Fault-Tolerant Termination Detection with Safra's Algorithm

Georgios Karlos
g.karlos@vu.nl

Wan Fokkink
w.j.fokkink@vu.nl

Per Fuchs
per.fuchs@cs.tum.edu

May 2021

Termination Detection

- Detect the completion of a computation
- Non-trivial in a distributed setting

  [EWD 687a][Francez'80]

  - Global state / Local views
  - Requires distributed protocol

Termination Detection

- Detect the completion of a computation
- Non-trivial in a distributed setting

  [EWD 687a][Francez'80]
  - Global state / Local views
  - Requires distributed protocol
- Applications
  - Workpools (work-stealing, load-balancing, ...)
  - Distr. algorithms (routing, self-stabilization, ...)

Termination Detection

- Detect the completion of a computation
- Non-trivial in a distributed setting [EWD 687a][Francez'80]
    - Global state / Local views
    - Requires distributed protocol
- Applications
    - Workpools (work-stealing, load-balancing, ...)
    - Distr. algorithms (routing, self-stabilization, ...)

- Stable property [Chandy'80]
    - $\mathcal{O}(E)$ per snapshot, FIFO

Termination Detection

- Detect the completion of a computation
- Non-trivial in a distributed setting
  [EWD 687a][Francez'80]
  - Global state / Local views
  - Requires distributed protocol
- Applications
  - Workpools (work-stealing, load-balancing, ...)
  - Distr. algorithms (routing, self-stabilization, ...)

- Stable property [Chandy'80]
  - $\mathcal{O}(E)$ per snapshot, FIFO
- Safra's Algorithm [EWD 998]
  - $\mathcal{O}(N)$ time
  - $\mathcal{O}(N)$ messages
  - No FIFO
  - No ACKs

Preliminaries

- Basic vs. Control algorithm/message $\rightarrow$ Detectee vs. Detector

Preliminaries

- Basic vs. Control algorithm/message $\rightarrow$ Detectee vs. Detector

System Model (fault-sensitive)

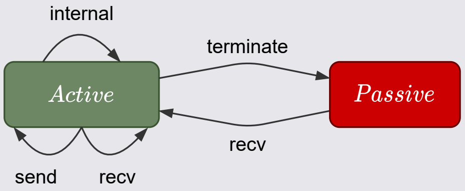
- N processes
- Async message-passing
- No global clock or shared memory
- Messages may arrive in any order
- Delays unbounded but finite

Preliminaries

- Basic vs. Control algorithm/message $\rightarrow$ Detectee vs. Detector

System Model (fault-sensitive)

- N processes
- Async message-passing
- No global clock or shared memory
- Messages may arrive in any order
- Delays unbounded but finite

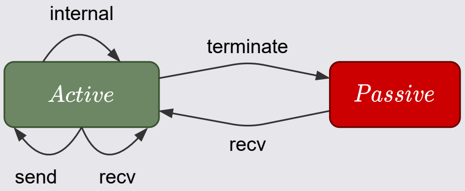


Preliminaries

- Basic vs. Control algorithm/message $\rightarrow$ Detectee vs. Detector

System Model (fault-sensitive)

- N processes
- Async message-passing
- No global clock or shared memory
- Messages may arrive in any order
- Delays unbounded but finite



Termination

All processes are *passive* **and** no (basic) messages are in transit

Safra's Algorithm

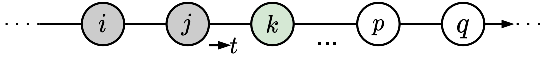
Based on the version from **[Demirbas'00]**

- Ring Overlay
- Process i: $count_i \rightarrow \#sent - \#recv$
- Token t: $count_t \rightarrow$ accumulates $count_i$
- @$passive_i \rightarrow$ termination or forward t
- Looking for $count_t = 0$

Safra's Algorithm

Based on the version from [Demirbas'00]

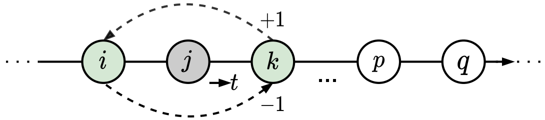
- Ring Overlay
- Process i: $count_i \rightarrow \#sent - \#recv$
- Token t: $count_t \rightarrow$ accumulates $count_i$
- @$passive_i \rightarrow$ termination or forward t
- Looking for $count_t = 0$, but...



Safra's Algorithm

Based on the version from **[Demirbas'00]**

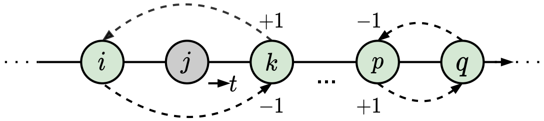
- Ring Overlay
- Process i: $count_i \rightarrow \#sent - \#recv$
- Token t: $count_t \rightarrow$ accumulates $count_i$
- @$passive_i \rightarrow$ termination or forward t
- Looking for $count_t = 0$, but...



Safra's Algorithm

Based on the version from [Demirbas'00]

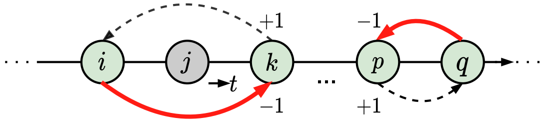
- Ring Overlay
- Process i: $count_i \rightarrow \#sent - \#recv$
- Token t: $count_t \rightarrow$ accumulates $count_i$
- @$passive_i \rightarrow$ termination or forward t
- Looking for $count_t = 0$, but...



Safra's Algorithm

Based on the version from [Demirbas'00]

- Ring Overlay
- Process i: $count_i \rightarrow \#sent - \#recv$
- Token t: $count_t \rightarrow$ accumulates $count_i$
- @$passive_i \rightarrow$ termination or forward t
- Looking for $count_t = 0$, but...

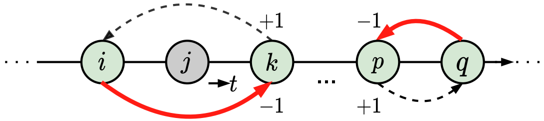


- $count_t$ **inconsistent** at $[k, i-1]$ and p

Safra's Algorithm

Based on the version from [Demirbas'00]

- Ring Overlay
- Process i: $count_i \rightarrow \#sent - \#recv$
- Token t: $count_t \rightarrow$ accumulates $count_i$
- @$passive_i \rightarrow$ termination or forward t
- Looking for $count_t = 0$, but...



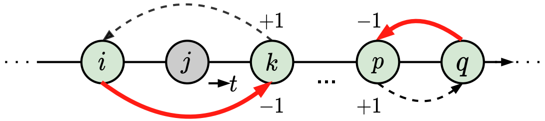
- $count_t$ **inconsistent** at $[k, i-1]$ and p

- seq_i: distinguish visited/unvisited by t
  seq_m: the senders sequence number

Safra's Algorithm

Based on the version from [Demirbas'00]

- Ring Overlay
- Process i: $count_i \rightarrow \#sent - \#recv$
- Token t: $count_t \rightarrow$ accumulates $count_i$
- @$passive_i \rightarrow$ termination or forward t
- Looking for $count_t = 0$, but...



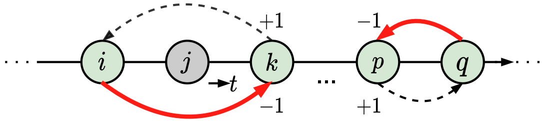
- $count_t$ **inconsistent** at $[k, i-1]$ and p

- seq_i: distinguish visited/unvisited by t
  seq_m: the senders sequence number
- $black_i$: receiver *blackens* the sender
  $black_t$: token accumulates $black_i$

Safra's Algorithm

Based on the version from **[Demirbas'00]**

- Ring Overlay
- Process i: $count_i \rightarrow \#sent - \#recv$
- Token t: $count_t \rightarrow$ accumulates $count_i$
- @$passive_i \rightarrow$ termination or forward t
- Looking for $count_t = 0$, but...



