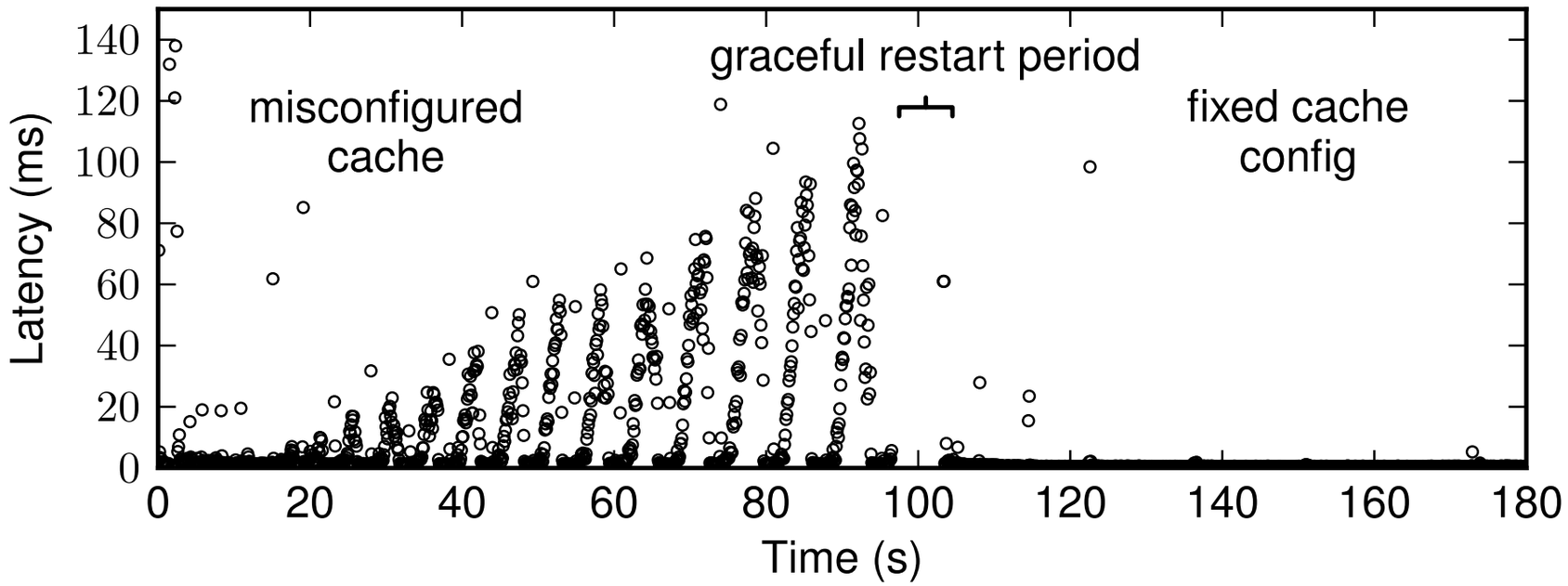
A Framework for the Dynamic Evolution of Highly-Available Dataflow Programs

Sebastian Ertel
Systems Engineering Group
TU Dresden, Germany

Pascal Felber
Institut d'informatique
Université de Neuchâtel, Switzerland

Middleware 2014

Jetty's Graceful Restart



- Downtimes ~8-10s ➪ Request latency ~1-2ms.

Smith, E.K.; Hicks, M.; Foster, J.S., "Towards standardized benchmarks for Dynamic Software Updating systems," HotSWUp'12

Hot Swapping

- Goal: Change a program dynamically, i.e. online.
- Problem: Preserving program correctness.
- Challenges:
  - *Consistency Problem* ➪ State quiescence (who and when).
  - Mutual references ➪ *Non-blocking* update coordination.
  - Referential transparency.
  - State transfer.
  - Scalability.
  - Location transparency.

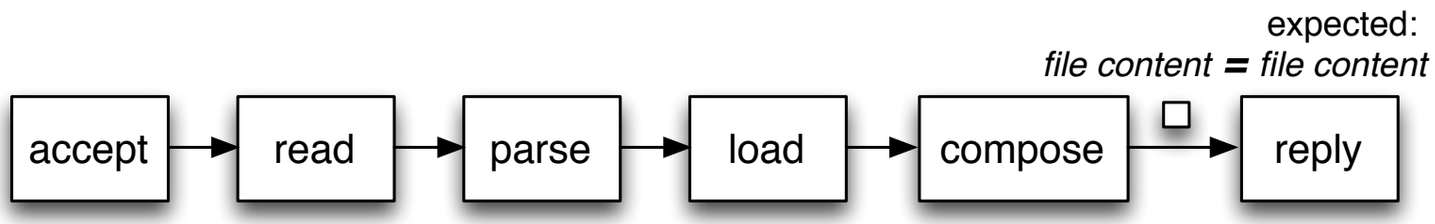
Feng, N. et.al.., "Dynamic evolution of network management software by software hot-swapping," IFIP/IEEE IM 2001.

Iulian Neamtiu and Michael Hicks. 2009. Safe and timely updates to multi-threaded programs. In PLDI '09.

Hayden, C.M.; Hicks, M.; et.al. A study of dynamic software update quiescence for multithreaded programs, HotSWUp'12

Hot Swapping

- Goal: Change a program dynamically, i.e. online.
- Problem: Preserving program correctness.
- Challenges:
  - *Consistency Problem* ➪ State quiescence (who and when).
  - Mutual references ➪ *Non-blocking* update coordination.



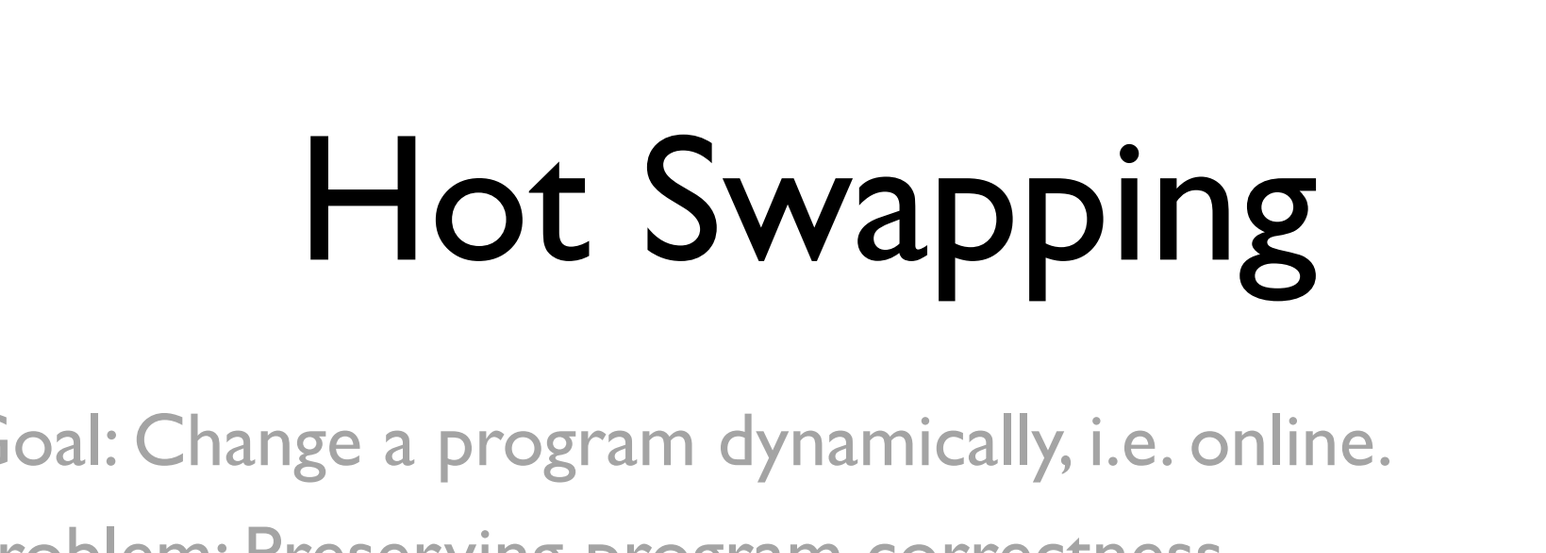
Feng, N. et.al.., "Dynamic evolution of network management software by software hot-swapping," IFIP/IEEE IM 2001.

Iulian Neamtiu and Michael Hicks. 2009. Safe and timely updates to multi-threaded programs. In PLDI '09.

Hayden, C.M.; Hicks, M.; et.al. A study of dynamic software update quiescence for multithreaded programs, HotSWUp'12

Hot Swapping

- Goal: Change a program dynamically, i.e. online.
- Problem: Preserving program correctness.
- Challenges:
  - *Consistency Problem* ➪ State quiescence (who and when).
  - Mutual references ➪ *Non-blocking* update coordination.



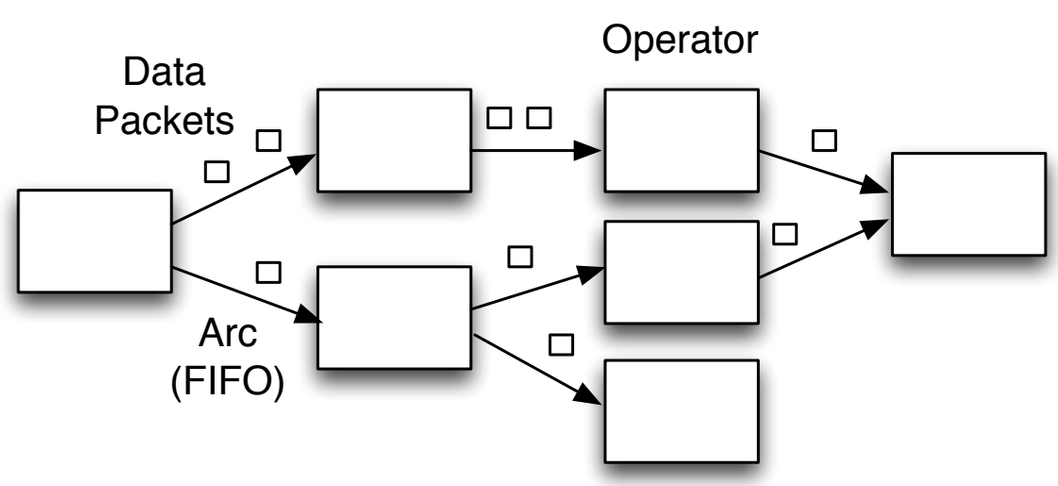
Feng, N. et.al.., "Dynamic evolution of network management software by software hot-swapping," IFIP/IEEE IM 2001.

Iulian Neamtiu and Michael Hicks. 2009. Safe and timely updates to multi-threaded programs. In PLDI '09.

Hayden, C.M.; Hicks, M.; et.al. A study of dynamic software update quiescence for multithreaded programs, HotSWUp'12

Dataflow in a Nutshell

- Widely adopted execution model for parallel processing.
- Building blocks: (FIFO) arcs, operators, dataflow graph.
- (Classic) Dataflow ➪ No operator state.
- FBP (flow-based programming) ➪ Operator state allowed!



