Inside Closed Process Groups

An OpenAIS Service

openais

Standards-Based Cluster Framework

Steven Dake

April 2007

www.openais.org

Agenda

- Virtual Synchrony
- Totem
- Example Application of Totem
- Closed Process Groups
- The Closed Process Group Interface

Definitions

- Group Messaging
  - Sending messages from 1 sender to many receivers.
- Processor
  - The entity responsible for executing group messaging and membership protocols.
- Regular Configuration Change
  - An event containing a unique view identifier and list of processors contained within the configuration. Denoted as RCn in examples.
- Transitional Configuration
  - An event containing a unique view identifier and list of processors transitioning from the old regular configuration to the new regular configuration. Denoted as TCn in examples.

Virtual Synchrony Property #1 – self delivery

- self delivery – A message sent by a processor is delivered to that processor.

  Example:

  Processor P1 sends message M1

  M1 is self-delivered to P1

Virtual Synchrony Property #2 – AGREED ordering

- agreed ordering – all processors agree upon delivery order of messages.

Example:
P1: M1 M2 M3 M4
P2: M1 M2 M3 M4
P3: M1 M2 M3 M4

Can't happen:
P1: M1 M2 M3 M4
P2: M1 M2 M4 M3
P3: M1 M2 M3 M4

Virtual Synchrony Property #3 – SAFE ordering

- SAFE ordering – agreed ordering extended such that a message may not be delivered until every processor within the configuration has a copy.
  - P1 sends M1 M2 M3.
  - P2 and P3 only receive M1 and M2.
  - P1 P2 P3 deliver M1 M2.
  - P2 and P3 recover M3.
  - M3 may now be delivered in safe order.

Virtual Synchrony Property #4 – Virtual Synchrony

- virtual synchrony – messages are delivered in agreed order and configuration changes are delivered in agreed order relative to messages.

  Example:

  P1: M1 M2 M3 M4 CT1 CR1

  P2: M1 M2 M3 M4 CT1 CR1

  Can't happen:

