# CS848 Paper Presentation

# Sinfonia: a new paradigm for building scalable distributed systems

**Aguilera, Merchant, Shah, Veitch, Karamanolis**

**SOSP 2007**

Presented by **Somayyeh Zangooei**

David R. Cheriton School of Computer Science

University of Waterloo

22 February 2010

# Motivation

- Increasing need for scalable distributed systems/applications
  - Large data centers (1000s servers)
  - Serve billions of users around the world
- Sharing data
- Current solution: use message-passing
  - Complex protocols
  - Error prone
  - Hard to use

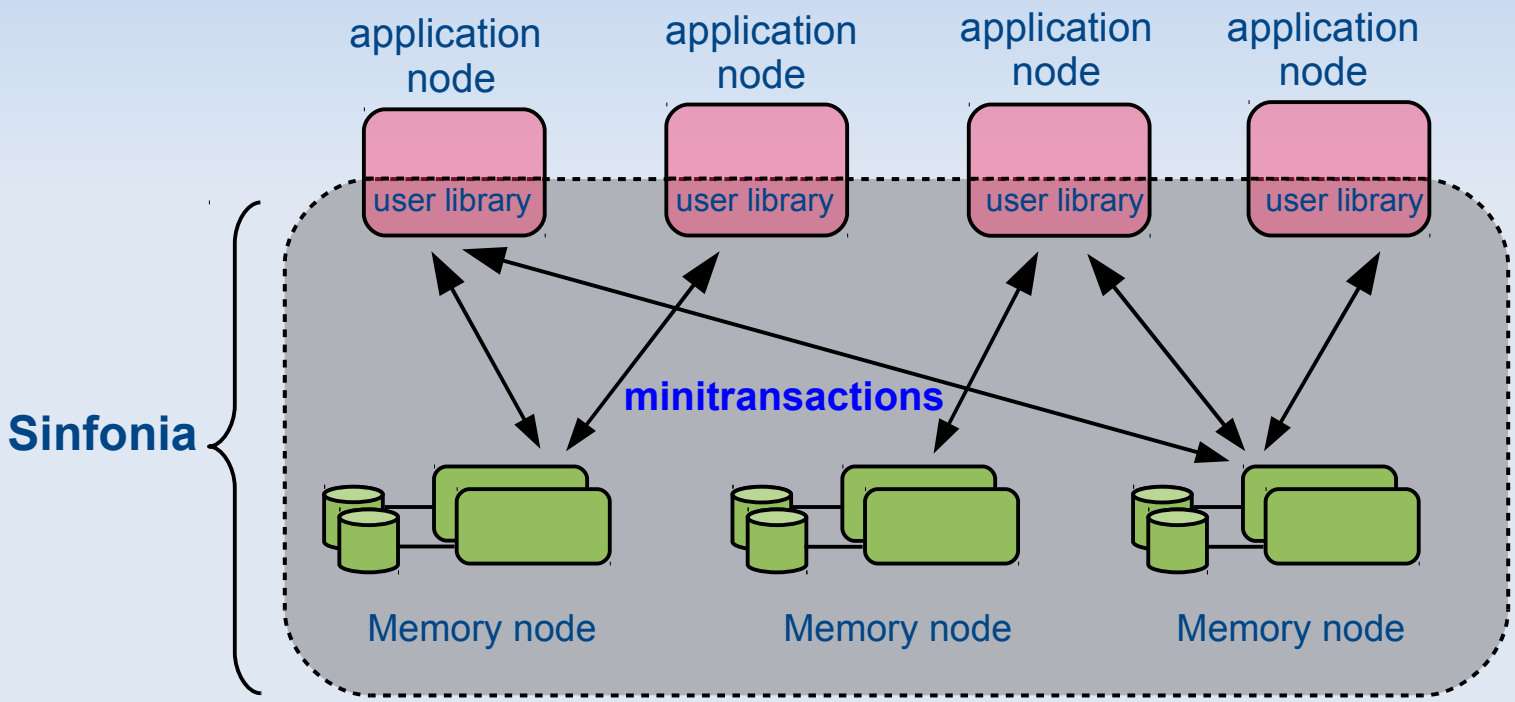
# Outline

- Sinfonia Structure
- Minitransactions
- Design Choices
- Two Applications
- Evaluation
- Conclusion
- Questions & Discussions

# Focus of Sinfonia

- Data Center Environment
  - Small and predictable network latencies
  - Trustworthy applications
  - Nodes may crash
- Target: Infrastructure applications
  - Applications that support other applications
  - Examples: lock managers, cluster file systems, and group communication services
  - Need to provide reliability, consistency, and scalability

# Sinfonia

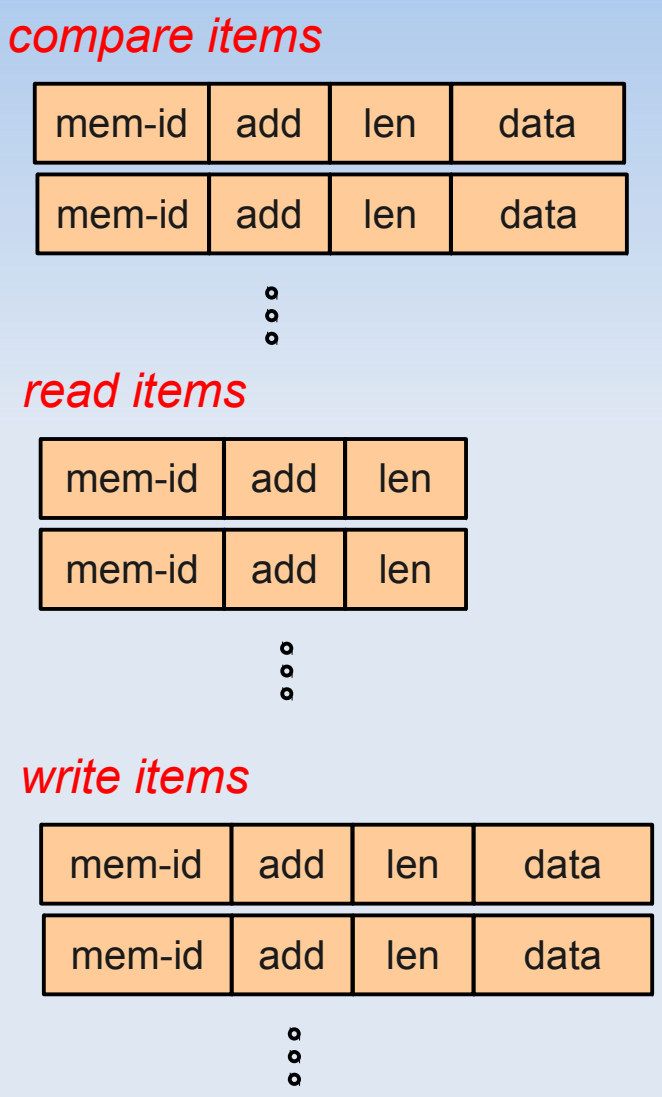

application node
application node
application node
application node
user library
user library
user library
user library
minitransactions
Sinfonia
Memory node
Memory node
Memory node

# Outline

- Sinfonia Structure
- Minitransactions
- Design Choices
- Two Applications
- Evaluation
- Conclusion
- Questions & Discussions