- $count_t$ **inconsistent** at $[k, i-1]$ and p

- seq_i: distinguish visited/unvisited by t
  seq_m: the senders sequence number
- $black_i$: receiver *blackens* the sender
  $black_t$: token accumulates $black_i$

Intuition

- $black_i = j$:
  i knows that count is inconsistent **at least** until j's count is read
- $black_t$: max $black_i$ so far

Safra's Algorithm

```
send_i(m):



recv_i(m, j):



recv_i(t):
```

Safra's Algorithm

```
send$_i$(m):
    $count_i$ ++
    $seq_m \leftarrow seq_i$

recv$_i$(m, j):

recv$_i$(t):
```

Safra's Algorithm

```
send_i(m):
    $count_i$ ++
    $seq_m \leftarrow seq_i$

recv_i(m, j):
    $count_i$ −−
    if $seq_m > seq_i \lor seq_m = seq_i \land j > i$
      $black_i \leftarrow max_i(black_i, j)$

recv_i(t):
```

Safra's Algorithm

```
**send$_i$(m):**
    $count_i$ ++
    $seq_m \leftarrow seq_i$

**recv$_i$(m, j):**
    $count_i$ −−
    if $seq_m > seq_i \vee seq_m = seq_i \wedge j > i$
      $black_i \leftarrow max_i(black_i, j)$

**recv$_i$(t):**
    wait passive$_i$
```

Safra's Algorithm

```
send_i(m):
    $count_i$ ++
    $seq_m \leftarrow seq_i$

recv_i(m, j):
    $count_i$ −−
    if $seq_m > seq_i \lor seq_m = seq_i \land j > i$
      $black_i \leftarrow max_i(black_i, j)$

recv_i(t):
    wait passive_i
    $count_t \mathrel{+}= count_i$
    $black_i \leftarrow max_i(black_i, black_t)$
```

Safra's Algorithm

```
**send$_i$(m):**
    $count_i$ ++
    $seq_m \leftarrow seq_i$

**recv$_i$(m, j):**
    $count_i$ −−
    **if** $seq_m > seq_i \vee seq_m = seq_i \wedge j > i$
      $black_i \leftarrow max_i(black_i, j)$

**recv$_i$(t):**
    **wait** passive$_i$
    $count_t$ += $count_i$
    $black_i \leftarrow max_i(black_i, black_t)$
    **if** $black_i \wedge count_t = 0$: **Announce**
```

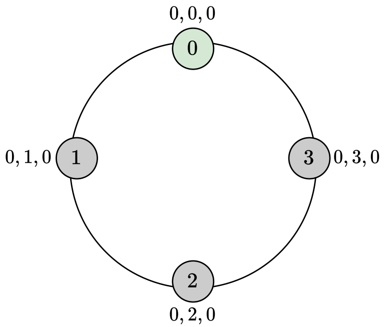
Safra's Algorithm

```
**send$_i$(m):**
    $count_i$ ++
    $seq_m \leftarrow seq_i$

**recv$_i$(m, j):**
    $count_i$ −−
    **if** $seq_m > seq_i \vee seq_m = seq_i \wedge j > i$
      $black_i \leftarrow max_i(black_i, j)$

**recv$_i$(t):**
    **wait** passive$_i$
    $count_t$ += $count_i$
    $black_i \leftarrow max_i(black_i, black_t)$
    **if** $black_i \wedge count_t = 0$: **Announce**
    **else**:
      send $black_t \leftarrow max_i(black_i, succ_i)$
      $black_i \leftarrow i$, $count_i \leftarrow 0$
```

Safra's Algorithm Example

```
send_i(m):
    $count_i$ ++
    $seq_m \leftarrow seq_i$

recv_i(m, j):
    $count_i$ --
    if $seq_m > seq_i \lor seq_m = seq_i \land j > i$
      $black_i \leftarrow max_i(black_i, j)$

recv_i(t):
    wait passive_i
    $count_t$ += $count_i$
    $black_i \leftarrow max_i(black_i, black_t)$
    if $black_i \land count_t = 0$: Announce
    else:
      send $black_t \leftarrow max_i(black_i, succ_i)$
      $black_i \leftarrow i$, $count_i \leftarrow 0$
```

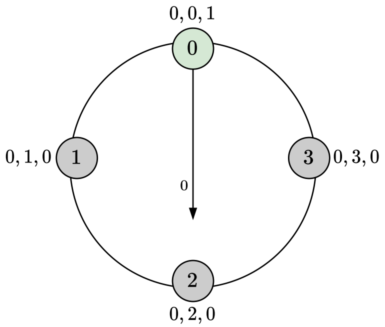
state: $p_i(seq_i, black_i, count_i)$, $t(black_t, count_t)$



Safra's Algorithm Example

```
send_i(m):
    $count_i$ ++
    $seq_m \leftarrow seq_i$

recv_i(m, j):
    $count_i$ −−
    if $seq_m > seq_i \lor seq_m = seq_i \land j > i$
      $black_i \leftarrow max_i(black_i, j)$

recv_i(t):
    wait passive_i
    $count_t$ += $count_i$
    $black_i \leftarrow max_i(black_i, black_t)$
    if $black_i \land count_t = 0$: Announce
    else:
      send $black_t \leftarrow max_i(black_i, succ_i)$
      $black_i \leftarrow i$, $count_i \leftarrow 0$
```

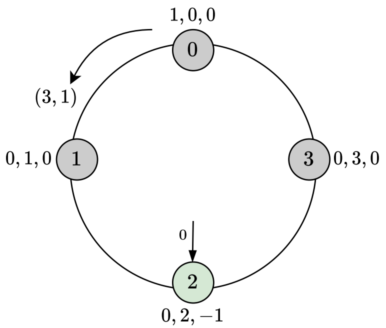
state: $p_i(seq_i, black_i, count_i)$, $t(black_t, count_t)$



Safra's Algorithm Example

```
send_i(m):
    $count_i$ ++
    $seq_m \leftarrow seq_i$

recv_i(m, j):
    $count_i$ −−
    if $seq_m > seq_i \lor seq_m = seq_i \land j > i$
      $black_i \leftarrow max_i(black_i, j)$

recv_i(t):
    wait passive_i
    $count_t$ += $count_i$
    $black_i \leftarrow max_i(black_i, black_t)$
    if $black_i \land count_t = 0$: Announce
    else:
      send $black_t \leftarrow max_i(black_i, succ_i)$
      $black_i \leftarrow i$, $count_i \leftarrow 0$
```

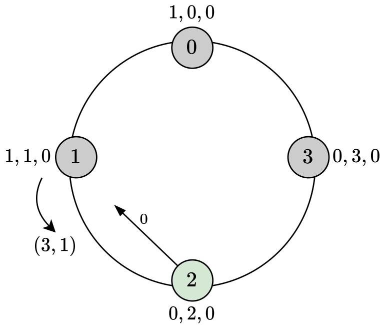
state: $p_i(seq_i, black_i, count_i)$, $t(black_t, count_t)$



Safra's Algorithm Example

```
send_i(m):
    $count_i$ ++
    $seq_m \leftarrow seq_i$

recv_i(m, j):
    $count_i$ −−
    if $seq_m > seq_i \lor seq_m = seq_i \land j > i$
      $black_i \leftarrow max_i(black_i, j)$

recv_i(t):
     wait passive_i
     $count_t$ += $count_i$
     $black_i \leftarrow max_i(black_i, black_t)$
     if $black_i \land count_t = 0$: Announce
     else:
       send $black_t \leftarrow max_i(black_i, succ_i)$
       $black_i \leftarrow i$, $count_i \leftarrow 0$
```

