Advanced Distributed Algorithms and Data Structures

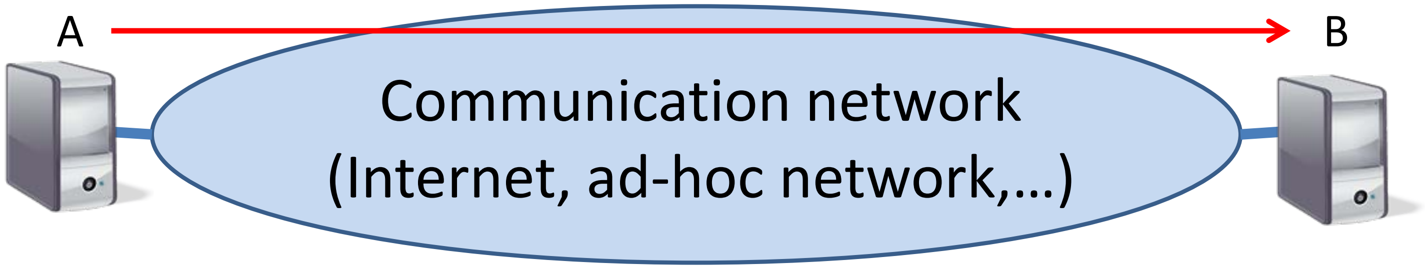
Chapter 4: Link Primitives

Christian Scheideler
Institut für Informatik
Universität Paderborn

Overview

- Model and basic primitives
- Universality
- Safe primitives

Model and Basic Primitives



A knows (IP address, MAC address,… of) resp. has access autorization for B : network can send message from A to B

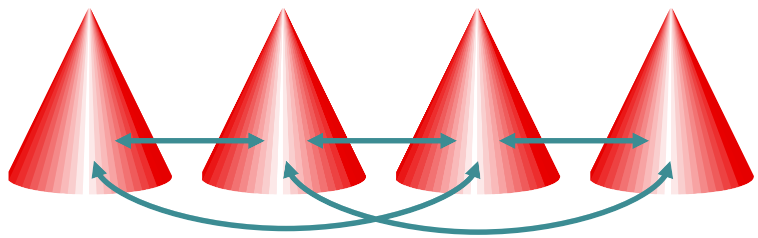
High-level view:

A knows B $\Rightarrow$ overlay edge (A,B) from A to B (A $\longrightarrow$ B)

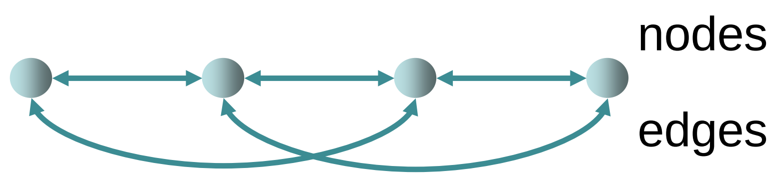
Set of all overlay edges forms directed graph known as overlay network.

Model and Basic Primitives

- Overlay network established by processes:



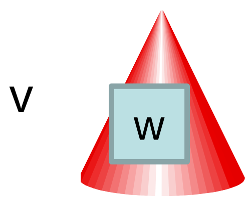
- Graph representation:



- Edge $A \rightarrow B$ means: A knows / has access to B

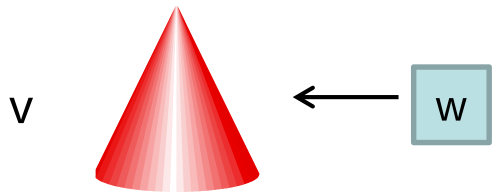
Model and Basic Primitives

- Edge set E_L: set of pairs (v,w) where v knows w (explicit edges).



:

- Edge set E_M: set of pairs (v,w) with a message in transit to v containing a reference to w (implicit edges).

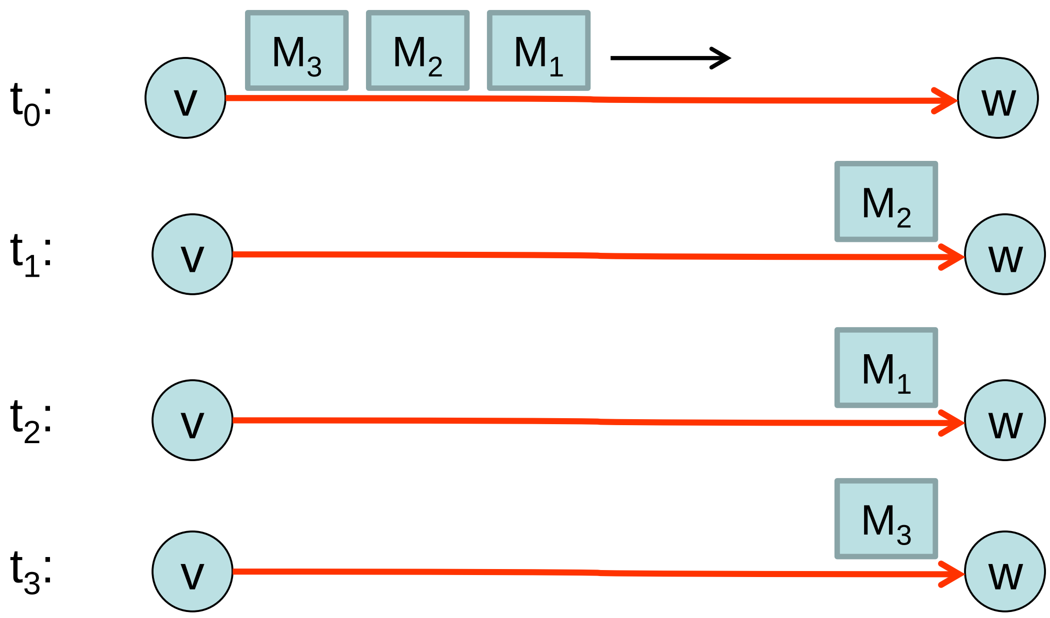


:

- Graph $G=(V,E_L \cup E_M)$: graph of all explicit and implicit edges.

Model and Basic Primitives

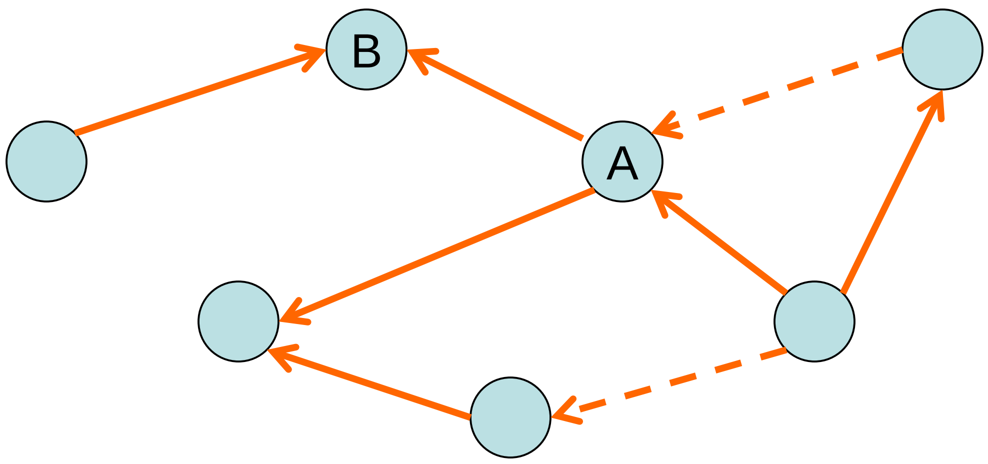
Asynchronous message passing



- all messages are eventually delivered
- but no FIFO delivery guaranteed

Model and Basic Primitives

Fundamental goal: topology of process graph (i.e., G) is kept weakly connected at any time

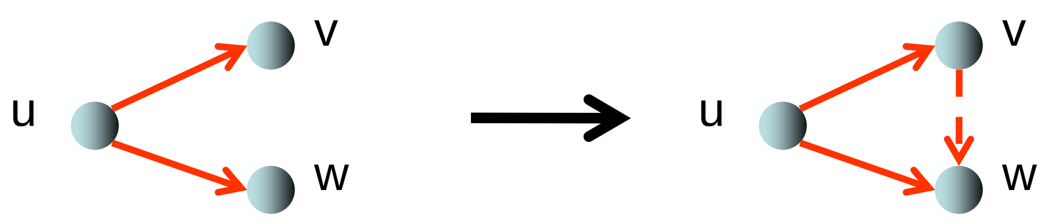


Fundamental rule: never just „throw away“ a reference!