# Minitransactions

- Minitransactions:
  - Atomically update data at multiple memory nodes
  - Consistes of: a set of compare items, a set of read items, a set of write items
- Semantics:
  - Check data in compare items (equality comparison)
  - If all match then apply read and write items



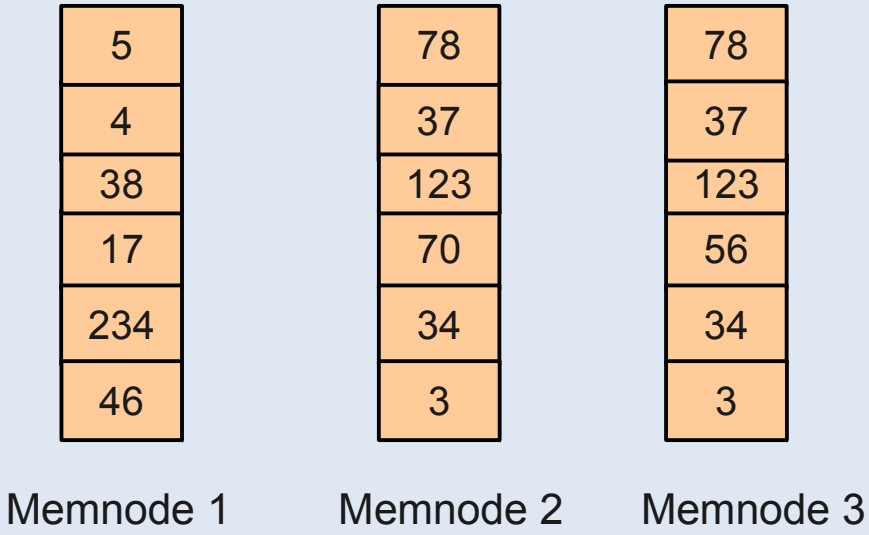
# Minitransactions (example)

- API:

```
Class Minitransaction{
  void cmp(memid,addr,len,data);
  void read(memid,addr,len,buf);
  void write(memid,addr,len,data);
  int exec_and_commit();
}
```

- Example:

```
t = new Minitransaction();
t.cmp(2,3,1,70);
t.write(1,2,1,45);
t.write(3,4,2,37,848);
status = t.exec_and_commit();
```



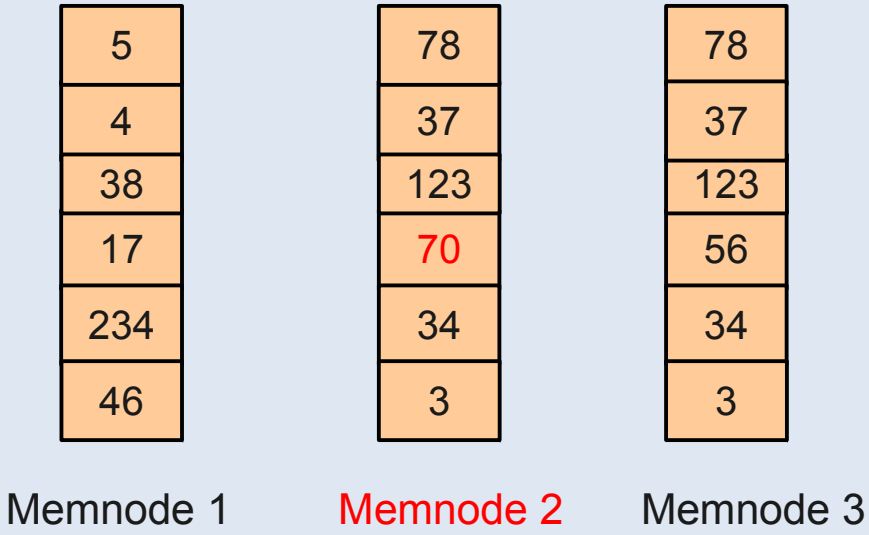
# Minitransactions (example)

- API:

```
Class Minitransaction{
  void cmp(memid,addr,len,data);
  void read(memid,addr,len,buf);
  void write(memid,addr,len,data);
  int exec_and_commit();
}
```

- Example:

```
t = new Minitransaction();
t.cmp(2,3,1,70);
t.write(1,2,1,45);
t.write(3,4,2,37,848);
status = t.exec_and_commit();
```



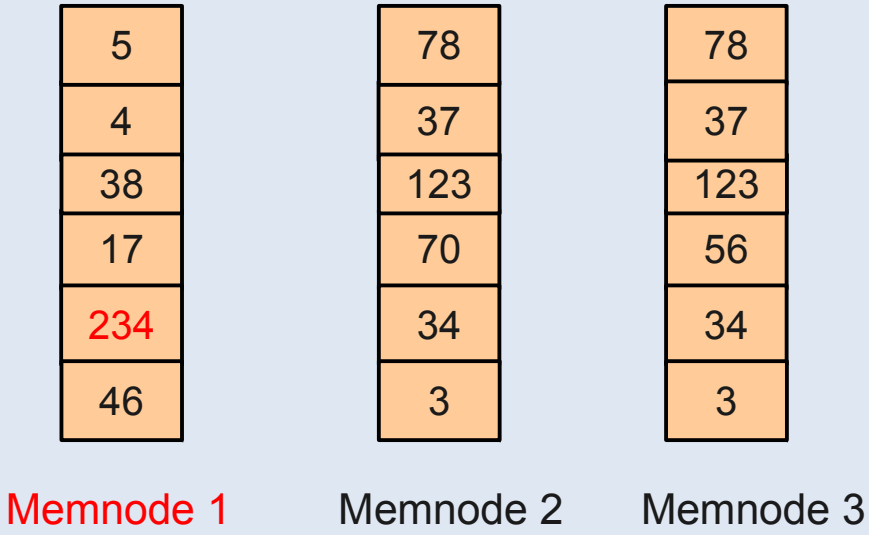
# Minitransactions (example)

- API:

```
Class Minitransaction{
  void cmp(memid,addr,len,data);
  void read(memid,addr,len,buf);
  void write(memid,addr,len,data);
  int exec_and_commit();
}
```

- Example:

```
t = new Minitransaction();
t.cmp(2,3,1,70);
t.write(1,2,1,45);
t.write(3,4,2,37,848);
status = t.exec_and_commit();
```



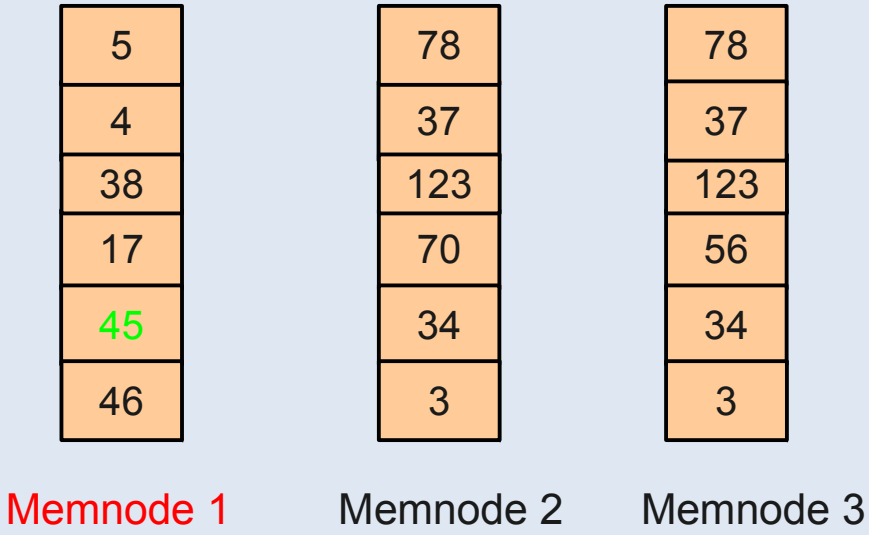
# Minitransactions (example)