DAG - Directed (Acyclic) Data Flow Graph

J. B. Dennis. Data flow supercomputers. Computer, 13(11):48–56, Nov. 1980.

J. P. Morrison. Flow-Based Programming. Nostrand Reinhold, 1994

Micah Beck, Richard Johnson, and Keshav Pingali. 1991. From control flow to dataflow. J. Parallel Distrib. Comput.

Hot Swapping

- Goal: Change a program dynamically, i.e. online.
- Problem: Preserving program correctness.
- Challenges:
  - *Consistency Problem* ➪ State quiescence (who and when). ✓
  - Mutual references ➪ *Non-blocking* update coordination.
  - Referential transparency.
  - State transfer.
  - Scalability.
  - Location transparency.

Feng, N. et.al.., "Dynamic evolution of network management software by software hot-swapping," IFIP/IEEE IM 2001.

Iulian Neamtiu and Michael Hicks. 2009. Safe and timely updates to multi-threaded programs. In PLDI '09.

Hayden, C.M.; Hicks, M.; et.al. A study of dynamic software update quiescence for multithreaded programs, HotSWUp'12

Update Appointment

- Dataflow building blocks: arcs, operators, dataflow graph.
- *No assumptions on execution environment.*
- Key insight:
  - That's a distributed system!
  - We know how to reason about time!

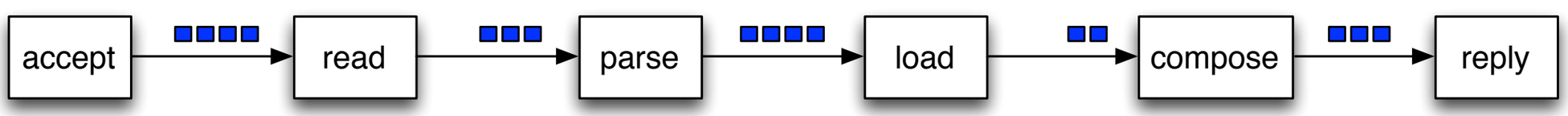
➪ *Introduction of a solid notion of time for (live) updates.*

K. Mani Chandy and Leslie Lamport. 1985. Distributed snapshots: determining global states of distributed systems. ACM Trans. Comput. Syst. 3, 1 (February 1985), 63-75.

Update Appointment

- Dataflow building blocks: arcs, operators, dataflow graph.
- *No assumptions on execution environment.*
- Key insight:
  - That's a distributed system!
  - We know how to reason about time!

➪ *Introduction of a solid notion of time for (live) updates.*

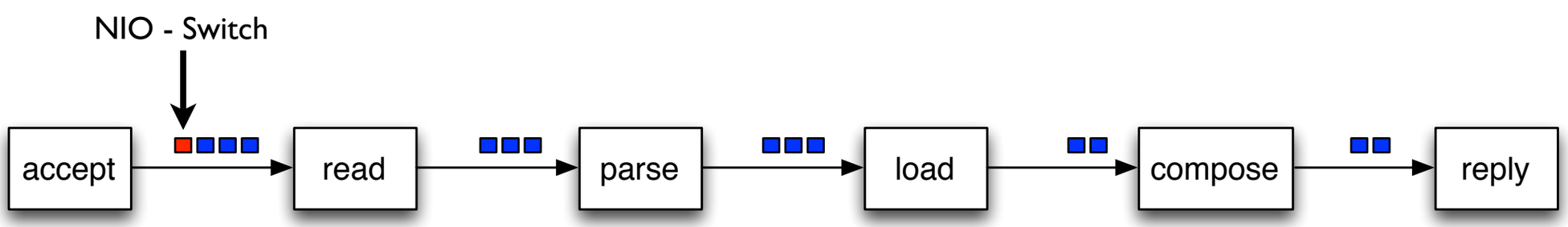


K. Mani Chandy and Leslie Lamport. 1985. Distributed snapshots: determining global states of distributed systems. ACM Trans. Comput. Syst. 3, 1 (February 1985), 63-75.

Update Appointment

- Dataflow building blocks: arcs, operators, dataflow graph.
- *No assumptions on execution environment.*
- Key insight:
  - That's a distributed system!
  - We know how to reason about time!

➪ *Introduction of a solid notion of time for (live) updates.*

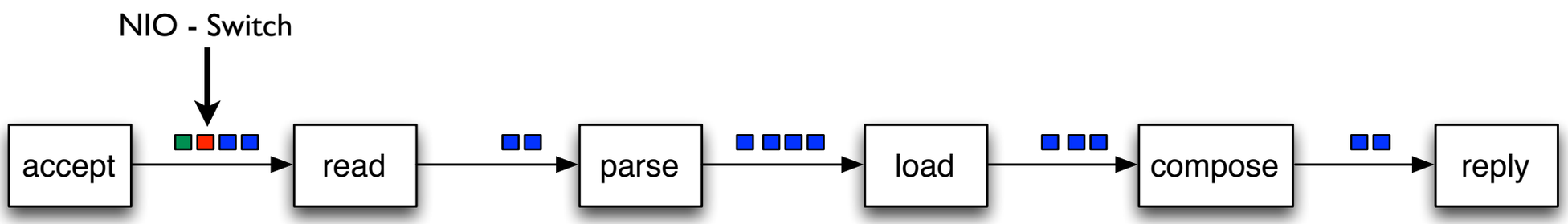


K. Mani Chandy and Leslie Lamport. 1985. Distributed snapshots: determining global states of distributed systems. ACM Trans. Comput. Syst. 3, 1 (February 1985), 63-75.

Update Appointment

- Dataflow building blocks: arcs, operators, dataflow graph.
- *No assumptions on execution environment.*
- Key insight:
  - That's a distributed system!
  - We know how to reason about time!

➪ *Introduction of a solid notion of time for (live) updates.*

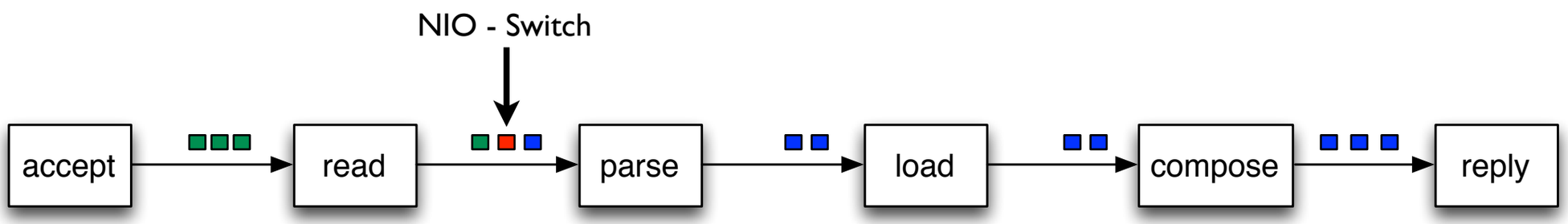


K. Mani Chandy and Leslie Lamport. 1985. Distributed snapshots: determining global states of distributed systems. ACM Trans. Comput. Syst. 3, 1 (February 1985), 63-75.

Update Appointment

- Dataflow building blocks: arcs, operators, dataflow graph.
- *No assumptions on execution environment.*
- Key insight:
  - That's a distributed system!
  - We know how to reason about time!

➪ *Introduction of a solid notion of time for (live) updates.*

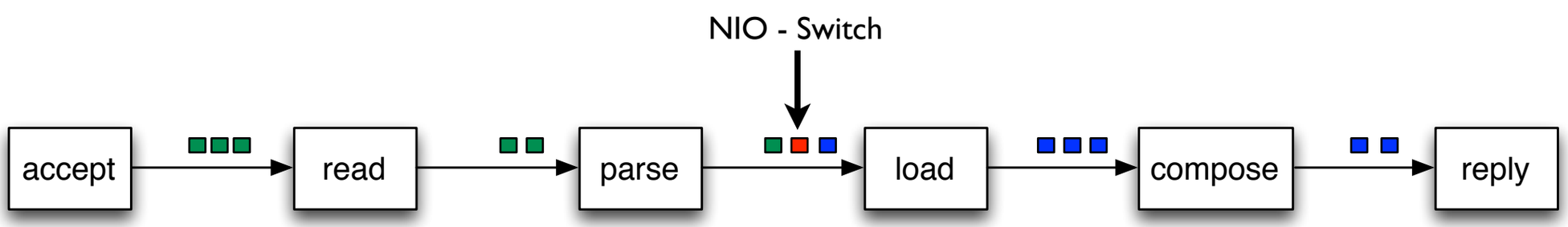


K. Mani Chandy and Leslie Lamport. 1985. Distributed snapshots: determining global states of distributed systems. ACM Trans. Comput. Syst. 3, 1 (February 1985), 63-75.

Update Appointment

- Dataflow building blocks: arcs, operators, dataflow graph.
- *No assumptions on execution environment.*
- Key insight:
  - That's a distributed system!
  - We know how to reason about time!

➪ *Introduction of a solid notion of time for (live) updates.*

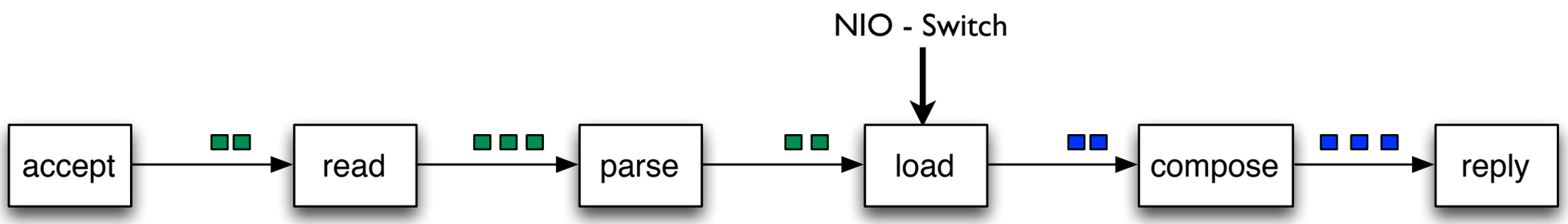


K. Mani Chandy and Leslie Lamport. 1985. Distributed snapshots: determining global states of distributed systems. ACM Trans. Comput. Syst. 3, 1 (February 1985), 63-75.

Update Appointment

- Dataflow building blocks: arcs, operators, dataflow graph.
- *No assumptions on execution environment.*
- Key insight:
  - That's a distributed system!
  - We know how to reason about time!

➪ *Introduction of a solid notion of time for (live) updates.*

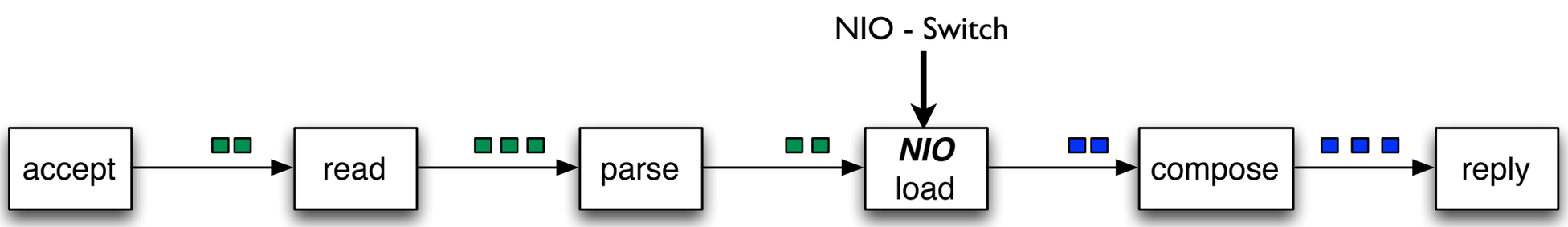


K. Mani Chandy and Leslie Lamport. 1985. Distributed snapshots: determining global states of distributed systems. ACM Trans. Comput. Syst. 3, 1 (February 1985), 63-75.