Model and Basic Primitives

Admissible rules for weak connectivity:

- Introduction:



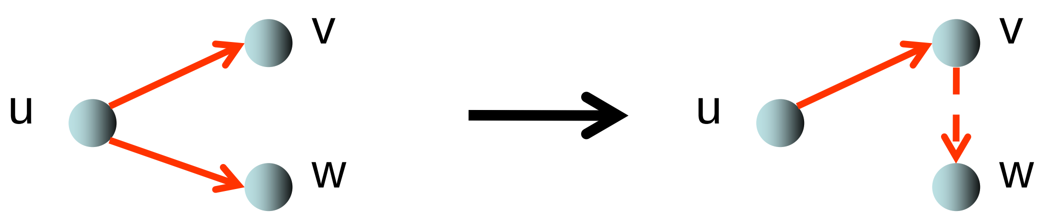
u introduces w to v by sending a message to v containing a reference to w

- special case: u introduces itself to v

Model and Basic Primitives

Admissible rules for weak connectivity:

- Delegation:



u delegates its reference of w to v (i.e., afterwards it does not store a reference of w any more)

- Fusion:

Model and Basic Primitives

Admissible rules for weak connectivity:

- Reversal:

u sends a reference of itself to v and deletes v´s reference

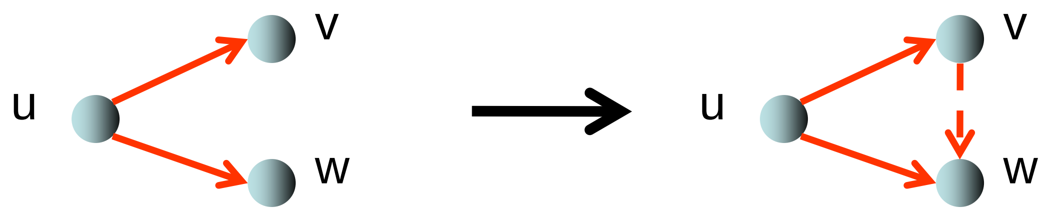
Remarks:

- Advantage: rules can be executed in a local, wait-free manner in arbitrary asynchronous environments
- Introduction, delegation and fusion preserve strong connectivity

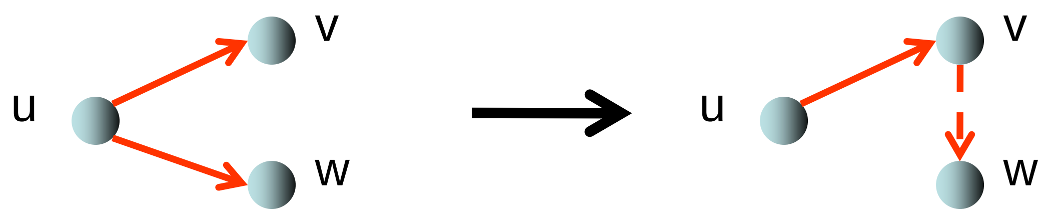
Universality

Theorem 4.1: The 3 primitives below are weakly universal, i.e., they can be used to transform any weakly connected graph G=(V,E) into any strongly connected graph G´=(V,E´).

introduction:



delegation:



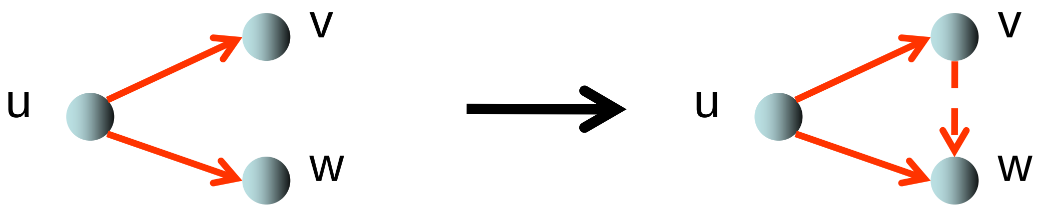
fusion:

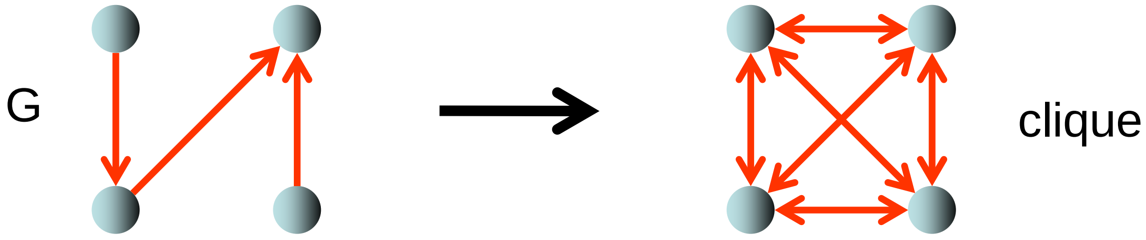
Universality

Proof: consists of two parts

1. Using the introduction rule, one can get from any weakly connected graph $G=(V,E)$ to the clique.

introduction:

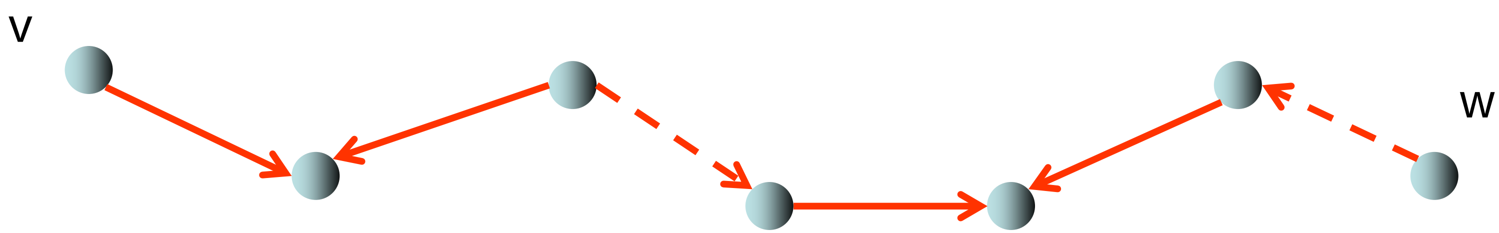




Universality

How does that work?

Consider any two nodes v and w. Since G is weakly connected, there is a path from v to w.



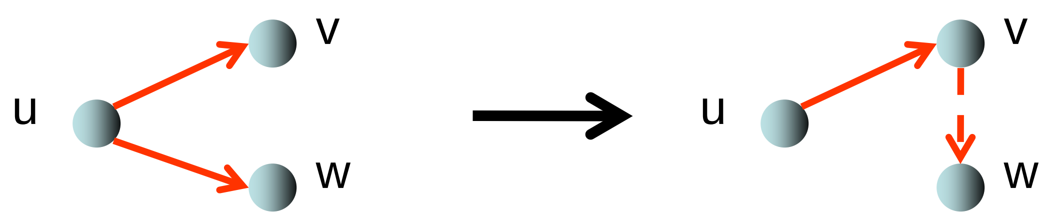
Exercise: If in each round every node introduces all of its neighbors and itself to all of its neighbors, then just O(log n) rounds are needed till the clique is reached.

Universality

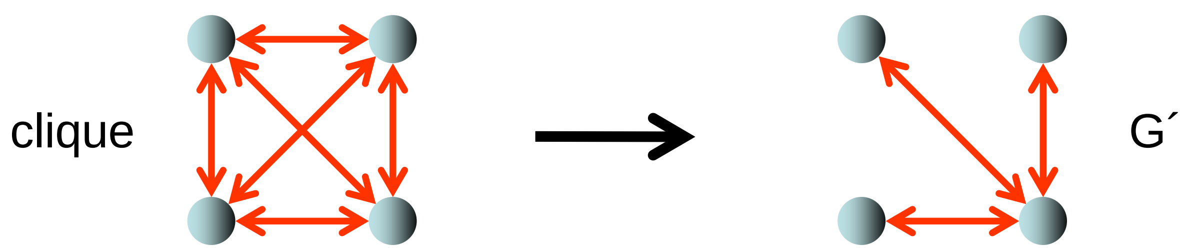
Proof:

2. Using the delegation and fusion primitives, one can get from the clique to $G´=(V,E´)$.

delegation:



fusion:

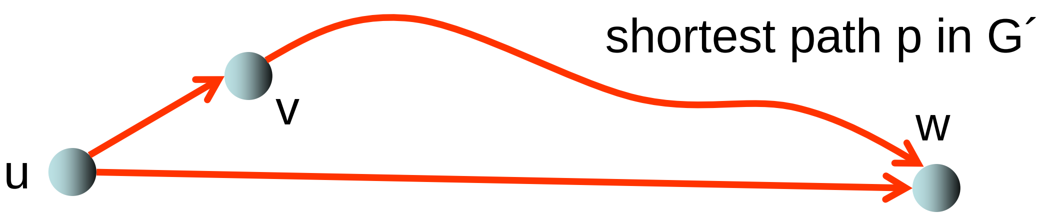


Universality

Proof: (details)

2. Suppose that G=(V,E) is a clique. Then G can be transformed into G´=(V,E´) in the following way without ever dropping edges of G´.

- Let (u,w) be an arbitrary edge that needs to be removed because it is not in E´. Since G´=(V,E´) is strongly connected, there is a directed path from u to w in G´. Let p be a shortest such path and let v be the next node along this path.

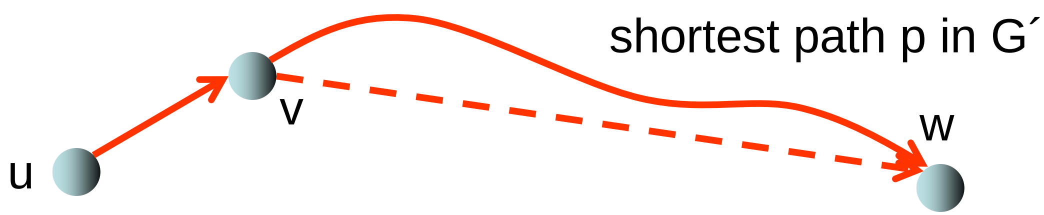


Universality

Proof: (details)

2. Suppose that G=(V,E) is a clique. Then G can be transformed into G´=(V,E´) in the following way without ever dropping edges of G´.