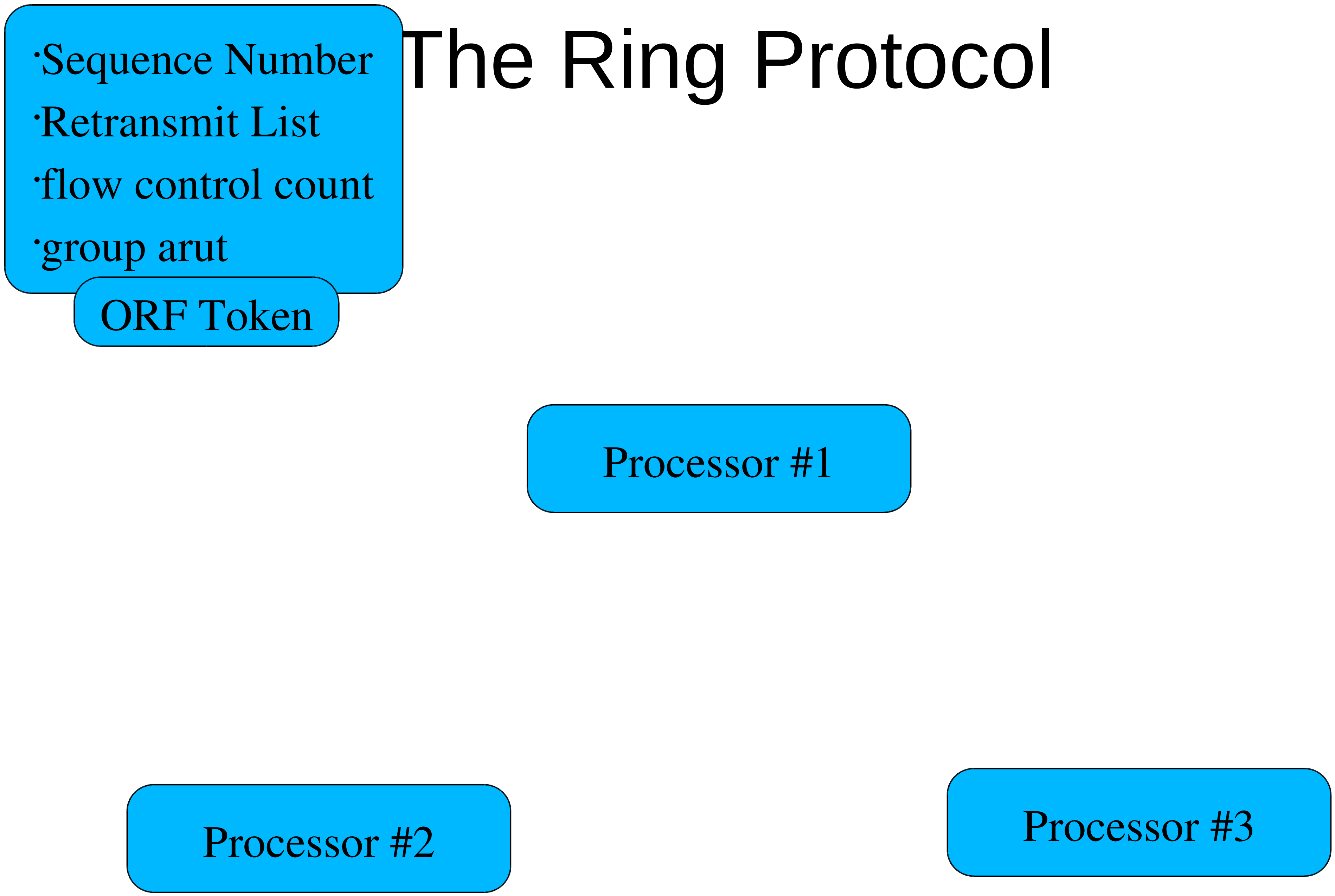
  P1: M1 M2 M3 M4 CT1 CR1

  P2: M1 M2 M3 CT1 CR1

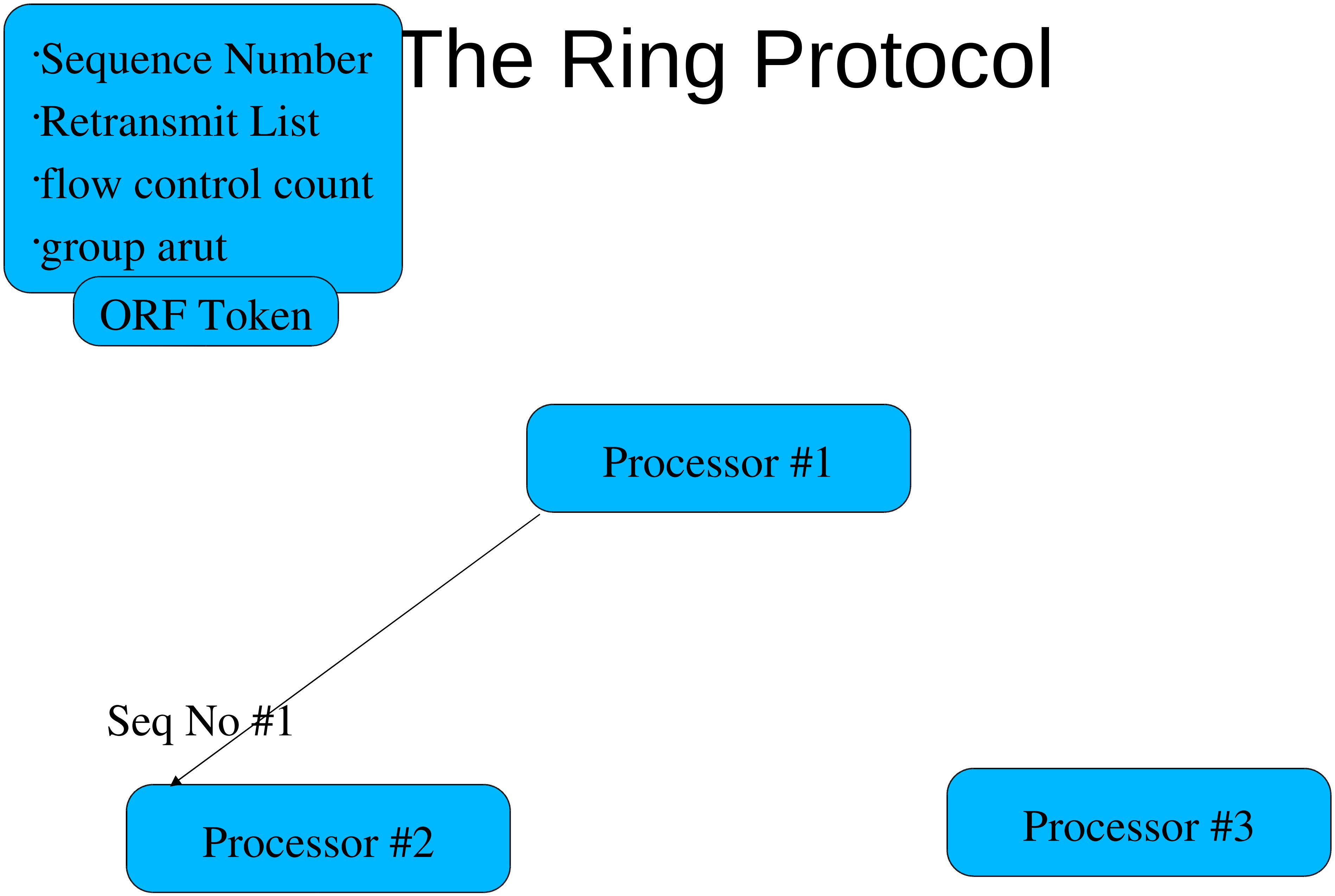
Totem – The Single Ring Protocol

- Encryption and Authentication of all messages
- Support for redundant network devices via RRP
- Support for jumbo frame sizes and full fragmentation of all messages to MTU
- Full marshalling of all communications protocols
- Designed with future multiple ring architecture in mind
- Usable as a library or via other interfaces such as CPG (closed process groups)