Update Appointment

- Dataflow building blocks: arcs, operators, dataflow graph.
- *No assumptions on execution environment.*
- Key insight:
  - That's a distributed system!
  - We know how to reason about time!

➪ *Introduction of a solid notion of time for (live) updates.*



K. Mani Chandy and Leslie Lamport. 1985. Distributed snapshots: determining global states of distributed systems. ACM Trans. Comput. Syst. 3, 1 (February 1985), 63-75.

Hot Swapping

- Goal: Change a program dynamically, i.e. online.
- Problem: Preserving program correctness.
- Challenges:
  - *Consistency Problem* ➪ State quiescence (who ✓ and when ✓).
  - Mutual references ➪ *Non-blocking* update coordination.
  - Referential transparency.
  - State transfer.
  - Scalability.
  - Location transparency.

Feng, N. et.al.., "Dynamic evolution of network management software by software hot-swapping," IFIP/IEEE IM 2001.

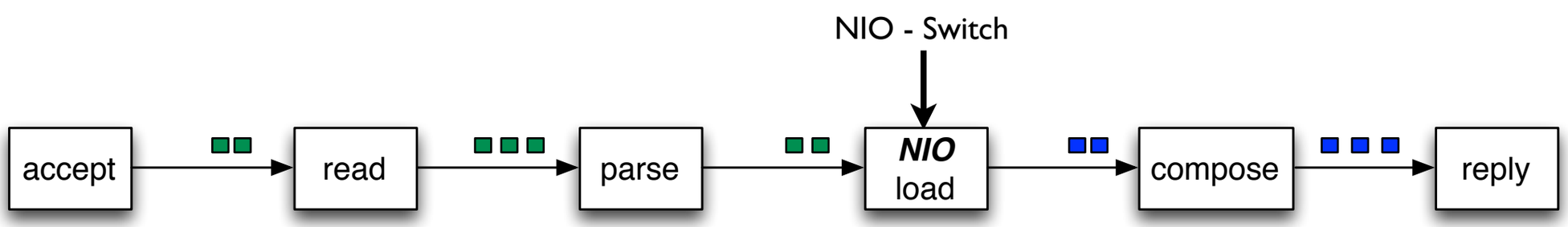
Iulian Neamtiu and Michael Hicks. 2009. Safe and timely updates to multi-threaded programs. In PLDI '09.

Hayden, C.M.; Hicks, M.; et.al. A study of dynamic software update quiescence for multithreaded programs, HotSWUp'12

Update Coordination

- Dataflow building blocks: arcs, operators, dataflow graph.
- *No assumptions on execution environment.*
- Key insight:
  - That's a distributed system!
  - We know how to reason about time!

➪ *Introduction of a solid notion of time for (live) updates.*

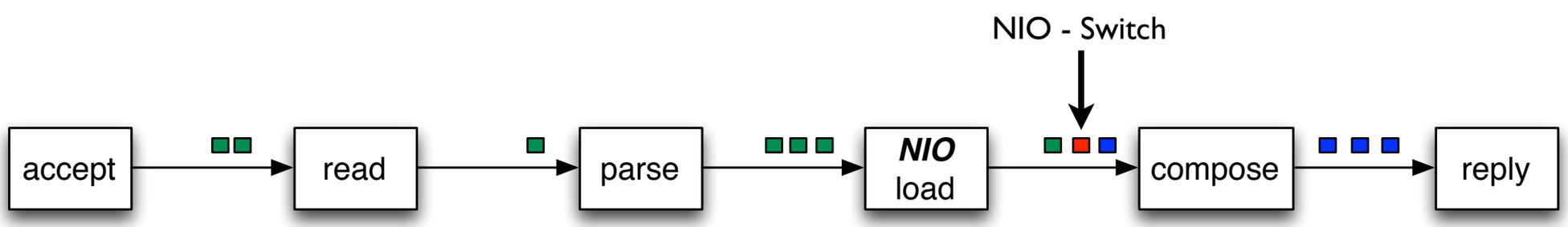


K. Mani Chandy and Leslie Lamport. 1985. Distributed snapshots: determining global states of distributed systems. ACM Trans. Comput. Syst. 3, 1 (February 1985), 63-75.

Update Coordination

- Dataflow building blocks: arcs, operators, dataflow graph.
- *No assumptions on execution environment.*
- Key insight:
  - That's a distributed system!
  - We know how to reason about time!

➪ *Introduction of a solid notion of time for (live) updates.*

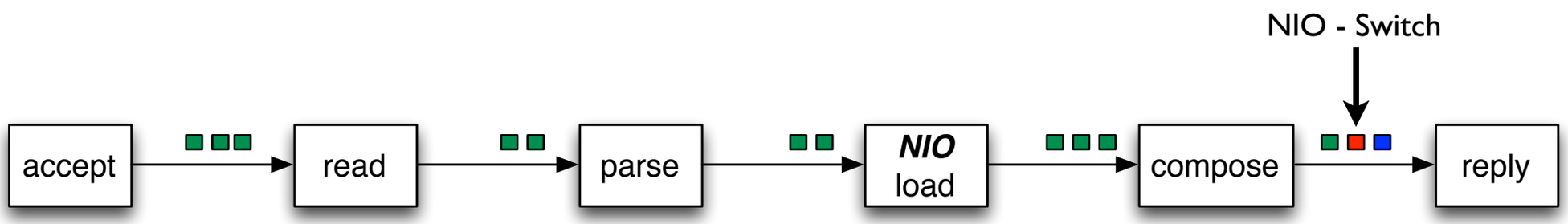


K. Mani Chandy and Leslie Lamport. 1985. Distributed snapshots: determining global states of distributed systems. ACM Trans. Comput. Syst. 3, 1 (February 1985), 63-75.

Update Coordination

- Dataflow building blocks: arcs, operators, dataflow graph.
- *No assumptions on execution environment.*
- Key insight:
  - That's a distributed system!
  - We know how to reason about time!

➪ *Introduction of a solid notion of time for (live) updates.*

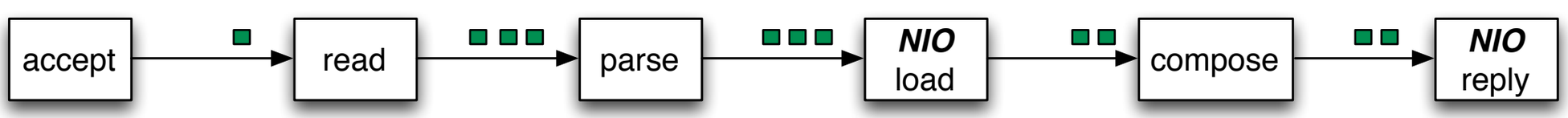


K. Mani Chandy and Leslie Lamport. 1985. Distributed snapshots: determining global states of distributed systems. ACM Trans. Comput. Syst. 3, 1 (February 1985), 63-75.

Update Coordination

- Dataflow building blocks: arcs, operators, dataflow graph.
- *No assumptions on execution environment.*
- Key insight:
  - That's a distributed system!
  - We know how to reason about time!

➪ *Introduction of a solid notion of time for (live) updates.*



K. Mani Chandy and Leslie Lamport. 1985. Distributed snapshots: determining global states of distributed systems. ACM Trans. Comput. Syst. 3, 1 (February 1985), 63-75.

Hot Swapping

- Goal: Change a program dynamically, i.e. online.
- Problem: Preserving program correctness.
- Challenges:
  - *Consistency Problem* ➪ State quiescence (who and when). ✓ ✓
  - Mutual references ➪ *Non-blocking* update coordination. ✓
  - Referential transparency.
  - State transfer.
  - Scalability.
  - Location transparency.

Feng, N. et.al.., "Dynamic evolution of network management software by software hot-swapping," IFIP/IEEE IM 2001.

Iulian Neamtiu and Michael Hicks. 2009. Safe and timely updates to multi-threaded programs. In PLDI '09.

Hayden, C.M.; Hicks, M.; et.al. A study of dynamic software update quiescence for multithreaded programs, HotSWUp'12

From Live Updates to Dynamic Evolution

Live updates ⇨ Dynamic Software Updates (DSU) ⇨ Dynamic Software Evolution

- Procedure: program replica + state transfer.
- Impact: typically small and local bug fixes or security patches.
- Occurrence: infrequent

Christopher M. Hayden, Edward K. Smith, Michael Hicks, and Jeffrey S. Foster. State transfer for clear and efficient runtime updates. IEEE ICDEW '11.

Christopher M. Hayden, Edward K. Smith, Michail Denchev, Michael Hicks, and Jeffrey S. Foster. Kitsune: efficient, general-purpose dynamic software updating for C. OOPSLA '12.

From Live Updates to Dynamic Evolution

Live updates ➪ Dynamic Software Updates (DSU) ➪ Dynamic Software Evolution

- Procedure: in-place code + state update.
- Impact: typically small and local bug fixes or security patches.
- Occurrence: infrequent

Michael Hicks and Scott Nettles. 2005. Dynamic software updating. ACM Trans. Program. Lang. Syst. (TOPLAS)

Iulian Neamtiu and Michael Hicks. 2009. Safe and timely updates to multi-threaded programs. In PLDI '09.

Cristiano Giuffrida and Andrew S. Tanenbaum. Cooperative update: a new model for dependable live update. In HotSWUp '09

From Live Updates to Dynamic Evolution

Live updates ➪ Dynamic Software Updates (DSU) ➪ Dynamic Software Evolution

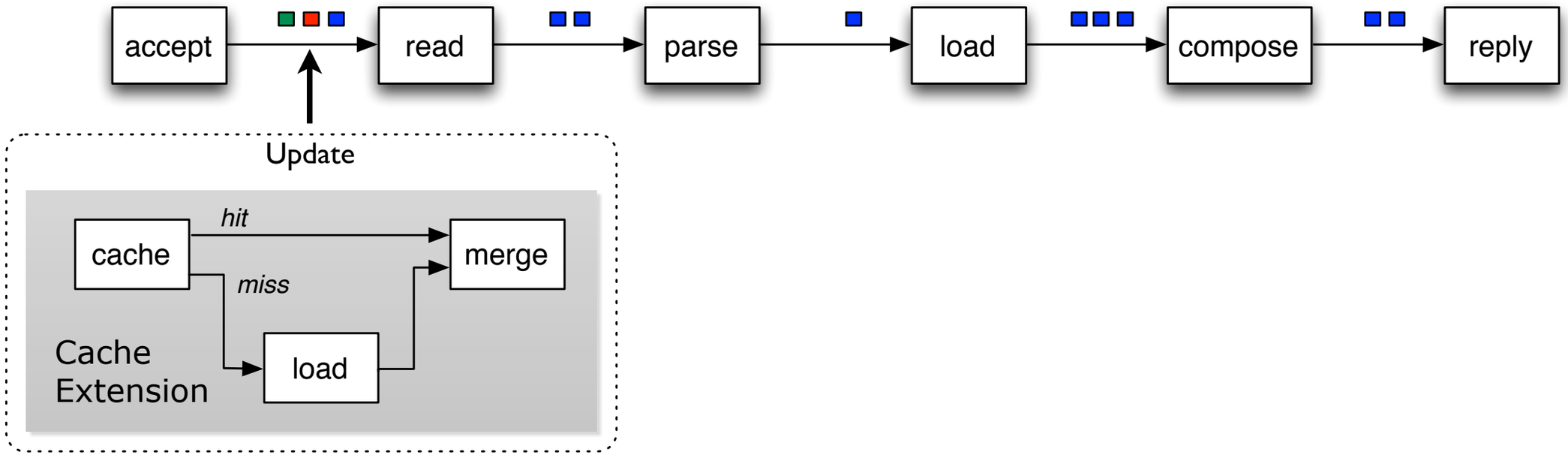
- Procedure: in-place code + state update.
- Impact: local changes (operators) + program updates (graph). ✓
- Occurrence: part of the development process.

