
# Contention-Related Crash Failures

**Anaïs Durand**

LIP6, Sorbonne Université, Paris

April 1st, 2019

# Set Agreement and Renaming in the Presence of Contention-Related Crash Failures

*SSS 2018*

Joint work with:

**Michel Raynal**

**Gadi Taubenfeld**

# Computational Model

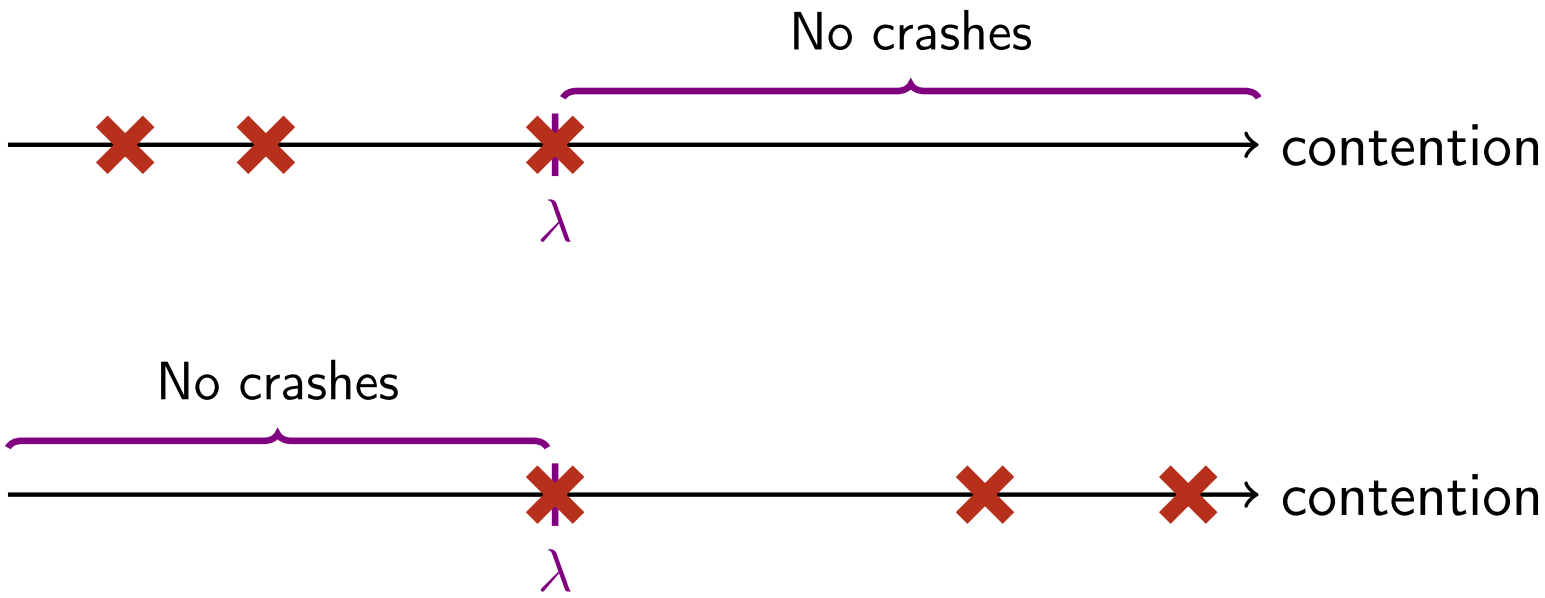
- Asynchronous deterministic system
- $n$ processes $p_1, \ldots, p_n$
- Atomic read/write registers
- $0 \leq t < n$ process crashes
- Participation required

# Process crashes

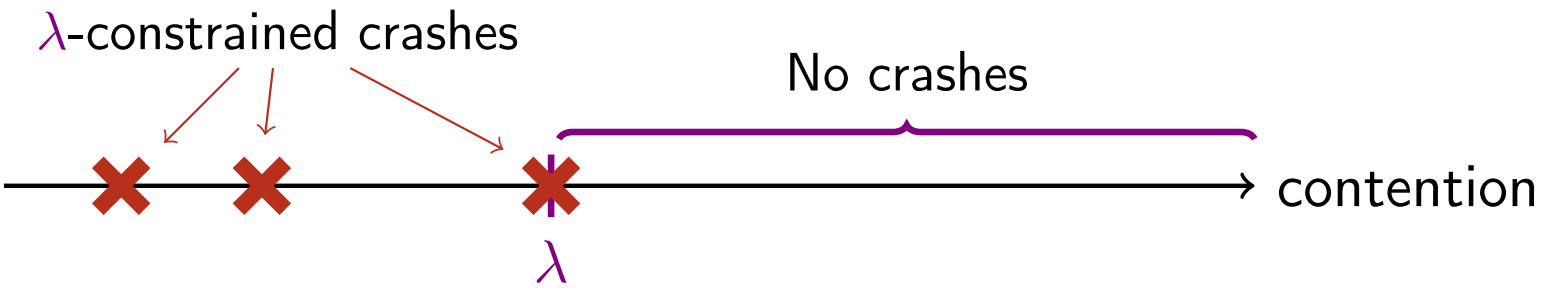
2 kinds of process crashes usually considered:

- **Initially dead** processes
- **"Classical"** (**any-time**) crashs: no constraints

- **Contention** = # processes that accessed a shared register
  $\approx$ # processes that started to compute
- $\lambda$ = predefined contention threshold
- 2 possible definitions:



- **Contention** = # processes that accessed a shared register
  $\approx$ # processes that started to compute
- $\lambda$ = predefined contention threshold
- 2 possible definitions:



# Contention-Related *vs.* Any-Time Crash Failures

- **Consensus:**
  - [Fischer *et al.*, 85]: **Impossible** with **one any-time** crash failure.
  - [Taubenfeld, 18]: Algorithm that tolerates **one** $(n-1)$**-constrained** crash failure for $n > 1$.

- $k$**-Set Agreement,** $1 \leq k < n$**:**
  - [Borowsky, Gafni, 93]: **Impossible** with $k$ **any-time** crash failures.
  - [Taubenfeld, 18]: Algorithm that tolerates $\ell + k - 2$ $(n-\ell)$**-constrained** crash failures for $\ell \geq 1$ and $n \geq 2\ell + k - 2$.

# Motivation

Consider a problem $P$ that can be solved with $t$ any-time crash failures, but impossible with $t + 1$ any-time crash failures.

Given $\lambda$, can $P$ be solved with both

$t_1$ **$\lambda$-constrained**

**and**

$t_2 \leq t$ **any-time**

crash failures, with $t_1 + t_2 > t$?