- Let (u,w) be an arbitrary edge that needs to be removed because it is not in E´. Since G´=(V,E´) is strongly connected, there is a directed path from u to w in G´. Let p be a shortest such path and let v be the next node along this path.
- Then node u delegates (u,w) to v, i.e., (u,w) is transformed into (v,w).

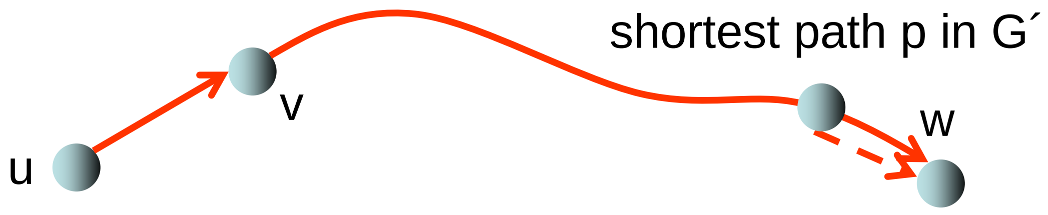


Universality

Proof: (details)

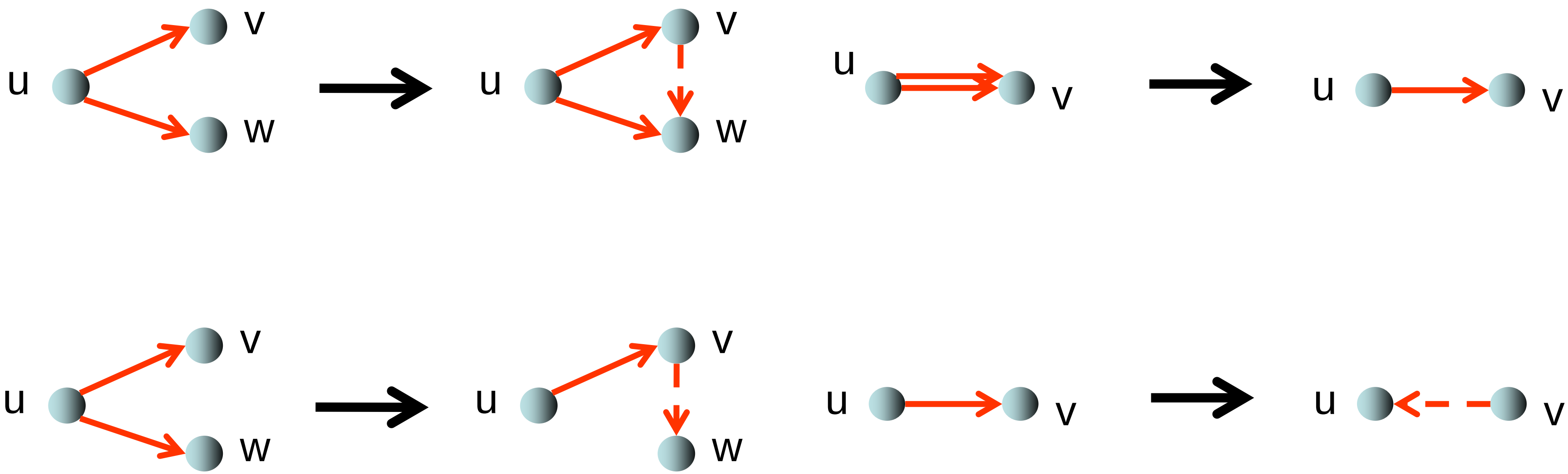
2. Suppose that $G=(V,E)$ is a clique. Then G can be transformed into $G'=(V,E')$ in the following way without ever dropping edges of G'.

- Let (u,w) be an arbitrary edge that needs to be removed because it is not in E'. Since $G'=(V,E')$ is strongly connected, there is a directed path from u to w in G'. Let p be a shortest such path and let v be the next node along this path.
- Then node u delegates (u,w) to v, i.e., (u,w) is transformed into (v,w).
- After at most $n-2$ further delegations along p, the edge can be fused with an edge in G'. Doing that for all $(u,w)\notin E'$, we get G'.



Universality

Theorem 4.2: The 4 rules below are universal in a sense that one can get from any weakly connected graph G=(V,E) to any other weakly connected graph G´=(V,E´).

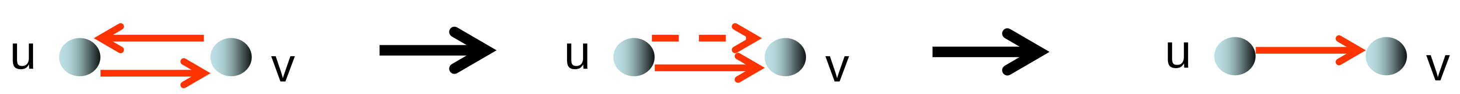


Universality

Theorem 4.2: The 4 rules below are universal in a sense that one can get from any weakly connected graph $G=(V,E)$ to any other weakly connected graph $G'=(V,E')$.

Proof:

- Let $G''=(V,E'')$ be the bidirected version of G', i.e., for all $(u,v)\in E'$, $(u,v)\in E''$ and $(v,u)\in E''$.
- Certainly, G'' is strongly connected. (Why?)
- Theorem 4.1: we can get from G to G''.
- From G'' to G': use reversal and fusion rule to remove wrong directions:



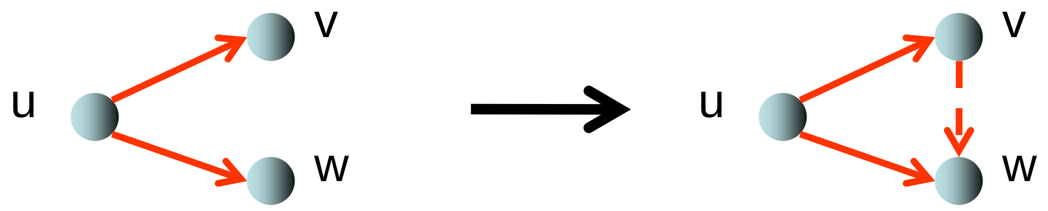
Universality

Remark:

- Each of the four rules is necessary for universality.
  - Introduction: only one that generates new edge
  - Fusion: only one that removes edge
  - Delegation: only one that moves edge away
  - Reversal: only one that makes nodes unreachable
- Theorems 4.1 and 4.2 only show that in principle it is possible to get from any weakly connected graph to any other weakly resp. strongly connected graph.
- Later, we will see distributed algorithms for that.

Safe Primitives

Recall the definition of the introduction rule:



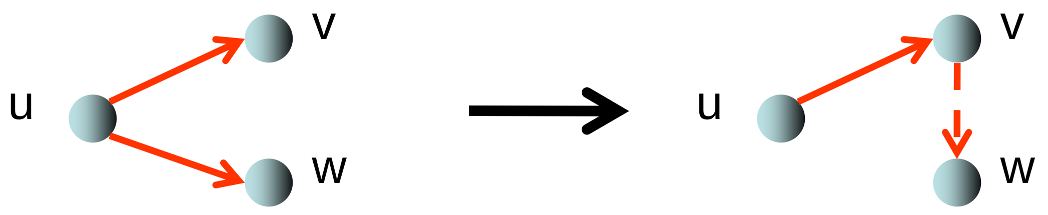
u introduces w to v by sending a message to v containing a reference to w

This violates w´s right to decide who shall connect to it.

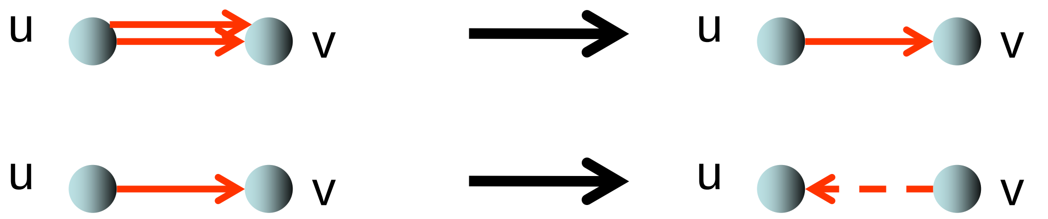
(But self-introduction is fine.)

Safe Primitives

Same problem with delegation:

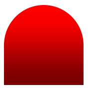


But fusion and reversal are fine:

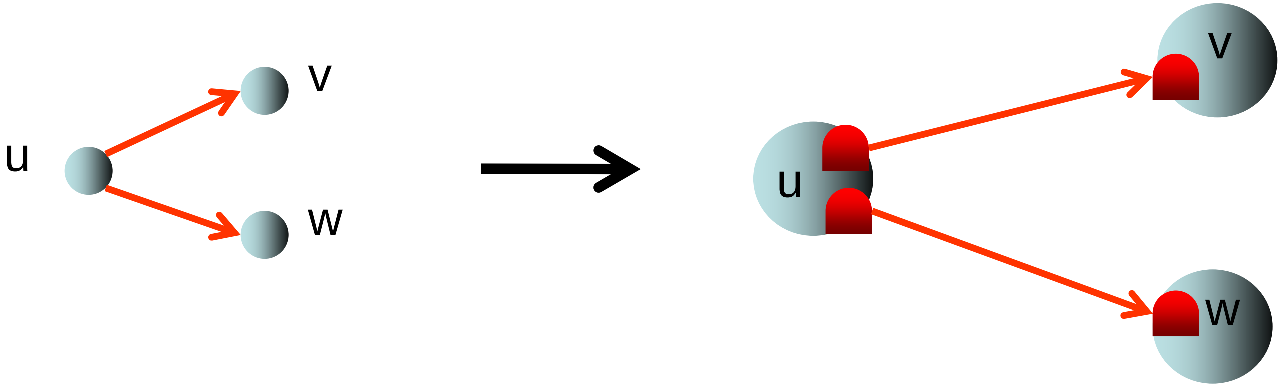


Safe Primitives

How to obtain safe forms of introduction and delegation?