- API:

```
Class Minitransaction{
  void cmp(memid,addr,len,data);
  void read(memid,addr,len,buf);
  void write(memid,addr,len,data);
  int exec_and_commit();
}
```

- Example:

```
t = new Minitransaction();
t.cmp(2,3,1,70);
t.write(1,2,1,45);
t.write(3,4,2,37,848);
status = t.exec_and_commit();
```



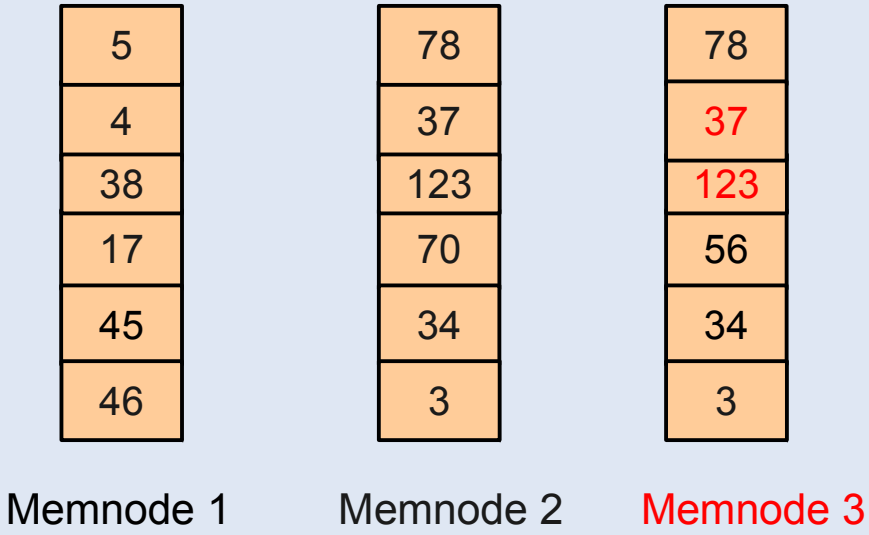
# Minitransactions (example)

- API:

```
Class Minitransaction{
  void cmp(memid,addr,len,data);
  void read(memid,addr,len,buf);
  void write(memid,addr,len,data);
  int exec_and_commit();
}
```

- Example:

```
t = new Minitransaction();
t.cmp(2,3,1,70);
t.write(1,2,1,45);
t.write(3,4,2,37,848);
status = t.exec_and_commit();
```



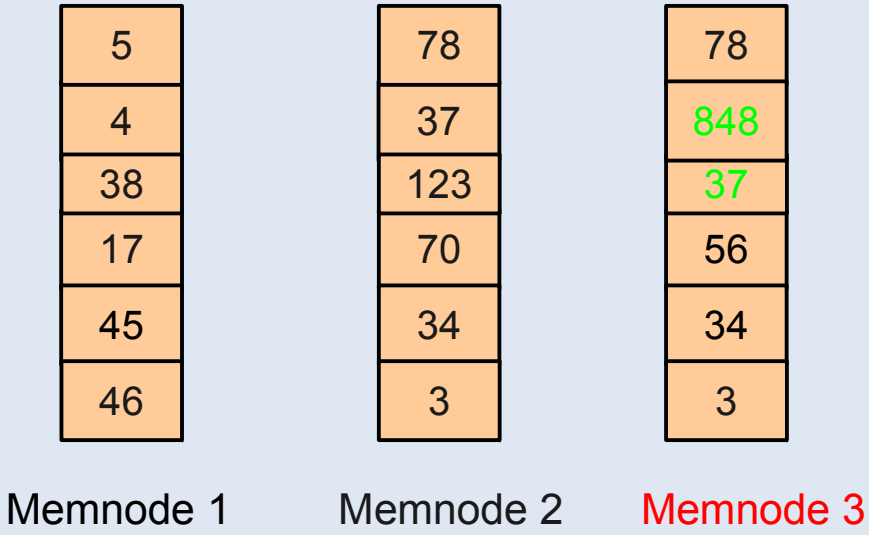
# Minitransactions (example)

- API:

```
Class Minitransaction{
  void cmp(memid,addr,len,data);
  void read(memid,addr,len,buf);
  void write(memid,addr,len,data);
  int exec_and_commit();
}
```

- Example:

```
t = new Minitransaction();
t.cmp(2,3,1,70);
t.write(1,2,1,45);
t.write(3,4,2,37,848);
status = t.exec_and_commit();
```



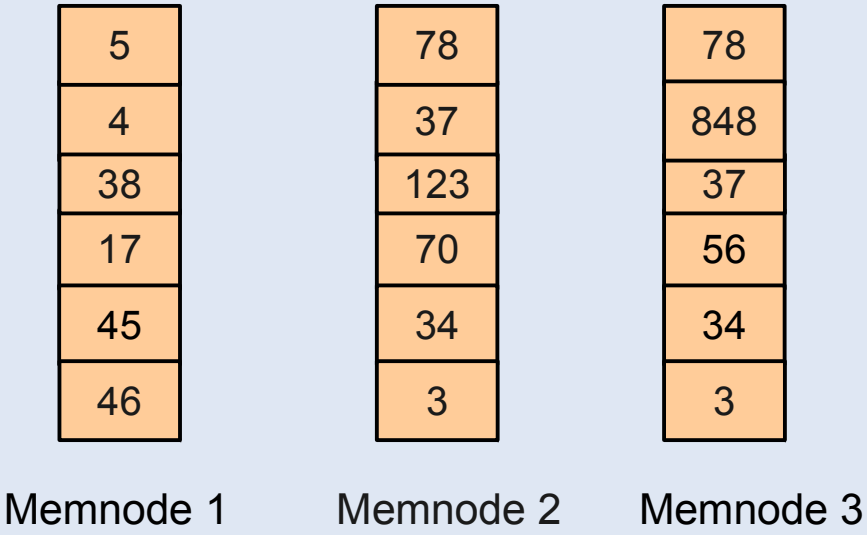
# Minitransactions (example)

- API:

```
Class Minitransaction{
  void cmp(memid,addr,len,data);
  void read(memid,addr,len,buf);
  void write(memid,addr,len,data);
  int exec_and_commit();
}
```

- Example:

