The openais Architecture

An inside look at an implementation of SA Forum's AIS

Steven Dake
Presented by Tim Anderson

History of openais

- Started life as “cmgr” in February 2002
  - Hotswap manager for ATCA
- Converted to AIS in May 2003
- AIS released as MontaVista product in Dec 2003
- Rearchitected to use virtual synchrony Jan 2004
- Released under Revised BSD license July 2004
- Mark Haverkamp contributed EVT service Aug/Sept 2004

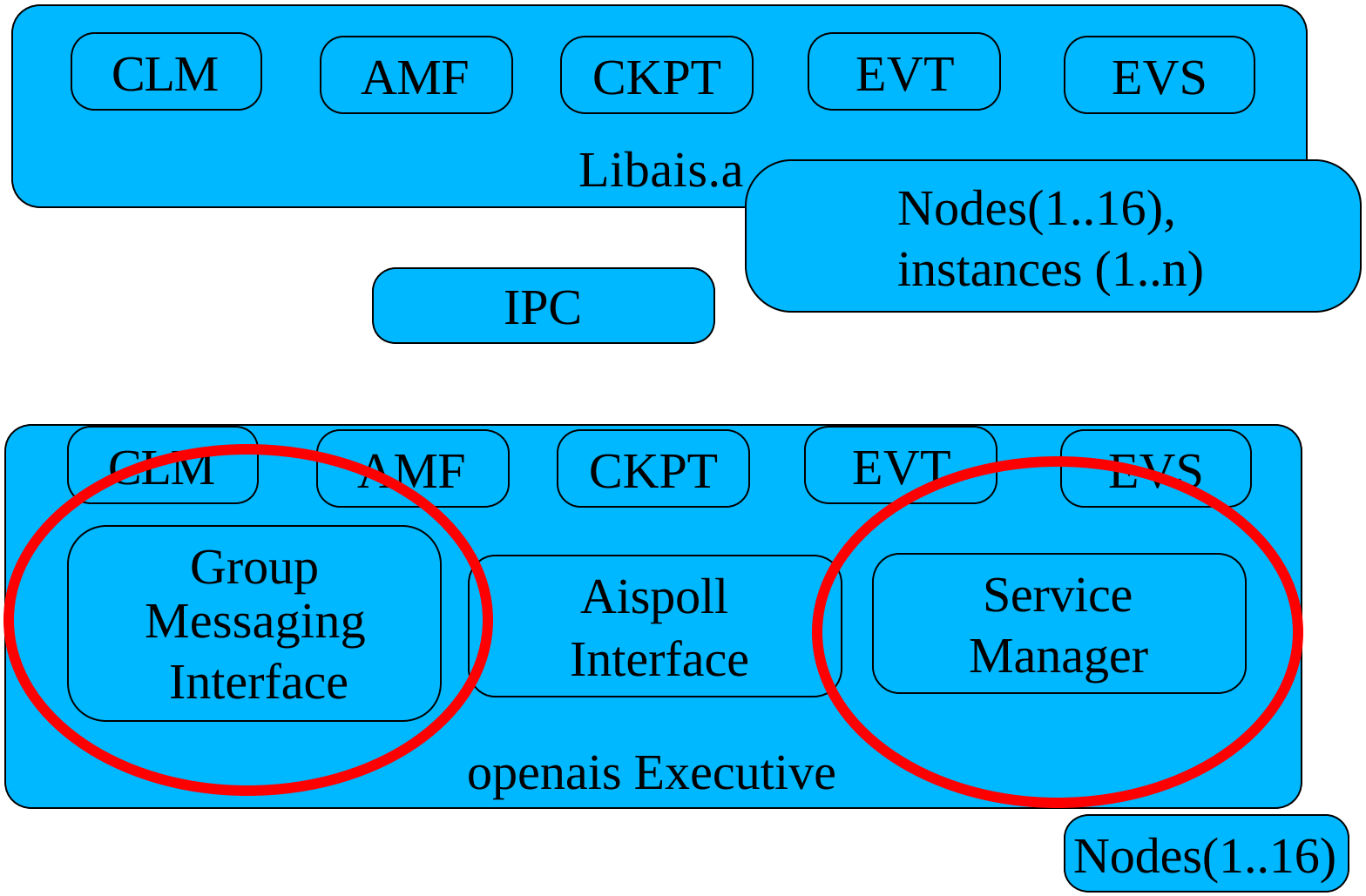
Setup and Configuration

Create shared key:
linux# ./keygen
OpenAIS Authentication key generator.
Gathering 1024 bits for key from /dev/random.
Writing openais key to /etc/ais/authkey.

Save /etc/ais/network.conf:
bindnetaddr: 192.168.1.0
mcastaddr: 226.94.1.1
mcastport:6000

Read QUICKSTART in source package for more details

The Architecture



Definitions

- Group Messaging
  - Sending messages from 1 sender to many receivers.
- Processor
  - The entity responsible for executing group messaging and membership protocols.
- Configuration
  - A view, or description, of the processors within a group.
- Agreed Order
  - All processors agree upon delivery order of messages delivered using group messaging.
- Virtual Synchrony
  - A model of group messaging whereby all messages within a configuration view are delivered in agreed order. Configuration changes are delivered in the same order relative to messages to every processor.

Group Messaging Interface

- Implements Extended Virtual Synchrony
- Compile-time configuration of maximum message size
- Encryption and Authentication of all messages
- 4 Priority Levels
- Uses multicast
- Implemented using UDP
- Multipathing in progress

