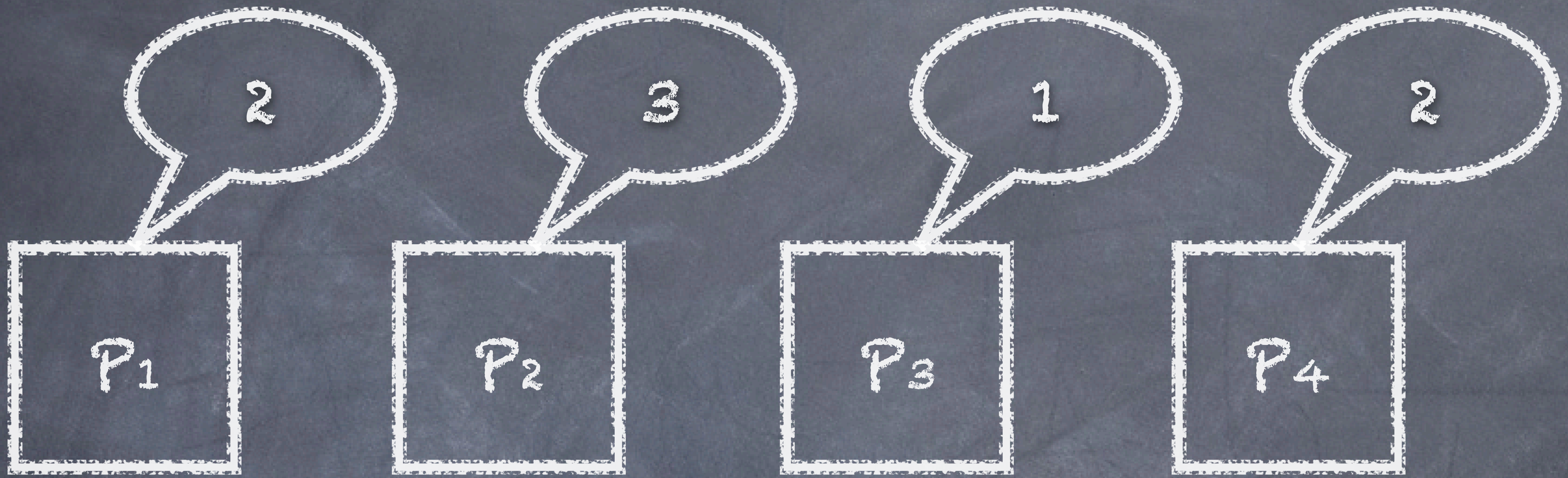
Tight Bounds for Anonymous Adopt-Commit Objects

