Towards A Formal Theory of On Chip Communications in the ACL2 Logic

Julien Schmaltz

Saarland University - Computer Science Department

Saarbrücken, Germany

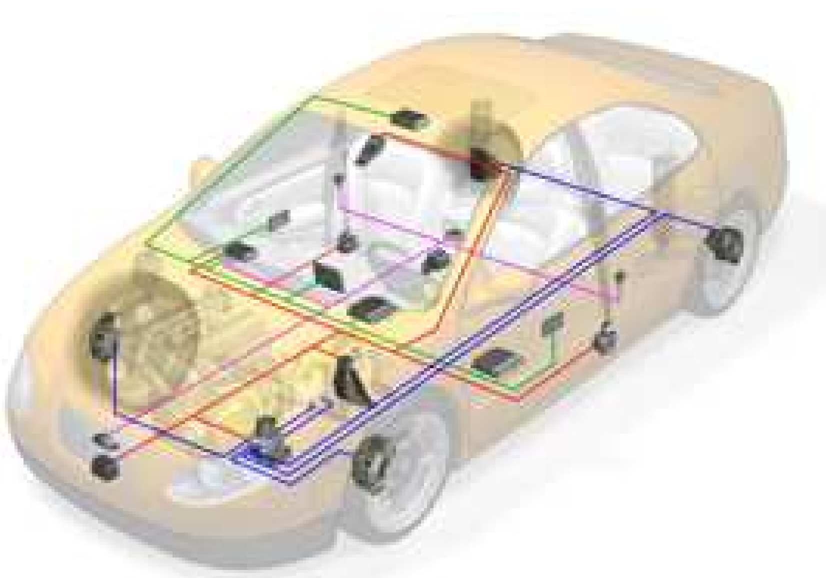
Dominique Borrione

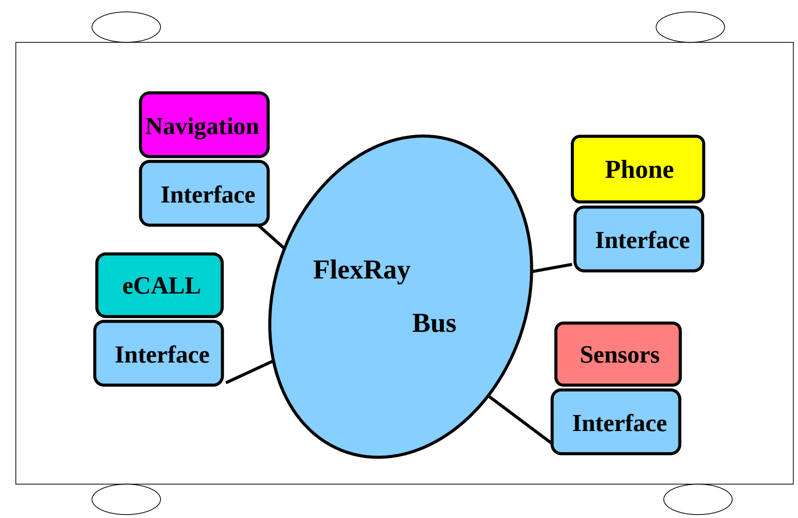
TIMA Laboratory - VDS Group

Grenoble, France

A Motivation Example

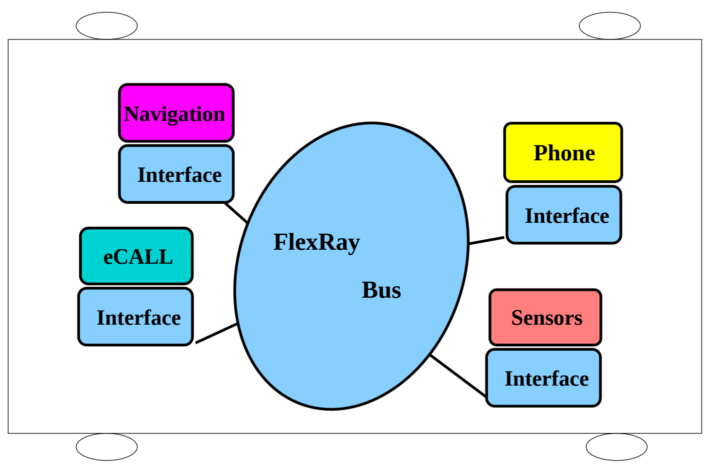
- eCall
  - Automatic emergency call system
  - A phone call is automatically emitted when car sensors detect an accident





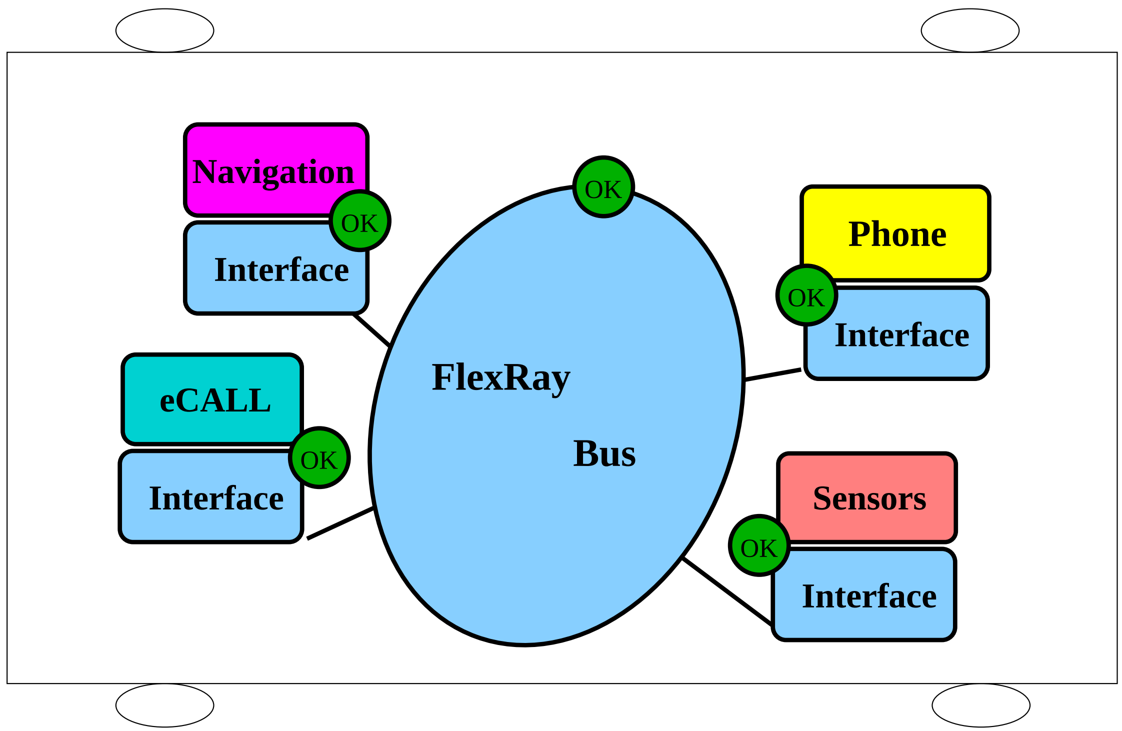
FlexRay Bus

- Basic protocol
  - Idle units send 1, to start send 0
  - “Sync edges” at each byte (from 1 to 0)
- Deterministic scheduling
  - Time is divided into rounds
  - Each unit has one slot per round