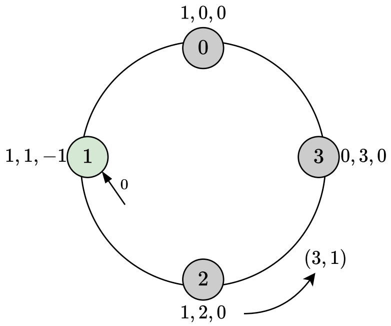
state: $p_i(seq_i, black_i, count_i)$, $t(black_t, count_t)$



Safra's Algorithm Example

```
send_i(m):
    $count_i$ ++
    $seq_m \leftarrow seq_i$

recv_i(m, j):
    $count_i$ −−
    if $seq_m > seq_i \lor seq_m = seq_i \land j > i$
        $black_i \leftarrow max_i(black_i, j)$

recv_i(t):
    wait passive_i
    $count_t$ += $count_i$
    $black_i \leftarrow max_i(black_i, black_t)$
    if $black_i \land count_t = 0$: Announce
    else:
        send $black_t \leftarrow max_i(black_i, succ_i)$
        $black_i \leftarrow i$, $count_i \leftarrow 0$
```

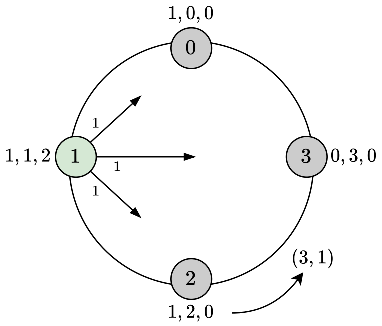
state: $p_i(seq_i, black_i, count_i)$, $t(black_t, count_t)$



Safra's Algorithm Example

```
send_i(m):
    $count_i$ ++
    $seq_m \leftarrow seq_i$

recv_i(m, j):
    $count_i$ −−
    if $seq_m > seq_i \lor seq_m = seq_i \land j > i$
      $black_i \leftarrow max_i(black_i, j)$

recv_i(t):
     wait passive_i
     $count_t$ += $count_i$
     $black_i \leftarrow max_i(black_i, black_t)$
     if $black_i \land count_t = 0$: Announce
     else:
       send $black_t \leftarrow max_i(black_i, succ_i)$
       $black_i \leftarrow i$, $count_i \leftarrow 0$
```

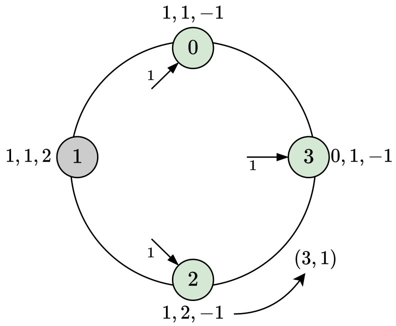
state: $p_i(seq_i, black_i, count_i)$, $t(black_t, count_t)$



Safra's Algorithm Example

```
send_i(m):
    $count_i$ ++
    $seq_m \leftarrow seq_i$

recv_i(m, j):
    $count_i$ −−
    if $seq_m > seq_i \lor seq_m = seq_i \land j > i$
      $black_i \leftarrow max_i(black_i, j)$

recv_i(t):
    wait passive_i
    $count_t$ += $count_i$
    $black_i \leftarrow max_i(black_i, black_t)$
    if $black_i \land count_t = 0$: Announce
    else:
      send $black_t \leftarrow max_i(black_i, succ_i)$
      $black_i \leftarrow i$, $count_i \leftarrow 0$
```

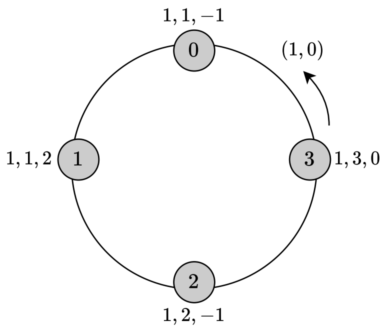
state: $p_i(seq_i, black_i, count_i)$, $t(black_t, count_t)$



Safra's Algorithm Example

```
send_i(m):
    $count_i$ ++
    $seq_m \leftarrow seq_i$

recv_i(m, j):
    $count_i$ −−
    if $seq_m > seq_i \lor seq_m = seq_i \land j > i$
      $black_i \leftarrow max_i(black_i, j)$

recv_i(t):
     wait passive_i
     $count_t$ += $count_i$
     $black_i \leftarrow max_i(black_i, black_t)$
     if $black_i \land count_t = 0$: Announce
     else:
       send $black_t \leftarrow max_i(black_i, succ_i)$
       $black_i \leftarrow i$, $count_i \leftarrow 0$
```

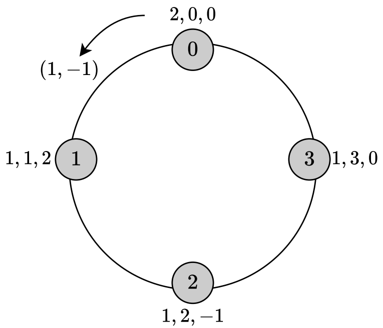
state: $p_i(seq_i, black_i, count_i)$, $t(black_t, count_t)$



Safra's Algorithm Example

```
send_i(m):
    $count_i$ ++
    $seq_m \leftarrow seq_i$

recv_i(m, j):
    $count_i$ −−
    if $seq_m > seq_i \lor seq_m = seq_i \land j > i$
      $black_i \leftarrow max_i(black_i, j)$

recv_i(t):
    wait passive_i
    $count_t$ += $count_i$
    $black_i \leftarrow max_i(black_i, black_t)$
    if $black_i \land count_t = 0$: Announce
    else:
      send $black_t \leftarrow max_i(black_i, succ_i)$
      $black_i \leftarrow i$, $count_i \leftarrow 0$
```

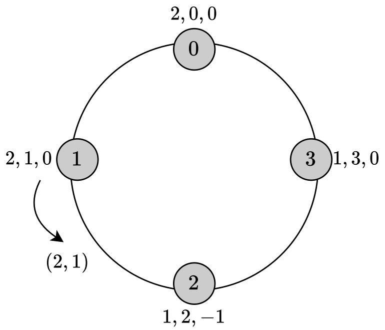
state: $p_i(seq_i, black_i, count_i)$, $t(black_t, count_t)$



Safra's Algorithm Example

```
send_i(m):
    $count_i$ ++
    $seq_m \leftarrow seq_i$

recv_i(m, j):
    $count_i$ −−
    if $seq_m > seq_i \lor seq_m = seq_i \land j > i$
      $black_i \leftarrow max_i(black_i, j)$

recv_i(t):
     wait passive_i
     $count_t$ += $count_i$
     $black_i \leftarrow max_i(black_i, black_t)$
     if $black_i \land count_t = 0$: Announce
     else:
       send $black_t \leftarrow max_i(black_i, succ_i)$
       $black_i \leftarrow i$, $count_i \leftarrow 0$
```

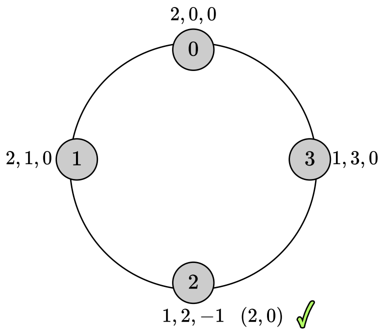
state: $p_i(seq_i, black_i, count_i)$, $t(black_t, count_t)$



