Distributed and Parallel Computing

Lecture 15

Alan P. Sexton

University of Birmingham

Spring 2018

Operations on a WFG

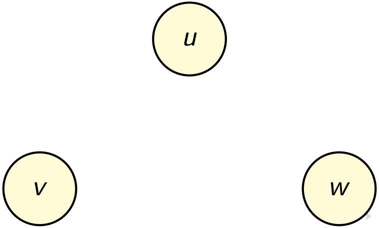
Conceptually, a WFG works as follows:

- There is one node v for each process v in the network
- A node v can be *active* or *blocked*
- An *active* node can make n-out-of-m requests of other nodes (and then becomes blocked) or grant requests to other nodes
- A *blocked* node can not make or grant requests but can become *active* if a sufficient number of its outstanding requests are granted
- When a *blocked* node with an outstanding n-out-of-m request has received n grants, it *purges* the remaining $m - n$ outstanding requests by informing the nodes involved that it no longer needs the resource requested

Mutually Exclusive Resources and the WFG

Mutually exclusive use of a resource requires a particular pattern on a WFG:

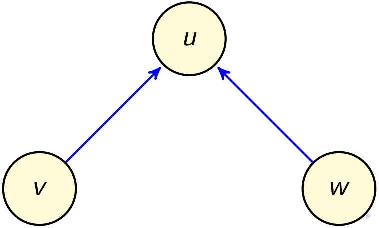
- u manages a mutually exclusive resource for v and w:



Mutually Exclusive Resources and the WFG

Mutually exclusive use of a resource requires a particular pattern on a WFG:

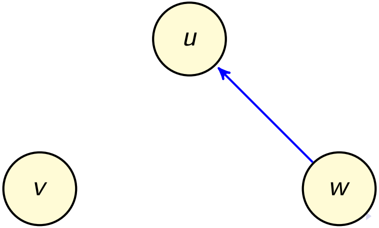
- u manages a mutually exclusive resource for v and w:
- v and w request the resource from u, so edges $v \to u$ and $w \to u$ are added to the WFG



Mutually Exclusive Resources and the WFG

Mutually exclusive use of a resource requires a particular pattern on a WFG:

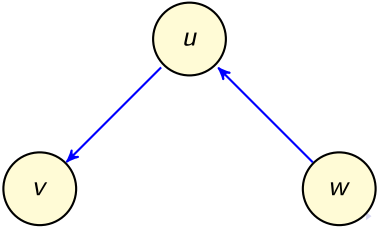
- u manages a mutually exclusive resource for v and w:
- v and w request the resource from u, so edges $v \rightarrow u$ and $w \rightarrow u$ are added to the WFG
- Let u grants v's request first: then the edge $v \rightarrow u$ is removed
  - Problem: v holds the resource but nothing stops u granting w's request immediately



Mutually Exclusive Resources and the WFG

Mutually exclusive use of a resource requires a particular pattern on a WFG:

- u manages a mutually exclusive resource for v and w:
- v and w request the resource from u, so edges $v \to u$ and $w \to u$ are added to the WFG
- Let u grants v's request first: then the edge $v \to u$ is removed
  - Problem: v holds the resource but nothing stops u granting w's request immediately
- Solution: granting v's request introduces a *new* dependency of u on v, which is modelled by adding an edge $u \to v$ for u to get back the resource from v



Representation of a Distributed WFG

We do not wish to centralise deadlock detection by getting the full global WFG onto a single node. Instead:

- Each node retains information about its local part of the WFG
- Distributed deadlock detection algorithm invoked by initiator:
  - Detects whether this node is deadlocked
  - Triggered after timeout when this node suspects it might be deadlocked

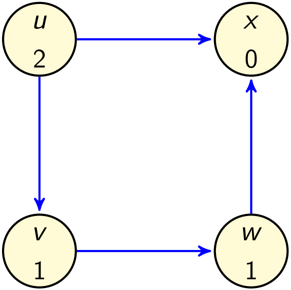
At each node u, have a number of variables:

- $\mathtt{OUT}_u$: The set of nodes u has sent a request to that are not yet granted or purged
- $\mathtt{IN}_u$: The set of nodes u has received a request from that are not yet granted or purged
- n_u: The number of grants that u currently needs to receive until it becomes unblocked. Note that $0 \leq n_u \leq |\mathtt{OUT}_u|$ and $n_u = 0 \Leftrightarrow \mathtt{OUT}_u = \{\}$