The Ring Protocol

- Sequence Number
- Retransmit List
- flow control count
- group arut

ORF Token

Processor #1

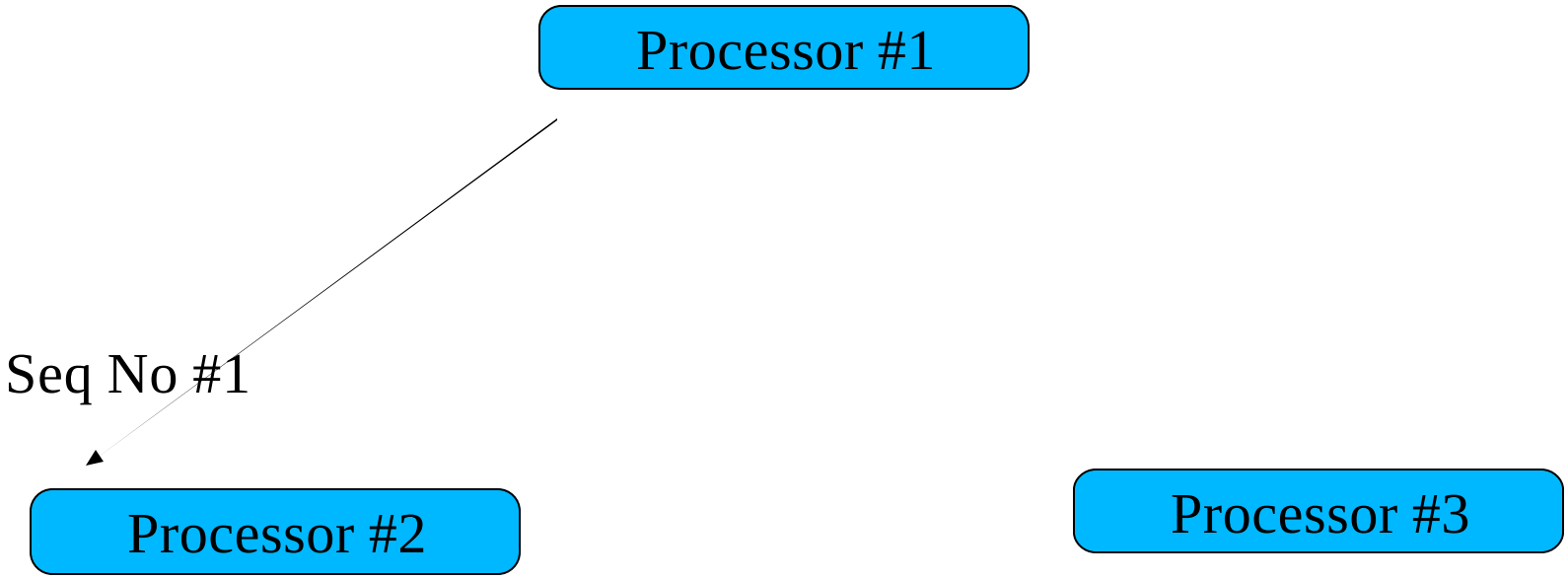
Processor #2

Processor #3

The Ring Protocol

- Sequence Number
- Retransmit List
- flow control count
- group arut

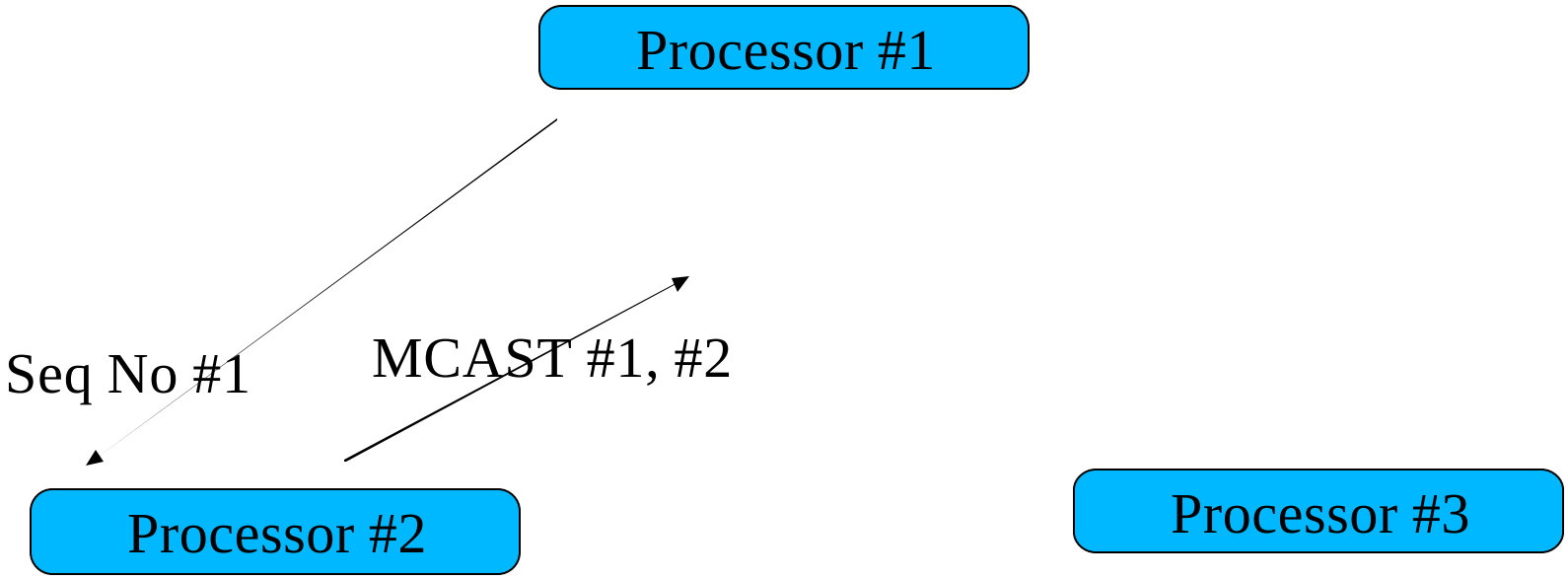
ORF Token



The Ring Protocol

- Sequence Number
- Retransmit List
- flow control count
- group arut

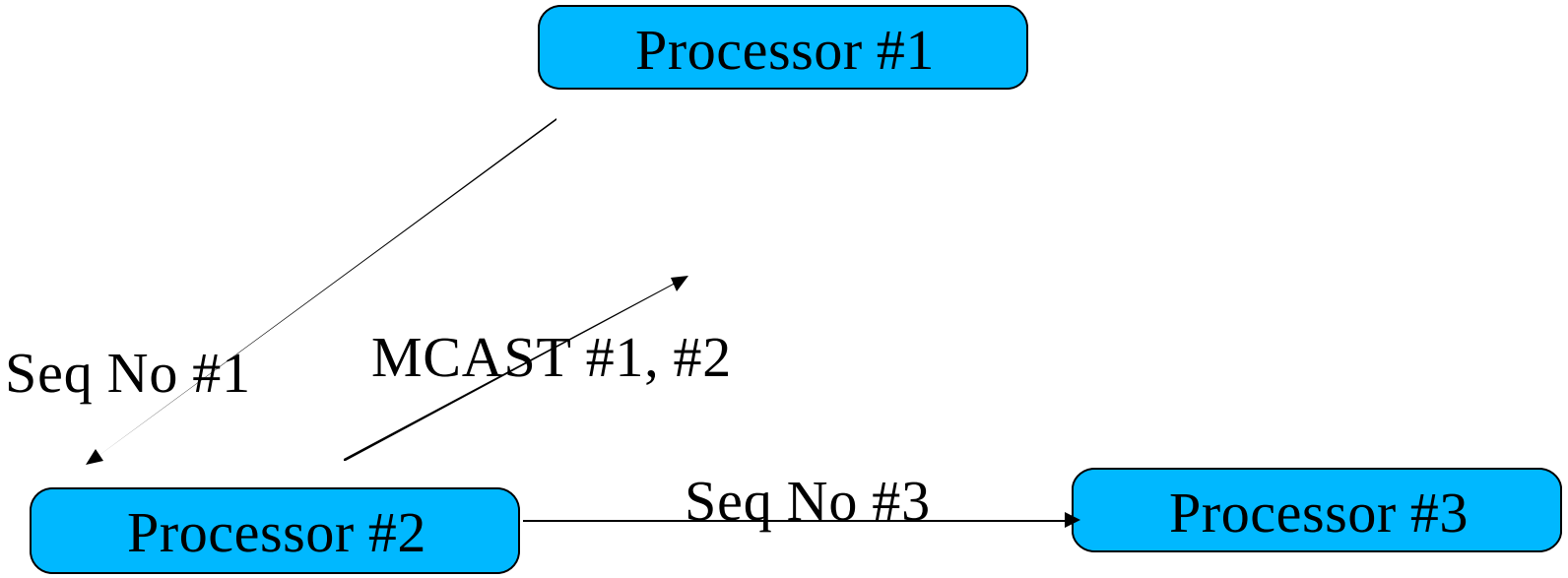
ORF Token



The Ring Protocol

- Sequence Number
- Retransmit List
