Service-Oriented Programming in MPI

Sarwar Alam, Humaira Kamal and Alan Wagner
University of British Columbia

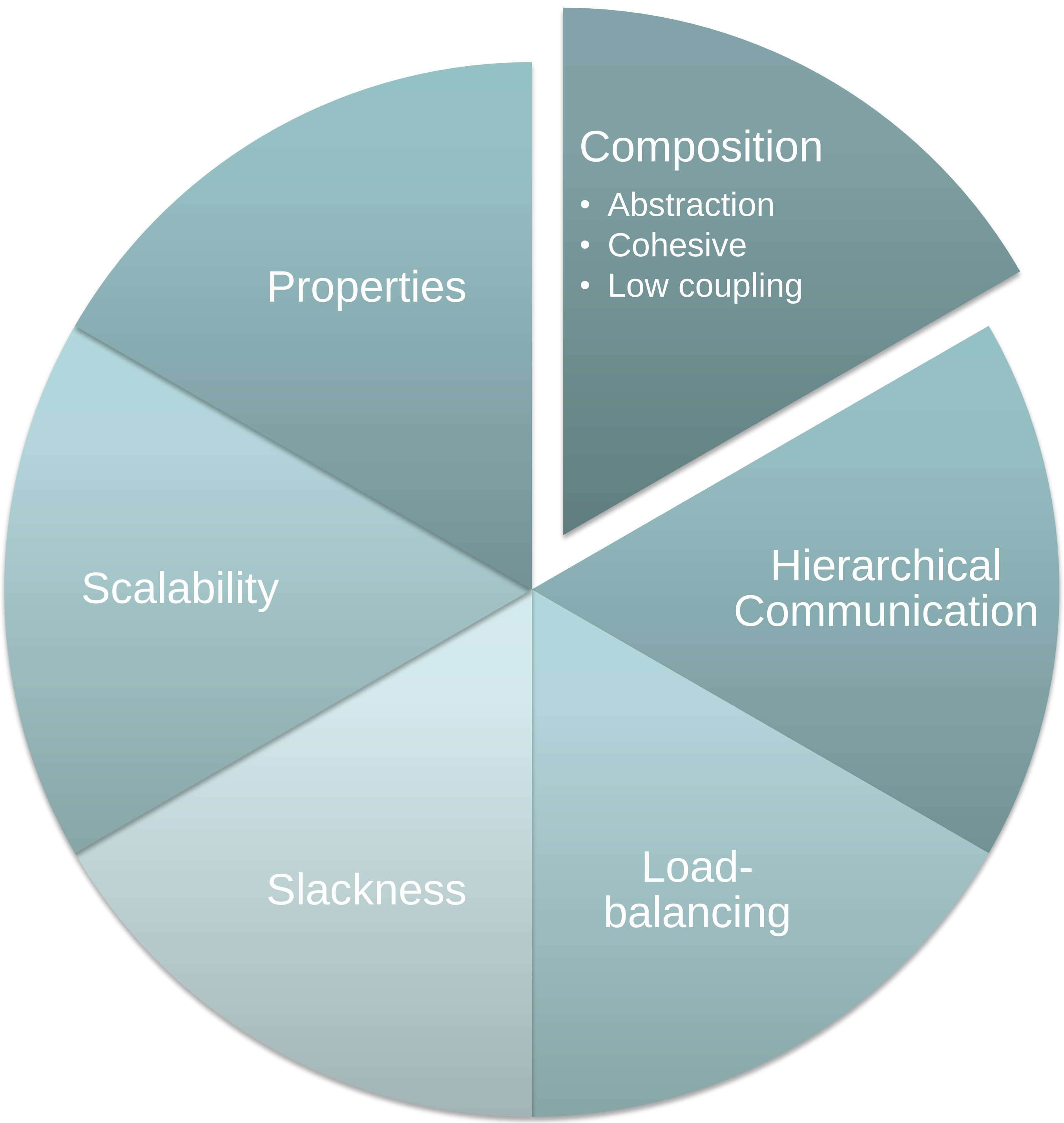
Network
Systems
Security
Lab

Overview

Problem: How to provide data structures to MPI?

- Fine-Grain MPI
- Service-Oriented Programming
- Performance Tuning

Issues


Composition
• Abstraction
• Cohesive
• Low coupling
Hierarchical Communication
Load-balancing
Slackness
Scalability
Properties

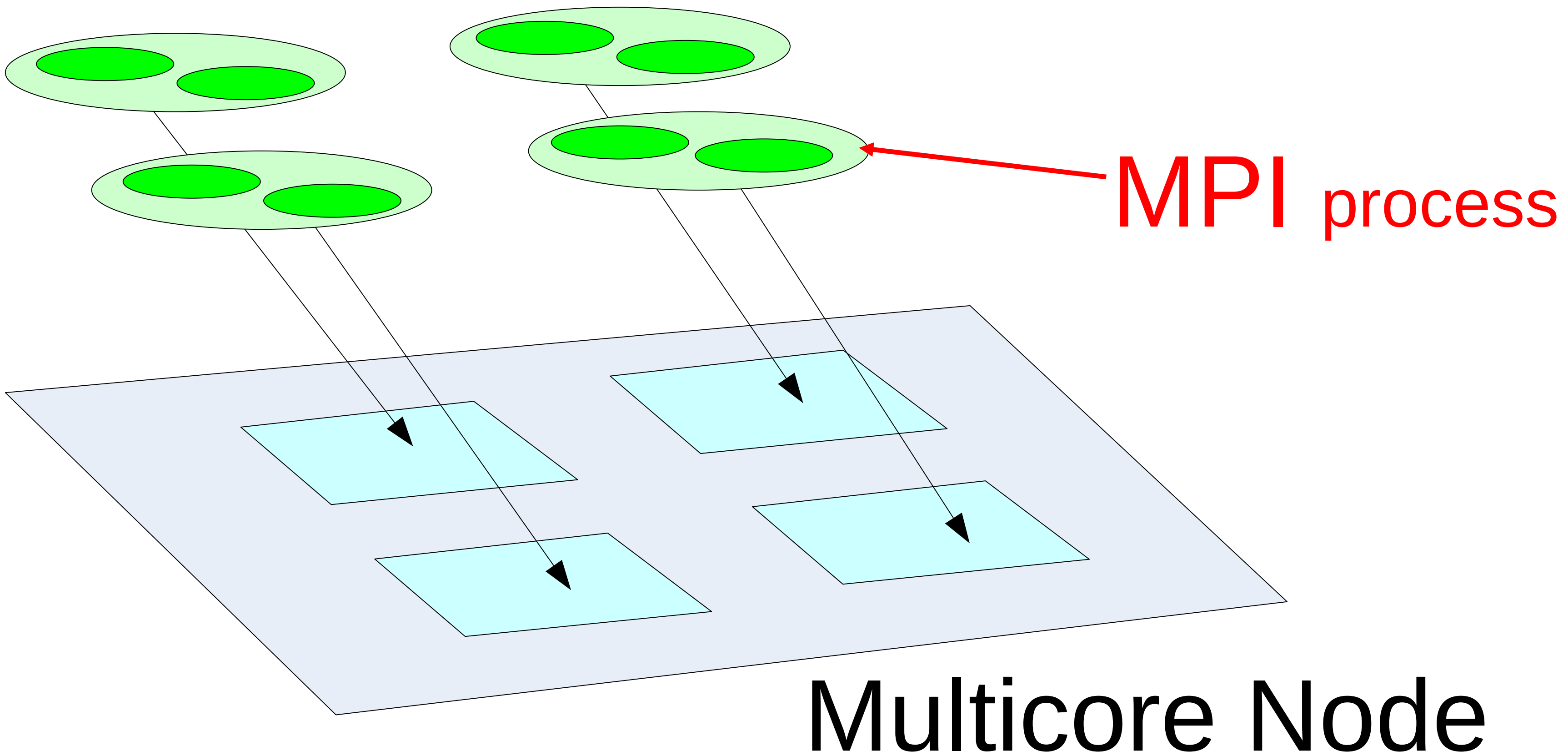
Fine-Grain MPI

MPI

- Advantages
  - Efficient over many fabrics
  - Rich communication library
- Disadvantages
  - Bound to OS processes
  - SPMD programming model
  - Course-grain