➪ Dynamic evolution of any (middleware) dataflow program.

Cristiano Giuffrida, Andrew S. Tanenbaum, et.al. Safe and automatic live update for operating systems. In ASPLOS '13.

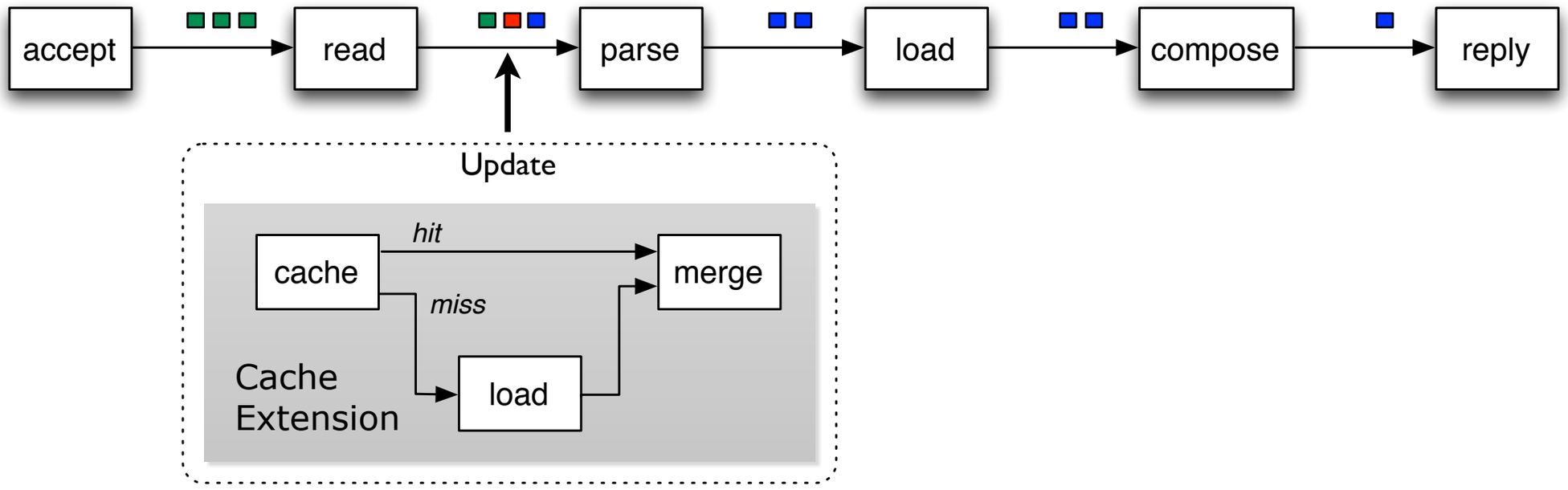
Graph Updates

- Marker-based coordination to transition the graph from one consistent state into the other.



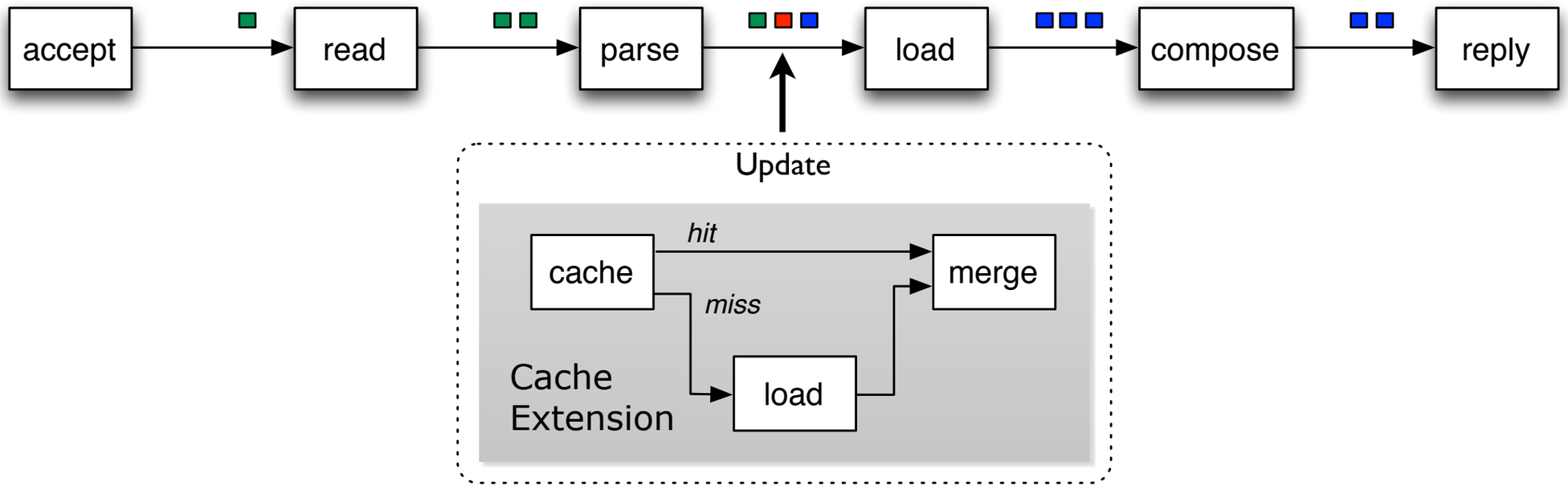
Graph Updates

- Marker-based coordination to transition the graph from one consistent state into the other.



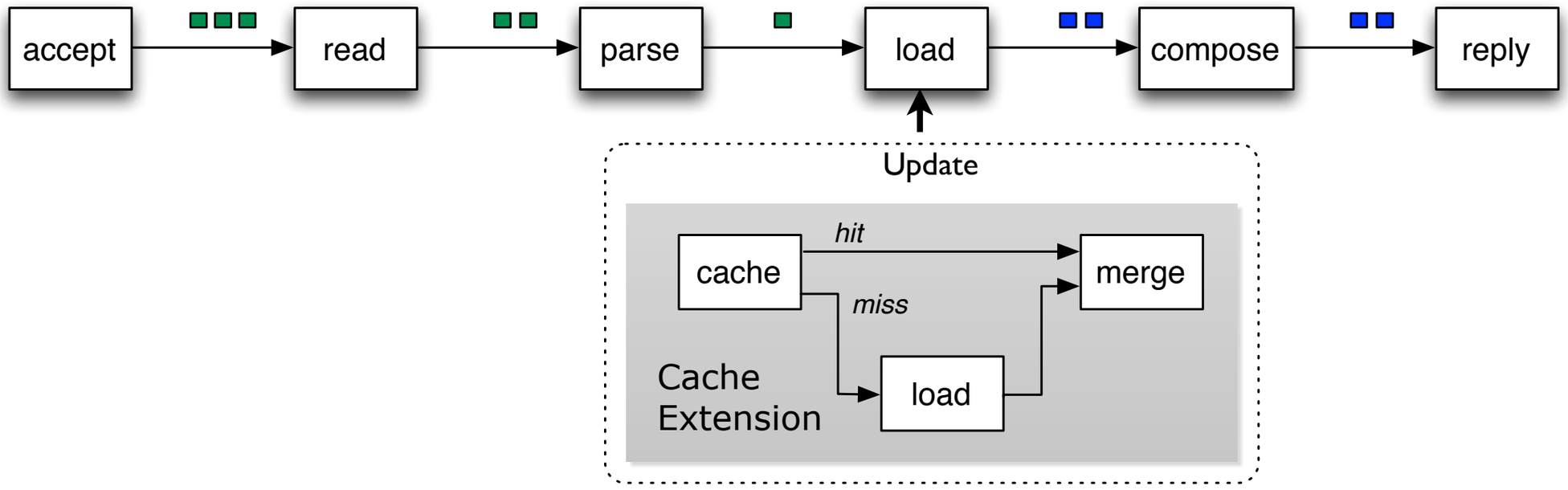
Graph Updates

- Marker-based coordination to transition the graph from one consistent state into the other.



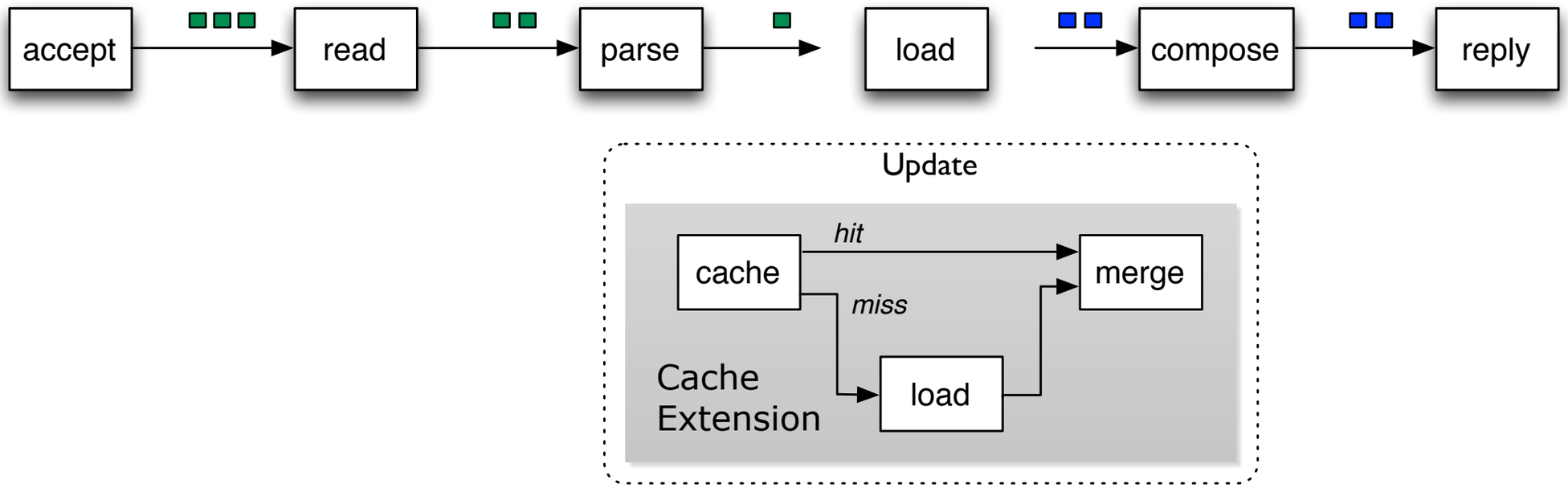
Graph Updates

- Marker-based coordination to transition the graph from one consistent state into the other.