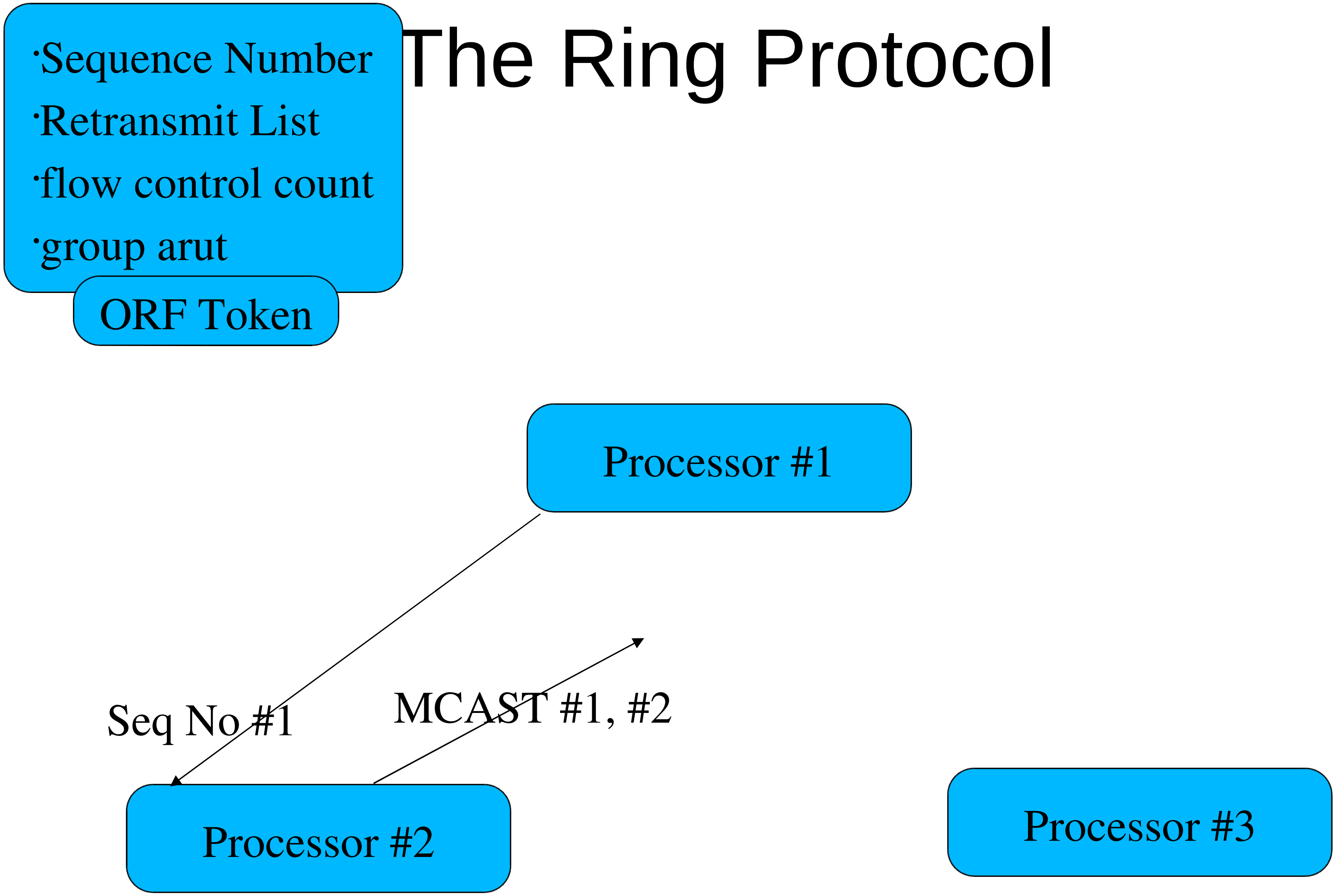
The Ring Protocol



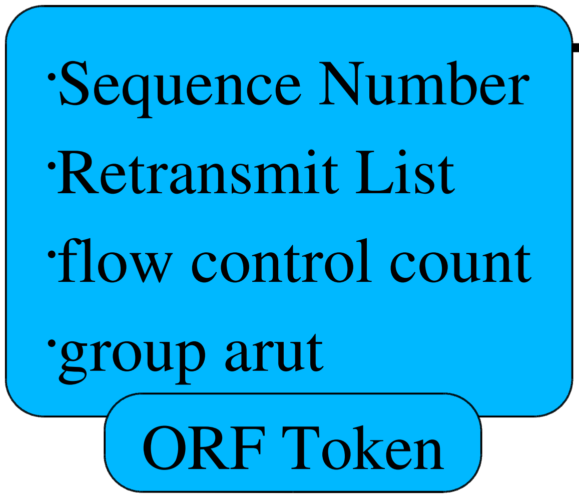
The Ring Protocol

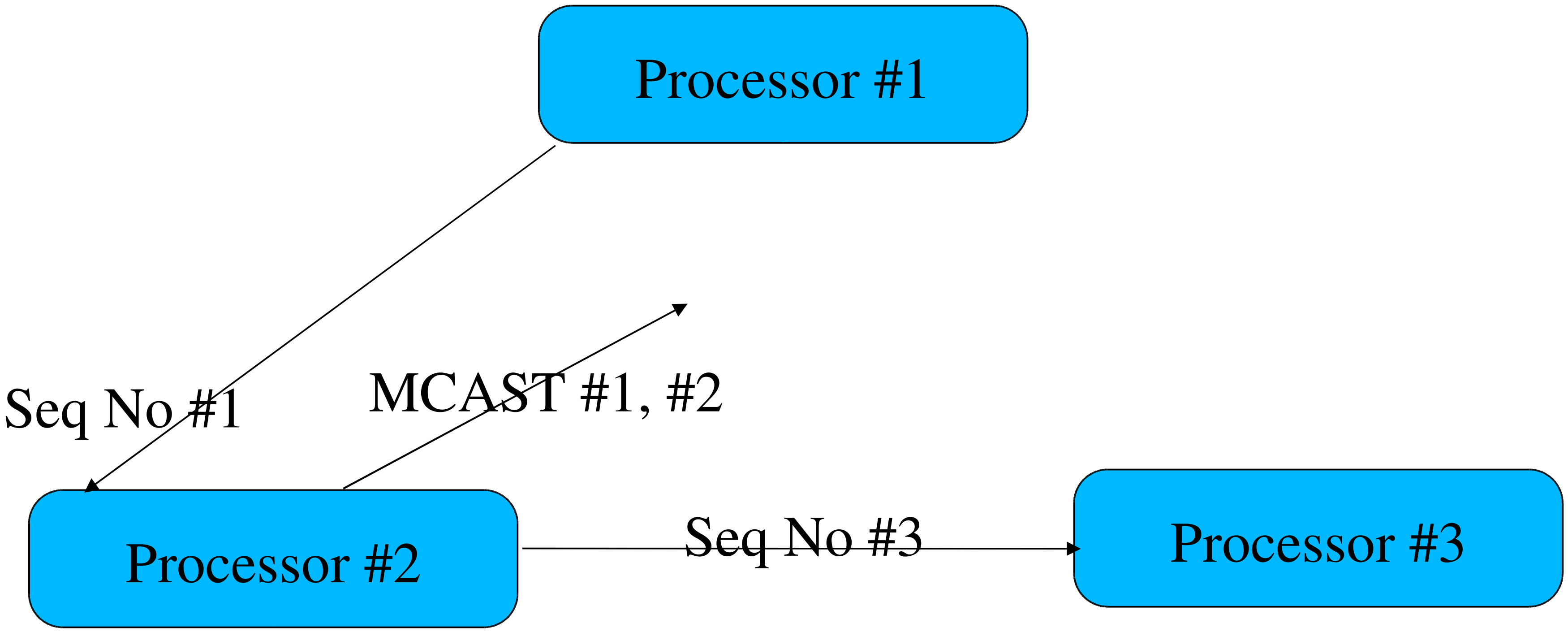


The Ring Protocol



The Ring Protocol

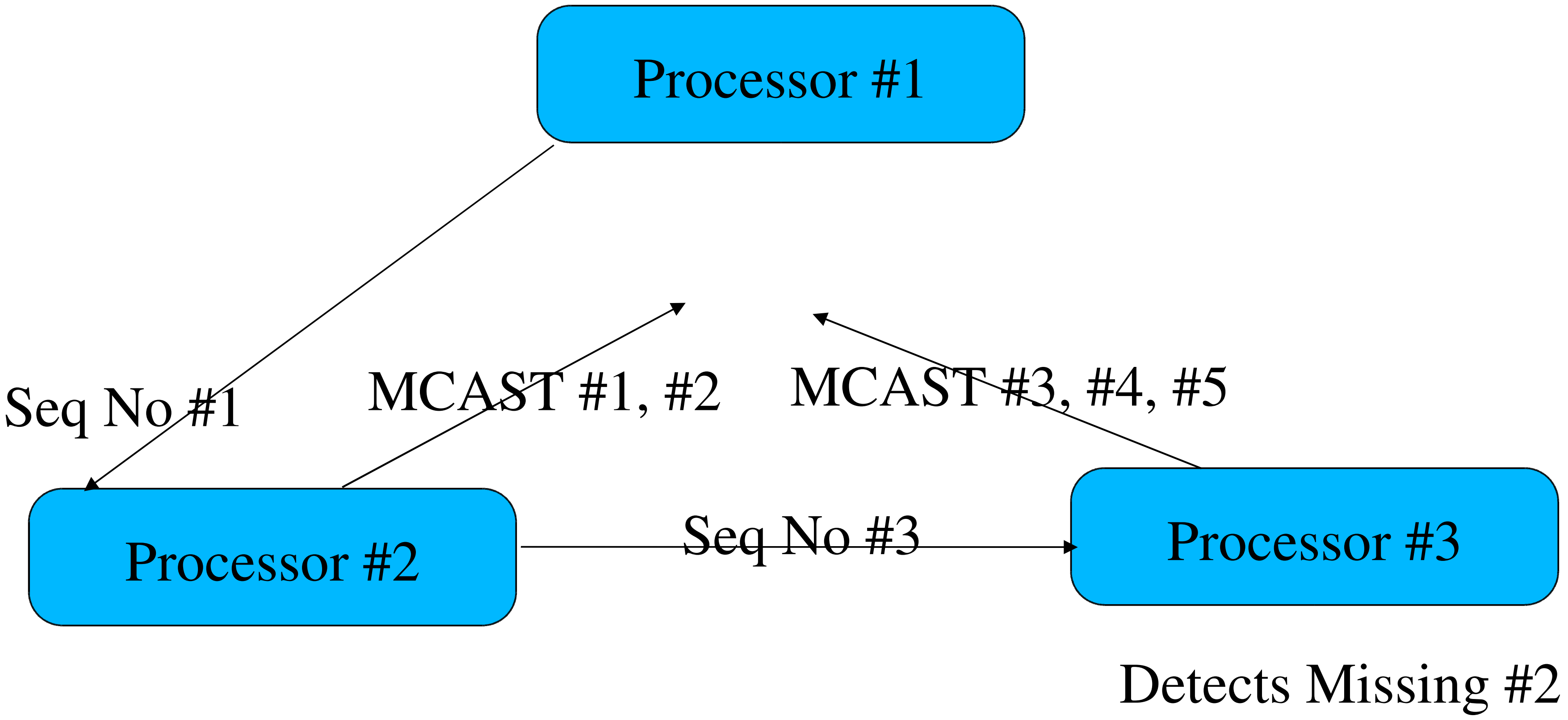




The Ring Protocol

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

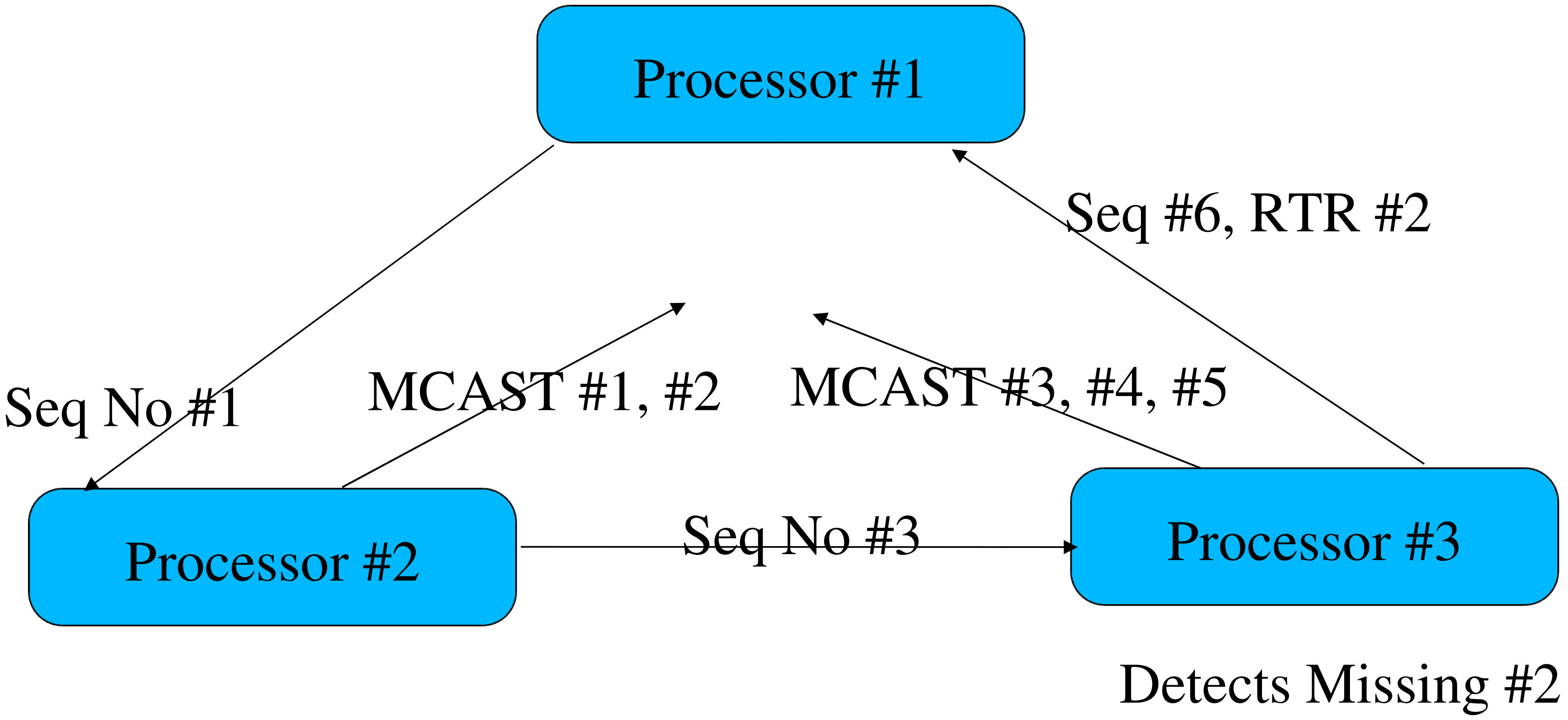
ORF Token



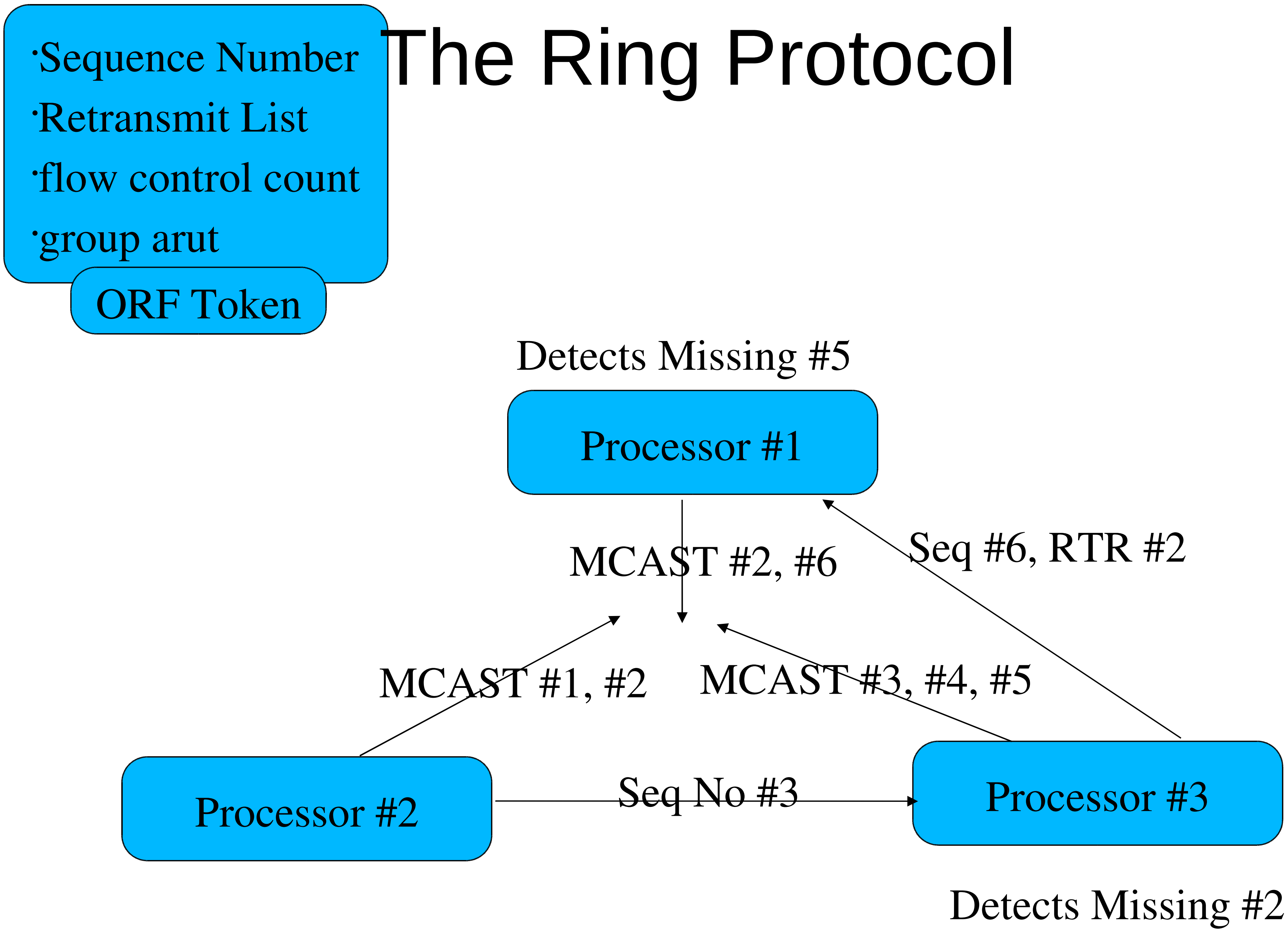
The Ring Protocol

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

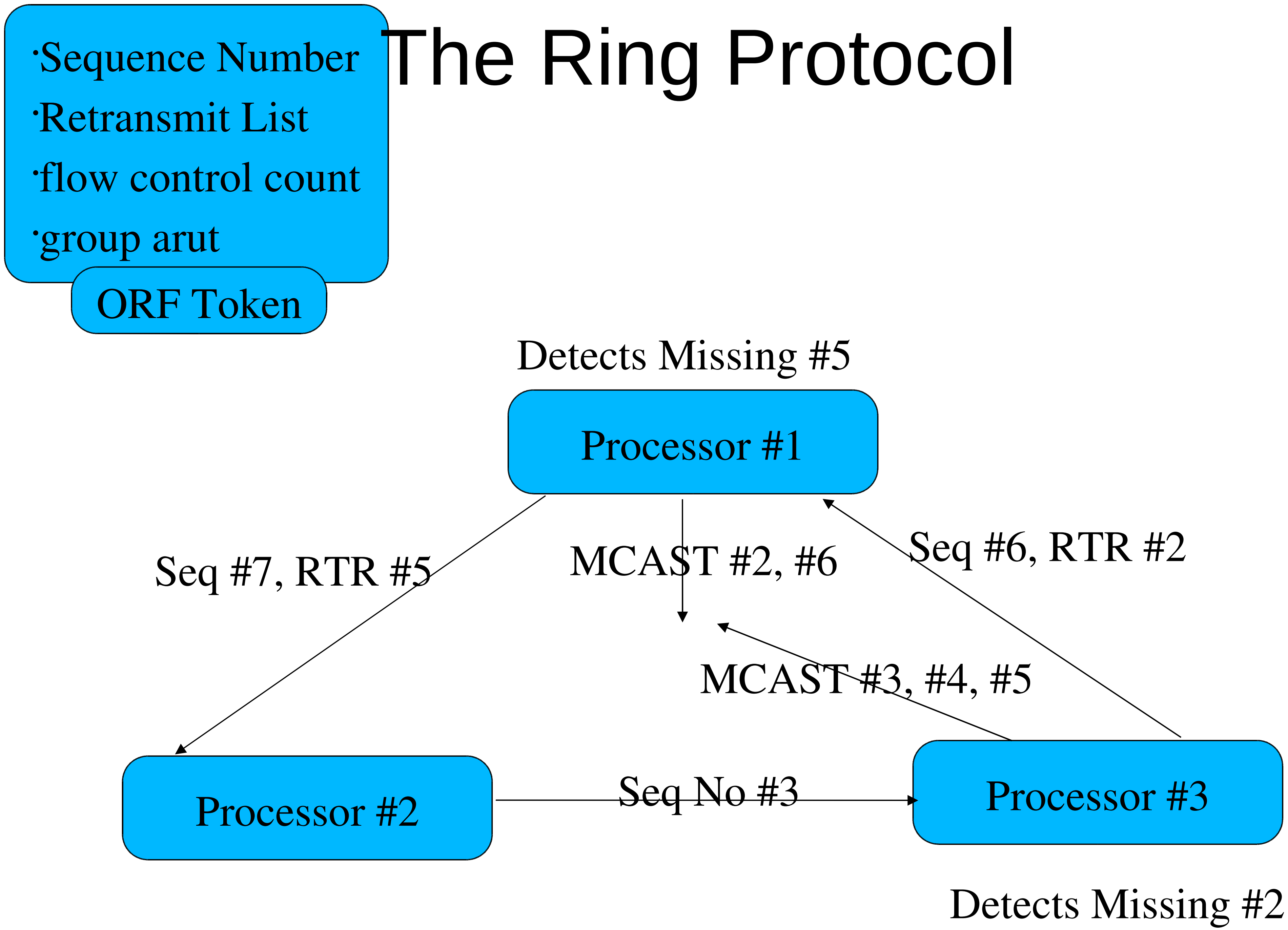
ORF Token



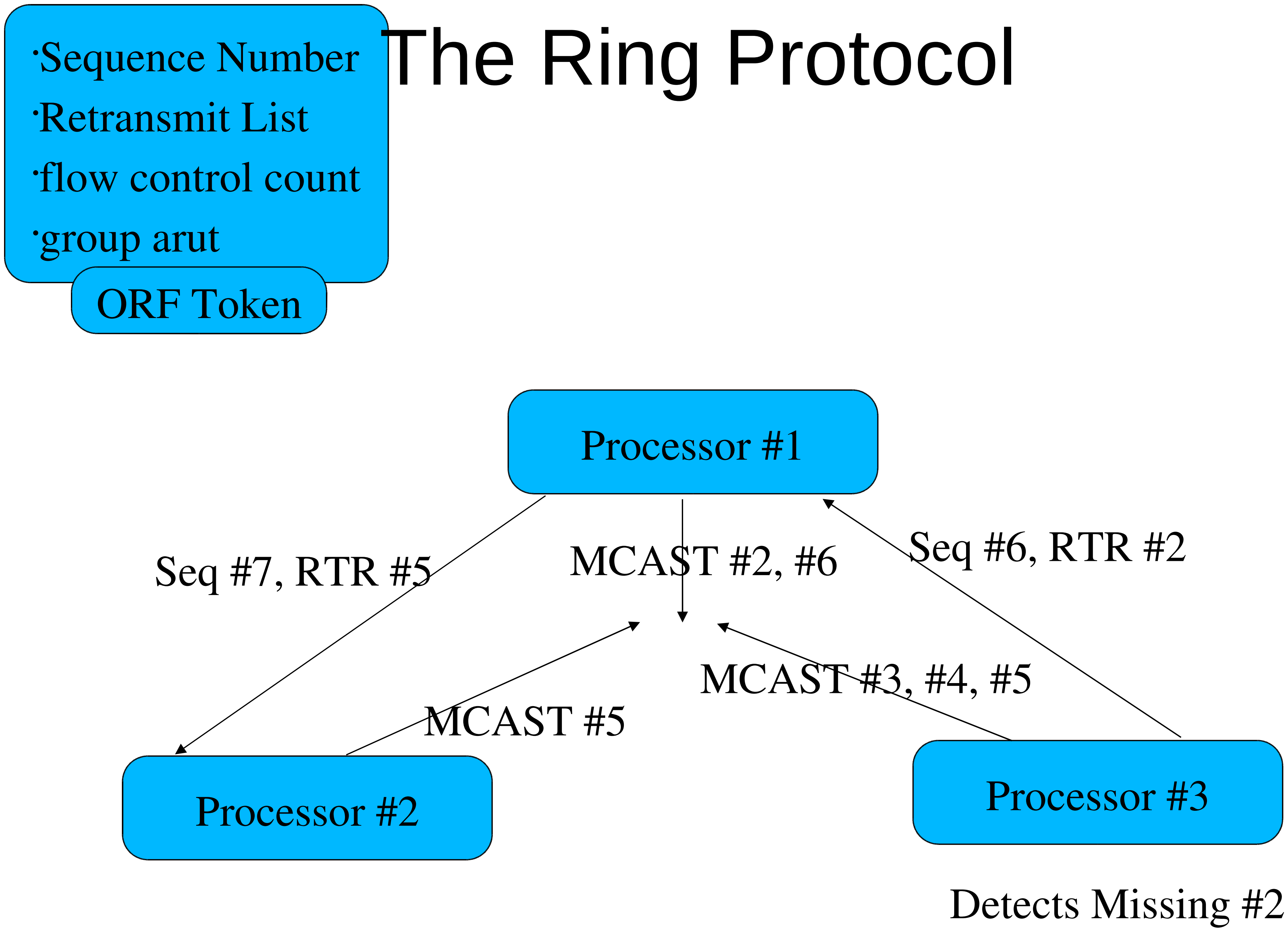
The Ring Protocol



The Ring Protocol



The Ring Protocol