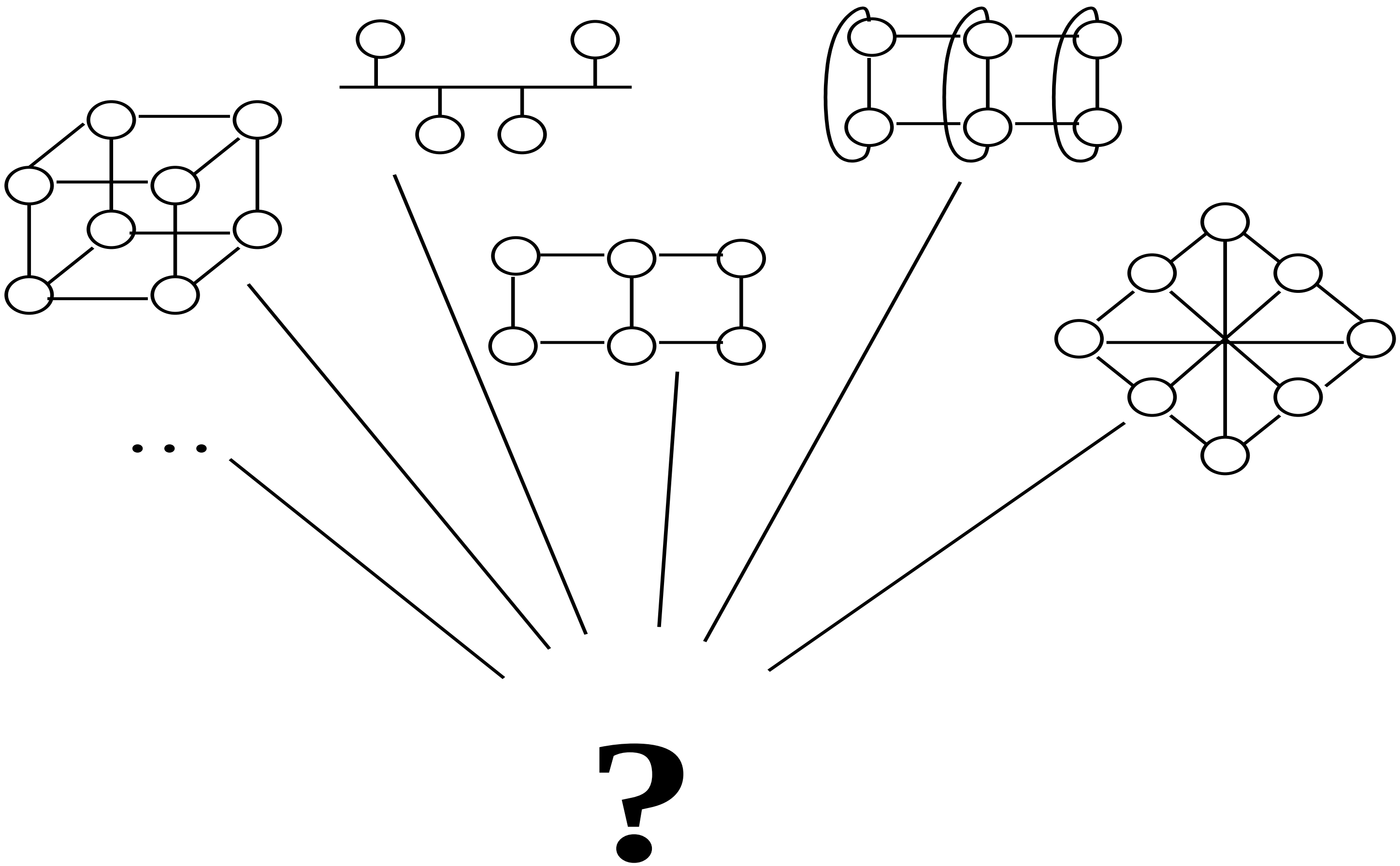
Verification

- Proof of each component
- Proof of their interconnection



Global Objective

One model for **all** architectures



Contribution

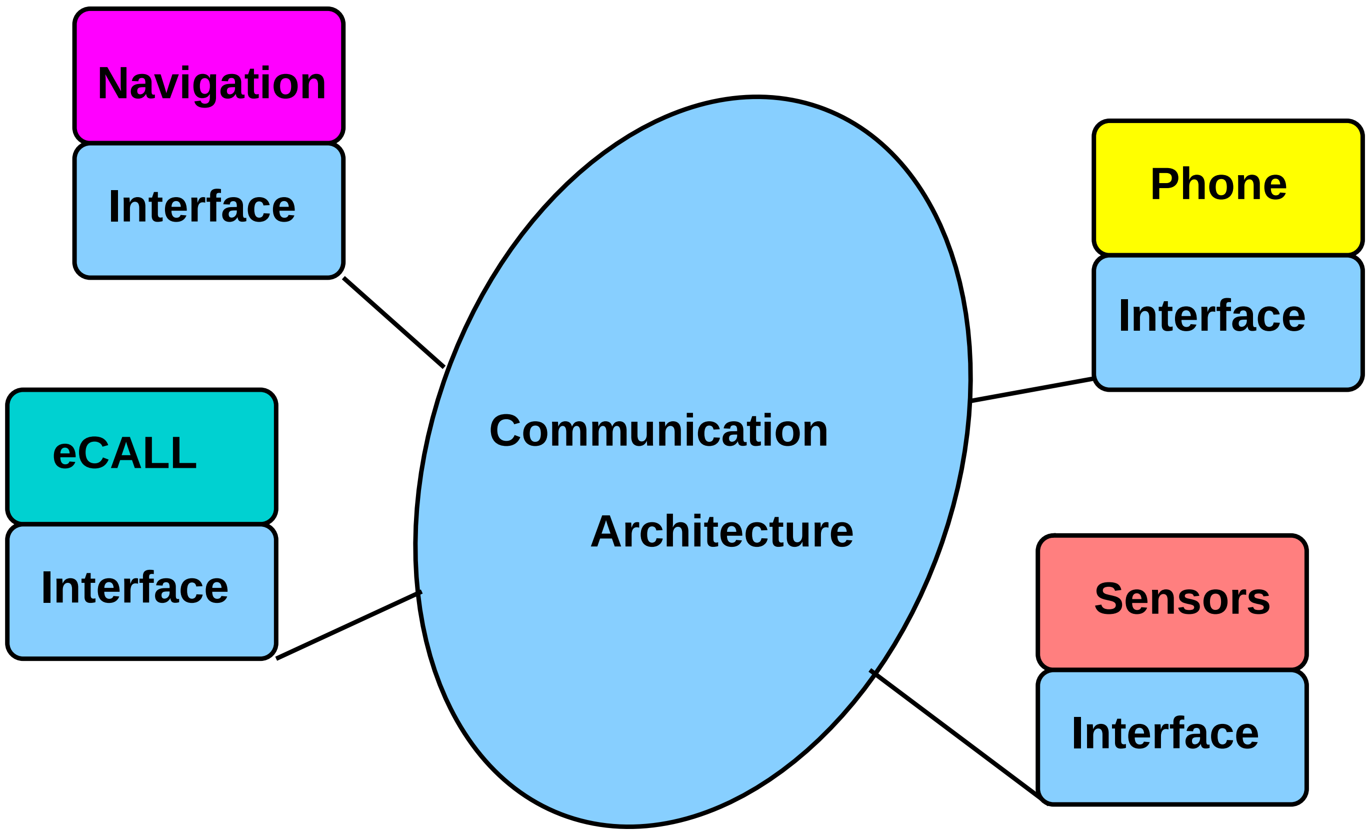
A functional formalism for communications: $GeNoC$ (Generic Network on Chip)

- Identifies the essential constituents and their properties
- Formalizes the interactions between them
- Correctness of the system is a consequence of the essential properties of the constituents
- **Mechanized support in ACL2**
  - **Encapsulation allows abstraction**
  - **Functional instantiation generates proof obligations automatically**

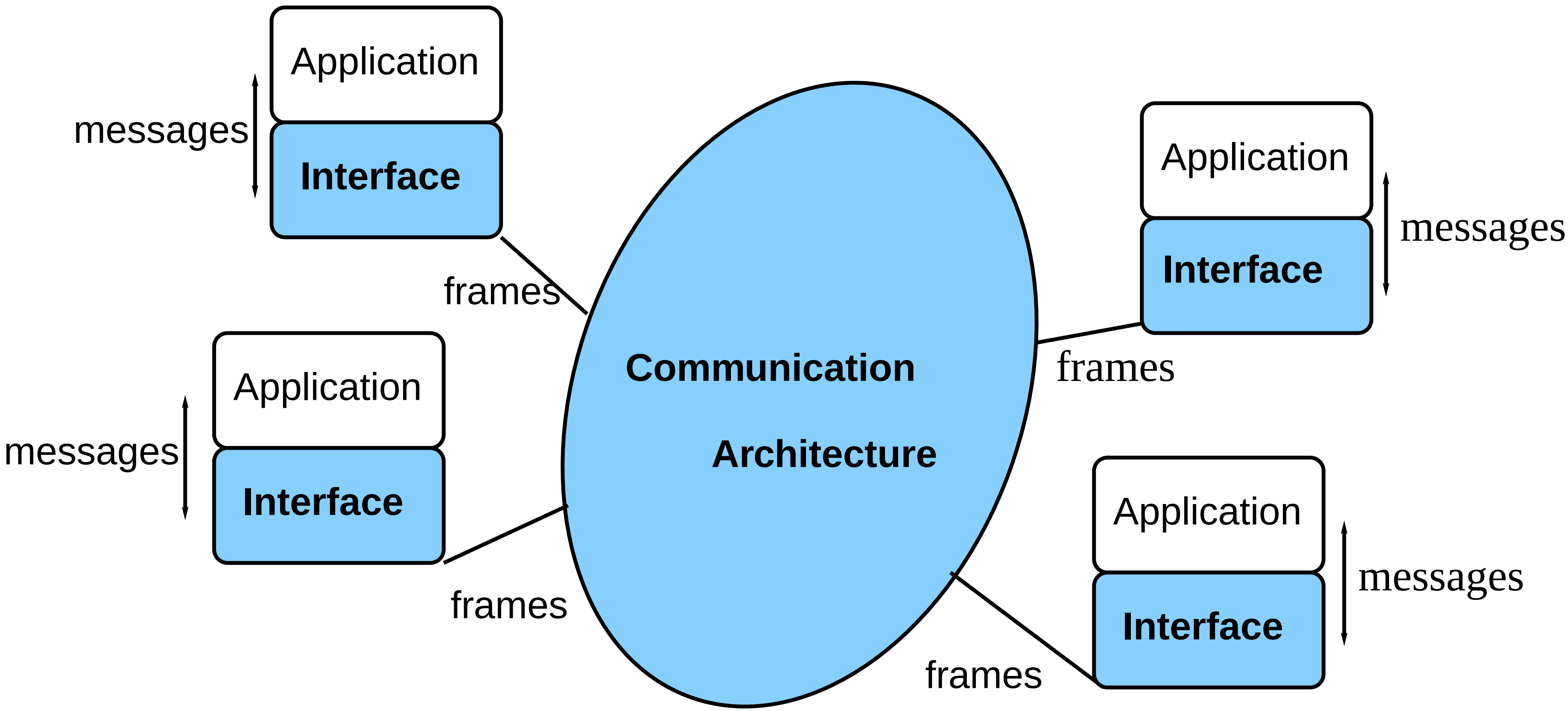
Outline

- Communication Principles
- $GeNoC$ Definition and Correctness
- ACL2 Theorem/Removing Quantifiers
- Abstraction using Encapsulation
- Applications of $GeNoC$