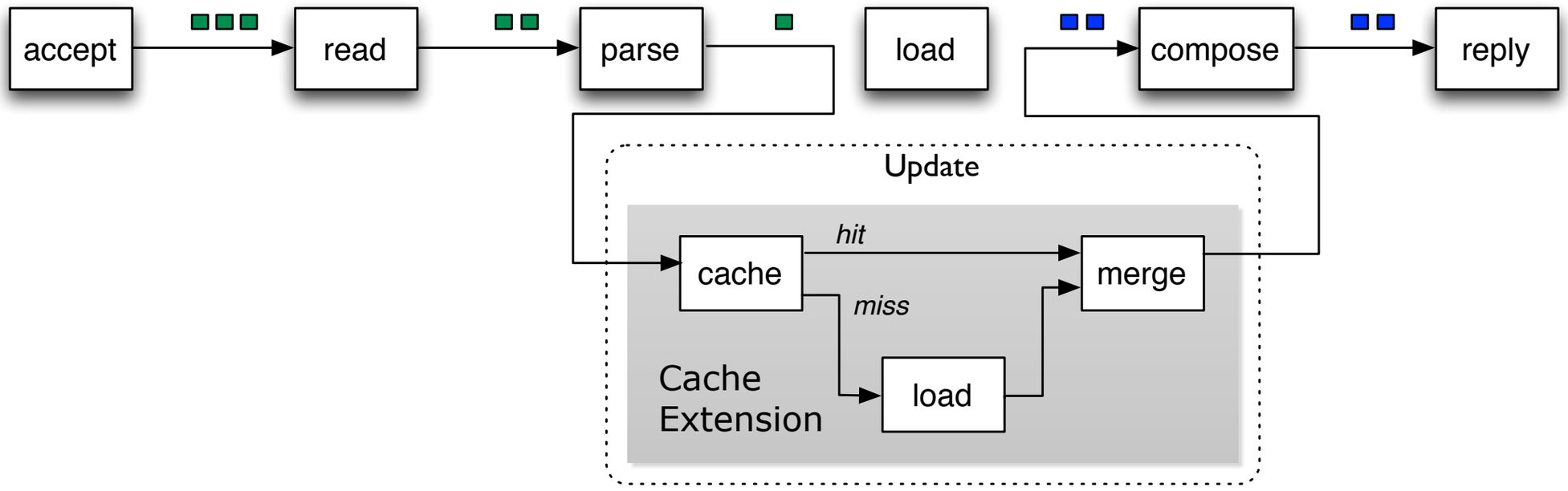
Graph Updates

- Marker-based coordination to transition the graph from one consistent state into the other.



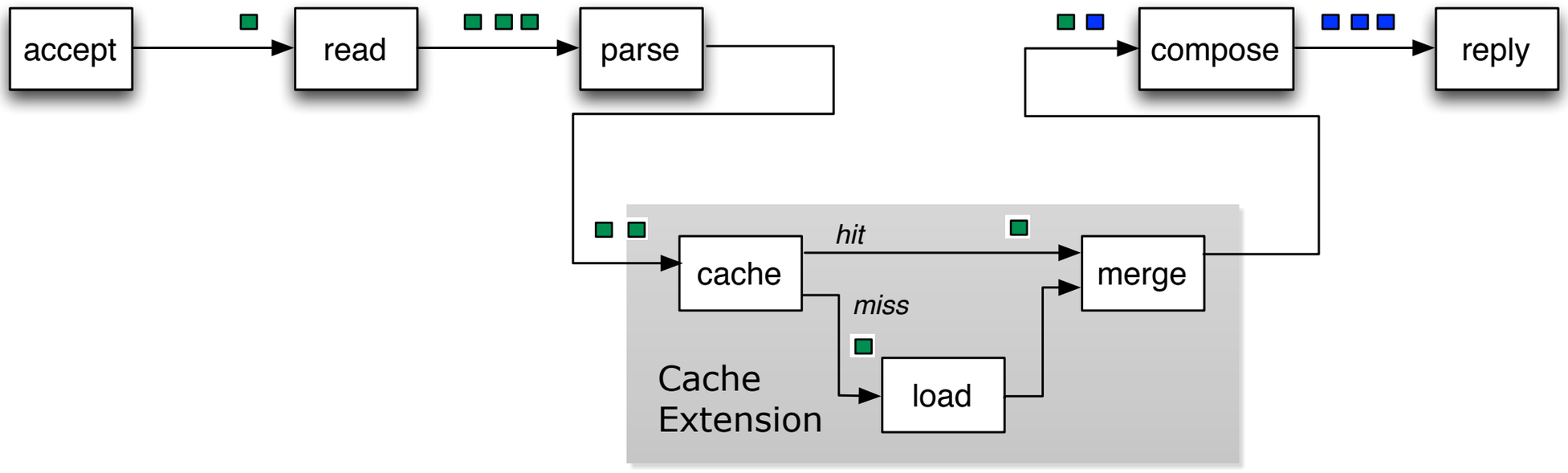
Graph Updates

- Marker-based coordination to transition the graph from one consistent state into the other.

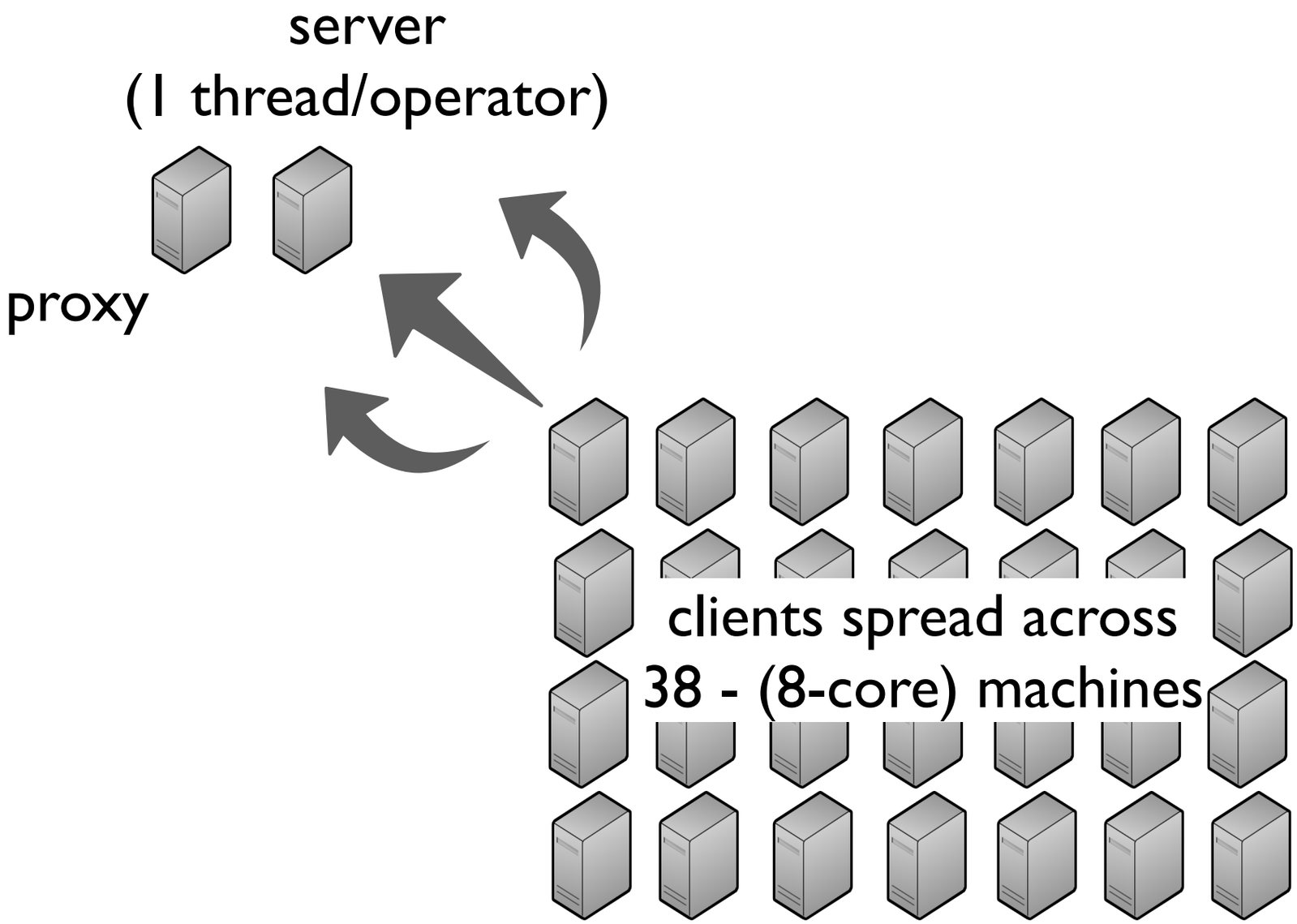


Graph Updates

- Marker-based coordination to transition the graph from one consistent state into the other.