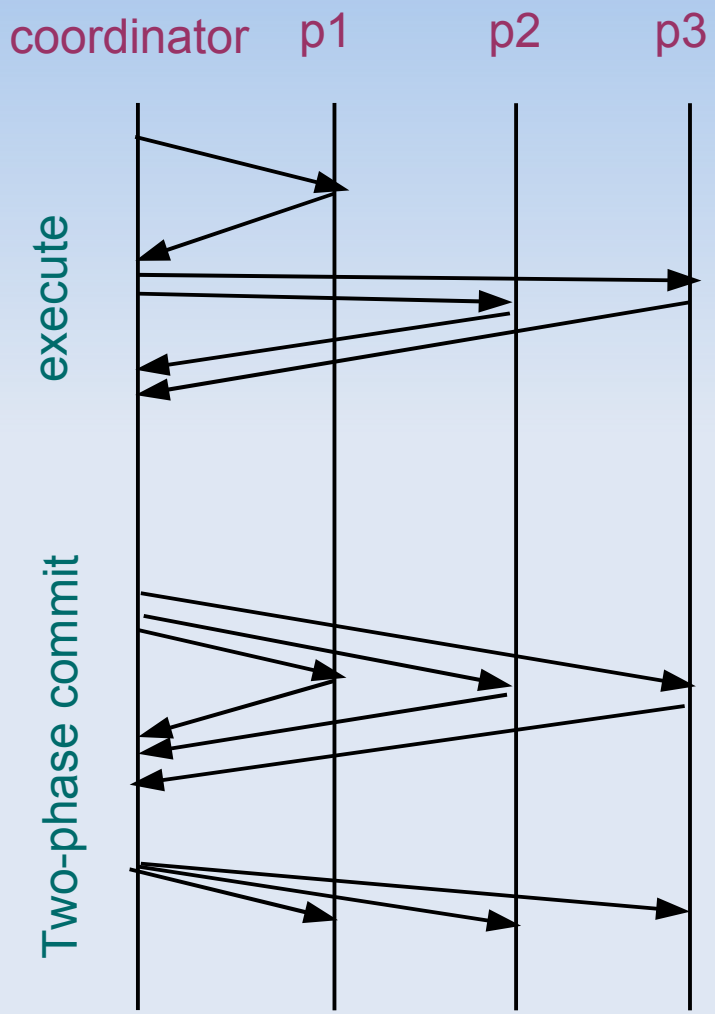
```
t = new Minitransaction();
t.cmp(2,3,1,70);
t.write(1,2,1,45);
t.write(3,4,2,37,848);
status = t.exec_and_commit();
```



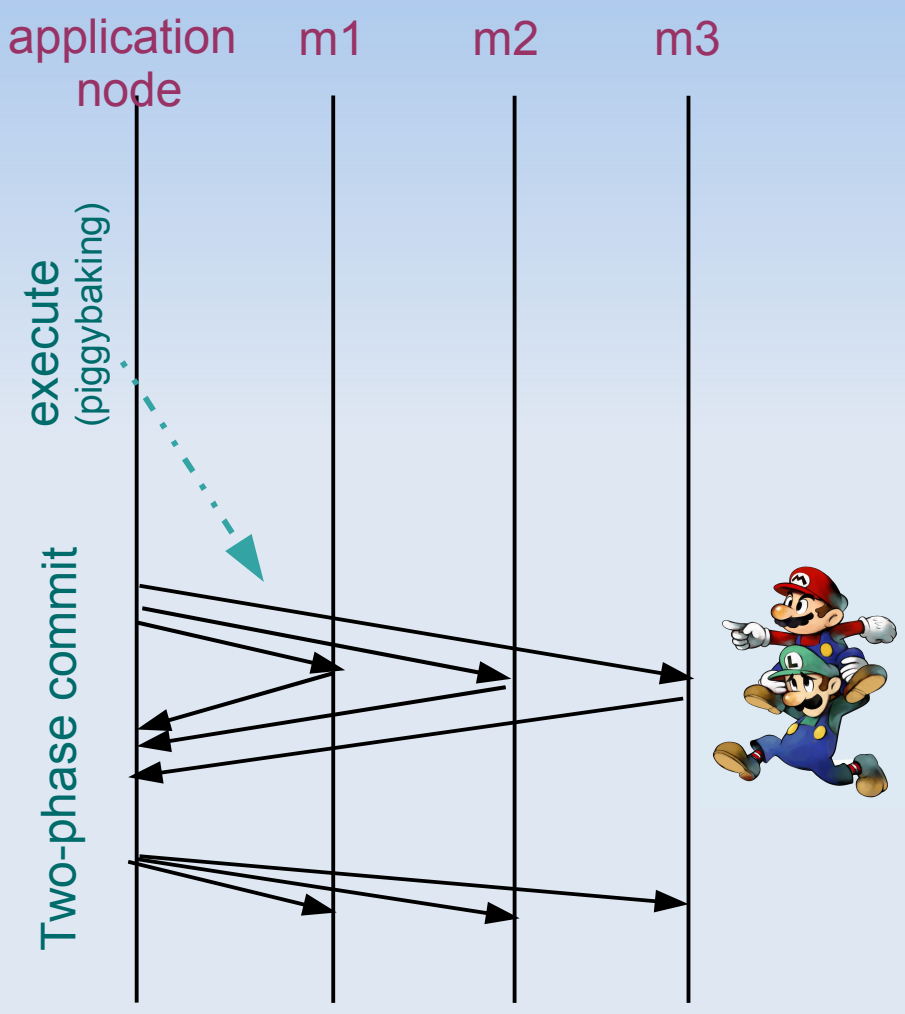
# Minitransactions

- Balance between:
  - Functionality (Power): powerful enough, general-purpose, easy to use
  - Efficiency: can be executed and commited efficiently, with a small number of network round-trips

# Minitransaction Efficiency




Traditional transactions




Sinfonia minitransactions

# Outline

- Sinfonia Structure
- Minitransactions
- Design Choices
- Two Applications
- Evaluation
- Conclusion
- Questions & Discussions

# Caching and Load Balancing

- Caching
  - Sinfonia does not cache data at application nodes
  - Caching is left to application nodes
- Load balancing
  - Sinfonia does not balance data across memory nodes
  - Load balancing is left to application nodes
  - Sinfonia provides per-memory-node load information

# Fault Tolerance

- Mechanisms for fault tolerance:
  - Disk image
  - Logging
  - Replication
  - Backup
- Trade off between fault tolerance and amount of resources

# Sinfonia Modes

| Mode | RAM | RAM-REPL | LOG | LOG-REPL | NVRAM | NVRAM-REPL |
|---|---|---|---|---|---|---|
| Description | disk image off<br>log off<br>replication off<br>backup optional | disk image off<br>log off<br>replication on<br>backup optional | disk image on<br>log on disk<br>replication off<br>backup optional | disk image on<br>log on disk<br>replication on<br>backup optional | disk image on<br>log on nvram<br>replication off<br>backup optional | disk image on<br>log on nvram<br>replication on<br>backup optional |
| Memnode resources | 1 host | 2 hosts | 1 host, 2 disks[c] | 2 hosts, 4 disks[d] | 1 host, 1 disk, nvram | 2 hosts, 2 disks, nvram[d] |
| Memnode space | RAM available | RAM available | disk size | disk size | disk size | disk size |
| Fault tolerance[a] | •app crash | •app crash<br>•few memnode crashes | •app crash<br>•all memnode crashes but with downtime | •app crash<br>•few memnode crashes<br>•all memnode crashes but with downtime | •app crash<br>•all memnode crashes but with downtime | •app crash<br>•few memnode crashes<br>•all memnode crashes but with downtime |
| Performance[b] | first | second | third | fourth | first | second |

