es. spin_21: semafori

```
byte sem = 1;
active proctype P() {
  do :: printf("Sezione NON Critica P\n");
        atomic {    // WAIT
          sem > 0;
          sem--
        }
        printf("Sezione Critica P\n");
        sem++       // SIGNAL
  od
}
active proctype Q() {
  do ::  printf("Sezione NON Critica Q\n");
        atomic {    // WAIT
          sem > 0;
          sem--
        }
        printf("Sezione Critica Q\n");
        sem++       // SIGNAL
  od
}
```

es. spin_23: end

```
byte request = 0;
active proctype Server1() {
  do
  :: request == 1 ->
        printf("Service 1\n");
        request = 0;
  od
}
active proctype Server2() {
  do
  :: request == 2 ->
        printf("Service 2\n");
        request = 0;
  od
}
active proctype Client() {
  request = 1;
  request == 0;
  request = 2;
  request == 0;
}
```