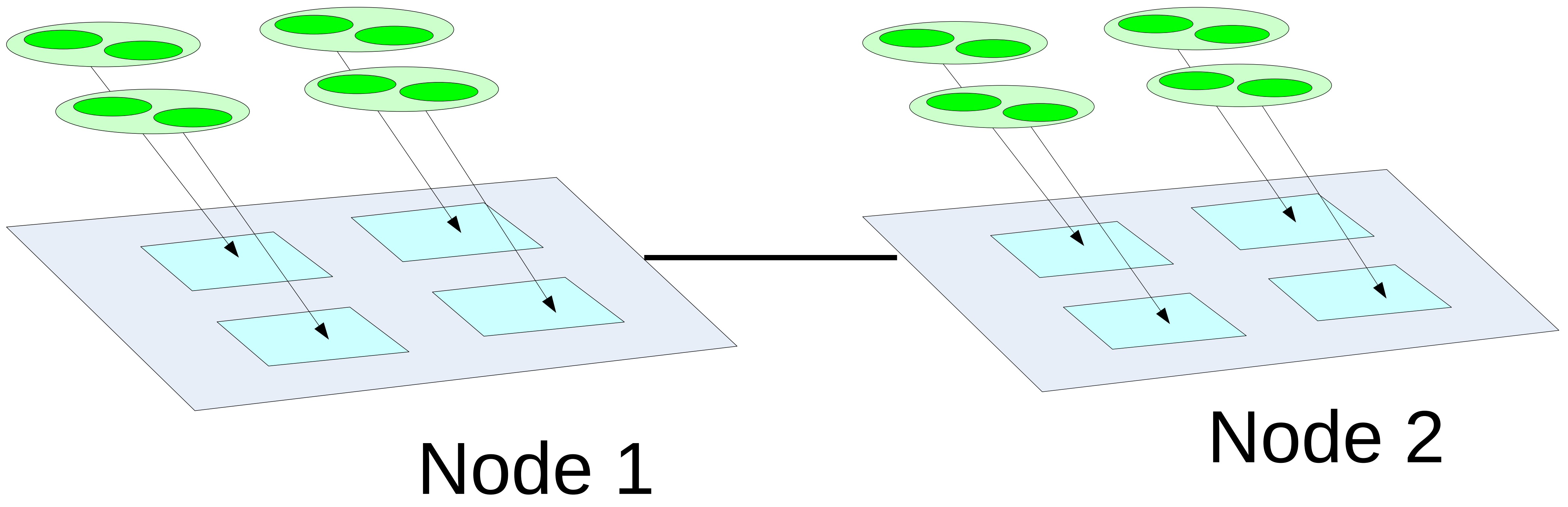
Fine-Grain MPI

Program: OS processes with co-routines (fibers)



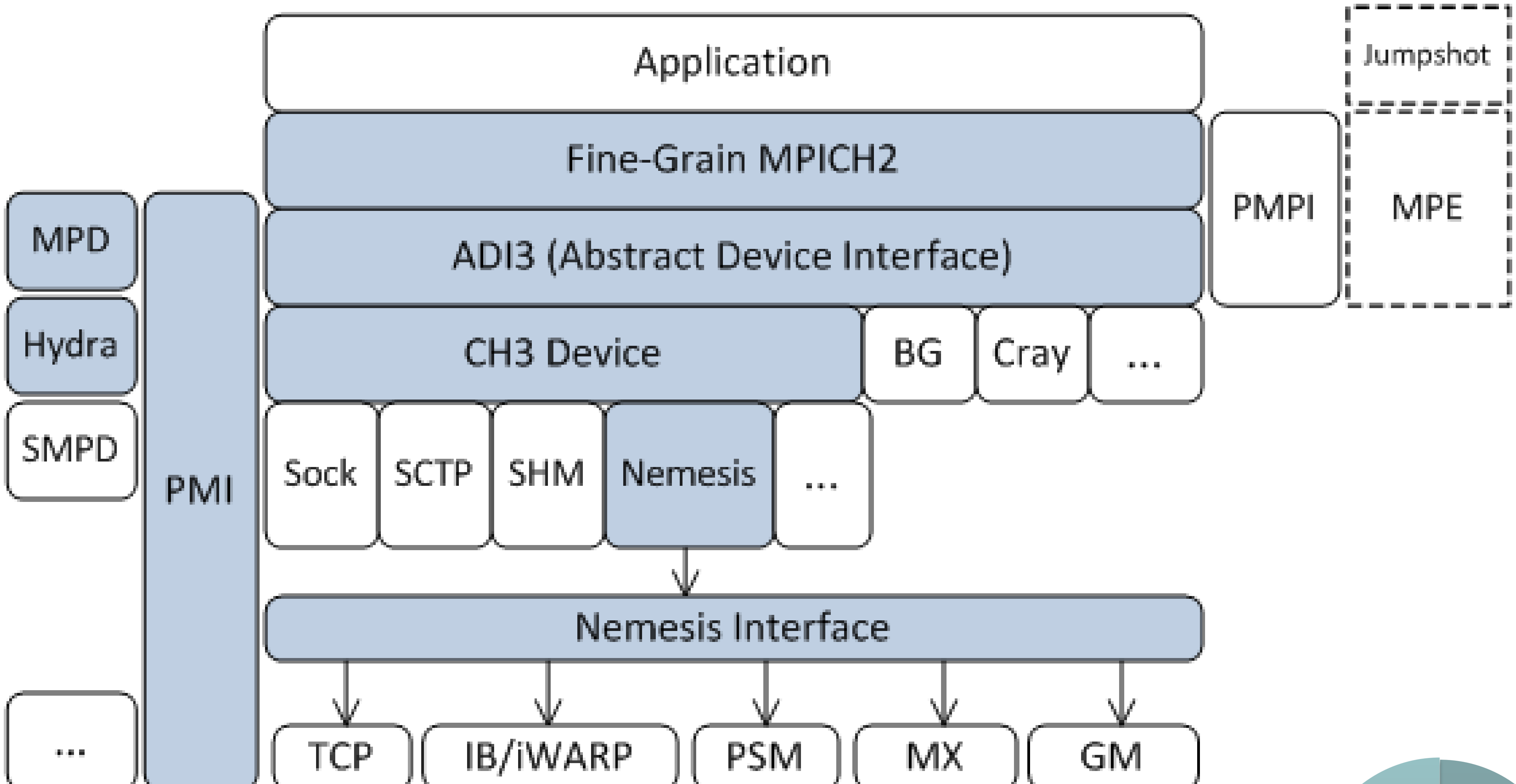
- Full-fledged MPI “processes”
- Combination of OS-scheduled and user-level light-weight processes inside each process

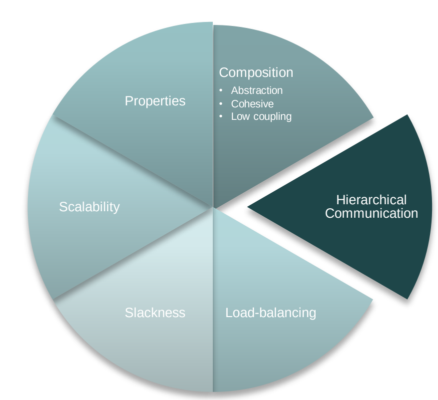
Fine-Grain MPI



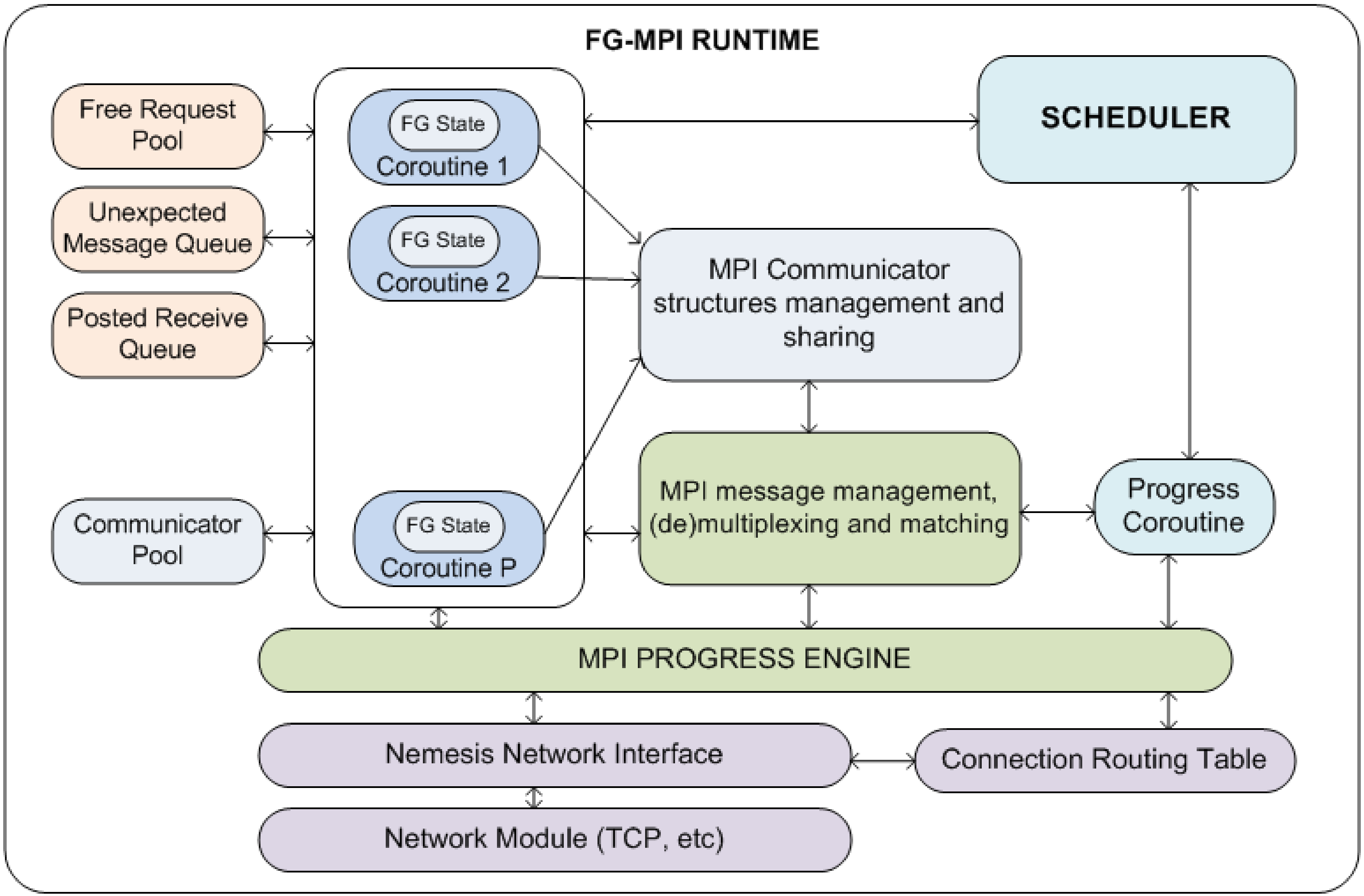
- One model, inside and between nodes
  - Interleaved Concurrency
  - Parallel: same node between nodes