We consider here: **$k$-set agreement** (for $k \geq 2$) and **renaming**

# $k$-Set Agreement

## Definition

- One-shot object
- Operation *propose*($v$): propose value $v$ and return a decided value
- Properties:
  - **Validity:** decided value $\in$ proposed values
  - **Agreement:** $\leq k$ decided values
  - **Termination:** every correct process decides

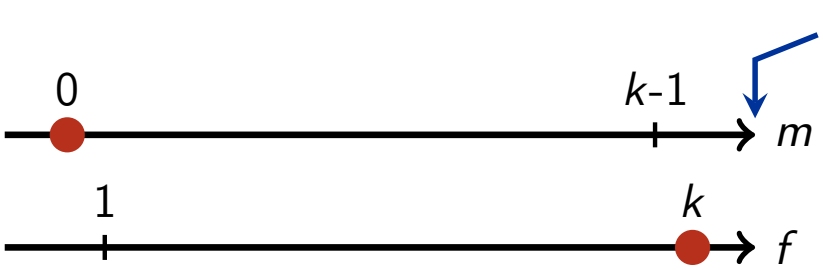
# $k$-Set Agreement Algorithm: Properties

- $\lambda = n - k$
- $k \geq 2$
- $k = m + f,\ m \geq 0,\ f \geq 1$

| | |
|---|---|
| **total # of faults** | $t = 2m + f - 1 = k + m - 1$ |
| **$\lambda$-constrained crashes** | $2m$ |
| **any-time crashes** | $f - 1$ |

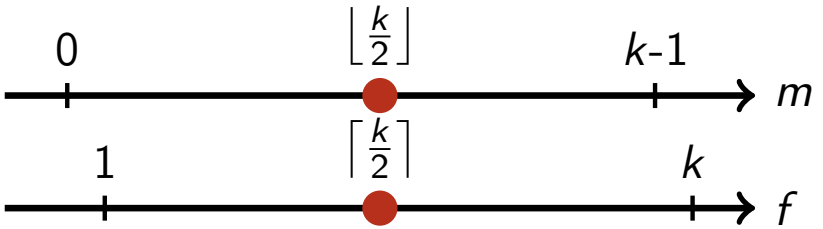
[Borowsky, Gafni, 93]: Impossible with $k$ any-time crash failures.

Parameters $f$ and $m$ allow the user to **tune** the proportion of each type of crash failures.

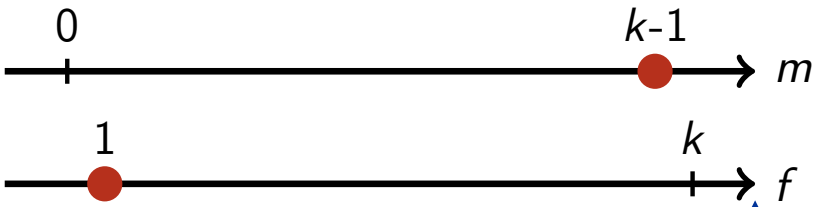



max #any-time ($= k$-1)

$t = k - 1$




$t = 2\lfloor \frac{k}{2} \rfloor + \lceil \frac{k}{2} \rceil - 1$




$t = 2k - 2$

max #$\lambda$-constrained ($= 2k$-2)

# $k$-Set Agreement: Shared Registers (1/2)

- $DEC$: atomic register, initially $\perp$

- $PART[1 \dots n]$: snapshot object, initially $[\texttt{down}, \dots, \texttt{down}]$

  - Atomic (linearizable) operations $write()$ and $snapshot()$
  - $\approx$ array of single-writer multi-reader atomic registers $PART[1 \dots n]$ such that:
    - $p_i$ invokes $write(v)$ = writes $v$ into $PART[i]$
    - $p_i$ invokes $snapshot()$ = obtains the value of the array $PART[1 \dots n]$ as if it read simultaneously and instantaneously all its entries

# $k$-Set Agreement: Shared Registers (2/2)

- *MUTEX*[1]: one-shot deadlock-free $f$-mutex

- *MUTEX*[2]: one-shot deadlock-free $m$-mutex

  - Operations *acquire*() and *release*() (invoked at most once)
  - Properties:
    - Mutual exclusion: $\leq m$ processes simultaneously in critical section
    - Deadlock-freedom: if $< m$ processes crashes, then $\geq 1$ process invoking *acquire*() terminates its invocation

# $k$-Set Agreement Algorithm (1/2)

**operation** $propose(in_i)$ **is**

(1) $PART.write(\texttt{up})$; % *signal participation*

# $k$-Set Agreement Algorithm (1/2)

**operation** $propose(in_i)$ **is**

(1) $PART.write(\mathtt{up})$; % *signal participation*

(2) **repeat**
(3) $part_i := PART.snapshot()$; % *wait for* $n - t$
(4) $count_i := |\{x \text{ such that } part_i[x] = \mathtt{up}\}|$; % *participants*
(5) **until** $count_i \geq n - t$ **end repeat**;

# $k$-Set Agreement Algorithm (1/2)

**operation** $propose(in_i)$ **is**

(1) $PART.write(\mathtt{up})$; % *signal participation*

(2) **repeat**
(3) $\quad part_i := PART.snapshot()$; % *wait for* $n - t$
(4) $\quad count_i := |\{x$ such that $part_i[x] = \mathtt{up}\}|$; % *participants*
(5) **until** $count_i \geq n - t$ **end repeat**;

(6) **if** $count_i \leq \lambda$ **then** % *split processes into groups*
(7) $\quad group_i := 2$; % $\rightsquigarrow$ *MUTEX*[2] *(m-mutex)*
(8) **else**
(9) $\quad group_i := 1$; % $\rightsquigarrow$ *MUTEX*[1] *(f-mutex)*
(10) **end if**

# $k$-Set Agreement Algorithm (1/2)

**operation** $propose(in_i)$ **is**

(1) $PART.write(\mathtt{up})$; % *signal participation*

(2) **repeat**
(3) $part_i := PART.snapshot()$; % *wait for* $n - t$
(4) $count_i := |\{x \text{ such that } part_i[x] = \mathtt{up}\}|$; % *participants*
(5) **until** $count_i \geq n - t$ **end repeat**;

(6) **if** $count_i \leq \lambda$ **then** % *split processes into groups*
(7) $group_i := 2$; % $\rightsquigarrow$ *MUTEX*[2] *(m-mutex)*
(8) **else**
(9) $group_i := 1$; % $\rightsquigarrow$ *MUTEX*[1] *(f-mutex)*
(10) **end if**

(11) launch in // the threads $T_1$ and $T_2$;

# $k$-Set Agreement Algorithm (2/2)

```
thread T_1 is
(12)   loop forever
(13)       if DEC ≠ ⊥ then
(14)           return(DEC);
(15)       end if;
(16)   end loop;
```

*% wait for a decided value*

# $k$-Set Agreement Algorithm (2/2)

```
thread T1 is                                   % wait for a decided value
(12)  loop forever
(13)      if DEC ≠ ⊥ then
(14)          return(DEC);
(15)      end if;
(16)  end loop;
```

```
thread T2 is                                   % decide a value if enters its CS
(17)  if group_i = 1 ∨ m > 0 then
(18)      MUTEX[group_i].acquire();
(19)          if DEC = ⊥ then
(20)              DEC := in_i;
(21)          end if
(22)      MUTEX[group_i].release();
(23)      return(DEC);
(24)  end if;
```