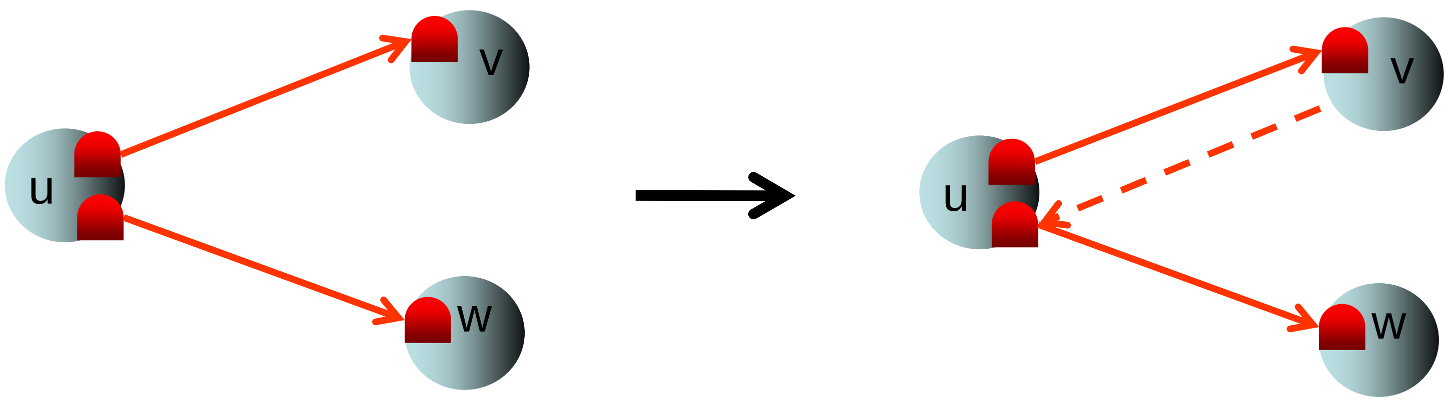
→Use the concept of relays ()

Extension of picture with relays:



Safe Primitives

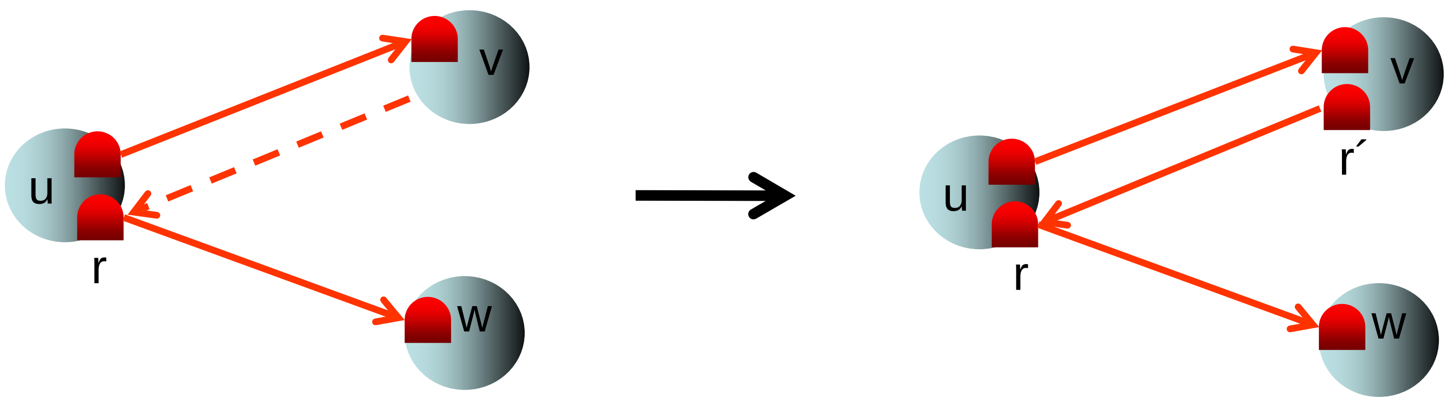
Safe introduction:



Instead of introducing w to v, u can only introduce its relay to w to v.

Safe Primitives

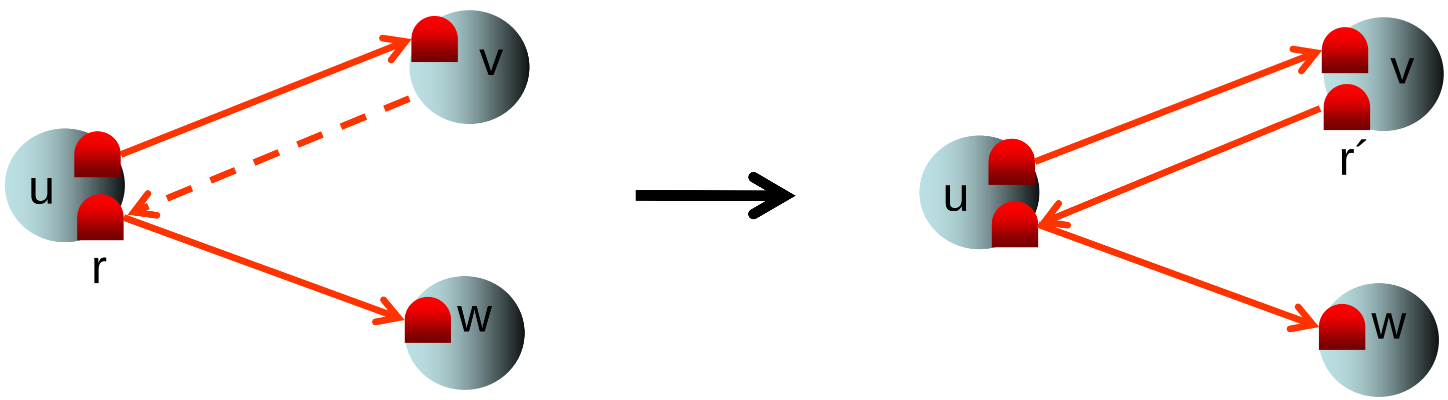
Safe introduction:



Once the reference of relay r to w is received by v, it is tied to a new relay r´ at v pointing to r.

Safe Primitives

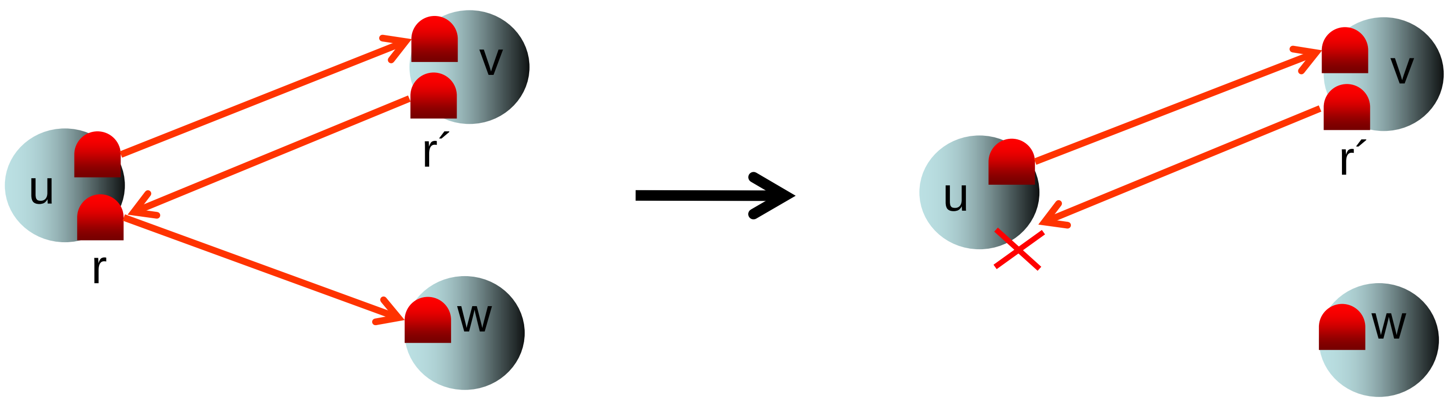
Safe introduction:



No access rights violated: u could have just forwarded anything from v to w by itself.