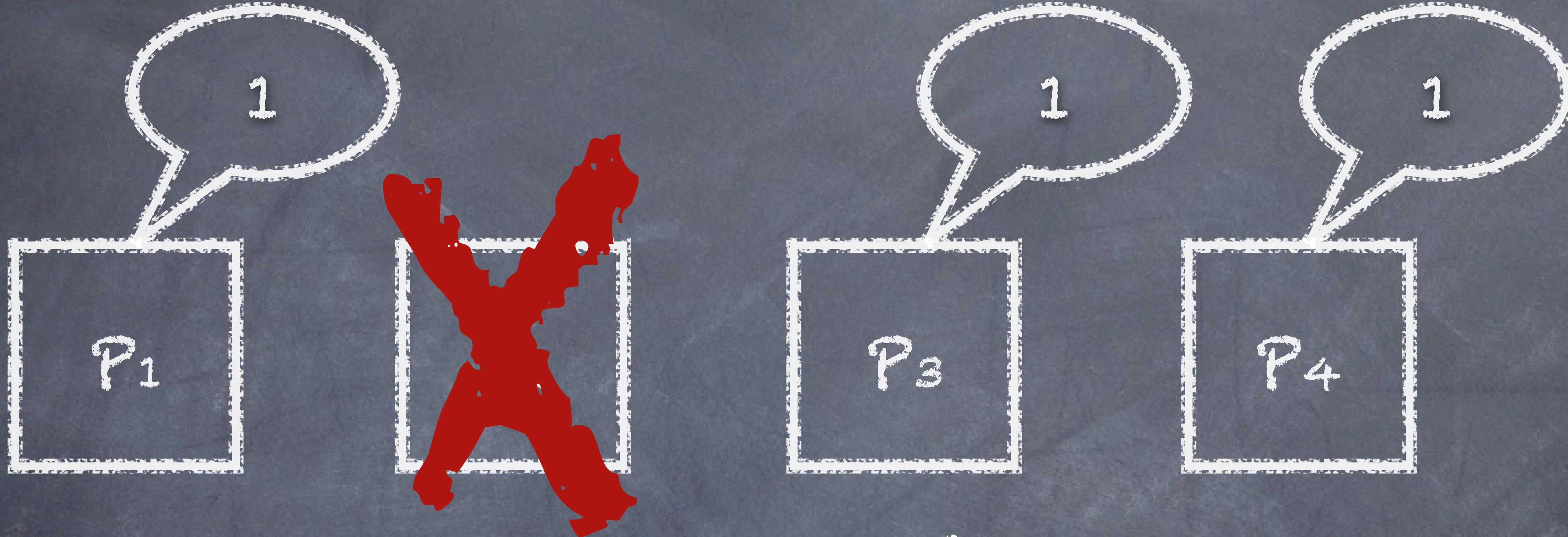
Faith Ellen
University of Toronto

joint work with Jim Aspnes
to appear at SPAA 2011

Consensus



Consensus



- termination: each nonfaulty process outputs a value
- agreement: all outputs are the same
- validity: every output is an input

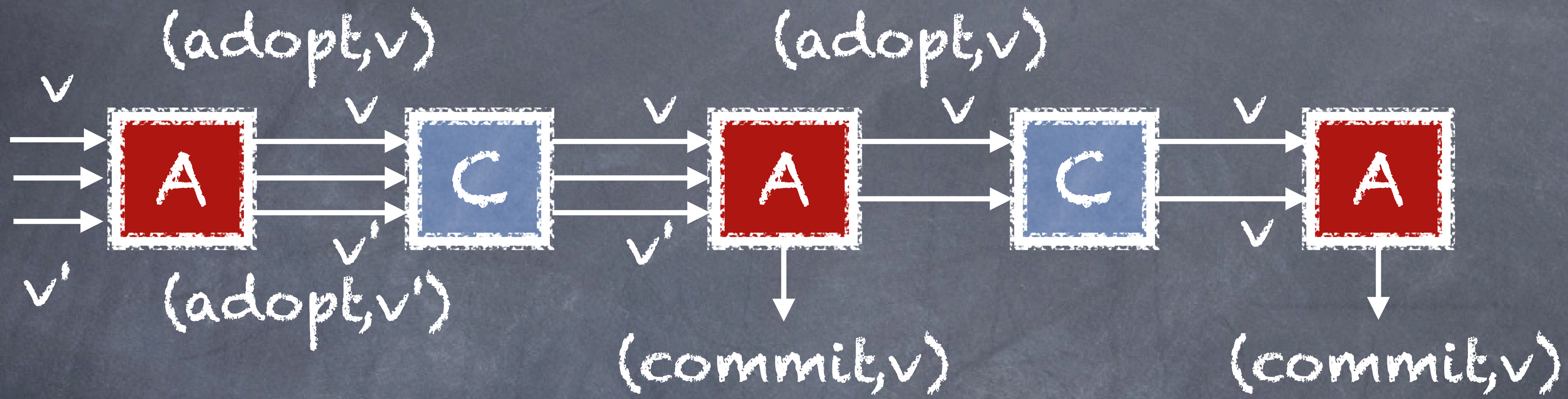
consensus using only r/w registers:

there is no deterministic algorithm that tolerates 1 process crash in an asynchronous system [FLP, LA]

there are randomized algorithms that tolerate any number of process crashes in asynchronous systems [A, AB, AC, AH, C, CIL]

- termination: each nonfaulty process outputs a value with probability 1

randomized consensus algorithms



convergence: if all inputs are v, all outputs are (commit,v)

coherence: if some output is (commit,v), every output is (commit,v) or (adopt,v)

probabilistic agreement: all outputs are the same with probability $\Delta > 0$

A = expected step complexity of adopt-commit

C = expected step complexity of conciliator = $O(\log n)$

if Δ is constant, expected step complexity of consensus is $O(A + C)$

m-valued adopt-commit objects

- O(n) deterministic [Gafni]
- O(log m) deterministic, anonymous [Aspnes]

anonymous: all processes run the same code

- O(min(n, log m / log log m)) deterministic, anonymous and matching randomized, anonymous lower bound

m-valued adopt-commit object

adoptCommit(u), u in [1,m]

possible outputs: {(adopt,v)| v in [1,m]}

U {(commit,v)| v in [1,m]}

termination: each nonfaulty process outputs a value

validity:: every output is an input

convergence: if all inputs are v, all outputs are (commit,v)

coherence: if some output is (commit,v), every output is (commit,v) or (adopt,v)

m-valued conflict detector

check(v), v in [1,m]

possible outputs: {true, false}

termination: each nonfaulty process outputs a value

in every execution in which all check operations have the same input, they all output false

in every execution that contains check(v) and check(v'), at least one of them outputs true