Detecting deadlock idea

Rather than searching for cycles or knots in the WFG (NP-hard), we simulate granting of grantable requests in the WFG until no more requests can be granted and see if the initiator node is unblocked

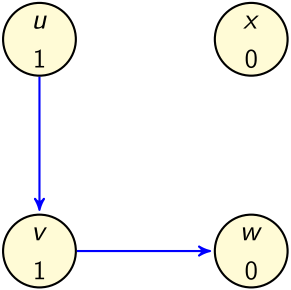
- Initiator is u



Detecting deadlock idea

Rather than searching for cycles or knots in the WFG (NP-hard), we simulate granting of grantable requests in the WFG until no more requests can be granted and see if the initiator node is unblocked

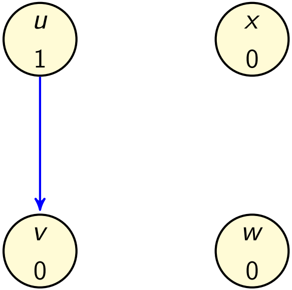
- Initiator is u
- x grants requests from u and w



Detecting deadlock idea

Rather than searching for cycles or knots in the WFG (NP-hard), we simulate granting of grantable requests in the WFG until no more requests can be granted and see if the initiator node is unblocked

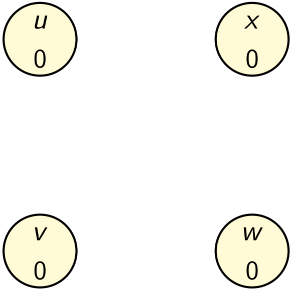
- Initiator is u
- x grants requests from u and w
- w grants request from v



Detecting deadlock idea

Rather than searching for cycles or knots in the WFG (NP-hard), we simulate granting of grantable requests in the WFG until no more requests can be granted and see if the initiator node is unblocked

- Initiator is u
- x grants requests from u and w
- w grants request from v
- v grants request from u



Bracha-Toueg Deadlock Detection Algorithm

Bracha-Toueg [1984] presented 3 variants of an algorithm for distributed deadlock detection:

1. On a network with instant messages where the base algorithm is static during deadlock detection
   - i.e. no requests, grants or purges occuring in parallel with deadlock detection
2. On a network with time delays in message delivery where the base algorithm is static
3. On a network with time delays in message delivery and the base algorithm is dynamic

Bracha-Toueg Deadlock Detection Algorithm

- Variation 1 requires that the $\mathtt{IN}_u$, $\mathtt{OUT}_u$ and n_u on each node u be pre-calculated from the local state and channel states of a globally consistent snapshot
- Variation 2 relaxes the need for the channel states to be used
- Variation 3 relaxes the need for a global snapshot to be pre-calculated
  - i.e. it integrates taking the snapshot with the deadlock detection

We will consider only the first variation, where we first apply a global snapshot algorithm which calculates $\mathtt{IN}_u$, $\mathtt{OUT}_u$ and n_u on each node u from the local and channel states.

Bracha-Toueg idea

Starting with the globally consistent $\mathtt{IN}_u$, $\mathtt{OUT}_u$ and n_u on each node u, execute 2 nested *Echo* algorithms to (virtually) construct a spanning tree of spanning trees.

- The first spanning tree is rooted at the initiator and traversed using Notify/Done messages
- The nested spanning trees are rooted at each active node, traversed using Grant/Ack messages and propagate all grants through the WFG

The Bracha-Toueg Algorithm

Initially: $\forall u, \texttt{notified}_u = \texttt{free}_u = \texttt{False}$. Initiator calls $\texttt{Notify}()$

$\texttt{Notify}_u()$:

- $\texttt{notified}_u \leftarrow \texttt{True}$
- for all $w \in \texttt{OUT}_u$, send $\langle\texttt{NOTIFY}\rangle$ to w
- if $n_u = 0$, then $\texttt{Grant}_u()$
- for all $w \in \texttt{OUT}_u$, await $\langle\texttt{DONE}\rangle$ from w

$\texttt{Grant}_u()$:

- $\texttt{free}_u \leftarrow \texttt{True}$
- for all $w \in \texttt{IN}_u$ send $\langle\texttt{GRANT}\rangle$ to w
- for all $w \in \texttt{IN}_u$ await $\langle\texttt{ACK}\rangle$ from w