# $k$-Set Agreement Algorithm: Validity & Agreement

```
thread T1 is
(12)  loop forever
(13)      if DEC ≠ ⊥ then
(14)          return(DEC);
(15)      end if;
(16)  end loop;

thread T2 is
(17)  if group_i = 1 ∨ m > 0 then
(18)      MUTEX[group_i].acquire();
(19)          if DEC = ⊥ then
(20)              DEC := in_i;
(21)          end if
(22)      MUTEX[group_i].release();
(23)      return(DEC);
(24)  end if;
```

a Decided value = $DEC$

# $k$-Set Agreement Algorithm: Validity & Agreement

```
thread T_1 is
(12)  loop forever
(13)      if DEC ≠ ⊥ then
(14)          return(DEC);
(15)      end if;
(16)  end loop;

thread T_2 is
(17)  if group_i = 1 ∨ m > 0 then
(18)      MUTEX[group_i].acquire();
(19)          if DEC = ⊥ then
(20)              DEC := in_i;
(21)          end if
(22)      MUTEX[group_i].release();
(23)      return(DEC);
(24)  end if;
```

- **a** Decided value = $DEC$
- **b** $DEC$ assigned to **proposed** values $in_i$ in CS

# $k$-Set Agreement Algorithm: Validity & Agreement

```
thread T_1 is
(12)  loop forever
(13)      if DEC ≠ ⊥ then
(14)          return(DEC);
(15)      end if;
(16)  end loop;

thread T_2 is
(17)  if group_i = 1 ∨ m > 0 then
(18)      MUTEX[group_i].acquire();
(19)          if DEC = ⊥ then
(20)              DEC := in_i;
(21)          end if
(22)      MUTEX[group_i].release();
(23)      return(DEC);
(24)  end if;
```

**a** Decided value = $DEC$

**b** $DEC$ assigned to **proposed** values $in_i$ in CS

**c** $MUTEX[1] \rightsquigarrow \leq f \neq$ values
$MUTEX[2] \rightsquigarrow \leq m \neq$ values

$\Rightarrow \leq f + m = k$ **decided values**

# $k$-Set Agreement Algorithm: Termination (1/5)

```
(1)  PART.write(up);
(2)  repeat
(3)      part_i := PART.snapshot();
(4)      count_i := |{x such that part_i[x] = up}|;
(5)  until count_i ≥ n − t end repeat;
```

a $\leq t$ crashes + participation required
$\rightsquigarrow$ eventually $count_i \geq n - t$ at every correct process $p_i$

# $k$-Set Agreement Algorithm: Termination (1/5)

```
(1)   PART.write(up);
(2)   repeat
(3)       part_i := PART.snapshot();
(4)       count_i := |{x such that part_i[x] = up}|;
(5)   until count_i ≥ n − t end repeat;
(6)   if count_i ≤ λ then
(7)       group_i := 2;
(8)   else
(9)       group_i := 1;
(10)  end if
```

a $\leq t$ crashes + participation required
$\rightsquigarrow$ eventually $count_i \geq n - t$ at every correct process $p_i$

b $\leq n - k$ processes with $count_i \leq n - k = \lambda$ when leaving loop (2)-(5)
$\rightsquigarrow \leq n - k$ processes in group 2

# $k$-Set Agreement Algorithm: Termination (1/5)

**thread** $T_1$ **is**

(12) **loop forever**
(13) **if** $DEC \neq \perp$ **then**
(14) **return**($DEC$);
(15) **end if**;
(16) **end loop**;

**thread** $T_2$ **is**

(17) **if** $group_i = 1 \vee m > 0$ **then**
(18) $MUTEX[group_i].acquire()$;
(19) **if** $DEC = \perp$ **then**
(20) $DEC := in_i$;
(21) **end if**
(22) $MUTEX[group_i].release()$;
(23) **return**($DEC$);
(24) **end if**;

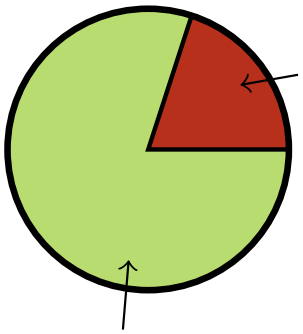
a $\leq t$ crashes + participation required
$\rightsquigarrow$ eventually $count_i \geq n - t$ at every correct process $p_i$

b $\leq n - k$ processes with $count_i \leq n - k = \lambda$ when leaving loop (2)-(5)
$\rightsquigarrow \leq n - k$ processes in group 2

c one process decides $\Rightarrow$ every correct process decides

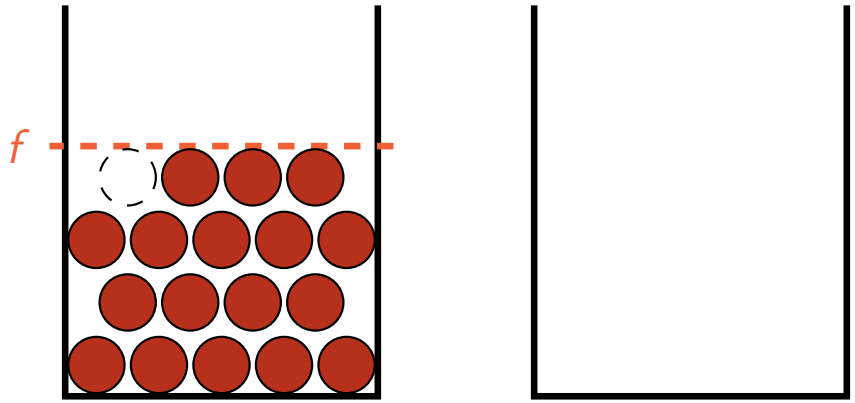
# $k$-Set Agreement Algorithm: Termination (2/5)

