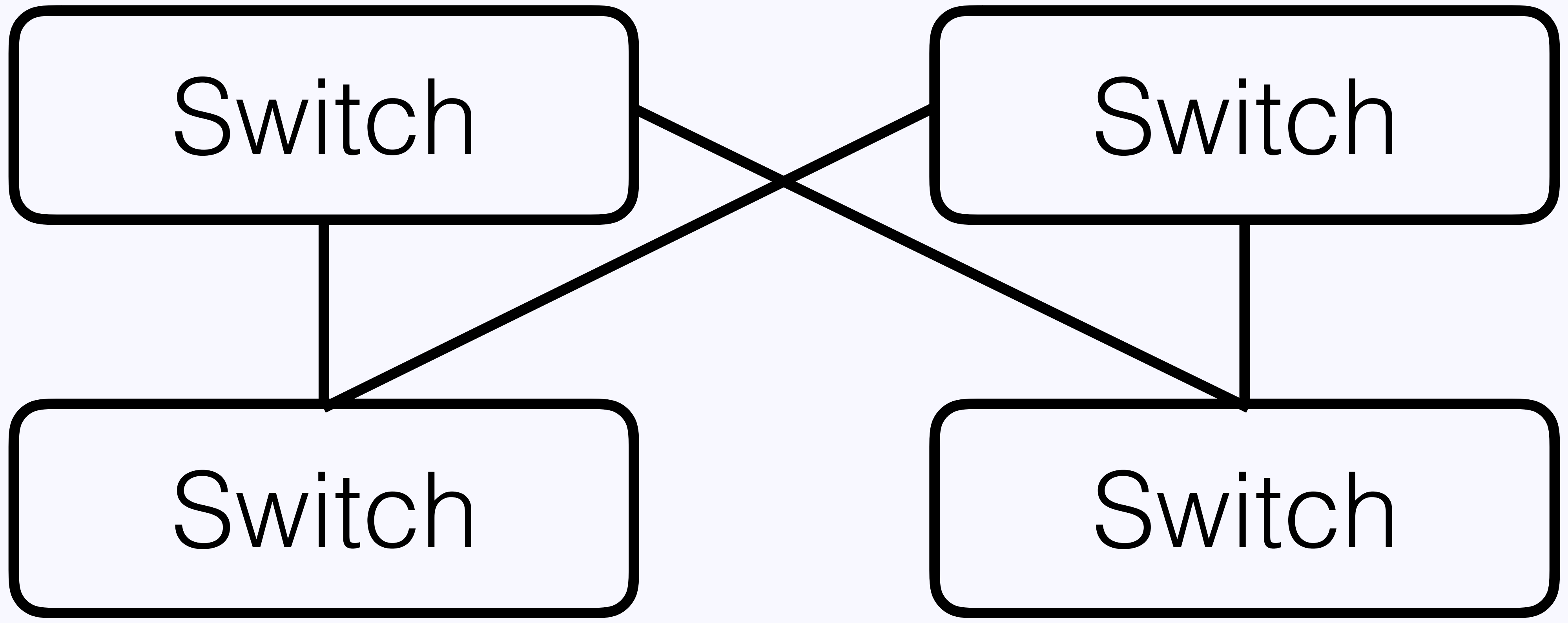
SCL: Simple Coordination Layer

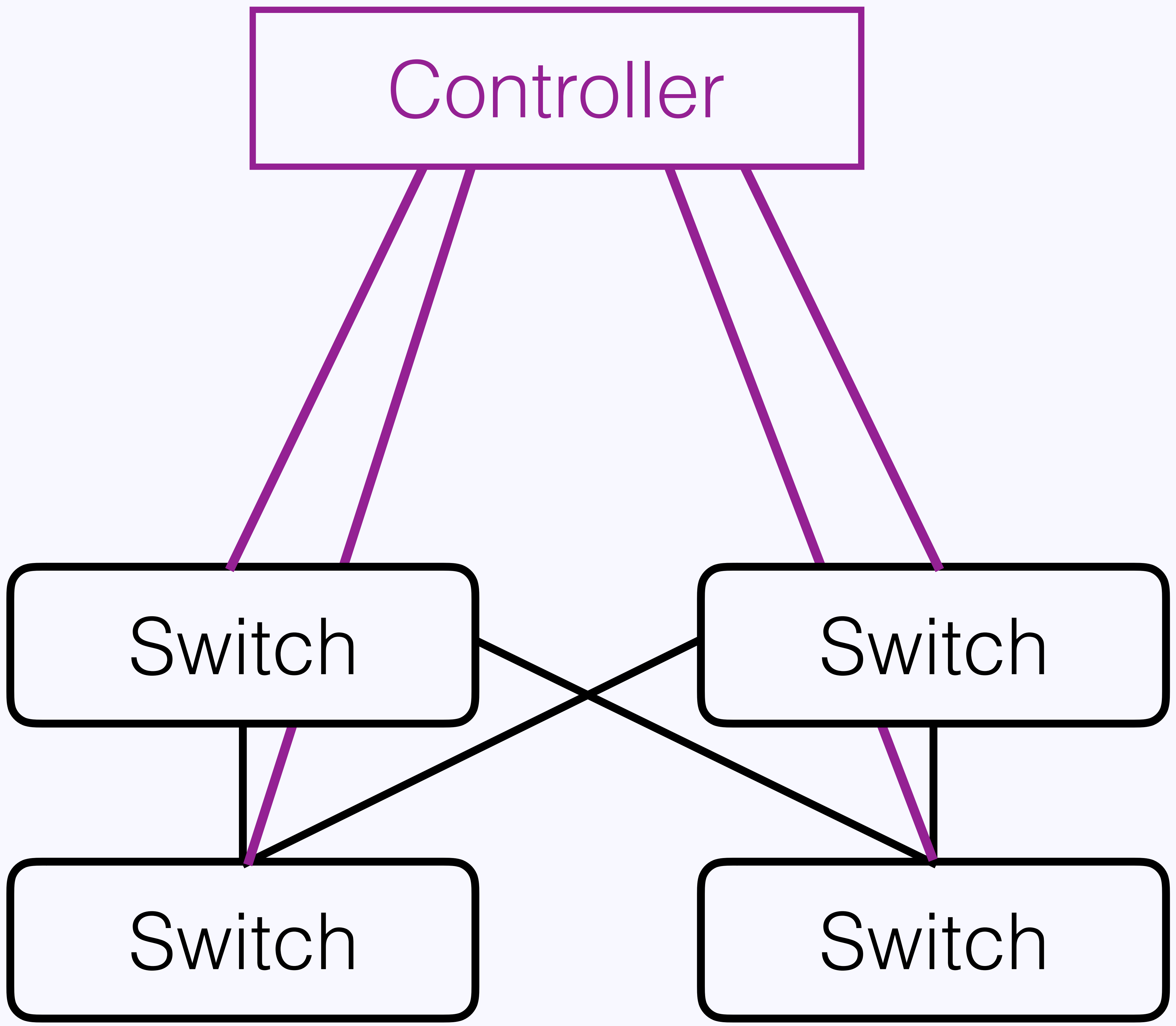
Aurojit Panda, Wenting Zheng, Xiaohe Hu, Arvind Krishnamurthy, Scott Shenker
UC Berkeley, Tsinghua University, University of Washington, ICSI

Software Defined Networks

- Forwarding implemented by switches.

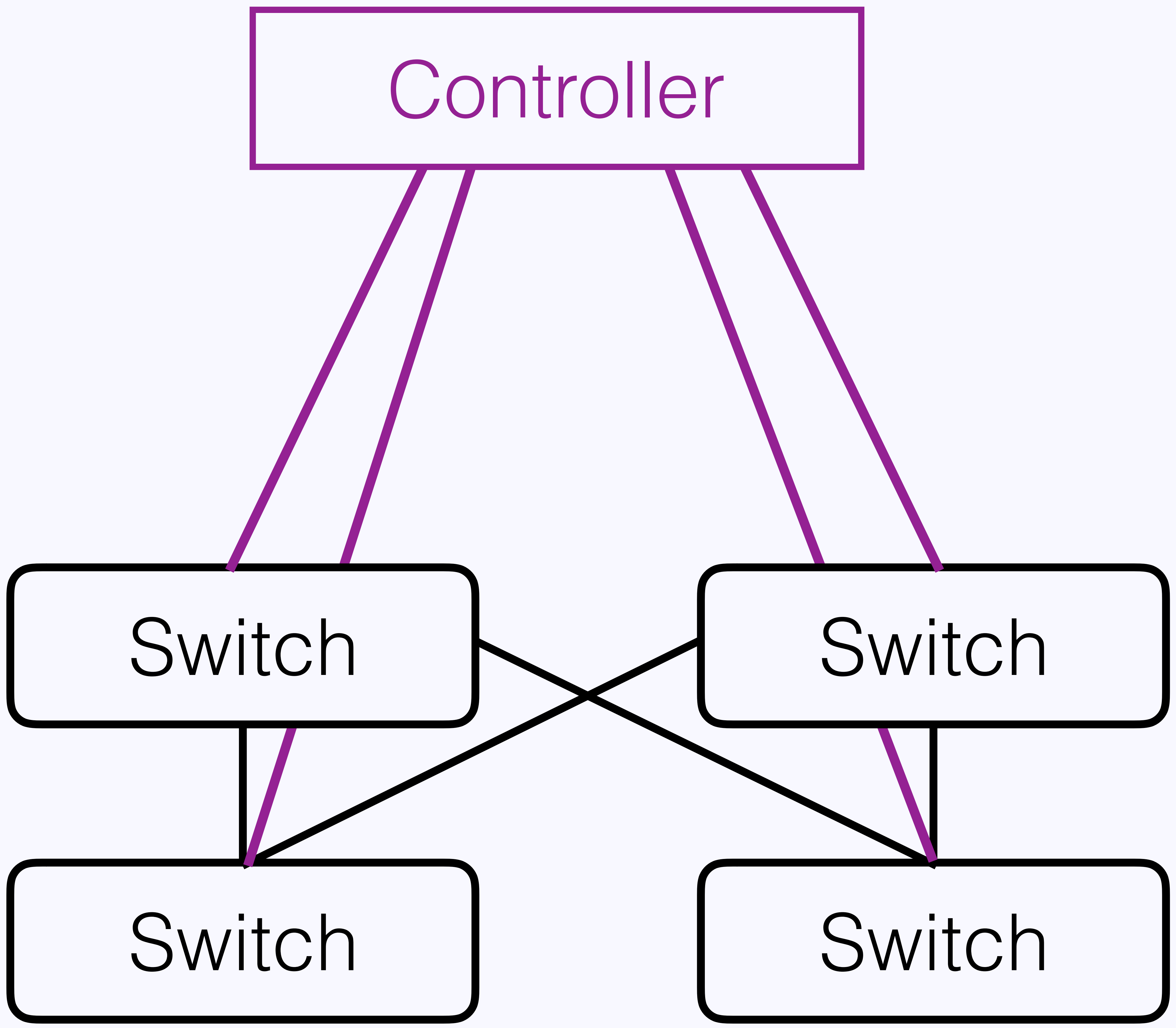


Software Defined Networks



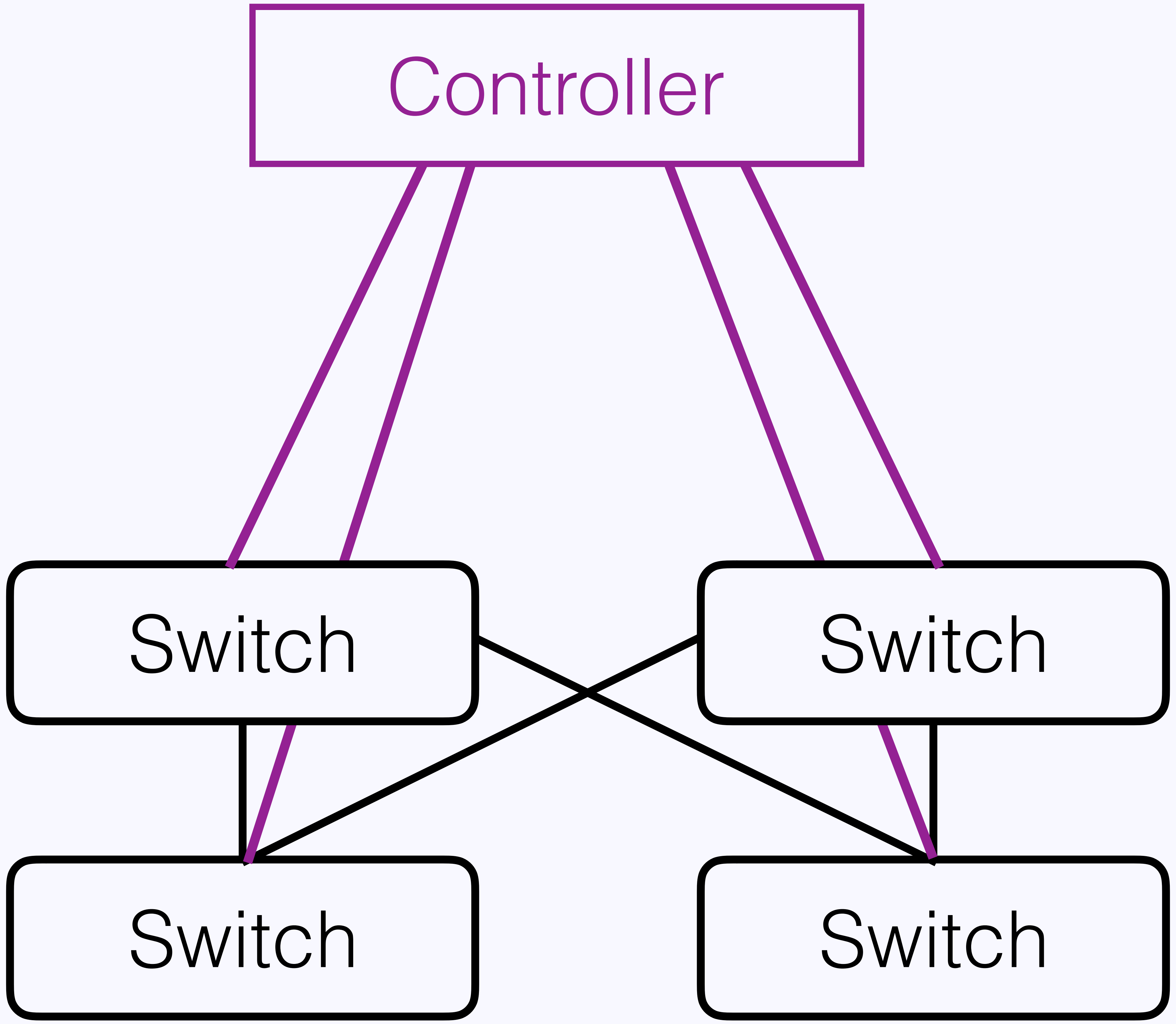
- Forwarding implemented by switches.
- Rules computed by controllers.

Software Defined Networks



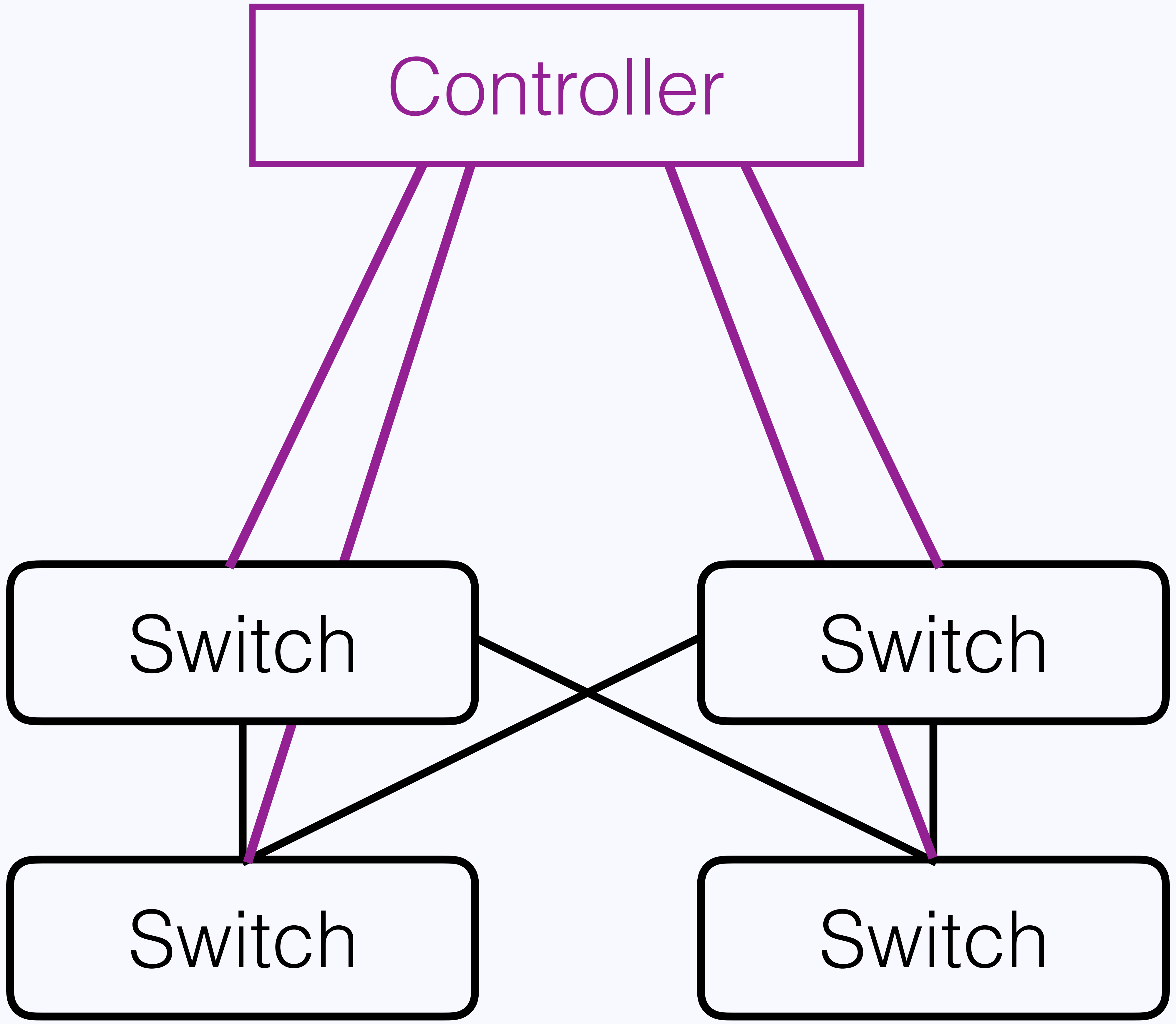
- Forwarding implemented by switches.
- Rules computed by controllers.
- Rules depend on **policy** and **network state.**

Software Defined Networks



- Forwarding implemented by switches.
- Rules computed by controllers.
- Rules depend on **policy** and **network state.**
- **Policy**: What paths are acceptable?

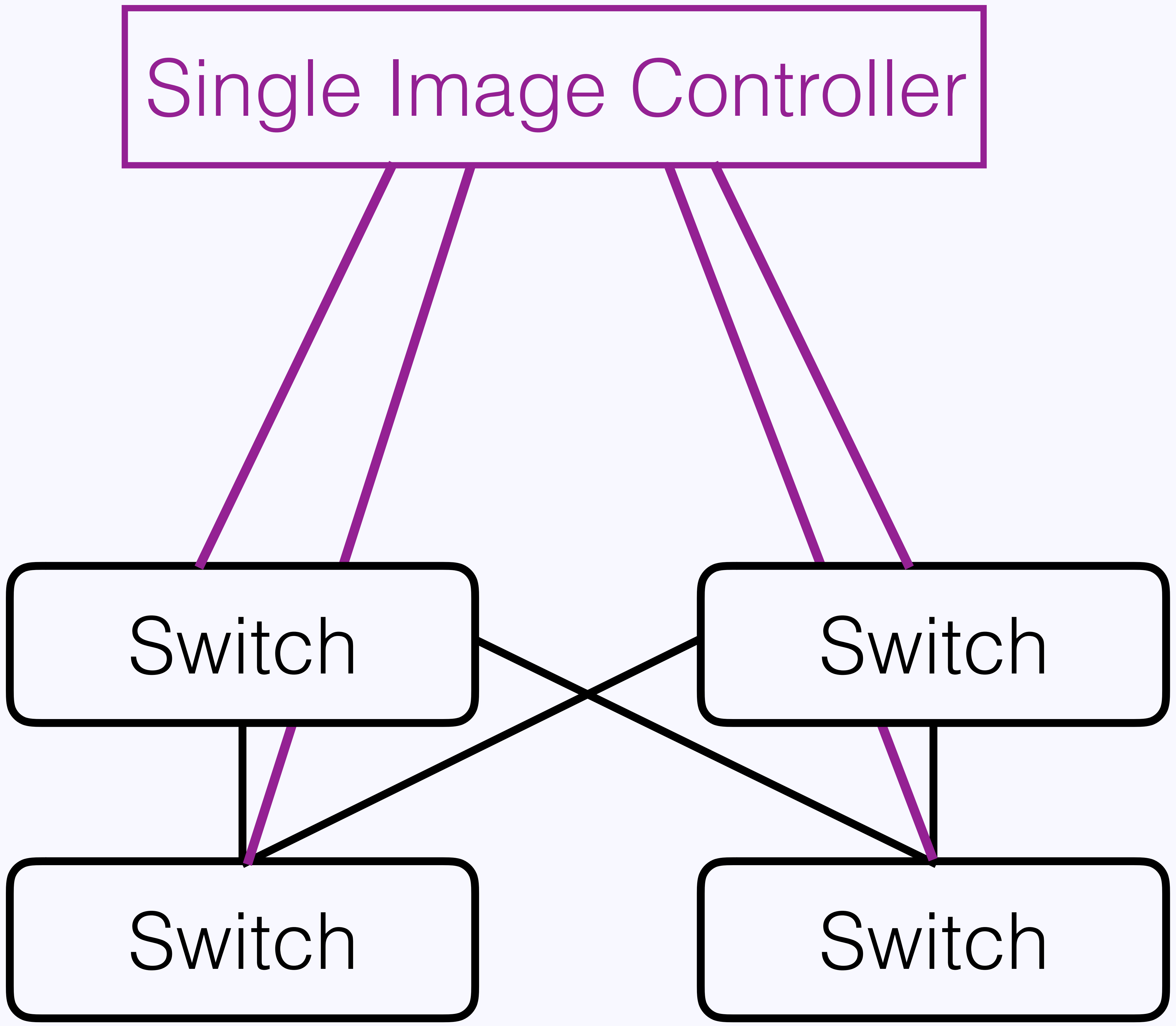
Software Defined Networks



- Forwarding implemented by switches.
- Rules computed by controllers.
- Rules depend on **policy** and **network state.**
- **Policy**: What paths are acceptable?
- **Network State**: Current state of links and switches

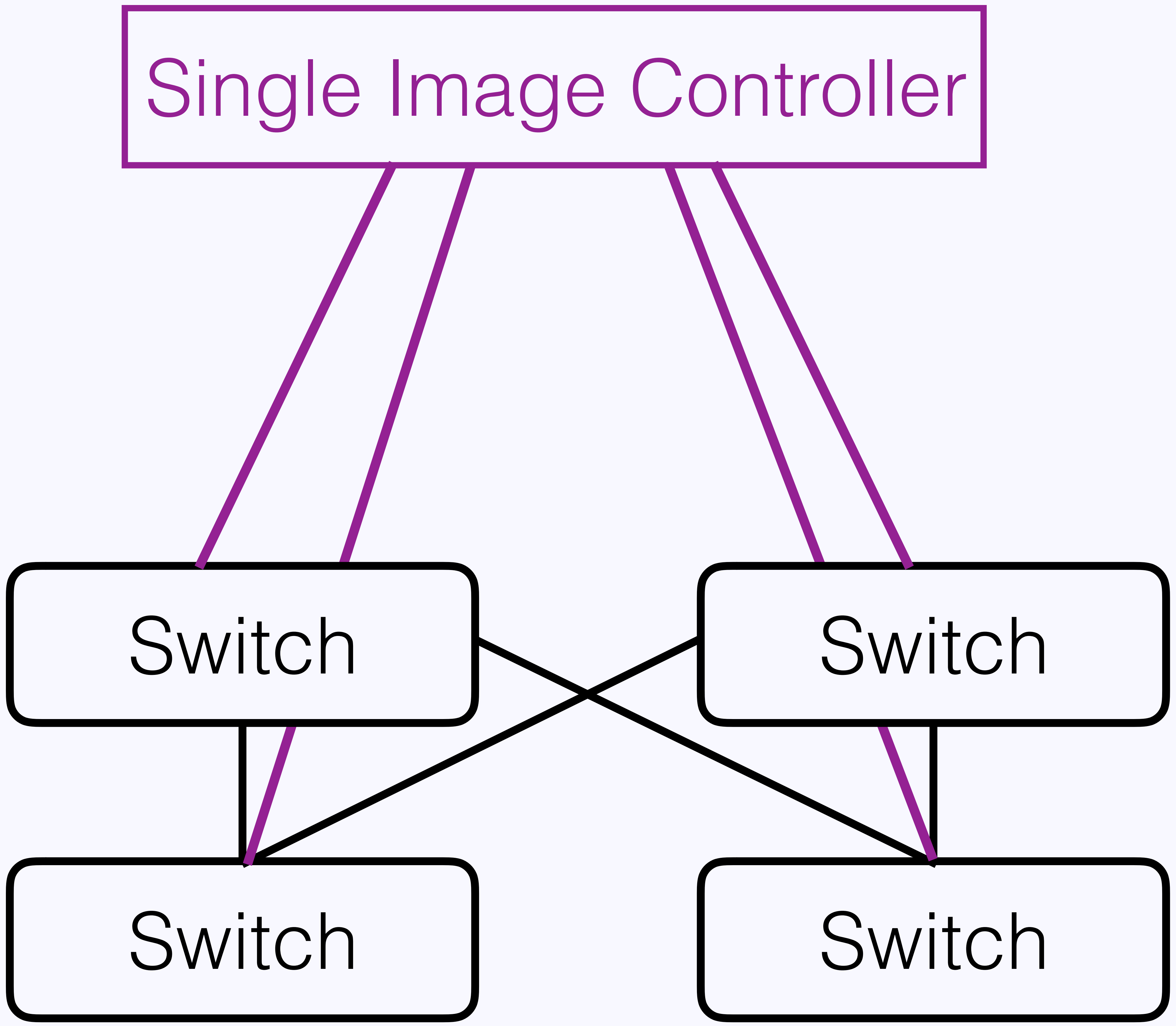
How to build controllers?

Single Image Controllers



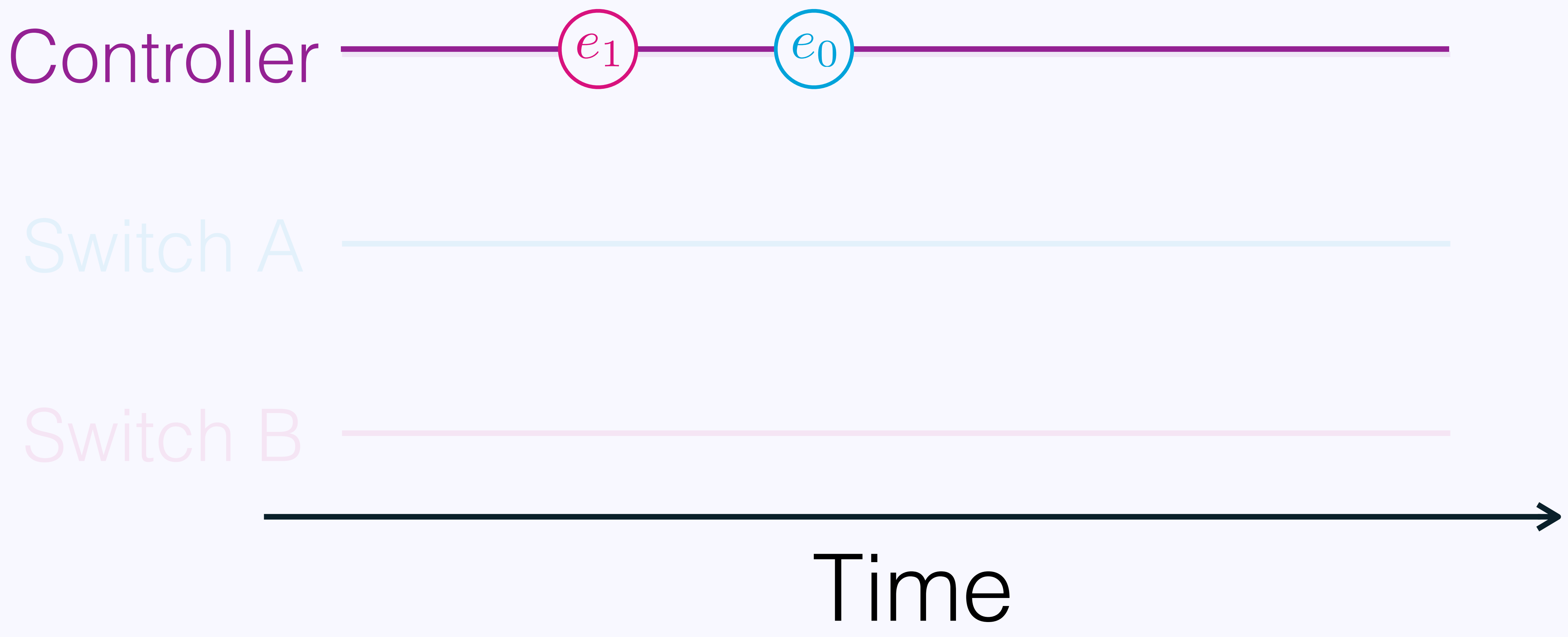
- Controller runs on a single server.

Single Image Controllers



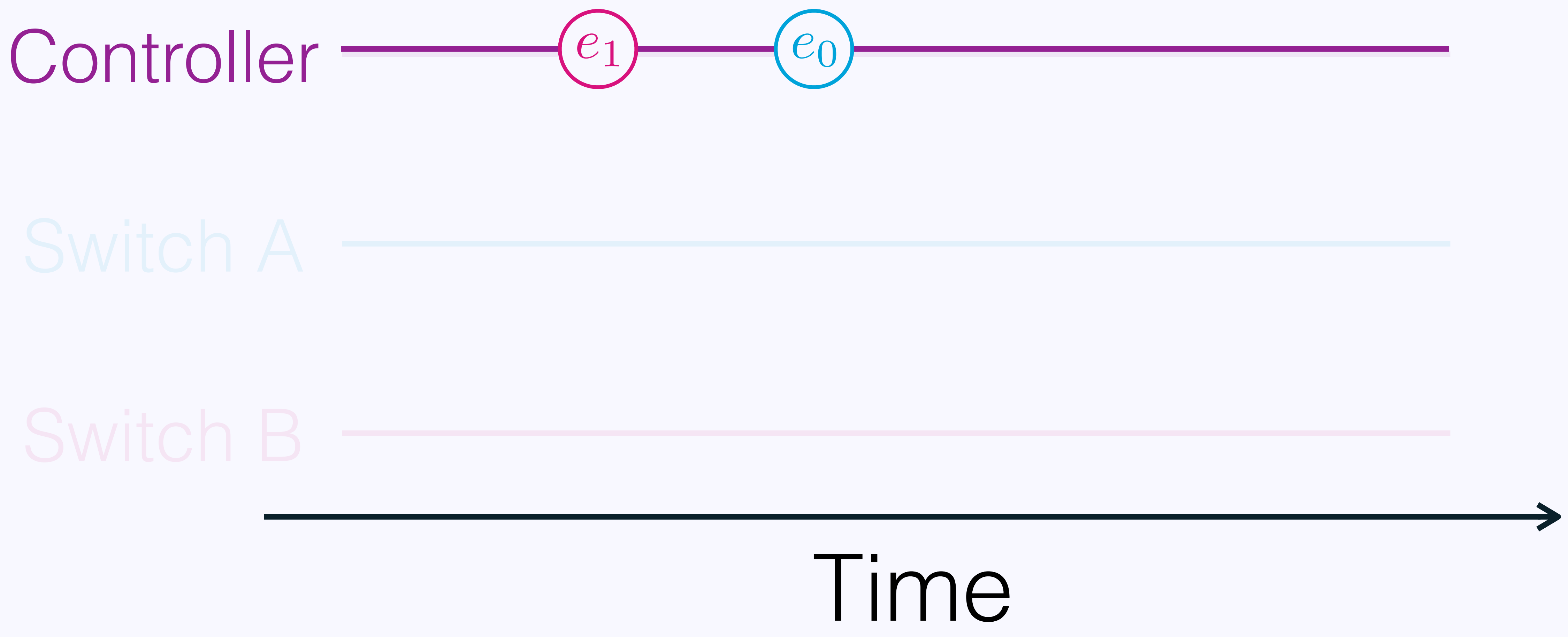
- Controller runs on a single server.
- Examples: Nox, Pox, Ryu, etc.

Single Image Controllers: Assumptions



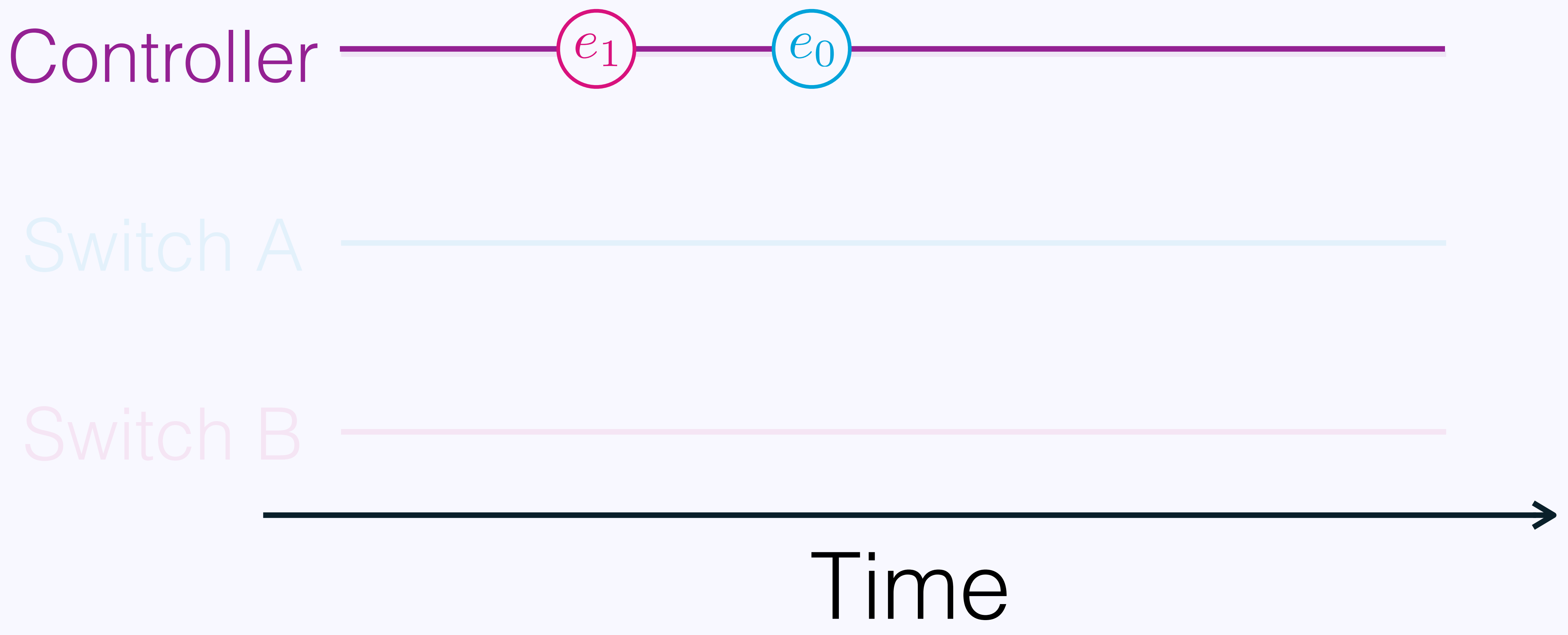
- The controller observes a sequence of events.