a conflict detector from an adopt-commit object

```
check(v)
(d,v') := adoptCommit(v)
if  (d,v') = (commit,v)
then return false
else return true
```

an adopt-commit object from a conflict detector and registers

```
adoptCommit(v)
if check(v) then conflict := true
             else u := proposal
if u = 0 then proposal := v
          else v := u
b := conflict
if b then return (adopt,v)
     else return (commit,v)
```

conflict

initially false

proposal

initially 0

a conflict detector from registers

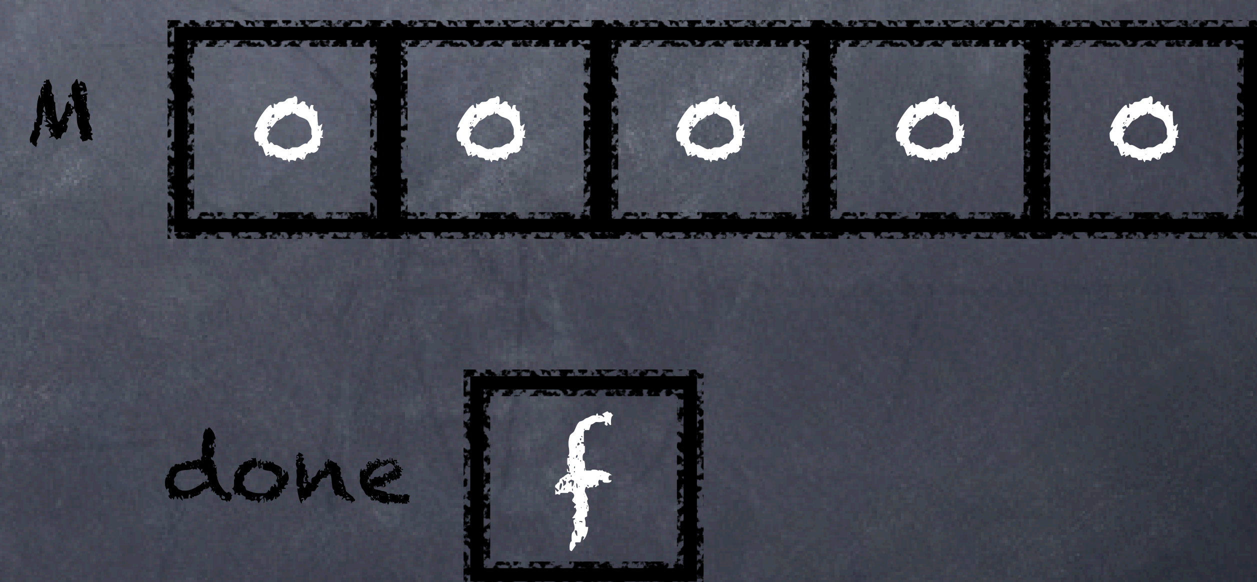
```
check(v)
w: for i := 1 to n do
      if done then goto r
      M[i] := v
done := true
r: for i := 1 to n do
      if M[i] ≠ v
      then return true
return false
```

done

initially false

M[1..n]

all initially 0



a conflict detector from registers

```
check(v)
w: for i := 1 to n do
     if done then goto r
     M[i] := v
done := true
r: for i := 1 to n do
     if M[i] ≠ v
     then return true
return false
```

done

initially false

M[1..n]

all initially 0

M				
2	0	0	0	0

done
f

a conflict detector from registers

```
check(v)
w: for i := 1 to n do
       if done then goto r
       M[i] := v
done := true
r: for i := 1 to n do
       if M[i] ≠ v
       then return true
return false
```

done

initially false

M[1..n]