Example Problem – Lock Service Client / Server Approach

- One server contains list of locks.
- A lock request is sent to the server.
- The server processes the request.
- The server responds to the client.
- maximum 1700 locks per second – tied directly to Ethernet access time.

Example Problem – Lock Server Virtual Synchrony Approach

- List of all locks contained on all processors
- processor acquires lock by sending message requesting lock
- when message is self-delivered lock is acquired
- because all processors have replica of locks, no request/response is required
- maximum locks per second – depends on cpu speed but at least 30,000 per second

What are Closed Process Groups

- Maintains membership at process group level
- A processor is uniquely identified by Processor and Process ID

Example:

P1 (pid 5), P2 (pid 6), P3 (pid 7), P4 (pid 8) joined to process group A

P1 (pid 5): M1 M2 M3 M4 CTA(P1,5)(P2,6)(P3,7)(P3,12) CRA(P1,5)(P2,6)(P3,7)(P3,12)

P2 (pid 6): M1 M2 M3 M4 CTA(P1,5)(P2,6)(P3,7)(P3,12) CRA(P1,5)(P2,6)(P3,7)(P3,12)

P3 (pid 7): M1 M2 M3 M4 CTA(P1,5)(P2,6)(P3,7)(P3,12) CRA(P1,5)(P2,6)(P3,7)(P3,12)

P3 (pid 12): M1 M2 M3 M4 CTA(P1,5)(P2,6)(P3,7)(P3,12) CRA(P1,5)(P2,6)(P3,7)(P3,12)

4 messages sent before P4 (pid12) fails and P3 (pid12) fails before it delivers M8