Single Image Controllers: Assumptions



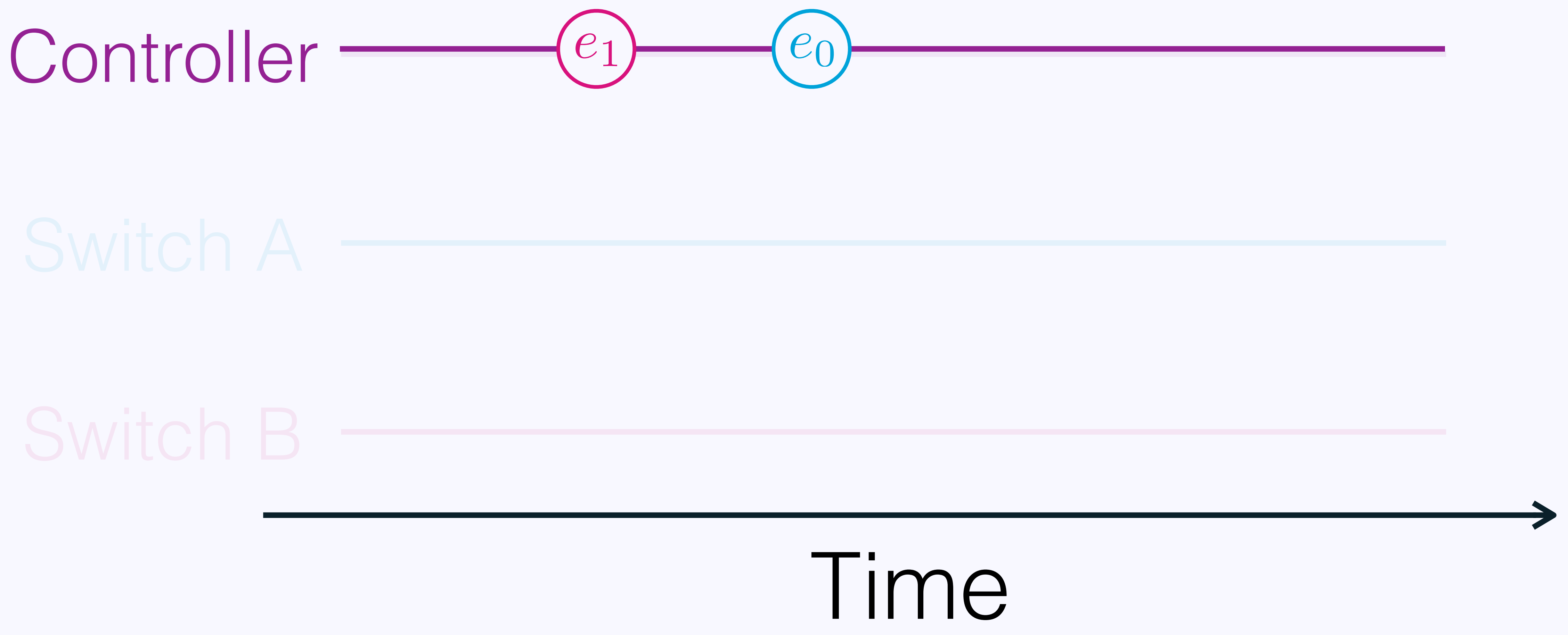
- The controller observes a sequence of events.
    - **Network state** computed using event sequence.

Single Image Controllers: Assumptions



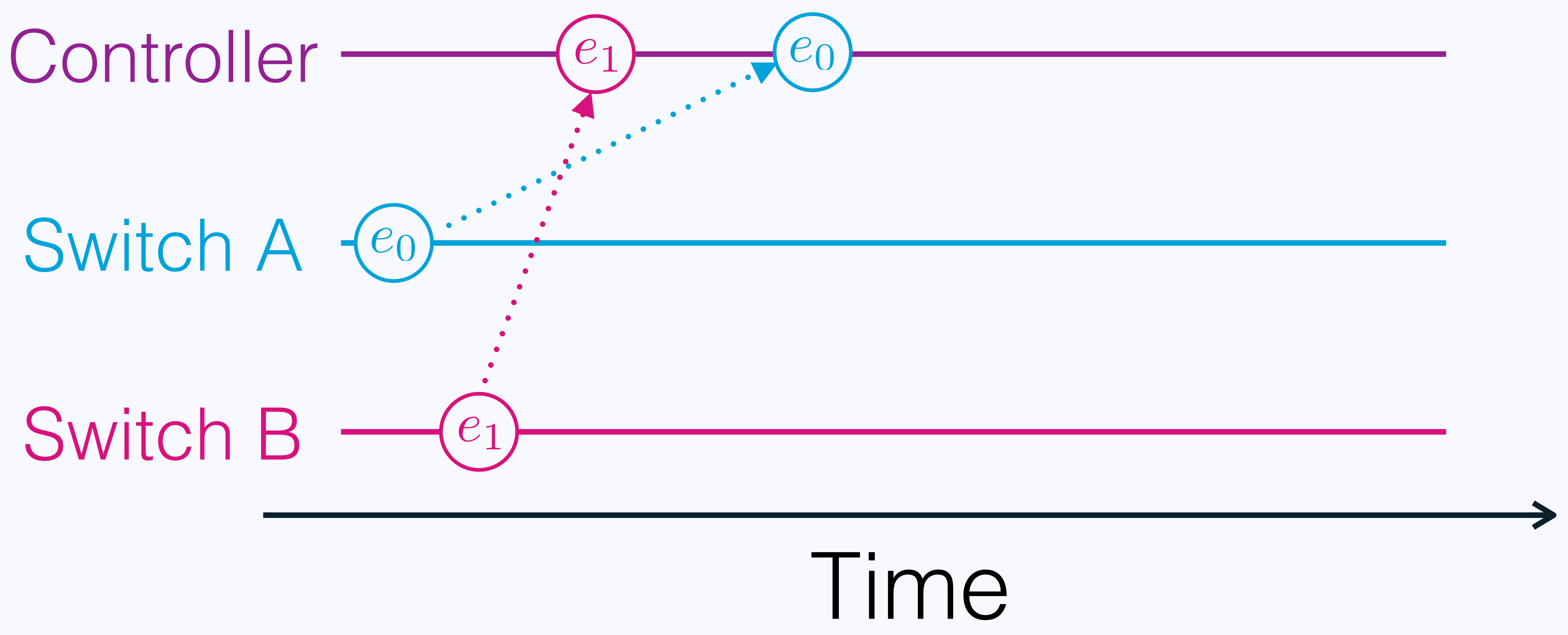
- The controller observes a sequence of events.
  - **Network state** computed using event sequence.
  - Applications react to sequence of events.

Single Image Controllers: Assumptions



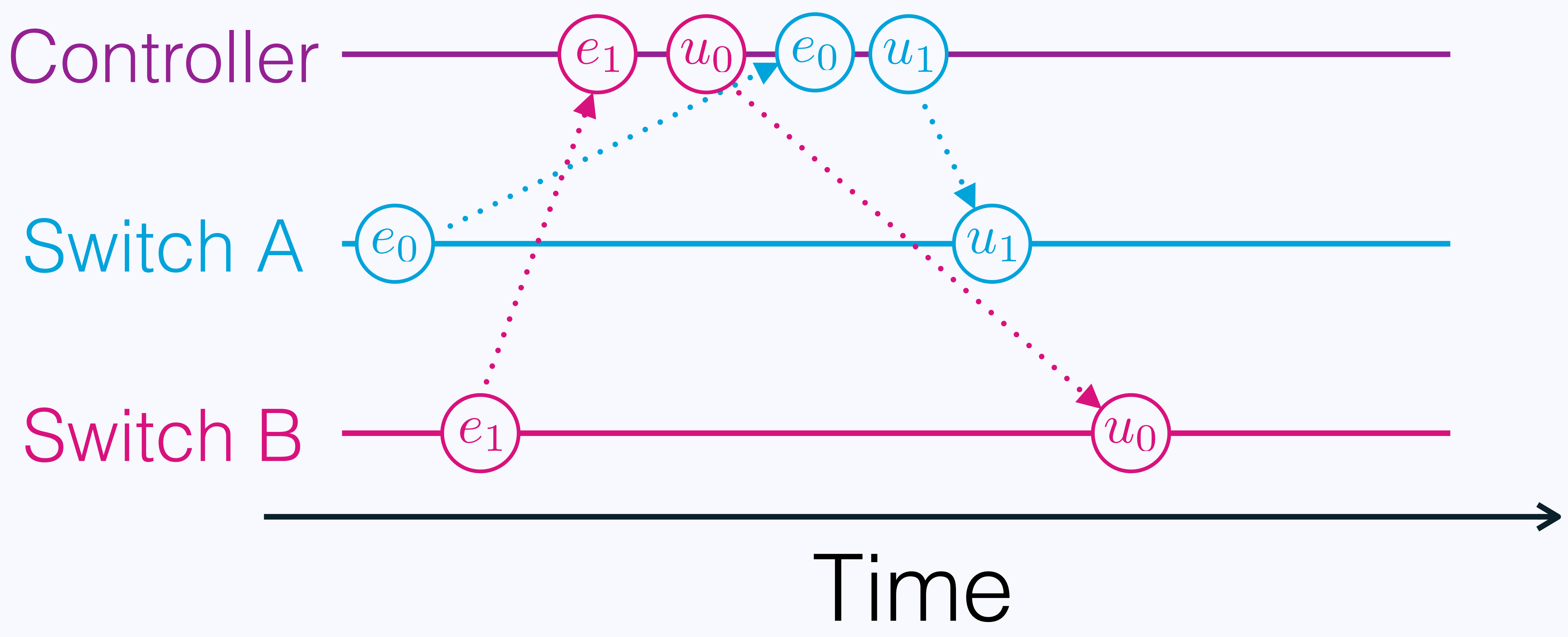
- The controller observes a sequence of events.
  - **Network state** computed using event sequence.
  - Applications react to sequence of events.
- Events and updates sent over TCP channels.

Single Image Controllers: Assumptions



- The controller observes a sequence of events.
  - **Network state** computed using event sequence.
  - Applications react to sequence of events.
- Events and updates sent over TCP channels.
- Events from **different switches** can be **reordered**.

Single Image Controllers: Assumptions



- The controller observes a sequence of events.
  - **Network state** computed using event sequence.
  - Applications react to sequence of events.
- Events and updates sent over TCP channels.
- Events from **different switches** can be **reordered**.
- Updates to different switches can be **reordered**.

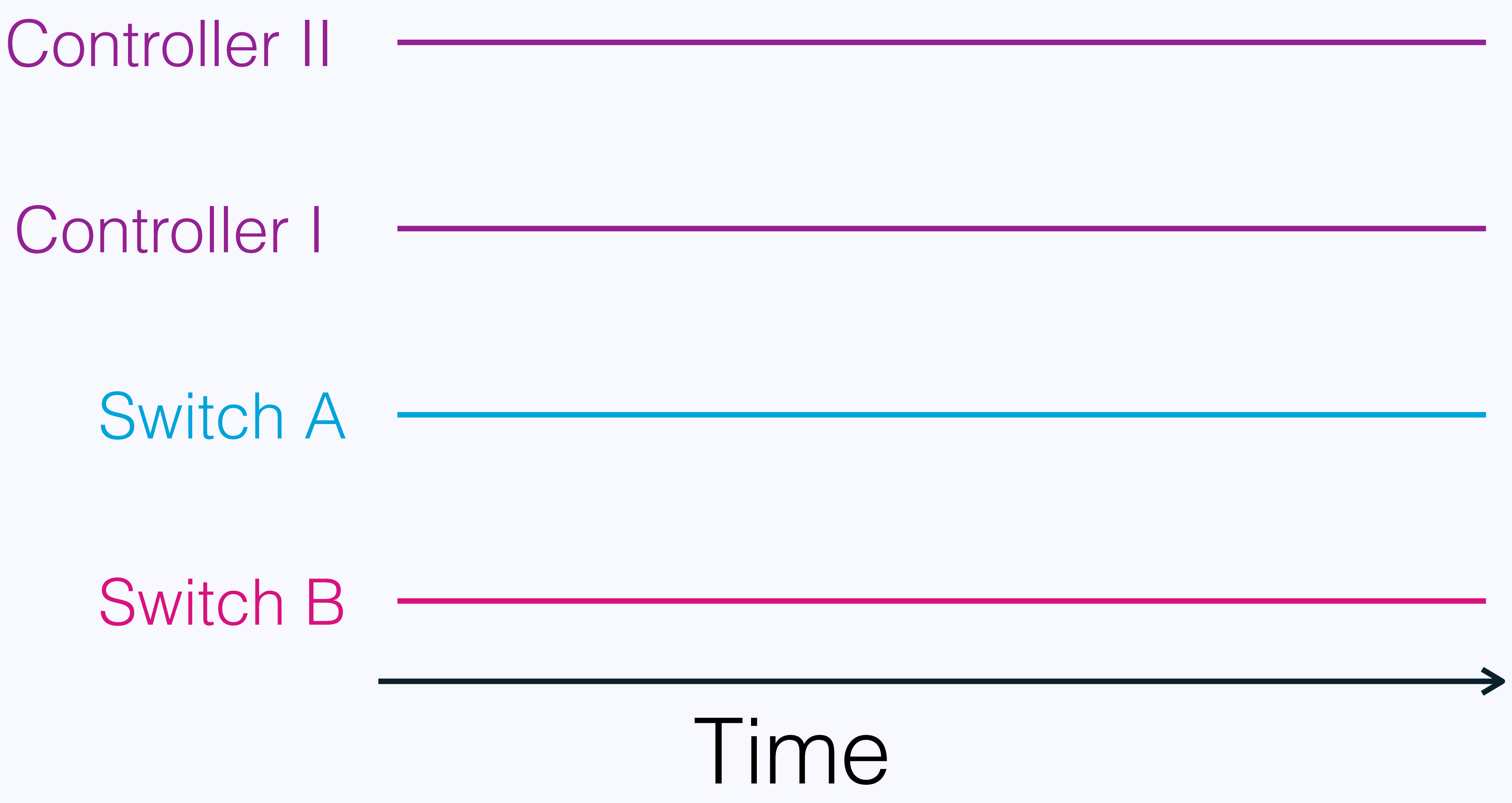
How to handle controller failures, scale controllers, etc.?

How to handle controller failures, scale controllers, etc.?

Move to **distributed controllers**.

How to build distributed controllers?

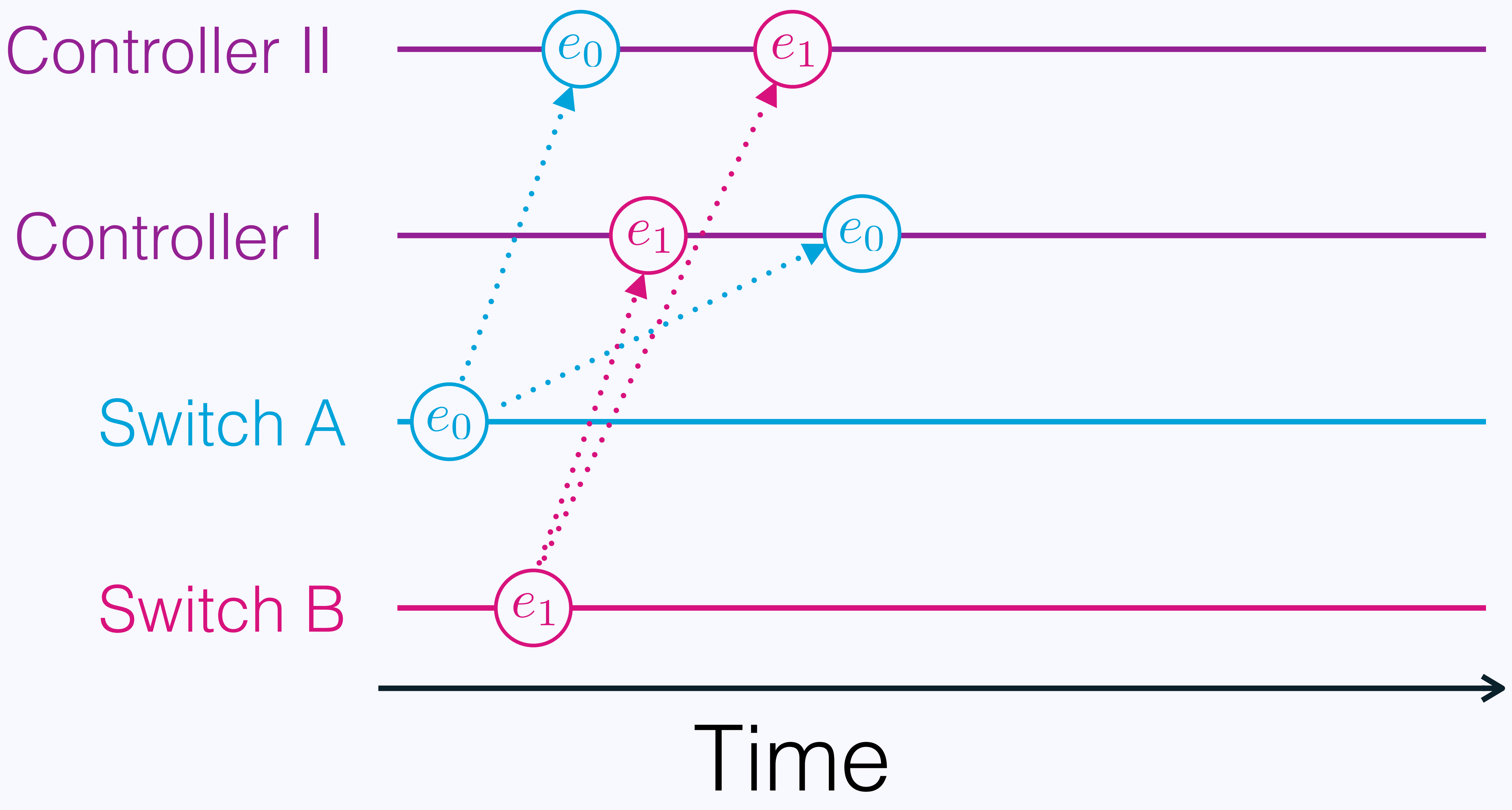
Why is this Harder?



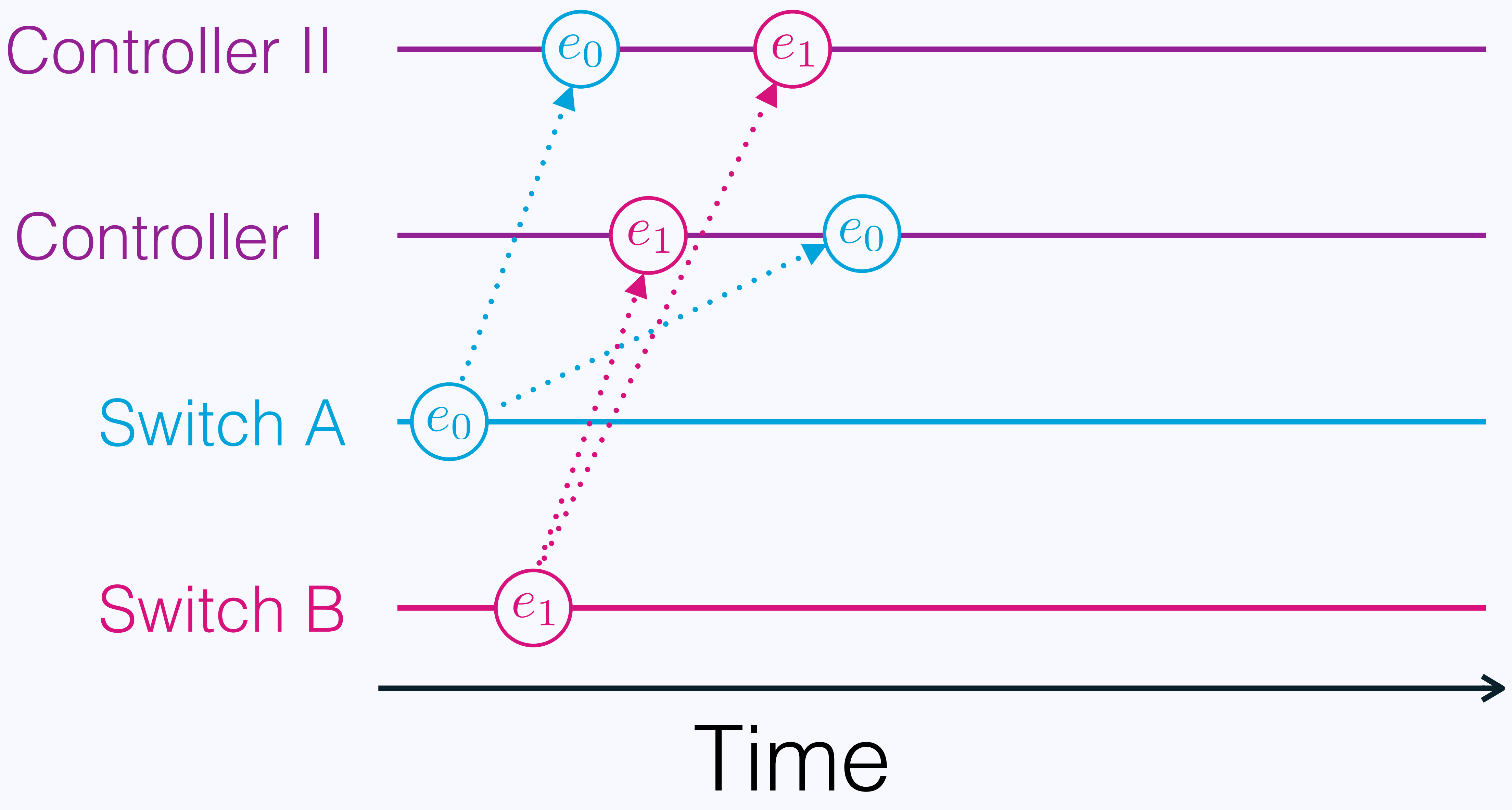
- Event **ordering** can differ across controllers.

Why is this Harder?

- Event **ordering** can differ across controllers.

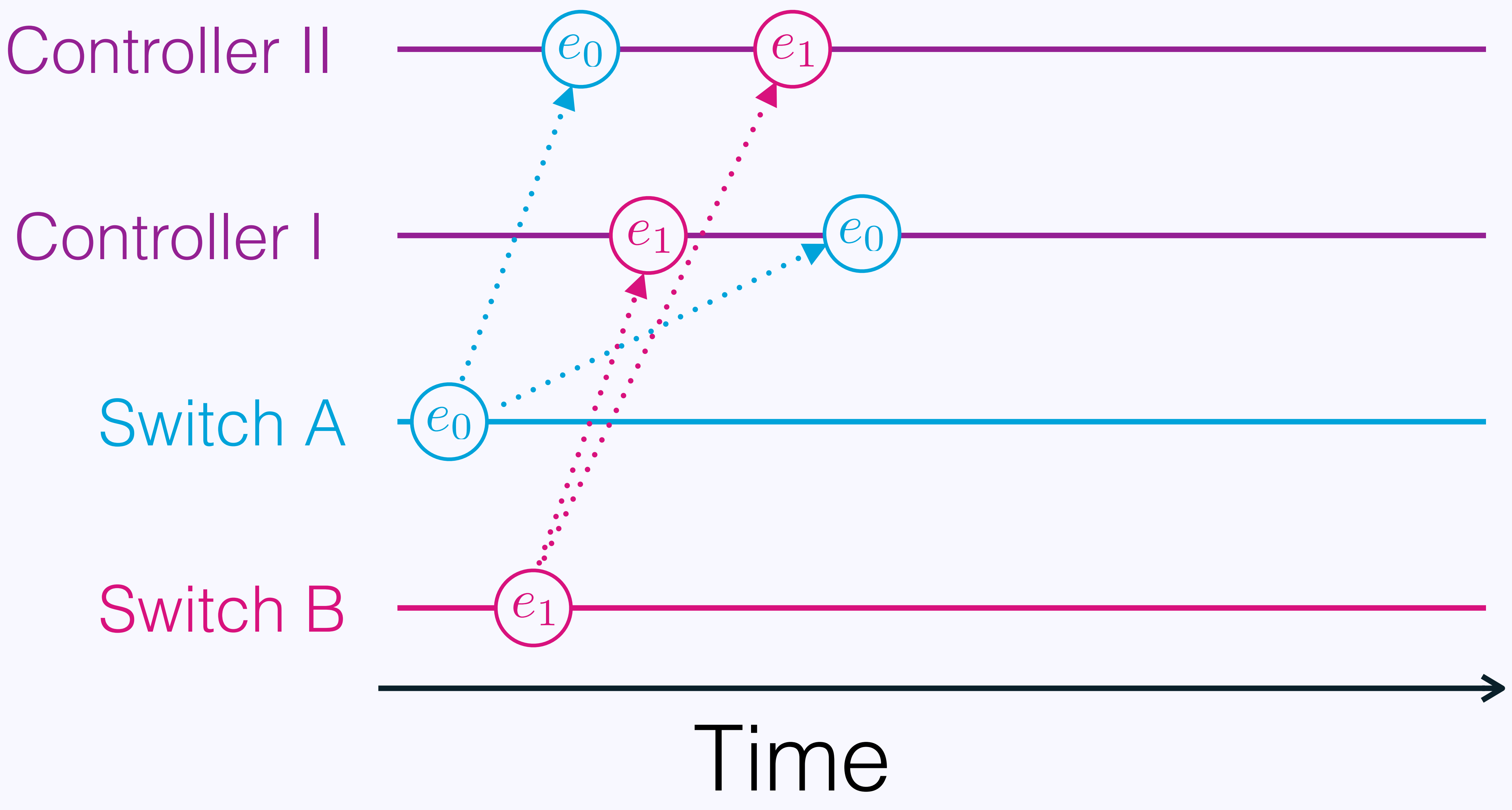


Why is this Harder?



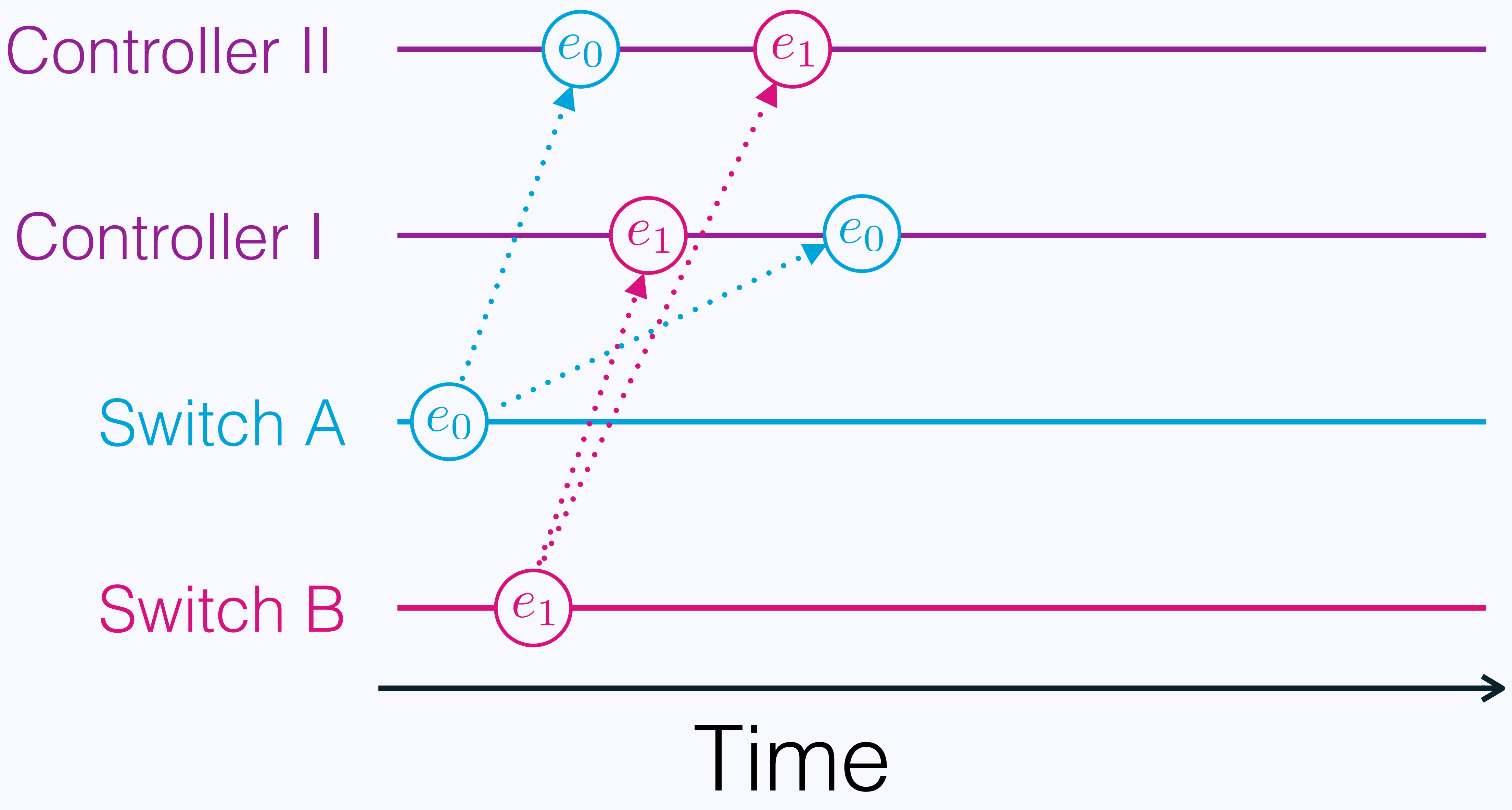
- Event **ordering** can differ across controllers.
    - Rules must converge despite this reordering.

Why is this Harder?



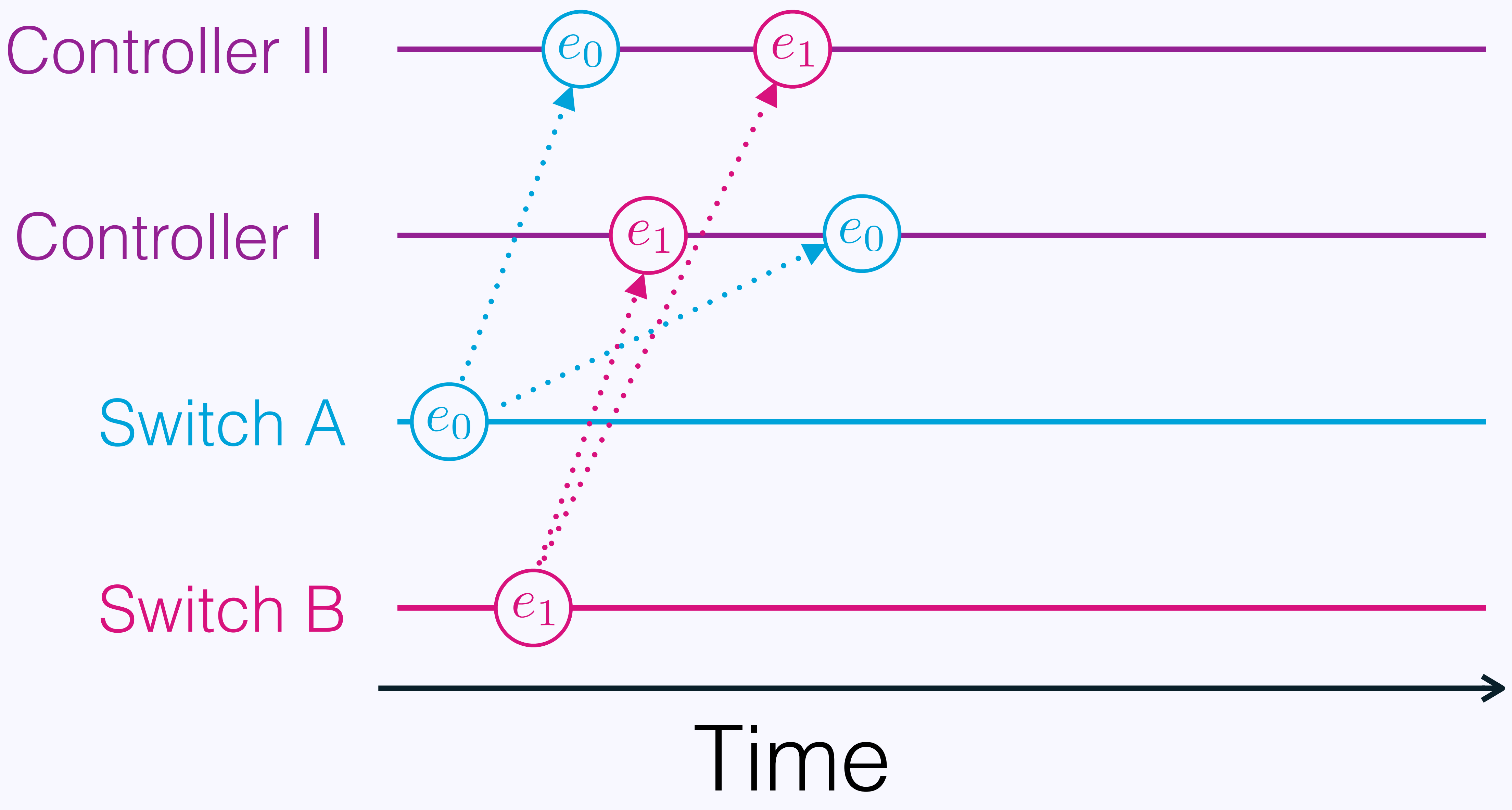
- Event **ordering** can differ across controllers.
  - Rules must converge despite this reordering.
- Two ways to handle this

Why is this Harder?