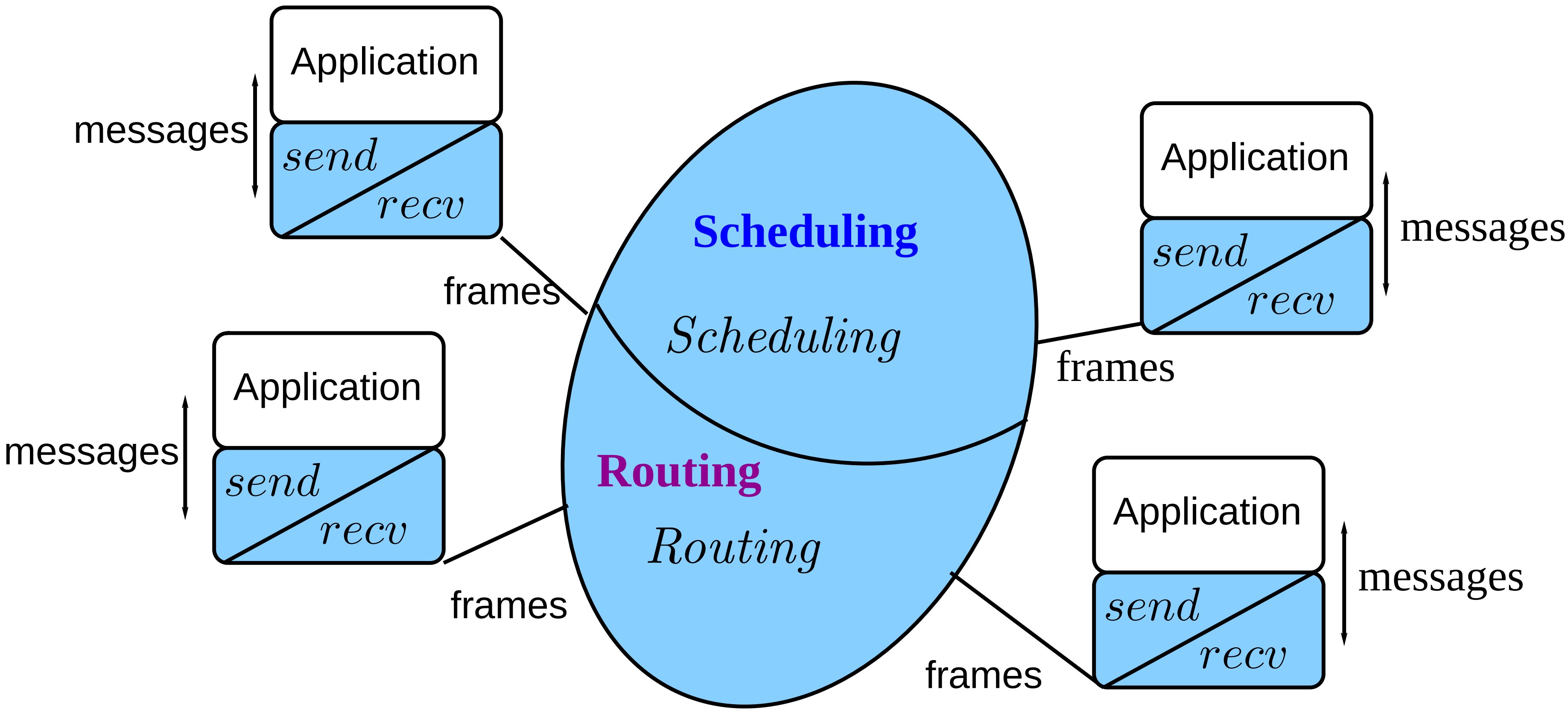
A Unifying Model



A Unifying Model

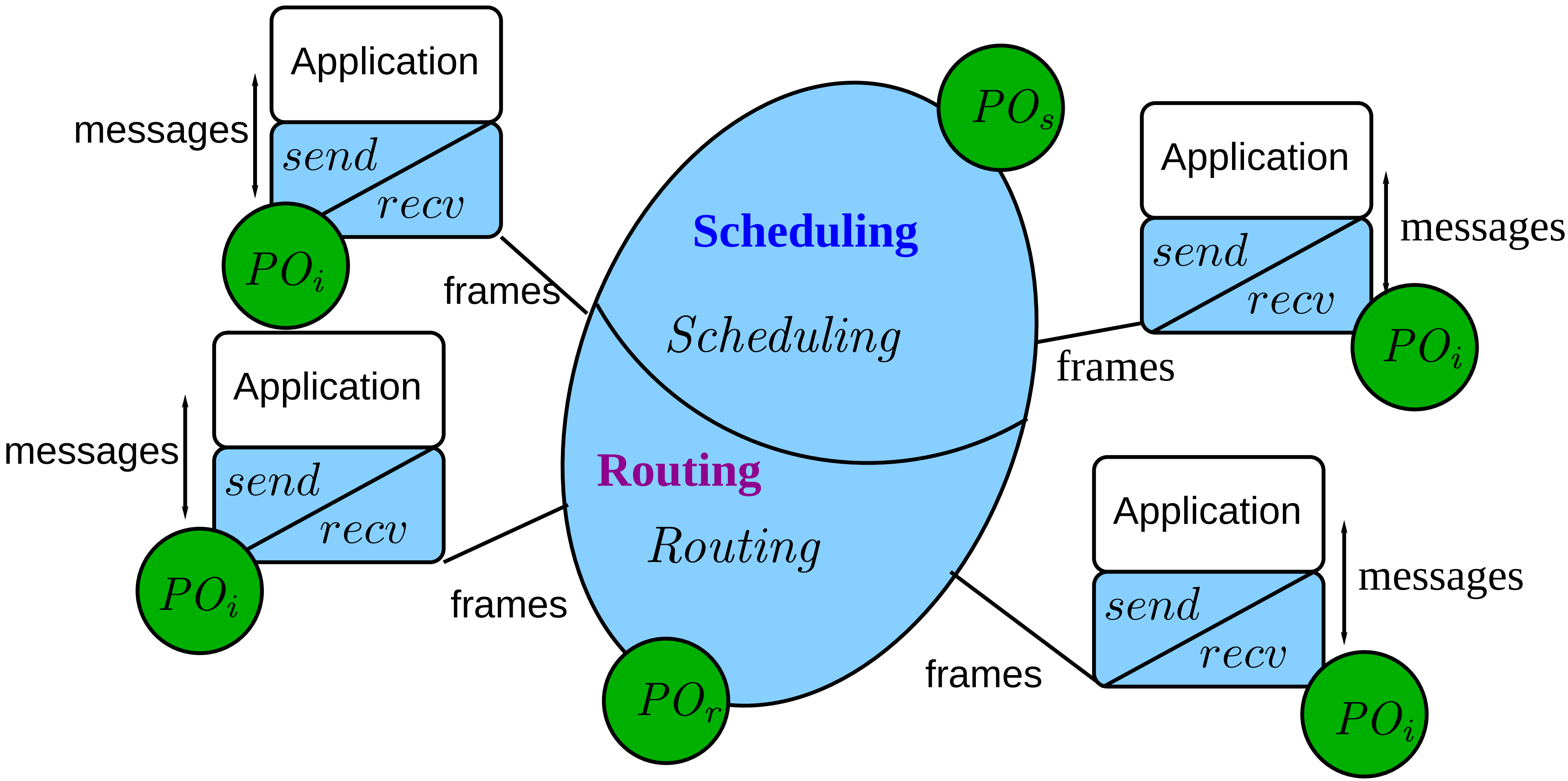


Functional Modeling

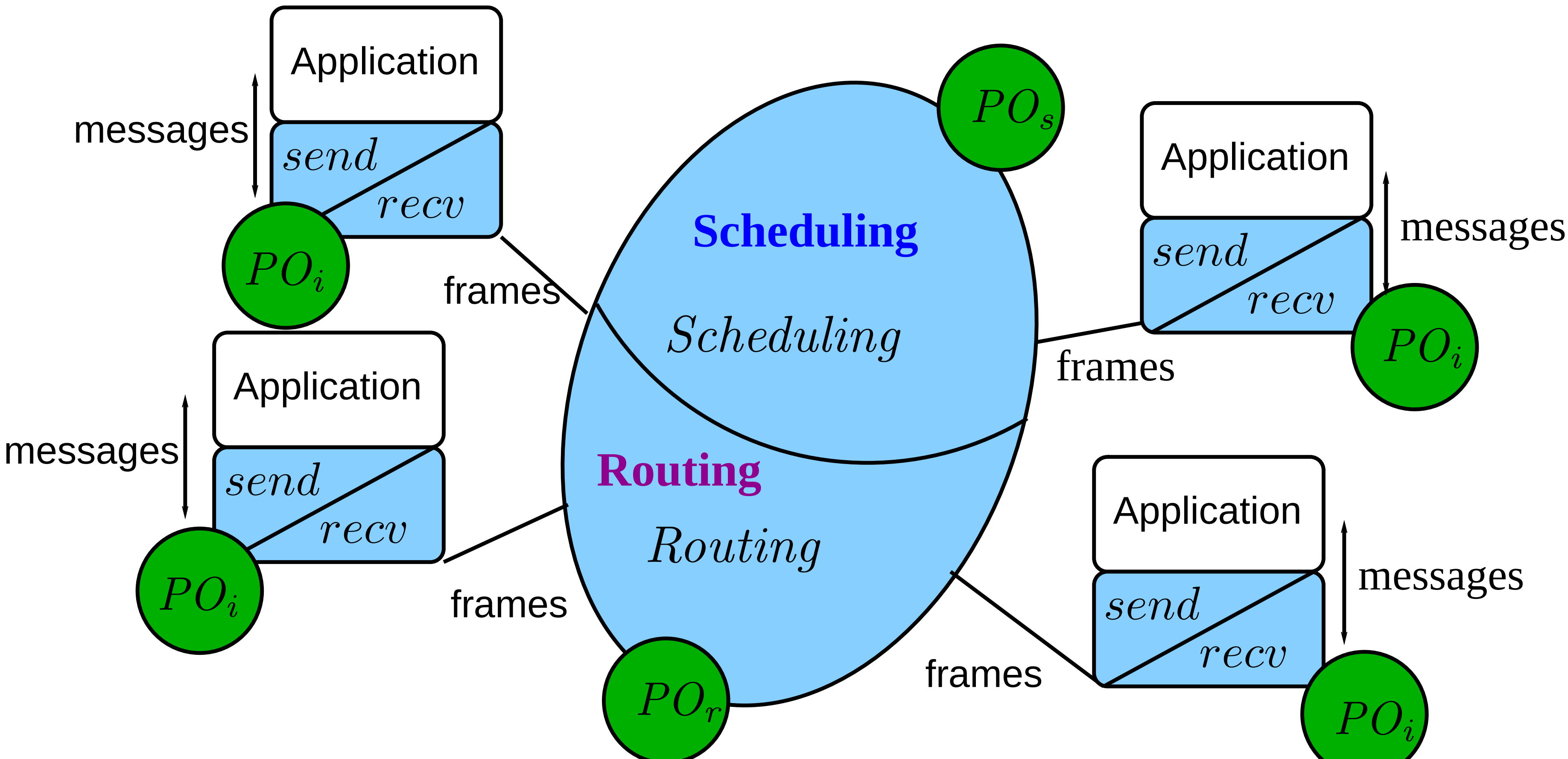


$$\text{System} = \mathcal{F}(Routing, Scheduling, recv, send)$$

Proof Obligations

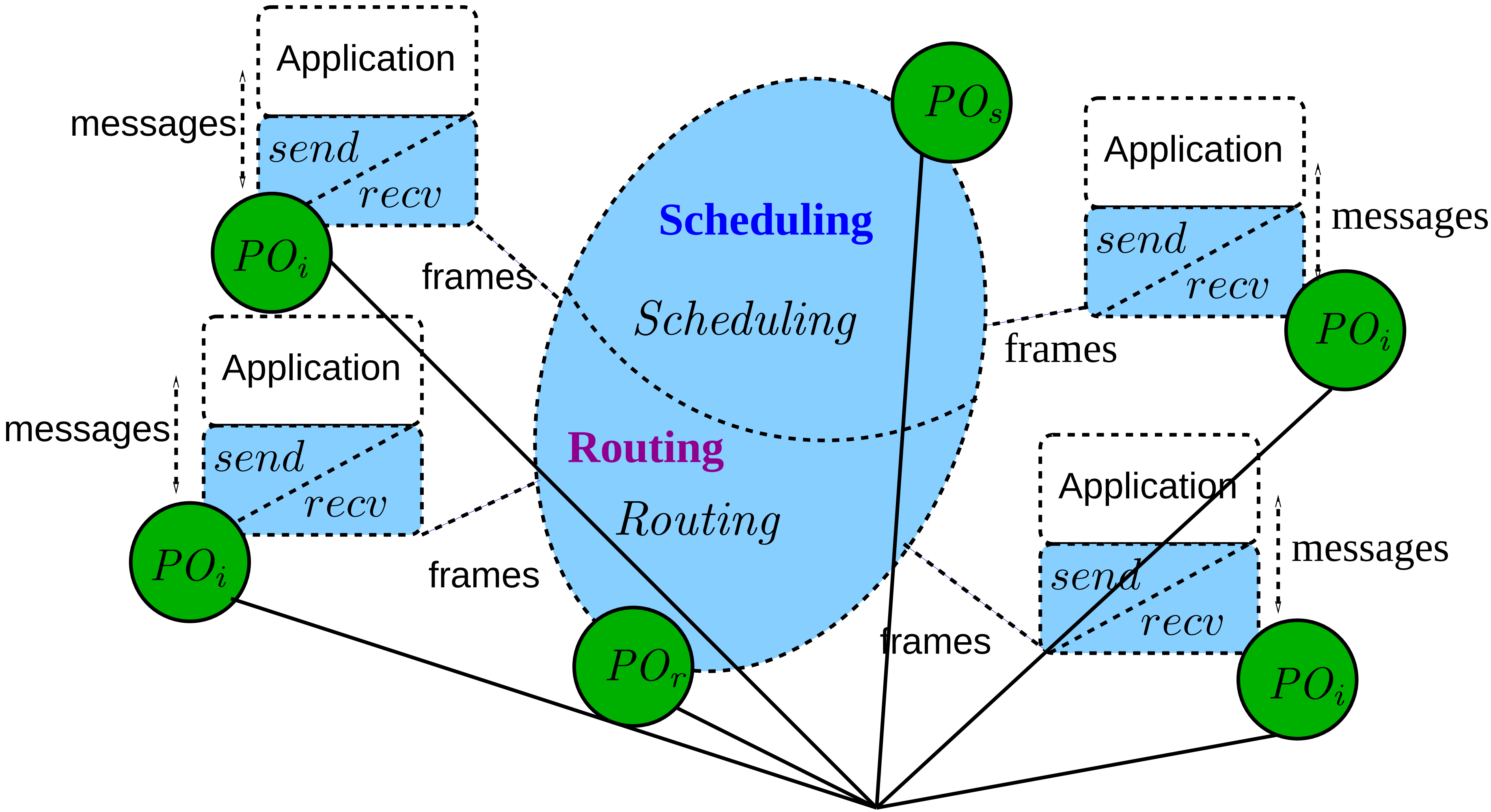


System Theorem