P1 (pid 5): M5 M6 M7 CTA(P1,5)(P2,6)(P3,7) M8 CRA(P1,5)(P2,6)(P3,7)

P2 (pid 6): M5 M6 M7 CTA(P1,5)(P2,6)(P3,7) M8 CRA(P1,5)(P2,6)(P3,7)

P3 (pid 7): M5 M6 M7 CTA(P1,5)(P2,6)(P3,7) M8 CRA(P1,5)(P2,6)(P3,7)

P4 (pid 12): M5 M6 M7 fails

CPG – Interface Properties

- Supports multiple instances in one application
- Mechanism to obtain current membership view
- Mechanism to join and leave a process group
- Mechanism to send and deliver messages to named groups

CPG – Initialize and Finalize

```
cpg_error_t cpg_initialize (
    cpg_handle_t *handle,
    cpg_callbacks_t *callbacks);

cpg_error_t cpg_finalize (
    cpg_handle_t handle);
```

example:

```
cpg_callbacks_t callbacks = {
    .cpg_deliver_fn = cpg_ex_deliver_fn,
    .cpg_confchg_fn = cpg_ex_confchg_fn
};
cpg_handle_t handle;

cpg_initialize (&handle, &callbacks);
cpg_finalize (handle);
```

CPG – Obtaining Current View

```
cpg_error_t cpg_membership_get (
    cpg_handle_t handle,
    struct cpg_name *group_name,
    struct cpg_address *member_list,
    int *member_list_entries);
```

example:

```
cpg_name = {
    .length = 7
    .value = “example”
};

struct cpg_address members[CPG_MEMBERS_MAX];
int member_list_entries;
cpg_membership_get (cpg_handle, &cpg_name, members,
    &member_list_entries);
```