all initially 0

M					
	2	2	0	0	0

done	
	f

a conflict detector from registers

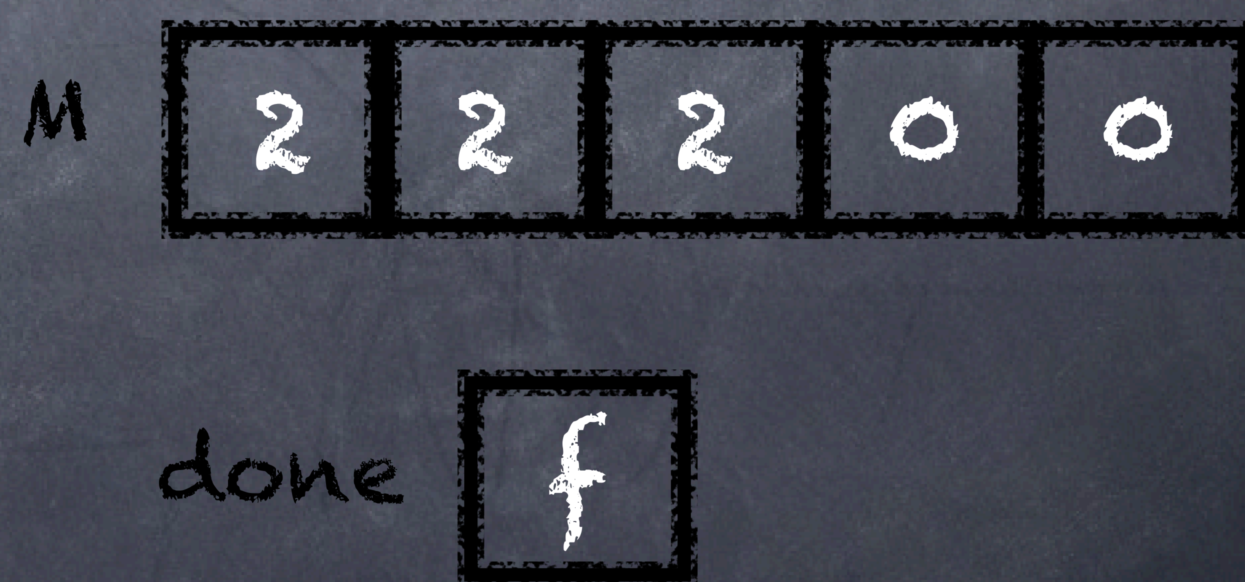
```
check(v)
w: for i := 1 to n do
      if done then goto r
      M[i] := v
done := true
r: for i := 1 to n do
      if M[i] ≠ v
      then return true
return false
```

done

initially false

M[1..n]

all initially 0



a conflict detector from registers

check(v)

```
w: for i := 1 to n do
      if done then goto r
      M[i] := v
done := true
r: for i := 1 to n do
      if M[i] ≠ v
      then return true
return false
```

done

initially false

M[1..n]

all initially 0

M	2	2	2	2	0

done	f

a conflict detector from registers

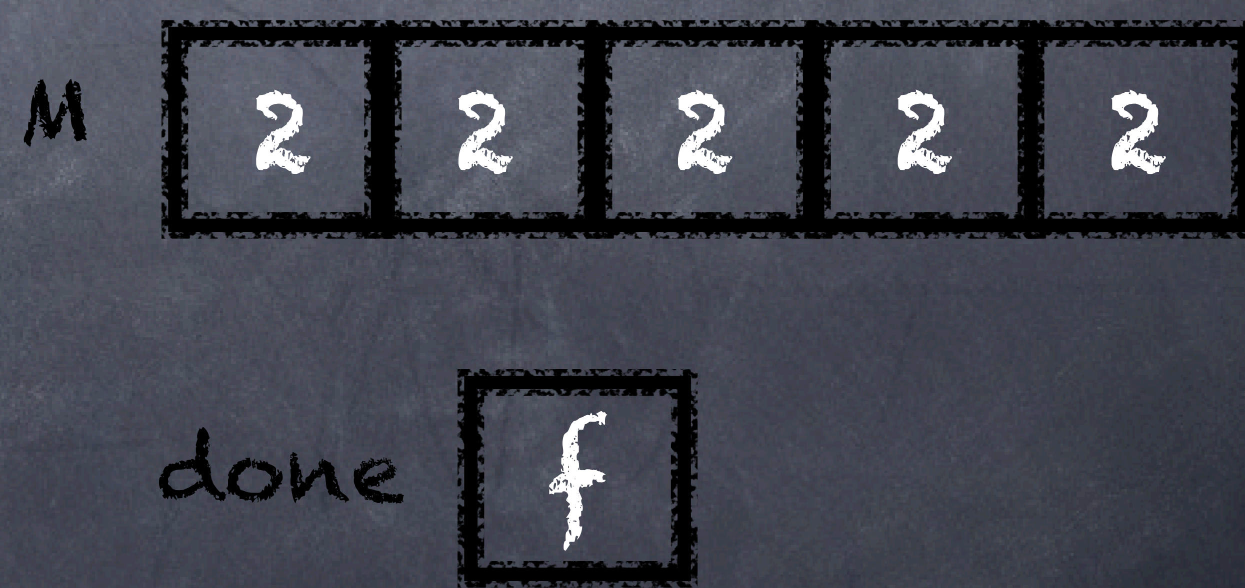
```
check(v)
w: for i := 1 to n do
       if done then goto r
       M[i] := v
done := true
r: for i := 1 to n do
       if M[i] ≠ v
       then return true
return false
```

done

initially false

M[1..n]

all initially 0



a conflict detector from registers

```
check(v)
w: for i := 1 to n do
       if done then goto r
       M[i] := v
done := true
r: for i := 1 to n do
       if M[i] ≠ v
       then return true
return false
```