Thm $\sim$ messages reach their destination

System Theorem



Thm $\sim$ messages reach their destination

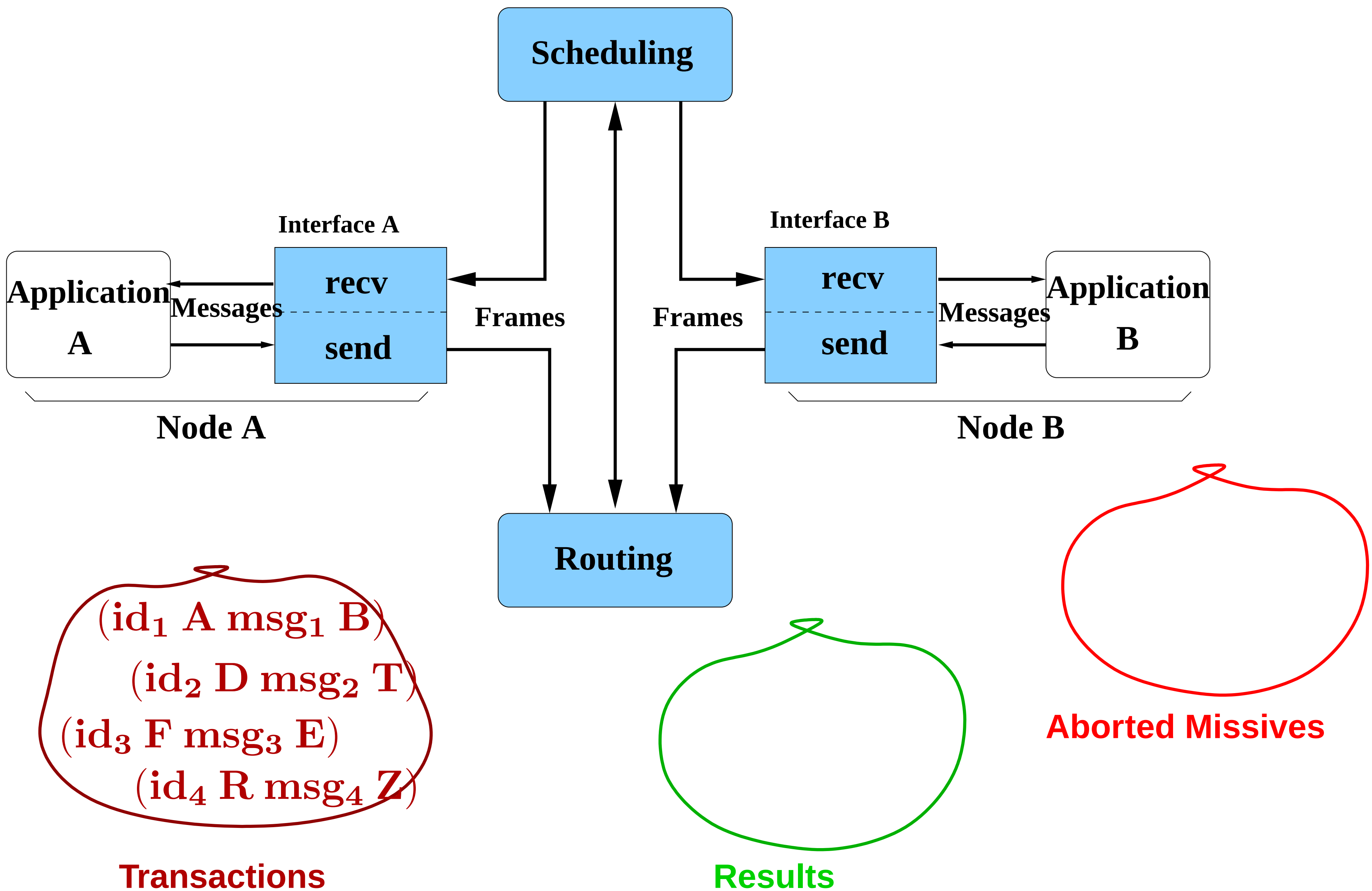
Outline

- Communication Principles
- $GeNoC$ Definition and Correctness
- ACL2 Theorem/Removing Quantifiers
- Abstraction using Encapsulation
- Applications of $GeNoC$