Integrated into MPICH2





System Details



Executing FG-MPI Programs

```
mpiexec -nfg 2 -n 8 myprog
```

- Example of SPMD MPI program
  - with 16 MPI processes,
  - assuming two nodes with quad-core.

8 pairs of processes executing in parallel, where each pair interleaves execution

Decoupled from Hardware

```
mpiexec -nfg 350 -n 4 myprog
```

- Fit the number of processes to the problem rather than the number of cores

Flexibility

```
mpiexec -nfg 1000 -n 4 myprog
```

```
mpiexec -nfg 500 -n 8 myprog
```

```
mpiexec -nfg 750 -n 4 myprog: -nfg 250 -n 4 myprog
```

- Move the boundary between light-weight user scheduled concurrency, and processes running in parallel.

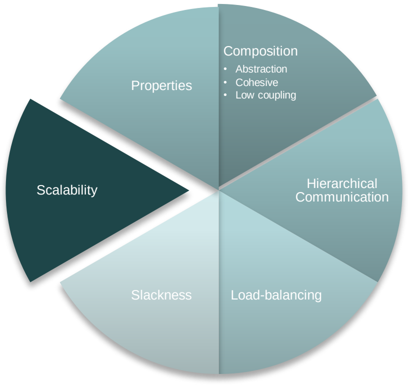
Scalability

```
mpiexec -nfg 30000 -n 8 myprog
```

- Can have hundreds and thousands of MPI processes.

```
mpiexec -nfg 16000 -n 6500 myprog
```

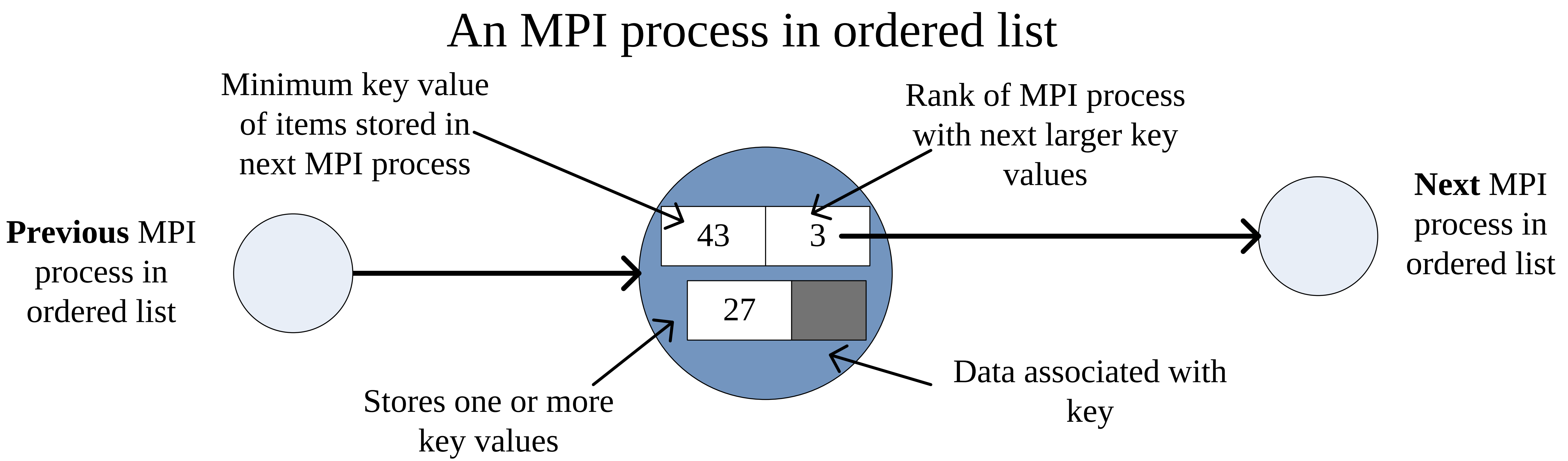
- 100 Million processes on 6500 cores



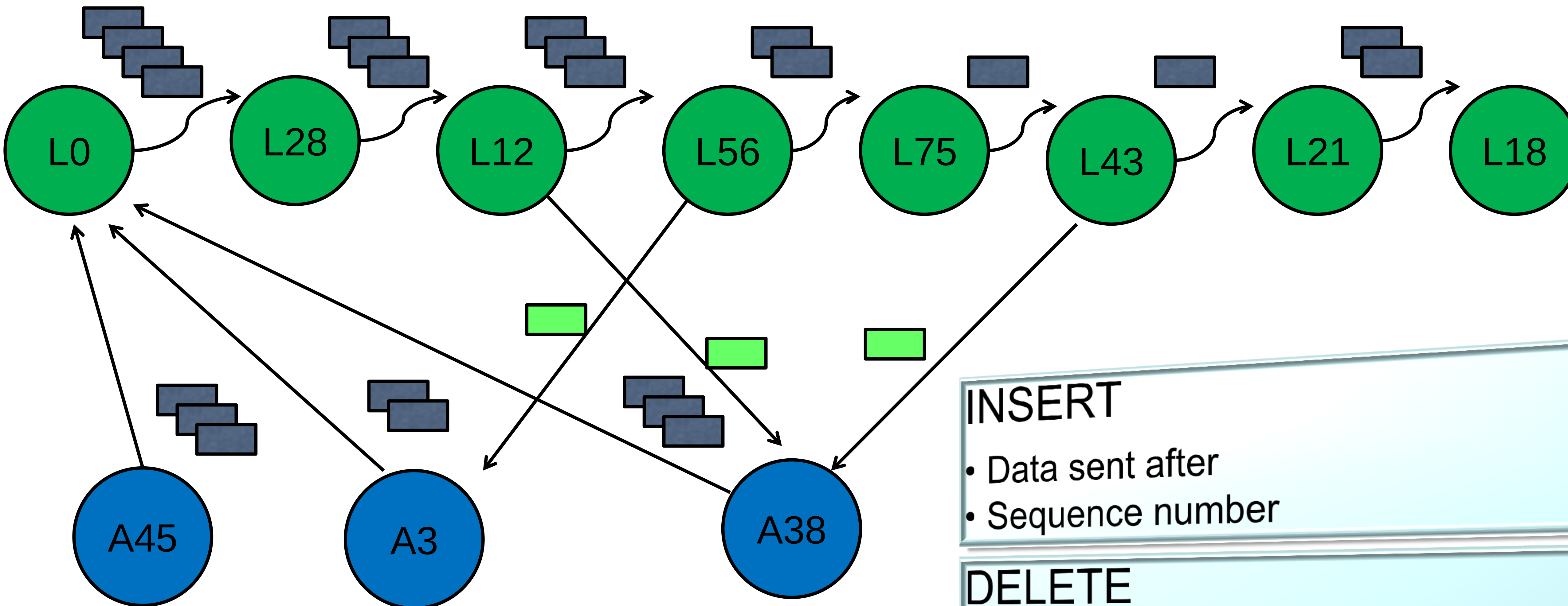
Service-Oriented Programming

- Linked List Structure
- Keys in sorted order
- Similar
  - Distributed hash table
  - Linda Tuple Spaces