done

initially false

M[1..n]

all initially 0

M					
	2	2	2	2	2

done	
	t

a conflict detector from registers

```
check(v)
w: for i := 1 to n do
      if done then goto r
      M[i] := v
done := true
r: for i := 1 to n do
      if M[i] ≠ v
      then return true
return false
```

done

initially false

M[1..n]

all initially 0

M				
2	2	2	2	0

done
f

a conflict detector from registers

```
check(v)
w: for i := 1 to n do
      if done then goto r
      M[i] := v
done := true
r: for i := 1 to n do
      if M[i] ≠ v
      then return true
return false
```

done

initially false

M[1..n]

all initially 0

M					
	3	2	2	2	0

done	
	f

a conflict detector from registers

```
check(v)
w: for i := 1 to n do
       if done then goto r
       M[i] := v
done := true
r: for i := 1 to n do
       if M[i] ≠ v
       then return true
return false
```

done

initially false

M[1..n]

all initially 0

M	3	2	2	2	2

done	f

a conflict detector from registers

```
check(v)
w: for i := 1 to n do
       if done then goto r
       M[i] := v
done := true
r: for i := 1 to n do
       if M[i] ≠ v
       then return true
return false
```

done

initially false

M[1..n]

all initially 0

M				
3	3	2	2	2

done
f

a conflict detector from registers

check(v)

w: for i := 1 to n do

    if done then goto
r

    M[i] := v

done := true

r: for i := 1 to n do

    if M[i] ≠ v

    then return true

return false

done

initially false

M[1..n]

all initially 0

M	3	3	2	2	2

done	t

a conflict detector from registers

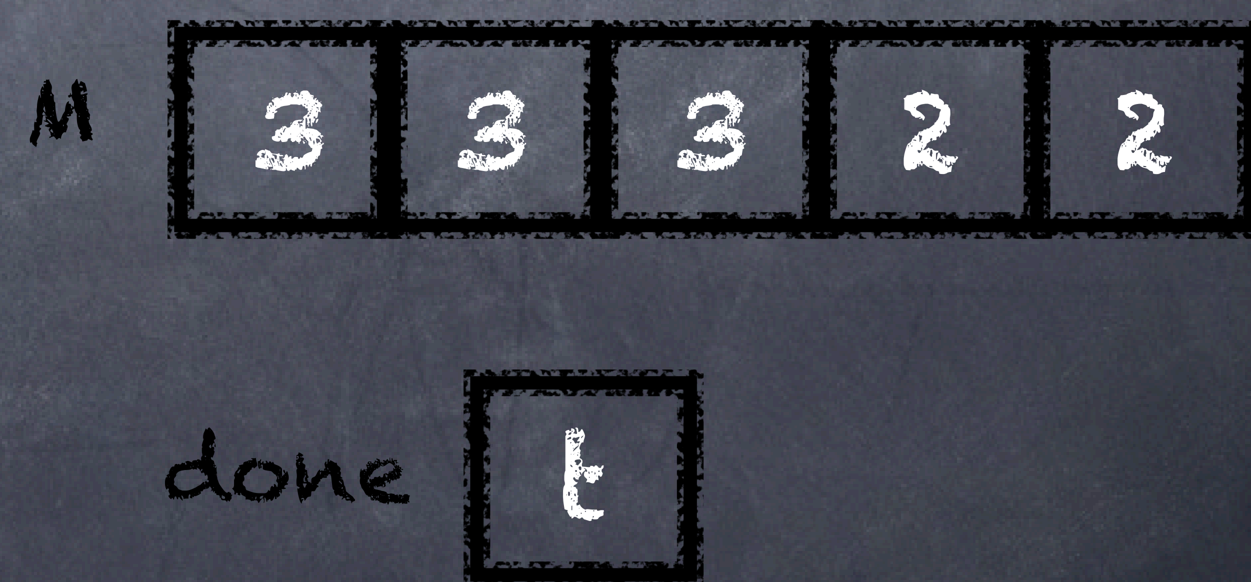
```
check(v)
w: for i := 1 to n do
       if done then goto r
       M[i] := v
done := true
r: for i := 1 to n do
       if M[i] ≠ v
       then return true
return false
```

done

initially false

M[1..n]

all initially 0



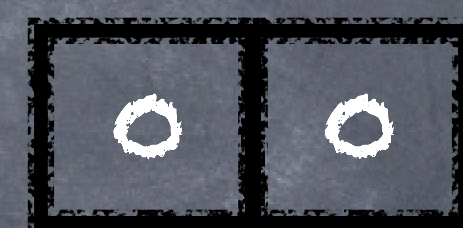
a 2-valued conflict detector from registers

check(v)

```
M[v] := v
if v = 1
then x := M[2]
else x := M[1]
if x ≠ 0
then return true
else return false
```