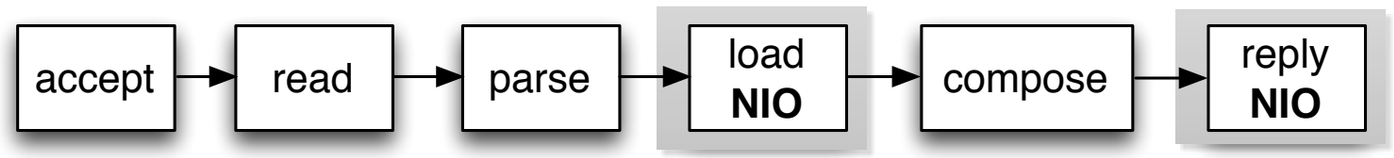


Experimental Setup



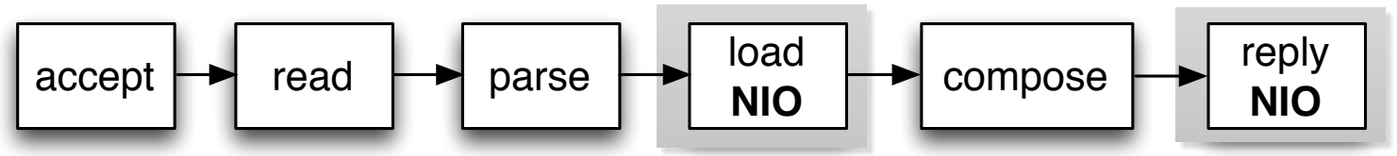
The NIO Switch

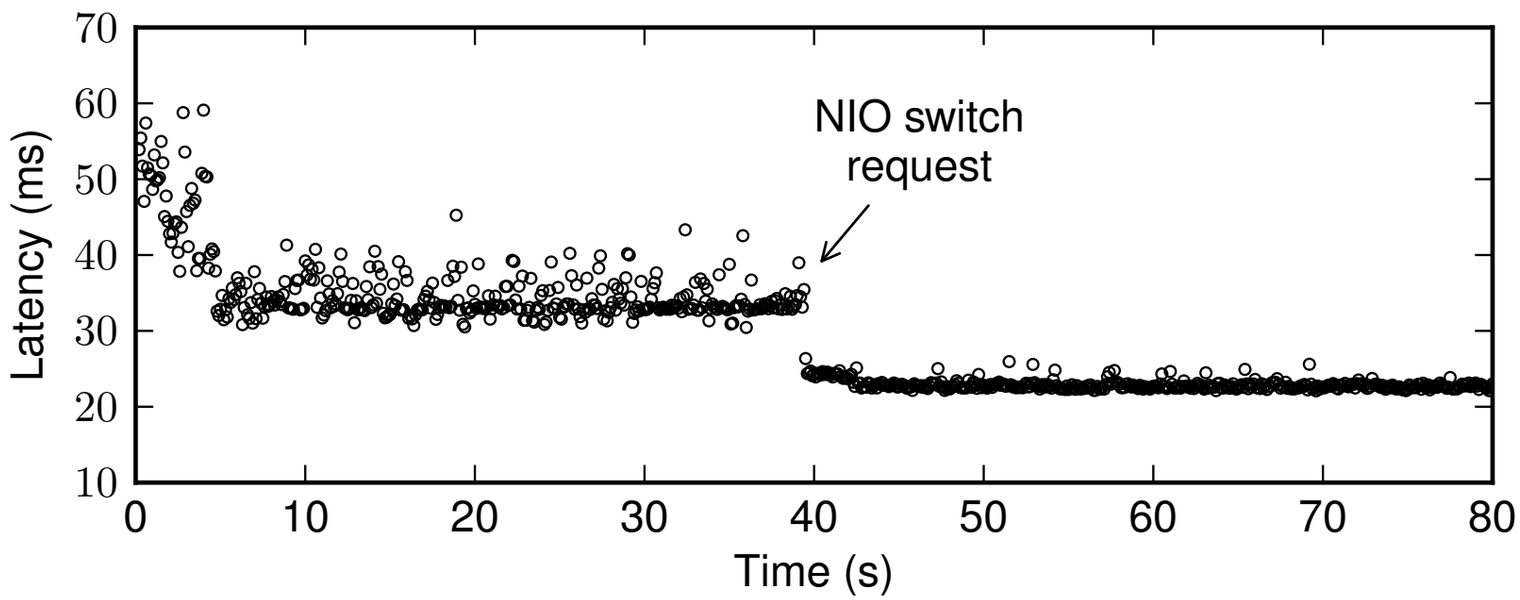
- Experiment: 30 concurrent clients request 1out of 10000 files of size 50kB.
- Update: Coordinated switch (after ~40s) of our HTTP server to support NIO.



The NIO Switch

- Experiment: 30 concurrent clients request 1out of 10000 files of size 50kB.
- Update: Coordinated switch (after ~40s) of our HTTP server to support NIO.



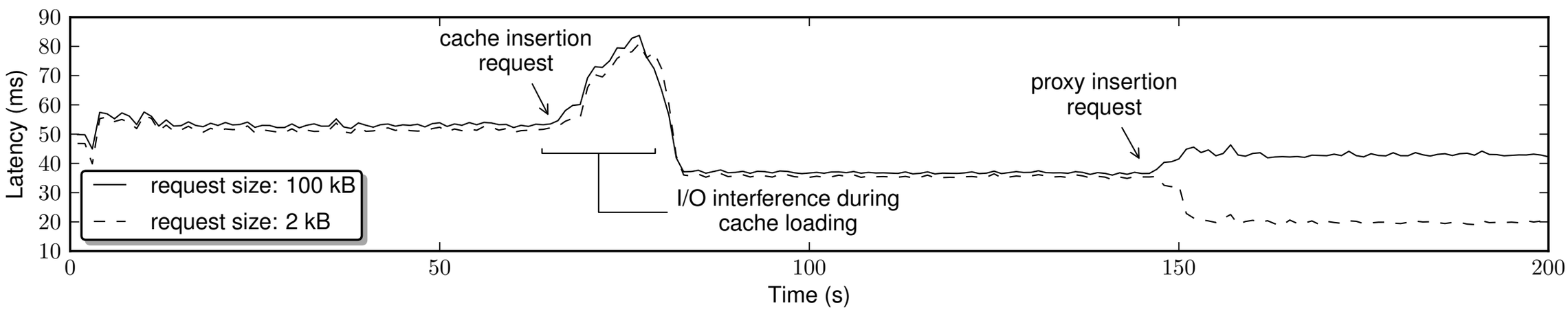


Dynamic Server Evolution

- Experiment:
  - Small (feeds - 2kB) & “large” (webpage - 100kB) files.
  - 30 concurrent clients for each.
  - Goal: avoid penalizing small requests by large requests.
  - Evolution: Cache insertion (~70s) and proxy update (~145s).

Dynamic Server Evolution

- Experiment:
  - Small (feeds - 2kB) & “large” (webpage - 100kB) files.
  - 30 concurrent clients for each.
  - Goal: avoid penalizing small requests by large requests.
  - Evolution: Cache insertion (~70s) and proxy update (~145s).



Conclusion

- Dataflow allows to take live updates to a new level:

  ➪ *Do not update, evolve!*

- See the paper for the solutions of the skipped challenges.

- Future work:

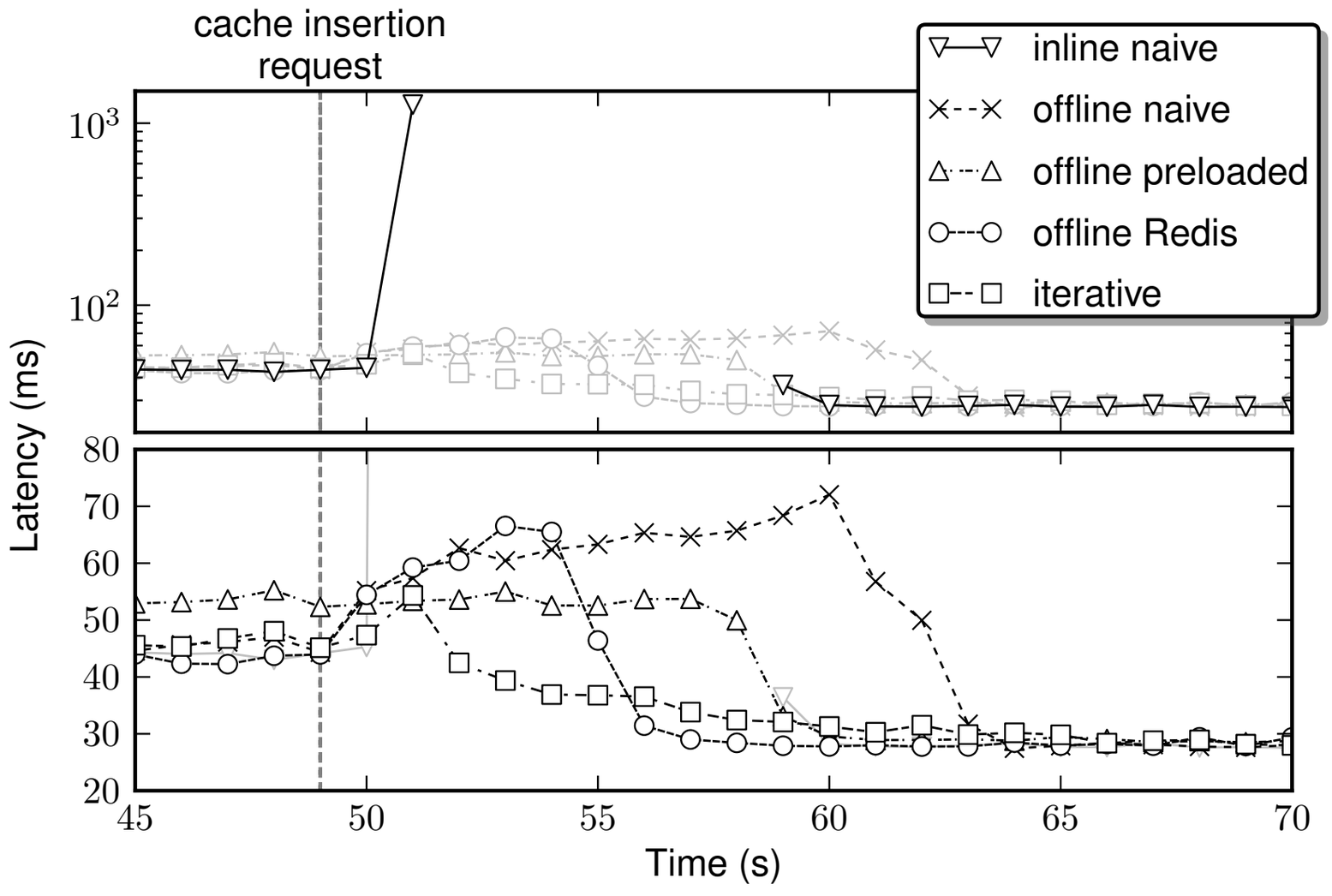
  - Scalable programming model.

  - External resource updates.

Thanks for your attention!

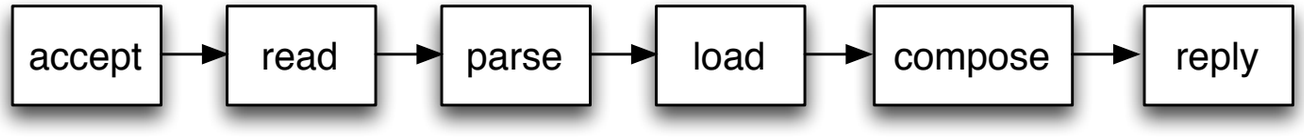
Questions?

Update Contention

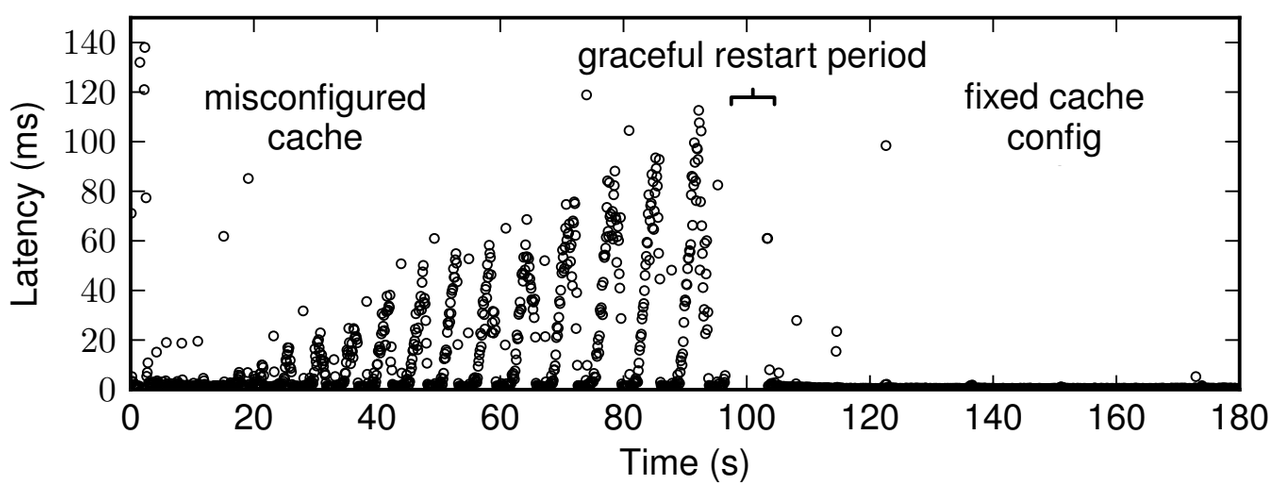


Motivation

- Jetty: Servlet Engine and HTTP Server.
- Web server as an example of a highly concurrent program.