- flow control count
- group arut

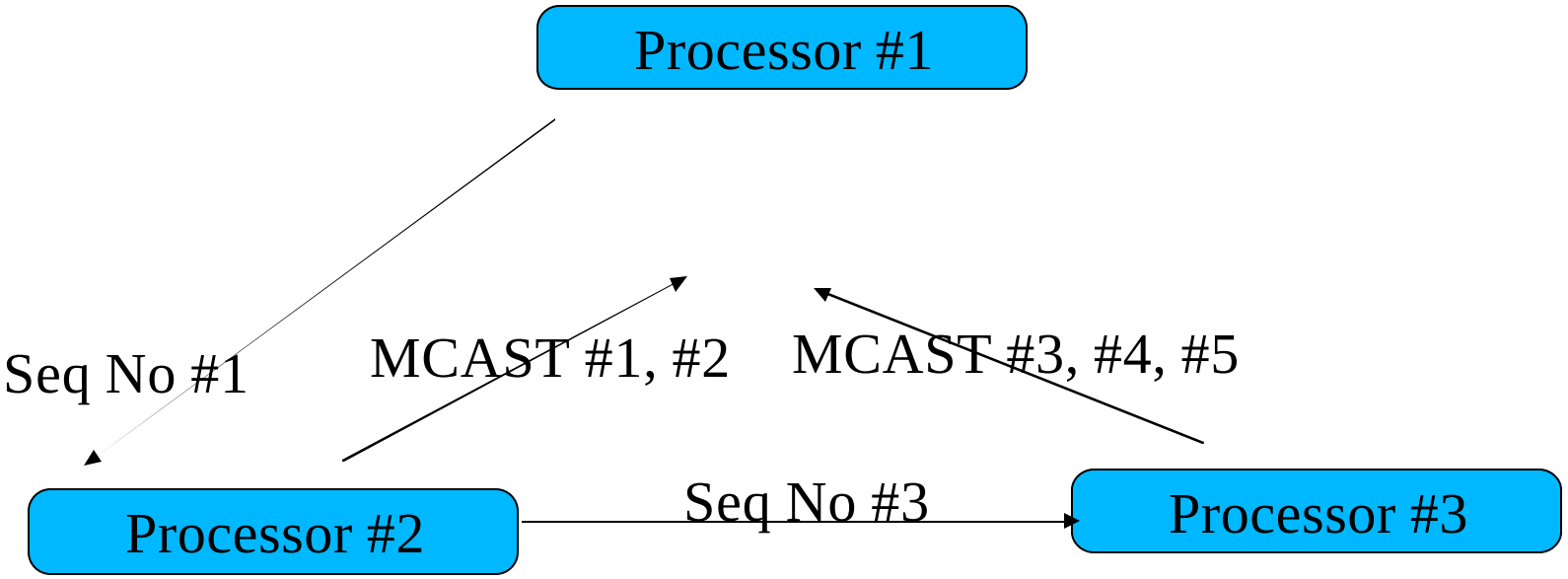
ORF Token



The Ring Protocol

- Sequence Number
- Retransmit List
- flow control count
- group arut

ORF Token

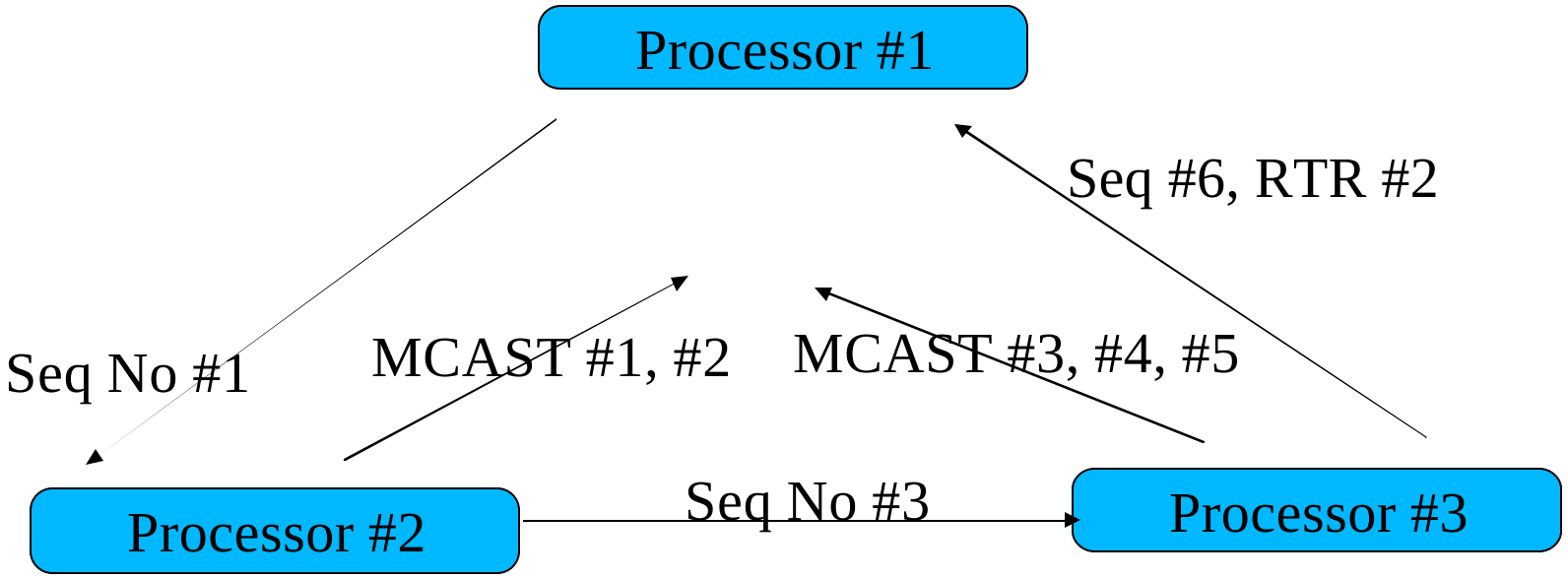


Detects Missing #2

The Ring Protocol

- Sequence Number
- Retransmit List (RTR)
- flow control count
- group arut

ORF Token

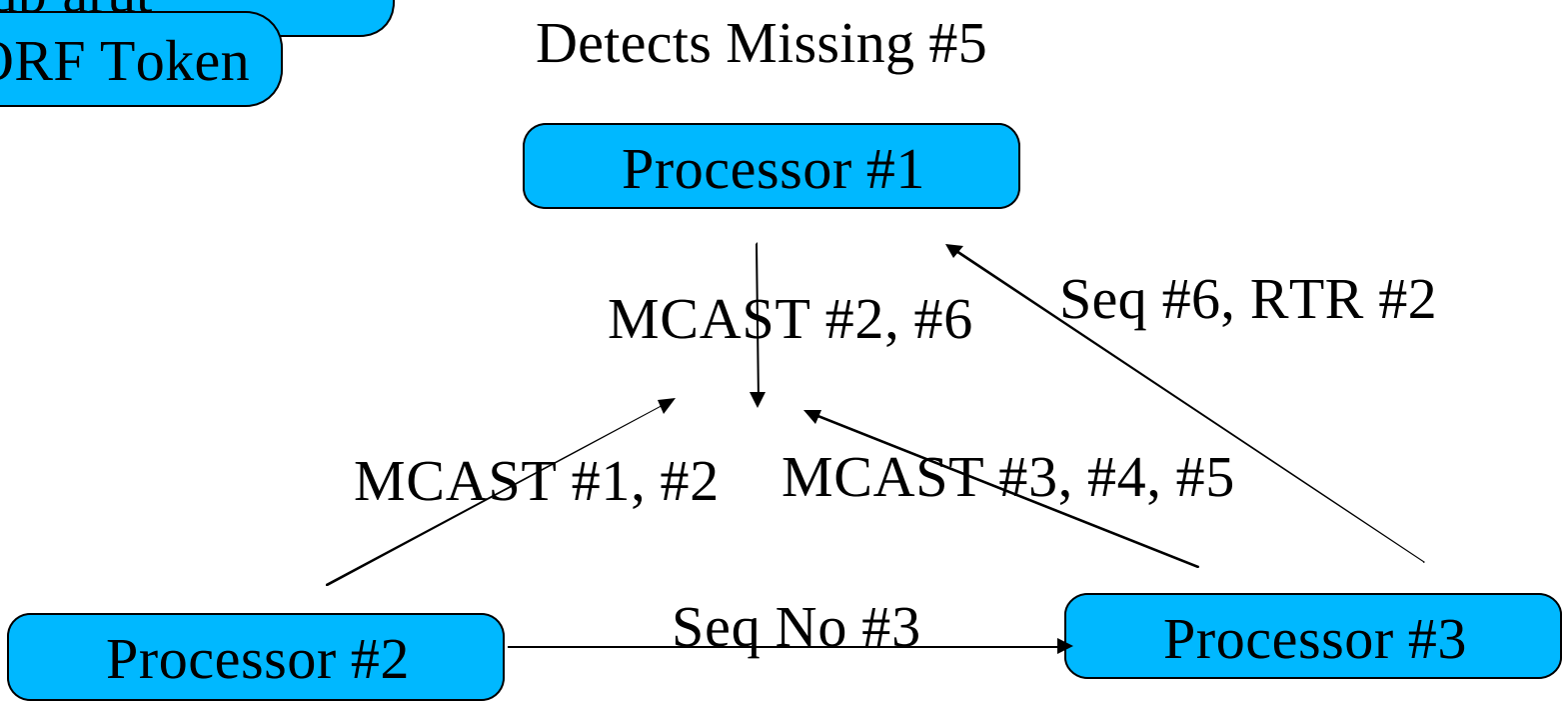


Detects Missing #2

The Ring Protocol

- Sequence Number
- Retransmit List (RTR)
- flow control count
- group arut

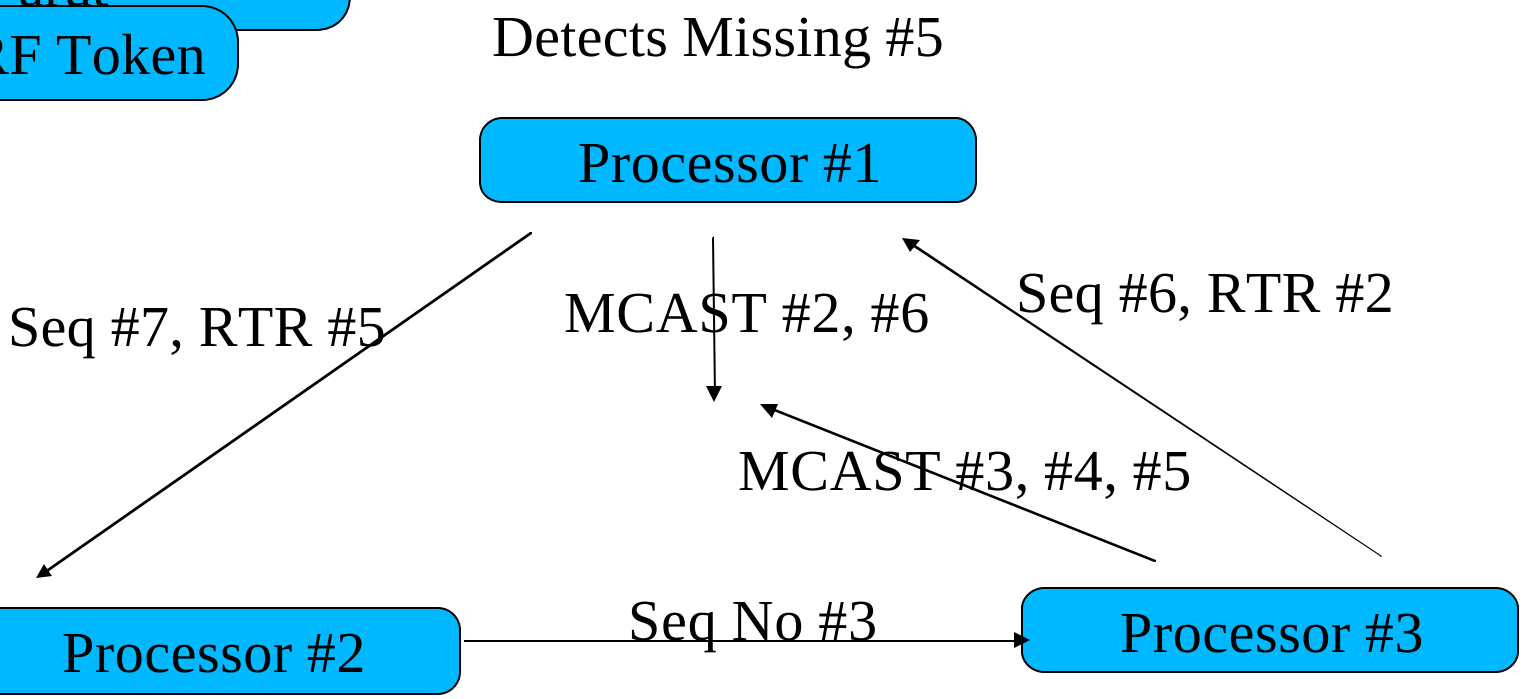