Safra's Algorithm Example

```
send_i(m):
    $count_i$ ++
    $seq_m \leftarrow seq_i$

recv_i(m, j):
    $count_i$ −−
    if $seq_m > seq_i \lor seq_m = seq_i \land j > i$
      $black_i \leftarrow max_i(black_i, j)$

recv_i(t):
     wait passive_i
     $count_t$ += $count_i$
     $black_i \leftarrow max_i(black_i, black_t)$
     if $black_i \land count_t = 0$: Announce
     else:
       send $black_t \leftarrow max_i(black_i, succ_i)$
       $black_i \leftarrow i$, $count_i \leftarrow 0$
```

state: $p_i(seq_i, black_i, count_i)$, $t(black_t, count_t)$



Fault-Tolerance

System Model (fault-tolerant)

- Spontaneous and permanent failures
- No Byzantine failures
- Reliable channels
- At all times, $\forall_{i,j} \; \exists \; i \longleftrightarrow j$ channel

Fault-Tolerance

System Model (fault-tolerant)

- Spontaneous and permanent failures
- No Byzantine failures
- Reliable channels
- At all times, $\forall_{i,j} \; \exists \; i \longleftrightarrow j$ channel

- *Perfect* failure detector [Mittal'08]
  - Local. Monitors status of others
  - Impl. by heartbeats and timeouts
  - Introduces *some* synchrony assumptions
  - sync mechanism with async interface

Fault-Tolerance

System Model (fault-tolerant)

- Spontaneous and permanent failures
- No Byzantine failures
- Reliable channels
- At all times, $\forall_{i,j} \; \exists \; i \longleftrightarrow j$ channel

- *Perfect* failure detector [Mittal'08]
  - Local. Monitors status of others
  - Impl. by heartbeats and timeouts
  - Introduces *some* synchrony assumptions
  - sync mechanism with async interface

Termination

1. All **alive** processes are *passive*, **and**
2. For all messages in transit, either the sender or the receiver has crashed

Fault-Tolerant Algorithm

- **Ignore messages to/from crashed processes**
  - Disregard the counts involved, potentially retroactively
- Maintain the ring & keep the token going
  - Each process is responsible for its successor
  - Backup token to *new* successor
- Agreement on the set of crashed processes
  - Must know which counts to accumulate/discard
  - Share local knowledge of crashes through t

Fault-Tolerant Algorithm

- **Ignore messages to/from crashed processes**

  Disregard the counts involved, potentially retroactively

- Maintain the ring & keep the token going

  Each process is responsible for its successor

  Backup token to *new* successor

- Agreement on the set of crashed processes

  Must know which counts to accumulate/discard

  Share local knowledge of crashes through t

- $count_i[N]$: count each process separately
- $count_t[N]$: track contributions separately
- i's contribution $\rightarrow sum(count_i)$
- No in-flight msg $\rightarrow sum(count_t) = 0$

Fault-Tolerant Algorithm

- Ignore messages to/from crashed processes

  Disregard the counts involved, potentially retroactively

- **Maintain the ring & keep the token going**

  Each process is responsible for its successor

  Backup token to *new* successor

- Agreement on the set of crashed processes

  Must know which counts to accumulate/discard

  Share local knowledge of crashes through t

- $count_i[N]$: count each process separately
- $count_t[N]$: track contributions separately
- i's contribution $\rightarrow sum(count_i)$
- No in-flight msg $\rightarrow sum(count_t) = 0$

Fault-Tolerant Algorithm

- Ignore messages to/from crashed processes

  Disregard the counts involved, potentially retroactively

- $count_i[N]$: count each process separately
- $count_t[N]$: track contributions separately
- i's contribution $\rightarrow sum(count_i)$
- No in-flight msg $\rightarrow sum(count_t) = 0$

- **Maintain the ring & keep the token going**

  Each process is responsible for its successor

  Backup token to *new* successor

- Update $succ_i$ to next alive
- Multiple tokens in flight
- seq_t: increment when $succ_i < i$
- Only treat tokens with $seq_t = seq_i + 1$

- Agreement on the set of crashed processes

  Must know which counts to accumulate/discard

  Share local knowledge of crashes through t

Fault-Tolerant Algorithm

- Ignore messages to/from crashed processes

  Disregard the counts involved, potentially retroactively

- Maintain the ring & keep the token going

  Each process is responsible for its successor

  Backup token to *new* successor

- **Agreement on the set of crashed processes**

  Must know which counts to accumulate/discard

  Share local knowledge of crashes through t

- $count_i[N]$: count each process separately
- $count_t[N]$: track contributions separately
- i's contribution $\rightarrow sum(count_i)$
- No in-flight msg $\rightarrow sum(count_t) = 0$

- Update $succ_i$ to next alive
- Multiple tokens in flight
- seq_t: increment when $succ_i < i$
- Only treat tokens with $seq_t = seq_i + 1$

Fault-Tolerant Algorithm

- Ignore messages to/from crashed processes
  - Disregard the counts involved, potentially retroactively

- $count_i[N]$: count each process separately
- $count_t[N]$: track contributions separately
- i's contribution $\to sum(count_i)$
- No in-flight msg $\to sum(count_t) = 0$

- Maintain the ring & keep the token going
  - Each process is responsible for its successor
  - Backup token to *new* successor

- Update $succ_i$ to next alive
- Multiple tokens in flight
- seq_t: increment when $succ_i < i$
- Only treat tokens with $seq_t = seq_i + 1$

- **Agreement on the set of crashed processes**
  - Must know which counts to accumulate/discard
  - Share local knowledge of crashes through t

- $CRASHED_t$: global view
- $CRASHED_i$: (past) local view
- $REPORT_i$: local updates
- On crash: Force t to visit everyone

Fault-Tolerant Algorithm

```
failure$_i$(j):

recv$_i$(t):
```

Fault-Tolerant Algorithm

```
failure_i(j):
    $REPORT_i \leftarrow REPORT_i \cup \{j\}$

recv_i(t):
```

Fault-Tolerant Algorithm

```
failure_i(j):
    $REPORT_i \leftarrow REPORT_i \cup \{j\}$
    if $j = succ_i$:
        $succ_i \leftarrow$ NewSuccessor()
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $black_t \leftarrow i$

recv_i(t):
```

Fault-Tolerant Algorithm

```
failure_i(j):
    $REPORT_i \leftarrow REPORT_i \cup \{j\}$
    if $j = succ_i$:
        $succ_i \leftarrow$ NewSuccessor()
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $black_t \leftarrow i$

recv_i(t):
    if $seq_t \neq seq_i + 1$: return
```

Fault-Tolerant Algorithm

```
**failure$_i$(j):**
    $REPORT_i \leftarrow REPORT_i \cup \{j\}$
    **if** $j = succ_i$**:**
        $succ_i \leftarrow$ NewSuccessor()
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $black_t \leftarrow i$

**recv$_i$(t):**
    **if** $seq_t \neq seq_i + 1$**: return**
    **wait** passive$_i$
```

Fault-Tolerant Algorithm

```
**failure_i(j):**
    $REPORT_i \leftarrow REPORT_i \cup \{j\}$
    **if** $j = succ_i$:
        $succ_i \leftarrow$ NewSuccessor()
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $black_t \leftarrow i$

**recv_i(t):**
    **if** $seq_t \neq seq_i + 1$: **return**
    **wait** passive$_i$
    $CRASHED_t \leftarrow CRASHED_t \setminus CRASHED_i$
```

Fault-Tolerant Algorithm

```
failure_i(j):
    $REPORT_i \leftarrow REPORT_i \cup \{j\}$
    if $j = succ_i$:
        $succ_i \leftarrow$ NewSuccessor()
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $black_t \leftarrow i$

recv_i(t):
    if $seq_t \neq seq_i + 1$: return
    wait passive_i
    $CRASHED_t \leftarrow CRASHED_t \setminus CRASHED_i$
    $CRASHED_i \leftarrow CRASHED_i \cup CRASHED_t$
```