- Jetty's graceful restart after 95s into computation:
  - Minimal default installation, just serving files.
  - 38 concurrent clients request 20 kB file every 20 ms.
- Downtimes ~8-10s ➪ Request latency ~1-2ms.



Smith, E.K.; Hicks, M.; Foster, J.S., "Towards standardized benchmarks for Dynamic Software Updating systems," HotSWUp'12

Programming Model

- Implicit dataflow with Ohua:

```
(ns com.server)

(ohua
  (defn web-server [port]
    (-> port accept read parse load compose reply)))
```

```
public static class NIOFileLoader {
  @Function
  public Object[] load(String resource) throws IOException {
    FileChannel fc = new FileInputStream(resource).getChannel();
    return new Object[] { fc, fc.size() };
  }
}
```

```
(ohua com.server/web-server 80)
```

- Updates in Ohua:

```
(update
  [com.server/reply com.server.update/reply (new ReplyStateTransfer)]
  [com.server/load com.server.update/load])
```

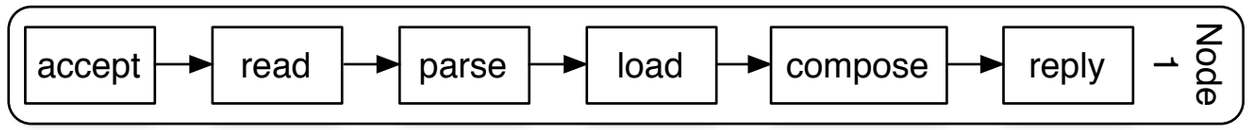
```
public class ReplyStateTransfer implements StateTransfer<server.Reply, server.nio.Reply>{
  @Override
  public void transfer(server.Reply oldVersion, server.nio.Reply newVersion) {
    // perform some fancy state transfer
  }
}
```

```
(update
  [com.server/web-server com.server.update/web-server-with-cache])
```

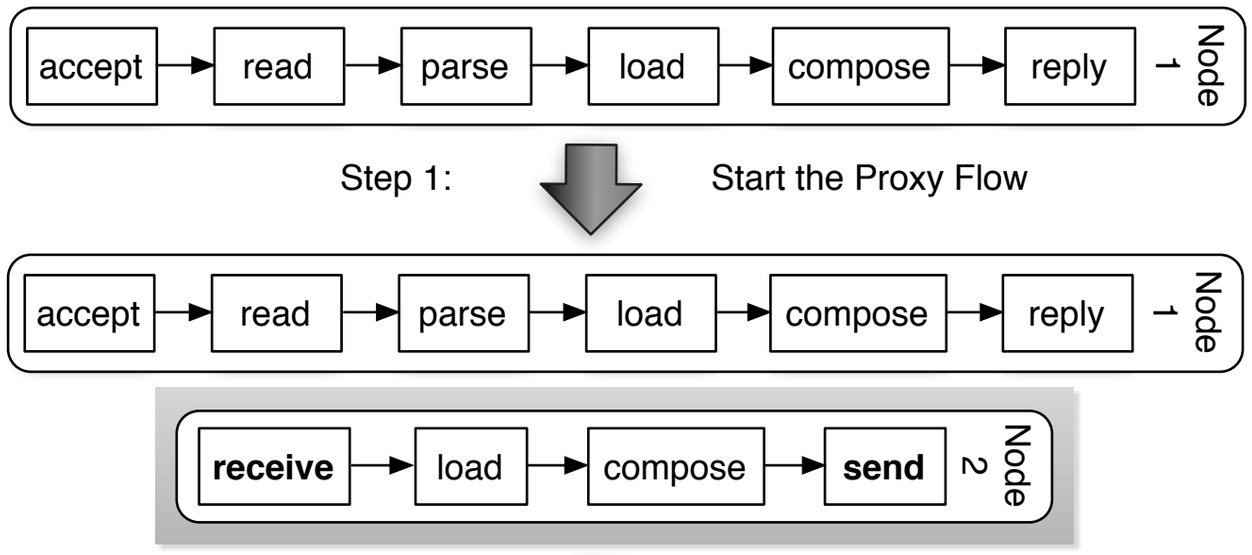
---

https://bitbucket.org/sertel/ohua-updates

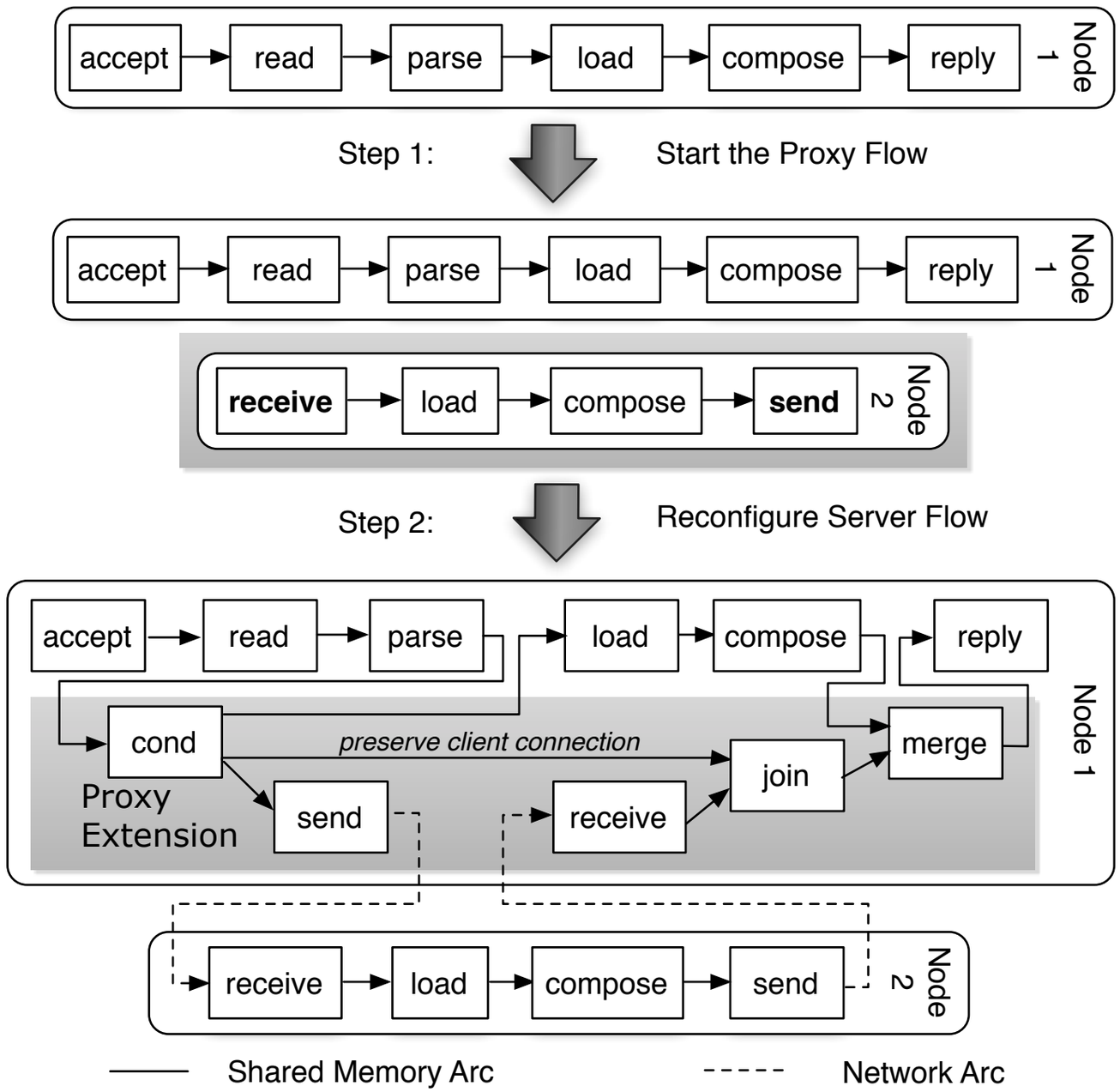
Server Evolution: Proxy



Server Evolution: Proxy

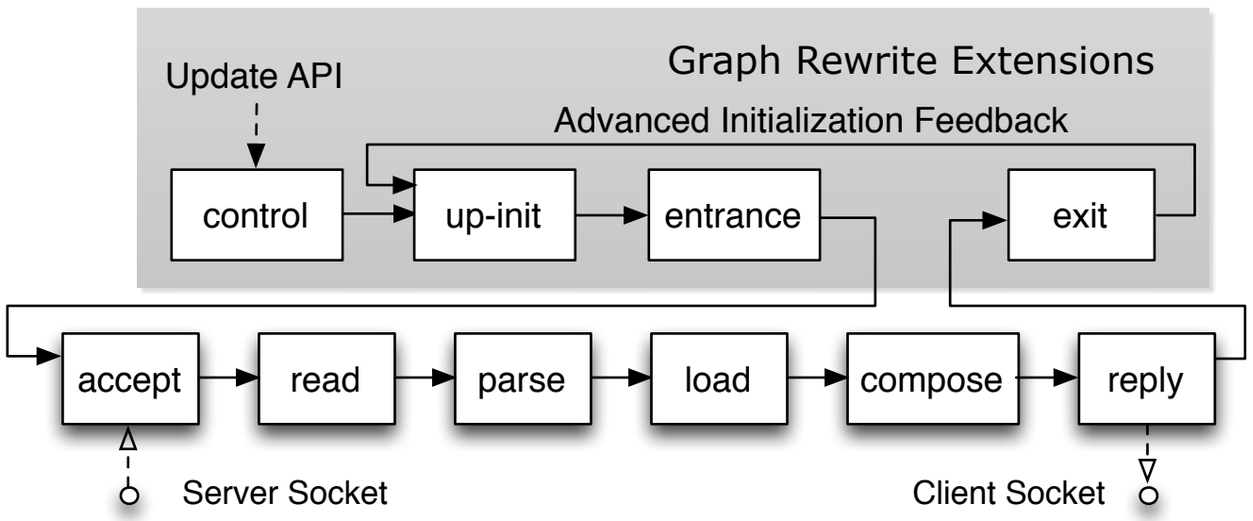


Server Evolution: Proxy



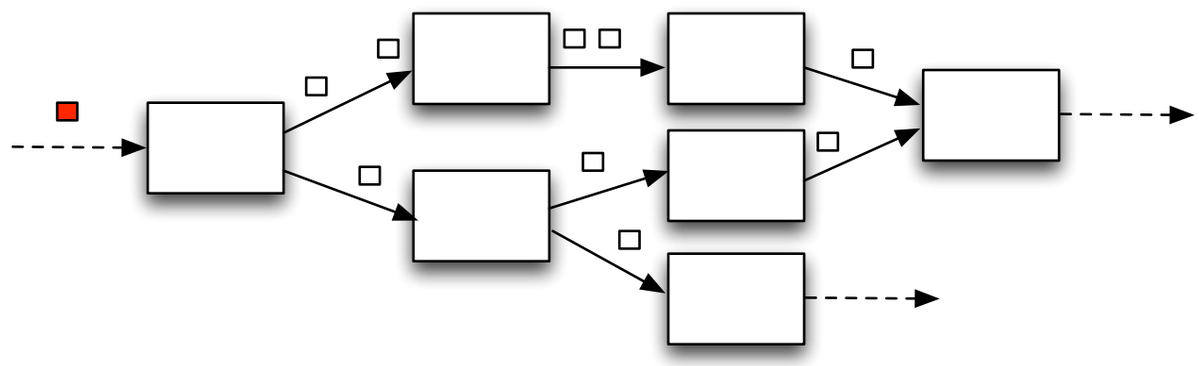
Graph Transformations

- Requirement: Interactive runtime.
- Problem: Interfacing highly concurrent (distributed) programs.
- Runtime Graph Rewrite:
  - Provide unique entrance and exit points.
  - Enhance runtime features with operators (dataflow style).