M[1..2]

both initially 0

M 

a 2-valued conflict detector from registers

check(v)

M[v] := v

if v = 1

then x := M[2]

else x := M[1]

if x ≠ 0

then return true

else return false

M[1..2]

both initially 0

a 2-valued conflict detector from registers

check(v)

M[v] := v

if v = 1

then x := M[2]

else x := M[1]

if x ≠ 0

then return true

else return false

M[1..2]

both initially 0

an m-valued conflict detector from registers

```
check(v)
for i := 1 to k do
    x := M[π_v[i]]
    if x = 0
    then M[π_v[i]] := v
    else if x ≠ v
          then return true
return false
```

$\pi_1, \ldots, \pi_m$

distinct

permutations

of $\{1,\ldots,k\}$

$M[1..k]$

all initially 0

M			
	0	0	0

k is $O(\log m / \log \log m)$

an m-valued conflict detector from registers

check(v)

for i := 1 to 3 do

 x := $M[\pi_v[i]]$

 if x = 0

 then $M[\pi_v[i]]$:= v

 else if x $\neq$ v

  then return true

return false

$\pi_1 = [1\ 2\ 3]$

$\pi_2 = [1\ 3\ 2]$

$\pi_3 = [2\ 1\ 3]$

$\pi_4 = [2\ 3\ 1]$

$\pi_5 = [3\ 1\ 2]$

$\pi_6 = [3\ 2\ 1]$

M	0	0	0

an m-valued conflict detector from registers

```
check(v)
for i := 1 to 3 do
    x := M[π_v[i]]
    if x = 0
    then M[π_v[i]] := v
    else if x ≠ v
          then return true
return false
```

$\pi_1 = [1\ 2\ 3]$

$\pi_2 = [1\ 3\ 2]$

$\pi_3 = [2\ 1\ 3]$

$\pi_4 = [2\ 3\ 1]$

$\pi_5 = [3\ 1\ 2]$

$\pi_6 = [3\ 2\ 1]$

M			
M	1	0	0

an m-valued conflict detector from registers

```
check(v)
for i := 1 to 3 do
    x := M[π_v[i]]
    if x = 0
    then M[π_v[i]] := v
    else if x ≠ v
        then return true
return false
```

$\pi_1 = [1\ 2\ 3]$

$\pi_2 = [1\ 3\ 2]$

$\pi_3 = [2\ 1\ 3]$

$\pi_4 = [2\ 3\ 1]$

$\pi_5 = [3\ 1\ 2]$

$\pi_6 = [3\ 2\ 1]$

M			
	1	1	0

an m-valued conflict detector from registers

```
check(v)
for i := 1 to 3 do
    x := M[π_v[i]]
    if x = 0
    then M[π_v[i]] := v
    else if x ≠ v
         then return true
return false
```

$\pi_1 = [1\ 2\ 3]$

$\pi_2 = [1\ 3\ 2]$

$\pi_3 = [2\ 1\ 3]$

$\pi_4 = [2\ 3\ 1]$

$\pi_5 = [3\ 1\ 2]$

$\pi_6 = [3\ 2\ 1]$

M			
	1	1	1

an m-valued conflict detector from registers

```
check(v)
for i := 1 to 3 do
    x := M[π_v[i]]
    if x = 0
    then M[π_v[i]] := v
    else if x ≠ v
            then return true
return false
```

$\pi_1 = [1\ 2\ 3]$

$\pi_2 = [1\ 3\ 2]$

$\pi_3 = [2\ 1\ 3]$

$\pi_4 = [2\ 3\ 1]$

$\pi_5 = [3\ 1\ 2]$

$\pi_6 = [3\ 2\ 1]$

M			
	0	0	0

an m-valued conflict detector from registers

```
check(v)
for i := 1 to 3 do
    x := M[π_v[i]]
    if x = 0
    then M[π_v[i]] := v
    else if x ≠ v
         then return true
return false
```

$\pi_1 = [1\ 2\ 3]$

$\pi_2 = [1\ 3\ 2]$

$\pi_3 = [2\ 1\ 3]$

$\pi_4 = [2\ 3\ 1]$

$\pi_5 = [3\ 1\ 2]$

$\pi_6 = [3\ 2\ 1]$

M			
	0	4	0

an m-valued conflict detector from registers

```
check(v)
for i := 1 to 3 do
    x := M[πv[i]]
    if x = 0
    then M[πv[i]] := v
    else if x ≠ v
            then return true
return false
```