Fault-Tolerant Algorithm

```
failure_i(j):
    $REPORT_i \leftarrow REPORT_i \cup \{j\}$
    if $j = succ_i$:
        $succ_i \leftarrow$ NewSuccessor()
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $black_t \leftarrow i$

recv_i(t):
    if $seq_t \neq seq_i + 1$: return
    wait passive_i
    $CRASHED_t \leftarrow CRASHED_t \setminus CRASHED_i$
    $CRASHED_i \leftarrow CRASHED_i \cup CRASHED_t$
    $REPORT_i \leftarrow REPORT_i \setminus CRASHED_t$
```

Fault-Tolerant Algorithm

```
**failure$_i$(j):**
    $REPORT_i \leftarrow REPORT_i \cup \{j\}$
    **if** $j = succ_i$:
        $succ_i \leftarrow$ NewSuccessor()
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $black_t \leftarrow i$

**recv$_i$(t):**
    **if** $seq_t \neq seq_i + 1$: **return**
    **wait** passive$_i$
    $CRASHED_t \leftarrow CRASHED_t \setminus CRASHED_i$
    $CRASHED_i \leftarrow CRASHED_i \cup CRASHED_t$
    $REPORT_i \leftarrow REPORT_i \setminus CRASHED_t$
    $count_t^i \leftarrow \sum_{j \notin CRASHED_i} count_i^j$
```

Fault-Tolerant Algorithm

```
**failure$_i$(j):**
    $REPORT_i \leftarrow REPORT_i \cup \{j\}$
    **if** $j = succ_i$**:**
        $succ_i \leftarrow$ NewSuccessor()
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $black_t \leftarrow i$

**recv$_i$(t):**
    **if** $seq_t \neq seq_i + 1$**: return**
    **wait** passive$_i$
    $CRASHED_t \leftarrow CRASHED_t \setminus CRASHED_i$
    $CRASHED_i \leftarrow CRASHED_i \cup CRASHED_t$
    $REPORT_i \leftarrow REPORT_i \setminus CRASHED_t$
    $count_t^i \leftarrow \sum_{j \notin CRASHED_i} count_i^j$
    **if** $\boxed{black_i = i}$**:**
        $sum_i \leftarrow \sum_{j \notin CRASHED_i} count_t^j$
        **if** $\boxed{sum_i = 0}$**: Announce**
```

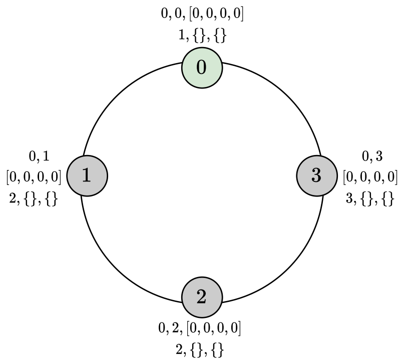
Fault-Tolerant Algorithm

```
failure_i(j):
    $REPORT_i \leftarrow REPORT_i \cup \{j\}$
    if $j = succ_i$:
        $succ_i \leftarrow$ NewSuccessor()
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $black_t \leftarrow i$

recv_i(t):
    if $seq_t \neq seq_i + 1$: return
    wait passive$_i$
    $CRASHED_t \leftarrow CRASHED_t \setminus CRASHED_i$
    $CRASHED_i \leftarrow CRASHED_i \cup CRASHED_t$
    $REPORT_i \leftarrow REPORT_i \setminus CRASHED_t$
    $count_t^i \leftarrow \sum_{j \notin CRASHED_i} count_i^j$
    if $\boxed{black_i = i}$:
        $sum_i \leftarrow \sum_{j \notin CRASHED_i} count_t^j$
        if $\boxed{sum_i = 0}$: Announce
    if $REPORT_i \neq \emptyset$:
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $CRASHED_i \leftarrow CRASHED_i \cup REPORT_i$
        $REPORT_i \leftarrow \emptyset$
        $black_t \leftarrow i$
    else:
        $black_t \leftarrow max_i(black_i, succ_i)$
```

Fault-Tolerant Algorithm Example

state: $p_i(seq_i, black_i, count_i, succ_i, CRASHED_i, REPORT_i)$,
$t(seq_t, black_t, count_t, CRASHED_t)$

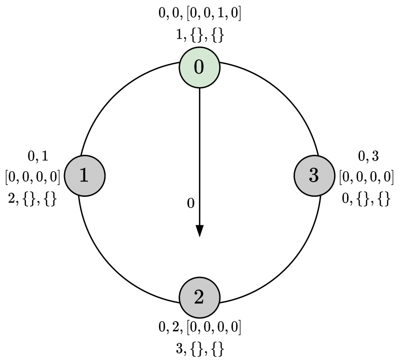
```
failure_i(j):
    $REPORT_i \leftarrow REPORT_i \cup \{j\}$
    if $j = succ_i$:
        $succ_i \leftarrow$ NewSuccessor()
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $black_t \leftarrow i$

recv_i(t):
    if $seq_t \neq seq_i + 1$: return
    wait passive_i
    $CRASHED_t \leftarrow CRASHED_t \setminus CRASHED_i$
    $CRASHED_i \leftarrow CRASHED_i \cup CRASHED_t$
    $REPORT_i \leftarrow REPORT_i \setminus CRASHED_t$
    $count_t^i \leftarrow \sum_{j \notin CRASHED_i} count_i^j$
    if $black_i = i$:
        $sum_i \leftarrow \sum_{j \notin CRASHED_i} count_t^j$
        if $sum_i = 0$: Announce
    if $REPORT_i \neq \emptyset$:
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $CRASHED_i \leftarrow CRASHED_i \cup REPORT_i$
        $REPORT_i \leftarrow \emptyset$
        $black_t \leftarrow i$
    else:
        $black_t \leftarrow max_i(black_i, succ_i)$
```



Fault-Tolerant Algorithm Example

state: $p_i(seq_i, black_i, count_i, succ_i, CRASHED_i, REPORT_i)$, $t(seq_t, black_t, count_t, CRASHED_t)$

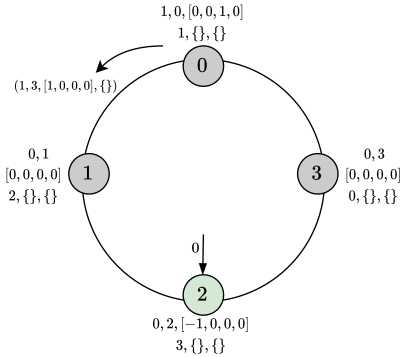
```
failure_i(j):
    $REPORT_i \leftarrow REPORT_i \cup \{j\}$
    if $j = succ_i$:
        $succ_i \leftarrow$ NewSuccessor()
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $black_t \leftarrow i$

recv_i(t):
    if $seq_t \neq seq_i + 1$: return
    wait passive_i
    $CRASHED_t \leftarrow CRASHED_t \setminus CRASHED_i$
    $CRASHED_i \leftarrow CRASHED_i \cup CRASHED_t$
    $REPORT_i \leftarrow REPORT_i \setminus CRASHED_t$
    $count_t^i \leftarrow \sum_{j \notin CRASHED_i} count_i^j$
    if $black_i = i$:
        $sum_i \leftarrow \sum_{j \notin CRASHED_i} count_t^j$
        if $sum_i = 0$: Announce
    if $REPORT_i \neq \emptyset$:
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $CRASHED_i \leftarrow CRASHED_i \cup REPORT_i$
        $REPORT_i \leftarrow \emptyset$
        $black_t \leftarrow i$
    else:
        $black_t \leftarrow max_i(black_i, succ_i)$
```



Fault-Tolerant Algorithm Example

state: $p_i(seq_i, black_i, count_i, succ_i, CRASHED_i, REPORT_i)$, $t(seq_t, black_t, count_t, CRASHED_t)$

