

Cooperative Update: A New Model for Dependable Live Update

Cristiano Giuffrida Andrew S. Tanenbaum

Vrije Universiteit Amsterdam

Second ACM Workshop on
Hot Topics in Software Upgrades

Disney World, Orlando, FL
25 October 2009



- 1 Motivation
- 2 Existing Models for Live Update
- 3 Problems and Challenges
- 4 Cooperative Update
- 5 Summary



- 1 Motivation
- 2 Existing Models for Live Update
- 3 Problems and Challenges
- 4 Cooperative Update
- 5 Summary



Characteristics

- Need for 24/7/365 operation.
- Disruption of service and loss of transient state is ill-affordable.
- Examples: business systems, mission/safety critical systems.

Consequences of downtime

- eBay lost \$5 million in 1999.
- Blackout affected 50 million people in U.S. and Canada in 2004.
- 26 American soldiers died during the Gulf War.



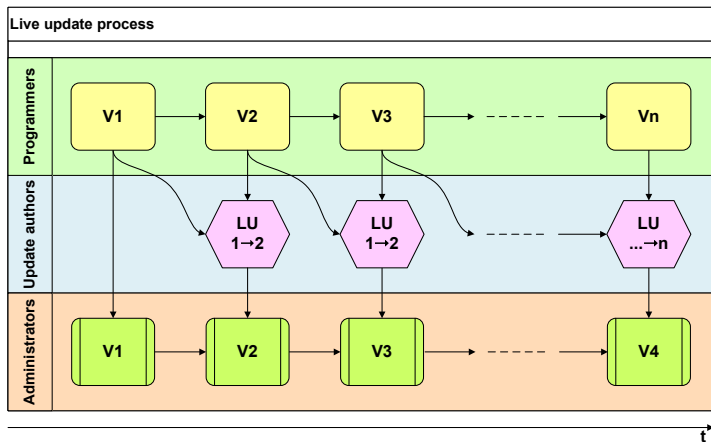
- A solution to support evolution in high-availability environments.
- Updates are generally perceived as a possible reliability threat.
- At high-availability sites, average time-to-update is **30** days.
- Live updates cannot afford weaker reliability guarantees.
- This motivates our focus on reliability.



- 1 Motivation
- 2 Existing Models for Live Update
- 3 Problems and Challenges
- 4 Cooperative Update
- 5 Summary



Software-based live update



A push model

- 1 The system is interrupted at an arbitrary point in time.
- 2 The update is glued into the running system and state transferred.
- 3 Execution is redirected to the new version.

Properties

- Suitable for binary backward compatibility.
- Safety constraints checked at runtime.
- Usually limited to type-safety enforced eagerly or lazily.



A pull model

- 1 The system reaches a valid update point and notifies the live update infrastructure (other variants are possible).
- 2 If an update is available, it is installed and state is transferred.
- 3 Execution is redirected to the new version.

Properties

- Suitable for source backward compatibility.
- Safety checks rely on predefined update points.
- Static/dynamic analysis can provide better safety guarantees.



The upsides

- Separation of concerns is good.
- Need to integrate into existing software systems.

The downsides

- OSS projects doubling in number and size every ≈ 14 months.
- Can we scale to large systems and complex updates?



- 1 Motivation
- 2 Existing Models for Live Update
- 3 Problems and Challenges**
- 4 Cooperative Update
- 5 Summary



- Complexity grows with the complexity of the update.
- This is a fundamental problem.
- But different models can handle this challenge in different ways.
- Note: state transfer is not only to handle datatype changes.
- See the paper for examples and more details.



- Chances of endless wait grow with the complexity of the update.
- A nontrivial update process may not complete in bounded time.
- Can affect both the interrupt and the invoke model.
- Problem arises when safety constraints are enforced eagerly.
- Problem worsens with stronger safety constraints.



- Manual inspection effort grows with the complexity of the update.
- Verify update compliance with any of the possible system states.
- Affects in different ways the interrupt model and the invoke model.
- For complex systems, the number of states may be very large.
- Number of states grows exponentially with the number of threads.



Dependability problems

- State transfer and manual inspection are error-prone.
- Indefinite wait at update time is bad.
- Trading safety for flexibility is painful.