Ordered Linked-List



Ordered Linked-List

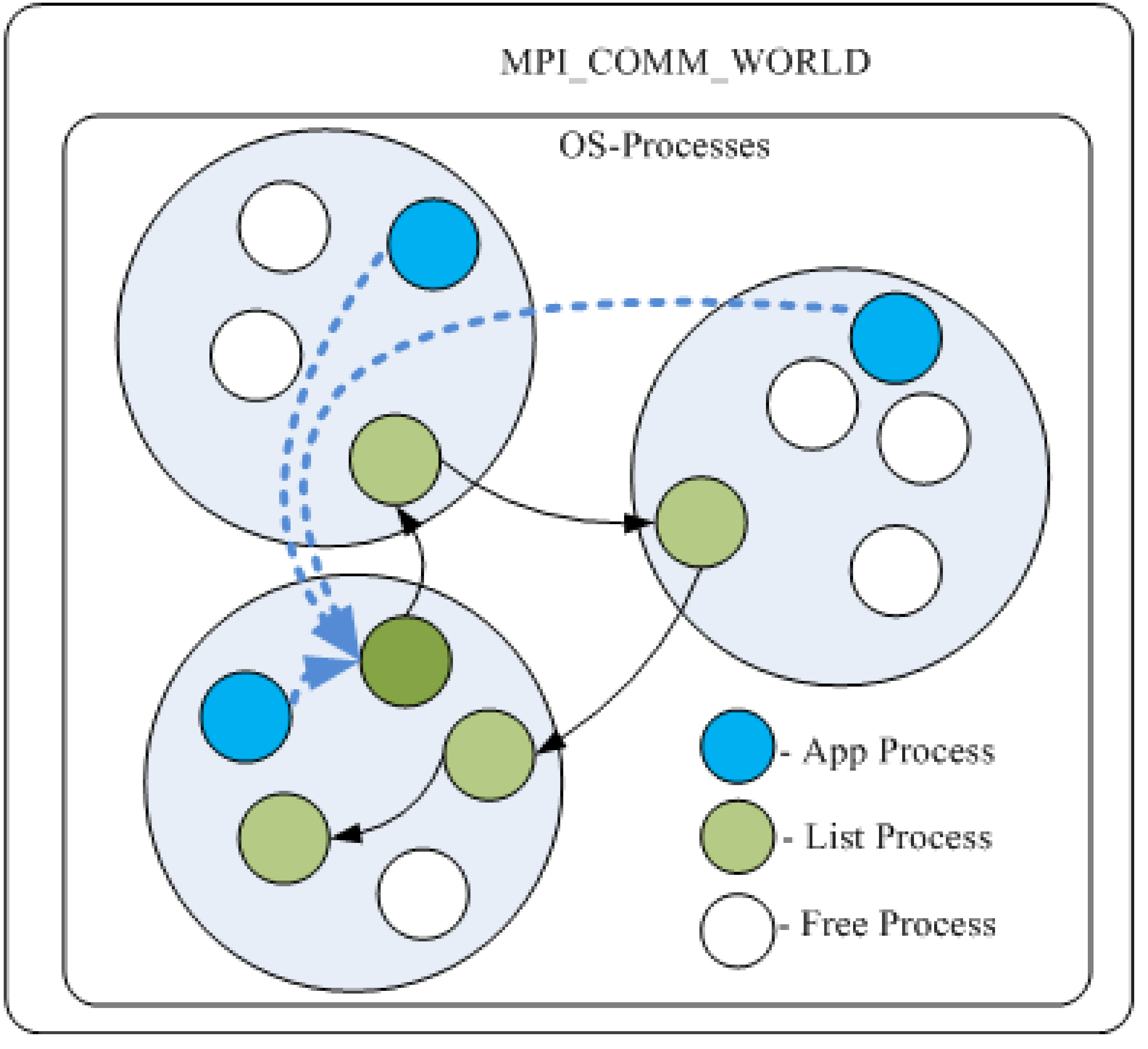


INSERT
- Data sent after
- Sequence number

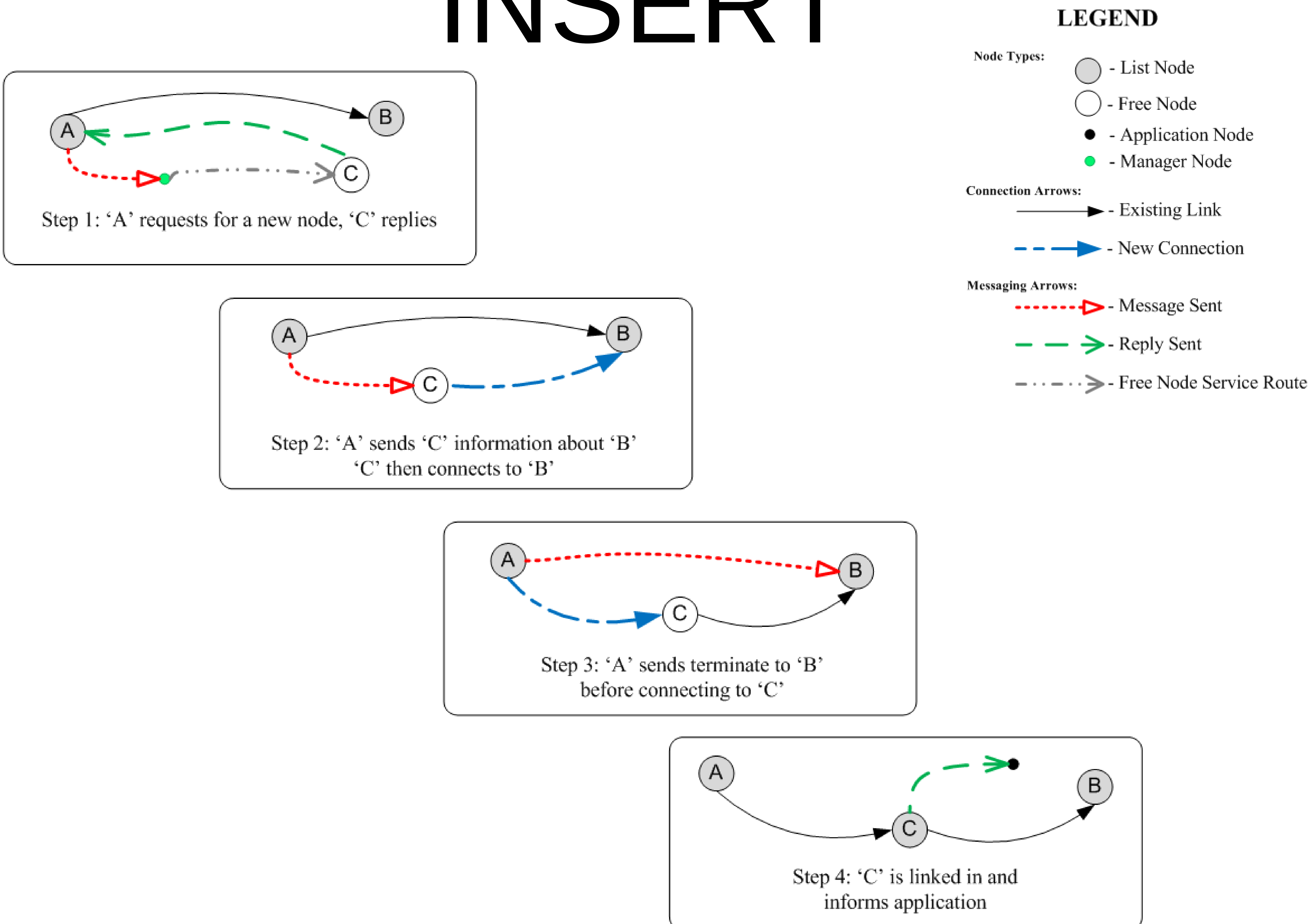
DELETE
- Success/Failure
- Sequence number