# Sinfonia Modes

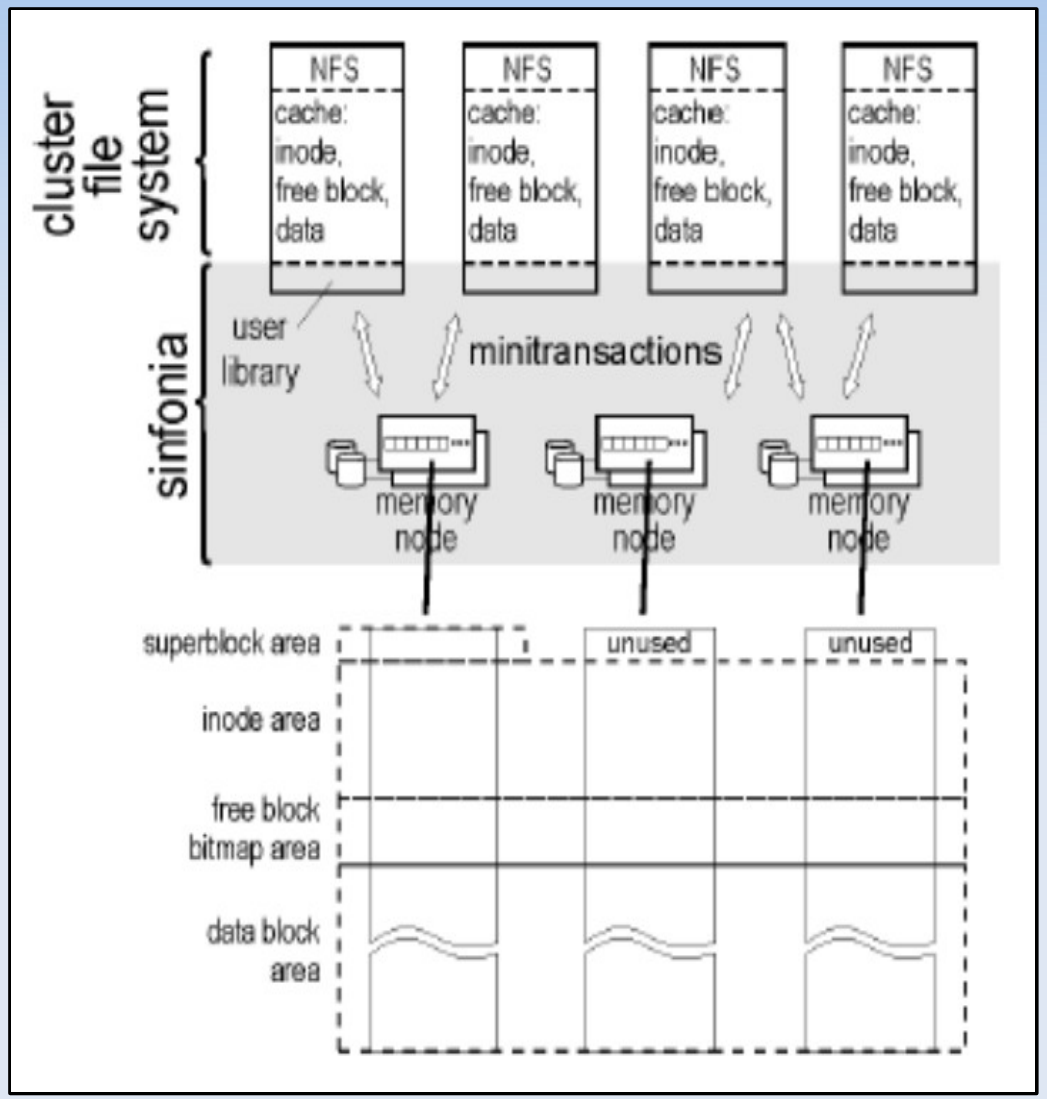
| Mode | RAM | RAM-REPL | LOG | LOG-REPL | NVRAM | NVRAM-REPL |
|---|---|---|---|---|---|---|
| Description | disk image off<br>log off<br>replication off<br>backup optional | disk image off<br>log off<br>replication on<br>backup optional | disk image on<br>log on disk<br>replication off<br>backup optional | disk image on<br>log on disk<br>replication on<br>backup optional | disk image on<br>log on nvram<br>replication off<br>backup optional | disk image on<br>log on nvram<br>replication on<br>backup optional |
| Memnode resources | 1 host | 2 hosts | 1 host, 2 disks[c] | 2 hosts, 4 disks[d] | 1 host, 1 disk, nvram | 2 hosts, 2 disks, nvram[d] |
| Memnode space | RAM available | RAM available | disk size | disk size | disk size | disk size |
| Fault tolerance[a] | •app crash | •app crash<br>•few memnode crashes | •app crash<br>•all memnode crashes but with downtime | •app crash<br>•few memnode crashes<br>•all memnode crashes but with downtime | •app crash<br>•all memnode crashes but with downtime | •app crash<br>•few memnode crashes<br>•all memnode crashes but with downtime |
| Performance[b] | first | second | third | fourth | first | second |

# Outline

- Sinfonia Structure
- Minitransactions
- Design Choices
- Two Applications
- Evaluation
- Conclusion
- Questions & Discussions