On receive $\langle\texttt{NOTIFY}\rangle$:

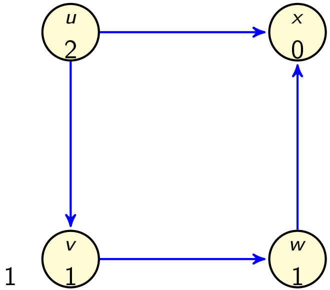
- If not $\texttt{notified}_u$, then $\texttt{Notify}_u()$
- u sends back $\langle\texttt{DONE}\rangle$

On receive $\langle\texttt{GRANT}\rangle$:

- If $n_u > 0$, then
  - $n_u \leftarrow n_u - 1$
  - if $n_u = 0$, then $\texttt{Grant}_u()$
- u sends back $\langle\texttt{ACK}\rangle$

Initiator u is not deadlocked if $\texttt{free}_u$ is $\texttt{True}$ at the end

A node awaiting $\langle\texttt{DONE}\rangle$ or $\langle\texttt{ACK}\rangle$ can process incoming $\langle\texttt{NOTIFY}\rangle$ and $\langle\texttt{GRANT}\rangle$ messages.



The Bracha-Toueg Algorithm

Initially: $\forall u, \texttt{notified}_u = \texttt{free}_u = \texttt{False}$. Initiator calls $\texttt{Notify}()$

$\texttt{Notify}_u()$:
- $\texttt{notified}_u \leftarrow \texttt{True}$
- for all $w \in \texttt{OUT}_u$, send $\langle\texttt{NOTIFY}\rangle$ to w
- if $n_u = 0$, then $\texttt{Grant}_u()$
- for all $w \in \texttt{OUT}_u$, await $\langle\texttt{DONE}\rangle$ from w

$\texttt{Grant}_u()$:
- $\texttt{free}_u \leftarrow \texttt{True}$
- for all $w \in \texttt{IN}_u$ send $\langle\texttt{GRANT}\rangle$ to w
- for all $w \in \texttt{IN}_u$ await $\langle\texttt{ACK}\rangle$ from w

On receive $\langle\texttt{NOTIFY}\rangle$:
- If not $\texttt{notified}_u$, then $\texttt{Notify}_u()$
- u sends back $\langle\texttt{DONE}\rangle$

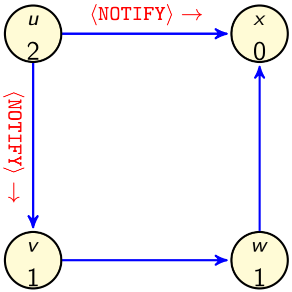
On receive $\langle\texttt{GRANT}\rangle$:
- If $n_u > 0$, then
  - $n_u \leftarrow n_u - 1$
  - if $n_u = 0$, then $\texttt{Grant}_u()$
- u sends back $\langle\texttt{ACK}\rangle$

Initiator u is not deadlocked if $\texttt{free}_u$ is $\texttt{True}$ at the end

A node awaiting $\langle\texttt{DONE}\rangle$ or $\langle\texttt{ACK}\rangle$ can process incoming $\langle\texttt{NOTIFY}\rangle$ and $\langle\texttt{GRANT}\rangle$ messages.

await $\langle\texttt{DONE}\rangle$ from v, x

2



The Bracha-Toueg Algorithm

Initially: $\forall u, \mathtt{notified}_u = \mathtt{free}_u = \mathtt{False}$. Initiator calls $\mathtt{Notify}()$

$\mathtt{Notify}_u()$:

$\mathtt{notified}_u \leftarrow \mathtt{True}$
for all $w \in \mathtt{OUT}_u$, send $\langle\mathtt{NOTIFY}\rangle$ to w
if $n_u = 0$, then $\mathtt{Grant}_u()$
for all $w \in \mathtt{OUT}_u$, await $\langle\mathtt{DONE}\rangle$ from w

$\mathtt{Grant}_u()$:

$\mathtt{free}_u \leftarrow \mathtt{True}$
for all $w \in \mathtt{IN}_u$ send $\langle\mathtt{GRANT}\rangle$ to w
for all $w \in \mathtt{IN}_u$ await $\langle\mathtt{ACK}\rangle$ from w