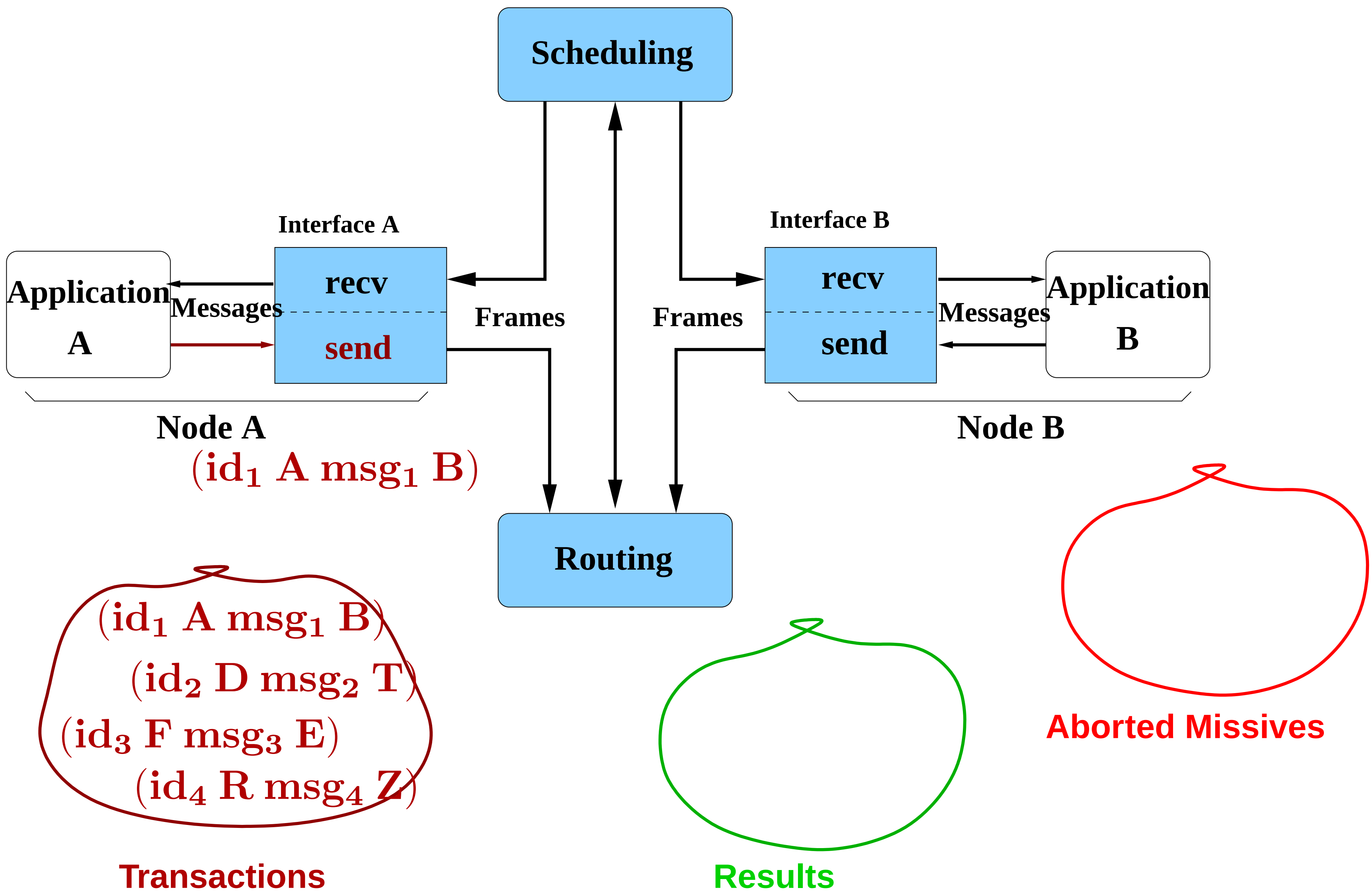
Overall Modeling Principles

- Function $GeNoC$
  - takes the list of pending communications
  - returns the list of results and the list of aborted communications
- Transactions
  - A transaction represents a pending communication, *i.e.* the intention of A of sending msg to B
  - It is a 4-tuple $(id\ A\ msg\ B)$