Most limiting assumptions

- The system is a hostile environment.
- The nature of the update is ignored.
- Assumptions derived from the transparency model.

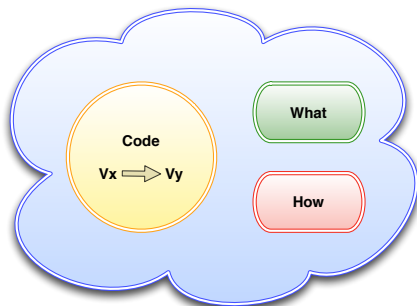


- 1 Motivation
- 2 Existing Models for Live Update
- 3 Problems and Challenges
- 4 Cooperative Update**
- 5 Summary



- Focus on dependability.
- No transparency or backward compatibility.
- The system cooperates in the update process.
- Live update integrated as part of the development cycle.
- The nature of the update characterizes the update process.





- Updates are made self-describing with enclosed metadata.
- Developers' specifications can simplify the update process.



- The system is receptive to changes.
- An update manager processes the update package.
- Specifications are translated into an update protocol.
- The update protocol drives the system into the target state.
- Live update is only performed after then.



The live update process

- 1 Initialization.
- 2 Notification.
- 3 Preparation.
- 4 State checking.
- 5 State transfer.
- 6 Replacement.

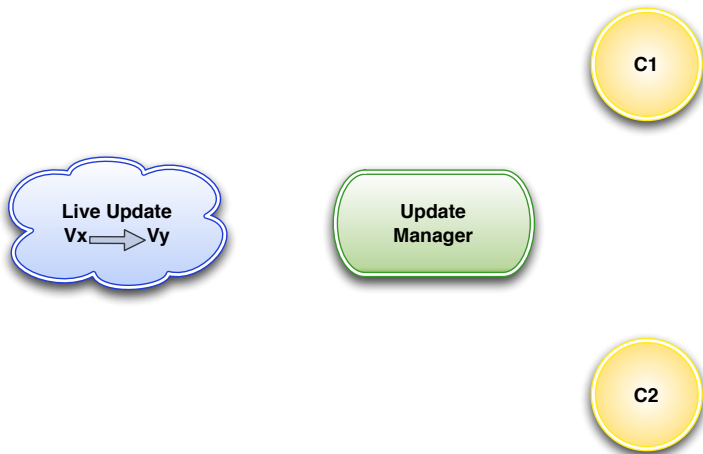


The live update process

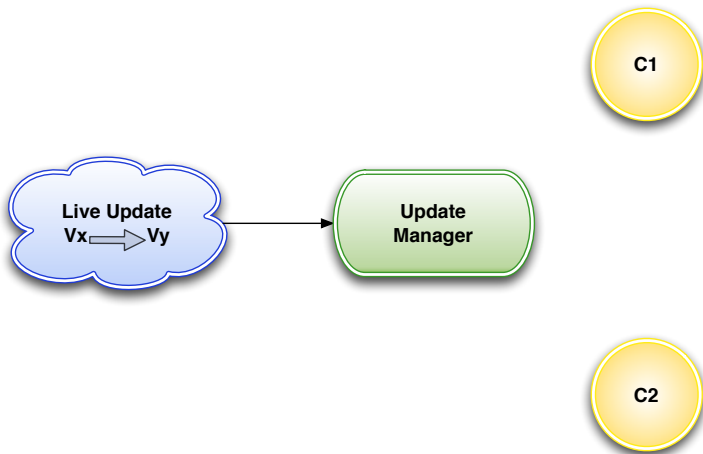
- 1 Initialization.
- 2 Notification.
- 3 Preparation.
- 4 State checking.
- 5 State transfer.
- 6 Replacement.