**d** If $m = 0$: $k = m + f = f$

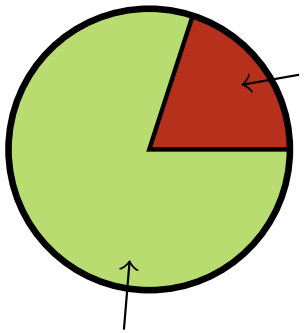


$f$-1 any-time crashes

$n - t$ correct processes



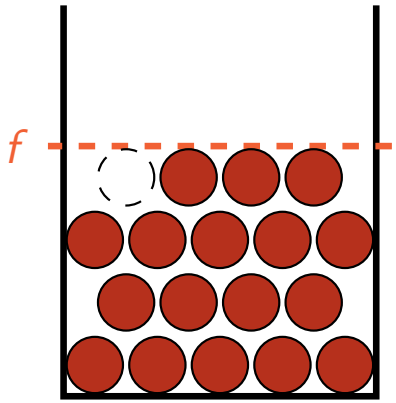
**d** If $m = 0$: $k = m + f = f$



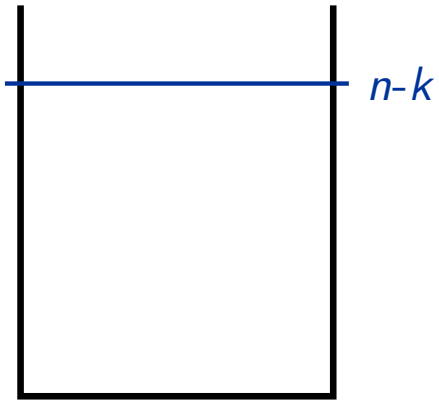
$f$-1 any-time crashes

$n - t$ correct processes

$n - t = n - (f - 1) = n - k + 1$

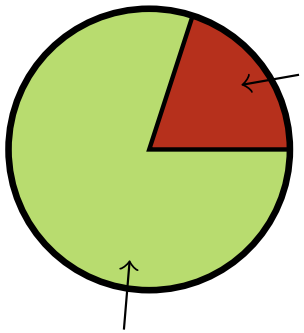



Group 1




Group 2

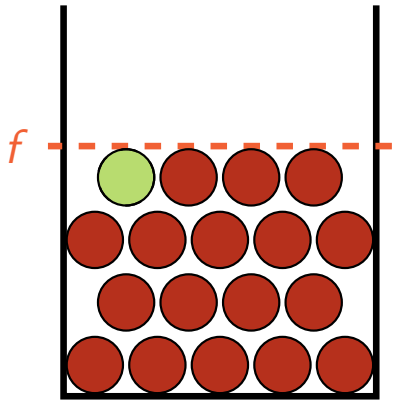
**d** If $m = 0$: $k = m + f = f$



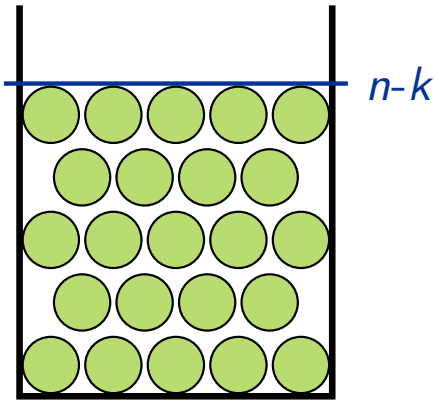
$f$-1 any-time crashes

$n - t$ correct processes

$n - t = n - (f - 1) = n - k + 1$

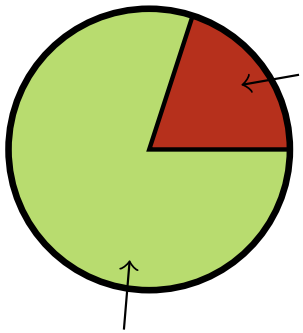



Group 1




Group 2

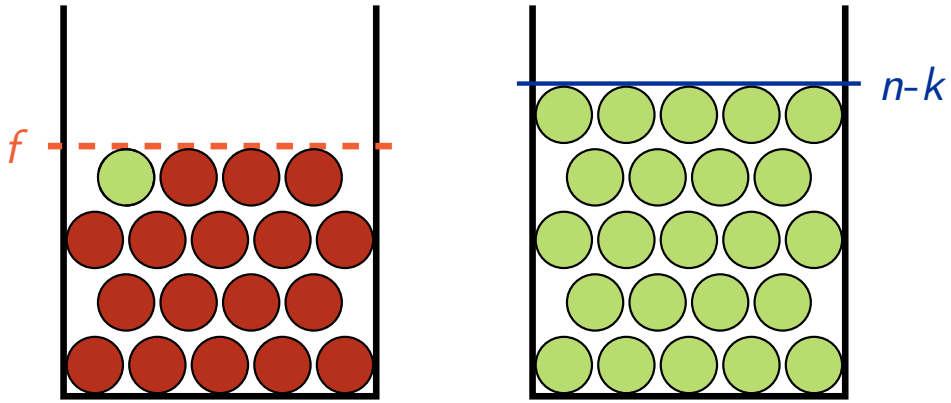
**d** If $m = 0$: $k = m + f = f$



$f$-1 any-time crashes

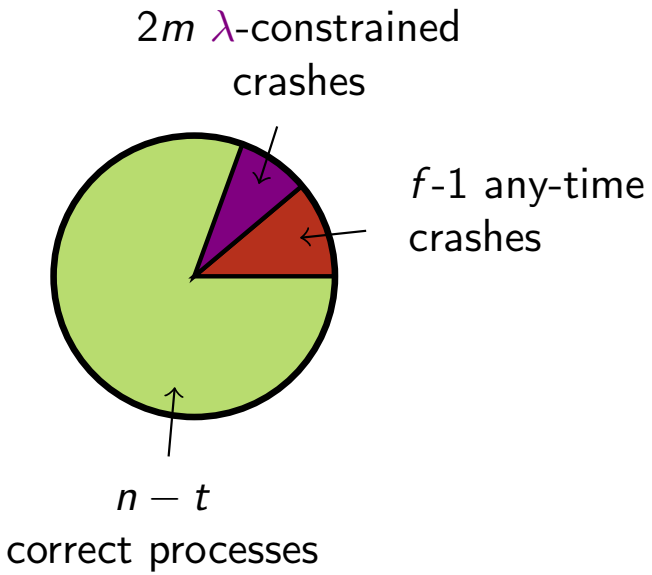
$n - t$ correct processes

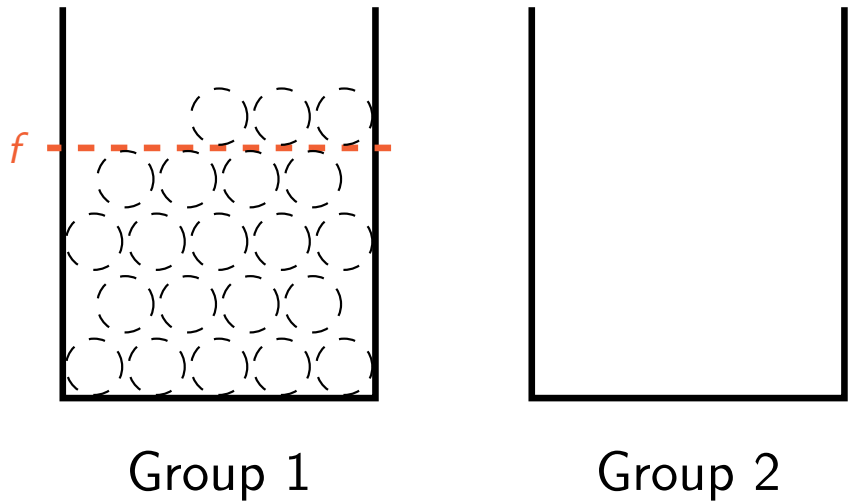
$n - t = n - (f - 1) = n - k + 1$




$\rightsquigarrow \geq 1$ correct process & $\leq f - 1$ (any-time) crashes in group 1 (Properties of DF $f$-mutex *MUTEX*[1]) $\Rightarrow$ at least one process decides