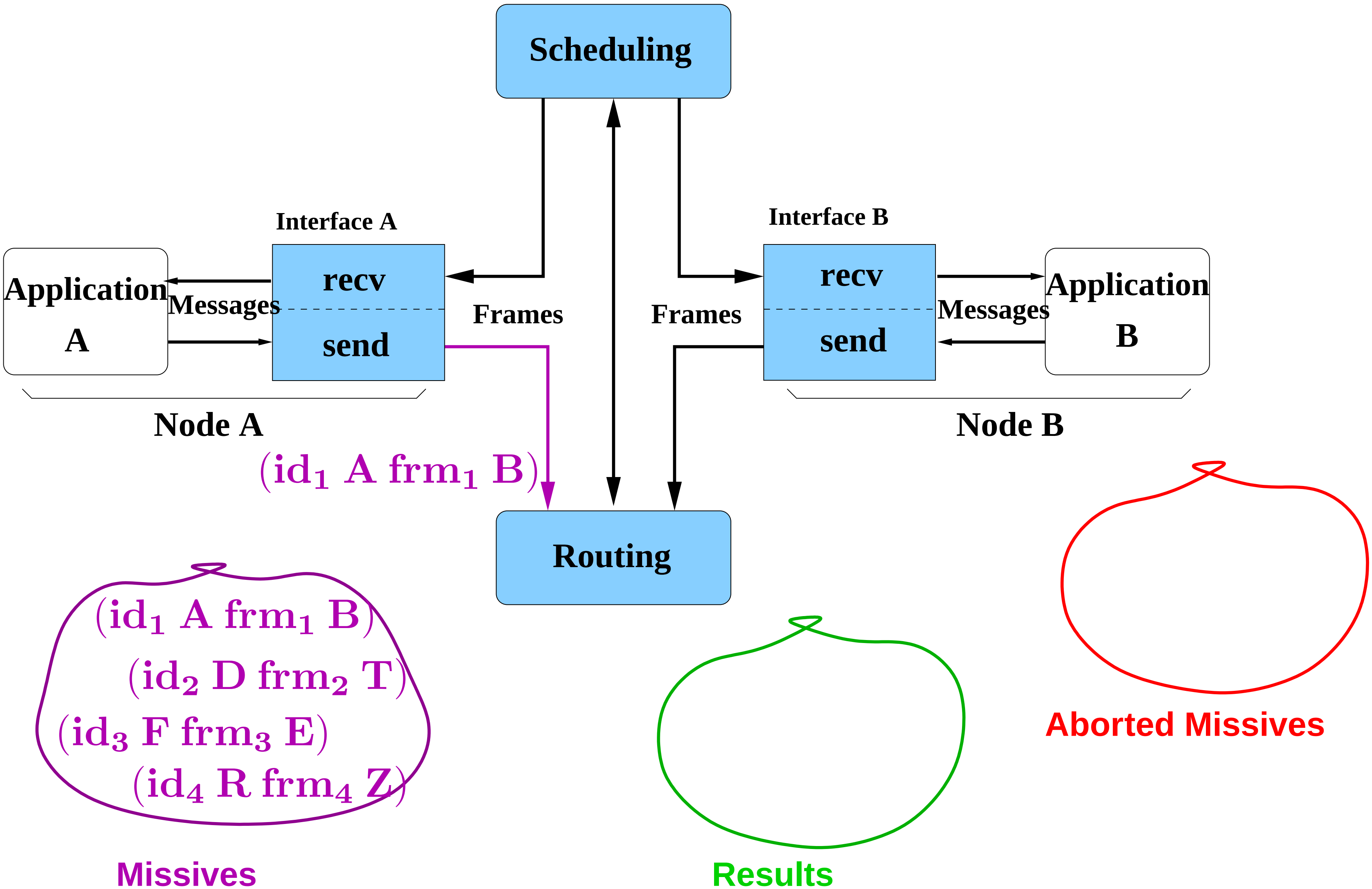
Function $GeNoC$



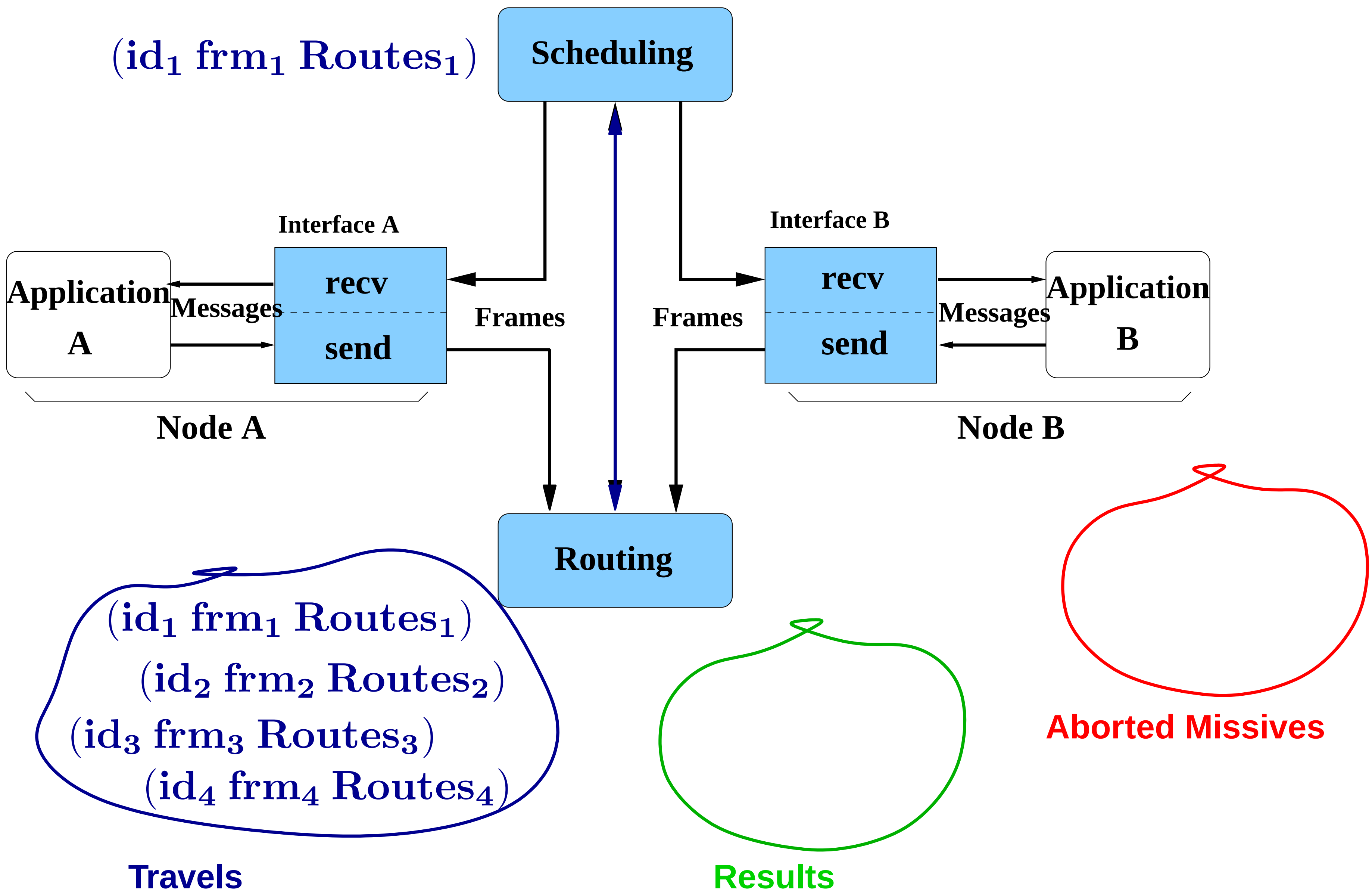
From transactions to missives



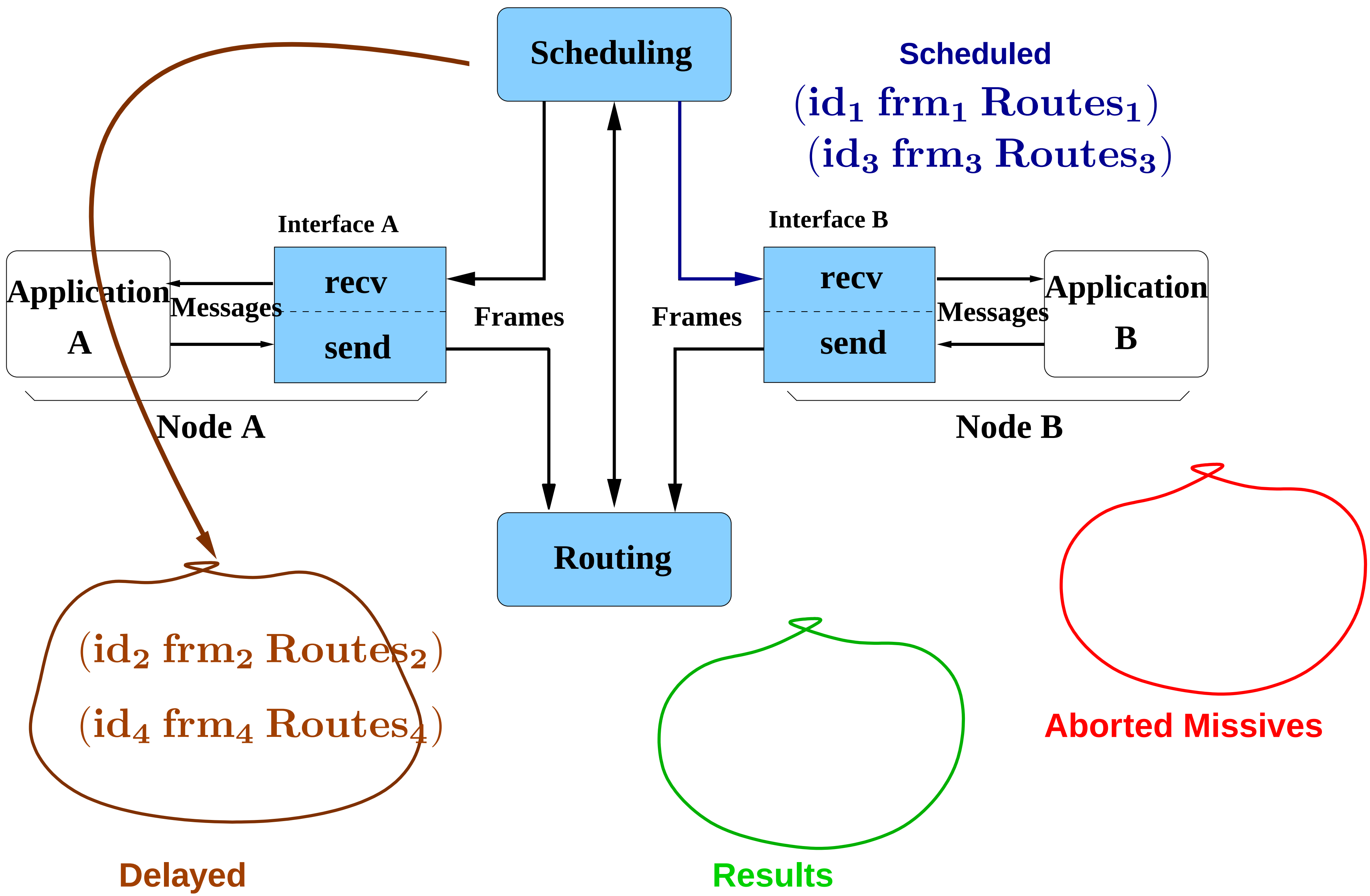
From transactions to missives



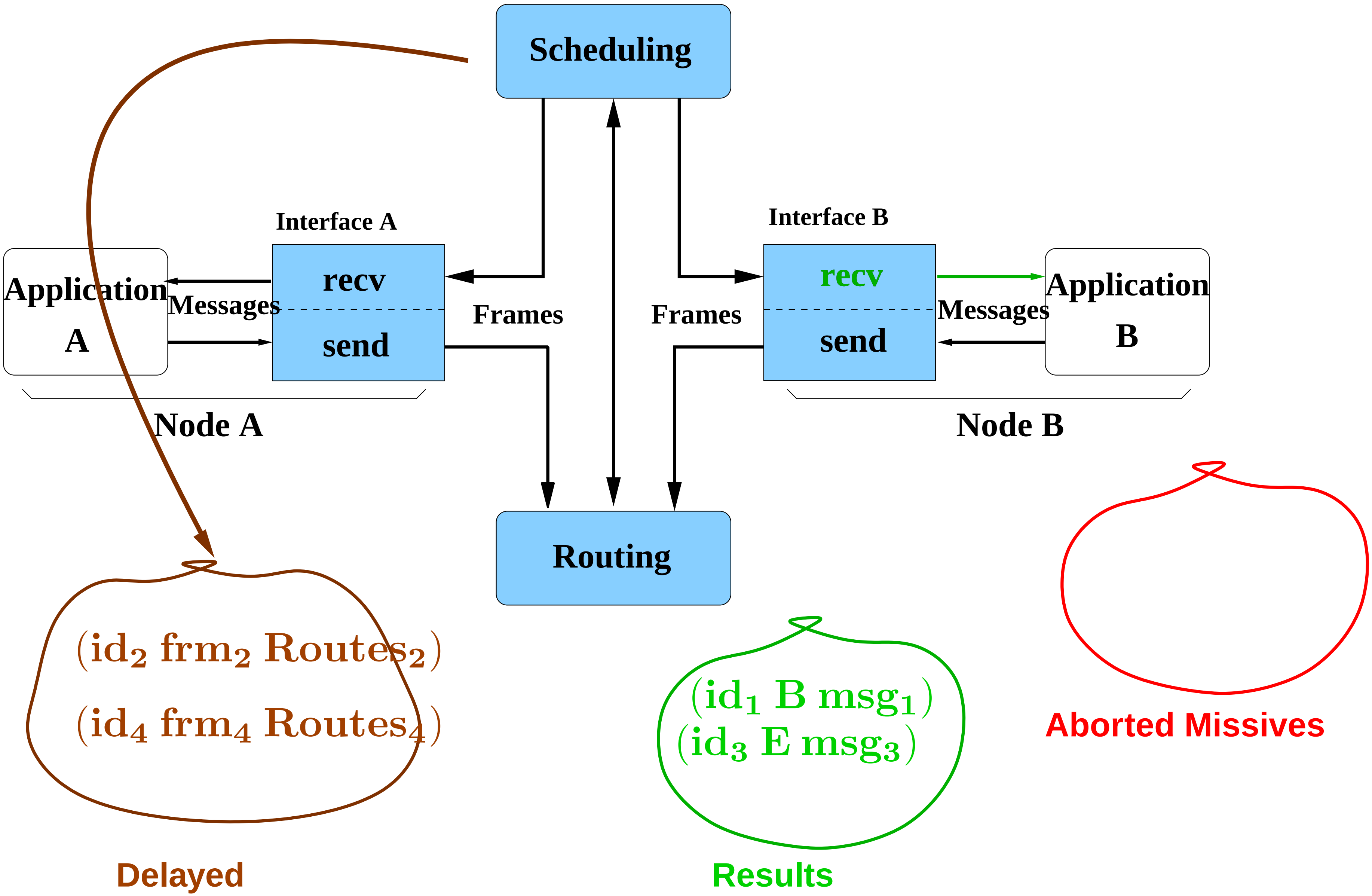
Routing Algorithm



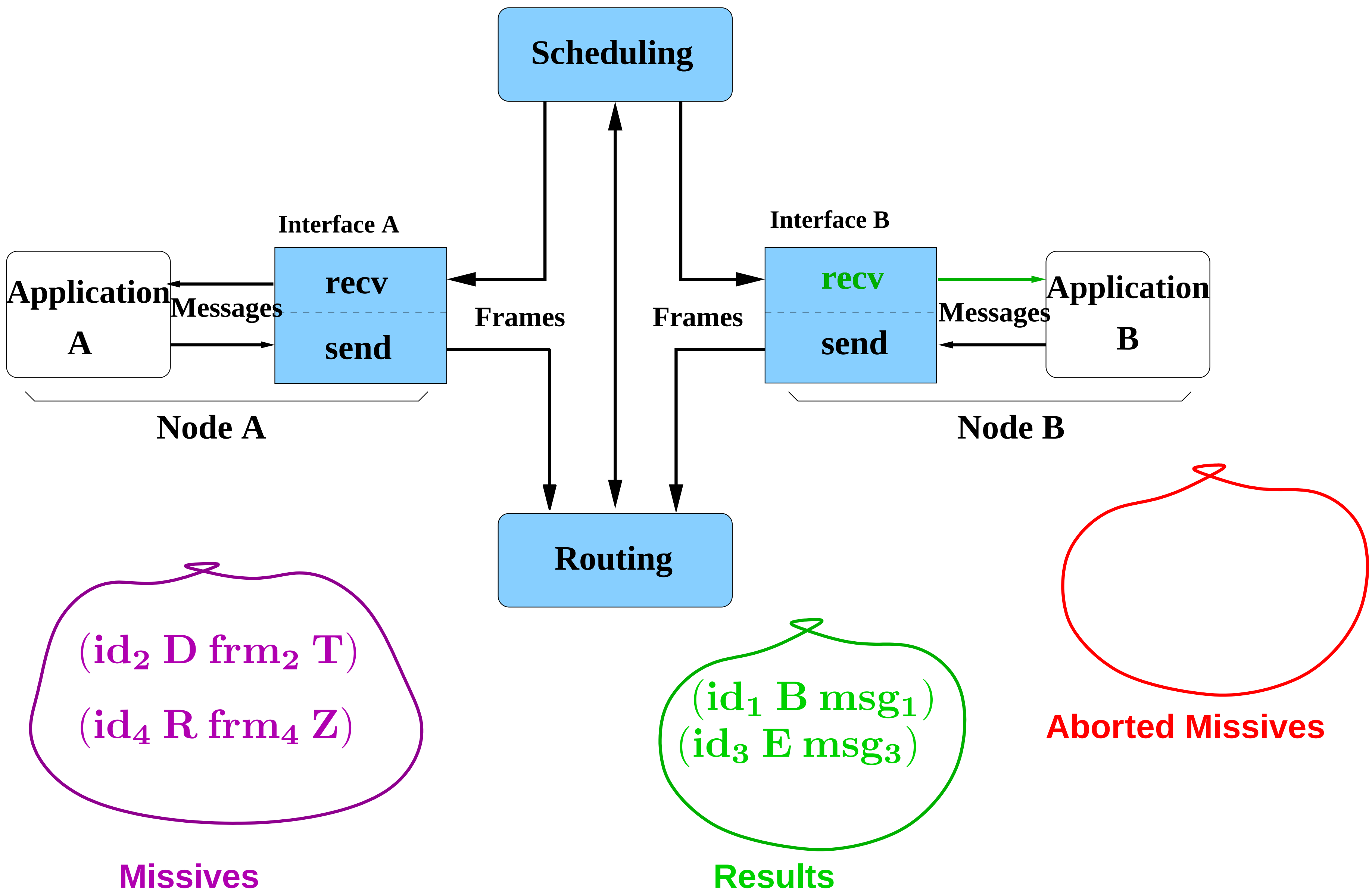
Scheduling Policy



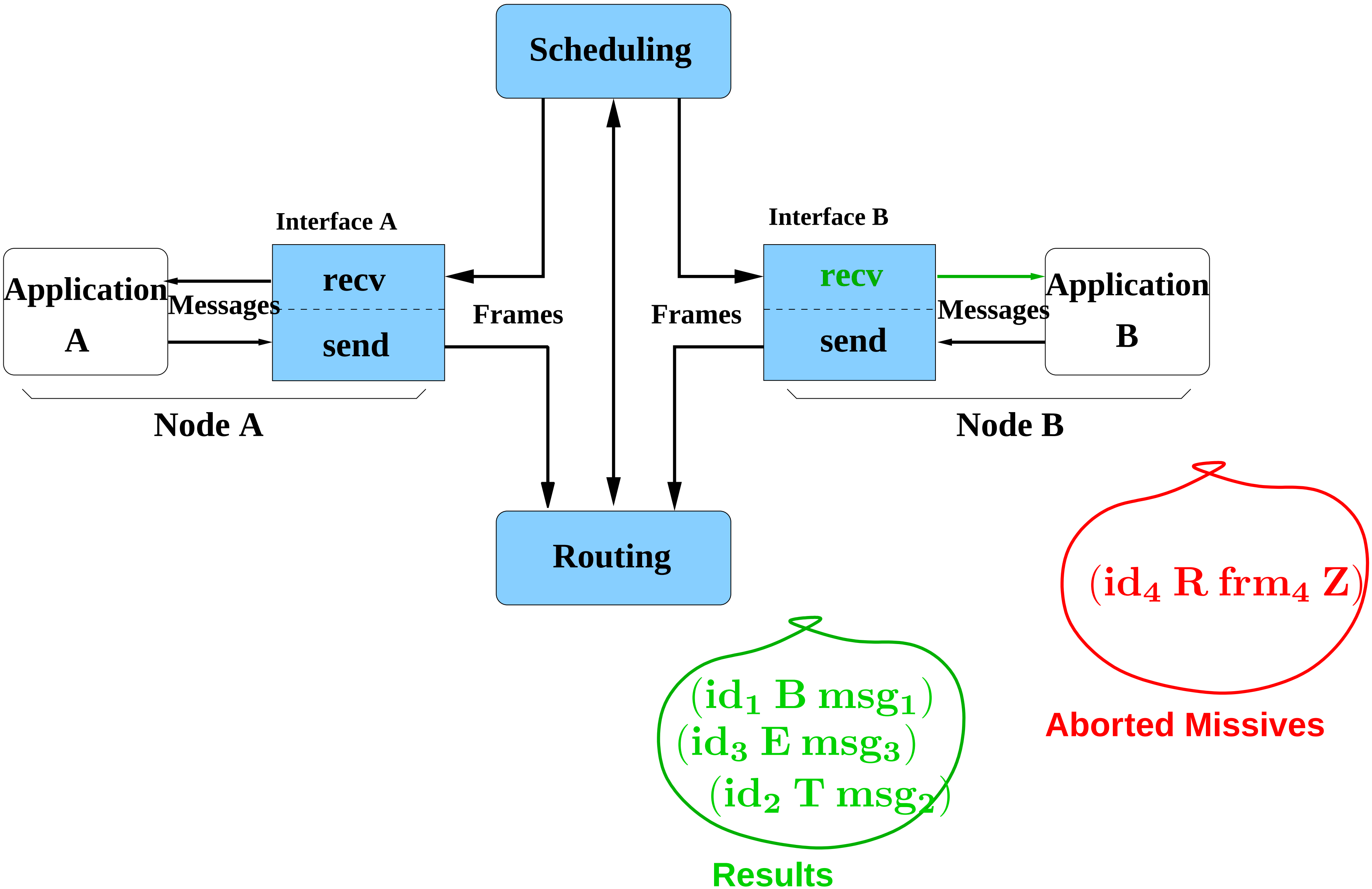
Results


Scheduling
Interface A
Interface B
Application A
recv
send
Messages
Frames
Frames
recv
send
Messages
Application B
Node A
Node B
Routing
$(id_2\ frm_2\ Routes_2)$
$(id_4\ frm_4\ Routes_4)$