Safe Primitives

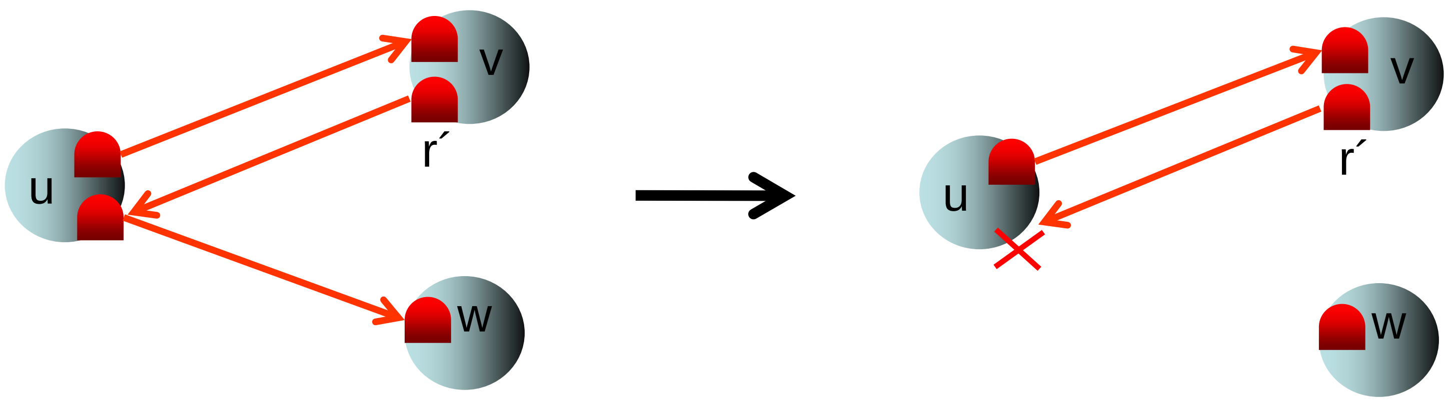
Safe introduction:



Most importantly, if u kills its relay to w, also v´s connection to w is gone.

Safe Primitives

Safe introduction:



→ Principle of least exposure: when killing all relays with incoming links, no request can reach a node any more

Safe Primitives

Possible outcome of safe introductions:

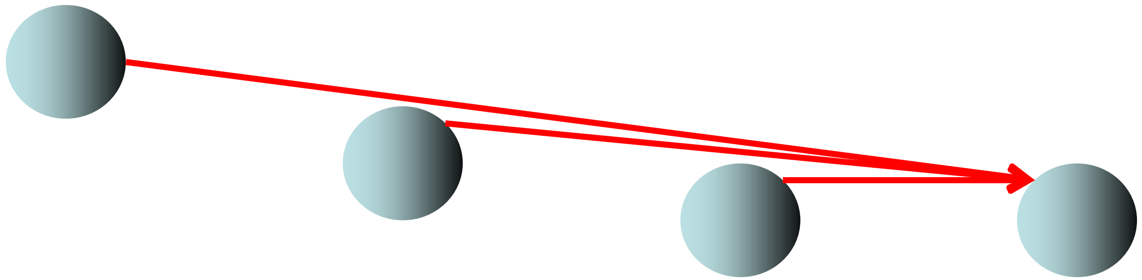
Remarks:

- Whenever a reference of some relay r is received, the Trusted Communication Layer (TCL) automatically creates a local relay r´ pointing to r and forwards r´ to the corresponding process instead. Thus, processes only have references to local relays.
- Any relay that is created by a process (not the TCL) is a sink relay (see v), i.e., all messages sent to it will be forwarded to the process owning it.
- Any one of the processes between u and v can send a message to v, but only v will ever see them since the TCLs of the other processes directly forward the messages (i.e., the TCL acts like a bridge).

Safe Primitives

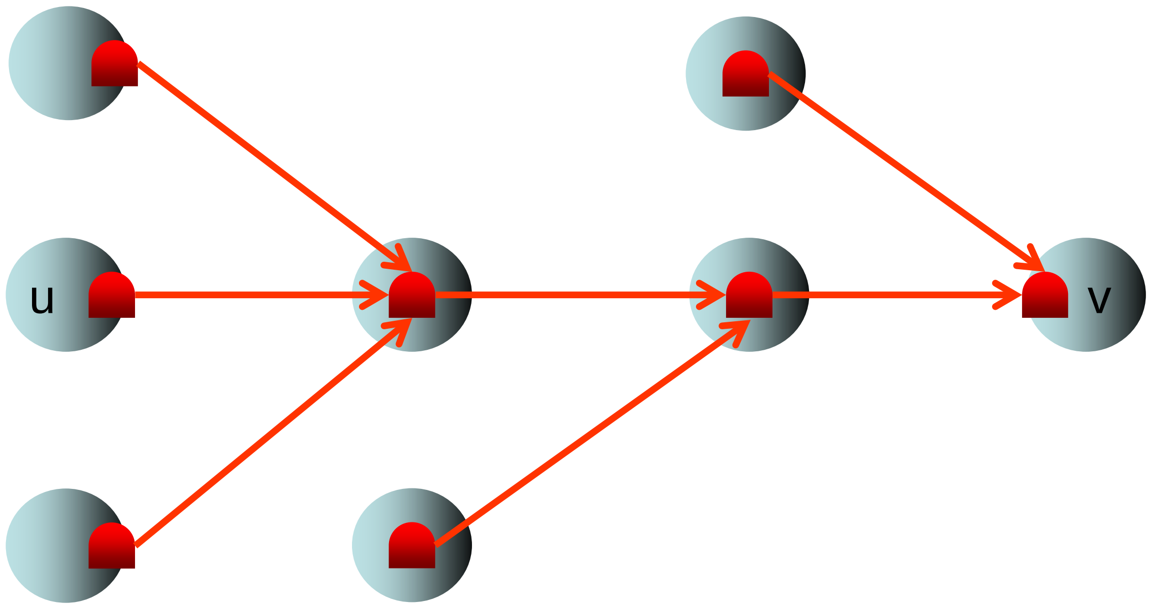
Possible outcome of safe introductions:

In our old graph terminology, the corresponds to the following connections (though there are now dependencies among them):



Safe Primitives

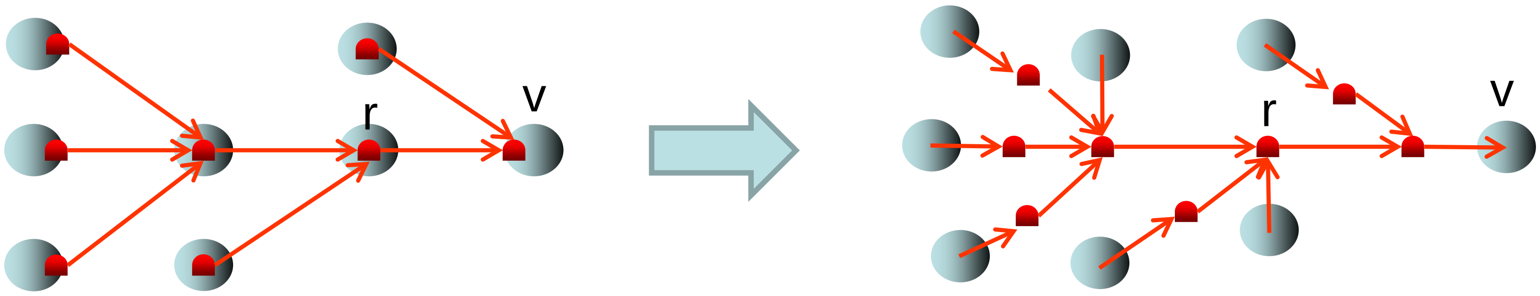
Another possible outcome of safe introductions:



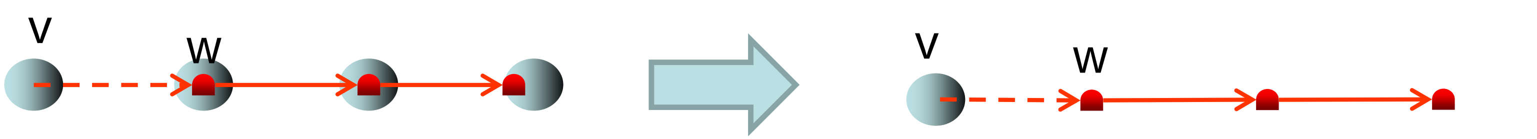
Safe Primitives

Relay graph $G=(V,E_L \cup E_M)$:

- $V=R \cup P$, where R is the set of relays and P is the set of processes
- E_L (explicit edges): set of edges (v,w) where either ($v \in P$ and $w \in R$), or ($v \in R$ and $w \in R$), or ($v \in R$ and $w \in P$)



- E_M (implicit edges): set of edges (v,w) where $v \in P$ and $w \in R$, which represents a message in transit to v with a reference to relay w



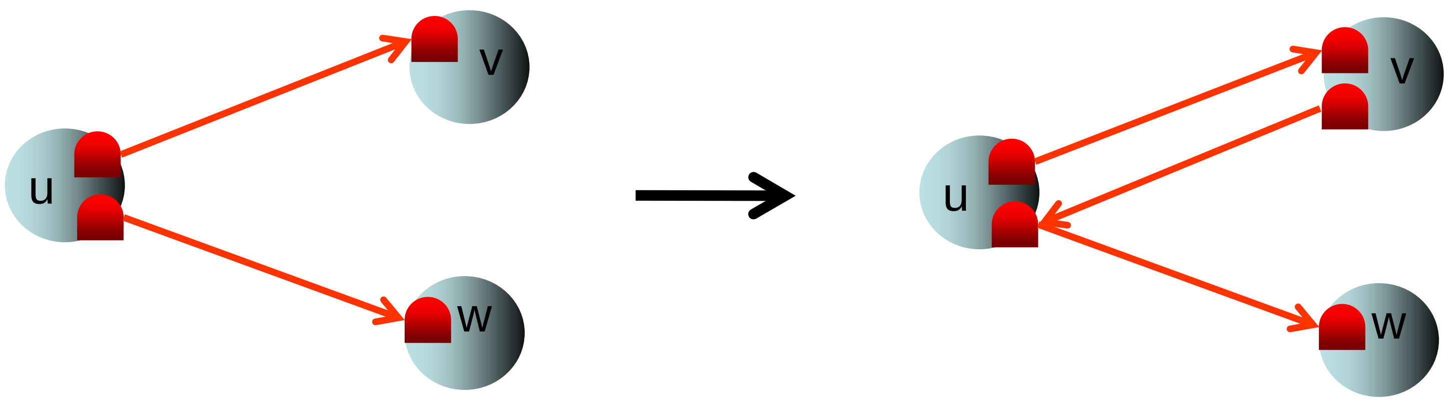
Safe Primitives

A relay graph $G=(R \cup P, E_L \cup E_M)$ is called

- weakly connected if for all pairs $v,w \in P$ there is a path from v to w in G when ignoring the directions of the edges
- strongly connected if for all pairs $v,w \in P$ there is a directed path from v to w in G