The live update process



The live update process

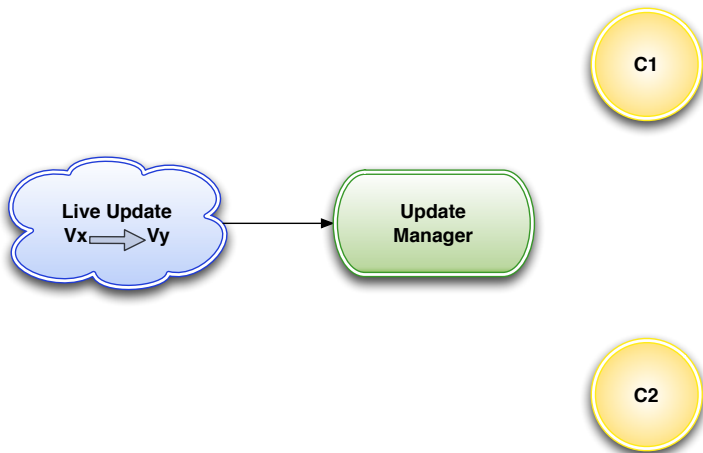


The live update process

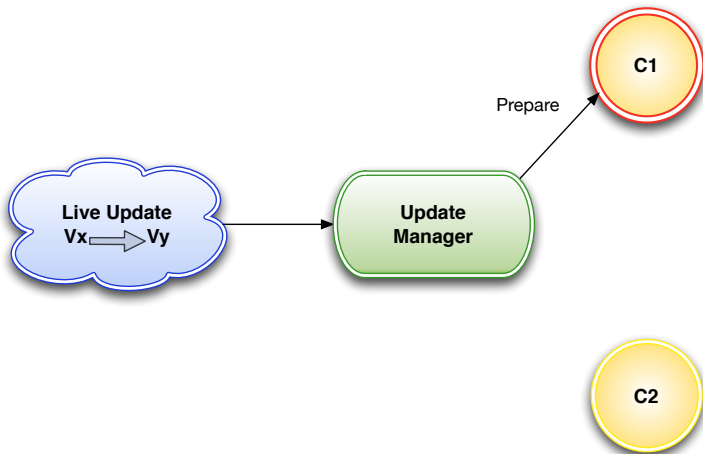
- 1 Initialization.
- 2 Notification.
- 3 Preparation.
- 4 State checking.
- 5 State transfer.
- 6 Replacement.