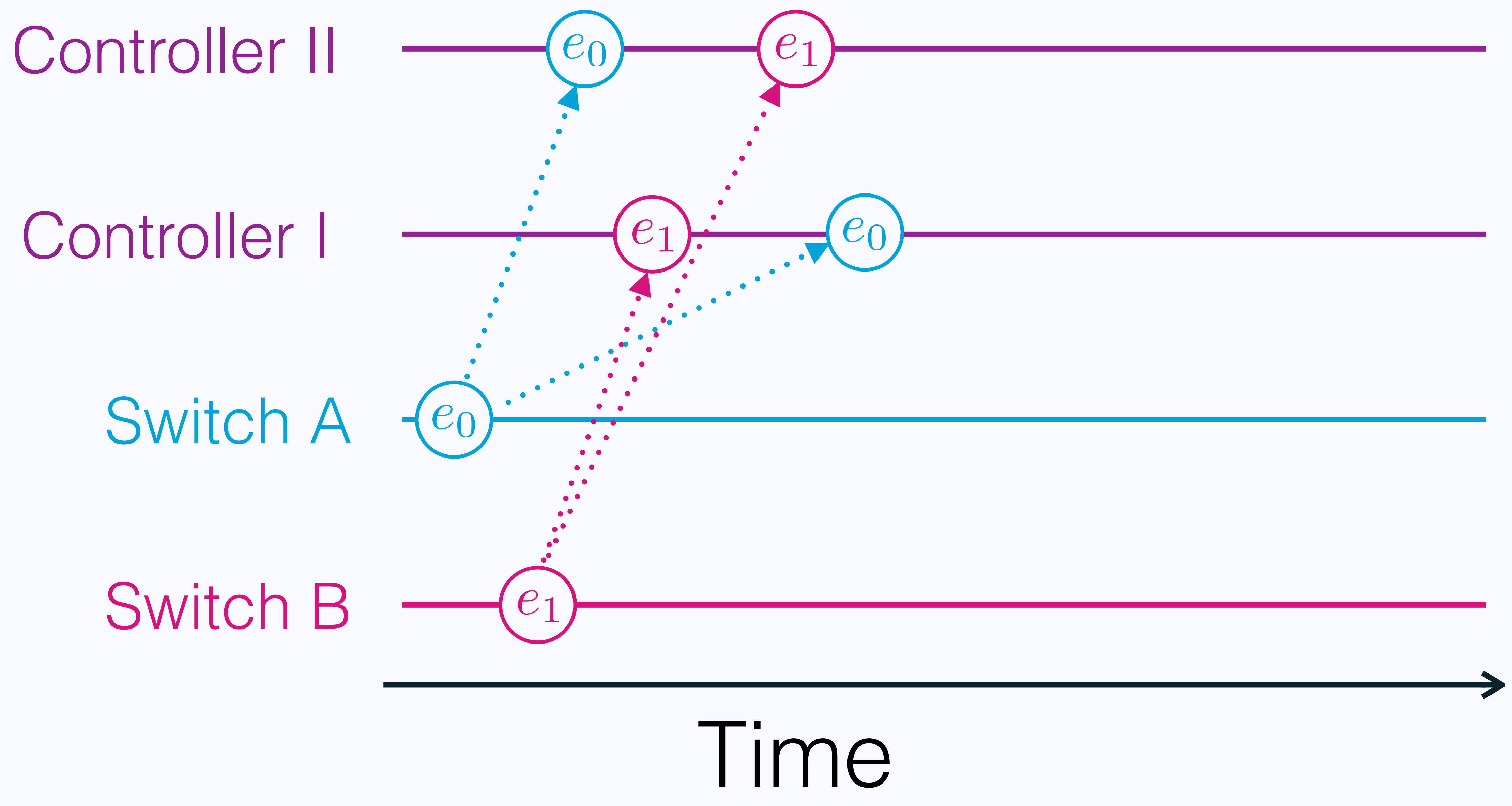
- Event **ordering** can differ across controllers.
  - Rules must converge despite this reordering.
- Two ways to handle this
  - Algorithms are correct despite reordering.

Why is this Harder?



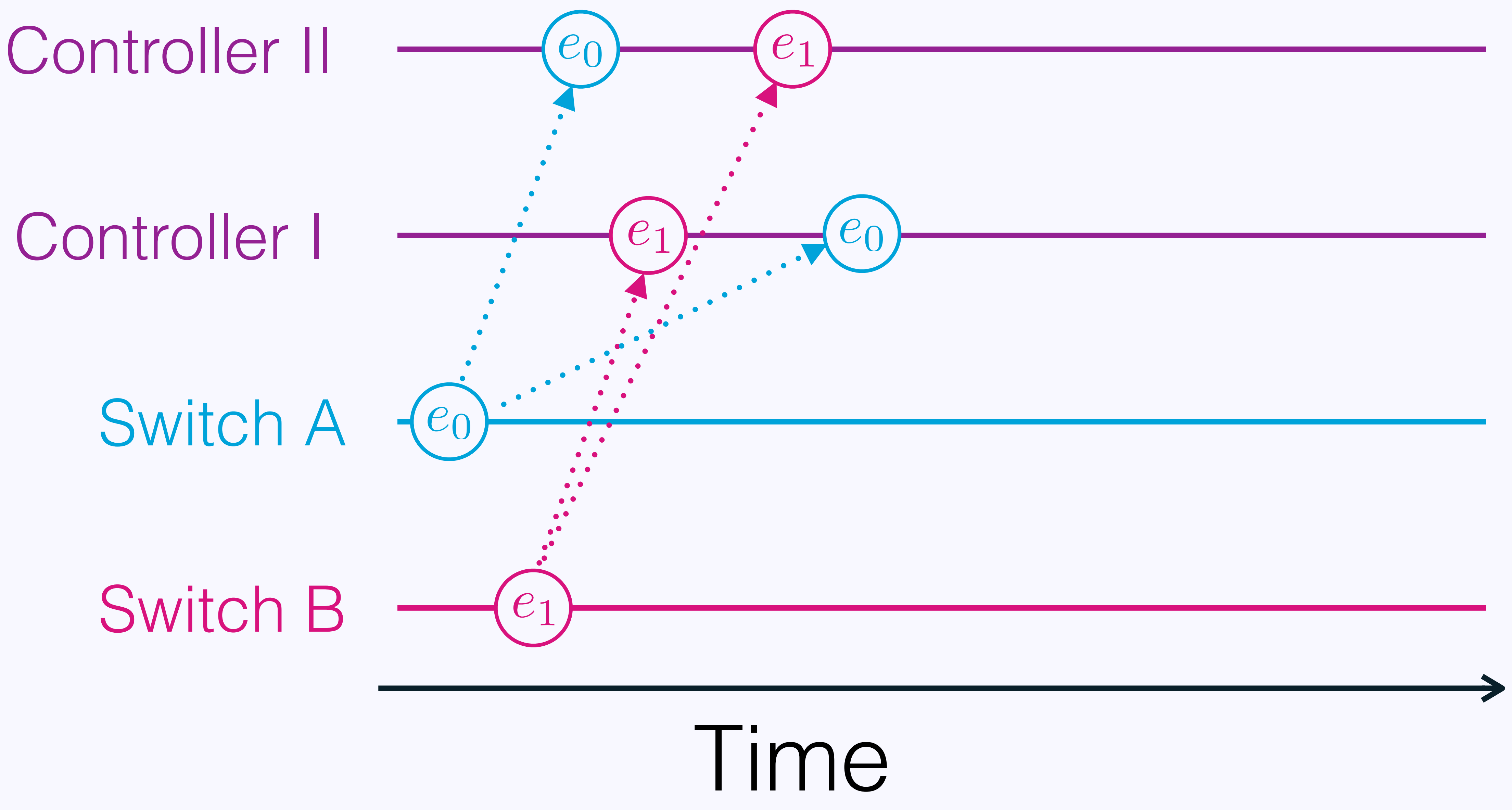
- Event **ordering** can differ across controllers.
  - Rules must converge despite this reordering.
- Two ways to handle this
  - Algorithms are correct despite reordering.
  - Mechanisms so controllers agree on ordering.

Why is this Harder?



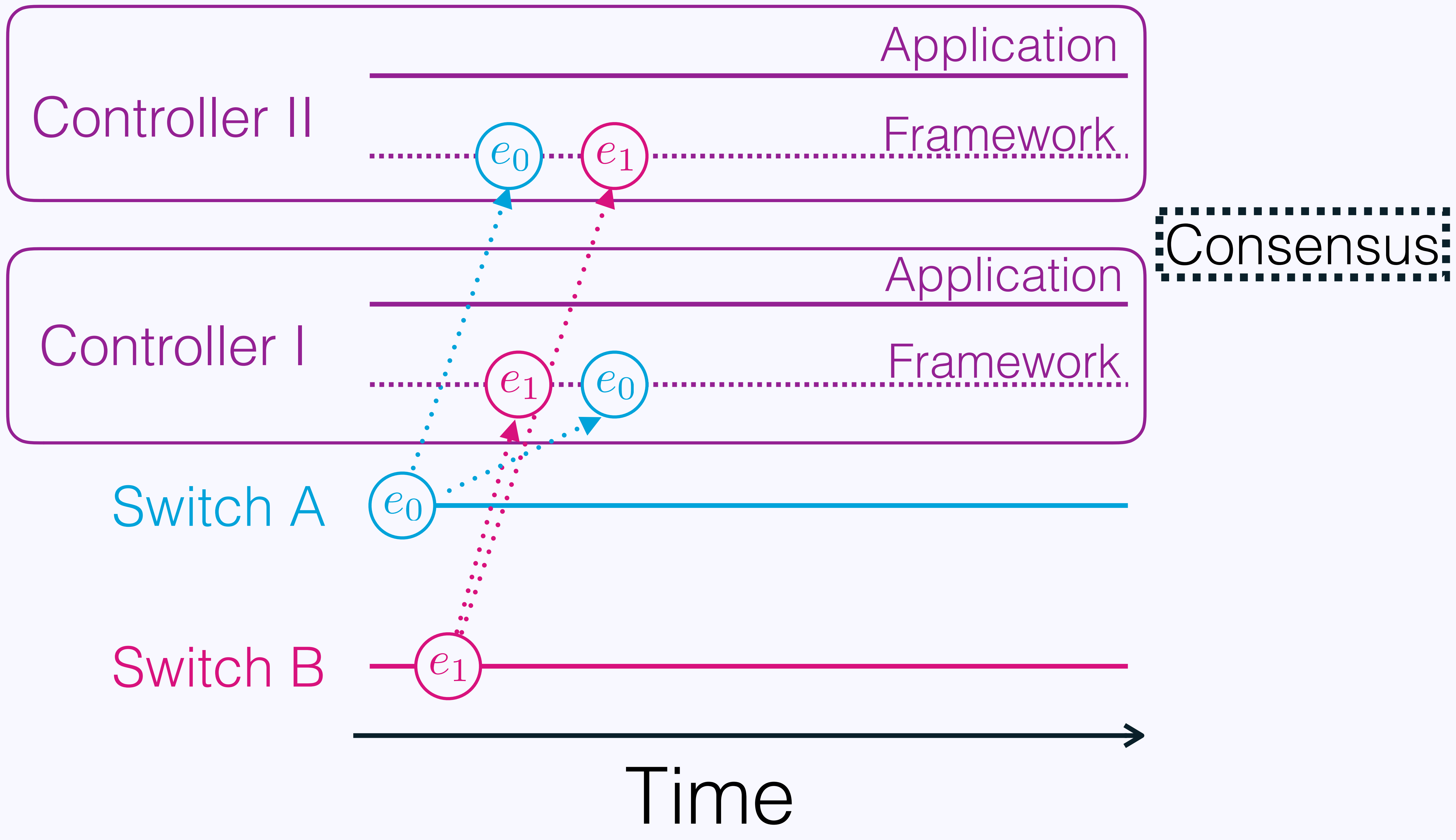
- Event **ordering** can differ across controllers.
  - Rules must converge despite this reordering.
- Two ways to handle this
  - Algorithms are correct despite reordering.
  - Mechanisms so controllers agree on ordering.
- Rely on **ordering mechanisms** for generality.

Why is this Harder?



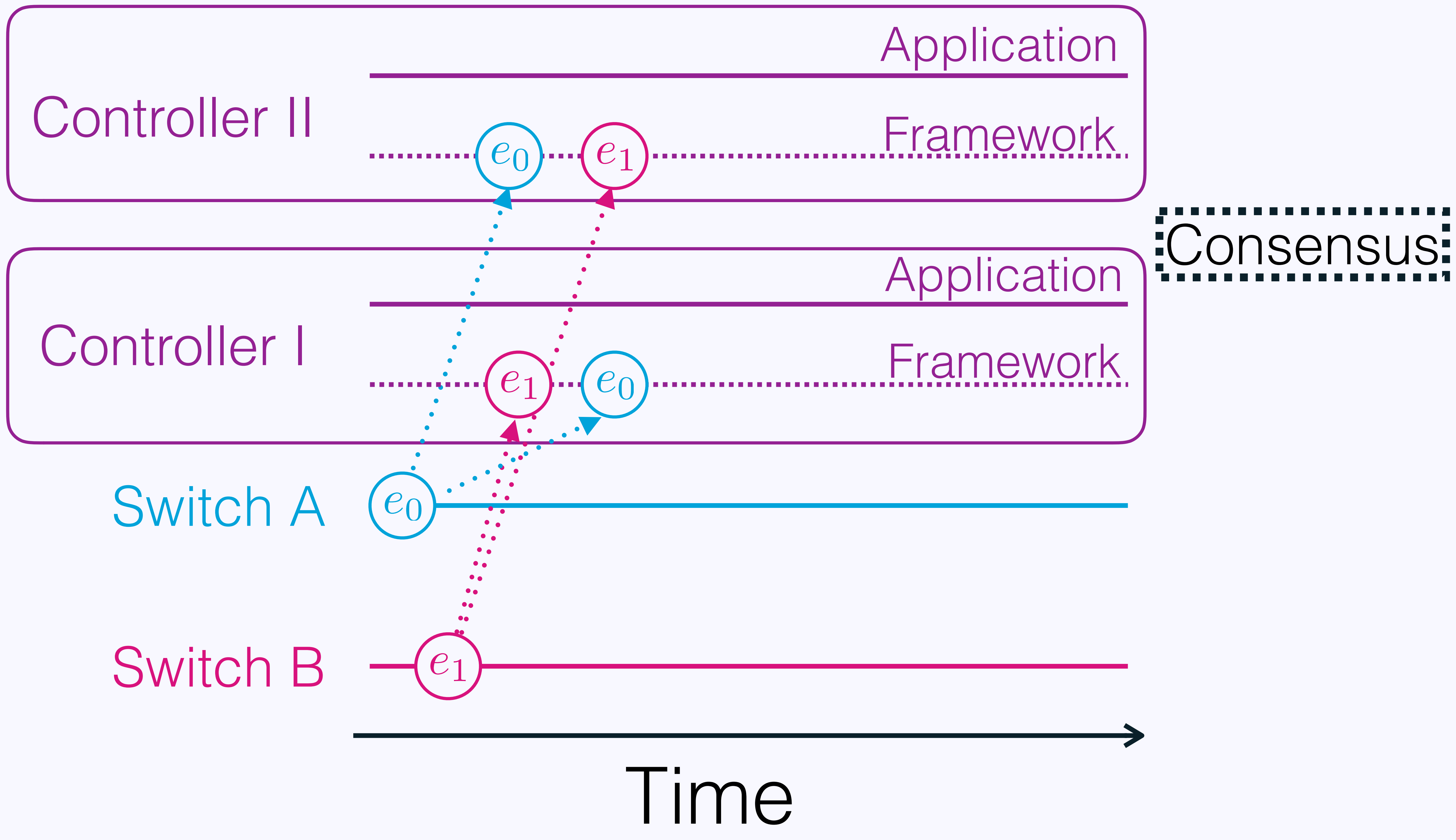
- Event **ordering** can differ across controllers.
  - Rules must converge despite this reordering.
- Two ways to handle this
  - Algorithms are correct despite reordering.
  - Mechanisms so controllers agree on ordering.
- Rely on **ordering mechanisms** for generality.
- How to implement event ordering?

Canonical Solution: Consensus



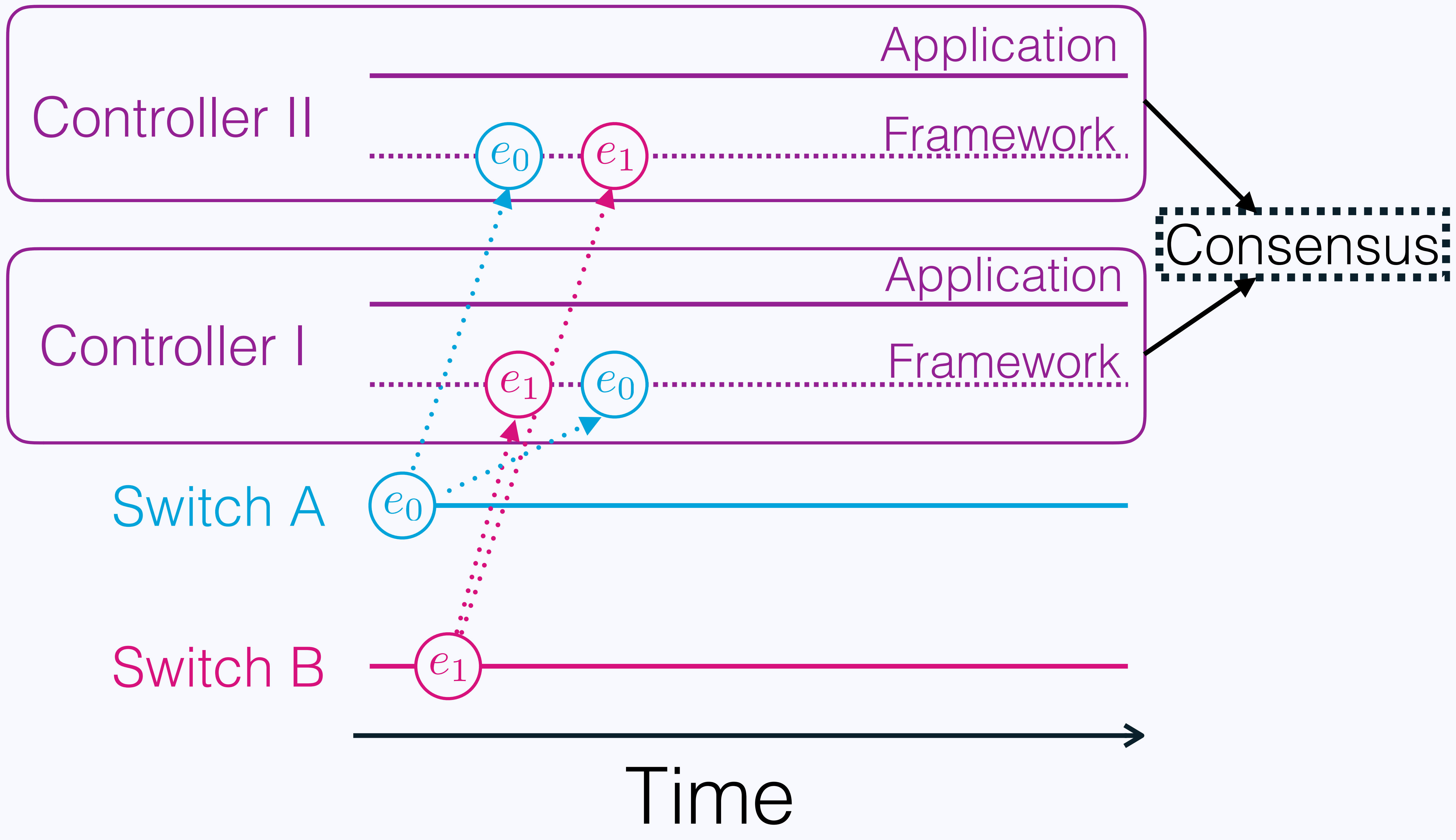
- Consensus: Protocol to get agreement on a value.

Canonical Solution: Consensus



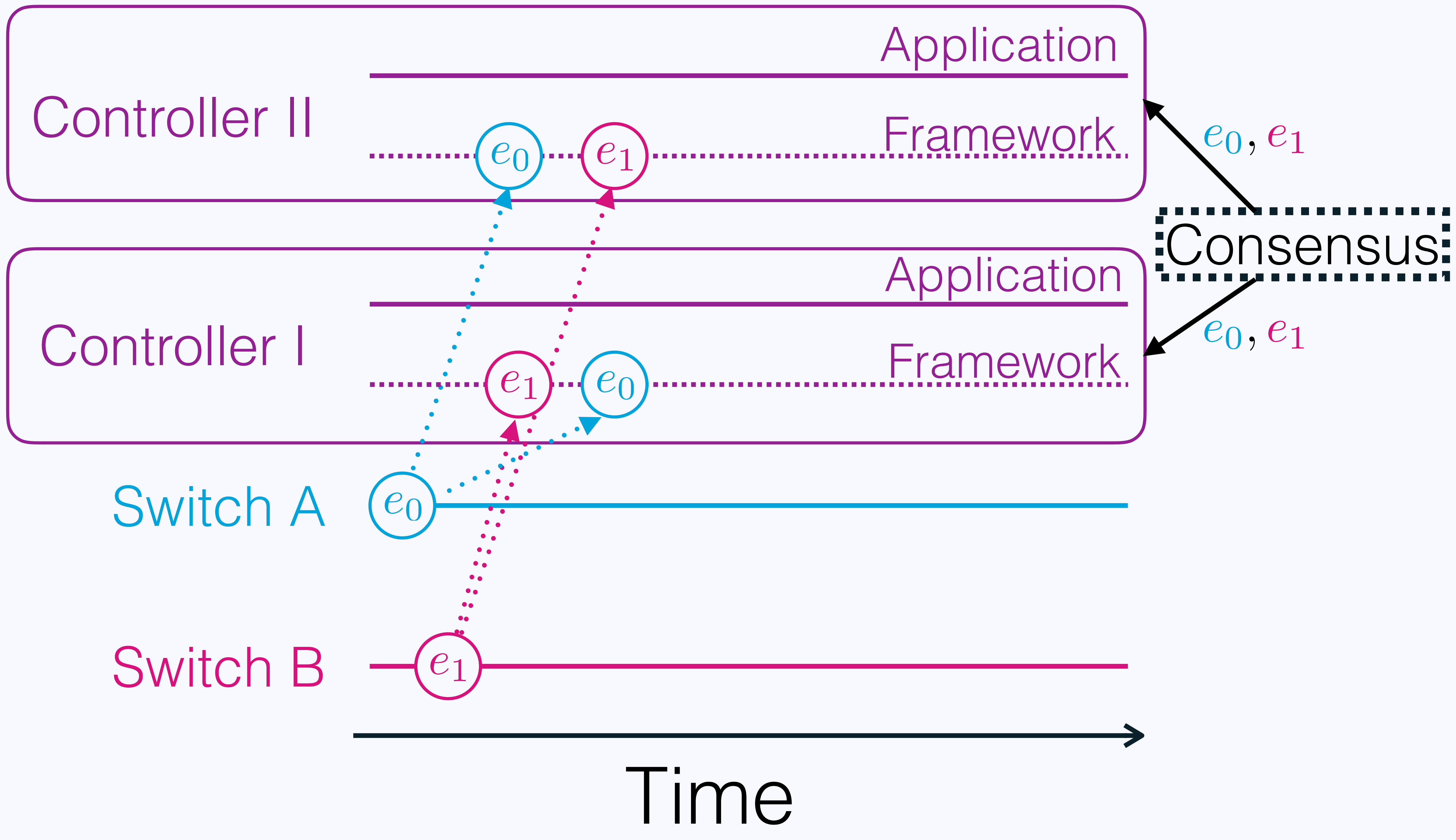
- Consensus: Protocol to get agreement on a value.
- Rely on **consensus** to agree on event order.

Canonical Solution: Consensus



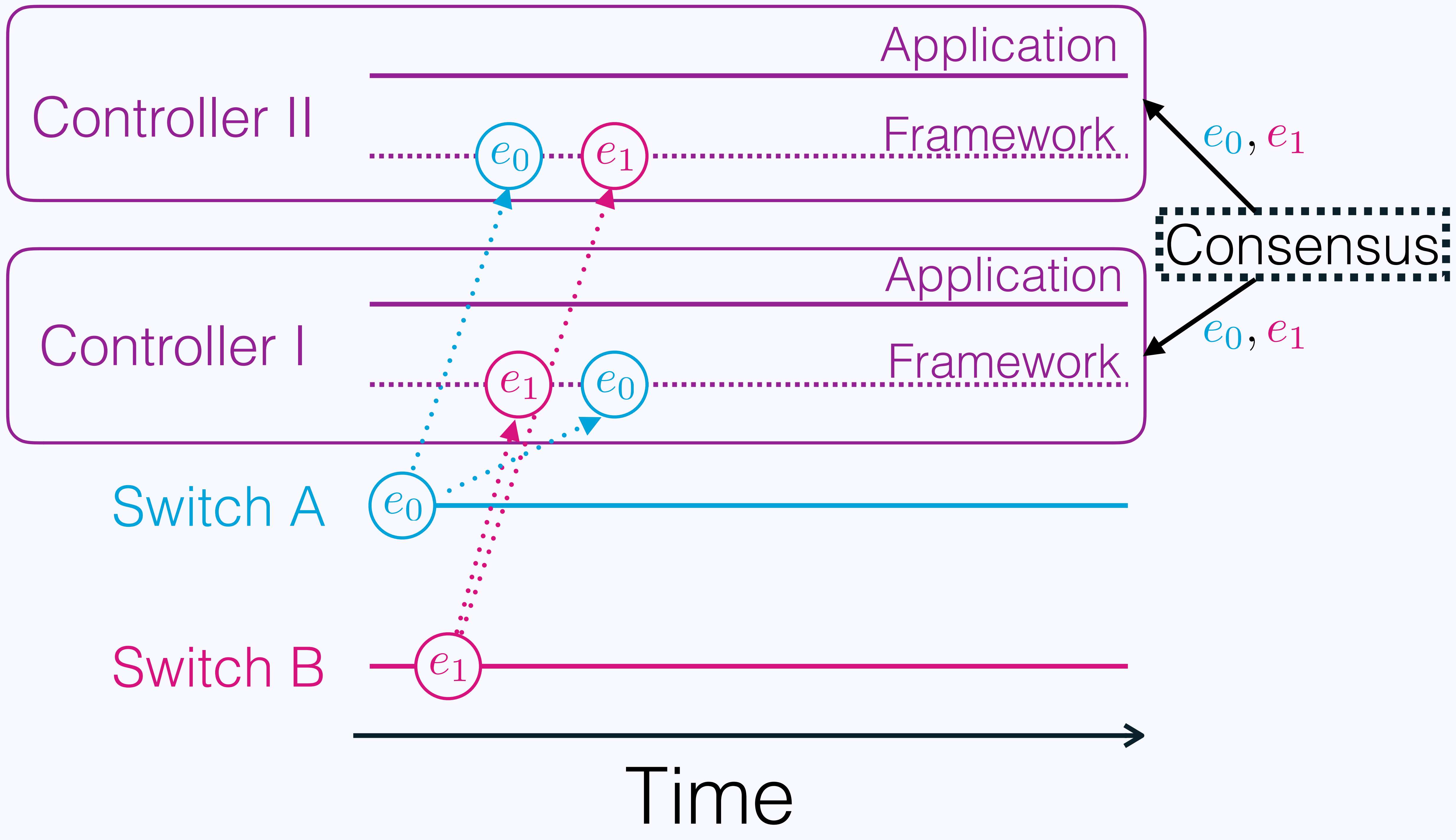
- Consensus: Protocol to get agreement on a value.
- Rely on **consensus** to agree on event order.

Canonical Solution: Consensus



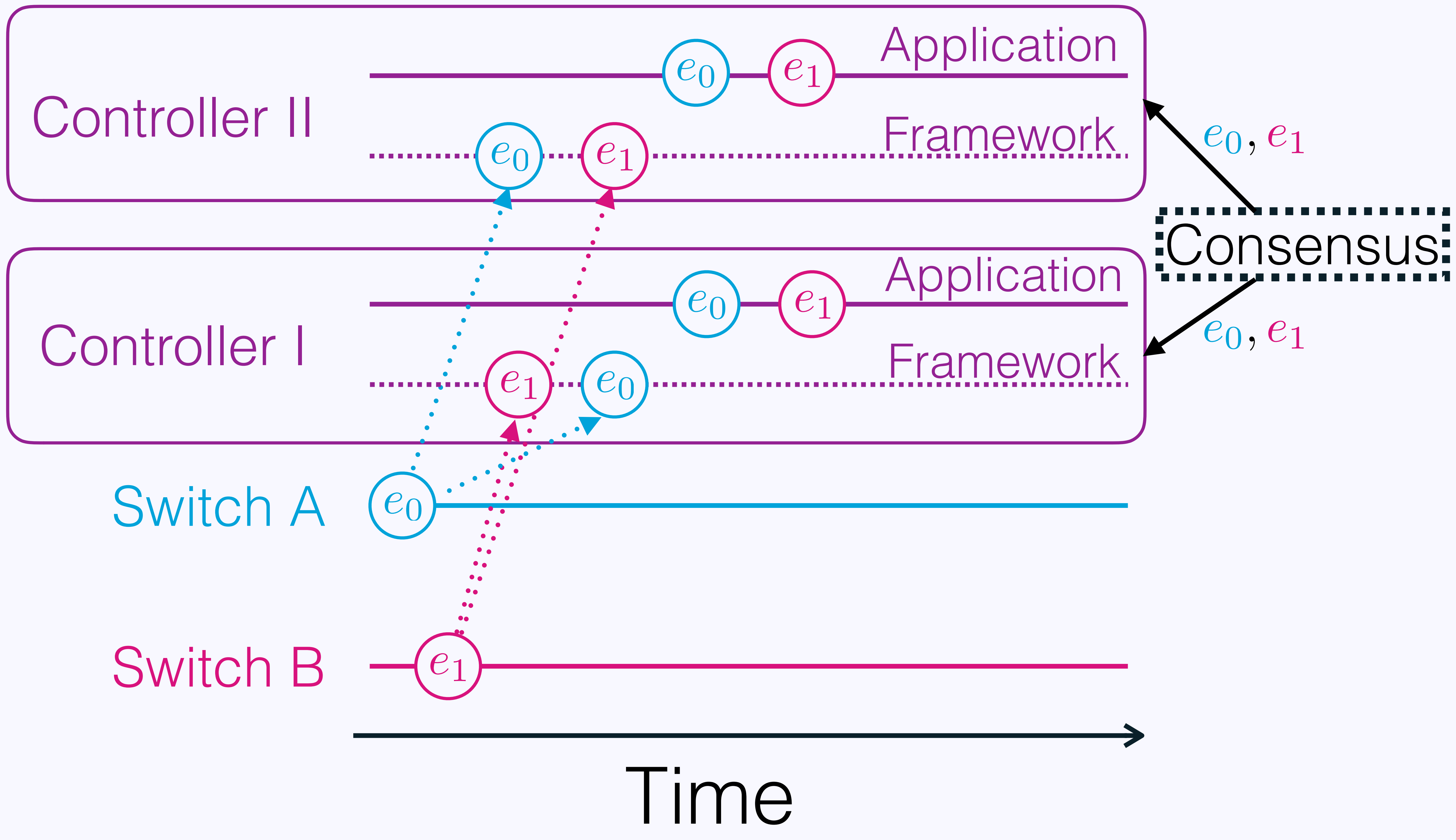
- Consensus: Protocol to get agreement on a value.
- Rely on **consensus** to agree on event order.

Canonical Solution: Consensus



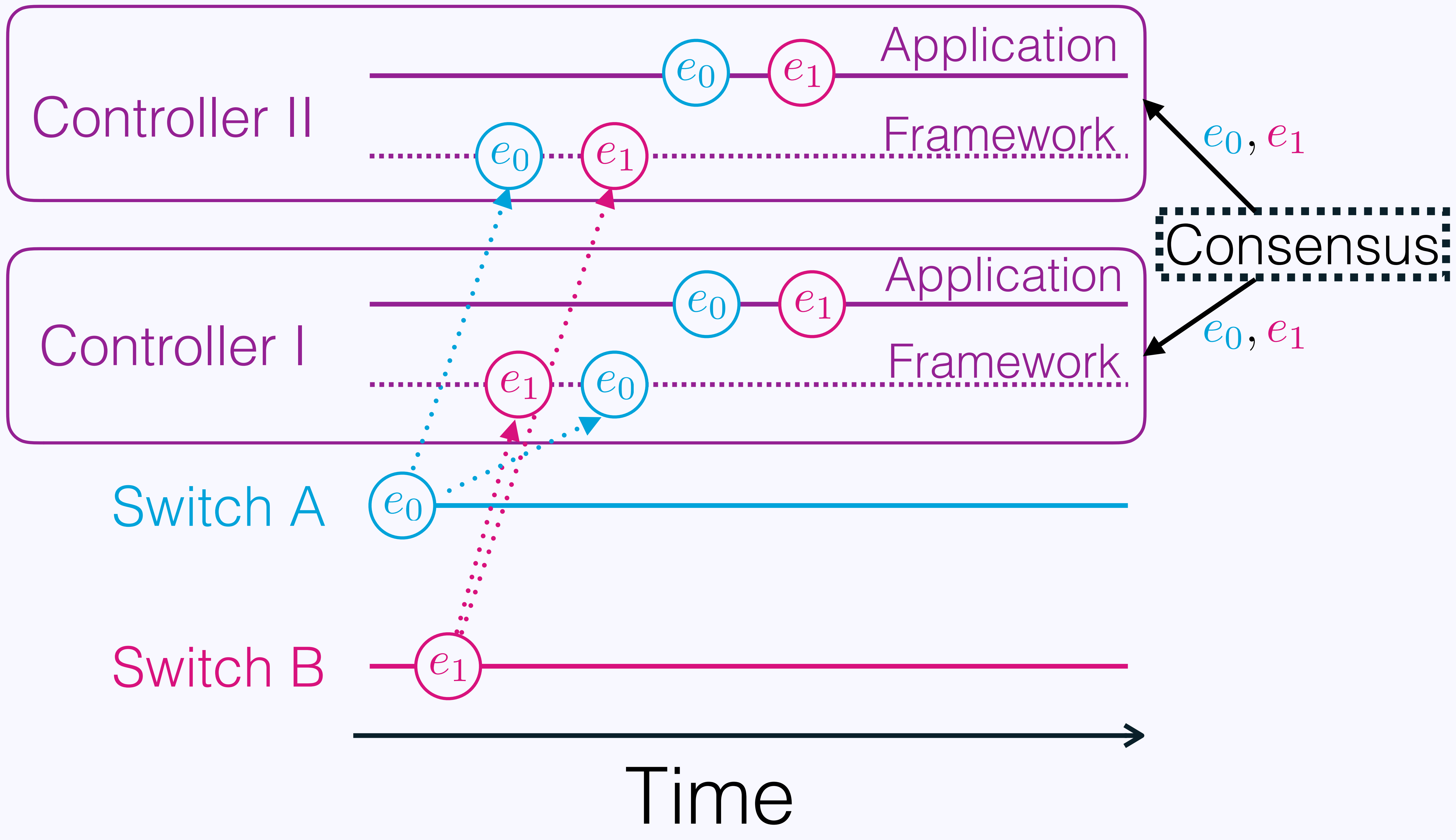
- Consensus: Protocol to get agreement on a value.
- Rely on **consensus** to agree on event order.
    - Applications always see events in agreed order.