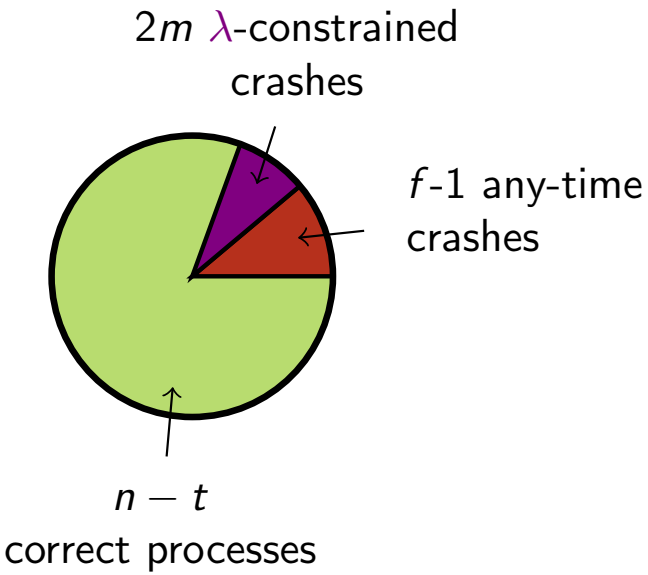
- **d** If $m > 0$:
  - $|\textbf{group 1}| \geq f$