On receive $\langle\mathtt{NOTIFY}\rangle$:

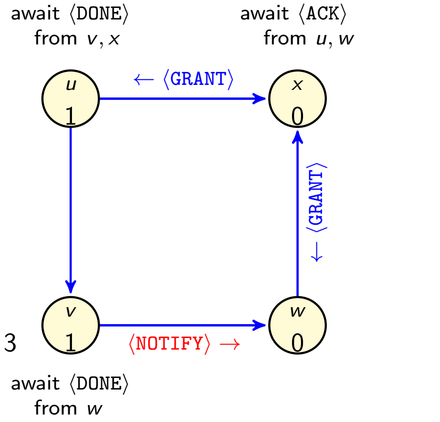
If not $\mathtt{notified}_u$, then $\mathtt{Notify}_u()$
u sends back $\langle\mathtt{DONE}\rangle$

On receive $\langle\mathtt{GRANT}\rangle$:

If $n_u > 0$, then
$n_u \leftarrow n_u - 1$
if $n_u = 0$, then $\mathtt{Grant}_u()$
u sends back $\langle\mathtt{ACK}\rangle$

Initiator u is not deadlocked if $\mathtt{free}_u$ is $\mathtt{True}$ at the end

A node awaiting $\langle\mathtt{DONE}\rangle$ or $\langle\mathtt{ACK}\rangle$ can process incoming $\langle\mathtt{NOTIFY}\rangle$ and $\langle\mathtt{GRANT}\rangle$ messages.



Initially: $\forall u, \texttt{notified}_u = \texttt{free}_u = \texttt{False}$. Initiator calls $\texttt{Notify}()$

$\texttt{Notify}_u()$:
- $\texttt{notified}_u \leftarrow \texttt{True}$
- for all $w \in \texttt{OUT}_u$, send $\langle\texttt{NOTIFY}\rangle$ to w
- if $n_u = 0$, then $\texttt{Grant}_u()$
- for all $w \in \texttt{OUT}_u$, await $\langle\texttt{DONE}\rangle$ from w

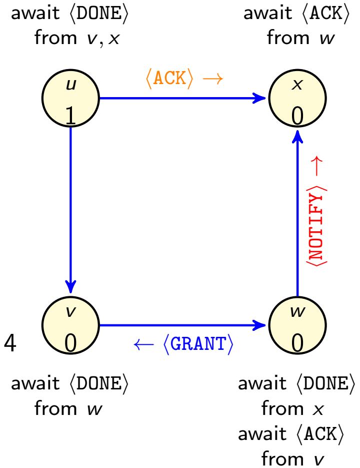
$\texttt{Grant}_u()$:
- $\texttt{free}_u \leftarrow \texttt{True}$
- for all $w \in \texttt{IN}_u$ send $\langle\texttt{GRANT}\rangle$ to w
- for all $w \in \texttt{IN}_u$ await $\langle\texttt{ACK}\rangle$ from w

On receive $\langle\texttt{NOTIFY}\rangle$:
- If not $\texttt{notified}_u$, then $\texttt{Notify}_u()$
- u sends back $\langle\texttt{DONE}\rangle$

On receive $\langle\texttt{GRANT}\rangle$:
- If $n_u > 0$, then
  - $n_u \leftarrow n_u - 1$
  - if $n_u = 0$, then $\texttt{Grant}_u()$
- u sends back $\langle\texttt{ACK}\rangle$

Initiator u is not deadlocked if $\texttt{free}_u$ is $\texttt{True}$ at the end

A node awaiting $\langle\texttt{DONE}\rangle$ or $\langle\texttt{ACK}\rangle$ can process incoming $\langle\texttt{NOTIFY}\rangle$ and $\langle\texttt{GRANT}\rangle$ messages.