FIND
- Sequence number
- Return data

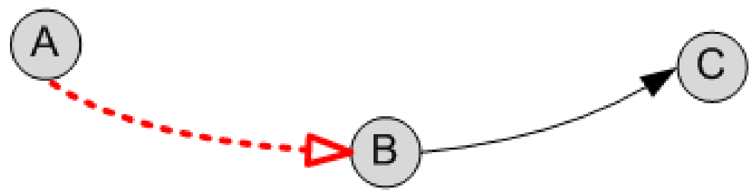
Ordered Linked-List



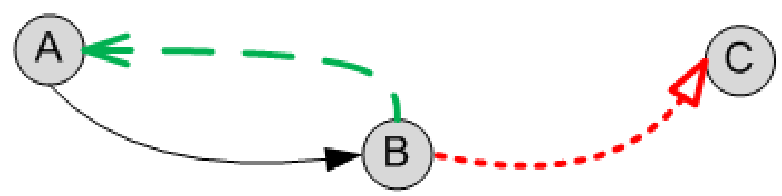
INSERT


LEGEND
Node Types:
- List Node
- Free Node
- Application Node
- Manager Node
Connection Arrows:
- Existing Link
- New Connection
Messaging Arrows:
- Message Sent
- Reply Sent
- Free Node Service Route
A
B
C
Step 1: ‘A’ requests for a new node, ‘C’ replies
Step 2: ‘A’ sends ‘C’ information about ‘B’
‘C’ then connects to ‘B’
Step 3: ‘A’ sends terminate to ‘B’
before connecting to ‘C’
Step 4: ‘C’ is linked in and
informs application

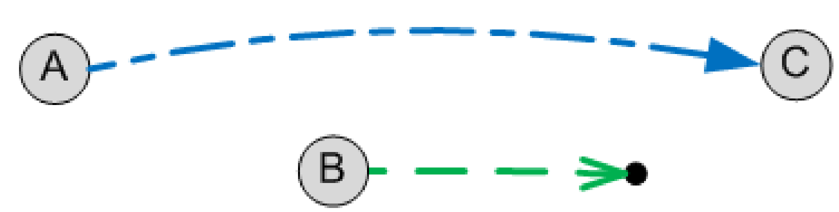
DELETE



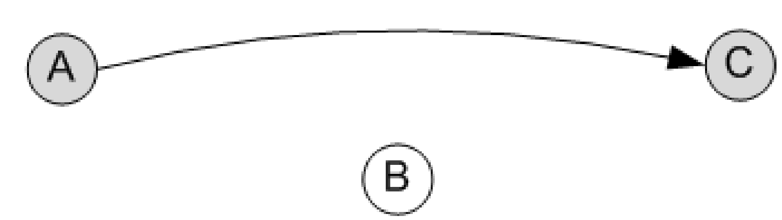
Step 1: 'A' passes delete to 'B' and waits



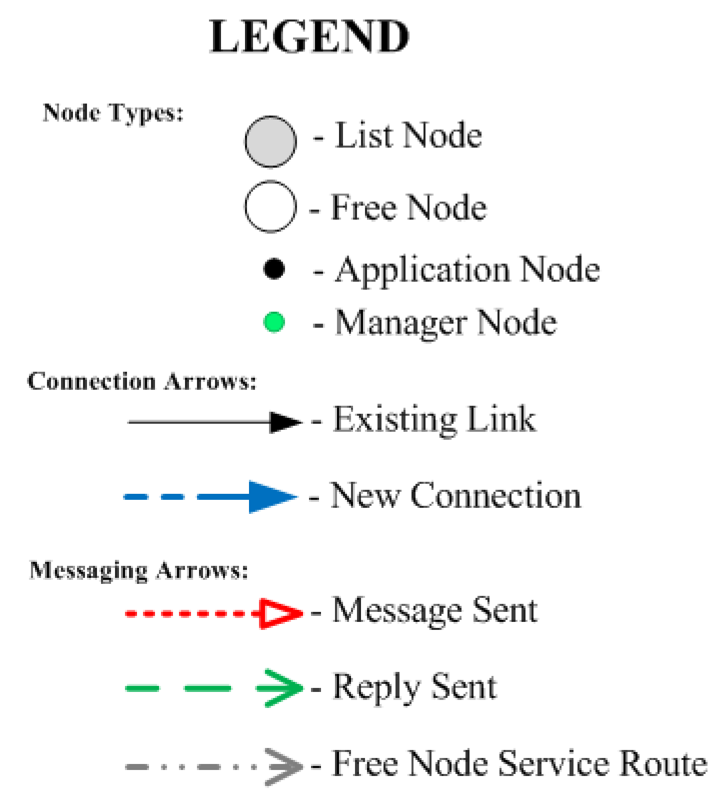
Step 2: 'B' sends terminate to 'C' before acknowledging to A



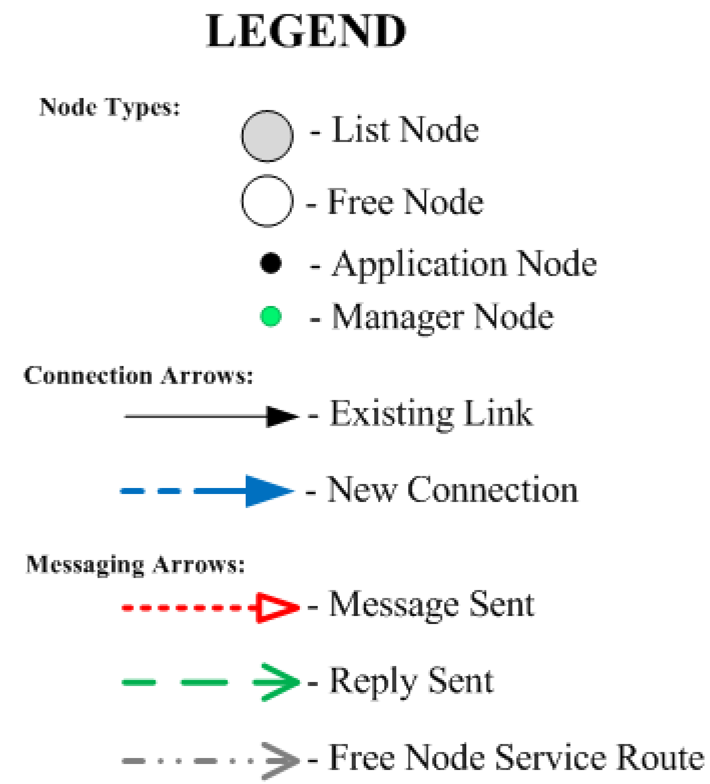
Step 3: 'A' accepts 'C' from 'B' then 'B' informs success to application