ORF Token



The Ring Protocol

- Sequence Number
- Retransmit List (RTR)
- flow control count
- group arut

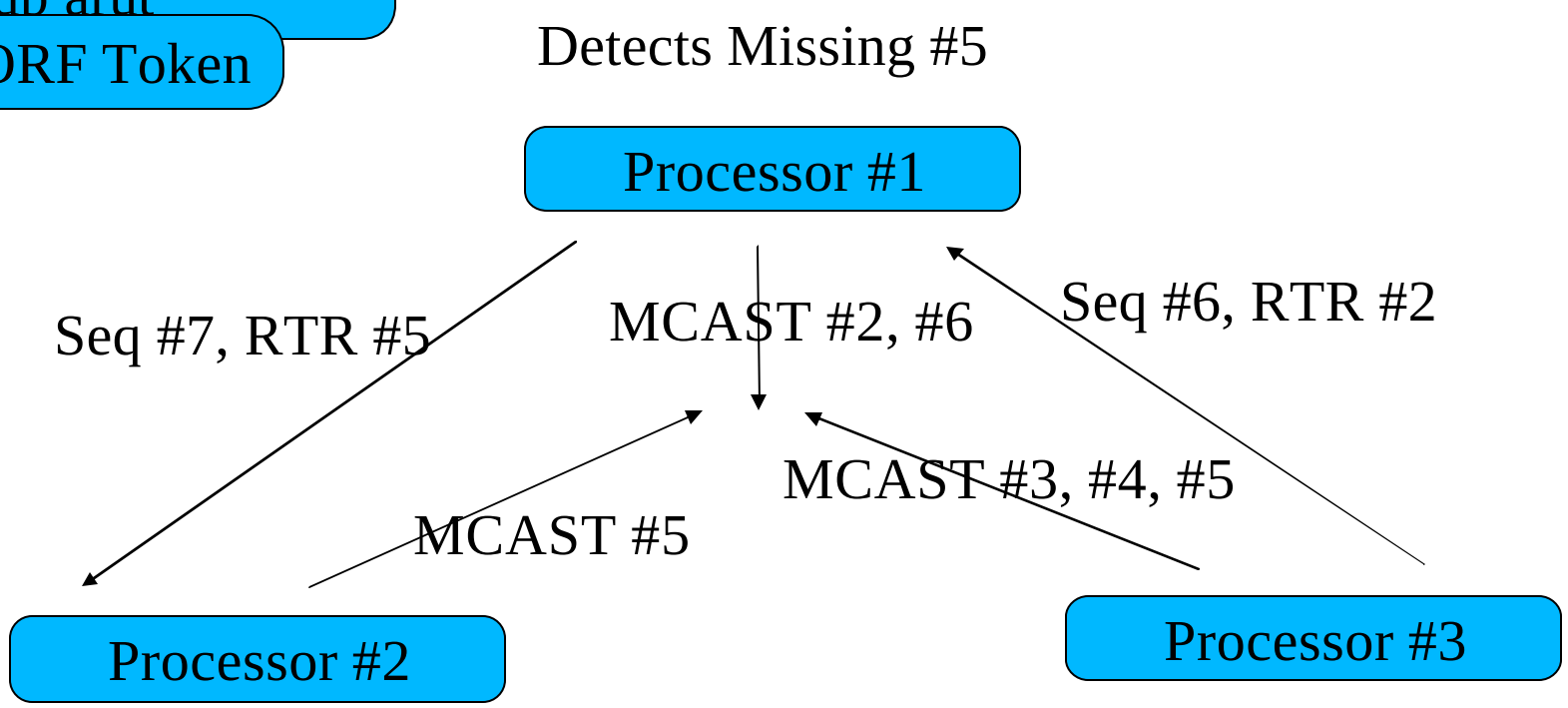
ORF Token



The Ring Protocol

- Sequence Number
- Retransmit List (RTR)
- flow control count
- group arut

ORF Token



Why Virtual Synchrony

- Integrated Membership
- Strong Membership Guarantees
- Agreed Ordering of Messages
- Self Delivery
- Use of multicast
- Group Wide Flow Control
- Performance

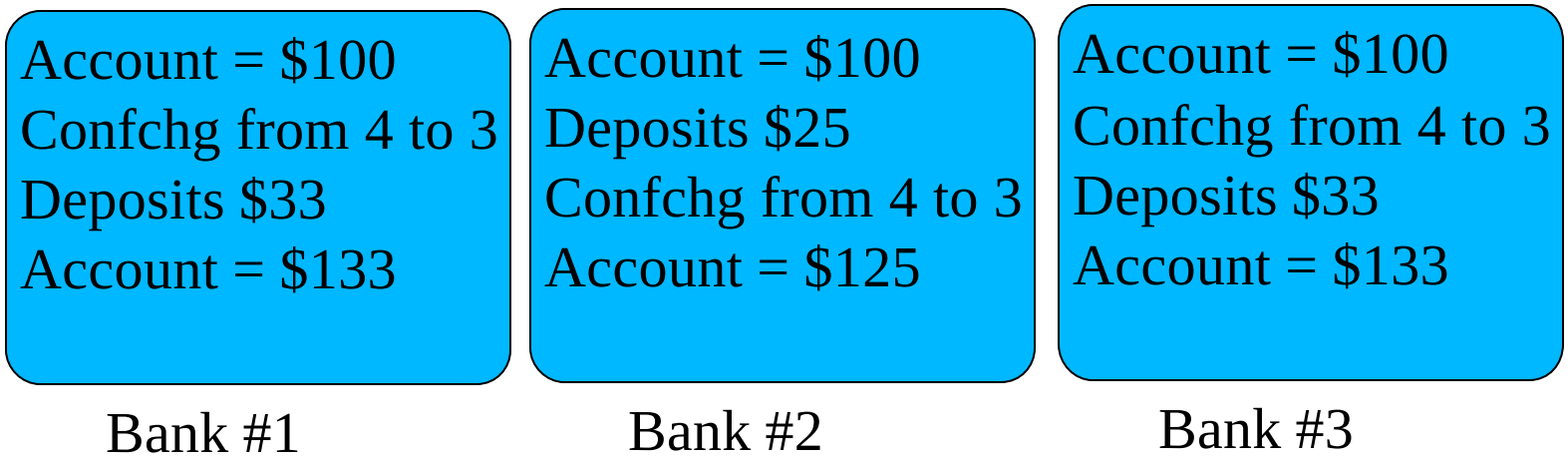
Why Virtual Synchrony – Integrated Membership

```
gmi_join (struct gmi_groupname *groupname,
        void (*deliver_fn) (
        struct gmi_groupname *groupname,
                struct in_addr source_addr,
                struct iovec *iovec,
                int iov_len),
        void (*confchg_fn) (
                struct sockaddr_in *member_list, int member_list_entries,
                struct sockaddr_in *left_list, int left_list_entries,
                struct sockaddr_in *joined_list, int joined_list_entries),
        gmi_join_handle *handle_out);
```