$\pi_1 = [1\ 2\ 3]$

$\pi_2 = [1\ 3\ 2]$

$\pi_3 = [2\ 1\ 3]$

$\pi_4 = [2\ 3\ 1]$

$\pi_5 = [3\ 1\ 2]$

$\pi_6 = [3\ 2\ 1]$

M	0	4	4

an m-valued conflict detector from registers

```
check(v)
for i := 1 to 3 do
    x := M[πv[i]]
    if x = 0
    then M[πv[i]] := v
    else if x ≠ v
            then return true
return false
```

$\pi_1 = [1\ 2\ 3]$

$\pi_2 = [1\ 3\ 2]$

$\pi_3 = [2\ 1\ 3]$

$\pi_4 = [2\ 3\ 1]$

$\pi_5 = [3\ 1\ 2]$

$\pi_6 = [3\ 2\ 1]$

M			
	4	4	4

an m-valued conflict detector from registers

```
check(v)
for i := 1 to 3 do
    x := M[π_v[i]]
    if x = 0
    then M[π_v[i]] := v
    else if x ≠ v
          then return true
return false
```

$\pi_1 = [1\ 2\ 3]$

$\pi_2 = [1\ 3\ 2]$

$\pi_3 = [2\ 1\ 3]$

$\pi_4 = [2\ 3\ 1]$

$\pi_5 = [3\ 1\ 2]$

$\pi_6 = [3\ 2\ 1]$

M

0	0	0

an m-valued conflict detector from registers

check(v)

```
for i := 1 to 3 do
    x := M[π_v[i]]
    if x = 0
    then M[π_v[i]] := v
    else if x ≠ v
        then return true
return false
```

$\pi_1 = [1\ 2\ 3]$

$\pi_2 = [1\ 3\ 2]$

$\pi_3 = [2\ 1\ 3]$

$\pi_4 = [2\ 3\ 1]$

$\pi_5 = [3\ 1\ 2]$

$\pi_6 = [3\ 2\ 1]$

M			
	1	0	0

an m-valued conflict detector from registers

```
check(v)
for i := 1 to 3 do
    x := M[π_v[i]]
    if x = 0
    then M[π_v[i]] := v
    else if x ≠ v
            then return true
return false
```

$\pi_1 = [1\ 2\ 3]$

$\pi_2 = [1\ 3\ 2]$

$\pi_3 = [2\ 1\ 3]$

$\pi_4 = [2\ 3\ 1]$

$\pi_5 = [3\ 1\ 2]$

$\pi_6 = [3\ 2\ 1]$

M

1	4	0

for any $u \neq v$, i is before j in π_u and j is before i in π_v, for some $i \neq j$

$\Omega(\min(n, \log m / \log\log m))$ lower bound on step complexity of anonymous m-valued conflict detectors for n processes

$E(v)$ = solo execution of check(v)

$W(v)$ = registers written to in $E(v)$

$R(v)$ = registers read from, but not written to, in $E(v)$

R3, R2, W5, W3, R1, W5, R2, W6, R3

$E(v)$ = solo execution of check(v)

$W(v)$ = registers written to in $E(v)$

$R(v)$ = registers read from, but not written to, in $E(v)$

R3, R2, W5, W3, R1, W5, R2, W6, R3

$\pi(v)$ = permutation of $W(v) \cup R(v)$ arranged according to first writes to registers in $W(v)$ and last reads from registers in $R(v)$

$E(v)$ = solo execution of check(v)

$W(v)$ = registers written to in $E(v)$

$R(v)$ = registers read from,
but not written to, in $E(v)$

R3, R2, W5, W3, R1, W5, R2, W6, R3

$\pi(v)$ = permutation of $W(v) \cup R(v)$
arranged according to
first writes to registers in $W(v)$ and
last reads from registers in $R(v)$

[5,3,1,2,6]

$E(v)$ = solo execution of check(v)

$W(v)$ = registers written to in $E(v)$

$R(v)$ = registers read from, but not written to, in $E(v)$

LEMMA 1 If $|E(v)| + |E(u)| \leq n$ then there exist i,j in $(W(v) \cup R(v)) \cap (W(u) \cup R(u))$ that occur in different orders in $\pi(v)$ and $\pi(u)$.

Proof: Suppose all i,j in (W(v) U R(v)) ∩ (W(u) U R(u)) occur in the same orders in π(v) and π(u).

E(v) = R3, R2, W5, W3, R1, W5, R2, W6, R3

π(v) = [5,3,1,2,6]

E(u) = R5, R1, W5, R3, R4, R1, W7, W2, R5, W2

π(u) = [5,3,4,1,7,2]

The adversary can construct an execution E' that is indistinguishable from E(v) to p and indistinguishable from E(u) to q.

Proof: Suppose all i,j in (W(v) ∪ R(v)) ∩ (W(u) ∪ R(u)) occur in the same orders in π(v) and π(u).