Canonical Solution: Consensus



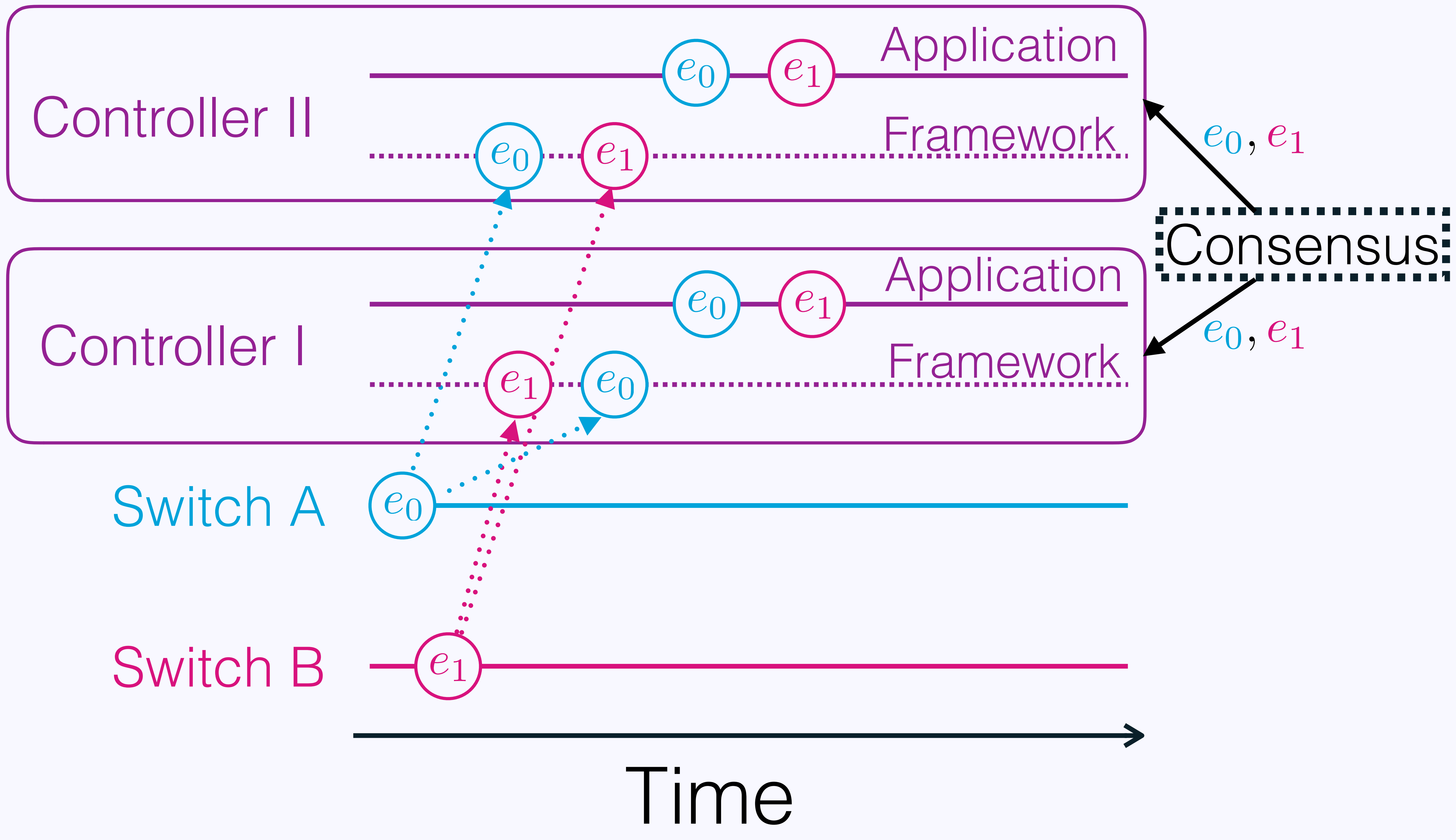
- Consensus: Protocol to get agreement on a value.
- Rely on **consensus** to agree on event order.
  - Applications always see events in agreed order.

Canonical Solution: Consensus



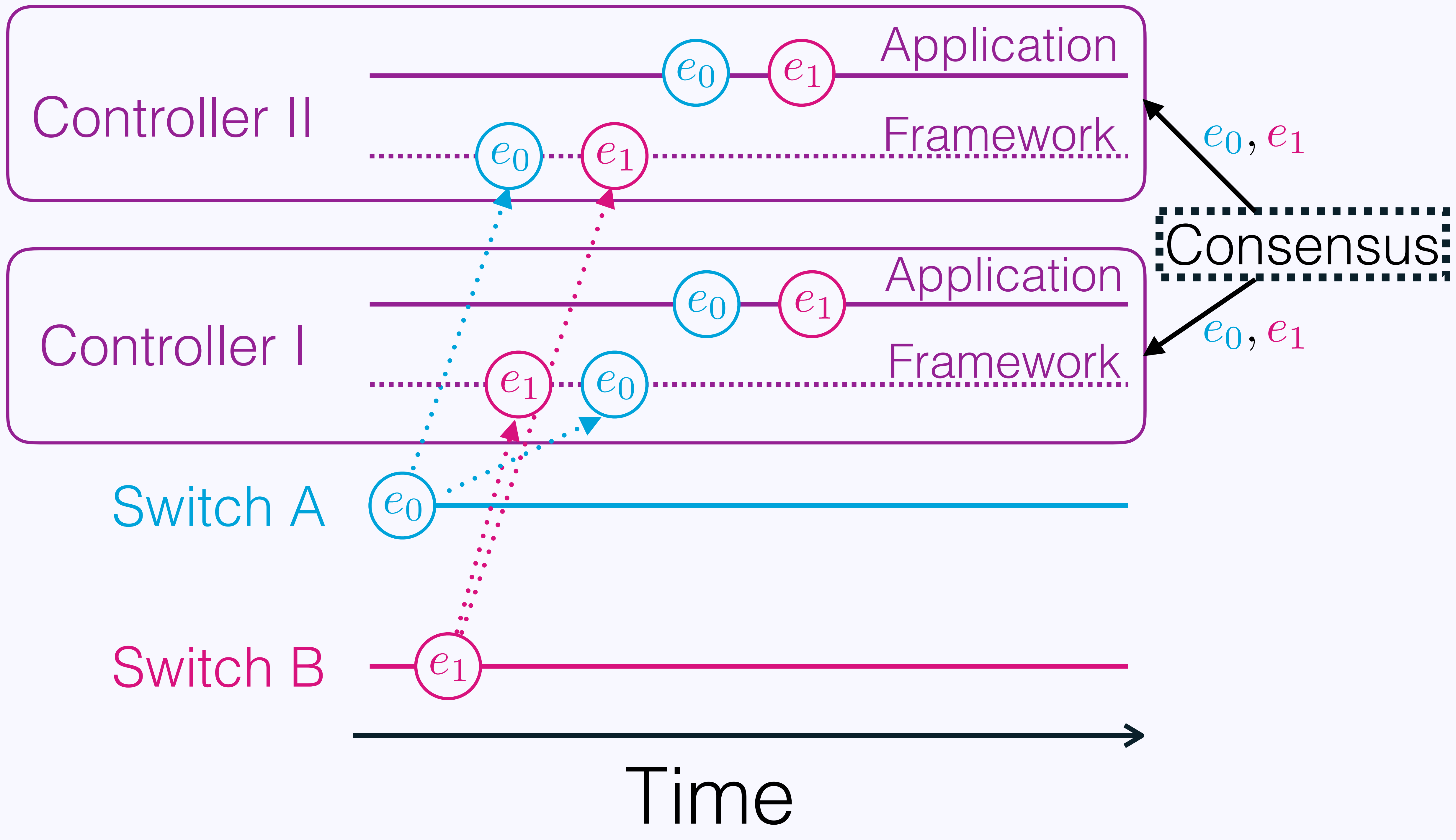
- Consensus: Protocol to get agreement on a value.
- Rely on **consensus** to agree on event order.
  - Applications always see events in agreed order.
- Can use same algorithms as single image controller.

Canonical Solution: Consensus



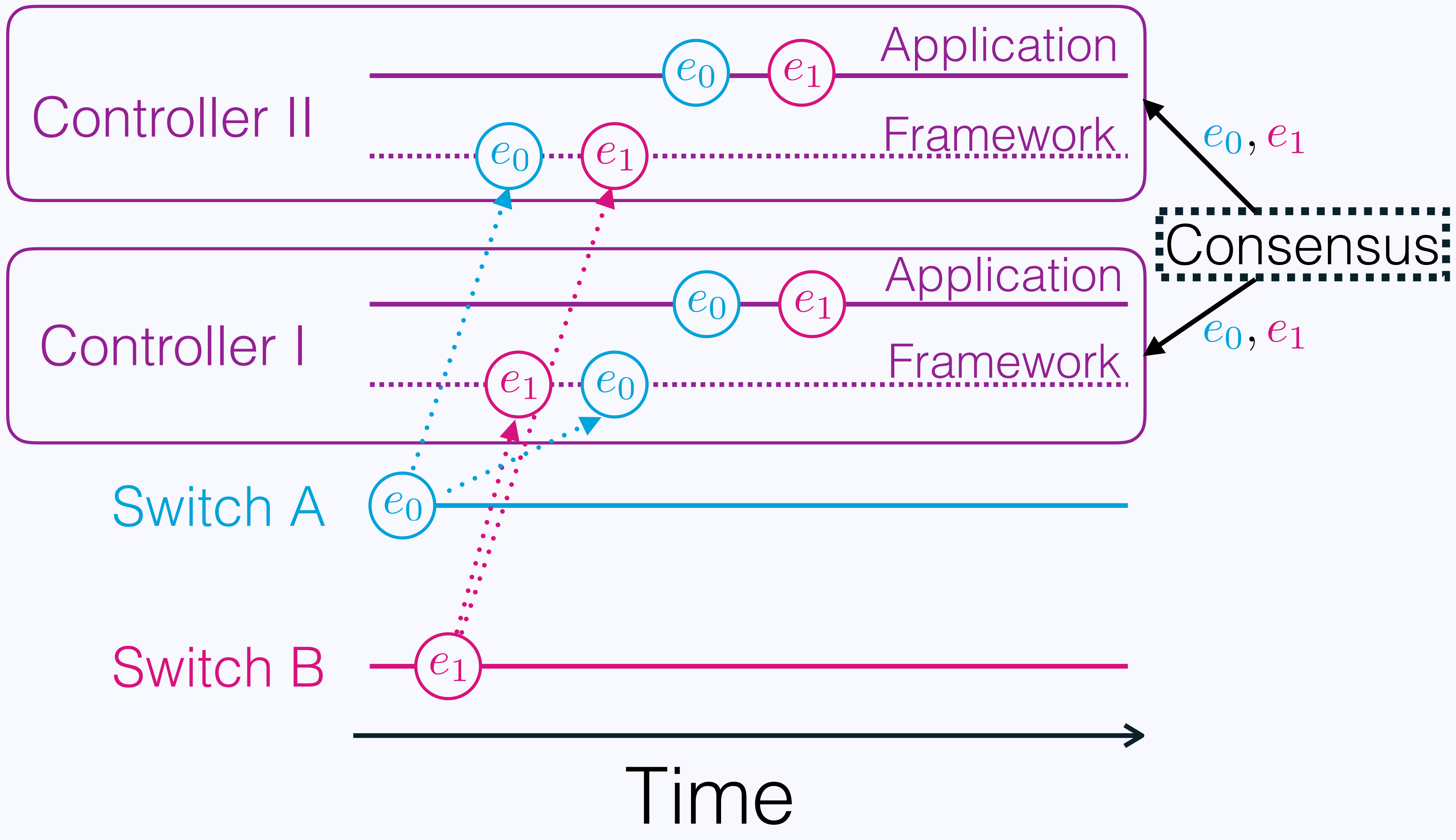
- Consensus: Protocol to get agreement on a value.
- Rely on **consensus** to agree on event order.
  - Applications always see events in agreed order.
- Can use same algorithms as single image controller.
- Controllers are **Replicated State Machines**.

Canonical Solution: Consensus



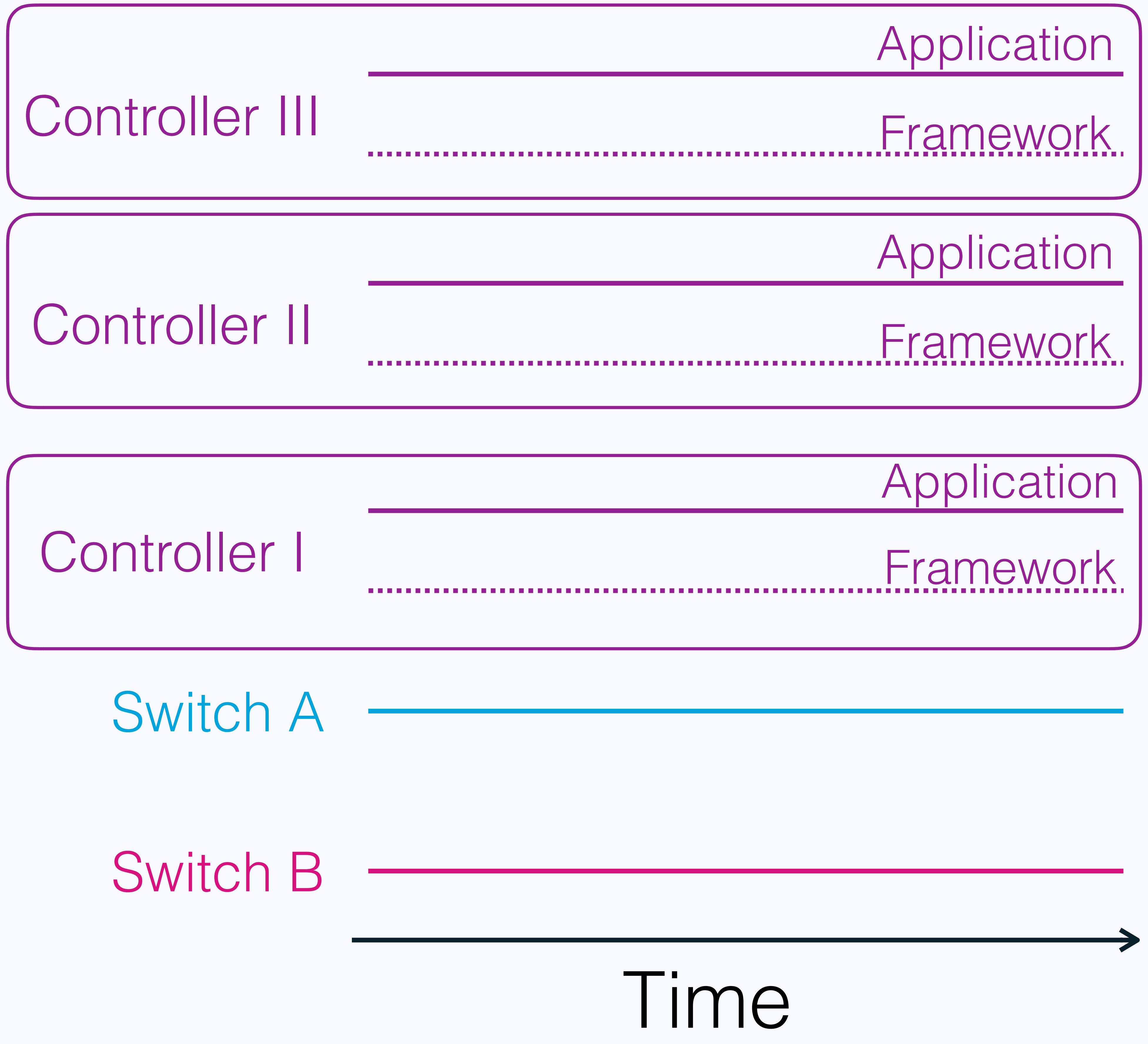
- Consensus: Protocol to get agreement on a value.
- Rely on **consensus** to agree on event order.
  - Applications always see events in agreed order.
- Can use same algorithms as single image controller.
- Controllers are **Replicated State Machines**.
- Adopted by **Onix**, **ONOS**, etc.

Canonical Solution: Consensus



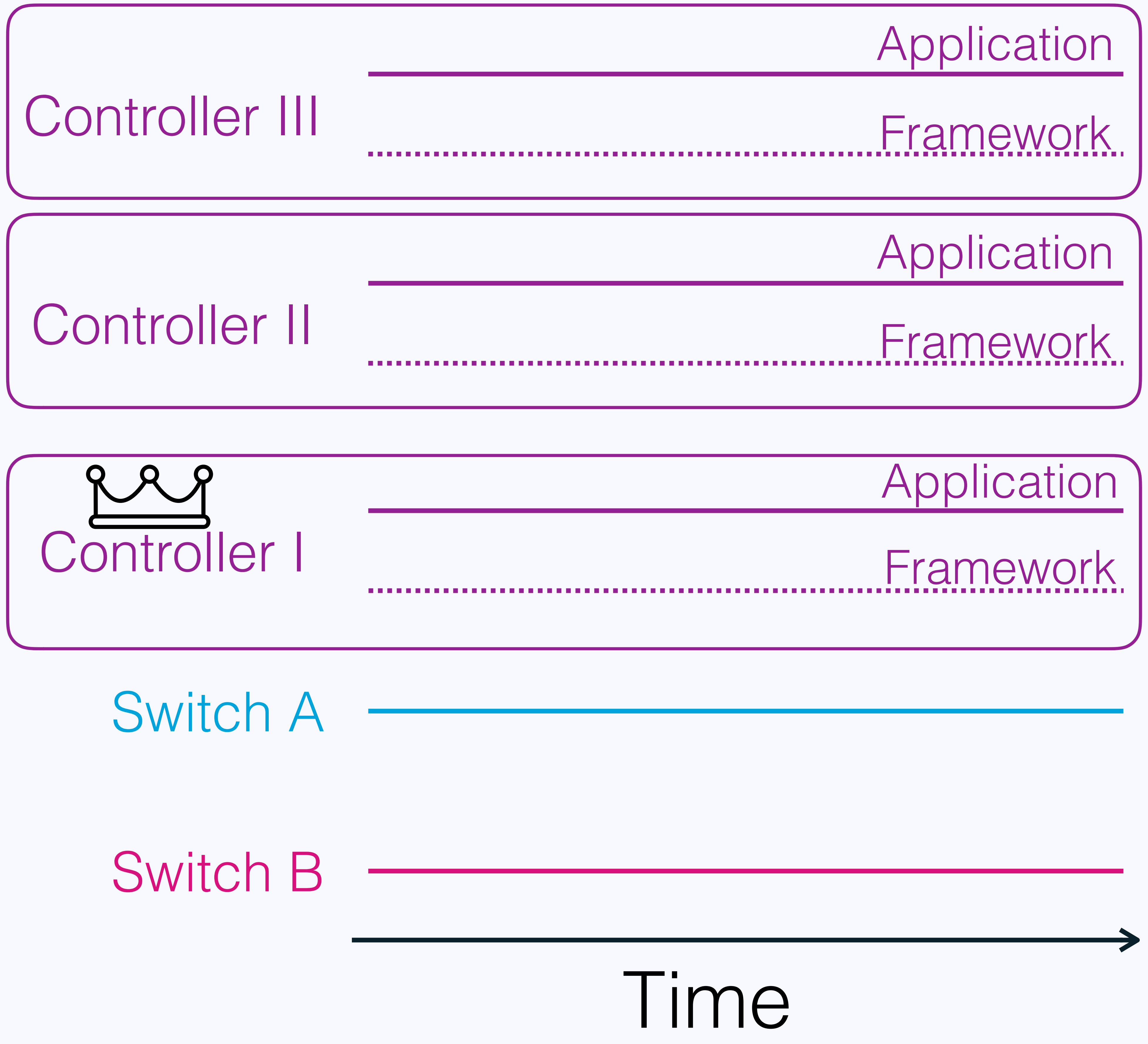
- Consensus: Protocol to get agreement on a value.
- Rely on **consensus** to agree on event order.
  - Applications always see events in agreed order.
- Can use same algorithms as single image controller.
- Controllers are **Replicated State Machines**.
- Adopted by **Onix**, **ONOS**, etc.
- How to implement consensus?

Canonical Consensus Mechanism



- Several algorithms in use - **ZAB**, **Raft**, **Paxos** variants (e.g., MultiPaxos)

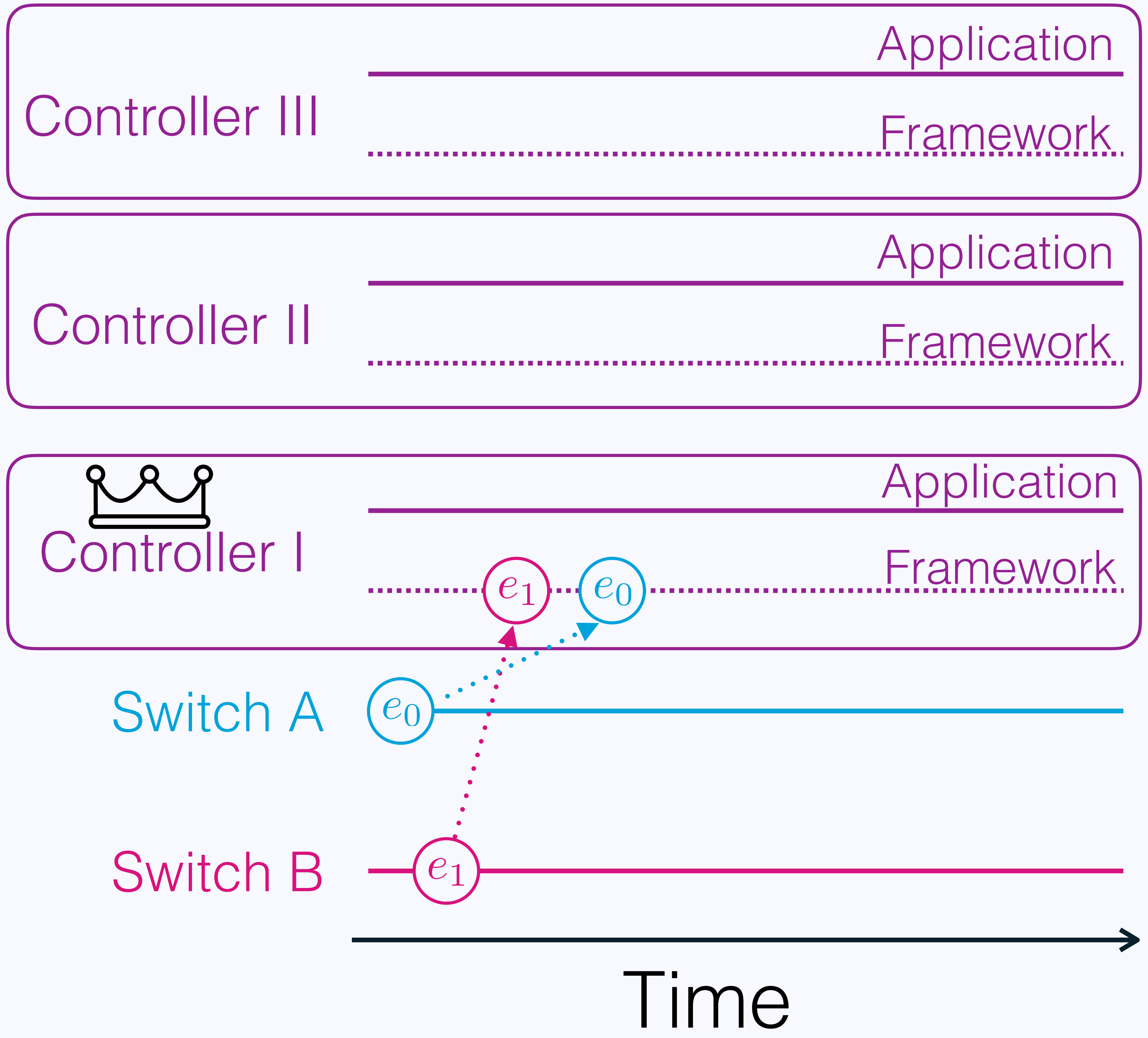
Canonical Consensus Mechanism



- Mechanism appoints a **leader**.

- Several algorithms in use - **ZAB**, **Raft**, **Paxos** variants (e.g., MultiPaxos)

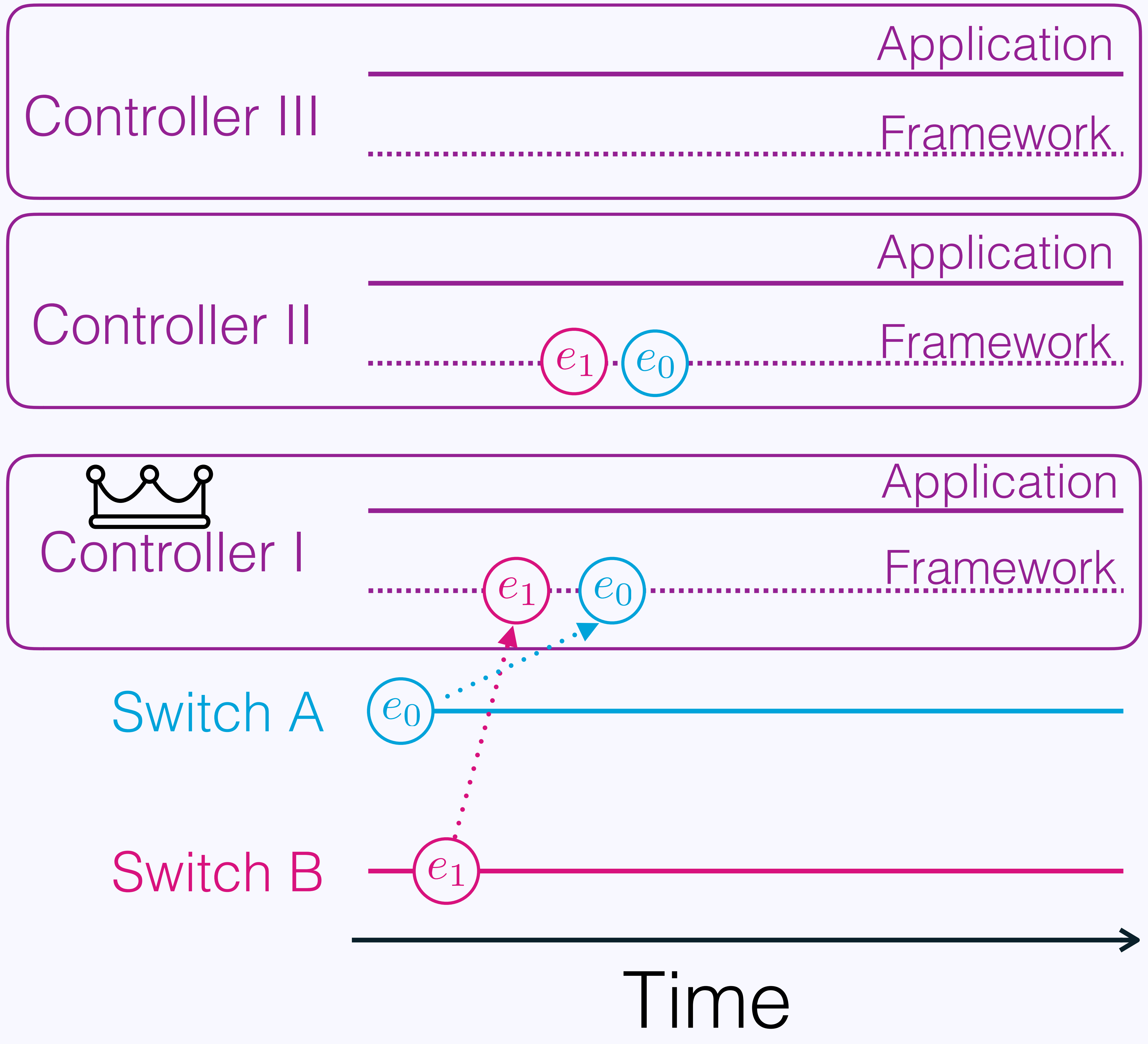
Canonical Consensus Mechanism



- Mechanism appoints a **leader**.
- **Leader** receives all network events - decides on order.

- Several algorithms in use - **ZAB**, **Raft**, **Paxos** variants (e.g., MultiPaxos)

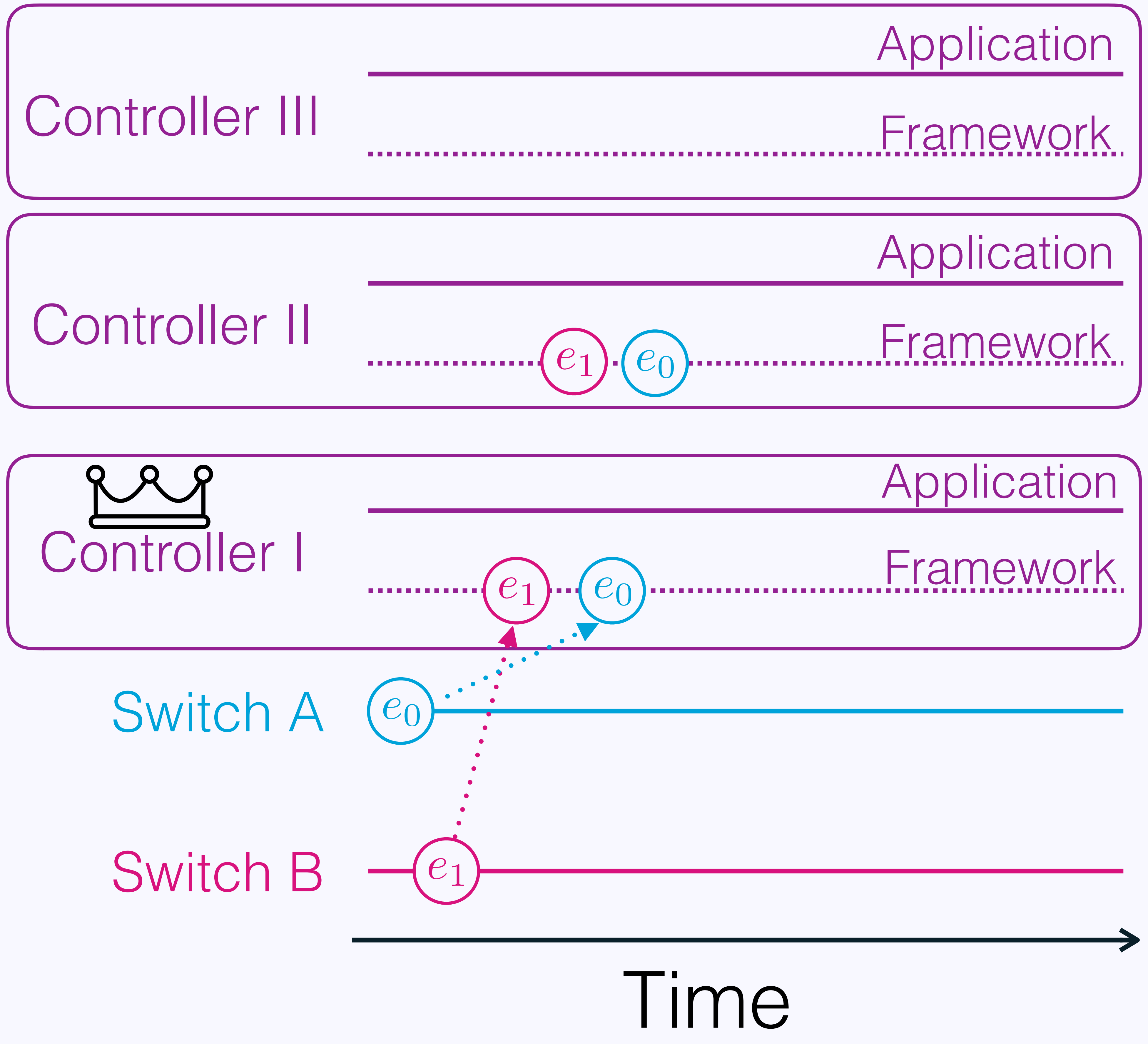
Canonical Consensus Mechanism



- Mechanism appoints a **leader**.
- **Leader** receives all network events - decides on order.
- Leader **replicates** ordered events at other controllers.

- Several algorithms in use - **ZAB**, **Raft**, **Paxos** variants (e.g., MultiPaxos)

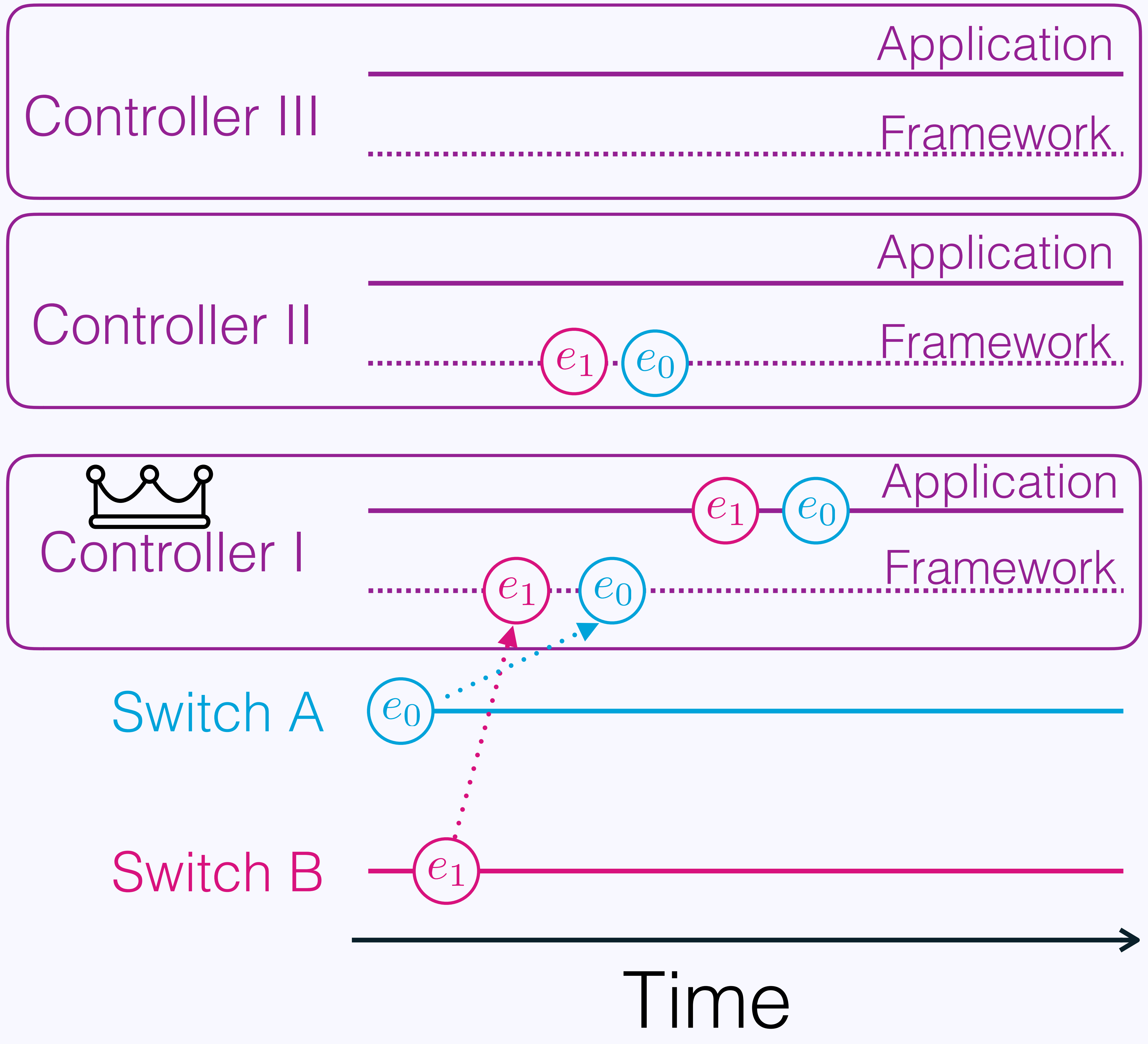
Canonical Consensus Mechanism



- Mechanism appoints a **leader**.
- **Leader** receives all network events - decides on order.
- Leader **replicates** ordered events at other controllers.
- Must wait for a **quorum** of controllers to confirm replication.