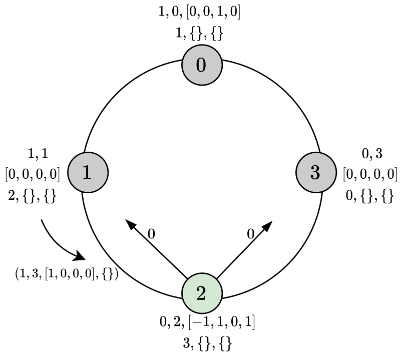
```
failure_i(j):
    $REPORT_i \leftarrow REPORT_i \cup \{j\}$
    if $j = succ_i$:
        $succ_i \leftarrow$ NewSuccessor()
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $black_t \leftarrow i$

recv_i(t):
    if $seq_t \neq seq_i + 1$: return
    wait passive$_i$
    $CRASHED_t \leftarrow CRASHED_t \setminus CRASHED_i$
    $CRASHED_i \leftarrow CRASHED_i \cup CRASHED_t$
    $REPORT_i \leftarrow REPORT_i \setminus CRASHED_t$
    $count_t^i \leftarrow \sum_{j \notin CRASHED_i} count_i^j$
    if $\boxed{black_i = i}$:
        $sum_i \leftarrow \sum_{j \notin CRASHED_i} count_t^j$
        if $\boxed{sum_i = 0}$: Announce
    if $REPORT_i \neq \emptyset$:
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $CRASHED_i \leftarrow CRASHED_i \cup REPORT_i$
        $REPORT_i \leftarrow \emptyset$
        $black_t \leftarrow i$
    else:
        $black_t \leftarrow max_i(black_i, succ_i)$
```



Fault-Tolerant Algorithm Example

state: $p_i(seq_i, black_i, count_i, succ_i, CRASHED_i, REPORT_i)$,
$t(seq_t, black_t, count_t, CRASHED_t)$

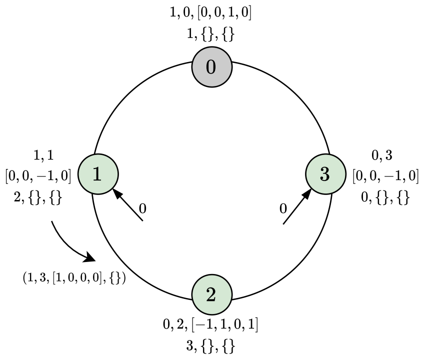
```
failure_i(j):
    $REPORT_i \leftarrow REPORT_i \cup \{j\}$
    if $j = succ_i$:
        $succ_i \leftarrow$ NewSuccessor()
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $black_t \leftarrow i$

recv_i(t):
    if $seq_t \neq seq_i + 1$: return
    wait passive_i
    $CRASHED_t \leftarrow CRASHED_t \setminus CRASHED_i$
    $CRASHED_i \leftarrow CRASHED_i \cup CRASHED_t$
    $REPORT_i \leftarrow REPORT_i \setminus CRASHED_t$
    $count_t^i \leftarrow \sum_{j \notin CRASHED_i} count_i^j$
    if $black_i = i$:
        $sum_i \leftarrow \sum_{j \notin CRASHED_i} count_t^j$
        if $sum_i = 0$: Announce
    if $REPORT_i \neq \emptyset$:
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $CRASHED_i \leftarrow CRASHED_i \cup REPORT_i$
        $REPORT_i \leftarrow \emptyset$
        $black_t \leftarrow i$
    else:
        $black_t \leftarrow max_i(black_i, succ_i)$
```



Fault-Tolerant Algorithm Example

state: $p_i(seq_i, black_i, count_i, succ_i, CRASHED_i, REPORT_i)$,
$t(seq_t, black_t, count_t, CRASHED_t)$

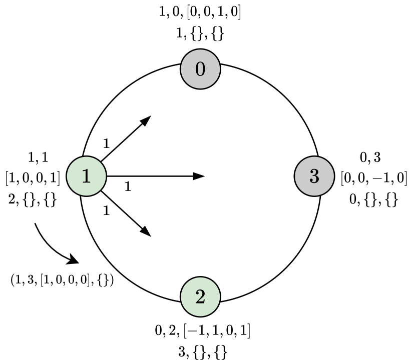
```
failure_i(j):
    $REPORT_i \leftarrow REPORT_i \cup \{j\}$
    if $j = succ_i$:
        $succ_i \leftarrow$ NewSuccessor()
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $black_t \leftarrow i$

recv_i(t):
    if $seq_t \neq seq_i + 1$: return
    wait passive_i
    $CRASHED_t \leftarrow CRASHED_t \setminus CRASHED_i$
    $CRASHED_i \leftarrow CRASHED_i \cup CRASHED_t$
    $REPORT_i \leftarrow REPORT_i \setminus CRASHED_t$
    $count_t^i \leftarrow \sum_{j \notin CRASHED_i} count_i^j$
    if $\boxed{black_i = i}$:
        $sum_i \leftarrow \sum_{j \notin CRASHED_i} count_t^j$
        if $\boxed{sum_i = 0}$: Announce
    if $REPORT_i \neq \emptyset$:
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $CRASHED_i \leftarrow CRASHED_i \cup REPORT_i$
        $REPORT_i \leftarrow \emptyset$
        $black_t \leftarrow i$
    else:
        $black_t \leftarrow max_i(black_i, succ_i)$
```



Fault-Tolerant Algorithm Example

state: $p_i(seq_i, black_i, count_i, succ_i, CRASHED_i, REPORT_i)$,
$t(seq_t, black_t, count_t, CRASHED_t)$

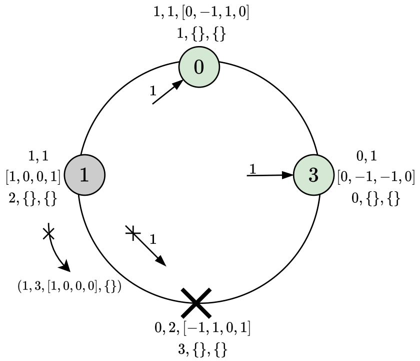
```
failure_i(j):
    $REPORT_i \leftarrow REPORT_i \cup \{j\}$
    if $j = succ_i$:
        $succ_i \leftarrow$ NewSuccessor()
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $black_t \leftarrow i$

recv_i(t):
    if $seq_t \neq seq_i + 1$: return
    wait passive_i
    $CRASHED_t \leftarrow CRASHED_t \setminus CRASHED_i$
    $CRASHED_i \leftarrow CRASHED_i \cup CRASHED_t$
    $REPORT_i \leftarrow REPORT_i \setminus CRASHED_t$
    $count_t^i \leftarrow \sum_{j \notin CRASHED_i} count_i^j$
    if $black_i = i$:
        $sum_i \leftarrow \sum_{j \notin CRASHED_i} count_t^j$
        if $sum_i = 0$: Announce
    if $REPORT_i \neq \emptyset$:
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $CRASHED_i \leftarrow CRASHED_i \cup REPORT_i$
        $REPORT_i \leftarrow \emptyset$
        $black_t \leftarrow i$
    else:
        $black_t \leftarrow max_i(black_i, succ_i)$
```



Fault-Tolerant Algorithm Example

state: $p_i(seq_i, black_i, count_i, succ_i, CRASHED_i, REPORT_i)$, $t(seq_t, black_t, count_t, CRASHED_t)$