The Bracha-Toueg Algorithm

Initially: $\forall u, \texttt{notified}_u = \texttt{free}_u = \texttt{False}$. Initiator calls `Notify()`

$\texttt{Notify}_u()$:
- $\texttt{notified}_u \leftarrow \texttt{True}$
- for all $w \in \texttt{OUT}_u$, send $\langle\texttt{NOTIFY}\rangle$ to w
- if $n_u = 0$, then $\texttt{Grant}_u()$
- for all $w \in \texttt{OUT}_u$, await $\langle\texttt{DONE}\rangle$ from w

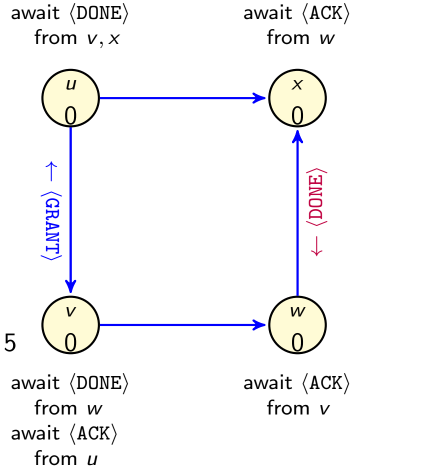
$\texttt{Grant}_u()$:
- $\texttt{free}_u \leftarrow \texttt{True}$
- for all $w \in \texttt{IN}_u$ send $\langle\texttt{GRANT}\rangle$ to w
- for all $w \in \texttt{IN}_u$ await $\langle\texttt{ACK}\rangle$ from w

On receive $\langle\texttt{NOTIFY}\rangle$:
- If not $\texttt{notified}_u$, then $\texttt{Notify}_u()$
- u sends back $\langle\texttt{DONE}\rangle$

On receive $\langle\texttt{GRANT}\rangle$:
- If $n_u > 0$, then
  - $n_u \leftarrow n_u - 1$
  - if $n_u = 0$, then $\texttt{Grant}_u()$
- u sends back $\langle\texttt{ACK}\rangle$

Initiator u is not deadlocked if $\texttt{free}_u$ is True at the end

A node awaiting $\langle\texttt{DONE}\rangle$ or $\langle\texttt{ACK}\rangle$ can process incoming $\langle\texttt{NOTIFY}\rangle$ and $\langle\texttt{GRANT}\rangle$ messages.



The Bracha-Toueg Algorithm

Initially: $\forall u, \texttt{notified}_u = \texttt{free}_u = \texttt{False}$. Initiator calls $\texttt{Notify}()$

$\texttt{Notify}_u()$:

- $\texttt{notified}_u \leftarrow \texttt{True}$
- for all $w \in \texttt{OUT}_u$, send $\langle\texttt{NOTIFY}\rangle$ to w
- if $n_u = 0$, then $\texttt{Grant}_u()$
- for all $w \in \texttt{OUT}_u$, await $\langle\texttt{DONE}\rangle$ from w

$\texttt{Grant}_u()$:

- $\texttt{free}_u \leftarrow \texttt{True}$
- for all $w \in \texttt{IN}_u$ send $\langle\texttt{GRANT}\rangle$ to w
- for all $w \in \texttt{IN}_u$ await $\langle\texttt{ACK}\rangle$ from w

On receive $\langle\texttt{NOTIFY}\rangle$:

- If not $\texttt{notified}_u$, then $\texttt{Notify}_u()$
- u sends back $\langle\texttt{DONE}\rangle$

On receive $\langle\texttt{GRANT}\rangle$:

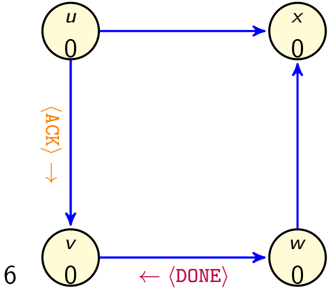
- If $n_u > 0$, then
  - $n_u \leftarrow n_u - 1$
  - if $n_u = 0$, then $\texttt{Grant}_u()$
- u sends back $\langle\texttt{ACK}\rangle$

Initiator u is not deadlocked if $\texttt{free}_u$ is $\texttt{True}$ at the end

A node awaiting $\langle\texttt{DONE}\rangle$ or $\langle\texttt{ACK}\rangle$ can process incoming $\langle\texttt{NOTIFY}\rangle$ and $\langle\texttt{GRANT}\rangle$ messages.

await $\langle\texttt{DONE}\rangle$ from v, x

await $\langle\texttt{ACK}\rangle$ from w



await $\langle\texttt{DONE}\rangle$ from w
await $\langle\texttt{ACK}\rangle$ from u

await $\langle\texttt{ACK}\rangle$ from v

The Bracha-Toueg Algorithm

Initially: $\forall u, \texttt{notified}_u = \texttt{free}_u = \texttt{False}$. Initiator calls $\texttt{Notify}()$