The Beauty of Abstraction

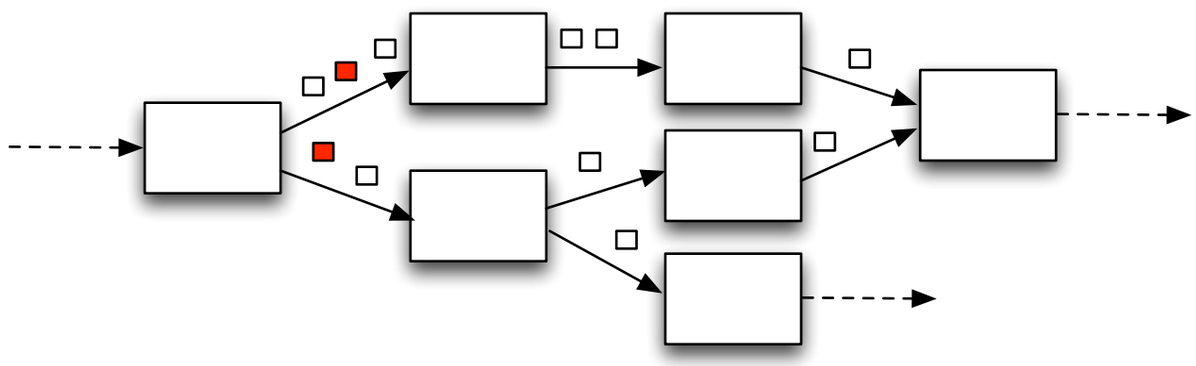
- Dataflow building blocks: arcs, operators, dataflow graph.
- *No assumptions on execution environment.*
- Key insight:
  - That's a distributed system!
  - We know how to reason about time! ➪ "When" solved!



K. Mani Chandy and Leslie Lamport. 1985. Distributed snapshots: determining global states of distributed systems. ACM Trans. Comput. Syst. 3, 1 (February 1985), 63-75.

The Beauty of Abstraction

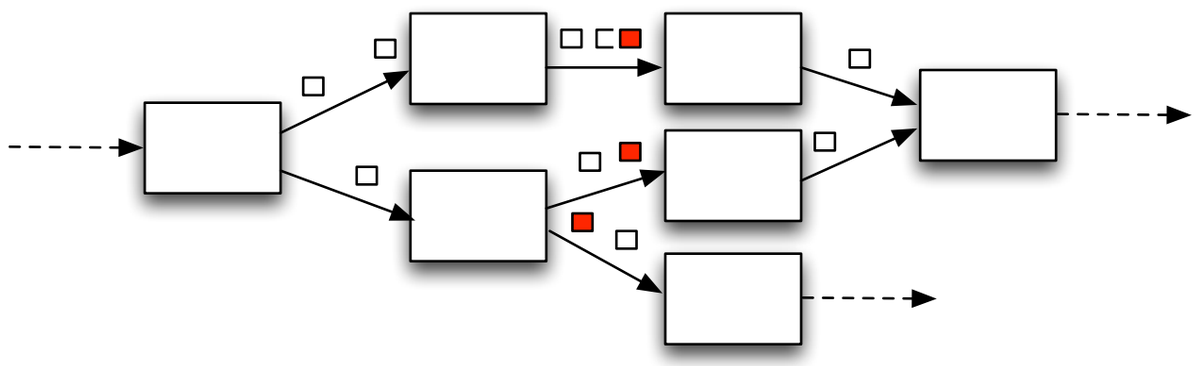
- Dataflow building blocks: arcs, operators, dataflow graph.
- *No assumptions on execution environment.*
- Key insight:
  - That's a distributed system!
  - We know how to reason about time! ➪ "When" solved!



K. Mani Chandy and Leslie Lamport. 1985. Distributed snapshots: determining global states of distributed systems. ACM Trans. Comput. Syst. 3, 1 (February 1985), 63-75.

The Beauty of Abstraction

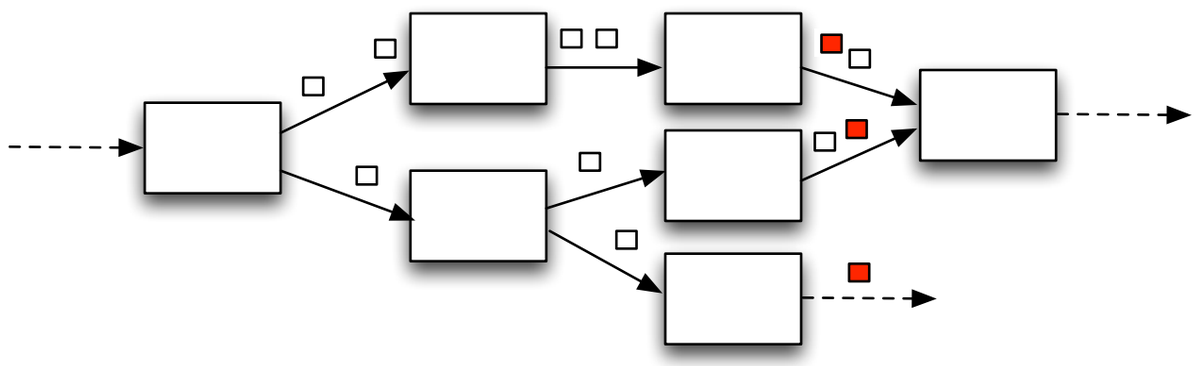
- Dataflow building blocks: arcs, operators, dataflow graph.
- *No assumptions on execution environment.*
- Key insight:
  - That's a distributed system!
  - We know how to reason about time! ➪ "When" solved!



K. Mani Chandy and Leslie Lamport. 1985. Distributed snapshots: determining global states of distributed systems. ACM Trans. Comput. Syst. 3, 1 (February 1985), 63-75.

The Beauty of Abstraction

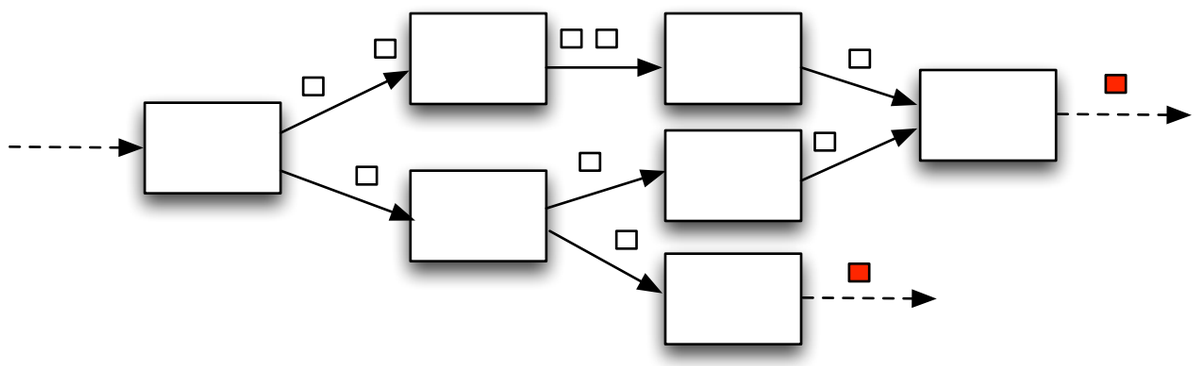
- Dataflow building blocks: arcs, operators, dataflow graph.
- *No assumptions on execution environment.*
- Key insight:
  - That's a distributed system!
  - We know how to reason about time! ➪ "When" solved!



K. Mani Chandy and Leslie Lamport. 1985. Distributed snapshots: determining global states of distributed systems. ACM Trans. Comput. Syst. 3, 1 (February 1985), 63-75.

The Beauty of Abstraction

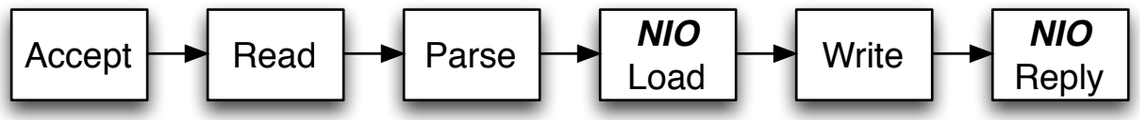
- Dataflow building blocks: arcs, operators, dataflow graph.
- *No assumptions on execution environment.*
- Key insight:
  - That's a distributed system!
  - We know how to reason about time! ➪ "When" solved!



K. Mani Chandy and Leslie Lamport. 1985. Distributed snapshots: determining global states of distributed systems. ACM Trans. Comput. Syst. 3, 1 (February 1985), 63-75.

Related Work

- Current state-of-the-art: STUMP and Kitsune (for C), Rubah (for Java)
  - ➪ "Stop the world", Update points and relaxed synchronization
- No support for non-blocking updates.
- No easy but yet efficient (runtime overhead, scalability) update algorithms.
- No support for fine-grained and complex updates (mutual references).



- ➪ Need to solve the consistency problem!
- Closest work: Cooperative Live Updates (for Operating Systems)

Iulian Neamtiu and Michael Hicks. 2009. Safe and timely updates to multi-threaded programs. In PLDI '09. ACM, New York, NY, USA

Hayden, C.M.; Saur, K.; Hicks, M.; Foster, J.S..2012. A study of dynamic software update quiescence for multithreaded programs, HotSWUp'12

Christopher M. Hayden, Edward K. Smith, Michail Denchev, Michael Hicks, and Jeffrey S. Foster. 2012. Kitsune: efficient, general-purpose dynamic software updating for C. In OOPSLA '12.

Luís Pina and Michael Hicks. 2013. Rubah: Efficient, General-purpose Dynamic Software Updating for Java. In HotSWUp'13.

Cristiano Giuffrida and Andrew S. Tanenbaum. 2009. Cooperative update: a new model for dependable live update. In HotSWUp '09.

Cristiano Giuffrida, Anton Kuijsten, and Andrew S. Tanenbaum. 2013. Safe and automatic live update for operating systems. In ASPLOS '13.