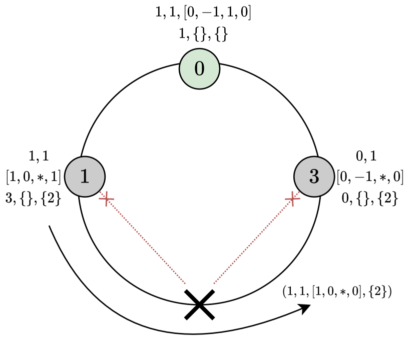
```
failure_i(j):
    $REPORT_i \leftarrow REPORT_i \cup \{j\}$
    if $j = succ_i$:
        $succ_i \leftarrow$ NewSuccessor()
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $black_t \leftarrow i$

recv_i(t):
    if $seq_t \neq seq_i + 1$: return
    wait passive_i
    $CRASHED_t \leftarrow CRASHED_t \setminus CRASHED_i$
    $CRASHED_i \leftarrow CRASHED_i \cup CRASHED_t$
    $REPORT_i \leftarrow REPORT_i \setminus CRASHED_t$
    $count_t^i \leftarrow \sum_{j \notin CRASHED_i} count_i^j$
    if $black_i = i$:
        $sum_i \leftarrow \sum_{j \notin CRASHED_i} count_t^j$
        if $sum_i = 0$: Announce
    if $REPORT_i \neq \emptyset$:
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $CRASHED_i \leftarrow CRASHED_i \cup REPORT_i$
        $REPORT_i \leftarrow \emptyset$
        $black_t \leftarrow i$
    else:
        $black_t \leftarrow max_i(black_i, succ_i)$
```



Fault-Tolerant Algorithm Example

state: $p_i(seq_i, black_i, count_i, succ_i, CRASHED_i, REPORT_i)$,
$t(seq_t, black_t, count_t, CRASHED_t)$

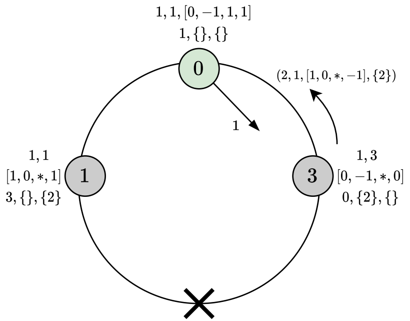
```
failure_i(j):
    $REPORT_i \leftarrow REPORT_i \cup \{j\}$
    if $j = succ_i$:
        $succ_i \leftarrow$ NewSuccessor()
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $black_t \leftarrow i$

recv_i(t):
    if $seq_t \neq seq_i + 1$: return
    wait passive_i
    $CRASHED_t \leftarrow CRASHED_t \setminus CRASHED_i$
    $CRASHED_i \leftarrow CRASHED_i \cup CRASHED_t$
    $REPORT_i \leftarrow REPORT_i \setminus CRASHED_t$
    $count_t^i \leftarrow \sum_{j \notin CRASHED_i} count_i^j$
    if $\boxed{black_i = i}$:
        $sum_i \leftarrow \sum_{j \notin CRASHED_i} count_t^j$
        if $\boxed{sum_i = 0}$: Announce
    if $REPORT_i \neq \emptyset$:
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $CRASHED_i \leftarrow CRASHED_i \cup REPORT_i$
        $REPORT_i \leftarrow \emptyset$
        $black_t \leftarrow i$
    else:
        $black_t \leftarrow max_i(black_i, succ_i)$
```



Fault-Tolerant Algorithm Example

```
failure_i(j):
    $REPORT_i \leftarrow REPORT_i \cup \{j\}$
    if $j = succ_i$:
        $succ_i \leftarrow$ NewSuccessor()
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $black_t \leftarrow i$

recv_i(t):
    if $seq_t \neq seq_i + 1$: return
    wait passive_i
    $CRASHED_t \leftarrow CRASHED_t \setminus CRASHED_i$
    $CRASHED_i \leftarrow CRASHED_i \cup CRASHED_t$
    $REPORT_i \leftarrow REPORT_i \setminus CRASHED_t$
    $count_t^i \leftarrow \sum_{j \notin CRASHED_i} count_i^j$
    if $black_i = i$:
        $sum_i \leftarrow \sum_{j \notin CRASHED_i} count_t^j$
        if $sum_i = 0$: Announce
    if $REPORT_i \neq \emptyset$:
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $CRASHED_i \leftarrow CRASHED_i \cup REPORT_i$
        $REPORT_i \leftarrow \emptyset$
        $black_t \leftarrow i$
    else:
        $black_t \leftarrow max_i(black_i, succ_i)$
```

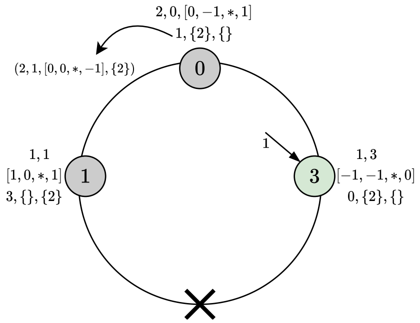
state: $p_i(seq_i, black_i, count_i, succ_i, CRASHED_i, REPORT_i)$, $t(seq_t, black_t, count_t, CRASHED_t)$



Fault-Tolerant Algorithm Example

state: $p_i(seq_i, black_i, count_i, succ_i, CRASHED_i, REPORT_i)$,
$t(seq_t, black_t, count_t, CRASHED_t)$

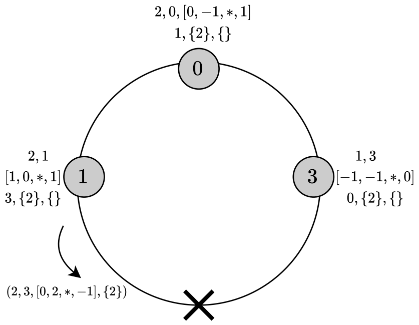
```
failure_i(j):
    $REPORT_i \leftarrow REPORT_i \cup \{j\}$
    if $j = succ_i$:
        $succ_i \leftarrow$ NewSuccessor()
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $black_t \leftarrow i$

recv_i(t):
    if $seq_t \neq seq_i + 1$: return
    wait passive_i
    $CRASHED_t \leftarrow CRASHED_t \setminus CRASHED_i$
    $CRASHED_i \leftarrow CRASHED_i \cup CRASHED_t$
    $REPORT_i \leftarrow REPORT_i \setminus CRASHED_t$
    $count_t^i \leftarrow \sum_{j \notin CRASHED_i} count_i^j$
    if $black_i = i$:
        $sum_i \leftarrow \sum_{j \notin CRASHED_i} count_t^j$
        if $sum_i = 0$: Announce
    if $REPORT_i \neq \emptyset$:
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $CRASHED_i \leftarrow CRASHED_i \cup REPORT_i$
        $REPORT_i \leftarrow \emptyset$
        $black_t \leftarrow i$
    else:
        $black_t \leftarrow max_i(black_i, succ_i)$
```



Fault-Tolerant Algorithm Example

state: $p_i(seq_i, black_i, count_i, succ_i, CRASHED_i, REPORT_i)$,
$t(seq_t, black_t, count_t, CRASHED_t)$

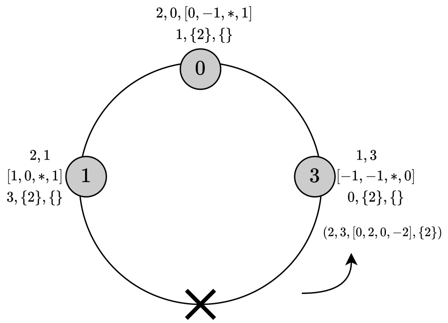
```
failure_i(j):
    $REPORT_i \leftarrow REPORT_i \cup \{j\}$
    if $j = succ_i$:
        $succ_i \leftarrow$ NewSuccessor()
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $black_t \leftarrow i$

recv_i(t):
    if $seq_t \neq seq_i + 1$: return
    wait passive_i
    $CRASHED_t \leftarrow CRASHED_t \setminus CRASHED_i$
    $CRASHED_i \leftarrow CRASHED_i \cup CRASHED_t$
    $REPORT_i \leftarrow REPORT_i \setminus CRASHED_t$
    $count_t^i \leftarrow \sum_{j \notin CRASHED_i} count_i^j$
    if $\boxed{black_i = i}$:
        $sum_i \leftarrow \sum_{j \notin CRASHED_i} count_t^j$
        if $\boxed{sum_i = 0}$: Announce
    if $REPORT_i \neq \emptyset$:
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $CRASHED_i \leftarrow CRASHED_i \cup REPORT_i$
        $REPORT_i \leftarrow \emptyset$
        $black_t \leftarrow i$
    else:
        $black_t \leftarrow max_i(black_i, succ_i)$
```