Messages delivered with deliver_fn,
configuration changes delivered with confchg_fn

Why Virtual Synchrony – Strong Membership Guarantees

Example: A bank stores money in a distributed fashion based how many banks are in its "network". The account starts with $300 and $99 is deposited. One bank closed forever around the time of the deposit. What could happen without strong membership guarantees?

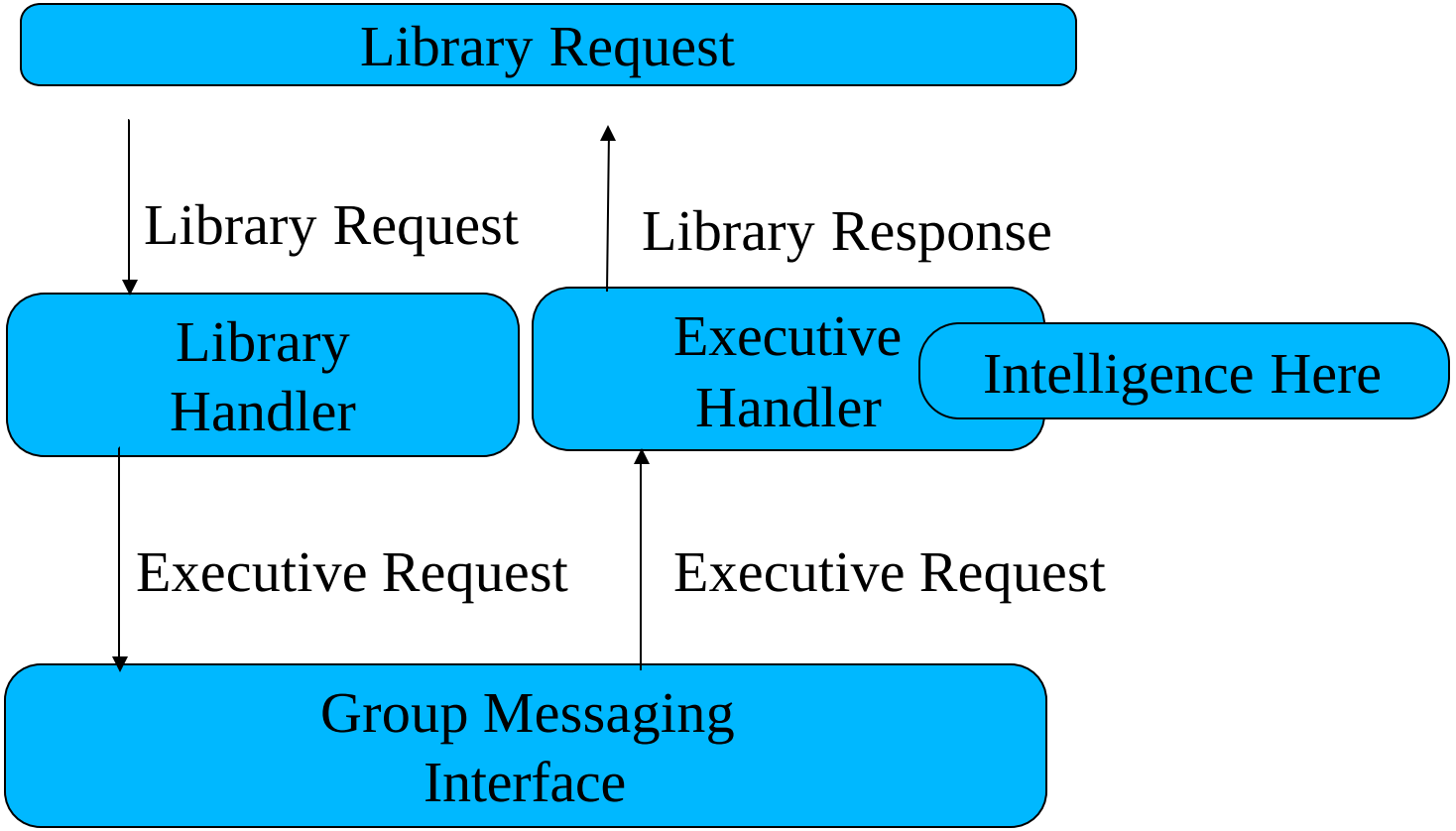


Why Virtual Synchrony – Agreed Ordering of Messages

Object Creation occurs on two seperate processors with the same “name”. What happens without agreed ordering of messages? Race conditions!

Processor #1	Processor #2
Created Object “A” Send A to other processors Receives create, but already exists	Created Object “A” Send A to other processors Receives create, but already exists

Why Virtual Syncrhony – self delivery



Why Virtual Syncrhony – Use of Multicast

```
int gmi_mcast (
        struct gmi_groupname *groupname,
        struct iovec *iovec,
        int iov_len,
        int priority);
```

The Service Handler

- Service Manager Manages service handlers.
- Every Service has 1 or more service handlers.
- Handles requests from library connections.
- Handles requests from group messaging delivery.
- Handles partitions and merges.
- Initializes the service for a new library connection
- Exits the service for a departing library connection.
- Initializes the service for the first time.