$(id_1\ B\ msg_1)$
$(id_3\ E\ msg_3)$
Aborted Missives
Delayed
Results

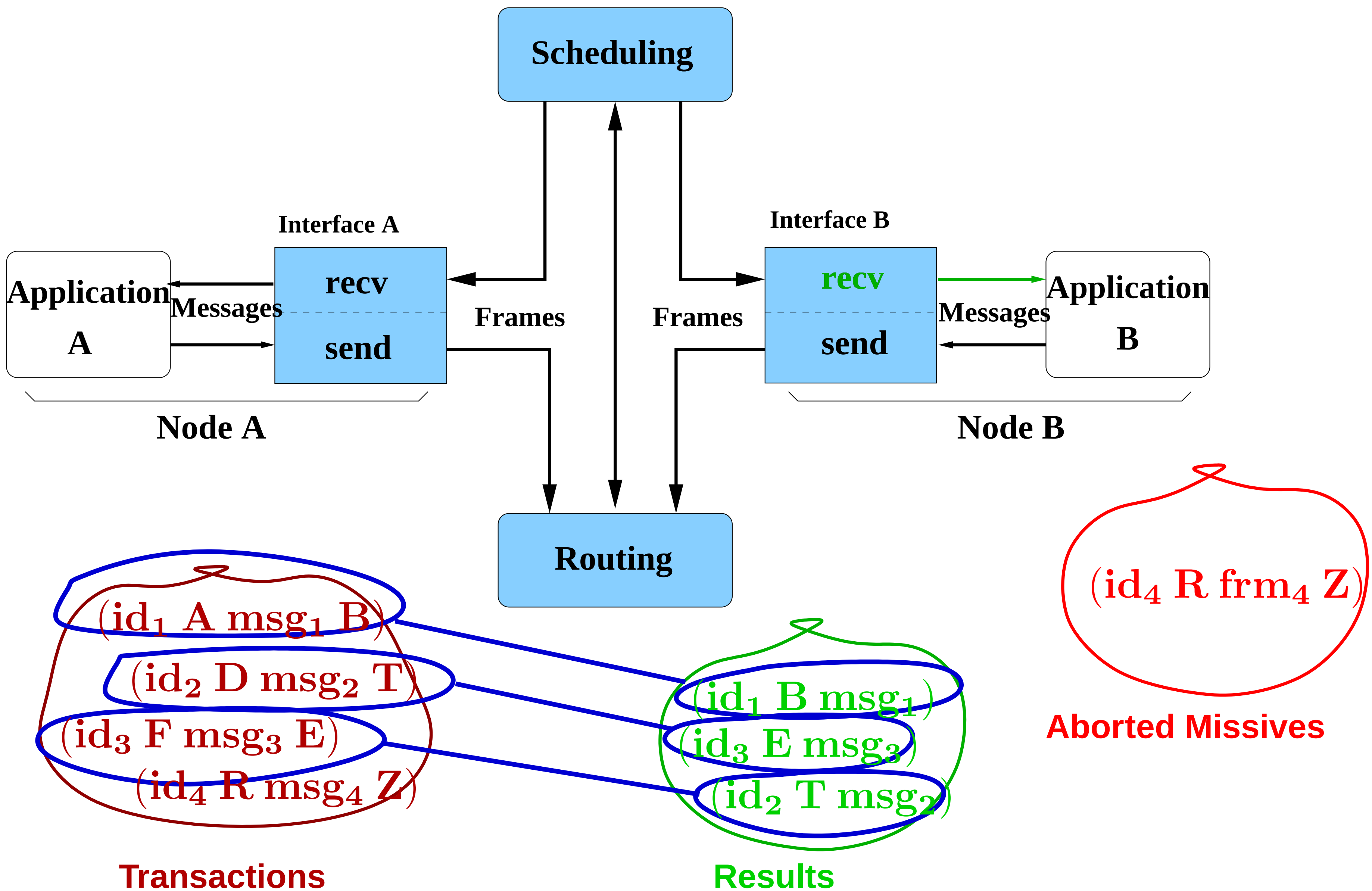
Aborted Missives



Aborted Missives



Correctness Criterion



Termination

Function $GeNoC$ is a recursive function and must be proved to terminate because:

- it is a prerequisite for mechanized reasoning (here ACL2)
- it is necessary to ensure liveness

To ensure the termination, we associate to every node a *finite* number of attempts. At every recursive call of $GeNoC$, every node with a pending transaction consumes one attempt.