- Several algorithms in use - **ZAB**, **Raft**, **Paxos** variants (e.g., MultiPaxos)

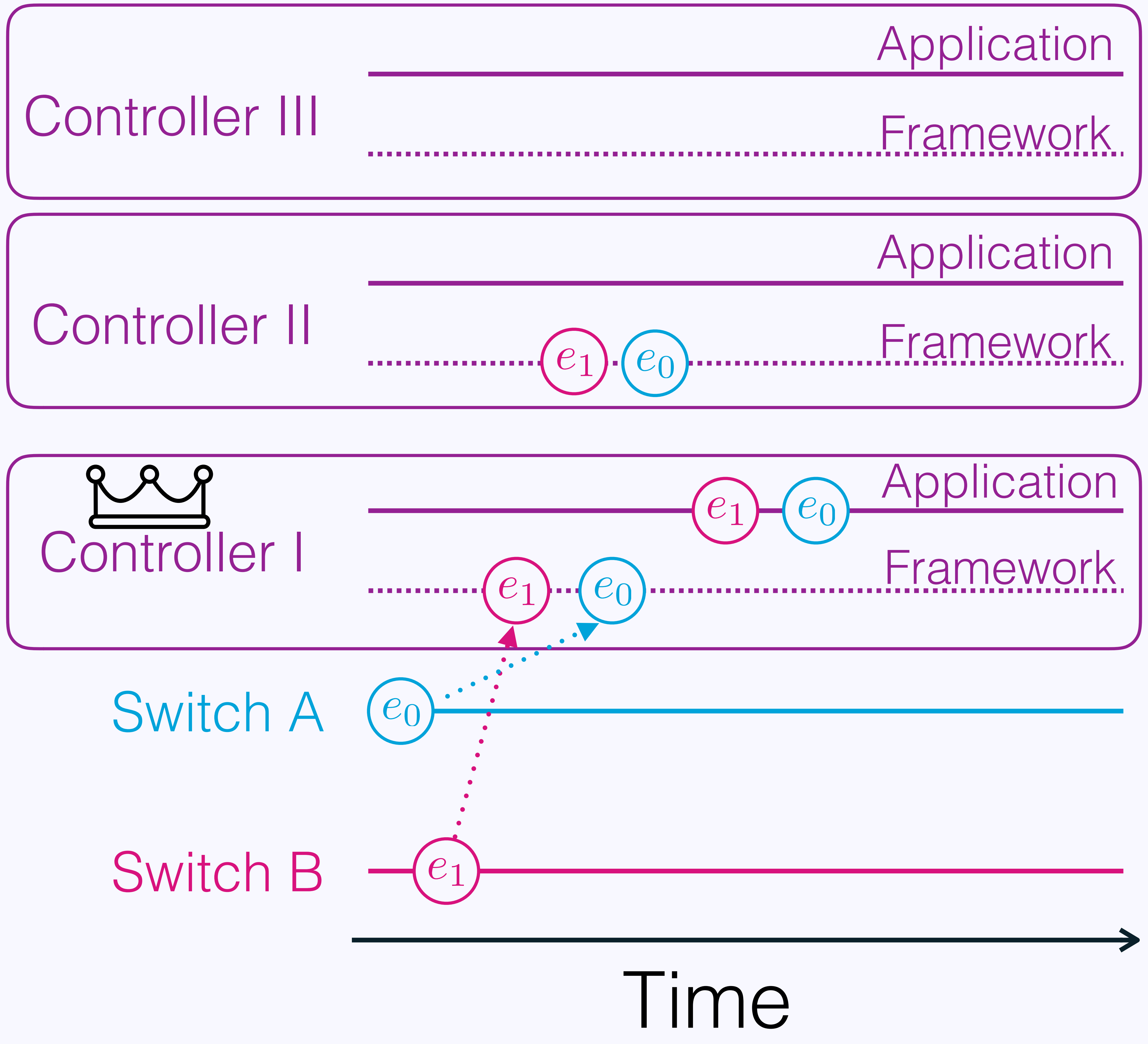
Canonical Consensus Mechanism



- Mechanism appoints a **leader**.
- **Leader** receives all network events - decides on order.
- Leader **replicates** ordered events at other controllers.
- Must wait for a **quorum** of controllers to confirm replication.
- Once quorum has confirmed delivers events to application.

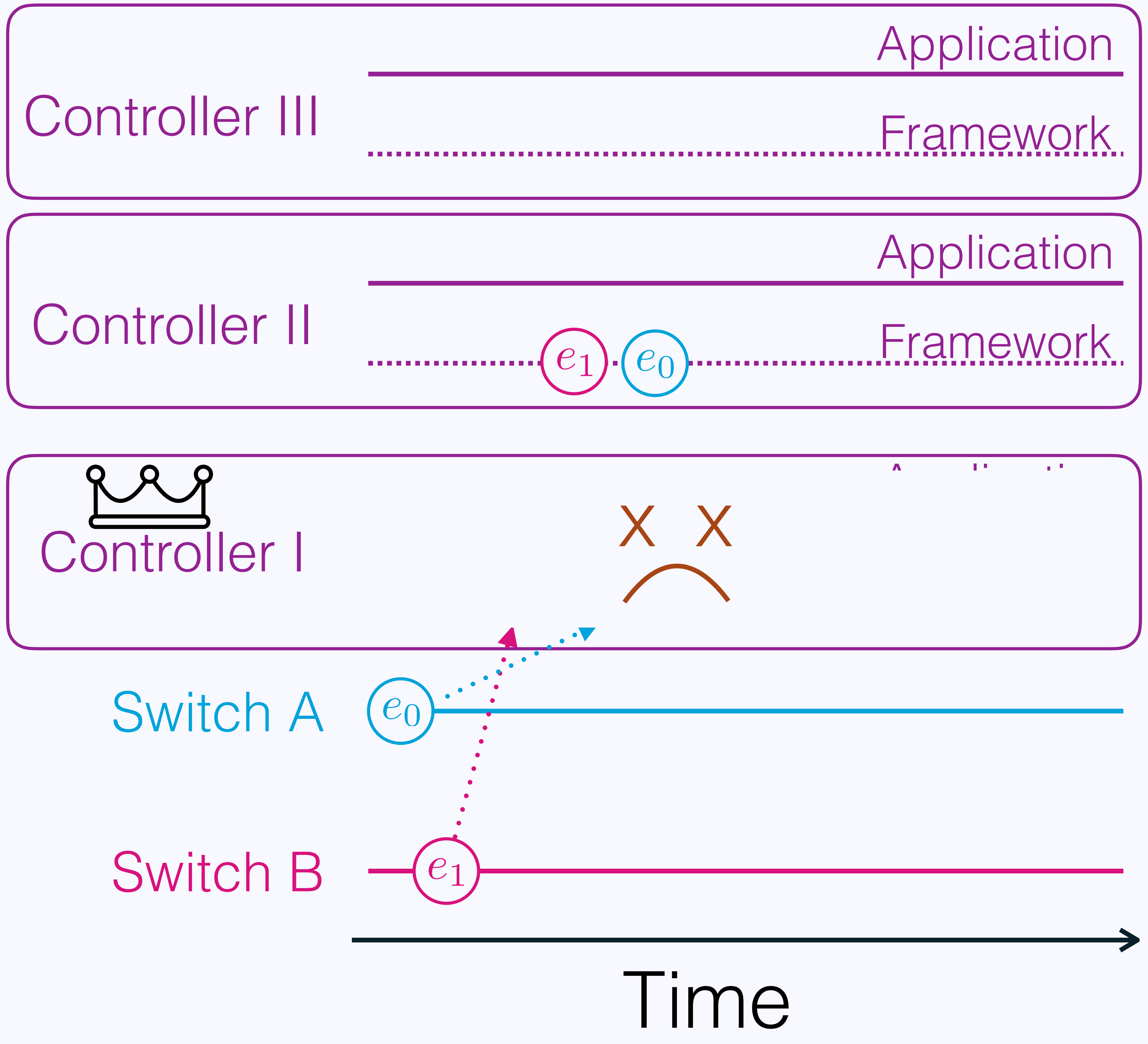
- Several algorithms in use - **ZAB**, **Raft**, **Paxos** variants (e.g., MultiPaxos)

Canonical Consensus Mechanism



- Several algorithms in use - **ZAB**, **Raft**, **Paxos** variants (e.g., MultiPaxos)

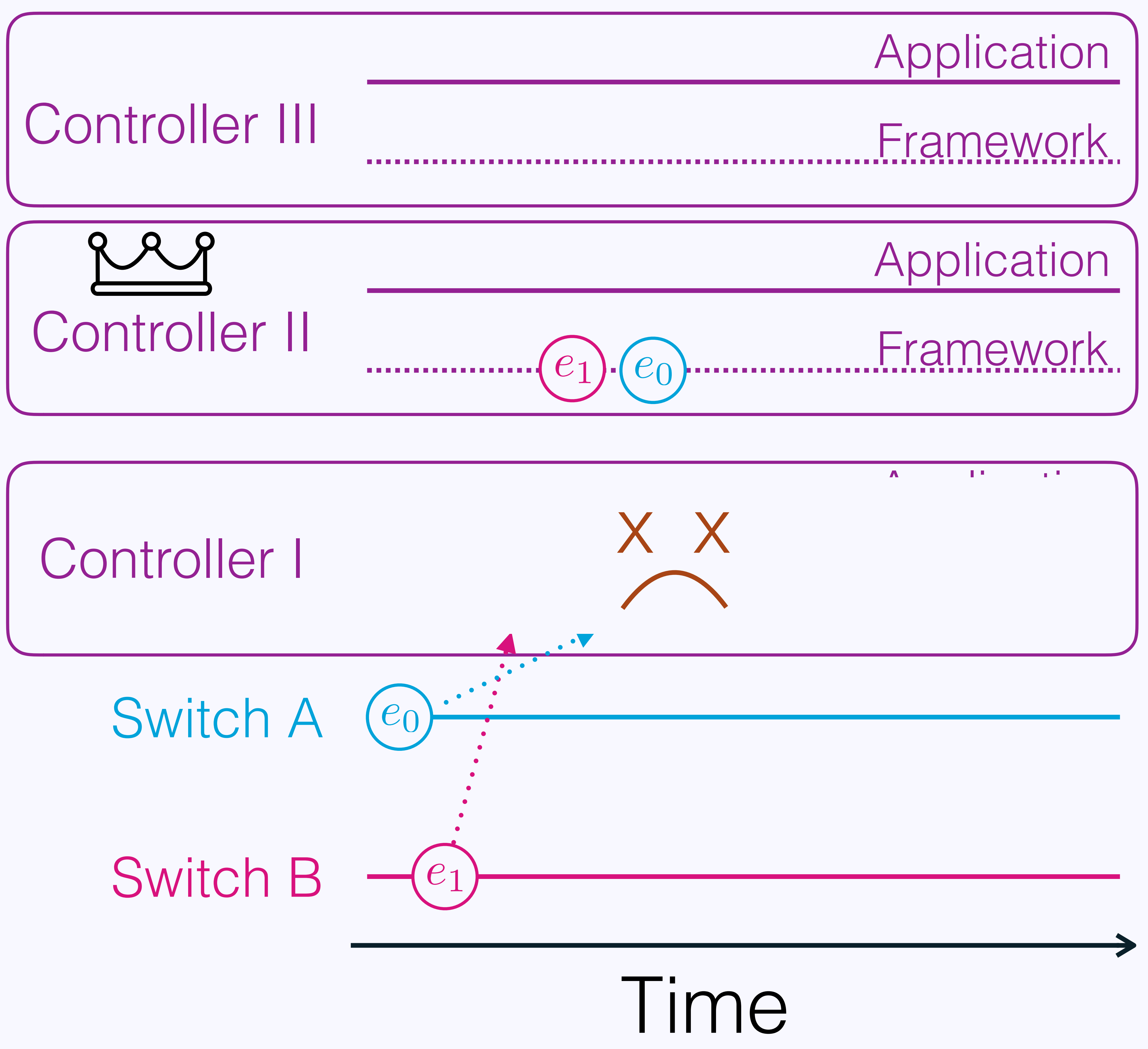
Canonical Consensus Mechanism



- If **leader fails** protocol appoints new leader.

- Several algorithms in use - **ZAB**, **Raft**, **Paxos** variants (e.g., MultiPaxos)

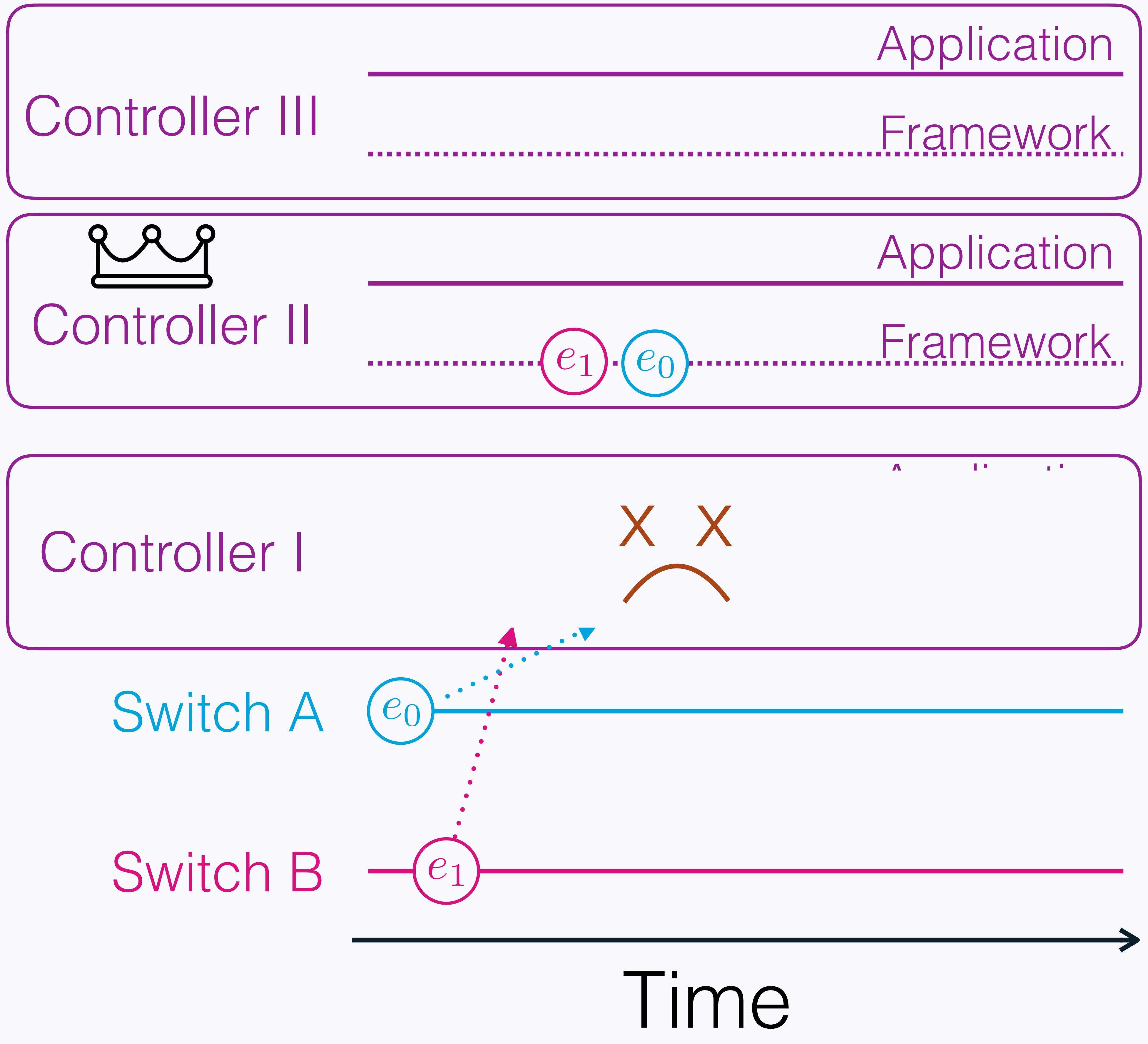
Canonical Consensus Mechanism



- If **leader fails** protocol appoints new leader.
- Protocol must ensure leader is one with newest data.

- Several algorithms in use - **ZAB**, **Raft**, **Paxos** variants (e.g., MultiPaxos)

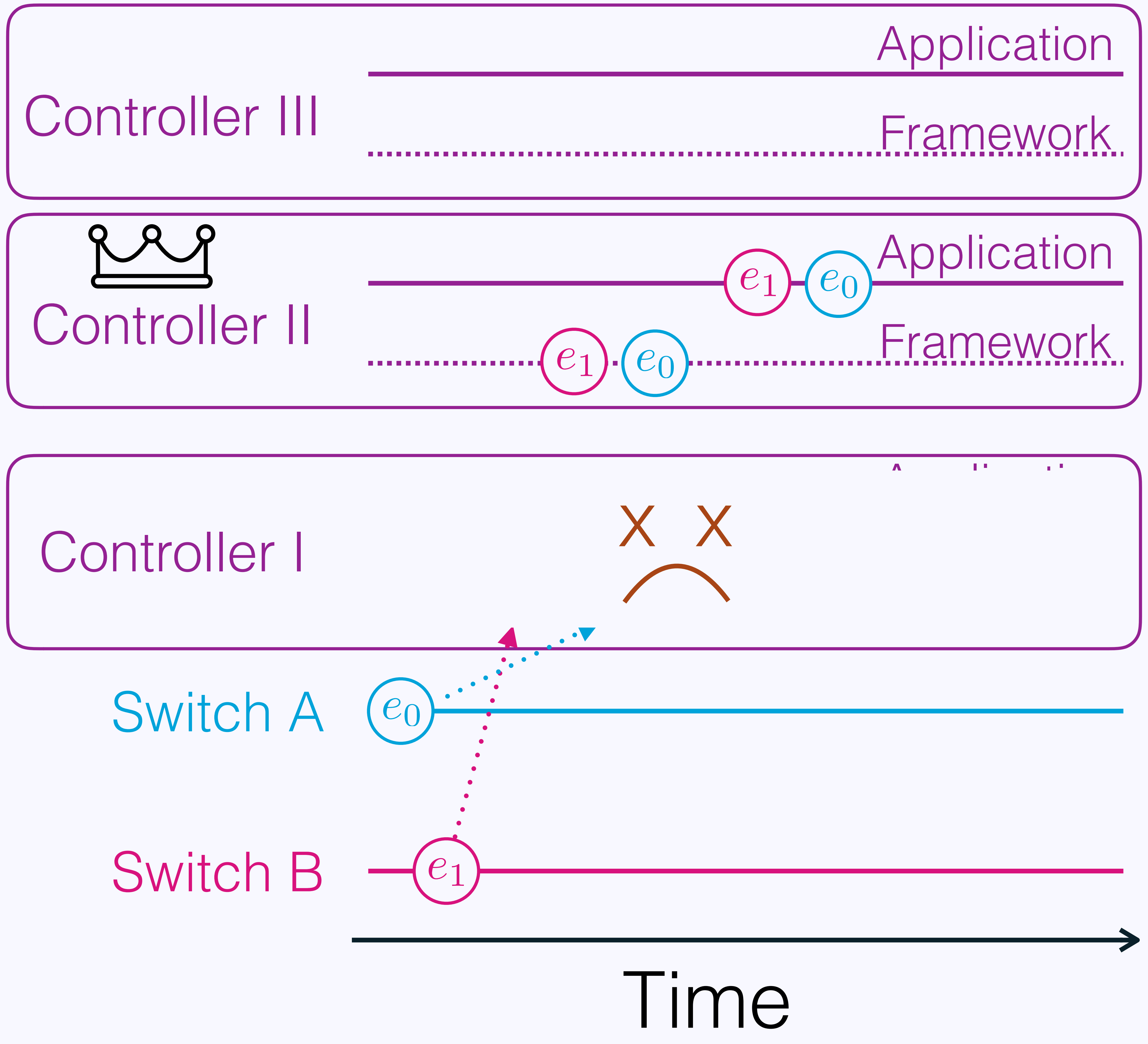
Canonical Consensus Mechanism



- If **leader fails** protocol appoints new leader.
- Protocol must ensure leader is one with newest data.
  - Quorum replication ensures order cannot be forgotten.

- Several algorithms in use - **ZAB**, **Raft**, **Paxos** variants (e.g., MultiPaxos)

Canonical Consensus Mechanism



- If **leader fails** protocol appoints new leader.
- Protocol must ensure leader is one with newest data.
  - Quorum replication ensures order cannot be forgotten.
- Controller can reconstruct state by replaying events.

- Several algorithms in use - **ZAB**, **Raft**, **Paxos** variants (e.g., MultiPaxos)

Canonical Consensus Mechanism: Limitations

- Fault Tolerance: at **least one partition** fails during **network partitions**.

Canonical Consensus Mechanism: Limitations

- Fault Tolerance: at **least one partition** fails during **network partitions**.
- Scalability: Worse performance worsens with more controllers.

Canonical Consensus Mechanism: Limitations

- Fault Tolerance: at **least one partition** fails during **network partitions**.
- Scalability: Worse performance worsens with more controllers.
- Control Plane Requirements: Performance is sensitive to losses, latency, etc.

Is consensus required?

Consensus Assumption

- **Network state** (topology and forwarding table) resides in **controllers**.

Consensus Assumption

- **Network state** (topology and forwarding table) resides in **controllers**.
- RSMs ensure **network state** is **not lost** when controllers fail.

Consensus Assumption

- **Network state** (topology and forwarding table) resides in **controllers**.
- RSMs ensure **network state** is **not lost** when controllers fail.
    - Similar to distributed **key value stores**.

Consensus Assumption is Wrong

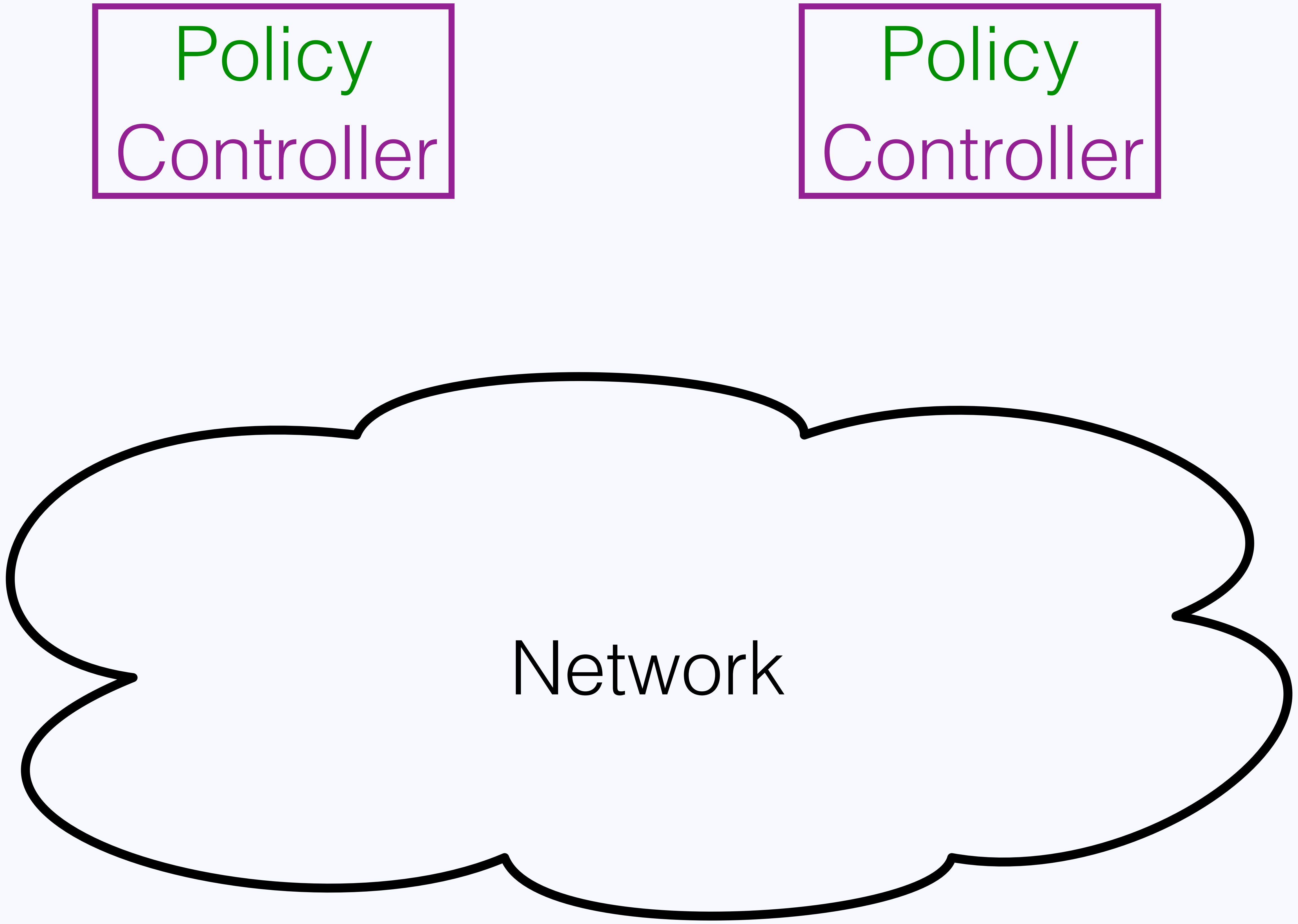
But we can **query the network** to **discover** current **network state**.

Consensus Assumption is Wrong

But we can **query the network** to **discover** current **network state**.

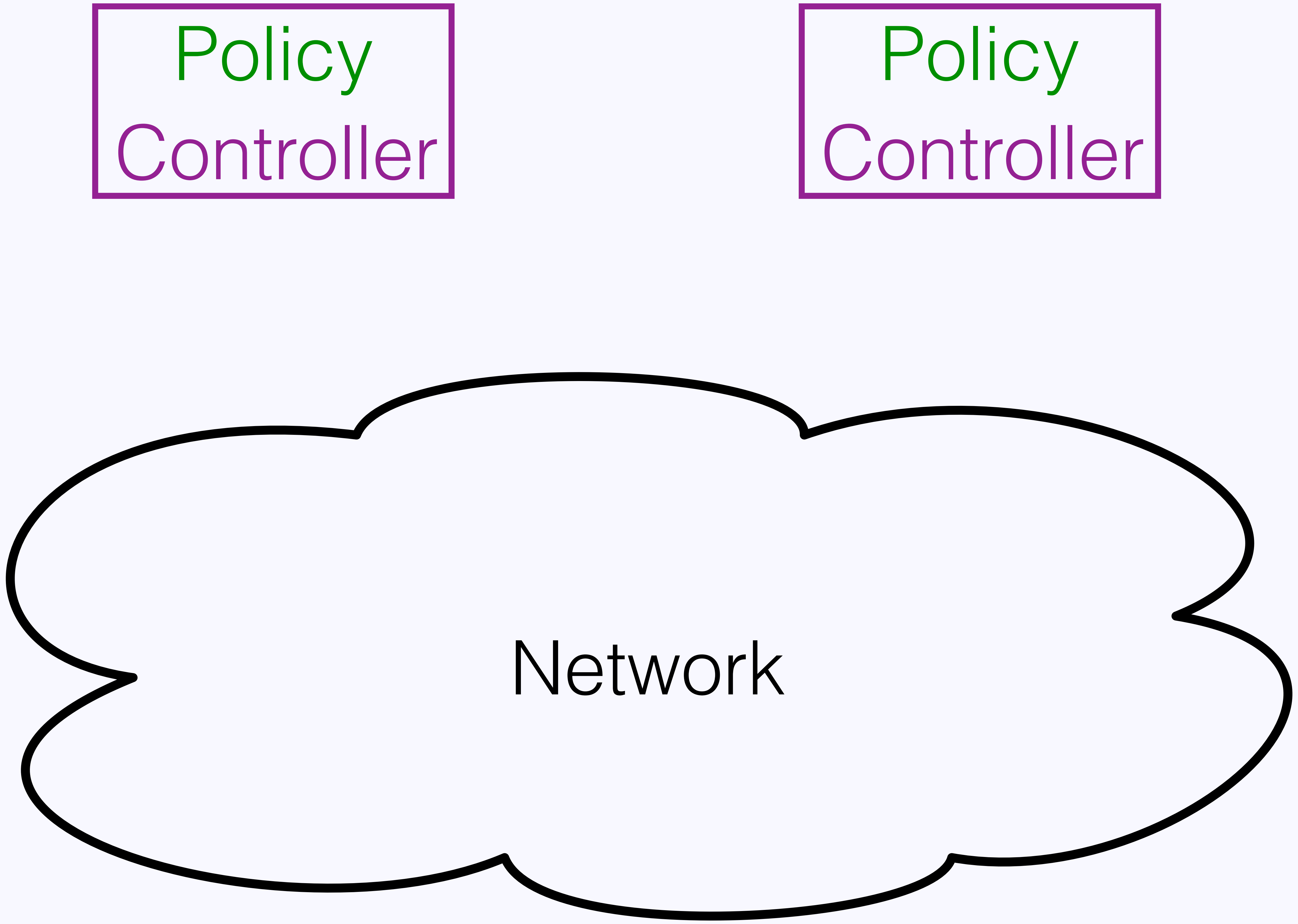
Safe to lose network state!

Distributed Controllers: An Alternative



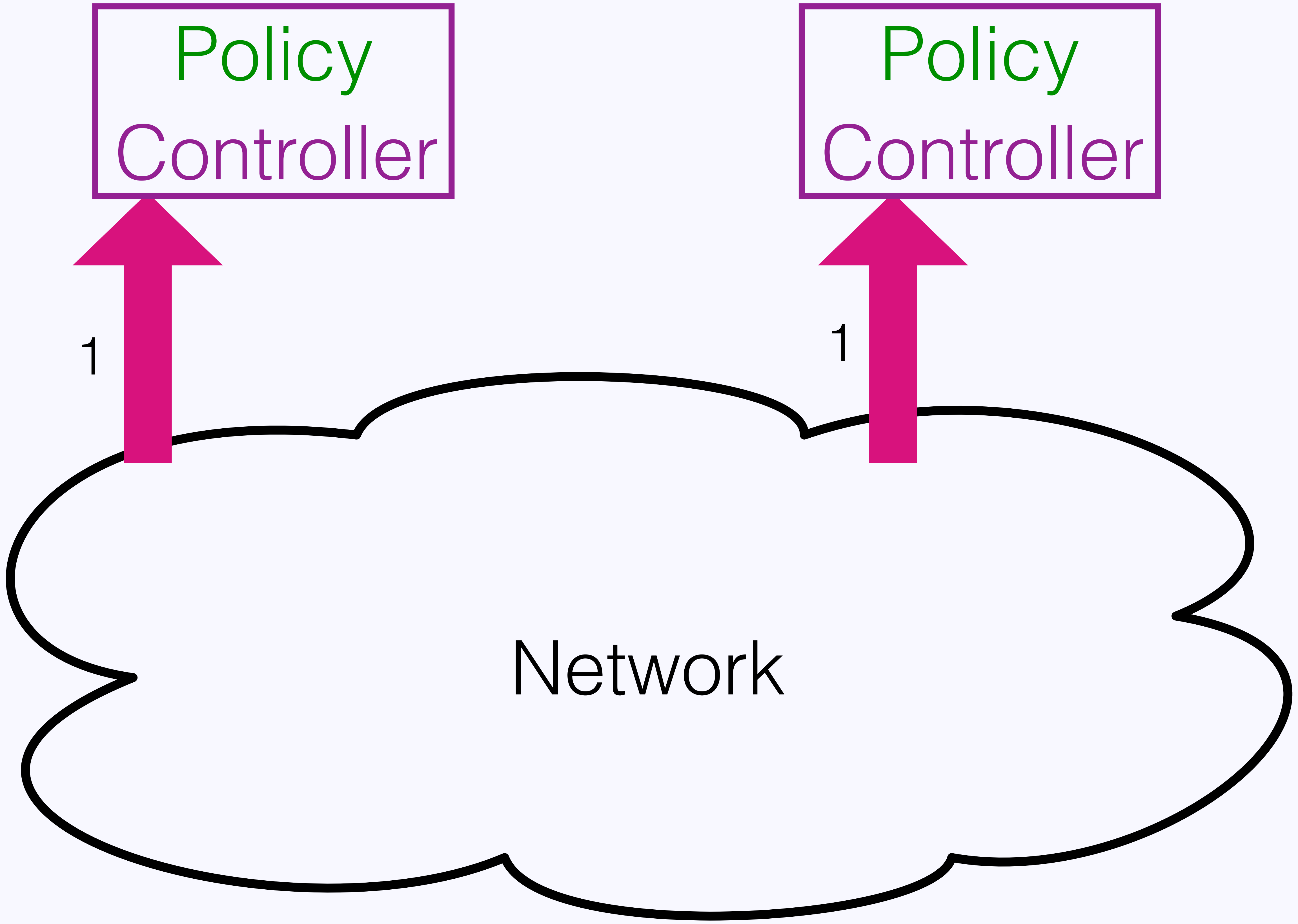
- Assume all controllers agree on policy.