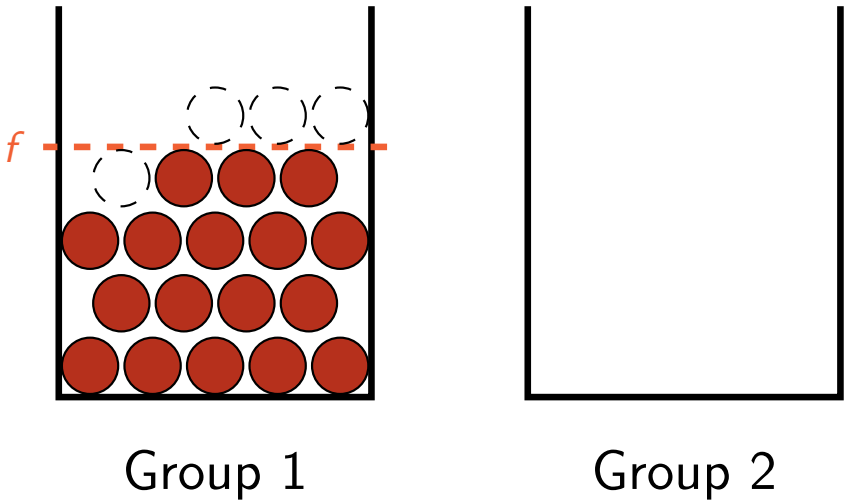


**d** If $m > 0$:

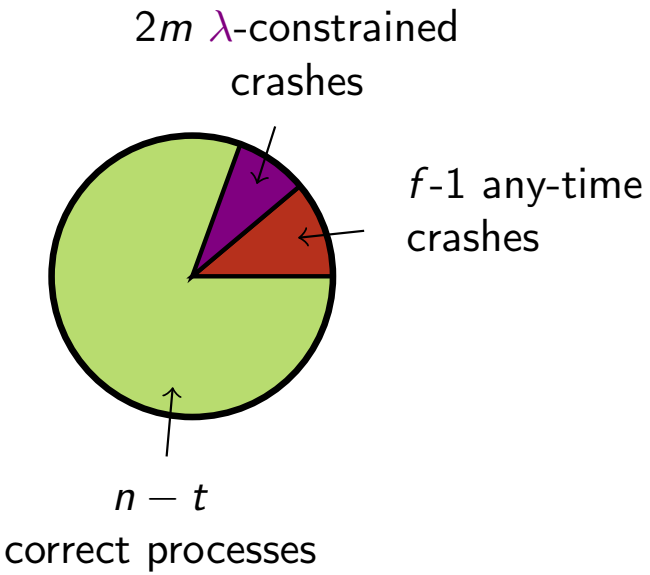
- $|\textbf{group 1}| \geq f$

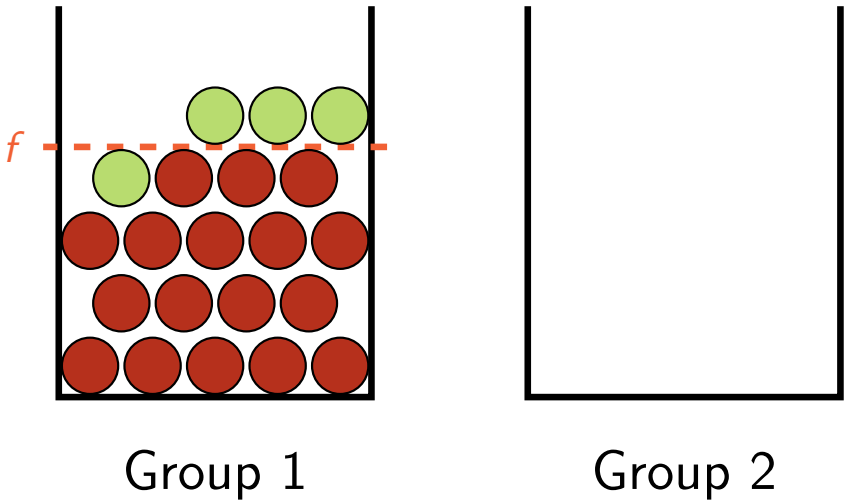


**d** If $m > 0$:

- $|\textbf{group 1}| \geq f$

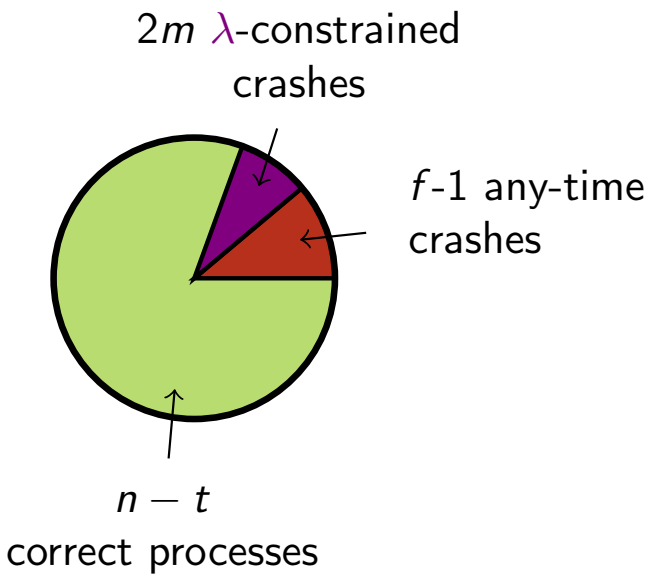


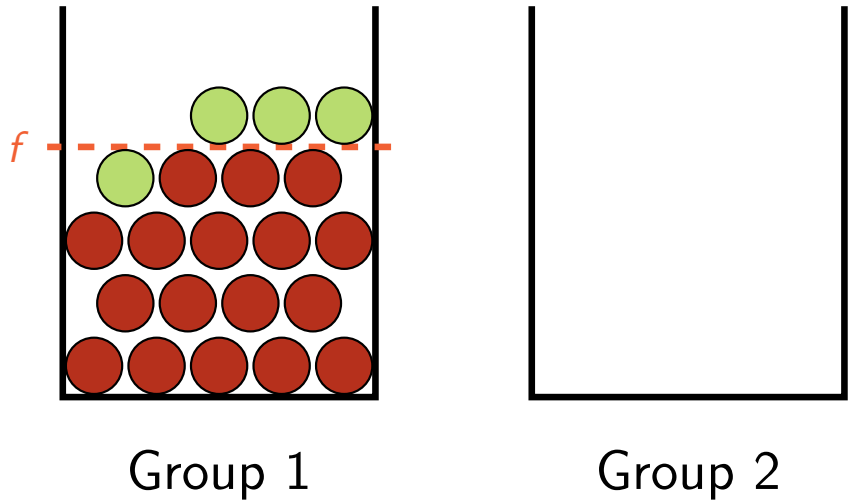



# $k$-Set Agreement Algorithm: Termination (3/5)

**d** If $m > 0$:

- $|\textbf{group 1}| \geq f$

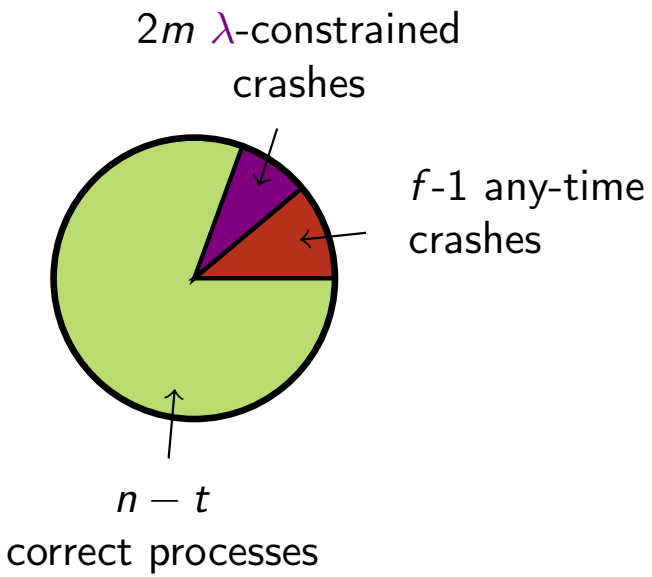


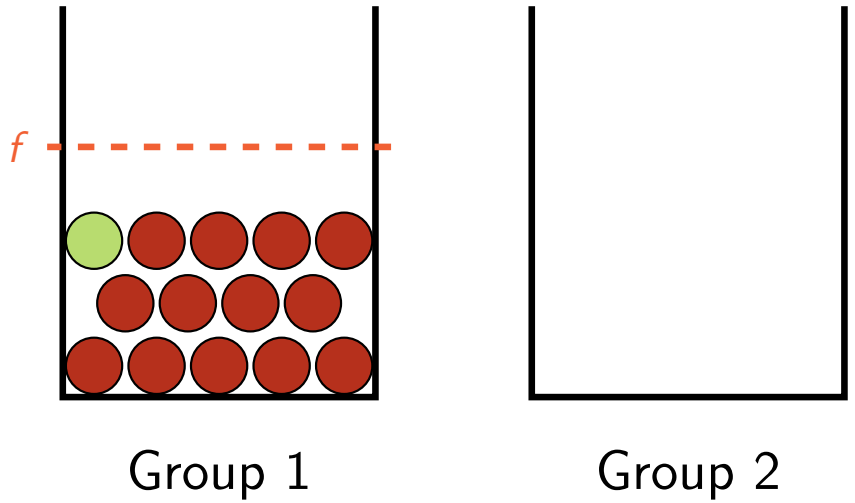




$\rightsquigarrow \geq 1$ correct process & $\leq f - 1$ (any-time) crashes in group 1
(Properties of DF $f$-mutex *MUTEX*[1]) $\Rightarrow$ at least one process decides

**d** If $m > 0$:

- $|\textbf{group 1}| < f$, **correct** $\in$ **group 1**

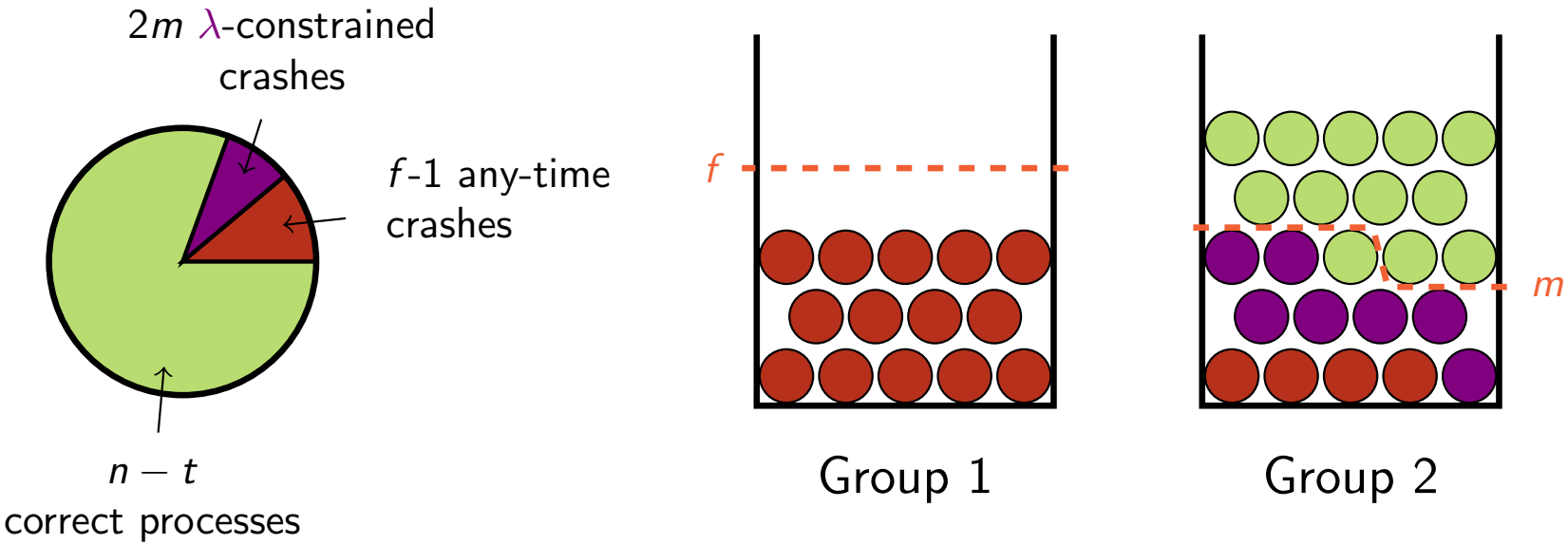



$\rightsquigarrow \geq 1$ correct process & $\leq f - 1$ (any-time) crashes in group 1
(Properties of DF $f$-mutex *MUTEX*[1]) $\Rightarrow$ at least one process decides