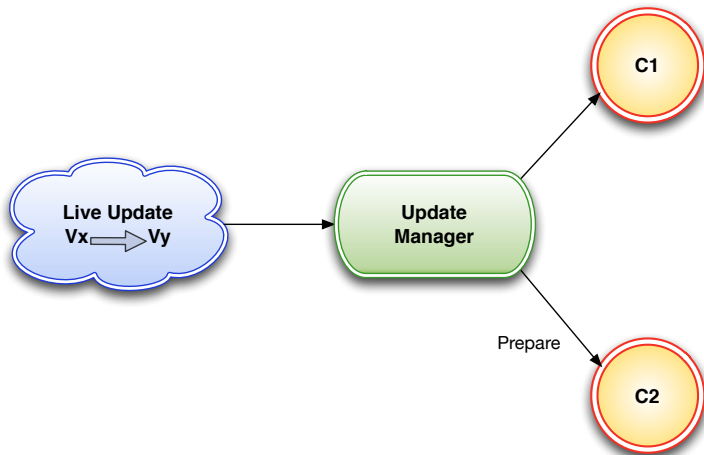
The live update process



The live update process



The live update process

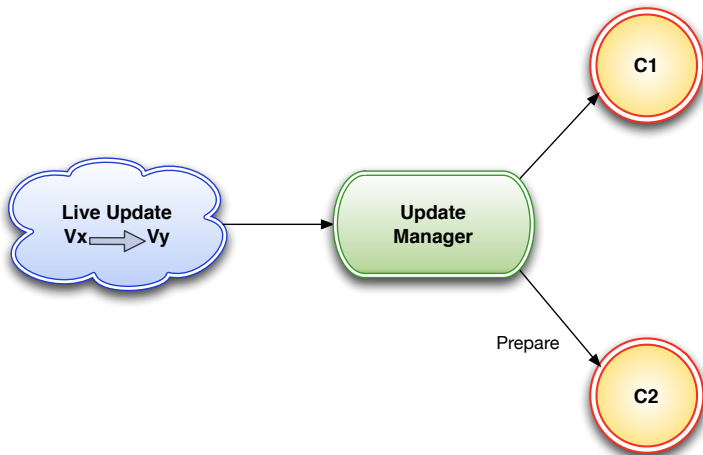


The live update process

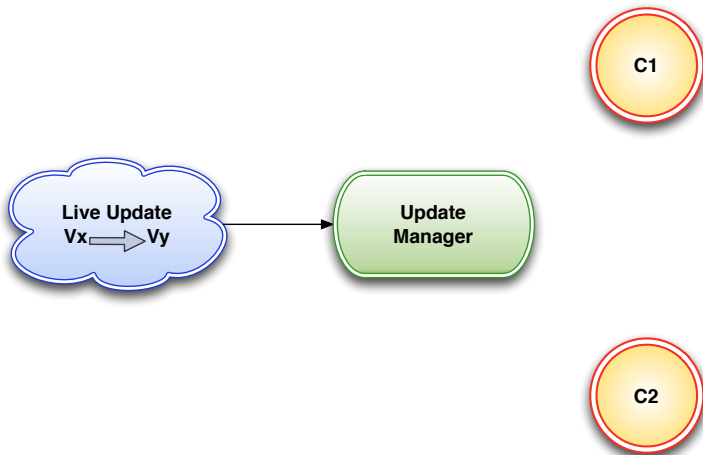
- 1 Initialization.
- 2 Notification.
- 3 **Preparation.**
- 4 State checking.
- 5 State transfer.
- 6 Replacement.