The Service Handler – Details

```
struct service_handler {
        struct libais_handler *libais_handlers;
        int libais_handlers_count;
        int (**aisexec_handler_fns) (void *msg, struct in_addr source_addr);
        int aisexec_handler_fns_count;
        int (*confchg_fn) (
                struct sockaddr_in *member_list, int member_list_entries,
                struct sockaddr_in *left_list, int left_list_entries,
                struct sockaddr_in *joined_list, int joined_list_entries);
        int (*libais_init_fn) (struct conn_info *conn_info, void *msg);
        int (*libais_exit_fn) (struct conn_info *conn_info);
        int *aisexec_init_fn) (void);
}
```

Flow Control

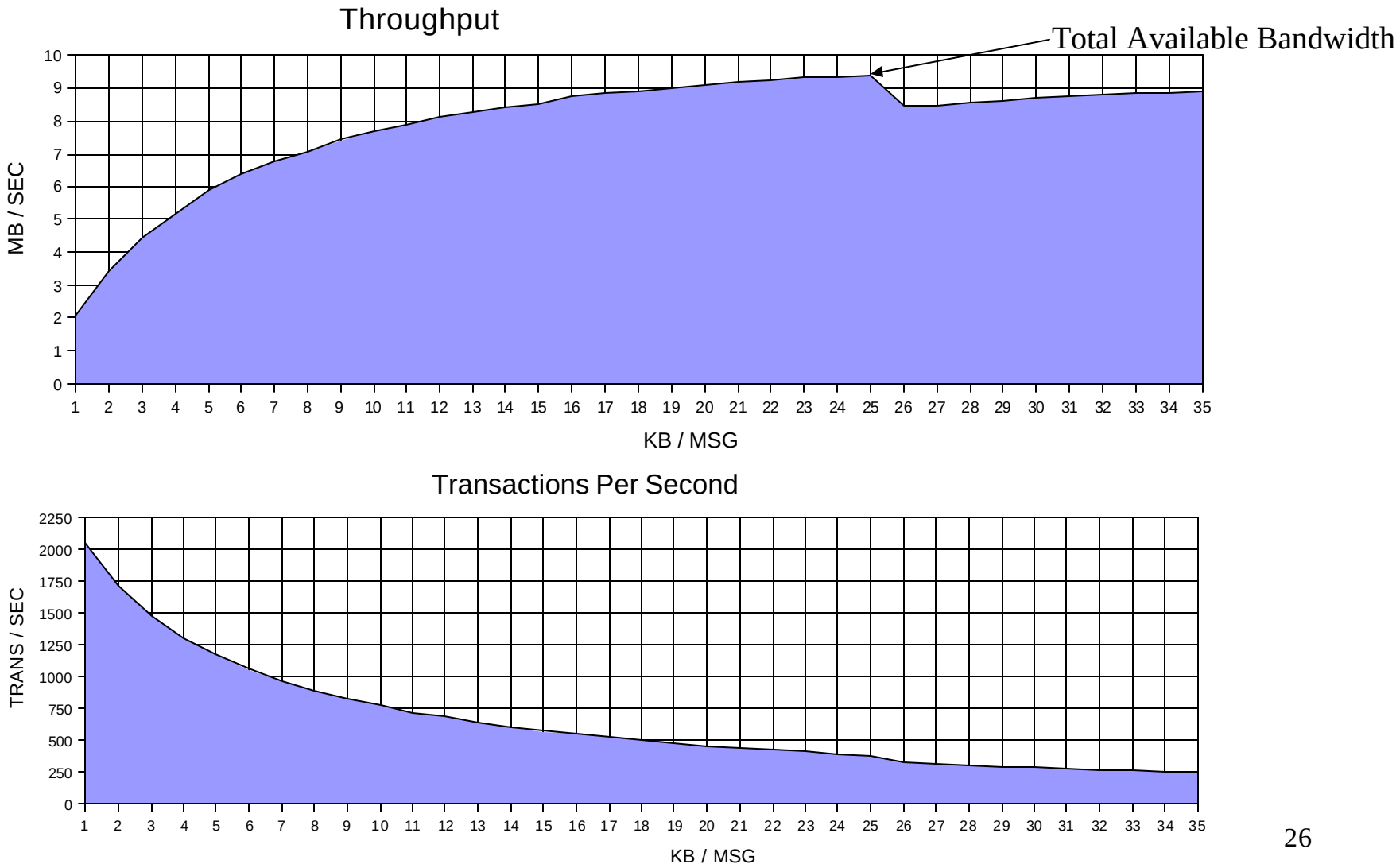
- Group Messaging Interface uses flow control on network
- Library can access executive much faster then network can transmit requests
- Library is flow controlled by group messaging interface.

Flow Control – Details

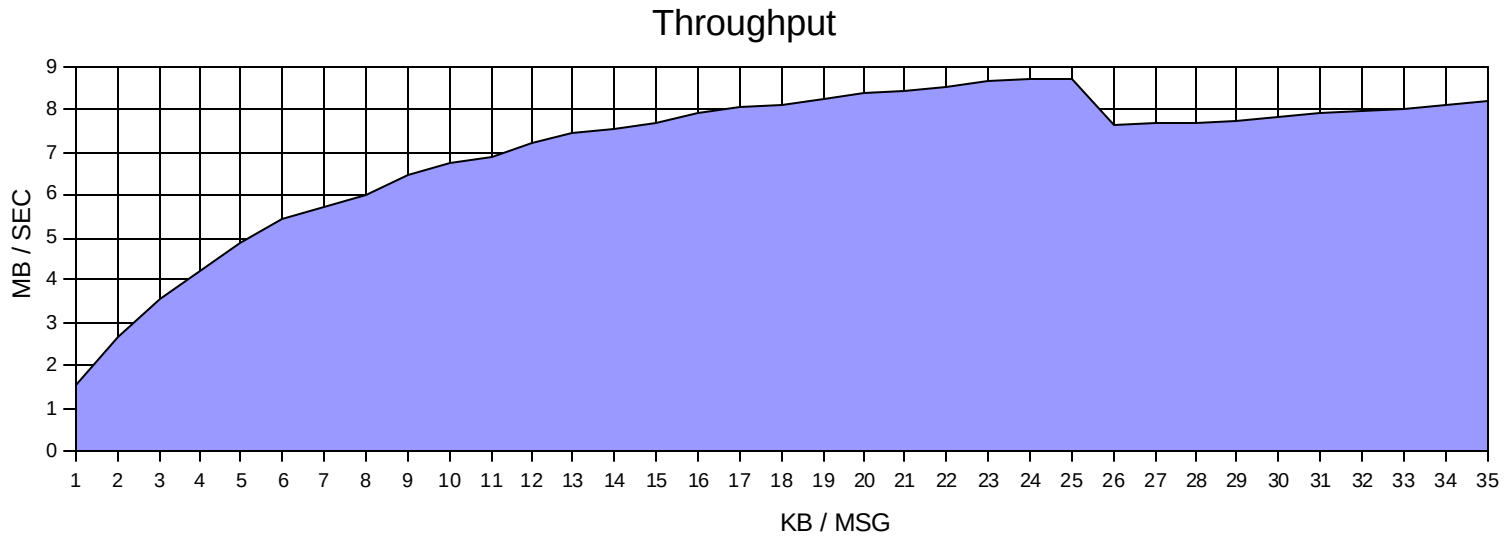
```
int gmi_send_ok (
        int priority,
        int msg_size);
```