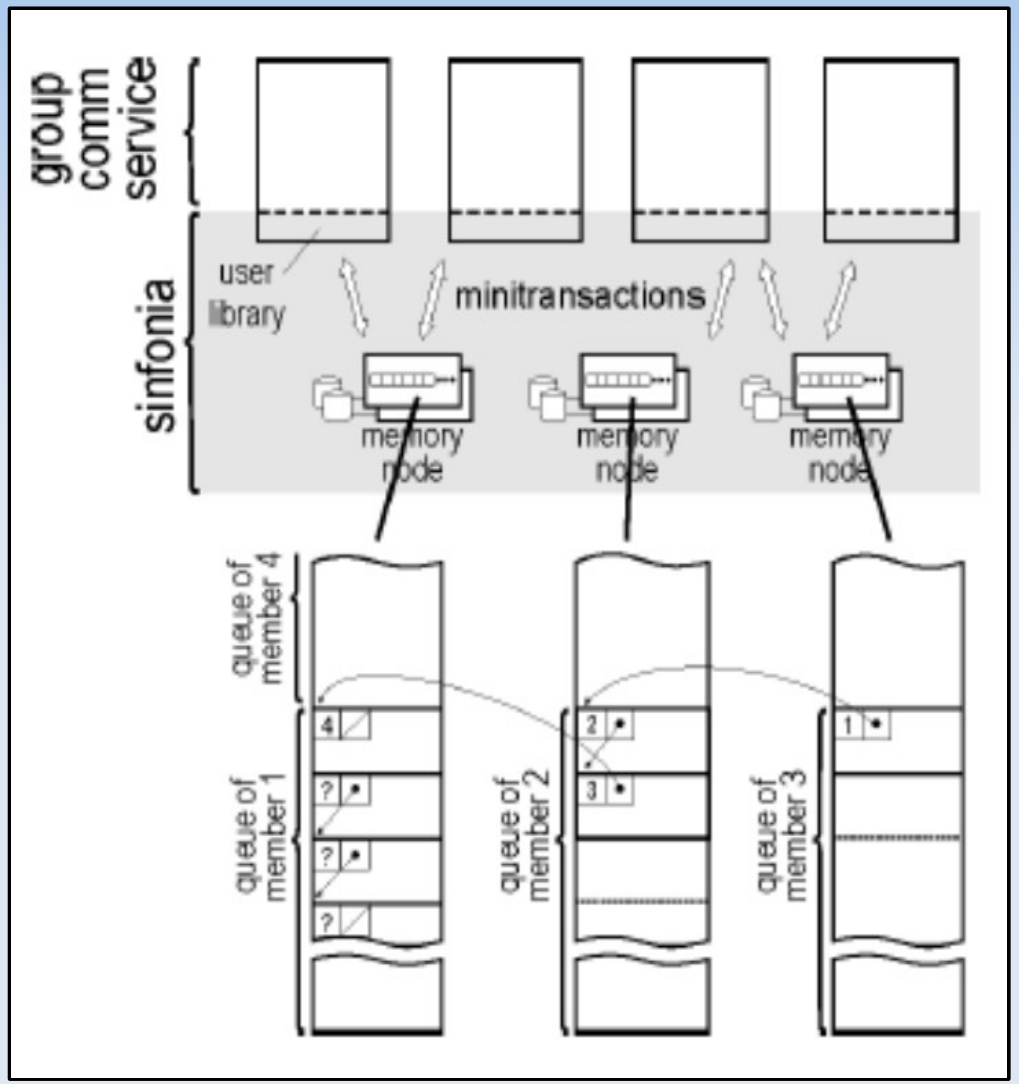
# Application: Cluster File System

- SinfoniaFS
  - Fault tolerant
  - Scalable
- Exports NFS v2
  - Each NFS function: a single minitransaction.
- For each function:
  - Validate cache
  - Modify data



# Application: Group Communication Service

- GCS: chat room
  - Join and leave
  - Broadcast msgs
- SinfoniaGCS
  - Messages stored in memory nodes
  - Private queue for each member
  - Global list



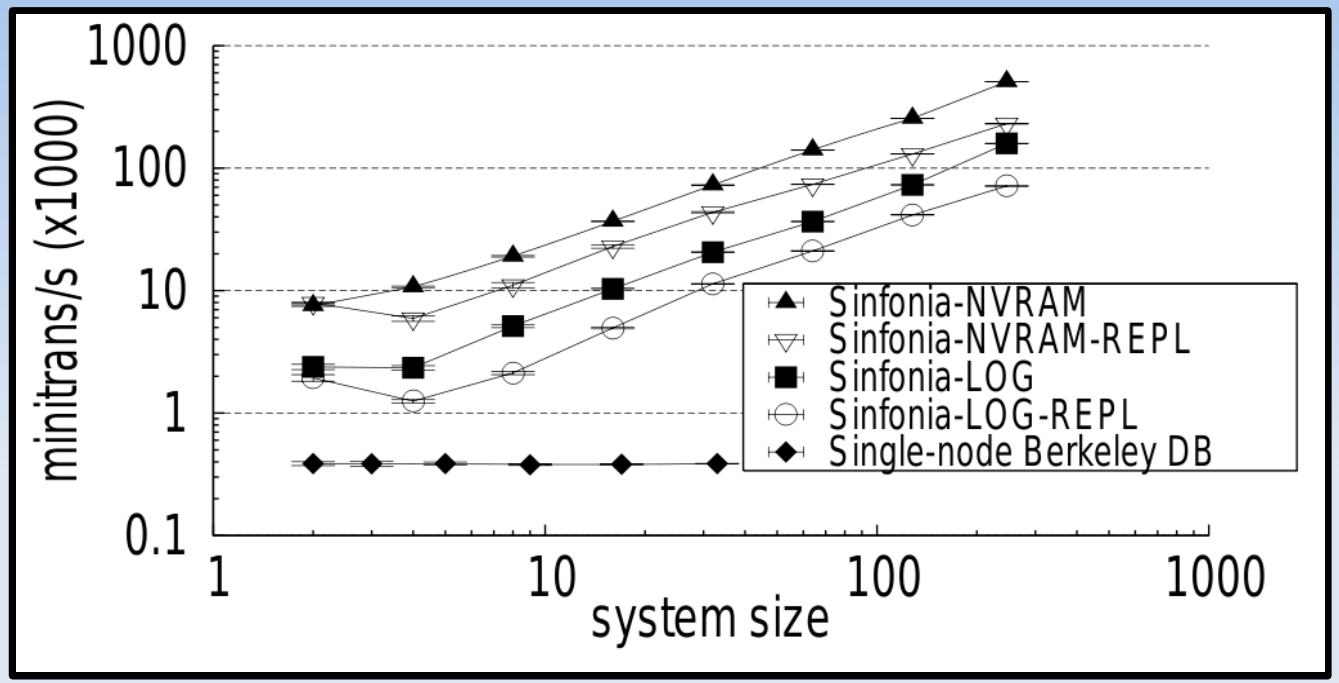
# Outline

- Sinfonia Structure
- Minitransactions
- Design Choices
- Two Applications
- Evaluation
- Conclusion
- Questions & Discussions

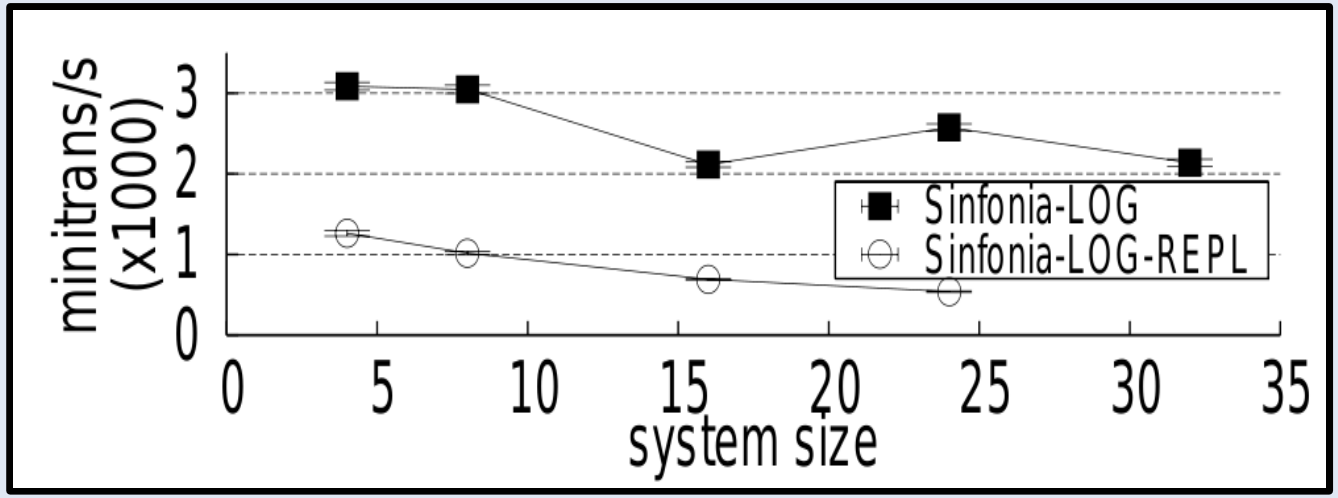
# Evaluation: Ease of Use

| | SinfoniaFS | LinuxNFS | SinfoniaGCS | Spread Toolkit |
|---|---|---|---|---|
| **lines of code** | 3,855 (C++) | 5,900 (C) | 2,492 (C++) | 22,148 (C) |
| **develop time** | 1 month | unknown | 2 months | years |
| **major versions** | 1 | 2 | 1 | 4 |