**d** If $m > 0$:

- $|\textbf{group 1}| < f$, **correct** $\notin$ **group 1**

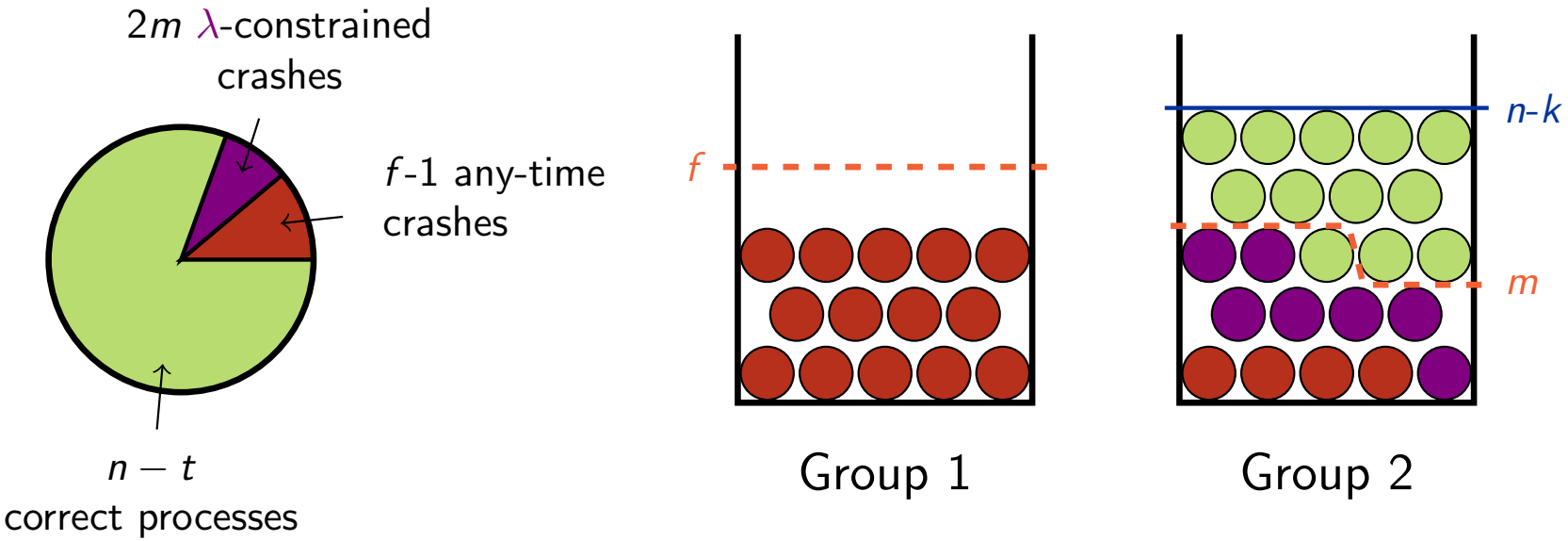



$$(n-k)-(n-t) = t-k = (2m+f-1)-(m+f) = m-1$$

$\rightsquigarrow \geq 1$ correct process & $\leq m-1$ crashes in group 2
(Properties of DF $m$-mutex *MUTEX*[2]) $\Rightarrow$ at least one process decides

**d** If $m > 0$:

- $|\textbf{group 1}| < f$, **correct** $\notin$ **group 1**




$$(n-k)-(n-t) = t-k = (2m+f-1)-(m+f) = m-1$$

$\rightsquigarrow \geq 1$ correct process & $\leq m-1$ crashes in group 2
(Properties of DF $m$-mutex $MUTEX[2]$) $\Rightarrow$ at least one process decides

# $k$-Set Agreement Algorithm: Properties

- $\lambda = n - k$
- $k \geq 2$
- $k = m + f,\ m \geq 0,\ f \geq 1$

| | |
|---|---|
| **total # of faults** | $t = 2m + f - 1 = k + m - 1$ |
| **$\lambda$-constrained crashes** | $2m$ |
| **any-time crashes** | $f - 1$ |

# $k$-Set Agreement Algorithm: Generalization

- $\lambda = n - \ell$
- $k \leq \ell \leq n$
- $k \geq 2$
- $k = m + f,\ m \geq 0,\ f \geq 1$

| | |
|---|---|
| **total # of faults** | $t = 2m + \ell - k + f - 1$ |
| **$\lambda$-constrained crashes** | $2m + \ell - k$ |
| **any-time crashes** | $f - 1$ |

# Conclusion

- Notion of **contention-related** crash failures
- Allows to circumvent impossibility results
- **Better understanding** of fault tolerance:
  In the $k$-set agreement algorithm, can trade **1 "strong"** any-time failure for **2 "weak"** $(n-k)$-constrained failures
- **Future work:**
  - Tight bounds?
  - General algorithm for $k$-set agreement, $\forall k \geq 1$.
  - What about crashes after the contention threshold $\lambda$?
  - What about other definitions of weak crash failures?

Thank you for your attention!

**Do you have any question?**

# Renaming

## Definition

- Initial name: $id_i$
- New name space: $\{1 \dots M\}$
- Operation *rename*($id_i$): return a new name
- Properties:
  - **Validity:** new name $\in \{1 \dots M\}$
  - **Agreement:** no 2 same new names
  - **Termination:** invokation of *rename*() by a correct process terminates