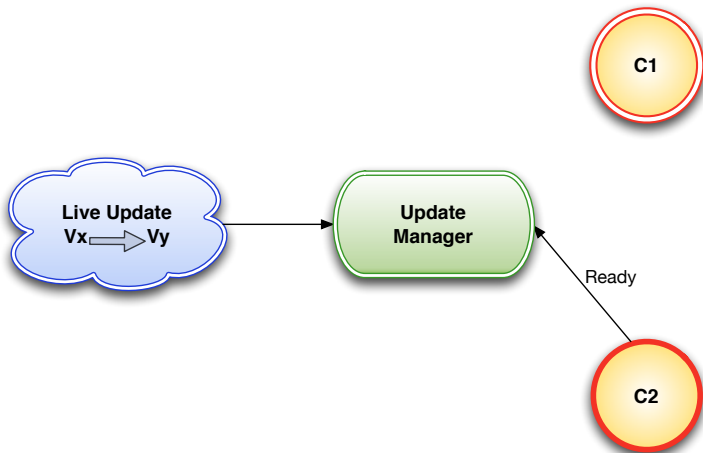
The live update process



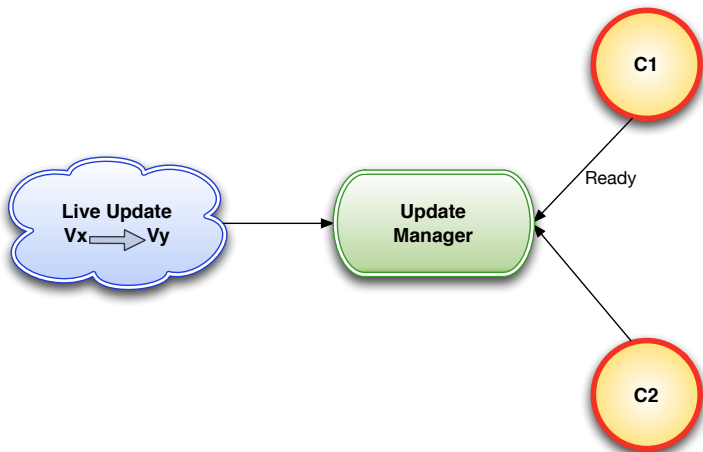
The live update process



The live update process



The live update process

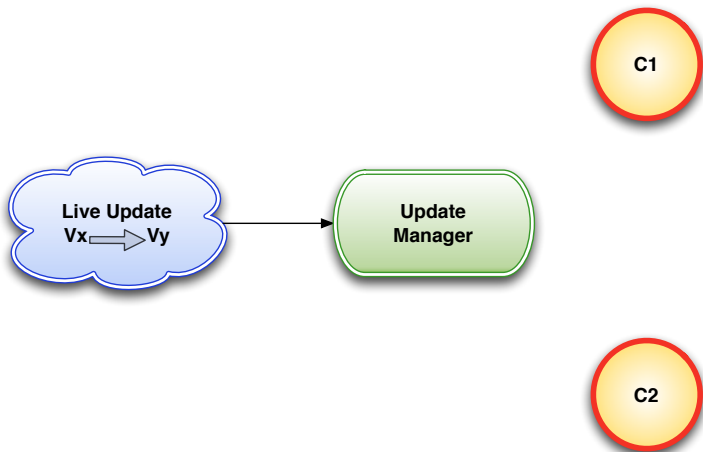


The live update process

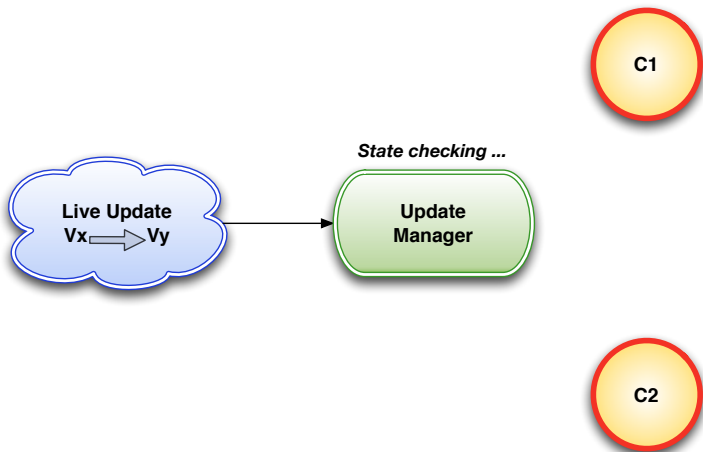
- 1 Initialization.
- 2 Notification.
- 3 Preparation.
- 4 State checking.
- 5 State transfer.
- 6 Replacement.



The live update process



The live update process

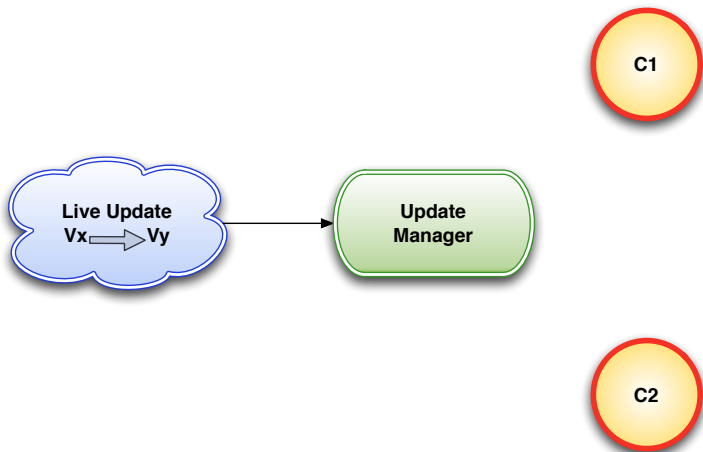


The live update process

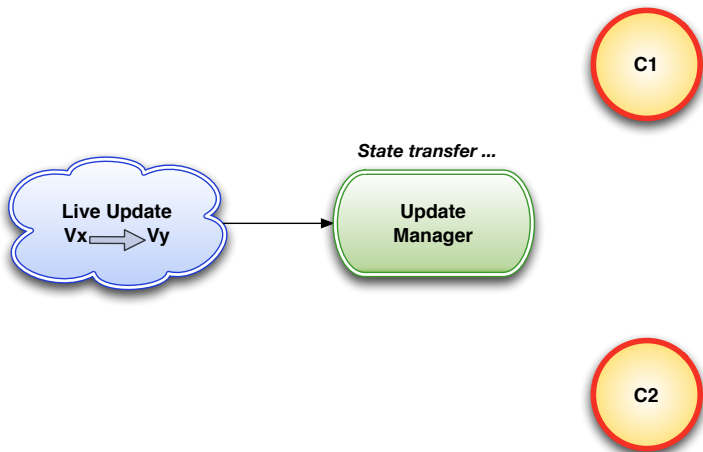
- 1 Initialization.
- 2 Notification.
- 3 Preparation.
- 4 State checking.
- 5 **State transfer.**
- 6 Replacement.