Distributed Controllers: An Alternative



- Assume all controllers agree on policy.
- Each controller

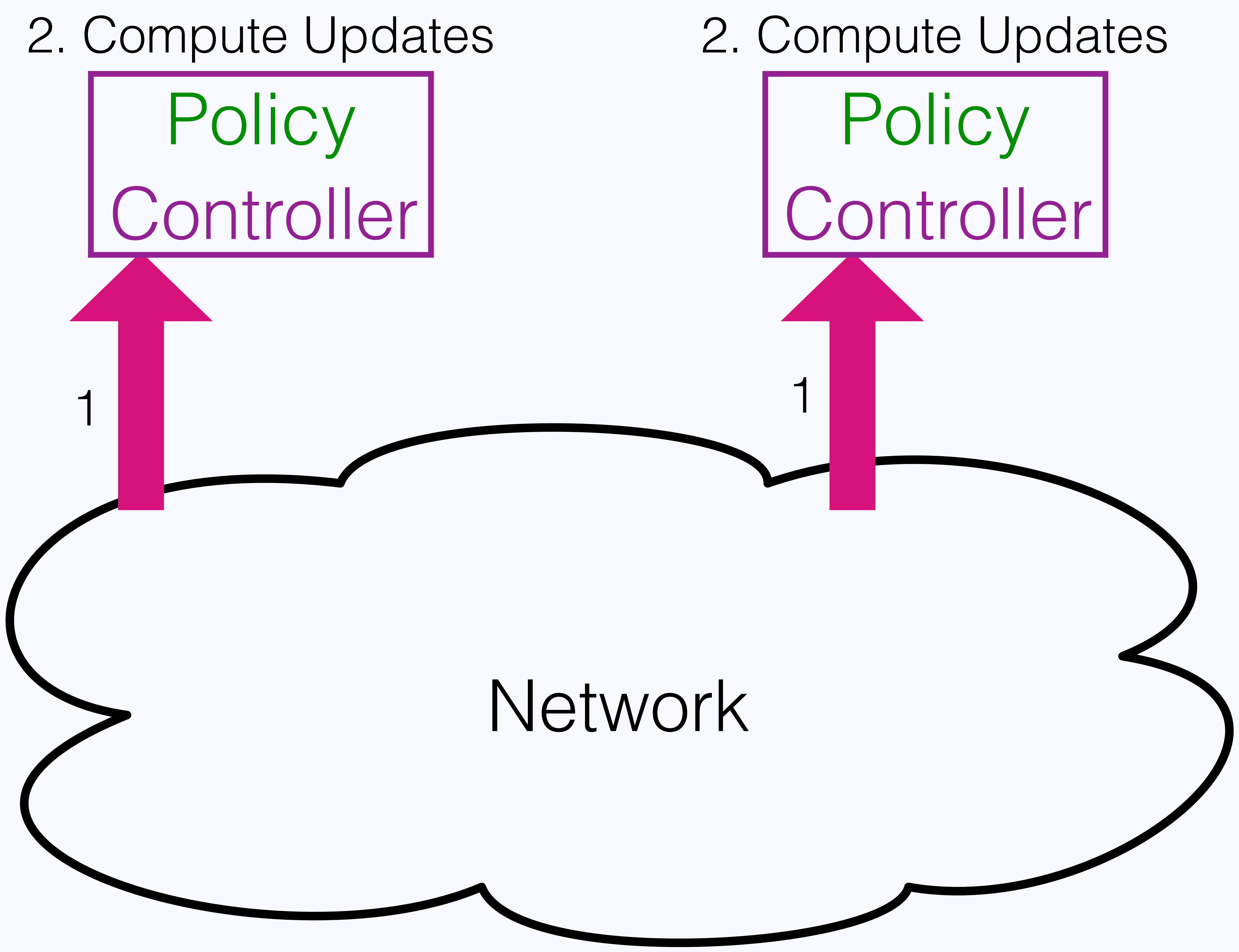
Distributed Controllers: An Alternative



- Assume all controllers agree on policy.
- Each controller

  1. Periodically queries **network state**.

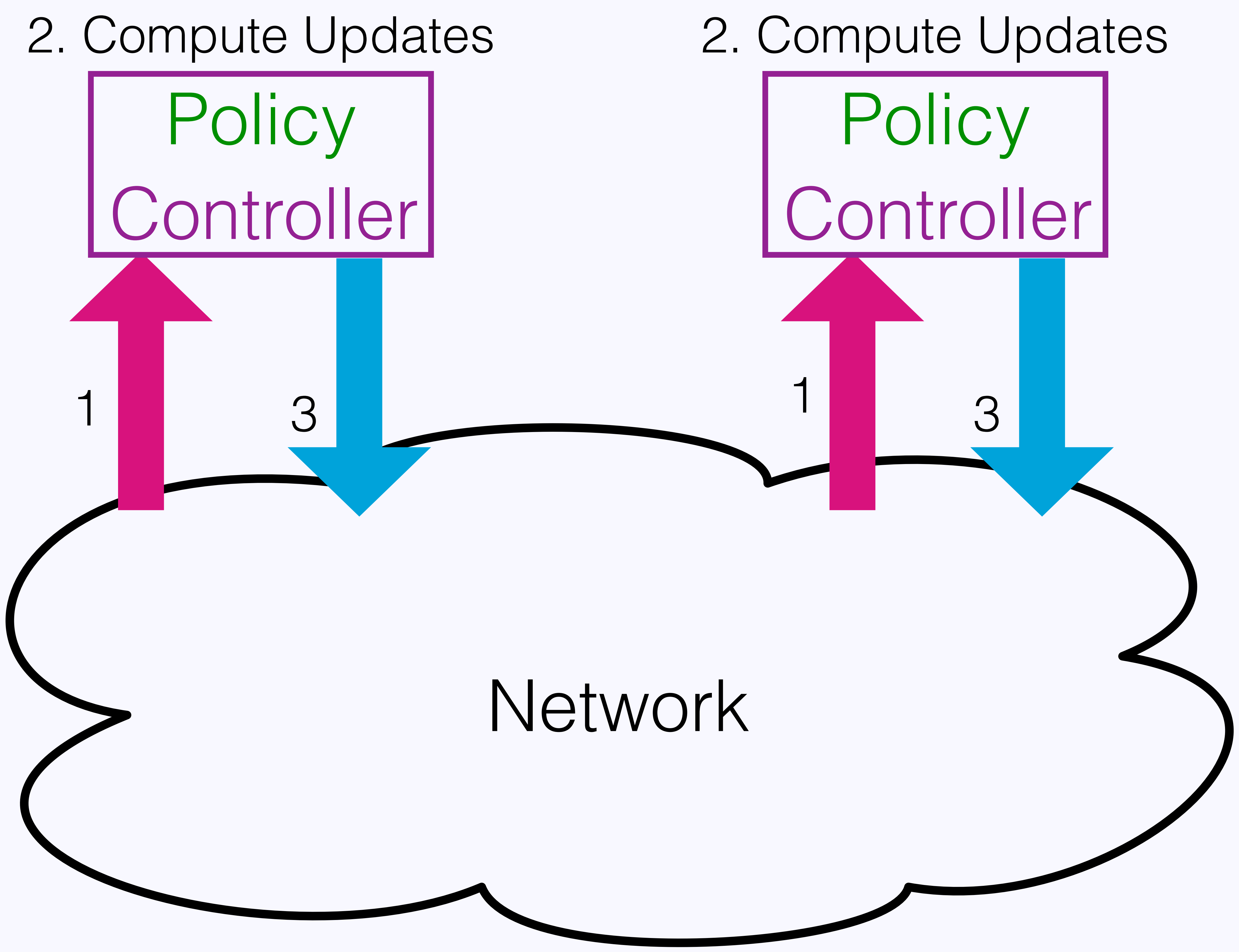
Distributed Controllers: An Alternative



- Assume all controllers agree on policy.
- Each controller

1. Periodically queries **network state**.
2. Uses state and policy to **compute updates**.

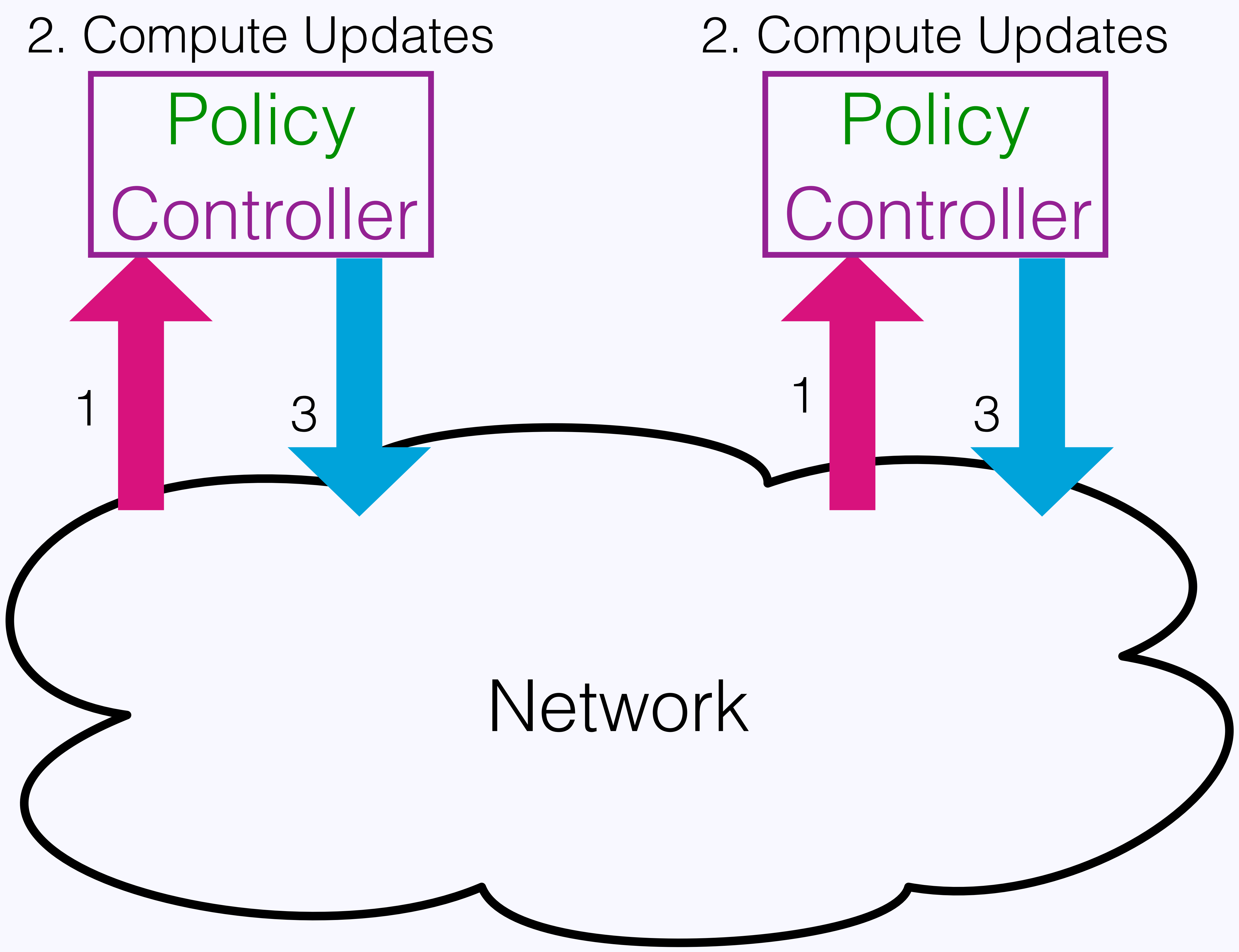
Distributed Controllers: An Alternative



- Assume all controllers agree on policy.
- Each controller

1. Periodically queries **network state**.
2. Uses state and policy to **compute updates**.
3. Installs updates in the network.

Distributed Controllers: An Alternative



- Assume all controllers agree on policy.
- Each controller

  1. Periodically queries **network state**.
  2. Uses state and policy to **compute updates**.
  3. Installs updates in the network.

- Converges assuming quiescence.

Challenges

- Programming model: how to write control applications?

Challenges

- Programming model: how to write control applications?
- Programming model: how to support existing event based algorithms?

Challenges

- Programming model: how to write control applications?
- Programming model: how to support existing event based algorithms?
- Efficiency: how to minimize control traffic?

Challenges

- Programming model: how to write control applications?
- Programming model: how to support existing event based algorithms?
- Efficiency: how to minimize control traffic?
- Safety: how to ensure some critical policies are never violated?

Challenges

- Programming model: how to write control applications?
- Programming model: how to support existing event based algorithms?
- Efficiency: how to minimize control traffic?
- Safety: how to ensure some critical policies are never violated?
- Safety: how to safely update network policies?

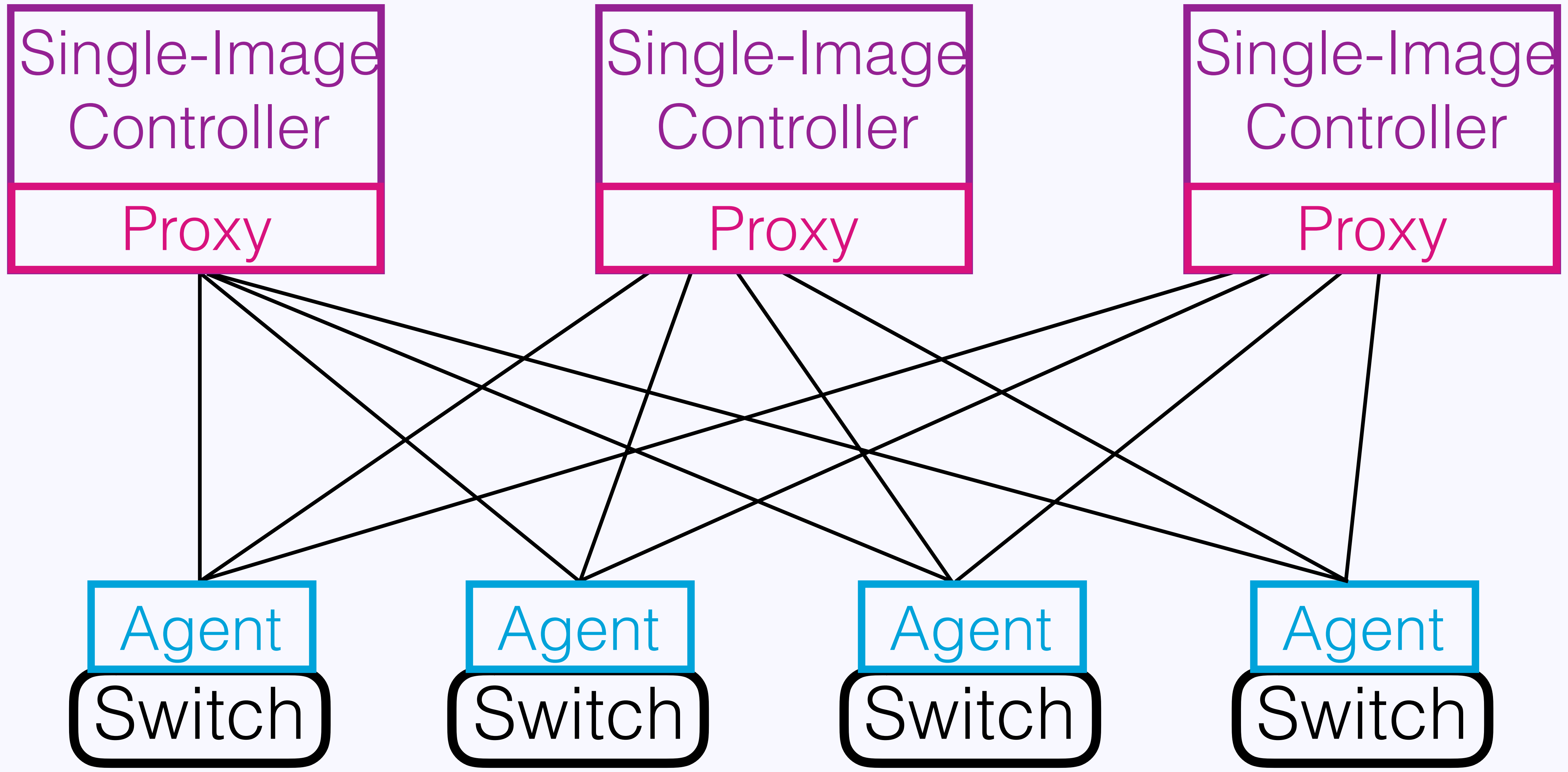
Challenges

- Programming model: how to write control applications?
- Programming model: how to support existing event based algorithms?
- Efficiency: how to minimize control traffic?
- Safety: how to ensure some critical policies are never violated?
- Safety: how to safely update network policies?
- Policies: what classes of policies can be implemented using this mechanism?

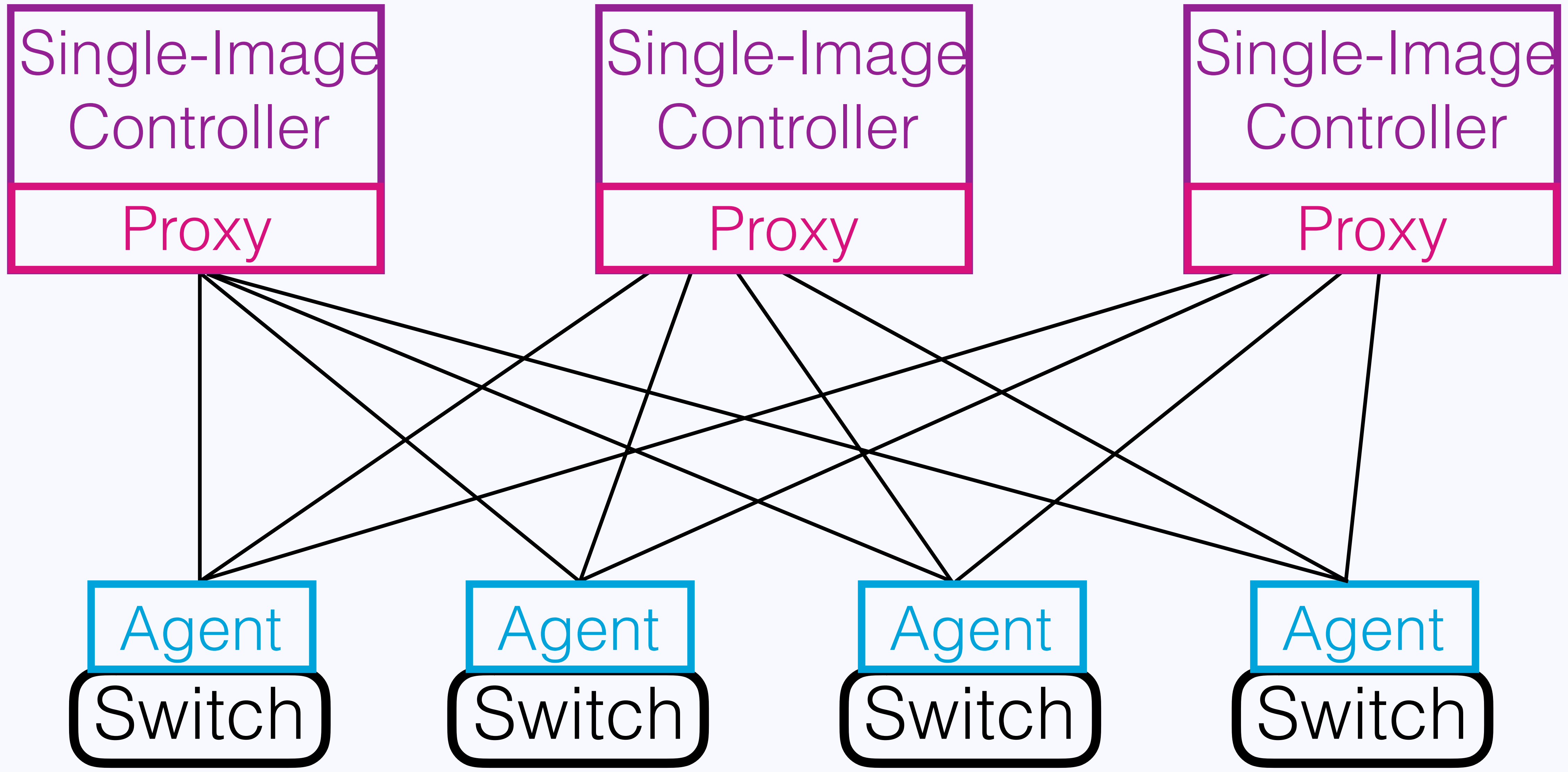
Challenges

- Programming model: how to write control applications?
- Programming model: how to support existing event based algorithms?
- Efficiency: how to minimize control traffic?
- Safety: how to ensure some critical policies are never violated?
- Safety: how to safely update network policies?
- Policies: what classes of policies can be implemented using this mechanism?

SCL: Programming Model and Architecture

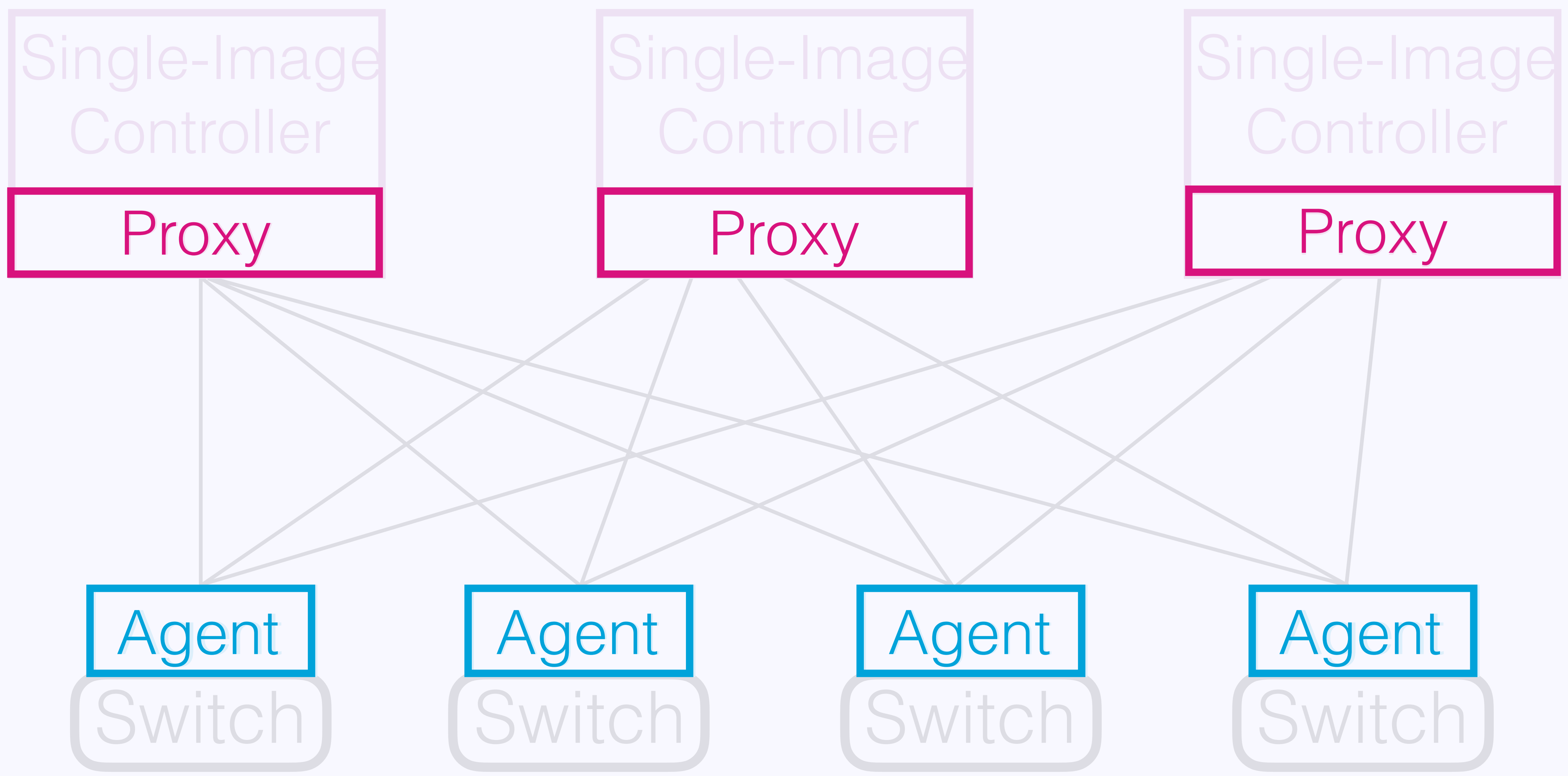


SCL: Programming Model and Architecture



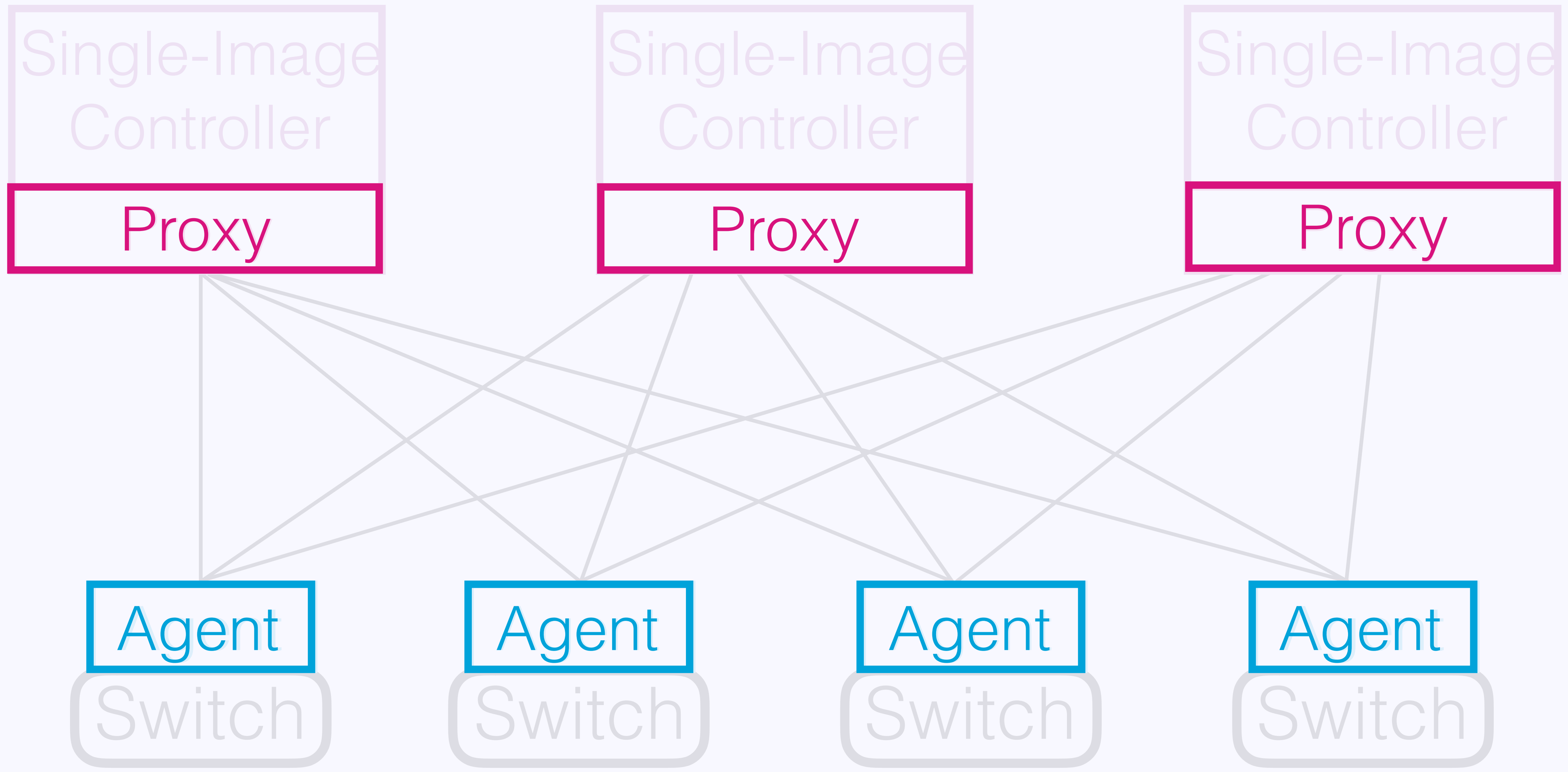
- Builds on standard single-image controller (Pox).

SCL: Programming Model and Architecture



- Builds on standard single-image controller (Pox).
- Switch Agents implement querying and channels.

SCL: Programming Model and Architecture



- Builds on standard single-image controller (Pox).
- Switch Agents implement querying and channels.
- Controller Proxies ensure convergence.

SCL Controller Requirements

- **Deterministic**: Controllers compute the same rule for given network state.

SCL Controller Requirements

- **Deterministic**: Controllers compute the same rule for given network state.
- **Idempotent**: The process of computing and updating rules is idempotent.

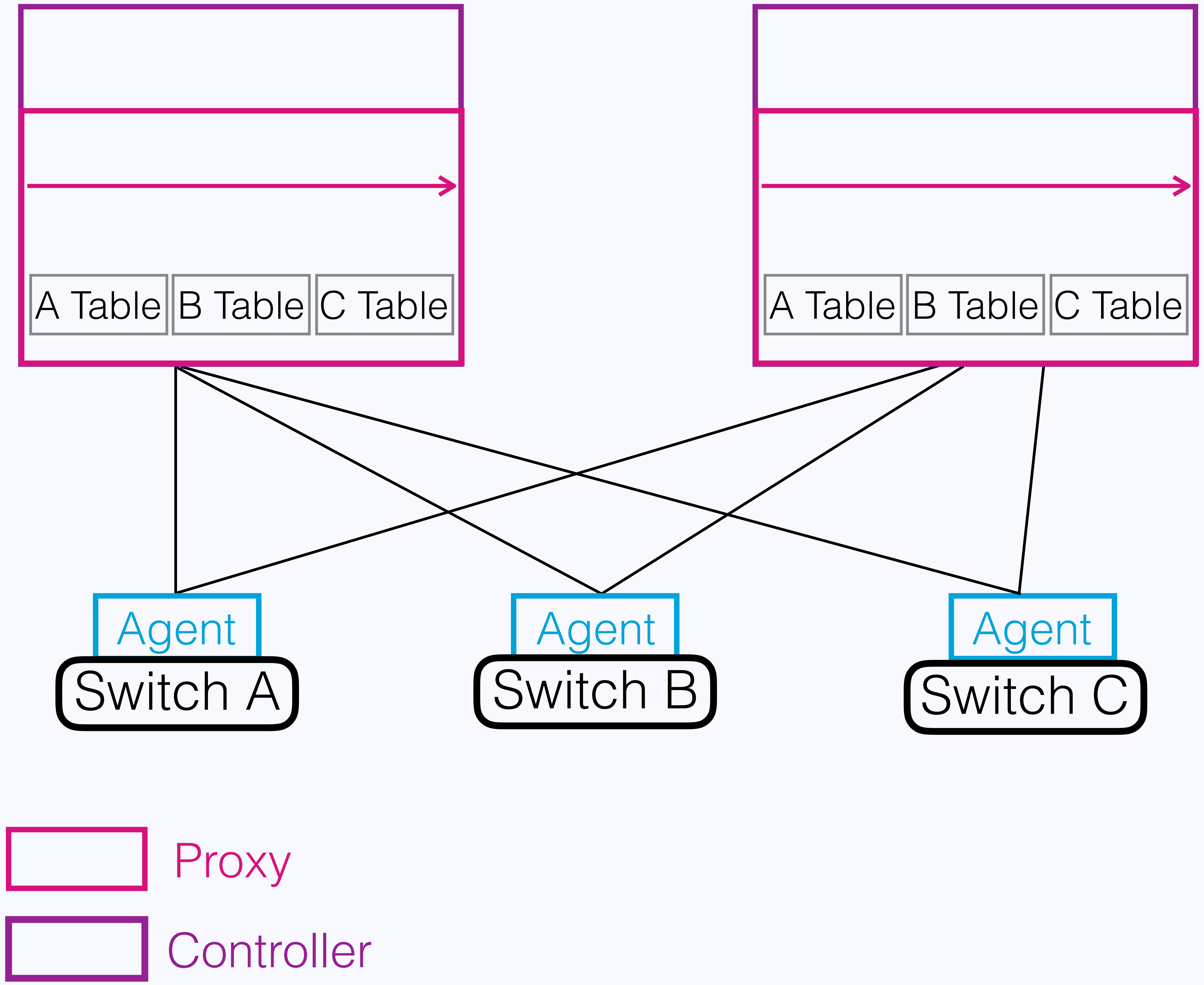
SCL Controller Requirements

- **Deterministic**: Controllers compute the same rule for given network state.
- **Idempotent**: The process of computing and updating rules is idempotent.
- **Proactive Applications**: Compute rules based on network state not packet-ins.

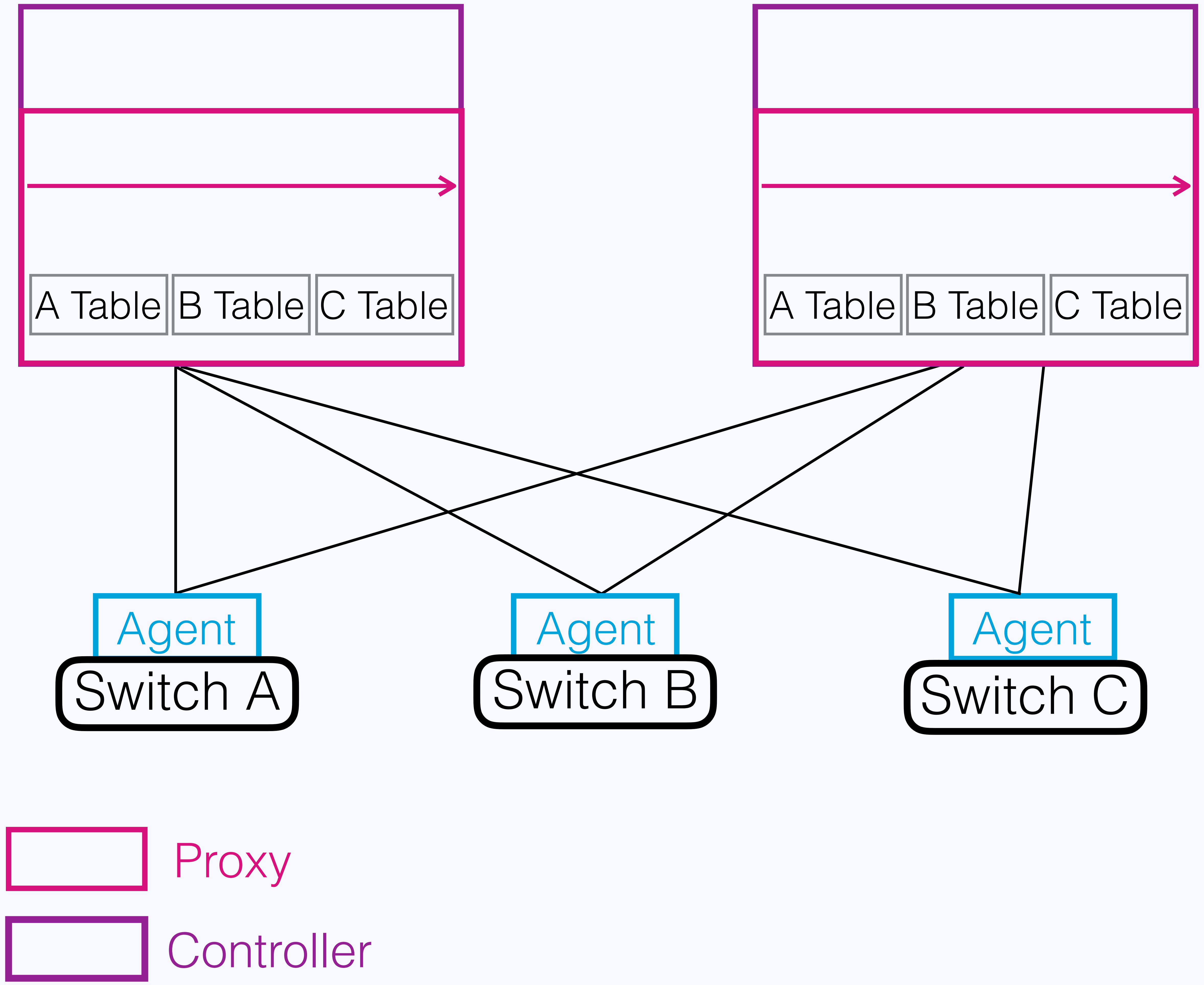
SCL Controller Requirements

- **Deterministic**: Controllers compute the same rule for given network state.
- **Idempotent**: The process of computing and updating rules is idempotent.
- **Proactive Applications**: Compute rules based on network state not packet-ins.
- **Triggered Updates**: Can trigger rule recomputation based on event log.