Formal Definition

From a list of transactions, $\mathcal{T}$, the set of nodes $NodeSet$ and a list of attempt numbers att, function $GeNoC$ produces:

- The list $\mathcal{R}$ of results
- The list $\mathcal{A}$ for aborted missives

$$\begin{aligned} GeNoC: \;\; & \mathcal{D}_{\mathcal{T}} \times GenNodeSet \times AttLst \longrightarrow \mathcal{D}_{\mathcal{R}} \times \mathcal{D}_{\mathcal{M}} \\ & (\mathcal{T}, NodeSet, att) \mapsto (\mathcal{R}, \mathcal{A}) \end{aligned}$$

Correctness Criterion

$$\forall res \in \mathcal{R},$$

$$\exists! trans \in \mathcal{T}, \begin{cases} & Id_{\mathcal{R}}(res) = Id_{\mathcal{T}}(trans) \\ \wedge & Msg_{\mathcal{R}}(res) = Msg_{\mathcal{T}}(trans) \\ \wedge & Dest_{\mathcal{R}}(res) = Dest_{\mathcal{T}}(trans) \end{cases}$$

For any result res, there exists a unique transaction $trans$ such that $trans$ and res have the same identifier, message, and destination.

Correctness Criterion

$$\forall res \in \mathcal{R},$$

$$\exists! trans \in \mathcal{T}, \begin{cases} & Id_{\mathcal{R}}(res) = Id_{\mathcal{T}}(trans) \\ \wedge & Msg_{\mathcal{R}}(res) = Msg_{\mathcal{T}}(trans) \\ \wedge & Dest_{\mathcal{R}}(res) = Dest_{\mathcal{T}}(trans) \end{cases}$$

- Typical formula scheme
- Always check for Id equality
  - In ACL2, the idea is filtering according to Id's

Outline

- Communication Principles
- $GeNoC$ Definition and Correctness
- ACL2 Theorem/Removing Quantifiers
- Abstraction using Encapsulation
- Applications of $GeNoC$

ACL2 Correctness Predicate

```
(defun genoc-thm (𝓡 𝒯/𝓡_ids)
  (and (equal (R-msgs 𝓡)
              (T-msgs 𝒯/𝓡_ids))
       (equal (R-dests 𝓡)
              (T-dests 𝒯/𝓡_ids))))
```

- $\mathcal{T}/\mathcal{R}_{ids}$ = $\mathcal{T}$ filtered according to the ids of $\mathcal{R}$
- Check that the messages and the destinations of $\mathcal{T}/\mathcal{R}_{ids}$ and $\mathcal{R}$ are equal.

ACL2 Theorem

```
(defthm GeNoC-is-correct
 (mv-let
  (𝓡 𝓐)
  (GeNoC 𝓣 NodeSet att)
  (declare (ignore 𝓐))
  (implies (𝓣_lstp 𝓣)
           (GeNoC-thm
            𝓡
            (filters 𝓣
                     (R-ids 𝓡))))))
```

Proof Obligations

- Interfaces
  - The composition $recv \circ send$ is an identity
- Routing $(id\ A\ frm\ B) \mapsto (id\ frm\ Routes)$
  - Missive/Travel matching
    - Same frame and identifier
    - Routes effectively go from the correct origin to the correct destination
- Scheduling
  - Mutual exclusion between $Scheduled$ and $Delayed$
  - No addition of new identifiers
  - Preserve frames and route correctness

Proof of the theorem

- Routing correctness + preserved by scheduling
  - $\longrightarrow$ right destination
- No modification on frames
  - $\longrightarrow$ every result is obtained by $recv \circ send$
- Interfaces correctness
  - $\longrightarrow$ received message = sent message
- Mutual exclusion between $Scheduled$ and $Delayed$ + no new identifiers
  - $\longrightarrow$ cut the proof in two parts

Outline

- Communication Principles
- $GeNoC$ Definition and Correctness
- ACL2 Theorem/Removing Quantifiers
- Abstraction using Encapsulation
- Applications of $GeNoC$

Encapsulation: Interfaces

- Function $send$ builds a frame from a message:

$$((send \ *) \Rightarrow *)$$

- Function $recv$ recovers a message from a frame:

$$((recv \ *) \Rightarrow *)$$

- Their composition is an identity:

(**defthm** **InterfaceCorrectness**

;; $recv \circ send(msg) = msg$

(equal ($recv$ ($send$ msg)) msg))

- Some additional constraints

Interfaces Encapsulate Event

```
(encapsulate
    (((send *) => *)
     ((recv *) => *))
    ;; local witnesses
    (local (defun send (msg) msg))
    (local (defun recv (frm) frm))
    ;; proof obligations
    (defthm InterfaceCorrectness
     (equal (recv (send msg)) msg))
    (defthm send-nil
     (not (send nil)))
    (defthm send-not-nil
     (implies msg (send msg))))
```

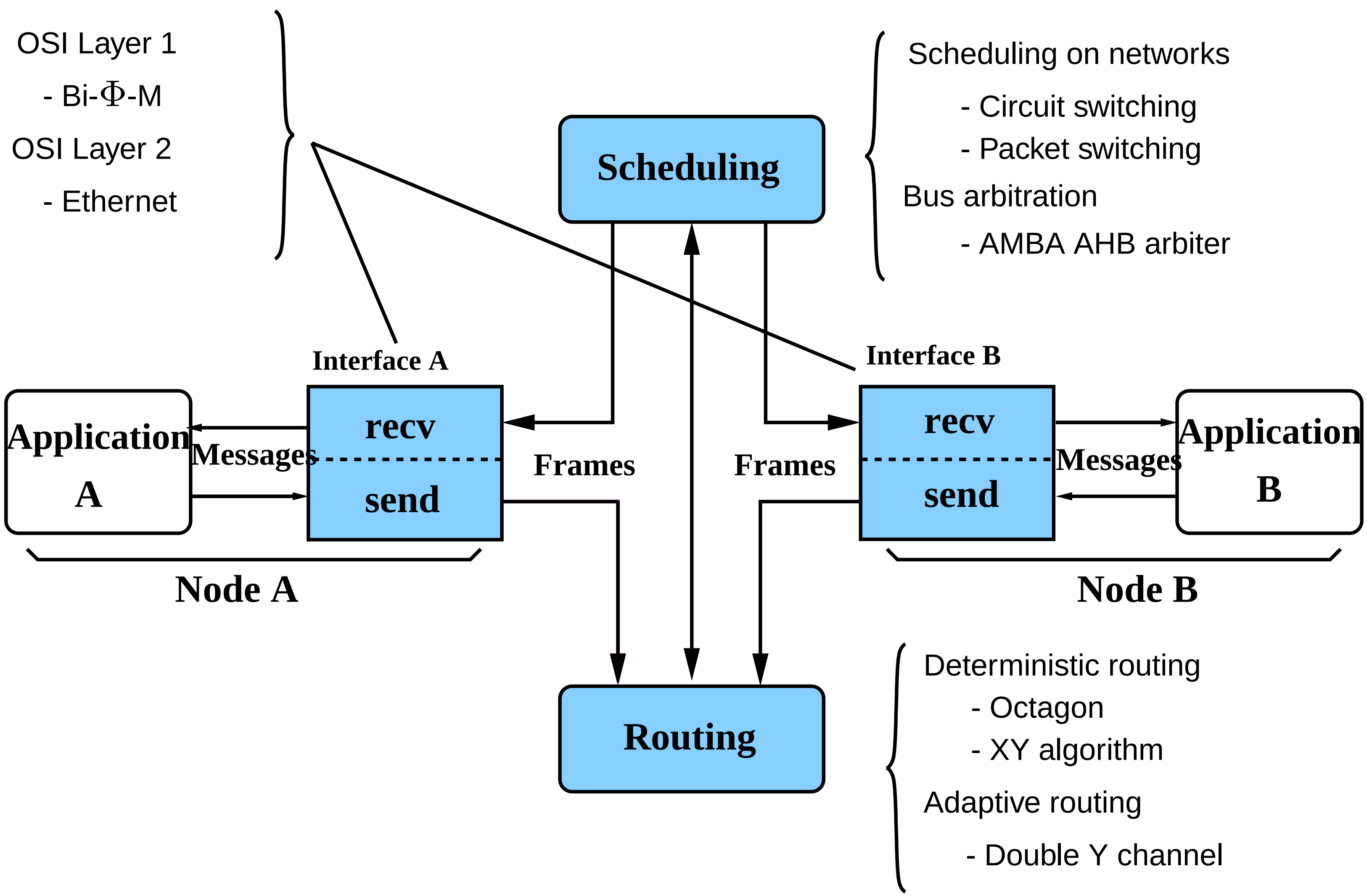
Checking Compliance

```
(defthm check-instance-interface
  t          ;; we prove true
  :rule-classes nil ;; no rule
  :hints (("GOAL"
  ;; we use InterfaceCorrectness
  ;; with recv_flexray for recv
  ;; and send_flexray for send
      :use
       (:functional-instance
           InterfaceCorrectness
           (recv  recv_flexray)
           (send  send_flexray)))))
```

Outline

- Communication Principles
- $GeNoC$ Definition and Correctness
- ACL2 Theorem/Removing Quantifiers
- Abstraction using Encapsulation
- Applications of $GeNoC$

Applications of $GeNoC$



Conclusion

- A generic model: $GeNoC$
  - Identifies the essential constituents and their properties
  - Formalizes the global property as a consequence of proof obligations
- Its expression in ACL2
  - 1864 lines, 71 functions and 119 theorems
  - One fourth is dedicated to the modules
  - Abstraction using encapsulation
  - Automatic generation of proof obligations using *functional instantiation*

Future Work

- Master/Slave protocols
- Deadlocks (structural and protocol level)
- Adding queues and channels
  - wormhole routing in Hermes (TIMA, Grenoble, France)
- Verified Distributed Stacks
  - “Verisoft” Stack (O.S., compiler, assembly, gates)
  - Interconnected Stacks through a time triggered FlexRay bus
    - Show that FlexRay matches $GeNoC$!
- . . .

THANK YOU !!