Fault-Tolerant Algorithm Example

state: $p_i(seq_i, black_i, count_i, succ_i, CRASHED_i, REPORT_i)$,
$t(seq_t, black_t, count_t, CRASHED_t)$

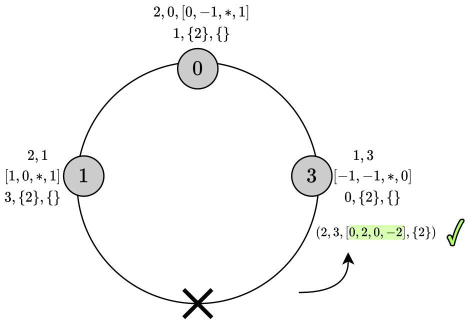
```
failure_i(j):
    $REPORT_i \leftarrow REPORT_i \cup \{j\}$
    if $j = succ_i$:
        $succ_i \leftarrow$ NewSuccessor()
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $black_t \leftarrow i$

recv_i(t):
    if $seq_t \neq seq_i + 1$: return
    wait passive_i
    $CRASHED_t \leftarrow CRASHED_t \setminus CRASHED_i$
    $CRASHED_i \leftarrow CRASHED_i \cup CRASHED_t$
    $REPORT_i \leftarrow REPORT_i \setminus CRASHED_t$
    $count_t^i \leftarrow \sum_{j \notin CRASHED_i} count_i^j$
    if $\boxed{black_i = i}$:
        $sum_i \leftarrow \sum_{j \notin CRASHED_i} count_t^j$
        if $\boxed{sum_i = 0}$: Announce
    if $REPORT_i \neq \emptyset$:
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $CRASHED_i \leftarrow CRASHED_i \cup REPORT_i$
        $REPORT_i \leftarrow \emptyset$
        $black_t \leftarrow i$
    else:
        $black_t \leftarrow max_i(black_i, succ_i)$
```



Fault-Tolerant Algorithm Example

```
failure_i(j):
    $REPORT_i \leftarrow REPORT_i \cup \{j\}$
    if $j = succ_i$:
        $succ_i \leftarrow$ NewSuccessor()
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $black_t \leftarrow i$

recv_i(t):
    if $seq_t \neq seq_i + 1$: return
    wait passive_i
    $CRASHED_t \leftarrow CRASHED_t \setminus CRASHED_i$
    $CRASHED_i \leftarrow CRASHED_i \cup CRASHED_t$
    $REPORT_i \leftarrow REPORT_i \setminus CRASHED_t$
    $count_t^i \leftarrow \sum_{j \notin CRASHED_i} count_i^j$
    if $black_i = i$:
        $sum_i \leftarrow \sum_{j \notin CRASHED_i} count_t^j$
        if $sum_i = 0$: Announce
    if $REPORT_i \neq \emptyset$:
        $CRASHED_t \leftarrow CRASHED_t \cup REPORT_i$
        $CRASHED_i \leftarrow CRASHED_i \cup REPORT_i$
        $REPORT_i \leftarrow \emptyset$
        $black_t \leftarrow i$
    else:
        $black_t \leftarrow max_i(black_i, succ_i)$
```

state: $p_i(seq_i, black_i, count_i, succ_i, CRASHED_i, REPORT_i)$,
$t(seq_t, black_t, count_t, CRASHED_t)$



Fault-Tolerance Cost

1. $\mathcal{O}(N)$ token size
   - Potentially lower impact because only passive nodes forward it
   - Stable storage assumption $\rightarrow \mathcal{O}(1)$ token size

Fault-Tolerance Cost

1. $\mathcal{O}(N)$ token size
    - Potentially lower impact because only passive nodes forward it
    - Stable storage assumption $\rightarrow \mathcal{O}(1)$ token size
2. New crash $\rightarrow +1$ round
    - Make sure everybody knows
    - Multiple crashes in the same round $\rightarrow$ extra rounds overlap

Fault-Tolerance Cost

1. $\mathcal{O}(N)$ token size
   - Potentially lower impact because only passive nodes forward it
   - Stable storage assumption $\rightarrow \mathcal{O}(1)$ token size

2. New crash $\rightarrow +1$ round
   - Make sure everybody knows
   - Multiple crashes in the same round $\rightarrow$ extra rounds overlap

3. Failure detector messages
   - Mandatory in FT algorithms
   - FD monitors only succ/pred $\rightarrow$ Lower #heartbeats but slower convergence

Experiments - Setup

- Emulation
    - Basic Algorithm:
        - Send message $\rightarrow$ wait $+$ receiver stub
        - Compute $\rightarrow$ sleep
    - Randomize #activities, message delays, crashing, who and when to crash (uniform, gaussian)

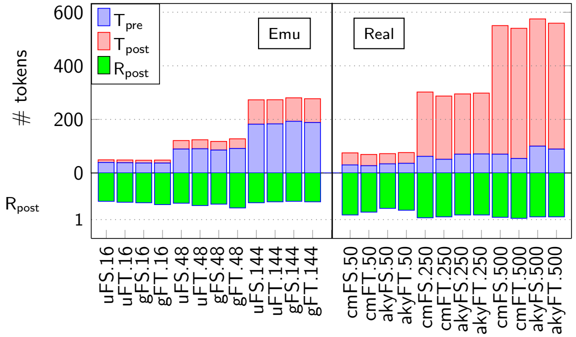
Experiments - Setup

- Emulation
  - Basic Algorithm:
    - Send message $\rightarrow$ wait + receiver stub
    - Compute $\rightarrow$ sleep
  - Randomize #activities, message delays, crashing, who and when to crash (uniform, gaussian)
- On top of two distributed algorithms:
  - Chandy-Mishra routing **[Chandy'82]**
  - Afek-Kutten-Yung self-stabilizing spanning-tree **[Afek'97]**
  - Run on DAS-5 **[Bal'16]**

Experiments - Results (1)

Fault-free runs

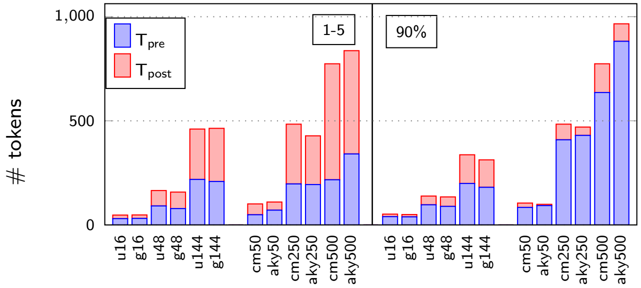
- No overhead compared to FS
- Detection within 1 round



Experiments - Results (2)

Faulty-runs

- Total tokens $< 2N$
- Increase with more crashes
- Influenced by the basic algorithm



Conclusion

- Fault-tolerant variant of Safra's Termination Detection algorithm
- $\Theta(N)$ increase in token size
  - Future work: use stable storage for the per process counters $\rightarrow \mathcal{O}(1)$ token size
- $+1$ round when a process crashes
  - extra rounds for multiple crashes on the same round overlap
- Can tolerate N - 1 failures
- No overhead in the absence of faults (Single-round detection)

`https://github.com/gkarlos/FTSEmu`

`https://github.com/PerFuchs/safra-termination-detection-fault-tolerant`