# Evaluation: Scalability

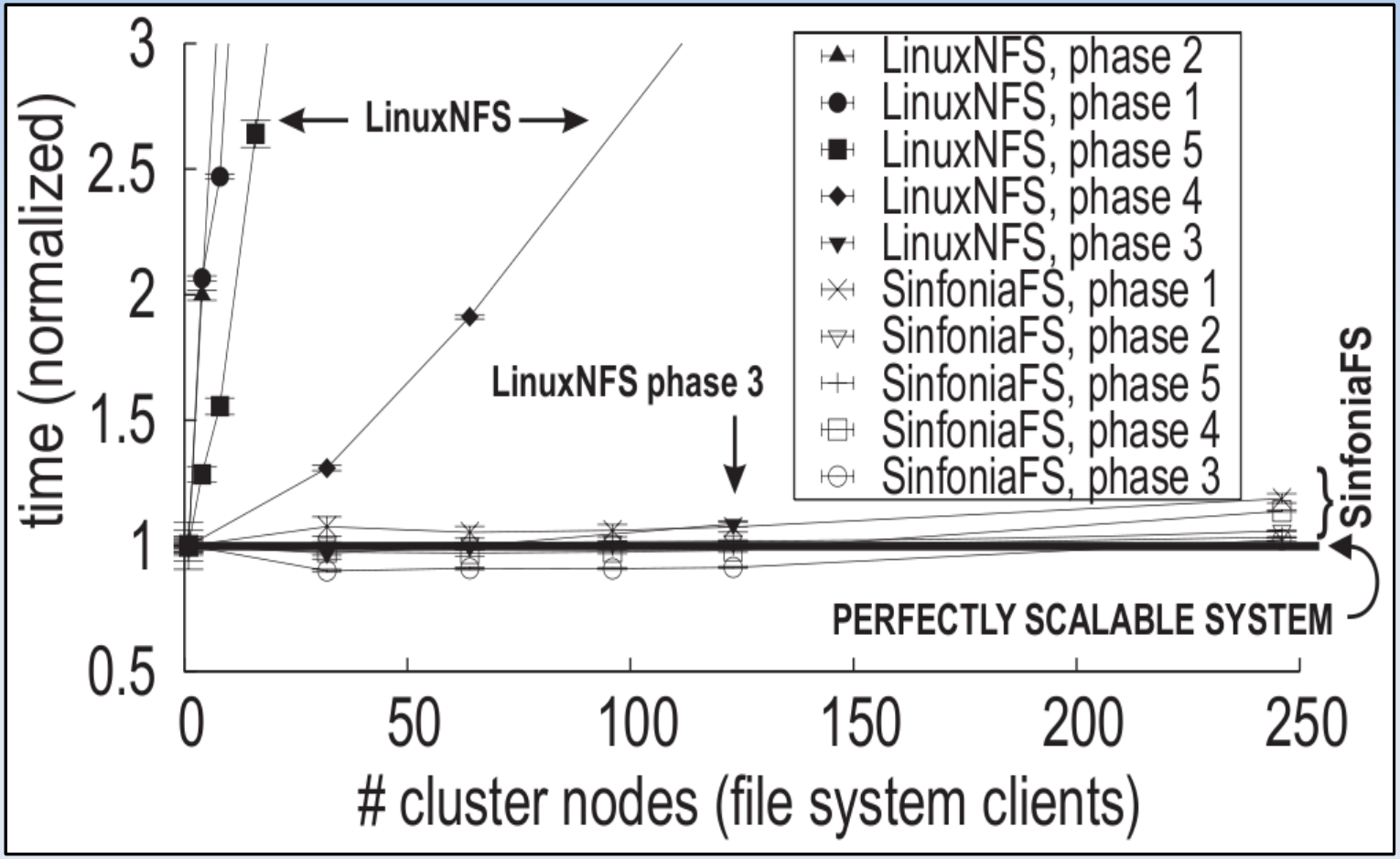



spread= 2
**scalable**

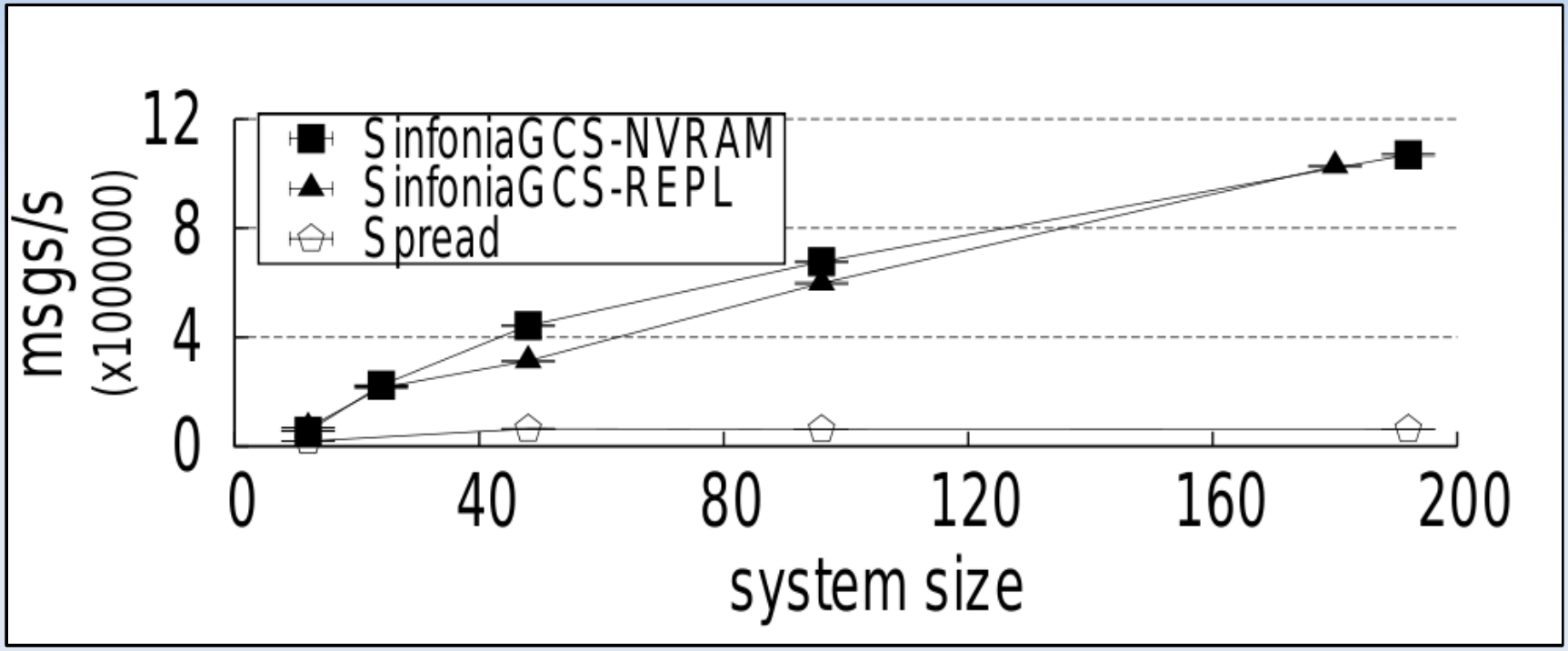



spread= # of memory node
**not scalable**

# Evaluation: SinfoniaFS



# Evaluation: SinfoniaGCS



# Conclusion

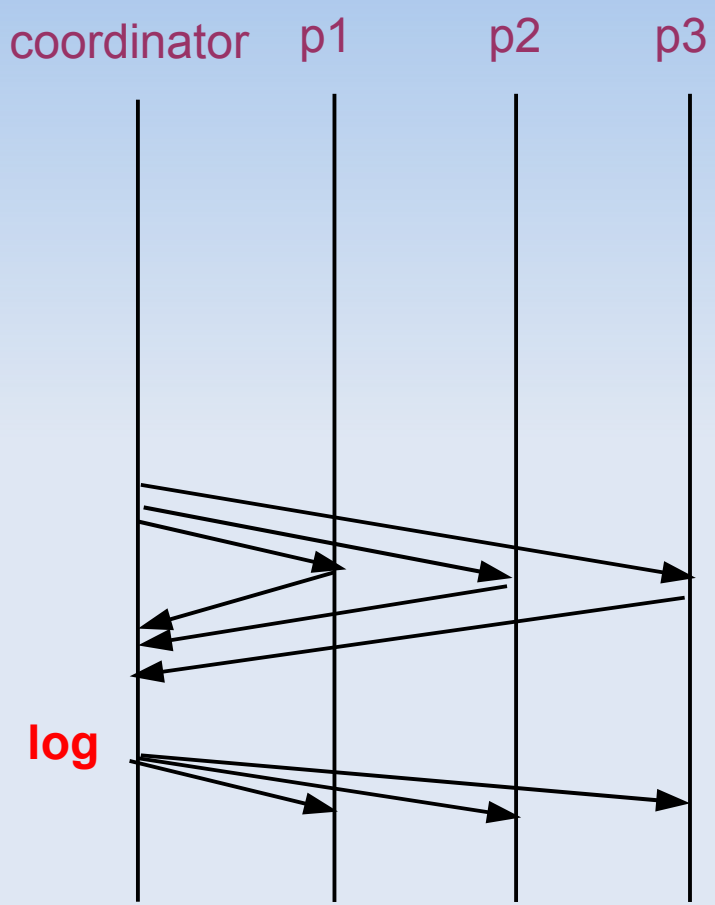
- Sinfonia: a service for building scalabe distributed systems
- Protocol design $\longrightarrow$ data structure design
- A sequence of minitransactions over unstructured data
- Effective in building infrastructure applications
- Extensions

# Thanks

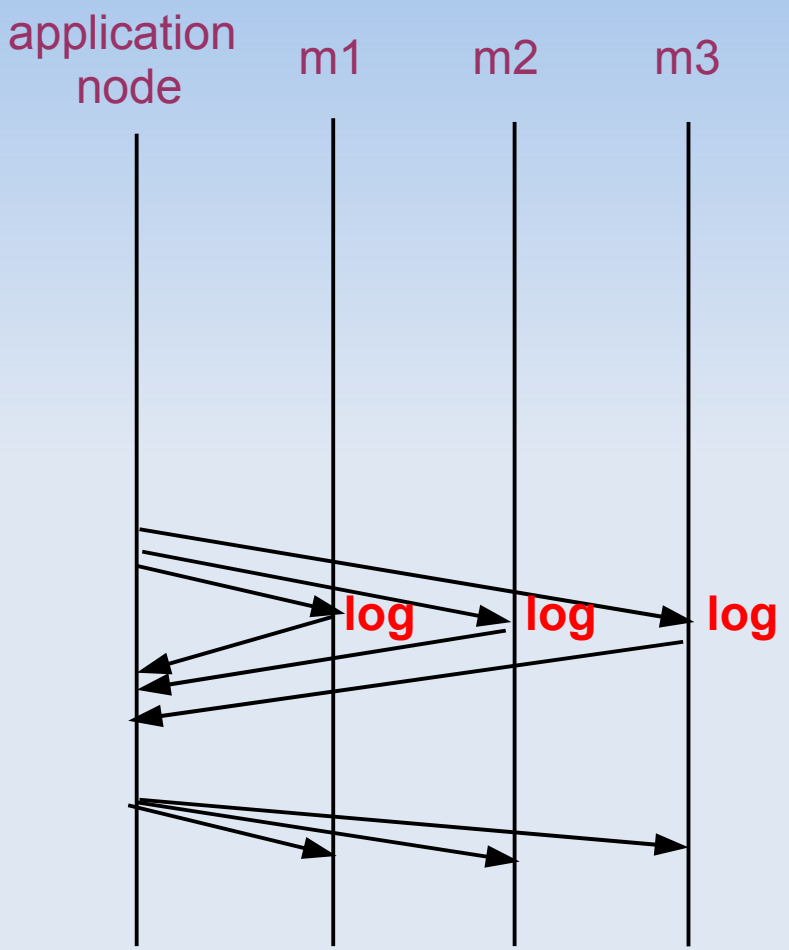