$\texttt{Notify}_u()$:

- $\texttt{notified}_u \leftarrow \texttt{True}$
- for all $w \in \texttt{OUT}_u$, send $\langle\texttt{NOTIFY}\rangle$ to w
- if $n_u = 0$, then $\texttt{Grant}_u()$
- for all $w \in \texttt{OUT}_u$, await $\langle\texttt{DONE}\rangle$ from w

$\texttt{Grant}_u()$:

- $\texttt{free}_u \leftarrow \texttt{True}$
- for all $w \in \texttt{IN}_u$ send $\langle\texttt{GRANT}\rangle$ to w
- for all $w \in \texttt{IN}_u$ await $\langle\texttt{ACK}\rangle$ from w

On receive $\langle\texttt{NOTIFY}\rangle$:

- If not $\texttt{notified}_u$, then $\texttt{Notify}_u()$
- u sends back $\langle\texttt{DONE}\rangle$

On receive $\langle\texttt{GRANT}\rangle$:

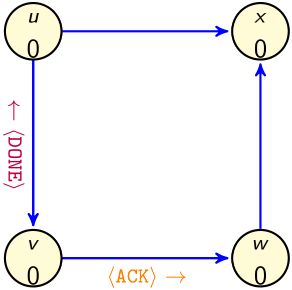
- If $n_u > 0$, then
  - $n_u \leftarrow n_u - 1$
  - if $n_u = 0$, then $\texttt{Grant}_u()$
- u sends back $\langle\texttt{ACK}\rangle$

Initiator u is not deadlocked if $\texttt{free}_u$ is $\texttt{True}$ at the end

A node awaiting $\langle\texttt{DONE}\rangle$ or $\langle\texttt{ACK}\rangle$ can process incoming $\langle\texttt{NOTIFY}\rangle$ and $\langle\texttt{GRANT}\rangle$ messages.

await $\langle\texttt{DONE}\rangle$ from x	await $\langle\texttt{ACK}\rangle$ from w

7



The Bracha-Toueg Algorithm

Initially: $\forall u, \texttt{notified}_u = \texttt{free}_u = \texttt{False}$. Initiator calls $\texttt{Notify}()$

$\texttt{Notify}_u()$:

- $\texttt{notified}_u \leftarrow \texttt{True}$
- for all $w \in \texttt{OUT}_u$, send $\langle\texttt{NOTIFY}\rangle$ to w
- if $n_u = 0$, then $\texttt{Grant}_u()$
- for all $w \in \texttt{OUT}_u$, await $\langle\texttt{DONE}\rangle$ from w

$\texttt{Grant}_u()$:

- $\texttt{free}_u \leftarrow \texttt{True}$
- for all $w \in \texttt{IN}_u$ send $\langle\texttt{GRANT}\rangle$ to w
- for all $w \in \texttt{IN}_u$ await $\langle\texttt{ACK}\rangle$ from w

On receive $\langle\texttt{NOTIFY}\rangle$:

- If not $\texttt{notified}_u$, then $\texttt{Notify}_u()$
- u sends back $\langle\texttt{DONE}\rangle$

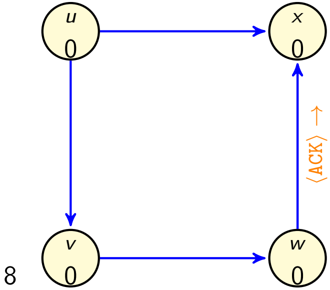
On receive $\langle\texttt{GRANT}\rangle$:

- If $n_u > 0$, then
  - $n_u \leftarrow n_u - 1$
  - if $n_u = 0$, then $\texttt{Grant}_u()$
- u sends back $\langle\texttt{ACK}\rangle$

Initiator u is not deadlocked if $\texttt{free}_u$ is $\texttt{True}$ at the end

A node awaiting $\langle\texttt{DONE}\rangle$ or $\langle\texttt{ACK}\rangle$ can process incoming $\langle\texttt{NOTIFY}\rangle$ and $\langle\texttt{GRANT}\rangle$ messages.

await $\langle\texttt{DONE}\rangle$ from x



The Bracha-Toueg Algorithm

Initially: $\forall u, \texttt{notified}_u = \texttt{free}_u = \texttt{False}$. Initiator calls $\texttt{Notify}()$