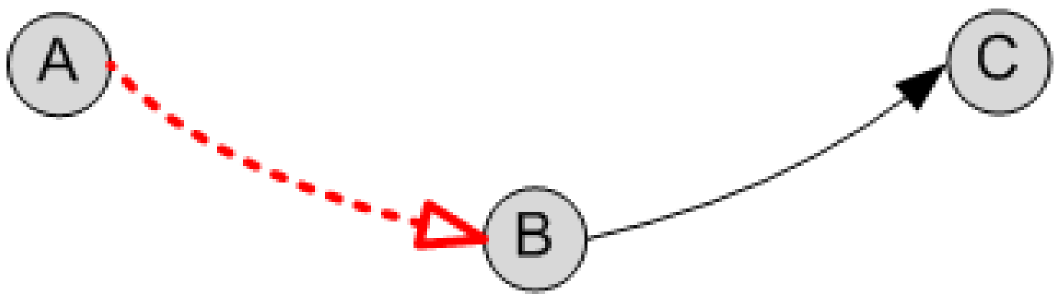


Step 4: 'B' converts to a free node

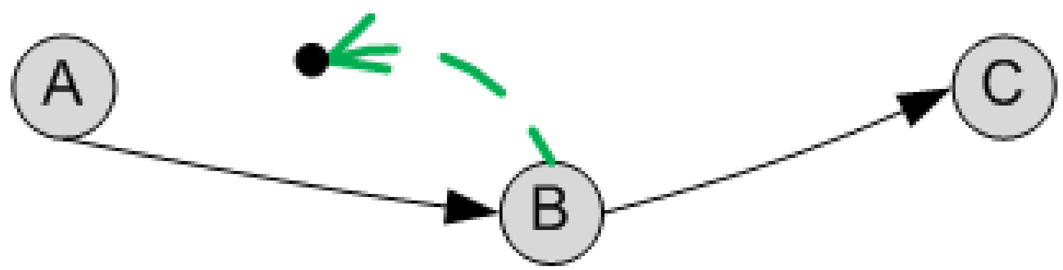


FIND



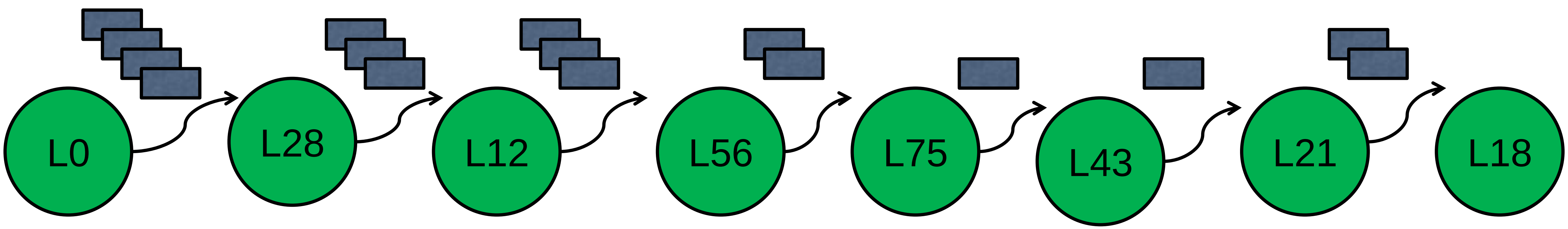


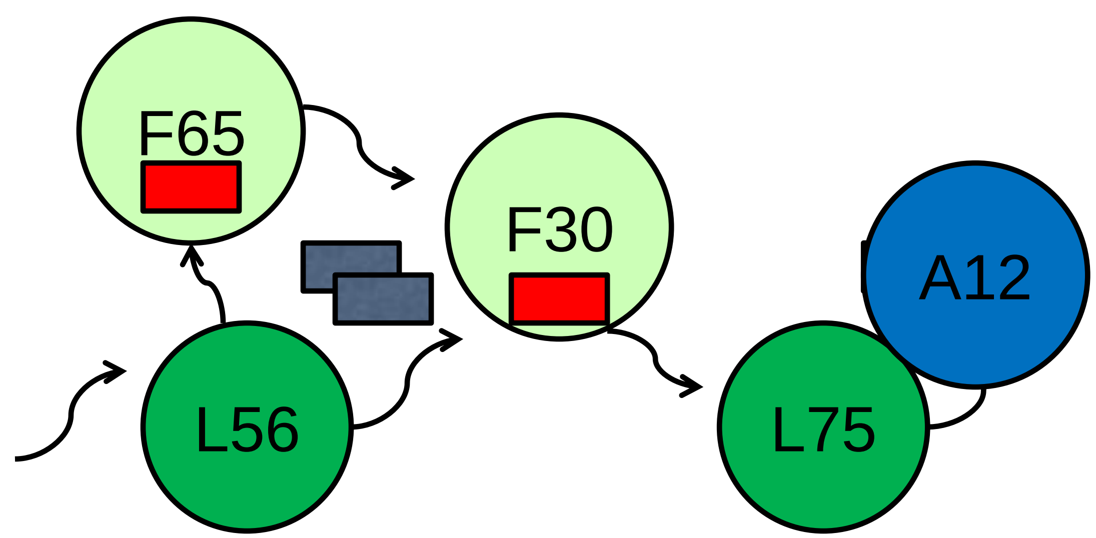
Step 1: ‘A’ passes find to ‘B’



Step 2: If match found or match not possible
‘B’ informs application
(else request passed to ‘C’)