Safe Primitives

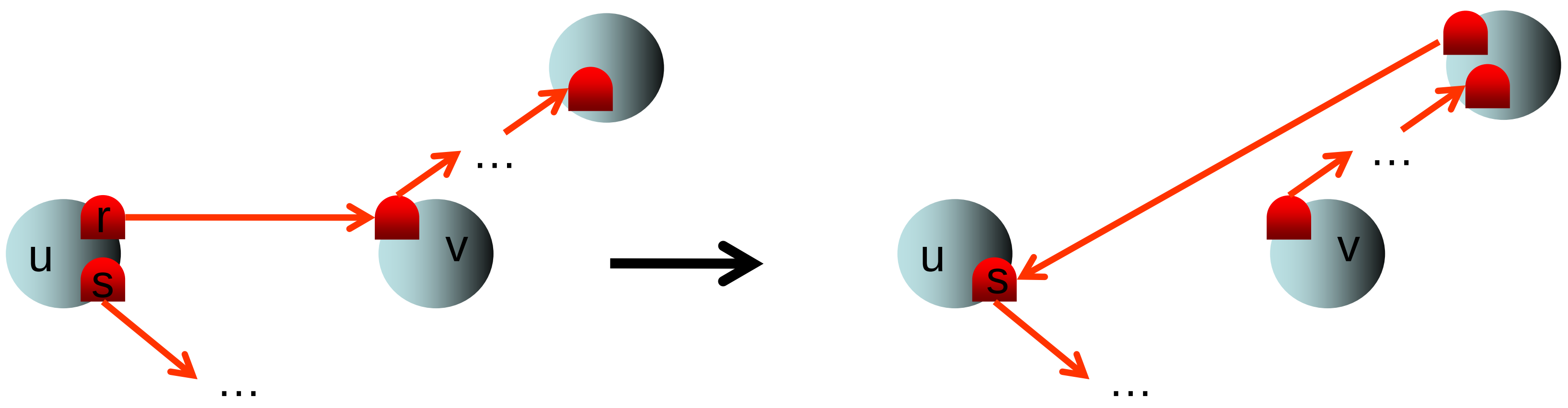
Safe introduction:



Certainly, safe introduction preserves weak (and strong) connectivity in relay graphs as this only adds an edge to G.

Safe Primitives

Safe reversal:

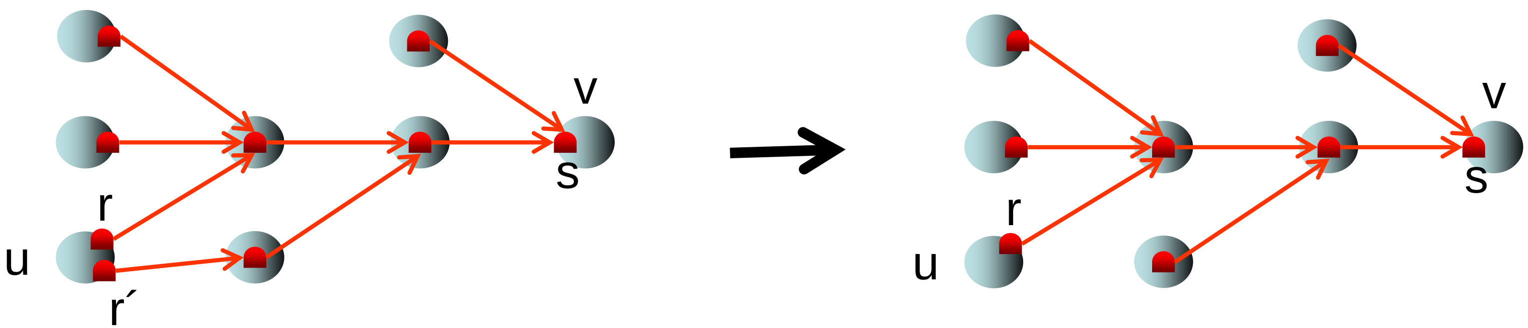


u safely introduces relay s to the process owning the sink relay of r and drops r (if r has no incoming links).

Exercise: safe reversal preserves weak connectivity

Safe Primitives

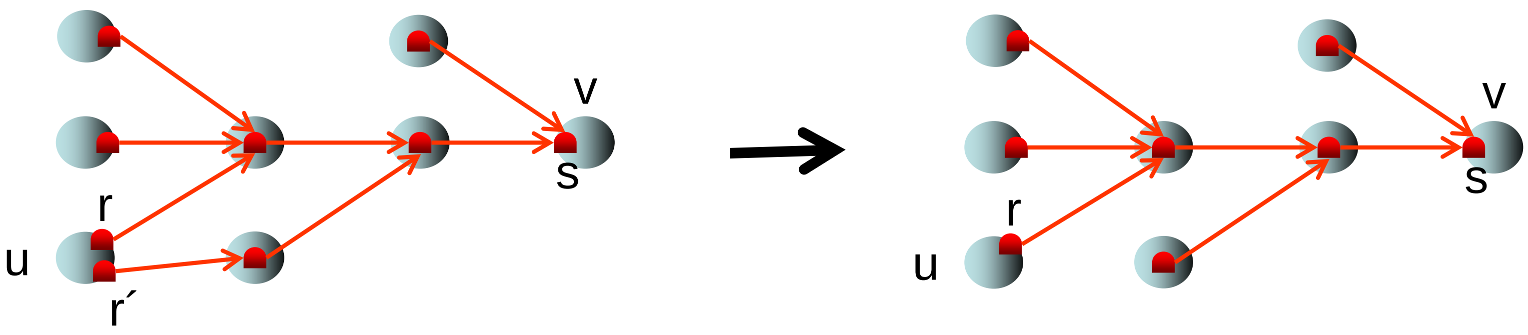
Safe fusion:



Whenever two relays (here, r and r´) of a process (here, u) point to the same sink relay (here, s), one of them can be dropped. Certainly, safe fusion preserves weak and strong connectivity.

Safe Primitives

Safe fusion:



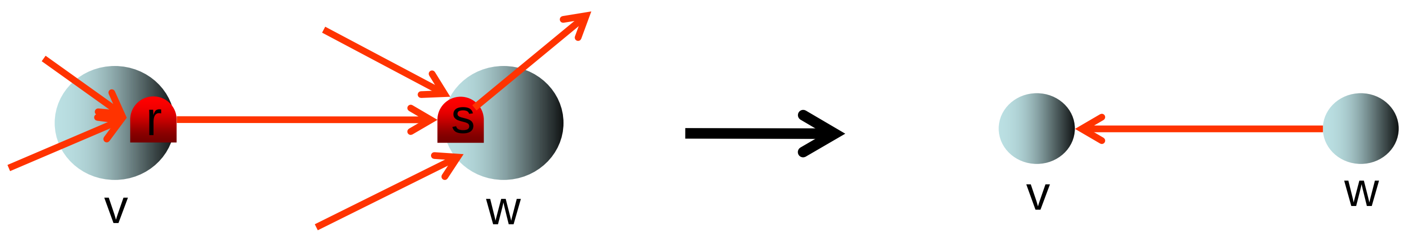
Remark: The TCL only tells processes whether two references point to the same relay or not (not to the same process). This allows processes to maximize anonymity since different relays can be used for different tasks.

Safe Primitives

Theorem 4.3: Safe introduction, fusion, and reversal are universal in a sense that one can get from any weakly connected relay graph $G=(R\cup P,E)$ to any other weakly connected relay graph $G´=(R\cup P,E´)$ (where w.l.o.g. E and $E´$ consist solely of explicit edges).

Proof:

- For any process $v\in P$ let $R(v)$ be the set of all relays local to v.
- Let $G_1=(P,E_1)$ be the graph where $(w,v)\in E_1$ if and only if there is an edge $(r,s)\in E$ with $r\in R(v)$ and $s\in R(w)$. Define $G_2=(P,E_2)$ in the same way for $E´$.



Safe Primitives

Theorem 4.3: Safe introduction, fusion, and reversal are universal in a sense that one can get from any weakly connected relay graph $G=(R\cup P,E)$ to any other weakly connected relay graph $G'=(R\cup P,E')$ (where w.l.o.g. E and E' consist solely of explicit edges).