$\texttt{Notify}_u()$:
- $\texttt{notified}_u \leftarrow \texttt{True}$
- for all $w \in \texttt{OUT}_u$, send $\langle\texttt{NOTIFY}\rangle$ to w
- if $n_u = 0$, then $\texttt{Grant}_u()$
- for all $w \in \texttt{OUT}_u$, await $\langle\texttt{DONE}\rangle$ from w

$\texttt{Grant}_u()$:
- $\texttt{free}_u \leftarrow \texttt{True}$
- for all $w \in \texttt{IN}_u$ send $\langle\texttt{GRANT}\rangle$ to w
- for all $w \in \texttt{IN}_u$ await $\langle\texttt{ACK}\rangle$ from w

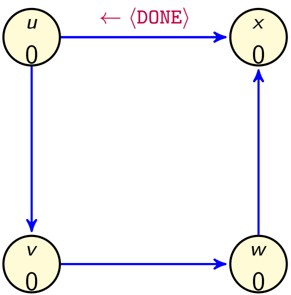
On receive $\langle\texttt{NOTIFY}\rangle$:
- If not $\texttt{notified}_u$, then $\texttt{Notify}_u()$
- u sends back $\langle\texttt{DONE}\rangle$

On receive $\langle\texttt{GRANT}\rangle$:
- If $n_u > 0$, then
  - $n_u \leftarrow n_u - 1$
  - if $n_u = 0$, then $\texttt{Grant}_u()$
- u sends back $\langle\texttt{ACK}\rangle$

Initiator u is not deadlocked if $\texttt{free}_u$ is $\texttt{True}$ at the end

A node awaiting $\langle\texttt{DONE}\rangle$ or $\langle\texttt{ACK}\rangle$ can process incoming $\langle\texttt{NOTIFY}\rangle$ and $\langle\texttt{GRANT}\rangle$ messages.

9



The Bracha-Toueg Algorithm

Initially: $\forall u, \texttt{notified}_u = \texttt{free}_u = \texttt{False}$. Initiator calls $\texttt{Notify}()$

$\texttt{Notify}_u()$:

$\texttt{notified}_u \leftarrow \texttt{True}$
for all $w \in \texttt{OUT}_u$, send $\langle\texttt{NOTIFY}\rangle$ to w
if $n_u = 0$, then $\texttt{Grant}_u()$
for all $w \in \texttt{OUT}_u$, await $\langle\texttt{DONE}\rangle$ from w

$\texttt{Grant}_u()$:

$\texttt{free}_u \leftarrow \texttt{True}$
for all $w \in \texttt{IN}_u$ send $\langle\texttt{GRANT}\rangle$ to w
for all $w \in \texttt{IN}_u$ await $\langle\texttt{ACK}\rangle$ from w

On receive $\langle\texttt{NOTIFY}\rangle$:

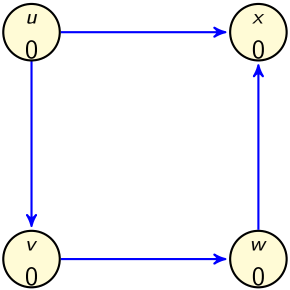
If not $\texttt{notified}_u$, then $\texttt{Notify}_u()$
u sends back $\langle\texttt{DONE}\rangle$

On receive $\langle\texttt{GRANT}\rangle$:

If $n_u > 0$, then
$n_u \leftarrow n_u - 1$
if $n_u = 0$, then $\texttt{Grant}_u()$
u sends back $\langle\texttt{ACK}\rangle$

Initiator u is not deadlocked if $\texttt{free}_u$ is $\texttt{True}$ at the end

A node awaiting $\langle\texttt{DONE}\rangle$ or $\langle\texttt{ACK}\rangle$ can process incoming $\langle\texttt{NOTIFY}\rangle$ and $\langle\texttt{GRANT}\rangle$ messages.



What about the Mutual Exclusion Pattern?

Bracha-Toueg doesn't guarantee that a deadlock won't occur in the future: only whether the initiator is deadlocked or not.

- The global snapshot will capture the state either before the resource is handed off to the first requestor, or after.
- The WFG will be different in the two cases
- Bracha-Toueg will give the correct answer for the particular case of the WFG that appears in the global state, even if the next operation would necessarily put the node into deadlock