The live update process



The live update process

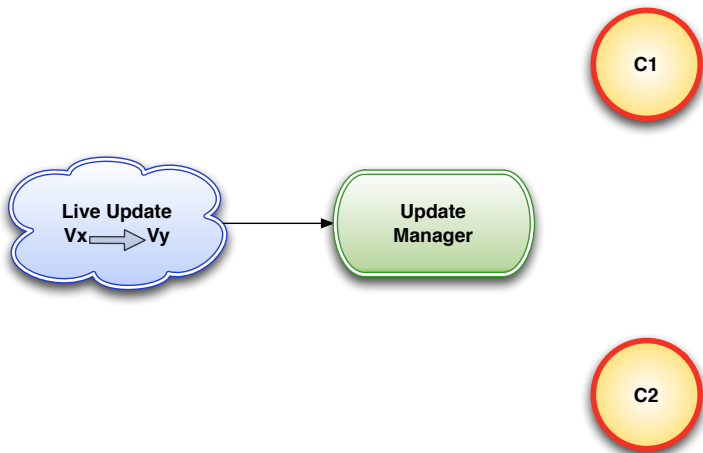


The live update process

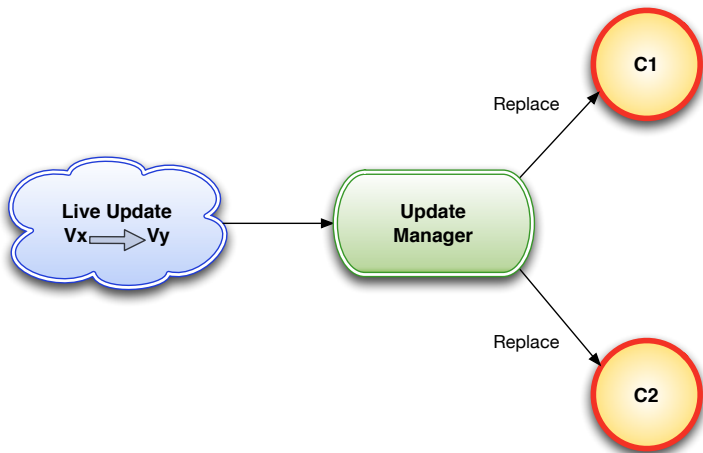
- 1 Initialization.
- 2 Notification.
- 3 Preparation.
- 4 State checking.
- 5 State transfer.
- 6 Replacement.



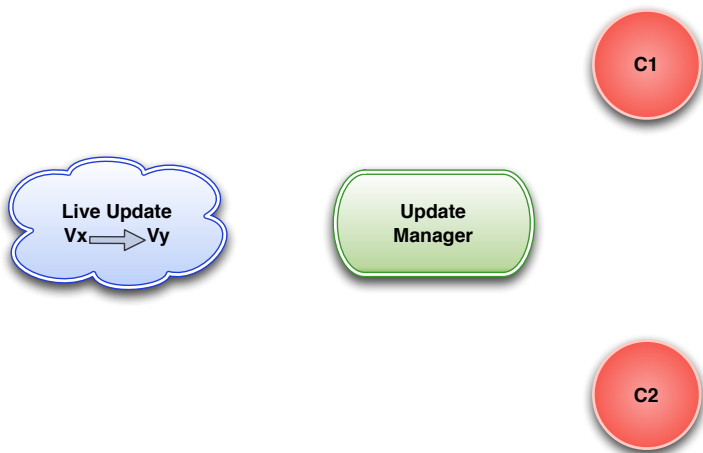
The live update process



The live update process



The live update process



Properties

- The live update process is deterministic and time-bounded.
- The update only occurs in a stable and known state.
- The feasibility of an update becomes a implementation problem.
- Flexibility to tune atomicity constraints at update time.

Dependability benefits

- 1 State transfer still required, but simpler.
- 2 Detection of safe update time at runtime is no longer an issue.
- 3 Ensuring correctness of execution at design time.



Implementation considerations

- Model effective with complex systems and a modular design.
- Components need a well-defined interface.
- Components need an appropriate level of isolation.
- A component could be a module, object, process, etc.
- Promotes a better system design and documentation of changes.



- 1 Motivation
- 2 Existing Models for Live Update
- 3 Problems and Challenges
- 4 Cooperative Update
- 5 Summary**



- A new cooperative model for dependable live update.
- Tailored to complex systems and a broad range of updates.
- System no longer assumed hostile and unaware of updates.
- Solves many recurring problems structurally.
- Sacrifices transparency for improved dependability and flexibility.