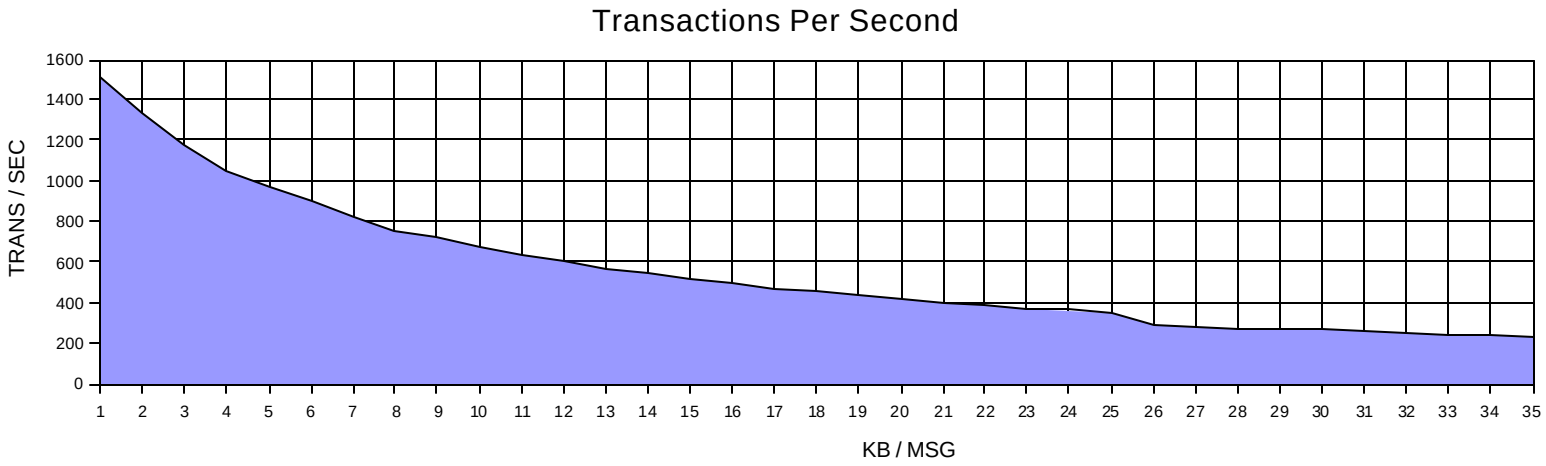
If gmi_send_ok is zero, library request receives SA_ERR_TRY_AGAIN. This support is handled by the service manager. The library handler is not concerned with flow control.

Performance / No Encryption or Auth Checkpoint Write from One Processor (100 mbit network)

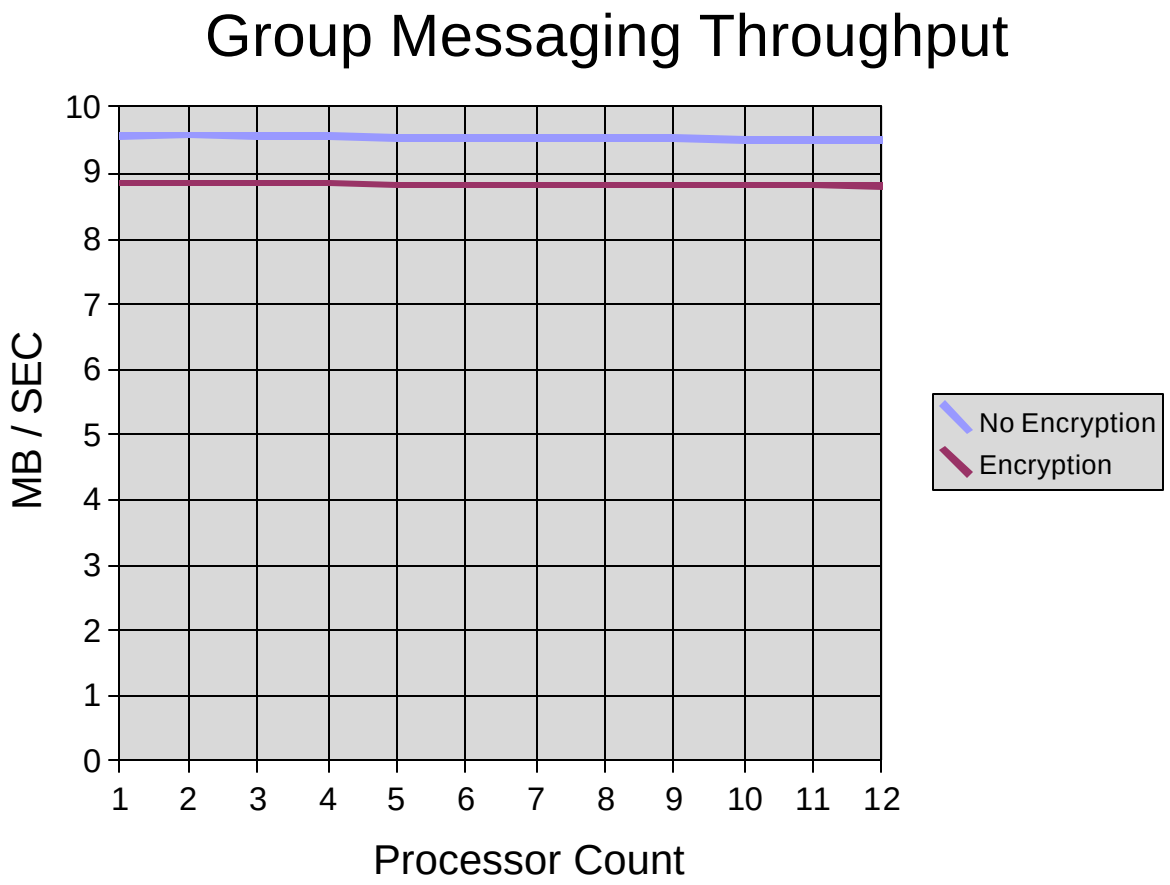


Performance / With Encryption and Auth
Checkpoint Write from One Processor
(100 mbit network)





Performance / Group Messaging Scalability with more Processors



Project Statistics

- Executive LOC: 16229
- Library LOC: 5951
- Include LOC: 2819
- Total LOC: 24999 (wc -l)
- BK Changesets since openais inception: 65

Production Release Criteria

- 0.7 (stable) to be released in 2004
- Includes AMF, CKPT, EVT, CLM, EVS
- At least 85% code coverage of every source file except for files in test directory
- Published valgrind analysis of any reported memory errors or leaks
- Code review of remaining uncovered code
- Initially support linux 2.4, linux 2.6 systems

Come Join In

We need developers to develop DLOCK and MSG services.
We need developers to develop Linux man pages.
We need developers to develop distro packaging.
We need user reports of failures and successes.

Web Address: http://developer.osdl.org/dev/openais

Mailing List: openais@lists.osdl.org

Download: http://developer.osdl.org/cherry/openais

Bitkeeper: bk clone bk://bk.osdl.org:openais ~/openais