SCL Proxies and Controllers

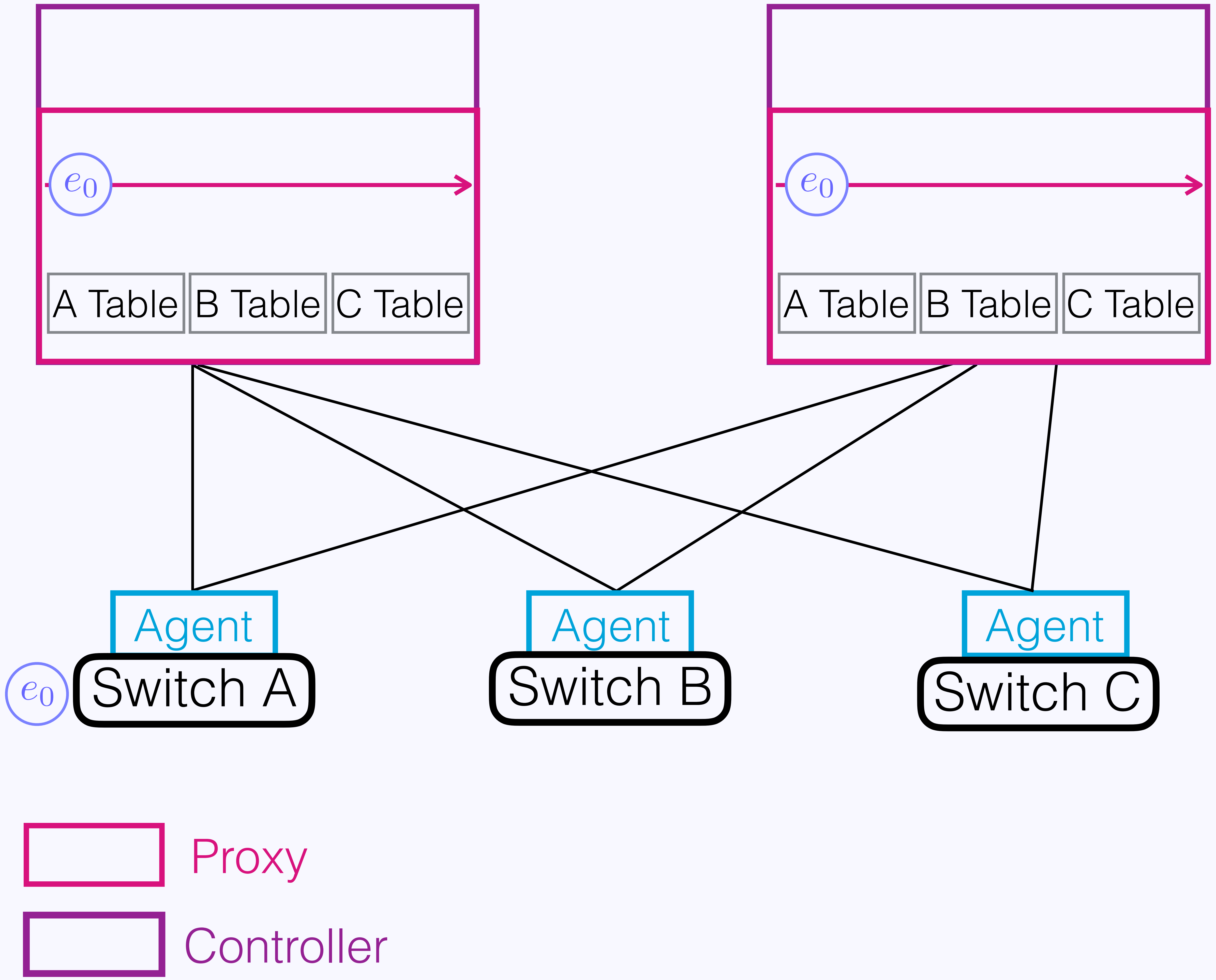


SCL Proxies and Controllers



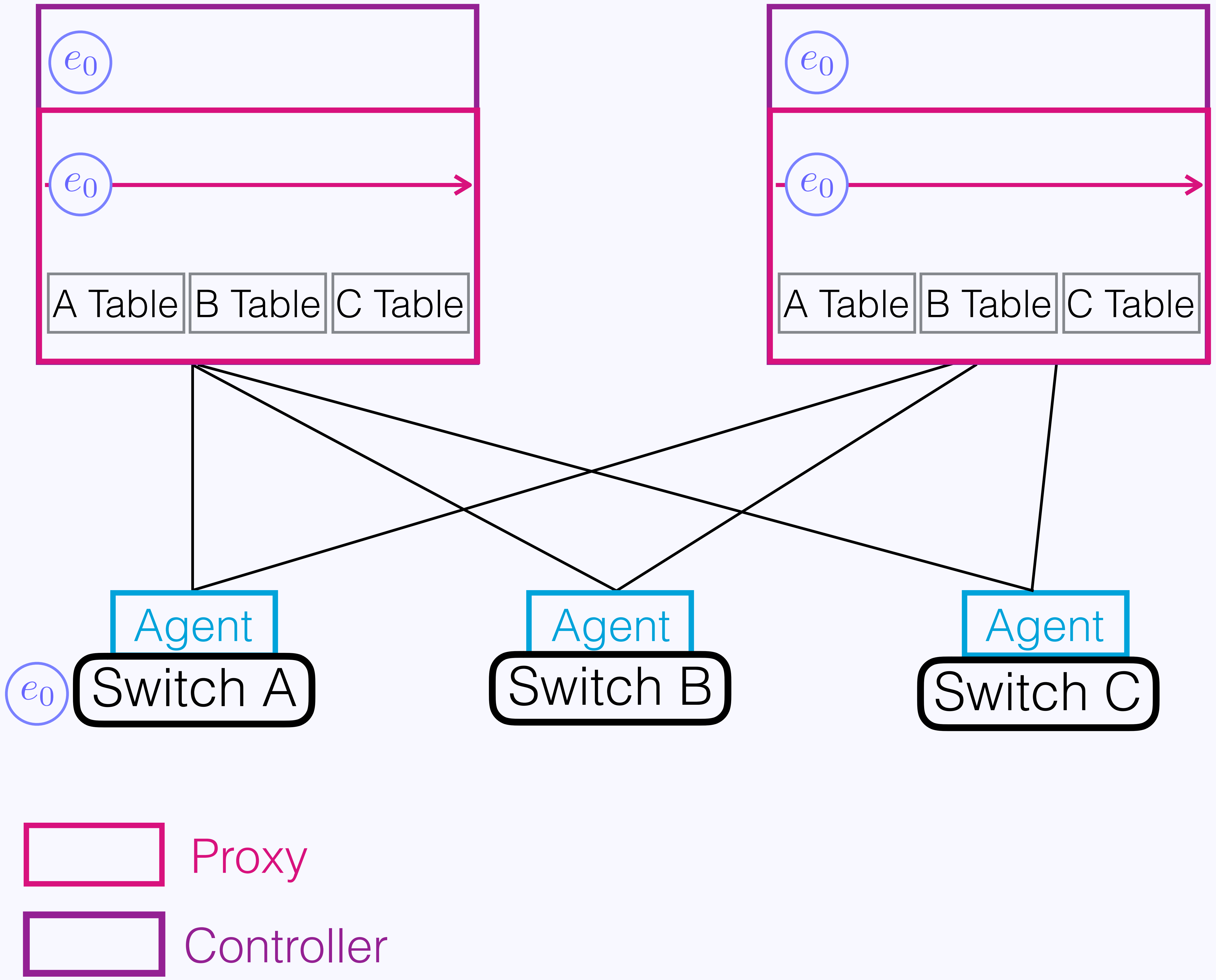
- Proxies maintain a log of all prior network events.

SCL Proxies and Controllers



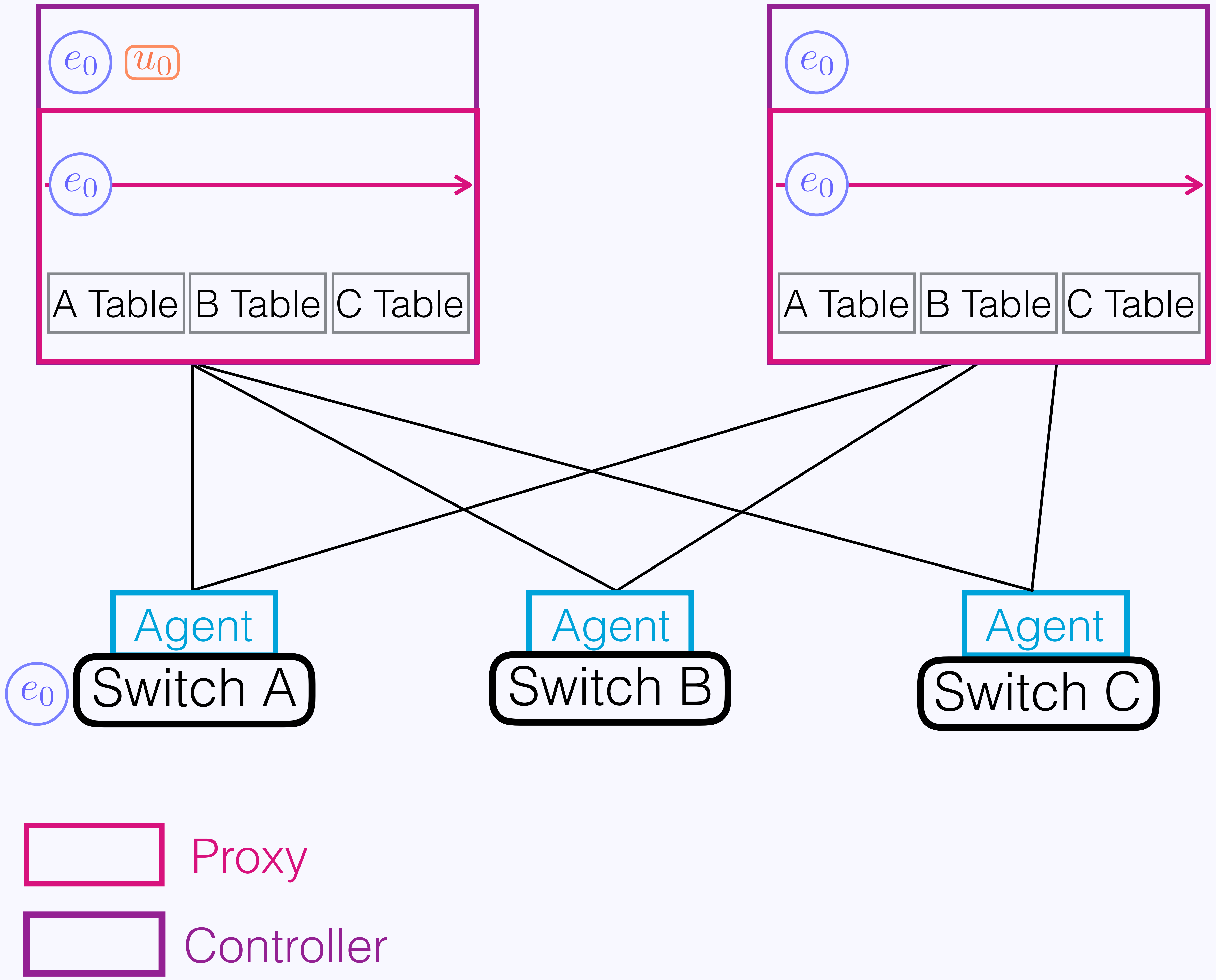
- Proxies maintain a log of all prior network events.
- All switch events are sent to all proxies.

SCL Proxies and Controllers



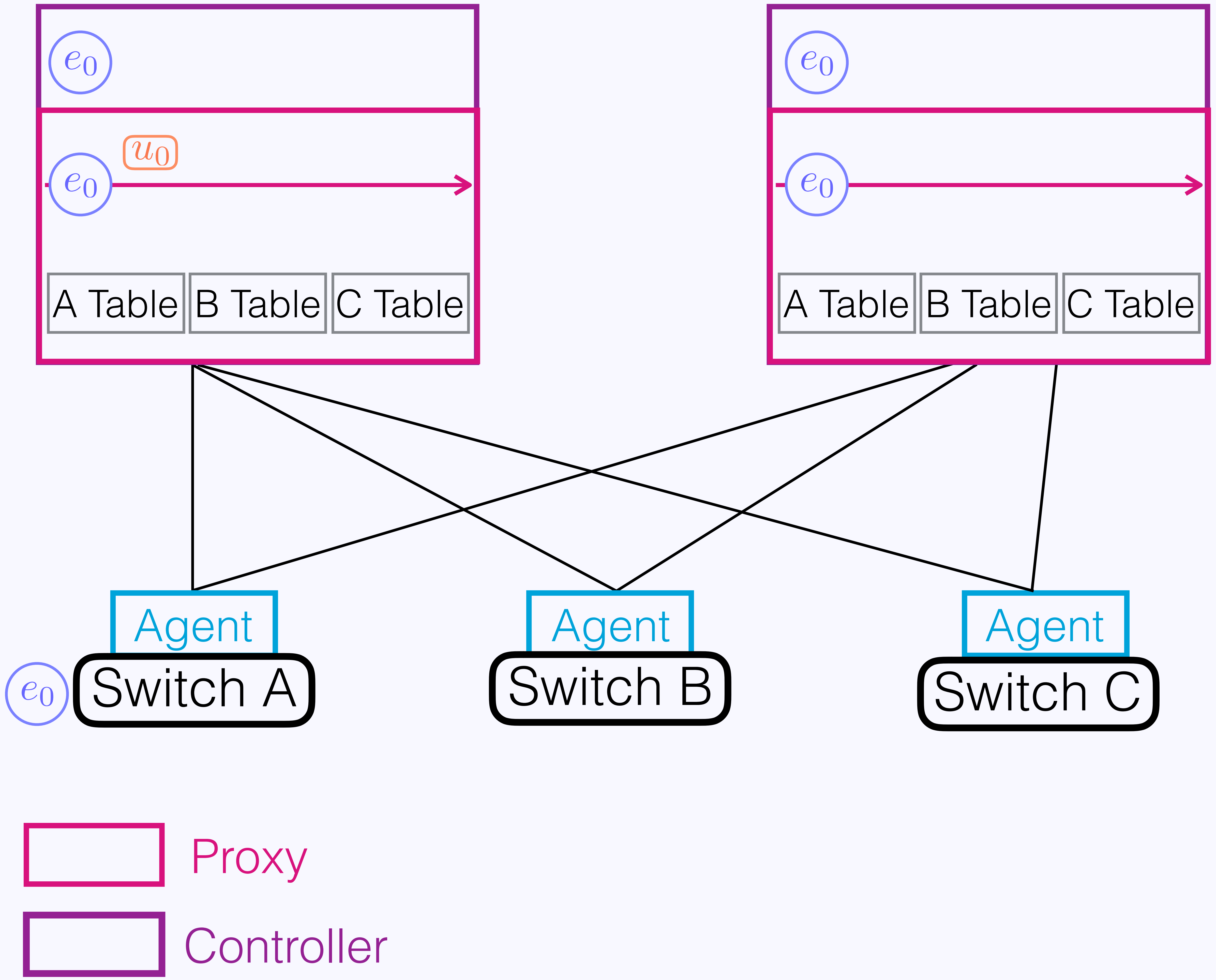
- Proxies maintain a log of all prior network events.
- All switch events are sent to all proxies.
- Proxy triggers controller computation.

SCL Proxies and Controllers



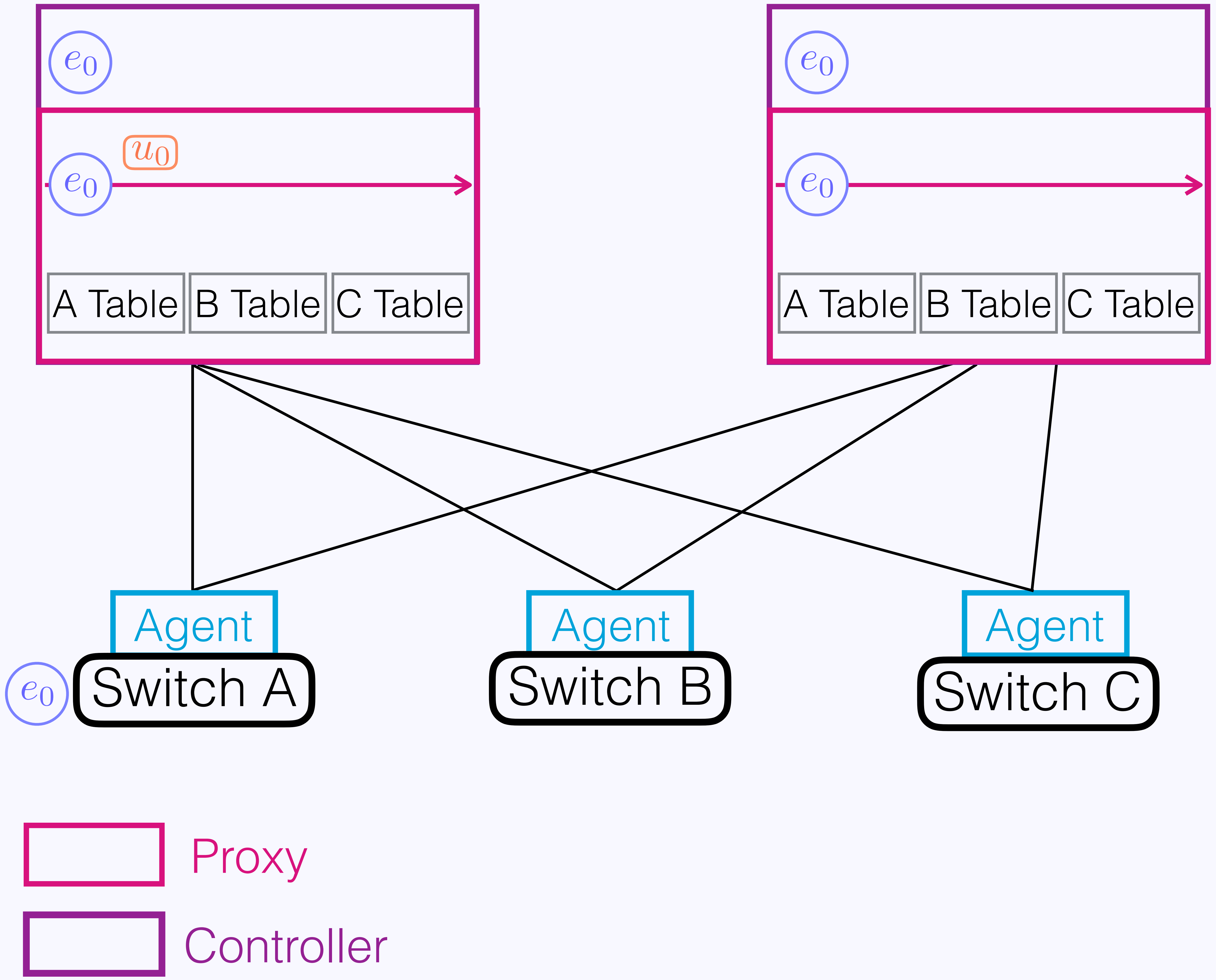
- Proxies maintain a log of all prior network events.
- All switch events are sent to all proxies.
- Proxy triggers controller computation.
    - Computation based on current log.

SCL Proxies and Controllers



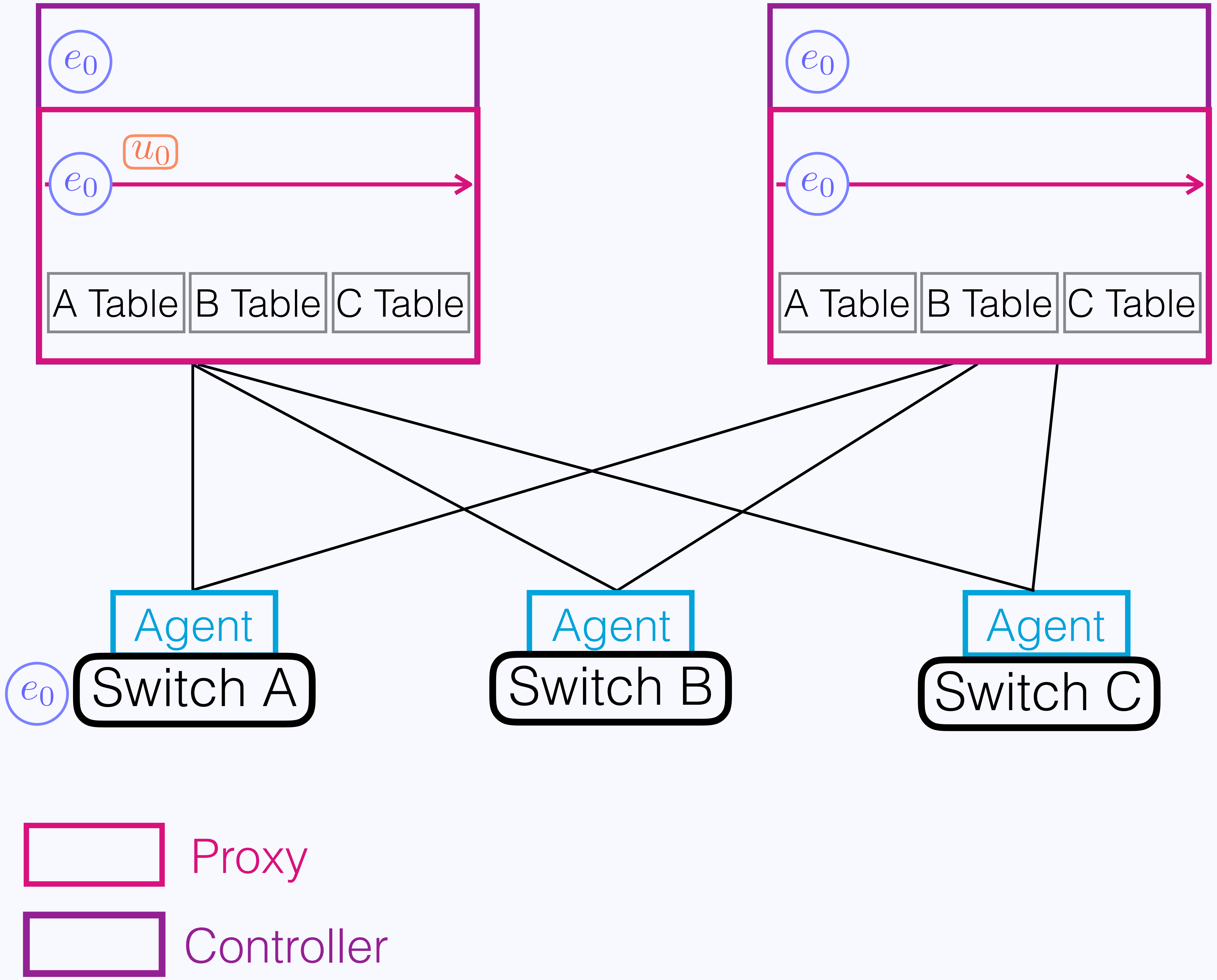
- Proxies maintain a log of all prior network events.
- All switch events are sent to all proxies.
- Proxy triggers controller computation.
  - Computation based on current log.
- Controller sends updates to proxy.

SCL Proxies and Controllers



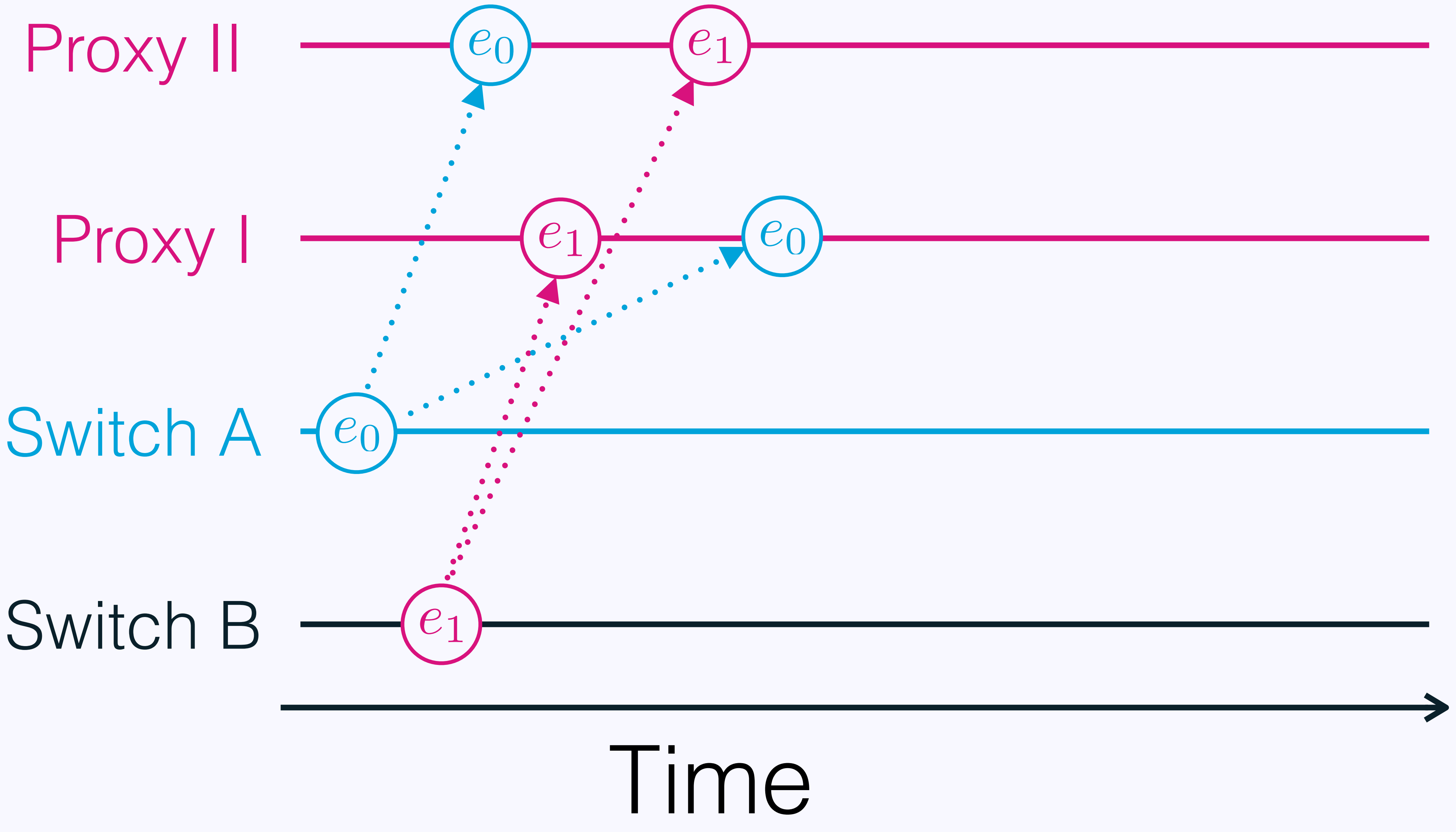
- Proxies maintain a log of all prior network events.
- All switch events are sent to all proxies.
- Proxy triggers controller computation.
  - Computation based on current log.
- Controller sends updates to proxy.
  - Proxy maintains state about installed rules.

SCL Proxies and Controllers



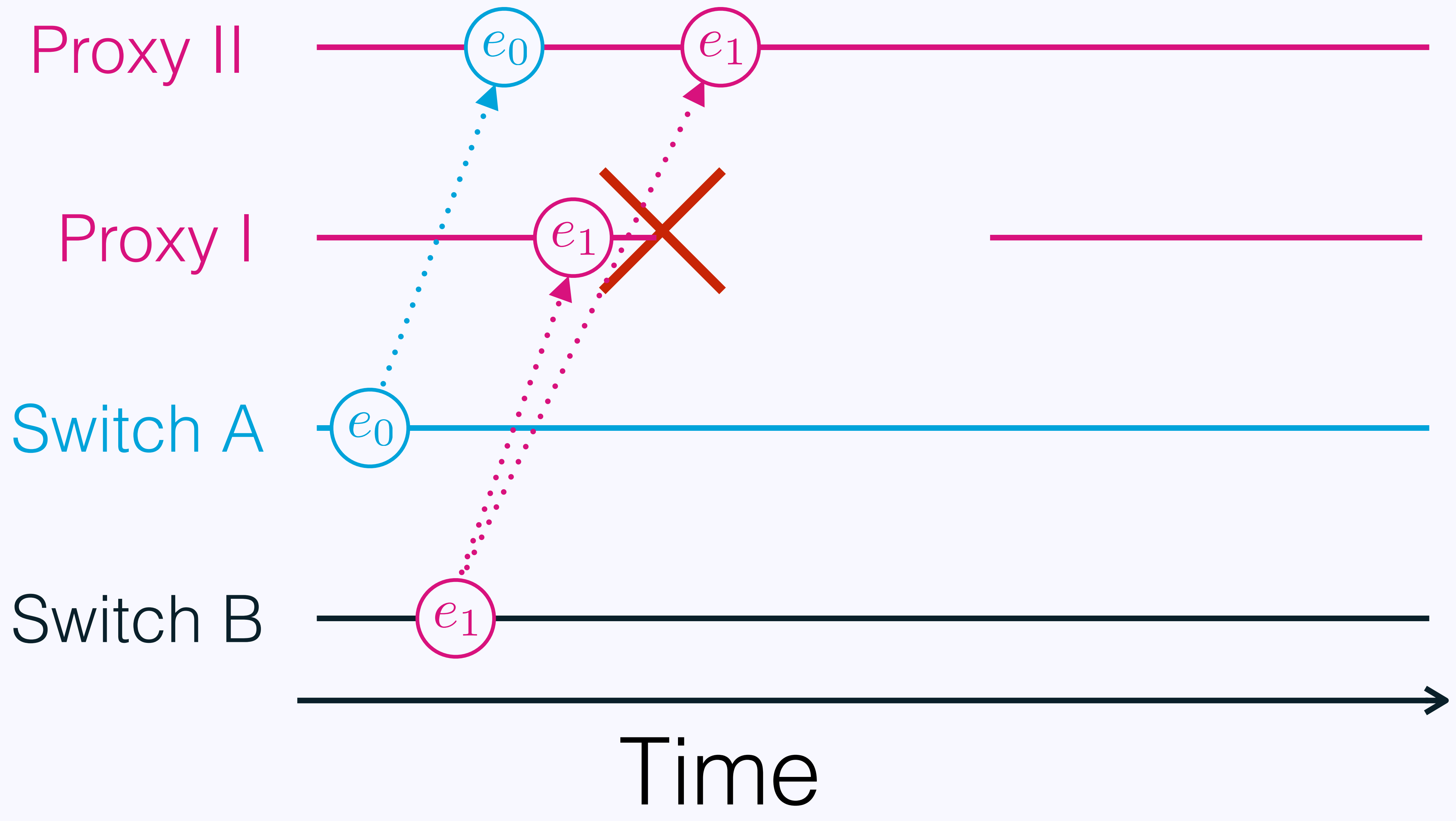
- Proxies maintain a log of all prior network events.
- All switch events are sent to all proxies.
- Proxy triggers controller computation.
  - Computation based on current log.
- Controller sends updates to proxy.
  - Proxy maintains state about installed rules.
  - Deduplicates updates before applying them.

SCL Proxies and Controllers: Challenges



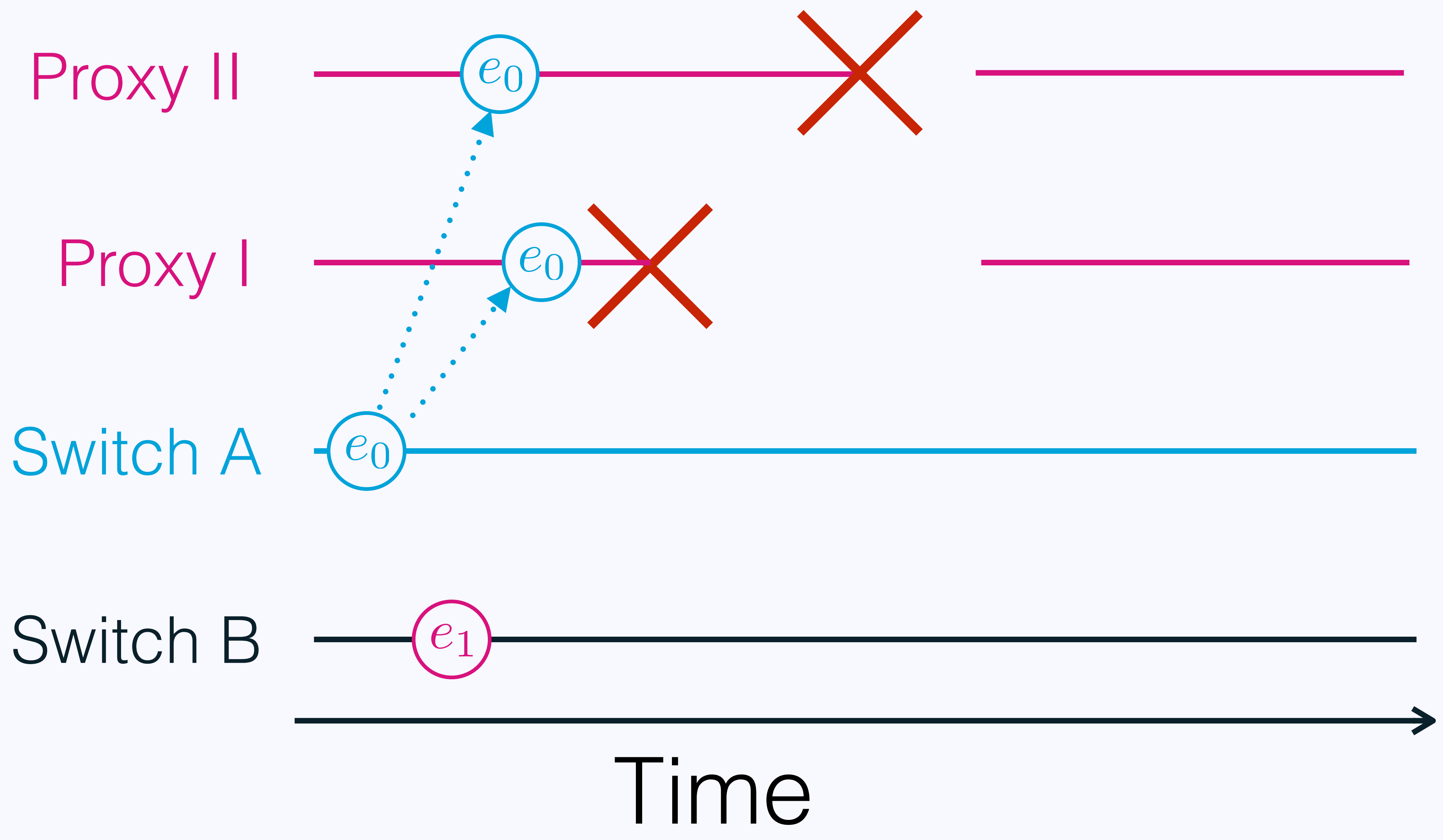
- **Agreement:** Proxies must eventually agree on order.