# Renaming Algorithm: Properties

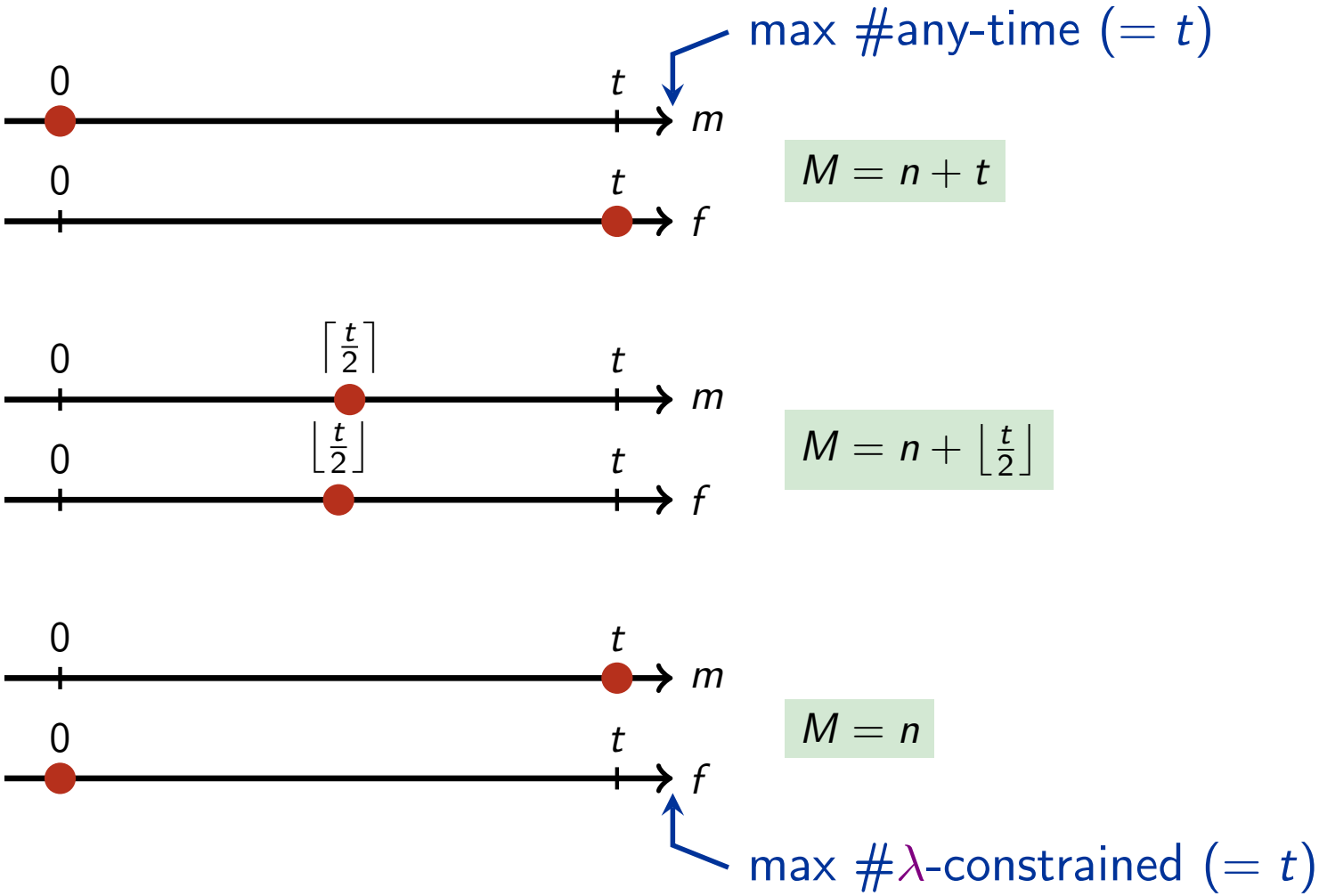
- $M = n + f$
- $\lambda = n - t - 1$
- $t = m + f$, $m \geq 0$, $f \geq 0$

| | |
|---|---|
| **total # of faults** | $t = m + f$ |
| **$\lambda$-constrained crashes** | $m$ |
| **any-time crashes** | $f$ |

[Herlihy, Shavit, 93]: Impossible with $f + 1$ any-time crash failures.

# Renaming Algorithm: Parameters

Parameters $f$ and $m$ allow the user to **tune** the proportion of each type of crash failures and the size of the new name space.



# Renaming Algorithm: Shared Registers

- $PART[1 \dots n]$: snapshot object, initially $[\texttt{down}, \dots, \texttt{down}]$

- $RENAMING_f$: $(n+f)$-renaming object that:
  - tolerates $\leq f$ any-time crash failures
  - does not require participation

  *e.g.* [Attiya, Welch, 04]

# Renaming Algorithm

**operation** *rename*($id_i$) **is**

```
(1)  PART.write(up);                                  % signal participation
(2)  repeat
(3)      part_i := PART.snapshot();                   % wait for n − t
(4)      count_i := |{x such that part_i[x] = up}|;   % participants
(5)  until count_i ≥ n − t end repeat;
```

# Renaming Algorithm

**operation** *rename*($id_i$) **is**

(1) $PART$.*write*(up); % *signal participation*
(2) **repeat**
(3) $part_i := PART$.*snapshot*(); % *wait for* $n - t$
(4) $count_i := |\{x \text{ such that } part_i[x] = \text{up}\}|$; % *participants*
(5) **until** $count_i \geq n - t$ **end repeat**;

(6) $newName_i := RENAMING_f$.*rename*($id_i$); % *get new name*
(7) **return**($newName_i$);

# Renaming Algorithm: Proof

```
(1)  PART.write(up);
(2)  repeat
(3)      part_i := PART.snapshot();
(4)      count_i := |{x such that part_i[x] = up}|;
(5)  until count_i ≥ n − t end repeat;
```

**a** $\leq t$ crashes + participation required
$\rightsquigarrow$ eventually $count_i \geq n - t$ at every correct process $p_i$

# Renaming Algorithm: Proof

(1) $PART.write(\mathtt{up})$;
(2) **repeat**
(3) $\quad part_i := PART.snapshot()$;
(4) $\quad count_i := |\{x \text{ such that } part_i[x] = \mathtt{up}\}|$;
(5) **until** $count_i \geq n - t$ **end repeat**;

a $\leq t$ crashes + participation required
$\rightsquigarrow$ eventually $count_i \geq n - t$ at every correct process $p_i$

b $n - t > \lambda \rightsquigarrow$ no $\lambda$-constrained crashes in $RENAMING_f$
$\rightsquigarrow \leq f$ crashes in $RENAMING_f$

# Renaming Algorithm: Proof

(1) $PART.write(\mathtt{up})$;
(2) **repeat**
(3) $\quad part_i := PART.snapshot()$;
(4) $\quad count_i := |\{x \text{ such that } part_i[x] = \mathtt{up}\}|$;
(5) **until** $count_i \geq n - t$ **end repeat**;

a $\leq t$ crashes + participation required
$\rightsquigarrow$ eventually $count_i \geq n - t$ at every correct process $p_i$

b $n - t > \lambda \rightsquigarrow$ no $\lambda$-constrained crashes in $RENAMING_f$
$\rightsquigarrow \leq f$ crashes in $RENAMING_f$

c participation not required for $RENAMING_f$ + properties of $RENAMING_f$
$\rightsquigarrow$ validity, agreement, & termination

# Generalization to One-Shot Concurrent Objects

Transform $OB$ = one-shot object tolerating $< X$ any-time crashes, participation not required

- $\lambda = n - t - 1$
- $t = m + f$, $m \geq 0$, $0 \leq f \leq X$

| total # of faults | $t = m + f$ |
|---|---|
| $\lambda$-constrained crashes | $m$ |
| any-time crashes | $f \leq X$ |

**operation** $op(in_i)$ **is**

(1) $PART.write(\mathtt{up})$;
(2) **repeat**
(3) $\quad part_i := PART.snapshot()$;
(4) $\quad count_i := |\{x$ such that $part_i[x] = \mathtt{up}\}|$;
(5) **until** $count_i \geq n - t$ **end repeat**;

(6) $res_i := OB.op(in_i)$;
(7) **return**$(res_i)$;