Ordered Linked-List



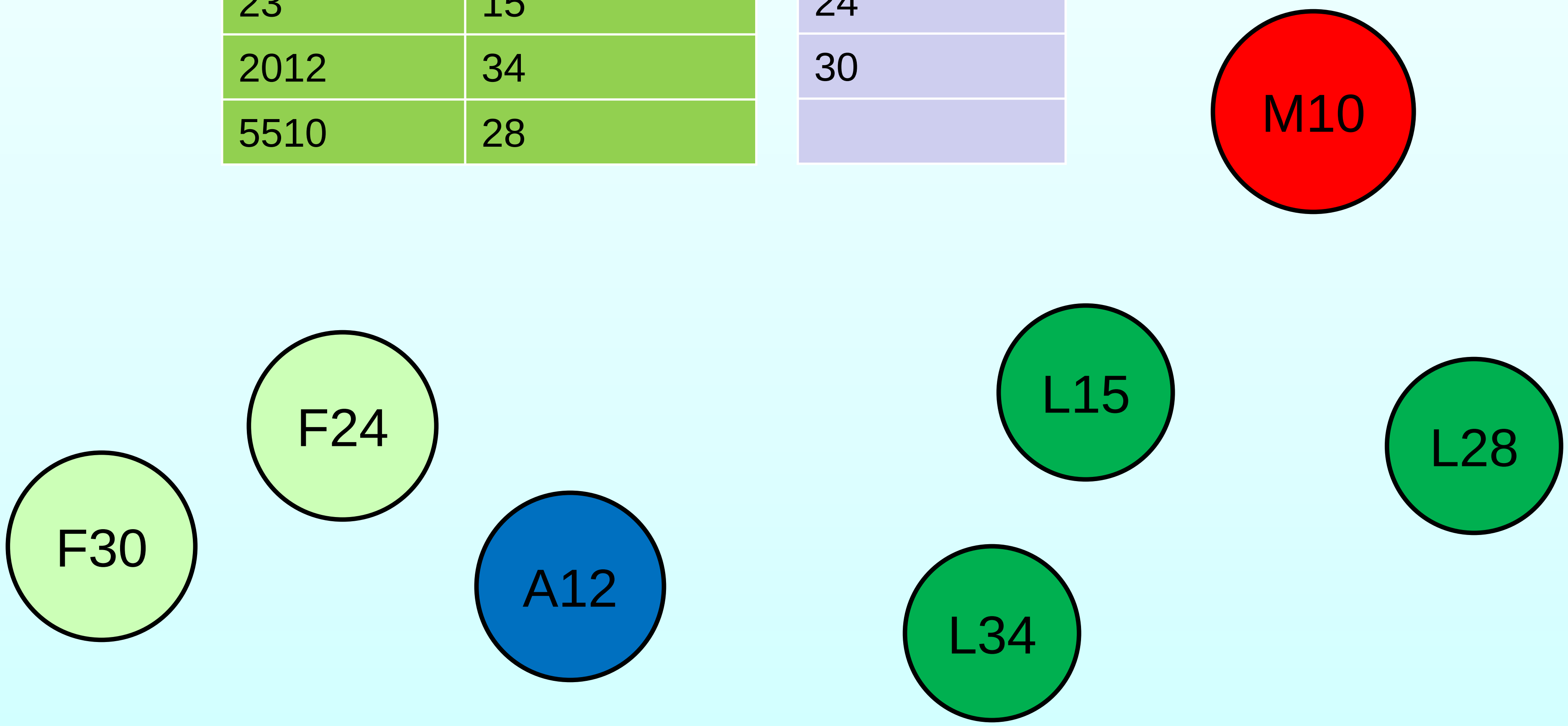


Shortcuts

Local Process Ecosystem

Key value	Rank (ptr)
23	15
2012	34
5510	28

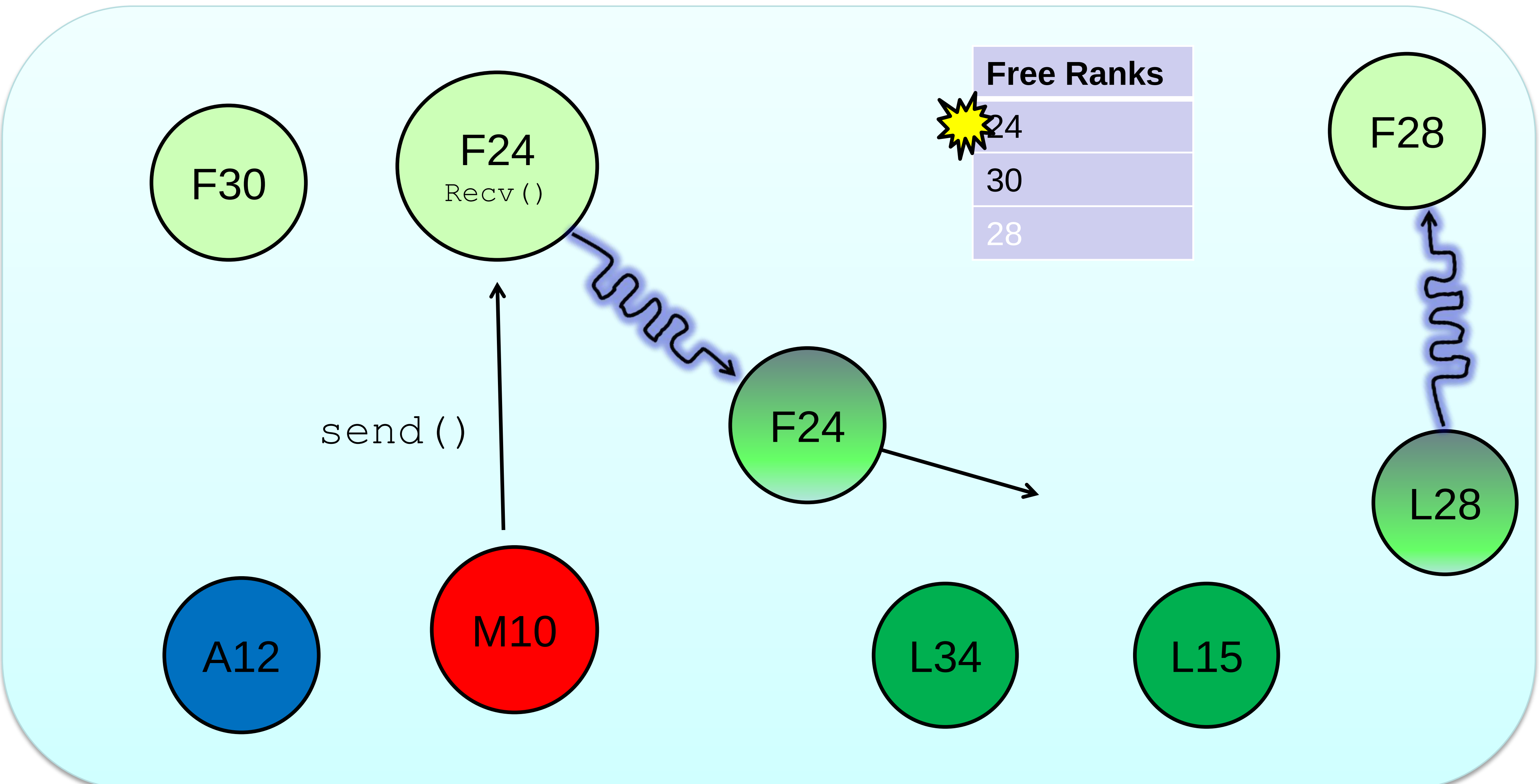
Free Ranks
24
30



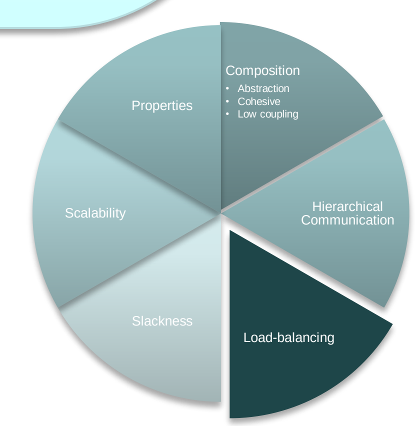
Local non-communication operations are ATOMIC

Re-incarnation

Local Process Ecosystem



Local non-communication operations are ATOMIC

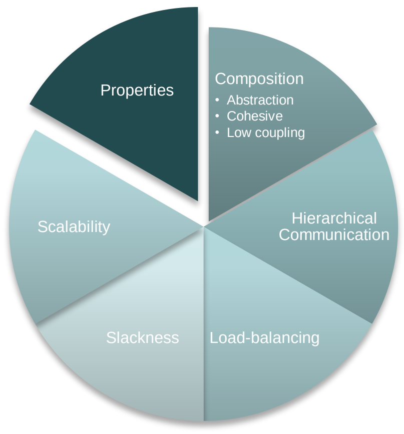


Granularity

- Added the ability for each process to manage a collection of consecutive items.
- Changes to INSERT, changes into a SPLIT operation
- Changes to DELETE, on delete of last item
- List Traversal consists of:
  - Jumping between processes
  - Jumping co-located processes
  - Search inside a process

Properties

- Total Ordered – operations are ordered by the order they arrive at the root
- Sequentially Consistent – each application process keeps a hold-back queue to return results in order
- No consistency – operations can occur in any order

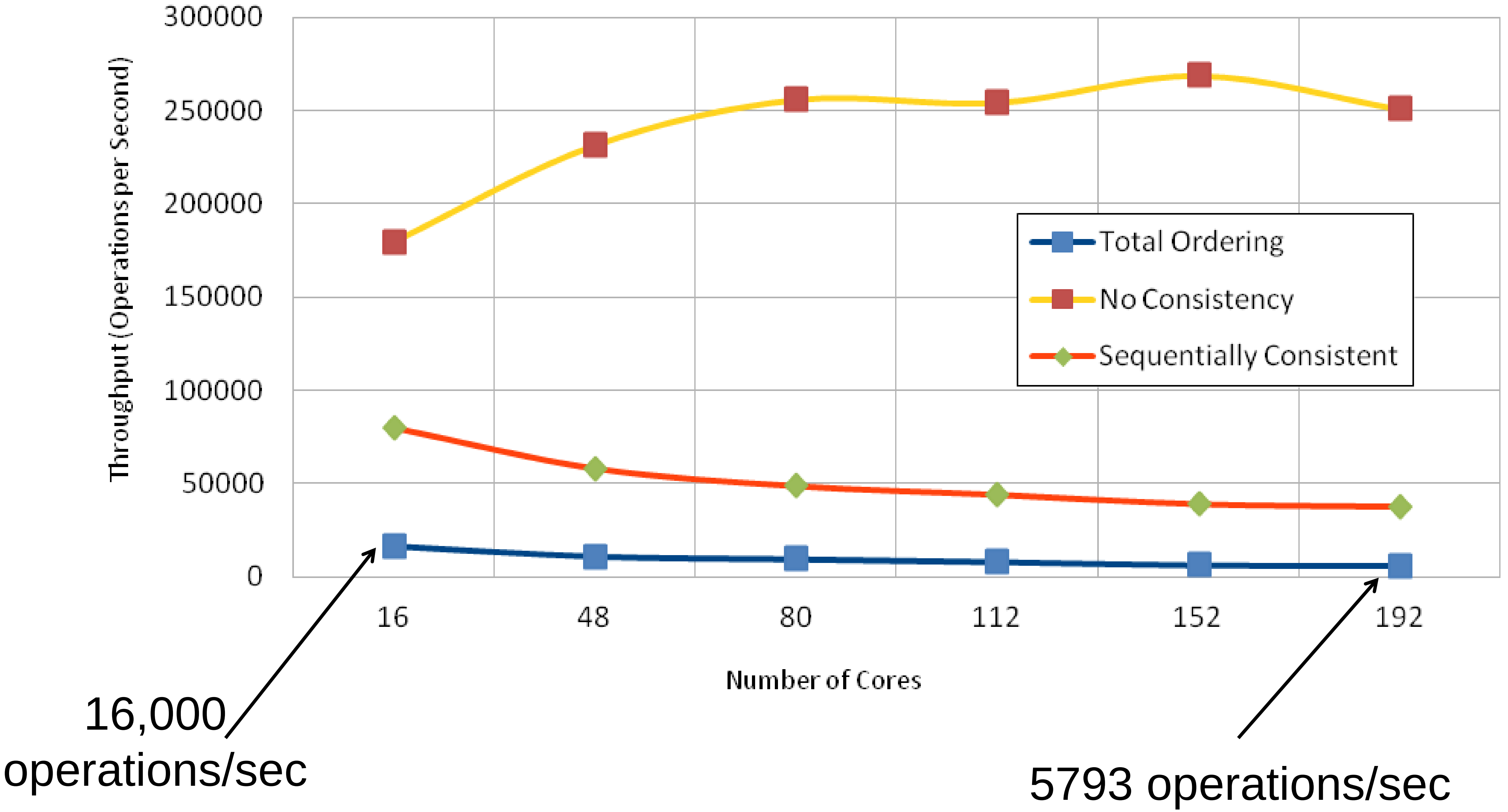


Performance Tuning

- **G** (granularity) the number of keys stored in each process.
- **K** (asynchrony) the number of messages in the channel between list processes.
- **W** (workload) the number of outstanding operations