Proof:

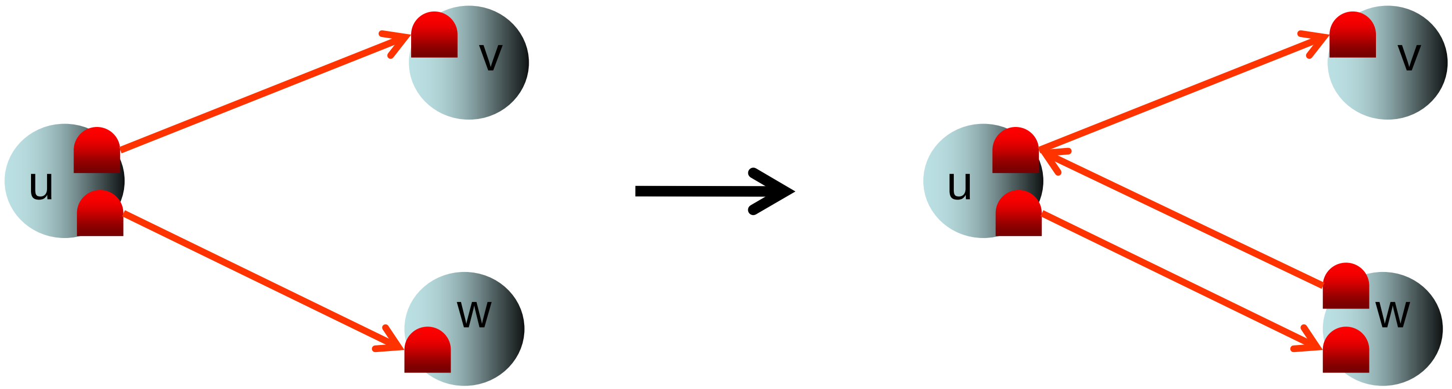
- For any process $v\in P$ let $R(v)$ be the set of all relays local to v.
- Let $G_1=(P,E_1)$ be the graph where $(w,v)\in E_1$ if and only if there is an edge $(r,s)\in E$ with $r\in R(v)$ and $s\in R(w)$. Define $G_2=(P,E_2)$ in the same way for E'.

First, we show how to emulate the standard introduction and delegation rules by our safe rules. The remaining proof then proceeds in three parts:

1. Transform G into G_1.
2. Transform G_1 into G_2.
3. Transform G_2 into G'.

Proof of Theorem 4.3

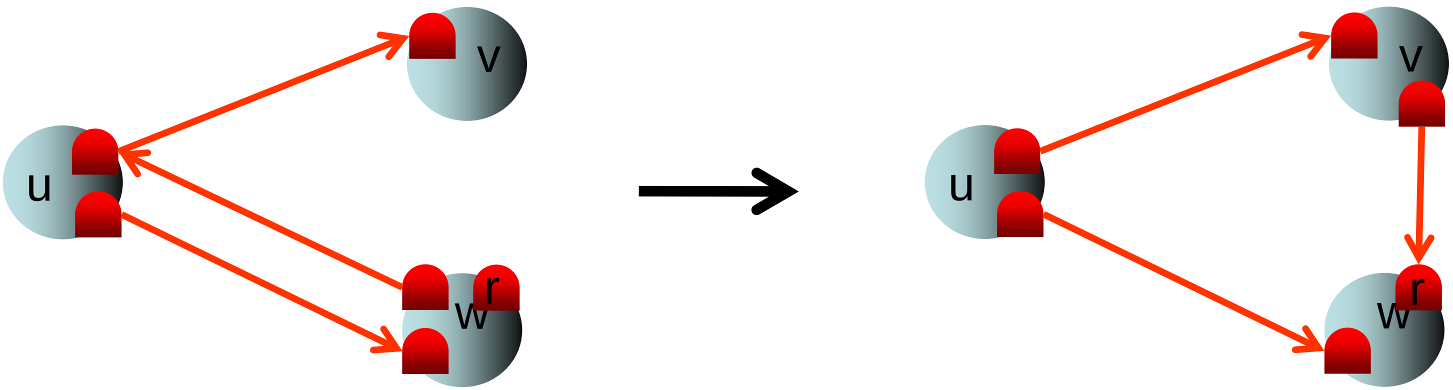
Emulation of introduction rule (u introduces w to v):



First, u introduces w to its relay to v (using the safe introduction rule).

Proof of Theorem 4.3

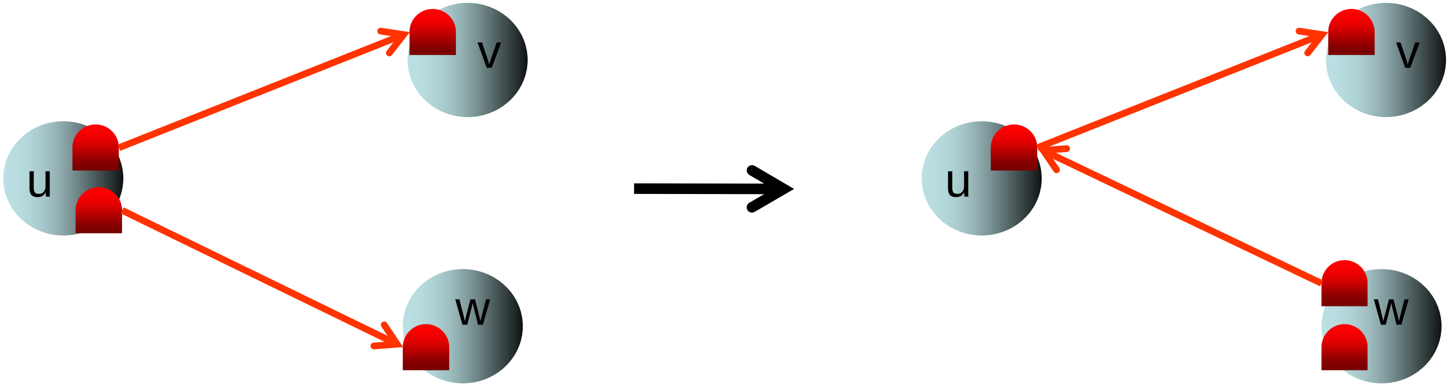
Emulation of introduction rule (u introduces w to v):



Then w establishes a new relay r, sends its reference via u to v and drops its relay to u (which resembles the safe reserval rule).

Proof of Theorem 4.3

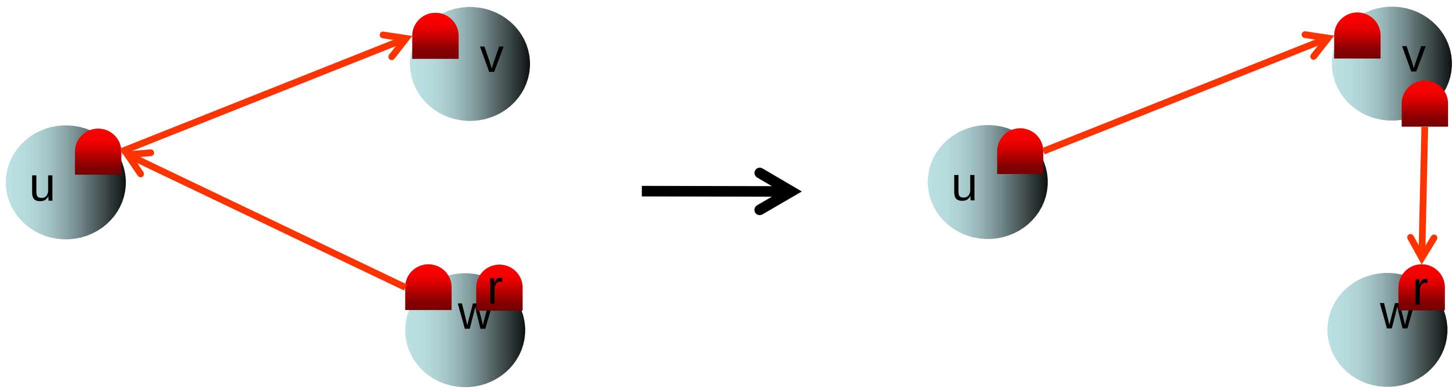
Emulation of delegation rule (u delegates w to v):



First, u introduces w to its relay to v and drops its relay to w (which resembles the safe reversal rule).

Proof of Theorem 4.3

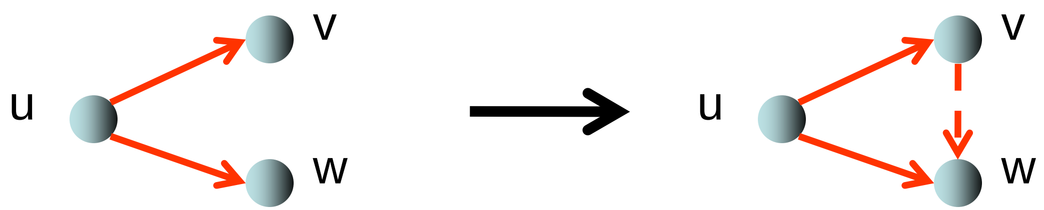
Emulation of delegation rule (u delegates w to v):



Then w establishes a new relay r, sends its reference to u (which will be forwarded to v) and drops its relay to u (which resembles the safe reserval rule).