CPG – Joining and Leaving

```
cpg_error_t cpg_join (
    cpg_handle_t handle,
    struct cpg_name *group_name);

cpg_error_t cpg_leave (
    cpg_handle_t handle,
    struct cpg_name *group_name);
```

example:

```
cpg_name = {
    .length = 7
    .value = “example”
};

cpg_join (handle, &cpg_name);

cpg_leave (handle, &cpg_name);
```

CPG – Publishing a Message

```
cpg_error_t cpg_mcast_joined (
    cpg_handle_t handle,
    cpg_guarantee_t guarantee,
    struct iovec *iovec,
    int iov_len);
```

example:

```
char buf[512000];
struct iovec iov = {
    .iov_base = buf,
    .iov_len = 512000
};

cpg_mcast_joined (handle, CPG_TYPE_AGREED, &iov, 1);
```

CPG – Dispatching Callbacks

```
cpg_error_t cpg_fd_get(
    cpg_handle_t handle,
    int *fd);

cpg_error_t cpg_dispatch (
    cpg_handle_t handle,
    cpg_dispatch_t dispatch_types);
```

example:

```
int fd;

cpg_fd_get (handle, &fd);
select (fd+1, 0, 0, 0);
cpg_dispatch (handle, CPG_DISPATCH_ALL);
```

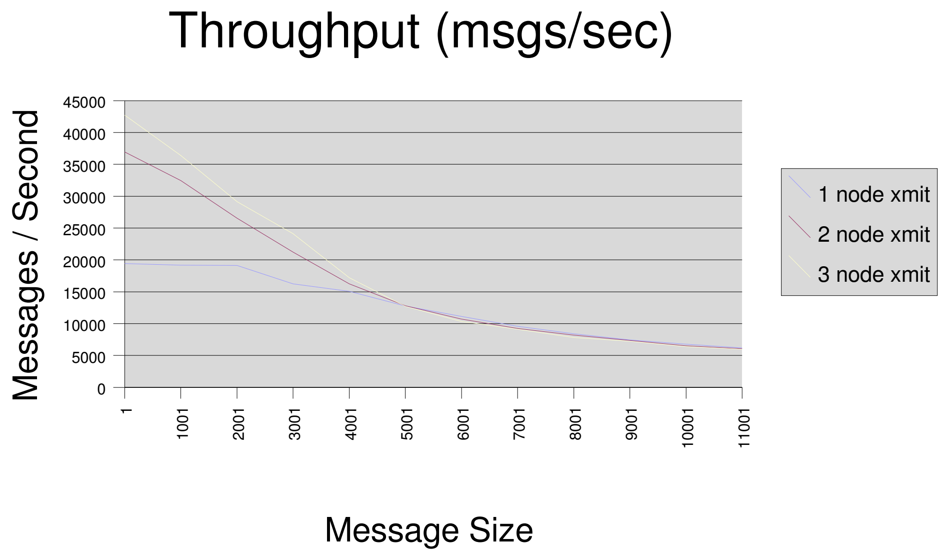
CPG – Delivery Callback

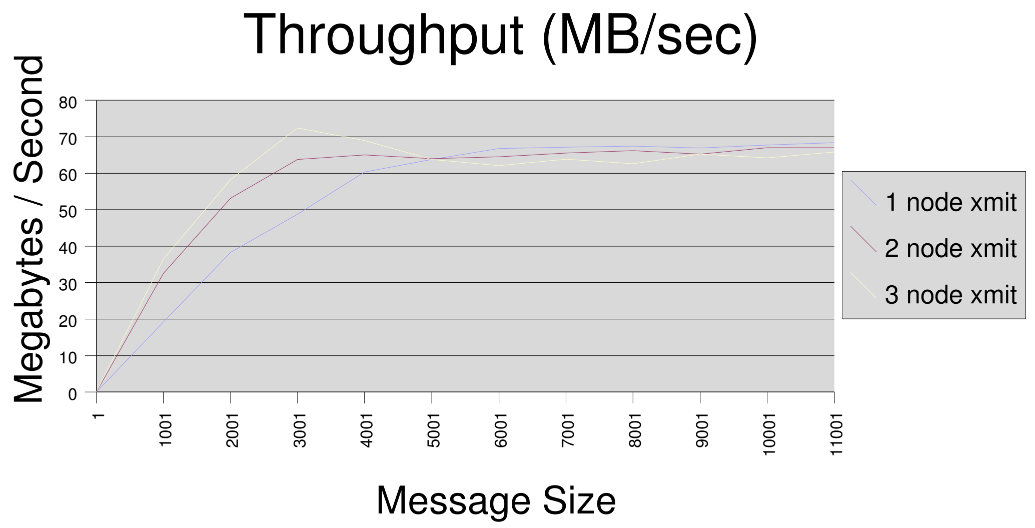
```
cpg_deliver_callback (
    cpg_handle_t handle,
    struct cpg_name *group_name,
    uint32_t nodeid,
    uint32_t pid,
    void *msg,
    int msg_len)
{
    printf (“Delivering message from %d(pid%d) with len %d”,
        nodeid, pid, msg_len);
}
```

CPG – Configuration Change Callback

```
cpg_confchg_callback (
    cpg_handle_t handle,
    struct cpg_name *group_name,
    struct cpg_address *member_list, int member_list_entries,
    struct cpg_address *left_list, int left_list_entries,
    struct cpg_address *joined_list, int joined_list_entries,
{
    printf (“The configuration changed\n”);
}
```

Performance





3 nodes, SMC 8508 switch, 8800 netmtu, 1-3 nodes sending, no encryption as tested with the tool cpgbench distributed with openais test directory

Summary

- Virtual Synchrony provides a powerful mechanism for clustered computing
- CPG provides a simple and powerful API for clustering
- Complete implementation of Totem with recovery of lost messages, encryption, authentication, and extended virtual synchrony
- Support for redundant network interface card operation and jumbo frames for GIGE networks
- High Performance operation