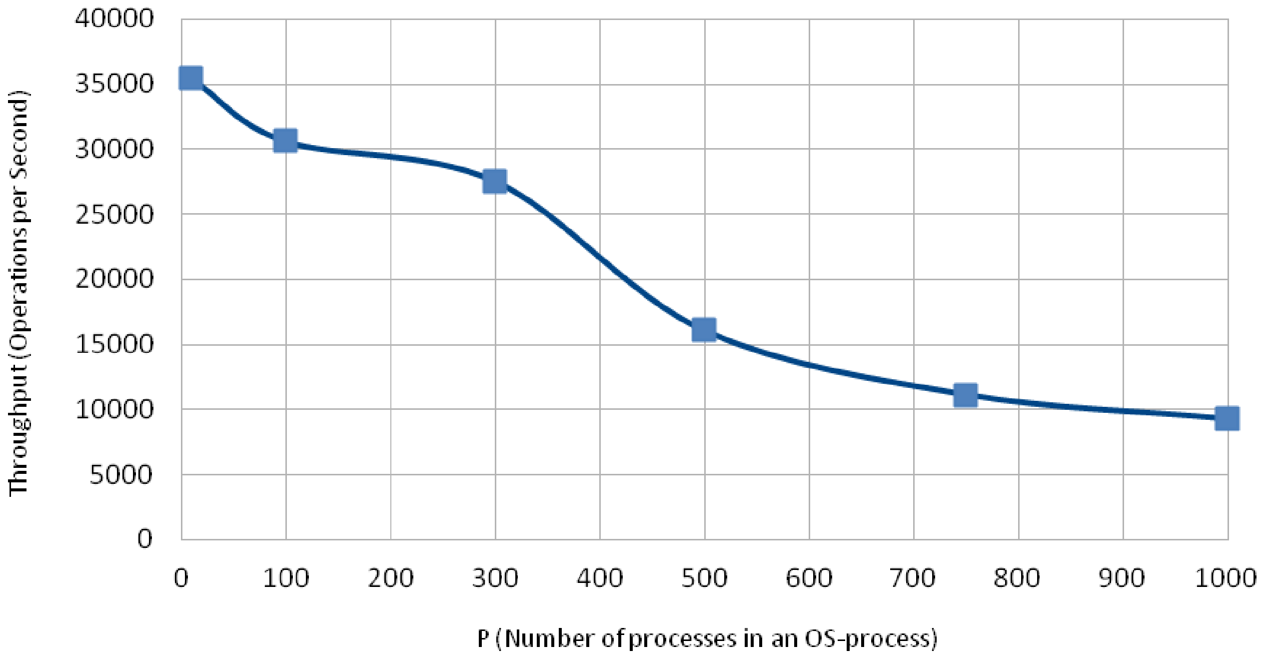
Steady-StateThroughput

Fixed list size, evenly distributed over O x M core



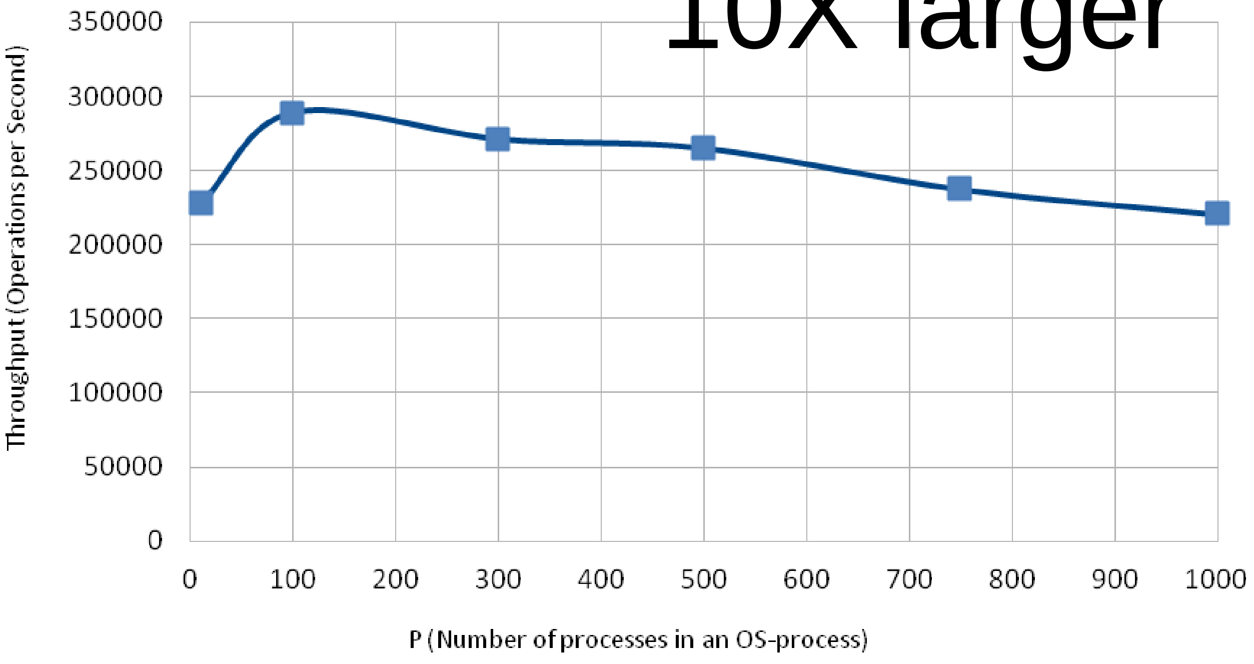
Granularity (G)

Fixed-size machine (176 cores), Fixed list size (2^20)



Sequentially Consistent

10X larger



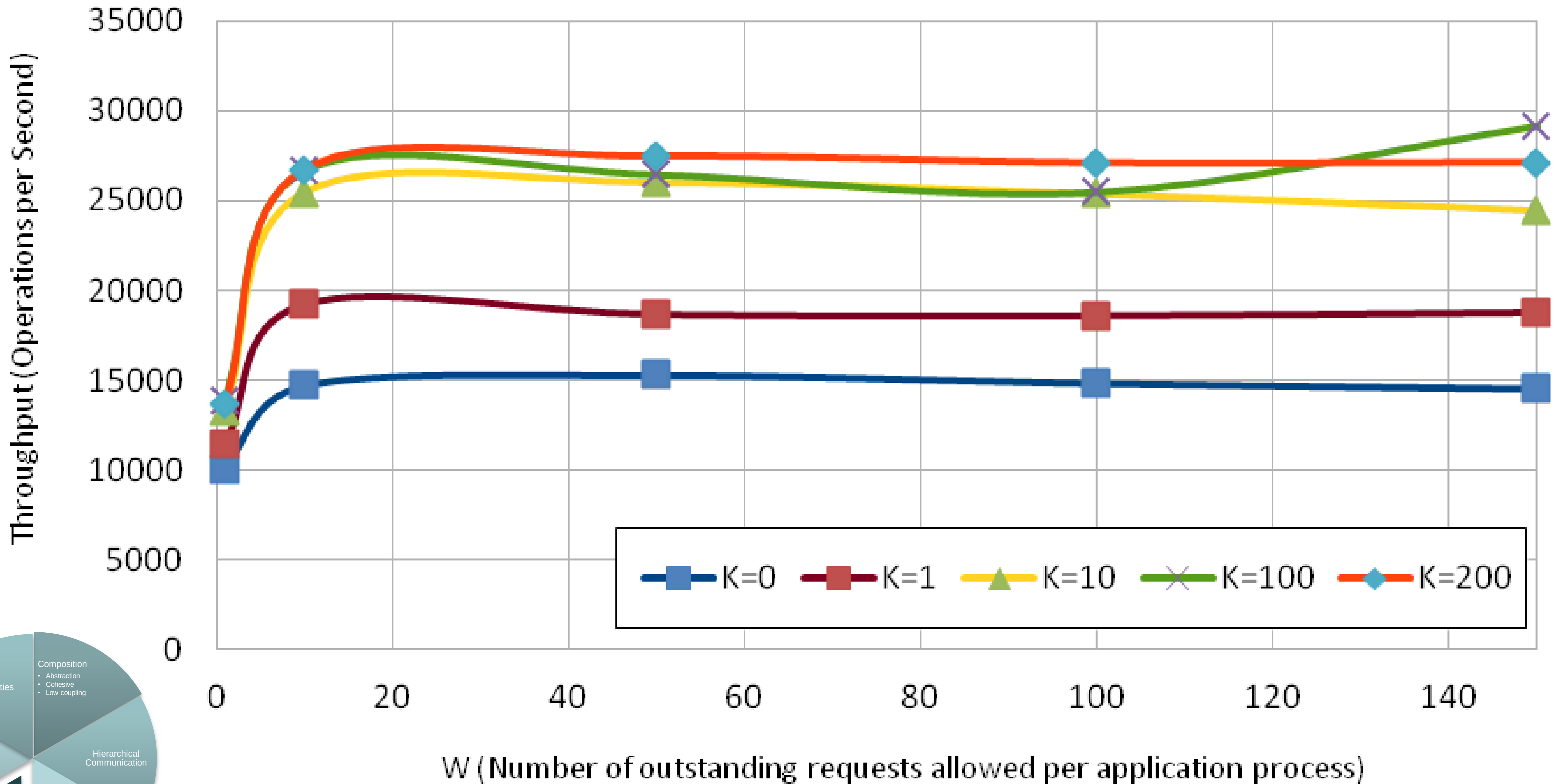
No-consistency

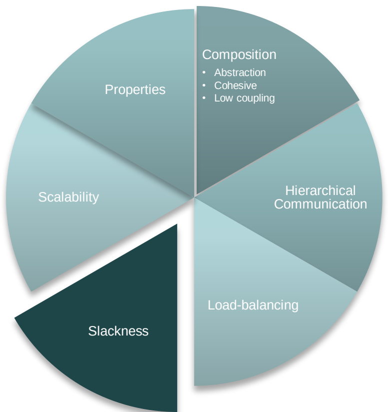
Moving work from INSIDE a process to BETWEEN processes

W and K

W : Number of outstanding requests (workload)

K : Degree of Asynchrony





Conclusions

- Reduced coupling and increased cohesion
- Scalability within clusters of multicore
- Performance tuning controls
  - Adapt to hierarchical network fabric
- Distributed systems properties pertaining to consistency

Thank-You