# Coordinator Crash

- Traditional 2PC blocks on coordinator crash
  - Not desirable in Sinfonia: Sinfonia does not have control on coordinators
- Traditional solution: 3PC
- Sinfonia Solution: modified 2PC+recovery coordinator

# Coordinator Crash

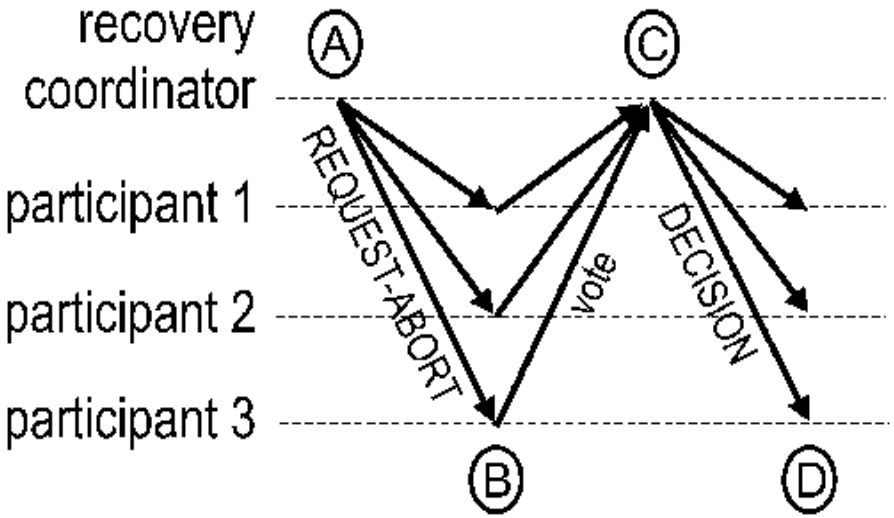



Traditional 2PC




Sinfonia 2PC

# Coordinator Crash




Ⓐ recovery triggered

Ⓑ if not yet chosen vote, choose abort vote otherwise keep previous vote

Ⓒ choose decision

Ⓓ if committing then apply *write items* release locks held for minitransaction