E(v) = R3, R2, W5, W3, R1, W5, R2, W6, R3

π(v) = [5,3,1,2,6]

E(u) = R5, R1, W5, R3, R4, R1, W7, W2, R5, W2

π(u) = [5,3,4,1,7,2]

The adversary can construct an execution E' that is indistinguishable from E(v) to p and indistinguishable from E(u) to q.

E(v) = R3, R2, W5, W3, R1, W5, R2, W6, R3

E(u) = R5, R1, W5, R3, R4, R1, W7, W2, R5, W2

W5,R3,W3,R1,R1,R2

Ri is scheduled immediately before corresponding Ri/Wi

E(v) = R3, R2, W5, W3, R1, W5, R2, W6, R3

E(u) = R5, R1, W5, R3, R4, R1, W7, W2, R5, W2

W5,W5,R3,W3,R1,R1,R2,W2

Ri is scheduled immediately before corresponding Ri/Wi

Wi is scheduled immediately after corresponding Ri/Wi

E(v) = R3, R2, W5, W3, R1, W5, R2, W6, R3

E(u) = R5, R1, W5, R3, R4, R1, W7, W2, R5, W2

R3,R2,R5,R1,W5,W5,R3,W3,R4,R1,R1,W7,W5,R2,W2,
W6,R5,R3,W2

Ri is scheduled immediately before corresponding Ri/Wi

Wi is scheduled immediately after corresponding Ri/Wi

R/W's between successive R/W's and R'/W's between successive R/W's are interleaved arbitrarily

R3,R2,R5,R1,W5,W5,R3,W3,R4,R1,R1,W7,W5,R2,W2,
W6,R5,R3,W2

Problem: q may read a value written by p or p may read a value written by q

Solution: add clones.

A clone of q is a process with the same input (and code) as q, which is run in lockstep with q, until immediately before some write. The clone performs that write later to ensure that q reads the value it last wrote to that register.

R3,R2,R5,R5,R1,R1,W5,W5,R3,W3,R4,R1,R1,W7,W5,
R2,W2,W6,W5,R5,R3,W2

For each i in $W(v) \cap W(u)$:
add one clone of q for each Ri by q
after its first Wi and
add one clone of p for each Ri by p
after its first Wi

This ensures that any read of M[i]
after the first two writes of M[i]
will see the same value in E'
it saw in E(v) or E(u)

R3,R2,R5,R5,R1,R1,W5,W5,R3,W3,R4,R1,R1,W7,W5,
R2,W2,W6,W5,R5,R3,W2

For each i in R(v) ∩ W(u):
all Ri's, Ri by p occur before the
first write Wi by q and, hence read 0.

For each i in W(v) ∩ R(u):
all Ri's, Ri by q occur before the
first write Ri by p and, hence read 0.

LEMMA 2 Let $\pi(1),\ldots,\pi(m)$ be finite sequences without repetition such that, for every two sequences, $\pi(v)$ and $\pi(u)$, there exist elements i and j that occur in $\pi(v)$ and $\pi(u)$ in different orders. Then $\Sigma\ \{1/|\pi(v)|!\ :\ v = 1,\ldots,m\} \leq 1$.

THEOREM The worst case step complexity of any deterministic anonymous m-valued conflict detector for n processes is $\Omega(\min(n, \log m / \log \log m))$.

Proof: Let $t = \max \{|E(v)| : v = 1,\ldots,m\}$.

Then $|\pi(v)| \leq |E(v)| \leq t$.
If $t > n/2$, the claim is true. Otherwise,
for all $v \neq u$, $|E(v)| + |E(u)| \leq n$.

By Lemma 1, for all u and v, there exist
elements i and j that occur in $\pi(v)$ and
$\pi(u)$ in different orders. Hence,
$m/t! = \Sigma \{1/t! : v = 1,\ldots,m\} \leq$
$\Sigma \{1/|\pi(v)|! : v = 1,\ldots,m\} \leq 1$, by Lemma 2.
So $m \leq t!$ and

t is $\Omega(\log m / \log \log m)$.

COROLLARY Any anonymous randomized m-valued conflict detectors for n processes has $\Omega(\min(n, \log m / \log \log m))$ step complexity with probability 1 against an oblivious adversary.

Suppose not.
For each $v = 1,\ldots,m$, there is a sequence of coin flips such that some solo execution $E(v)$ by a process with input v takes at most t steps, where $t \leq n/2$ and $t! \leq m$.
The proof of the theorem constructs an execution E' in which two processes with different inputs both perform check and return false.
This violates correctness.

THEOREM? Any ~~anonymous~~ randomized m-valued conflict detectors for n processes has $\Omega(\min(n, \log m / \log\log m))$ step complexity with probability 1 against an oblivious adversary.