Proof of Theorem 4.3

Remark: Since now w is always directly involved whenever it is introduced or delegated to a node v, w can also ensure that no corrupted information about it is sent to v. This is not guaranteed by the old way introduction and delegation is handled:

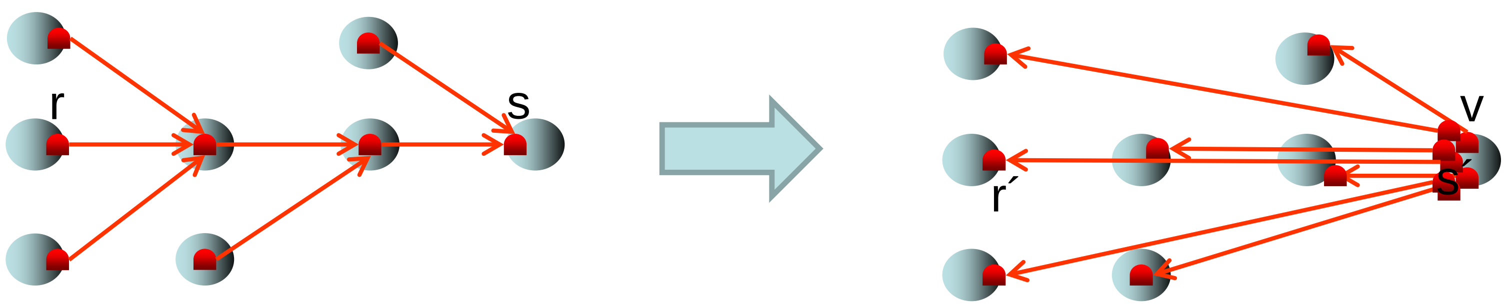


u sends a message to v containing w´s reference.

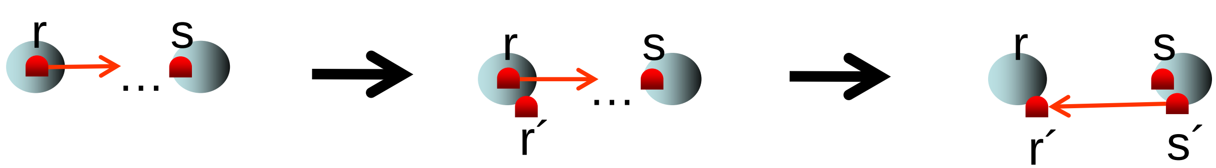
Proof of Theorem 4.3

Transforming G into G_1:

First, transform any relay tree in the following way starting with the most distant relays r from s



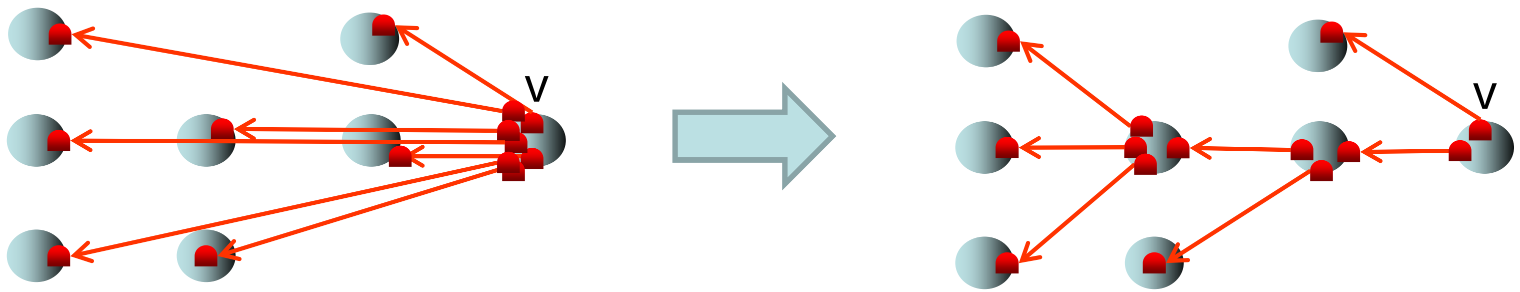
using safe reserval for any pair (r,s):



Proof of Theorem 4.3

Transforming G into G_1:

Then, transform the star back into the original tree, but with reversed, isolated edges



using the safe rules emulating the standard delegation rule. Since at the end just isolated edges are left, we can simplify that to our standard graph on processes, G_1.

Proof of Theorem 4.3

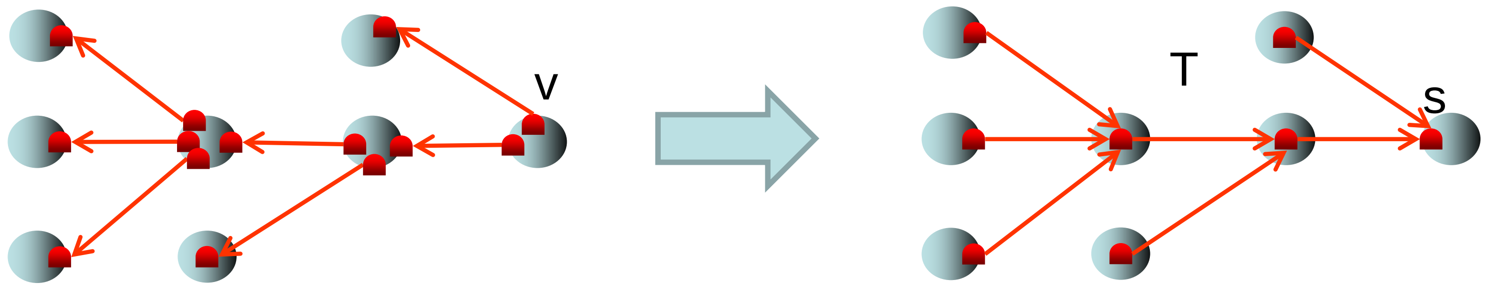
Transforming G_1 into G_2:

This follows from Theorem 4.2 since introduction, delegation, fusion, and reversal can be emulated by our safe primitives.

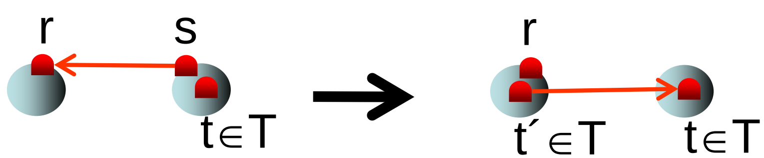
Proof of Theorem 4.3

Transforming G_2 into G':

For any relay tree T in G´, transform the individual edges belonging to it in G_2 into that tree starting with the closest relays to v

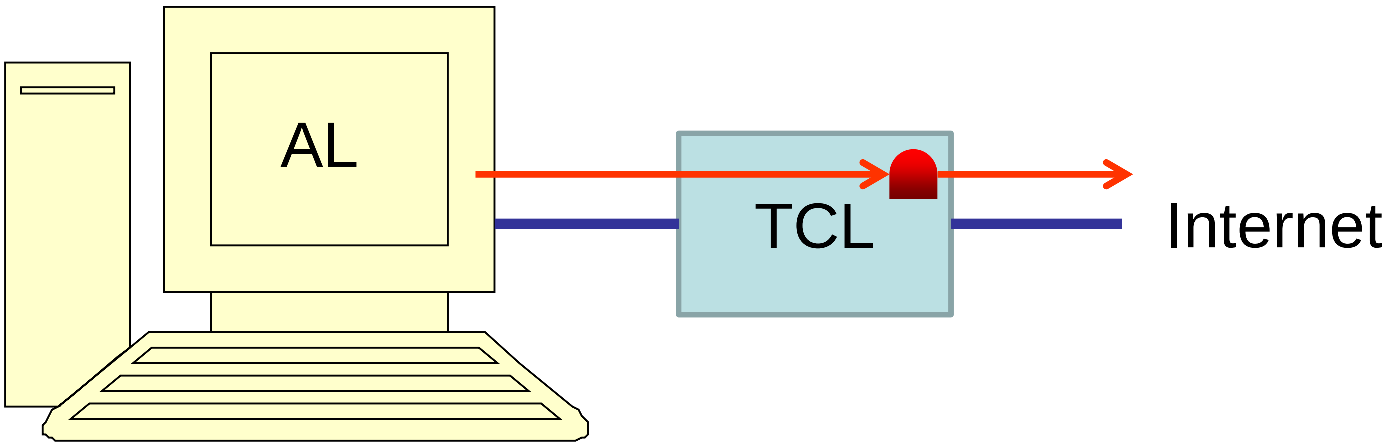


by using safe reserval for any pair (r,s):

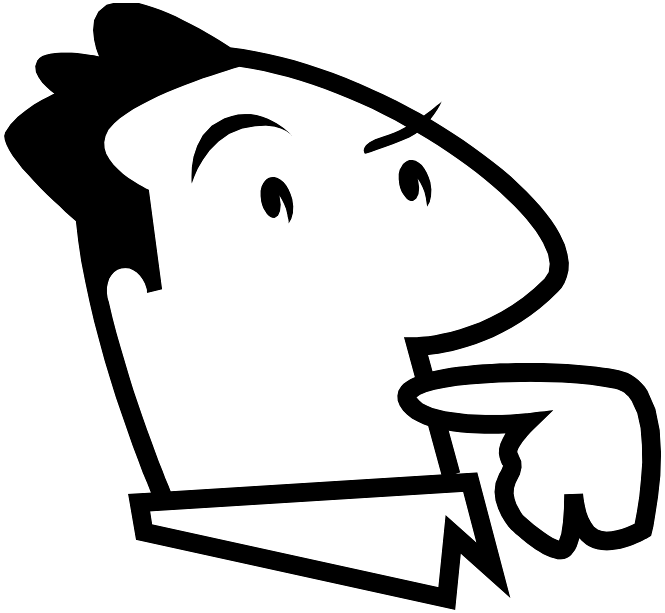


Safe Primitives

Embedding into Trusted Communication Model (TCM):



- AL manages references to relays that may be passed on to establish a connection to that relay
- TCL manages relays (on top of TCP/IP)



Questions?