SCL Proxies and Controllers: Challenges



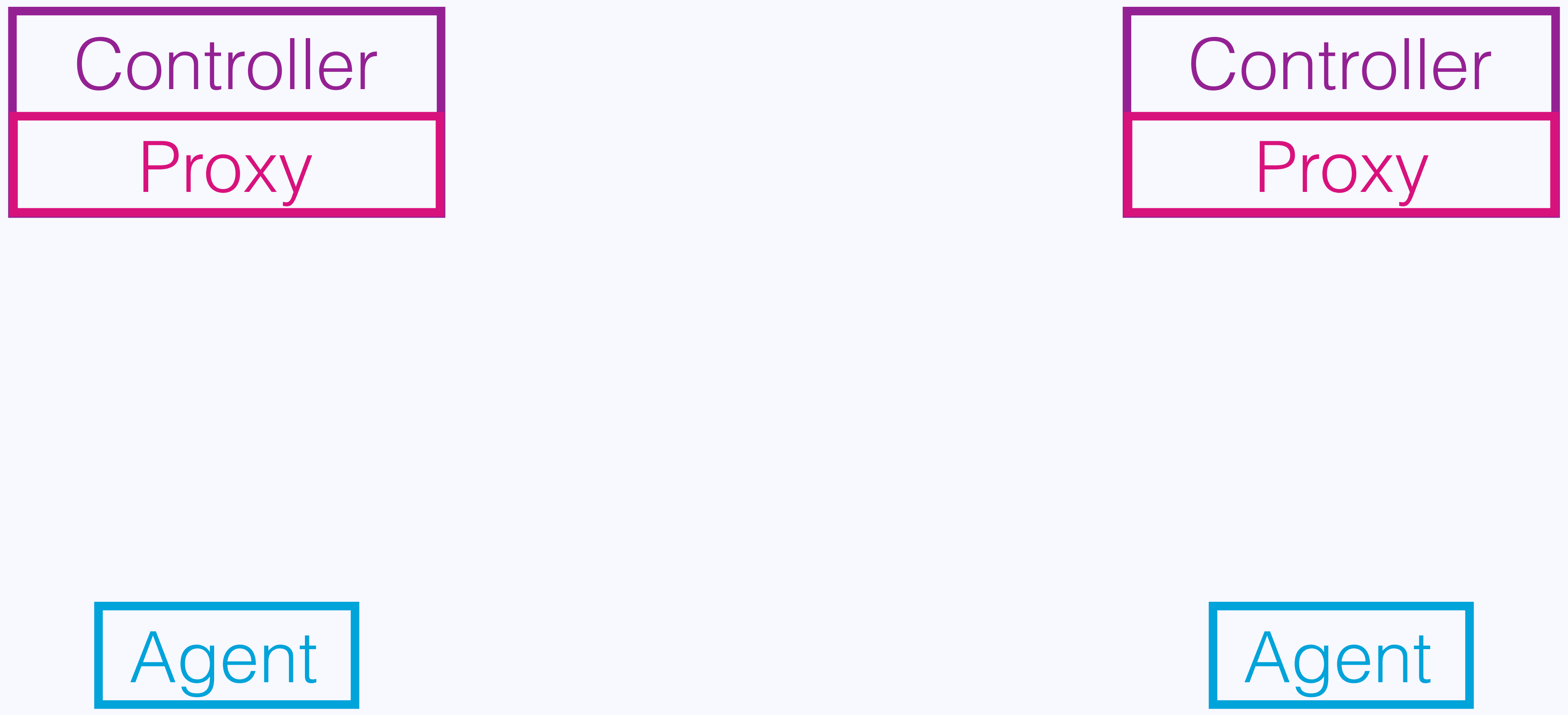
- **Agreement:** Proxies must eventually agree on order.
- **Agreement:** Must eventually agree on the set of events.

SCL Proxies and Controllers: Challenges



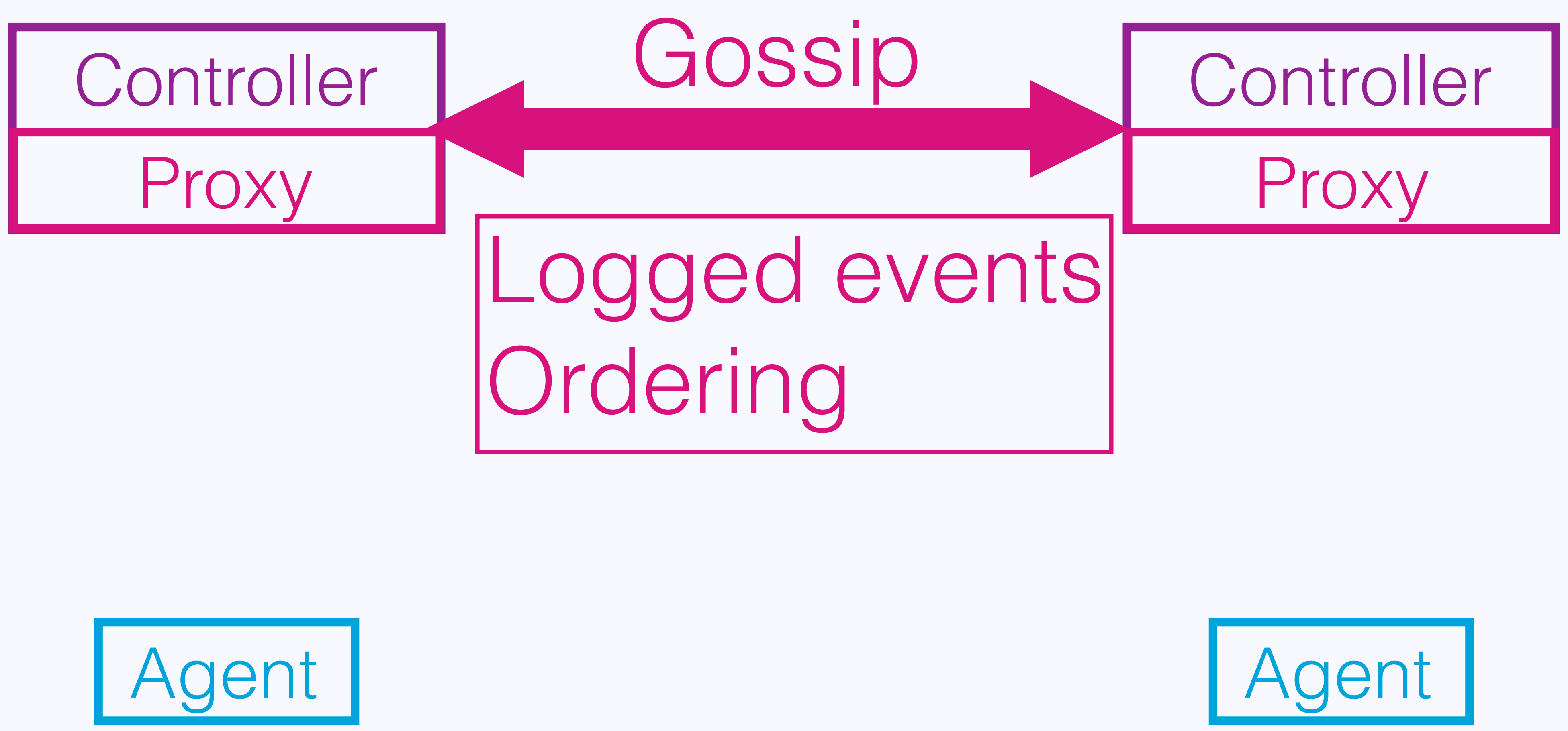
- **Agreement:** Proxies must eventually agree on order.
- **Agreement:** Must eventually agree on the set of events.
- **Awareness**: Controllers and network state agrees eventually.

Addressing SCL Challenges



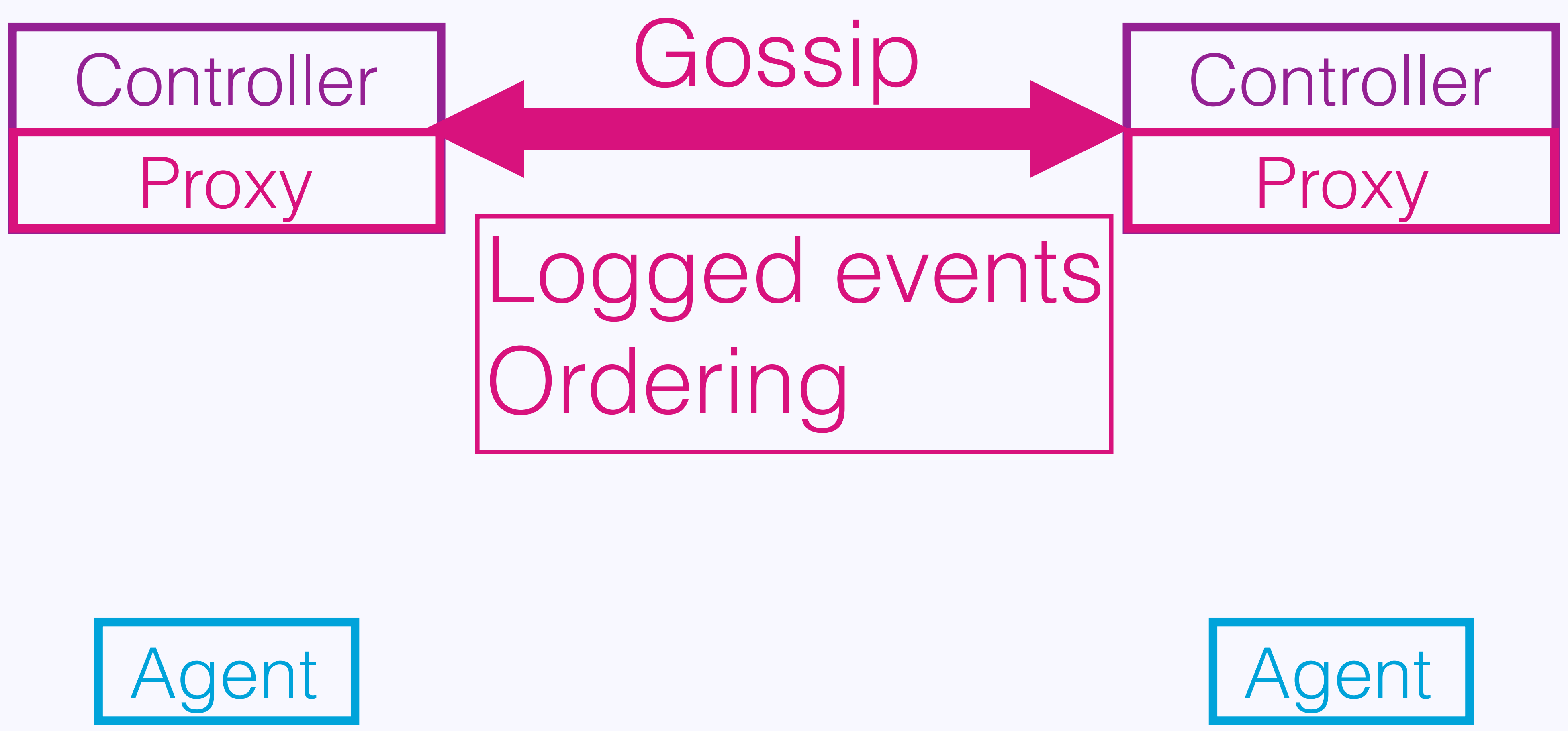
- Address these with two mechanisms.

Addressing SCL Challenges



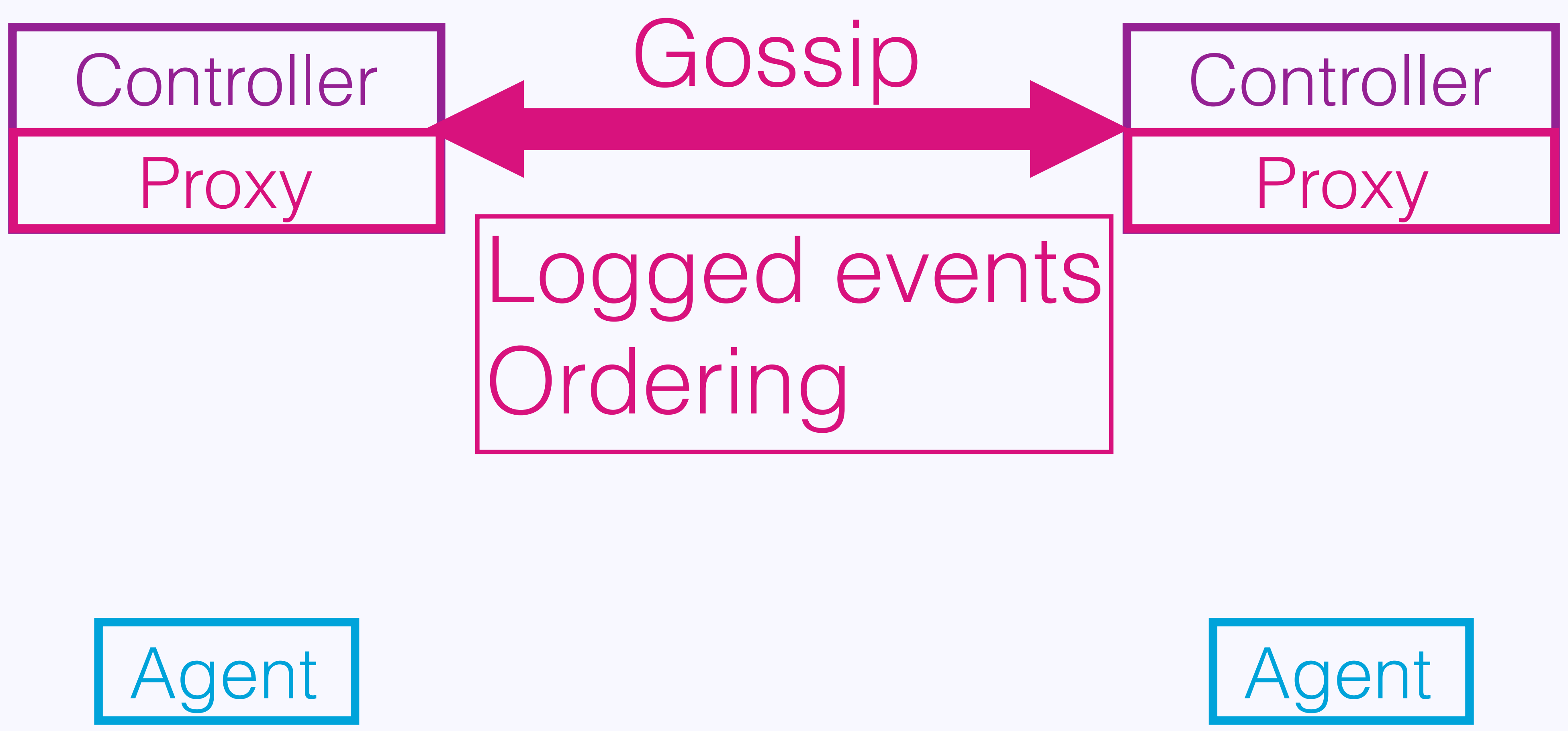
- Address these with two mechanisms.
- **Gossip** between controllers

Addressing SCL Challenges



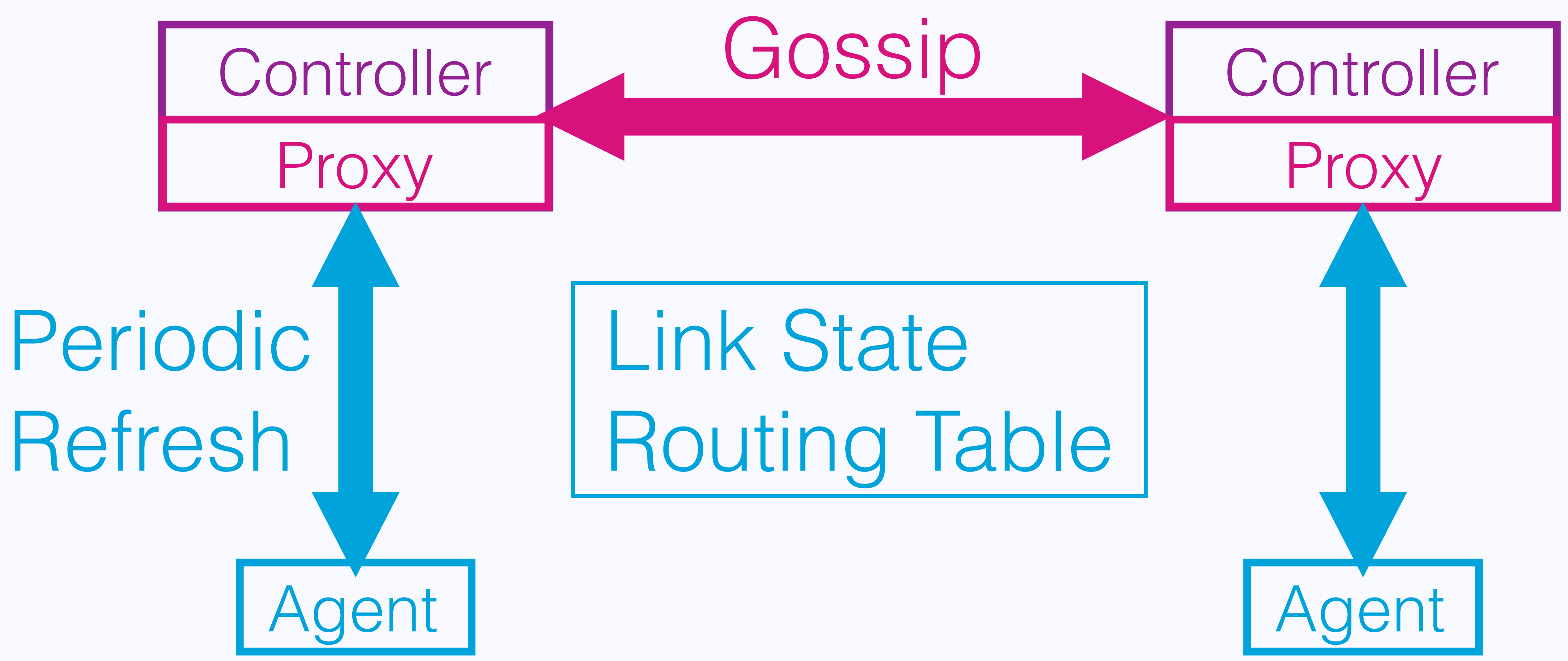
- Address these with two mechanisms.
- **Gossip** between controllers
  - Eventual **agreement** on observed **events**.

Addressing SCL Challenges



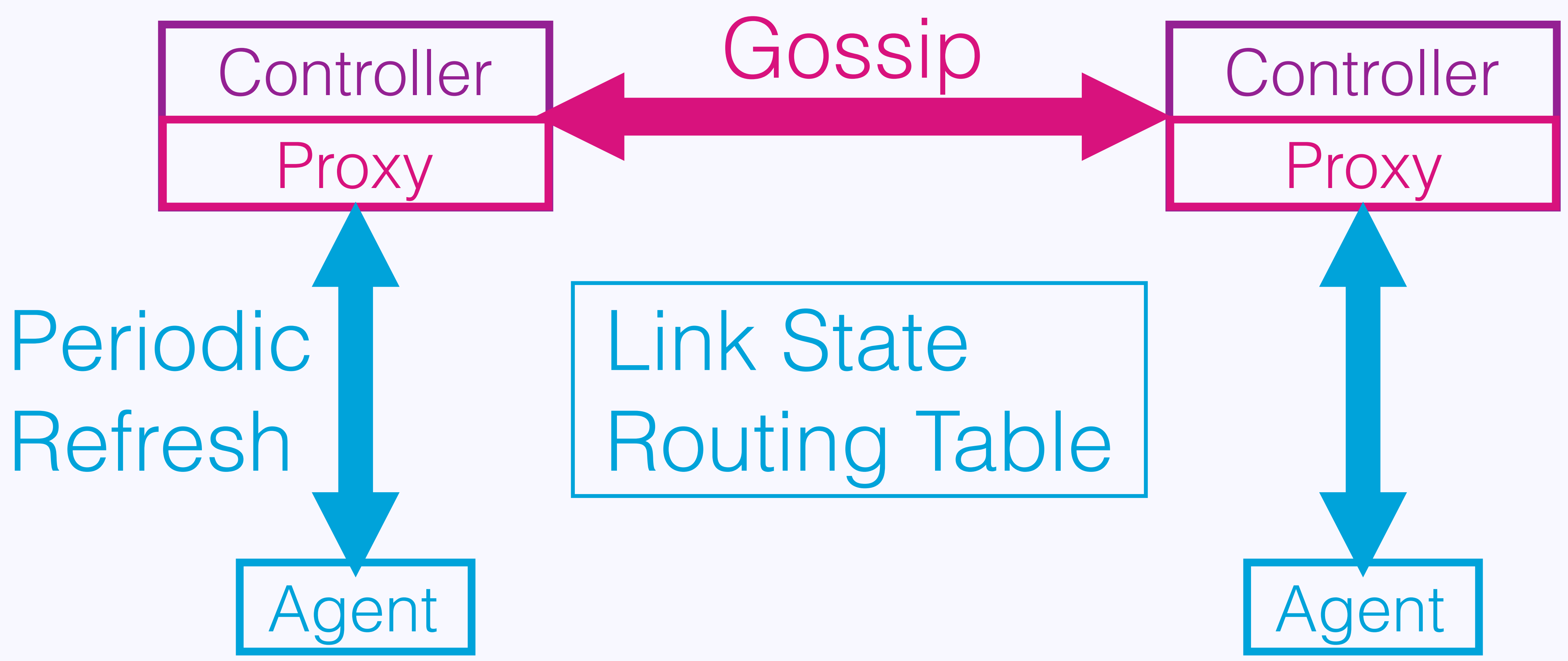
- Address these with two mechanisms.
- **Gossip** between controllers
  - Eventual **agreement** on observed **events**.
  - Also assures **agreement** on **ordering**.

Addressing SCL Challenges



- Address these with two mechanisms.
- **Gossip** between controllers
  - Eventual **agreement** on observed **events**.
  - Also assures **agreement** on **ordering**.
- **Periodically query** network for state.

Addressing SCL Challenges



- Address these with two mechanisms.
- **Gossip** between controllers
  - Eventual **agreement** on observed **events**.
  - Also assures **agreement** on **ordering**.
- **Periodically query** network for state.
  - Awareness of network state.

Why abandon consensus?

Conceptually Unnecessary

- **RSM assumption**: Truth about network lies in the controller.

Conceptually Unnecessary

- **RSM assumption**: Truth about network lies in the controller.
- **Reality**: Truth about the network lies within the network (dataplane).

Conceptually Unnecessary

- **RSM assumption**: Truth about network lies in the controller.
- **Reality**: Truth about the network lies within the network (dataplane).
  - Packets are processed by dataplane not by controllers.

Improves Performance and Resilience

Consensus	
SCL	

Improves Performance and Resilience

	Responsiveness
Consensus	
SCL	

Improves Performance and Resilience

	Responsiveness
Consensus	At least **1 RTT** between controllers
SCL	

Improves Performance and Resilience

	Responsiveness
Consensus	At least **1 RTT** between controllers
SCL	Respond **immediately**

Improves Performance and Resilience

	Responsiveness	**Scalability**
Consensus	At least **1 RTT** between controllers	
SCL	Respond **immediately**	

Improves Performance and Resilience

	Responsiveness	Scalability
Consensus	At least **1 RTT** between controllers	Latency **increases** with participants
SCL	Respond **immediately**	

Improves Performance and Resilience

	Responsiveness	Scalability
Consensus	At least **1 RTT** between controllers	Latency **increases** with participants
SCL	Respond **immediately**	**Does not increase** with # of participants

Improves Performance and Resilience

	Responsiveness	Scalability	Fault Tolerance
Consensus	At least **1 RTT** between controllers	Latency **increases** with participants	
SCL	Respond **immediately**	**Does not increase** with # of participants	

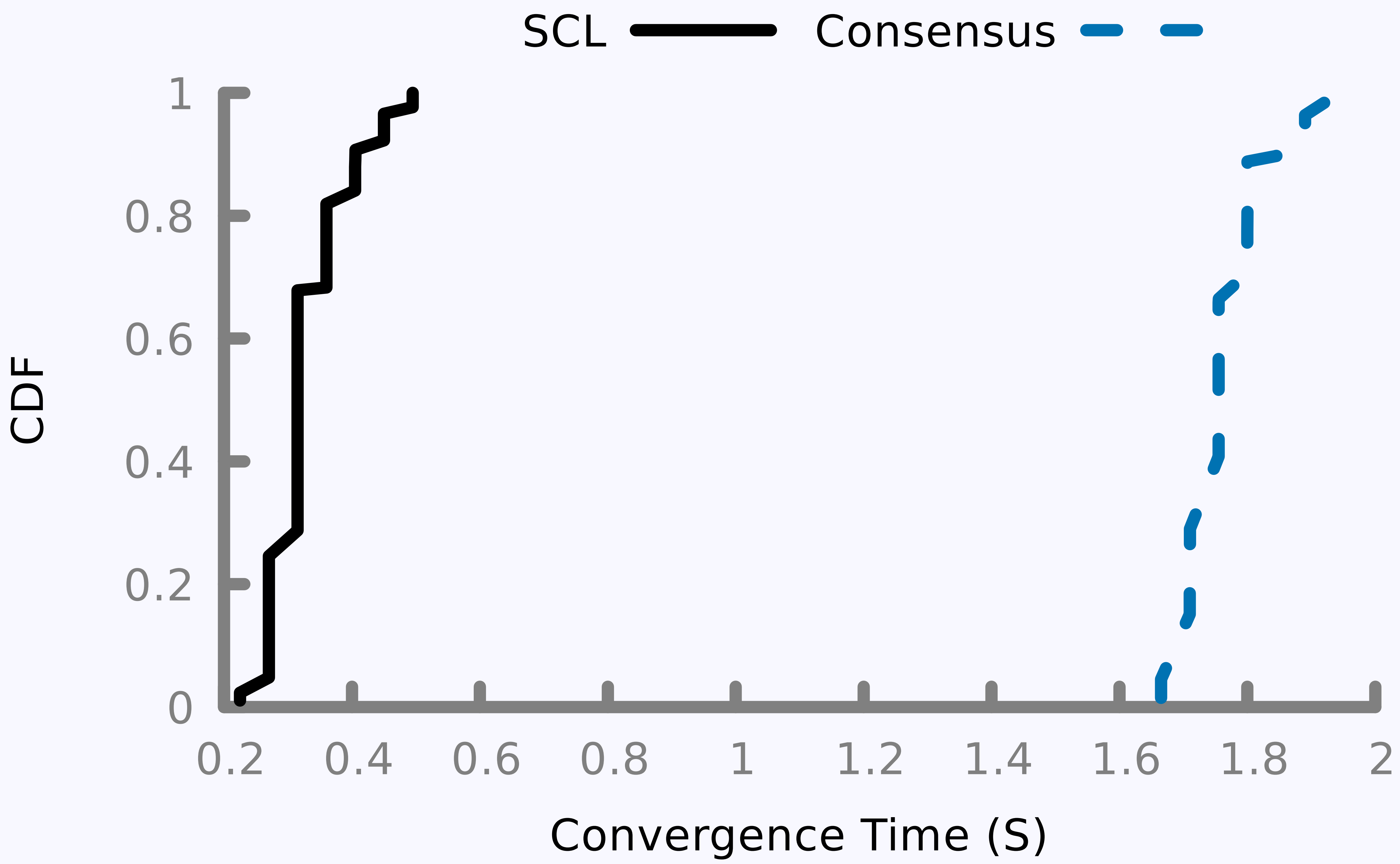
Improves Performance and Resilience

	Responsiveness	Scalability	Fault Tolerance
Consensus	At least **1 RTT** between controllers	Latency **increases** with participants	**Quorum** must be available for progress
SCL	Respond **immediately**	**Does not increase** with # of participants	

Improves Performance and Resilience

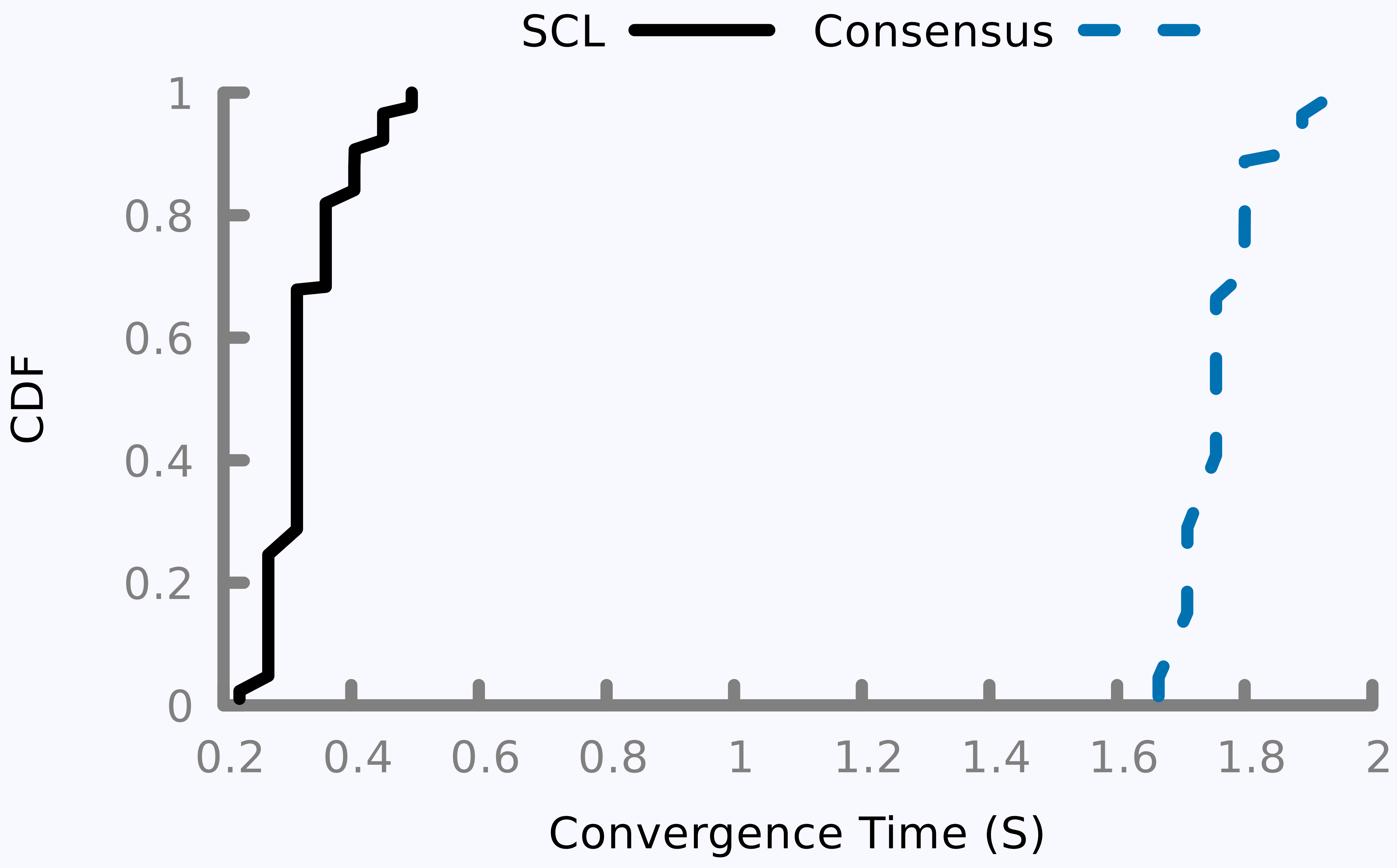
	Responsiveness	Scalability	Fault Tolerance
Consensus	At least **1 RTT** between controllers	Latency **increases** with participants	**Quorum** must be available for progress
SCL	Respond **immediately**	**Does not increase** with # of participants	Functional as long as a controller is **available**

What about Route Convergence?

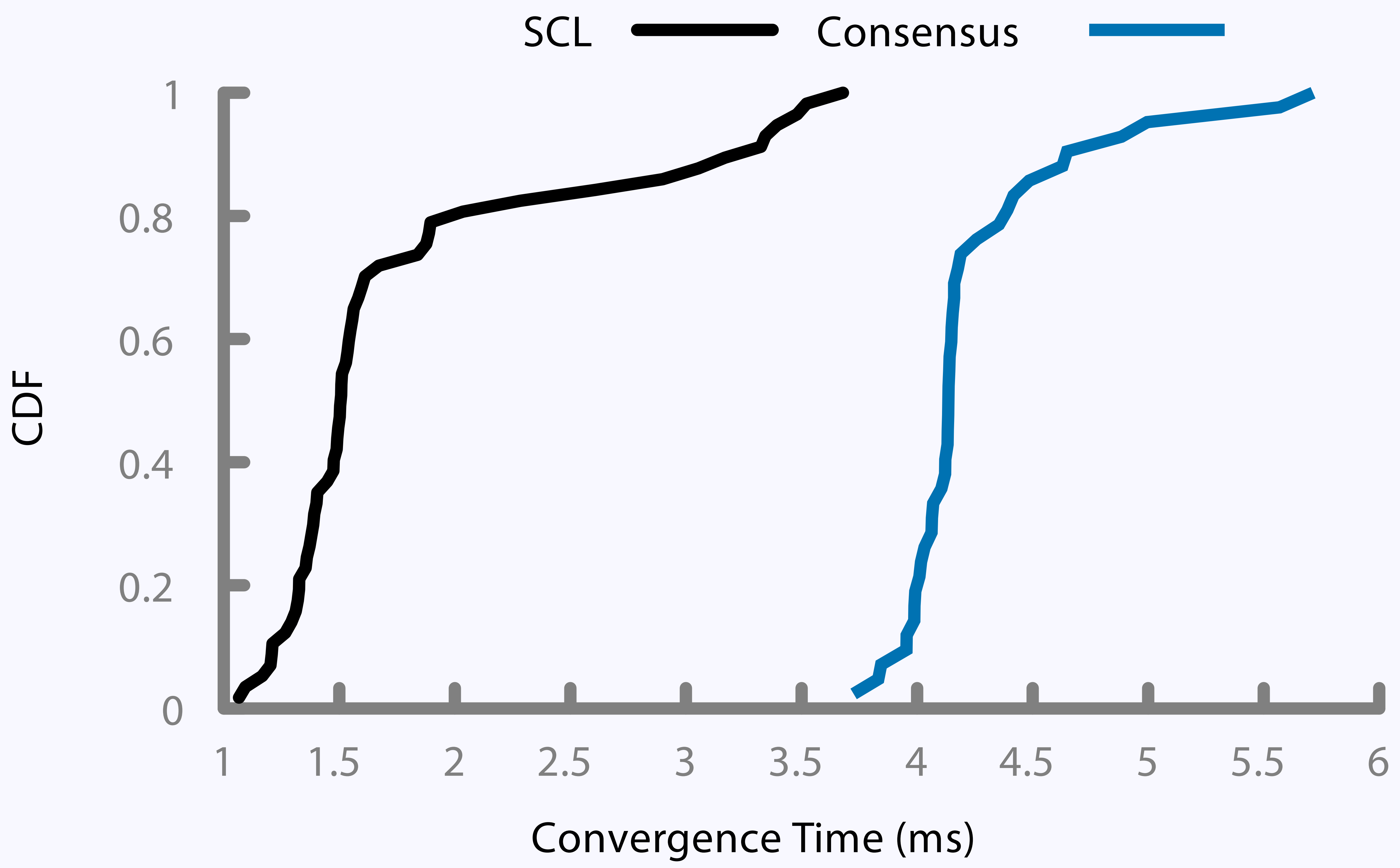


Convergence time in AS1221

What about Route Convergence?

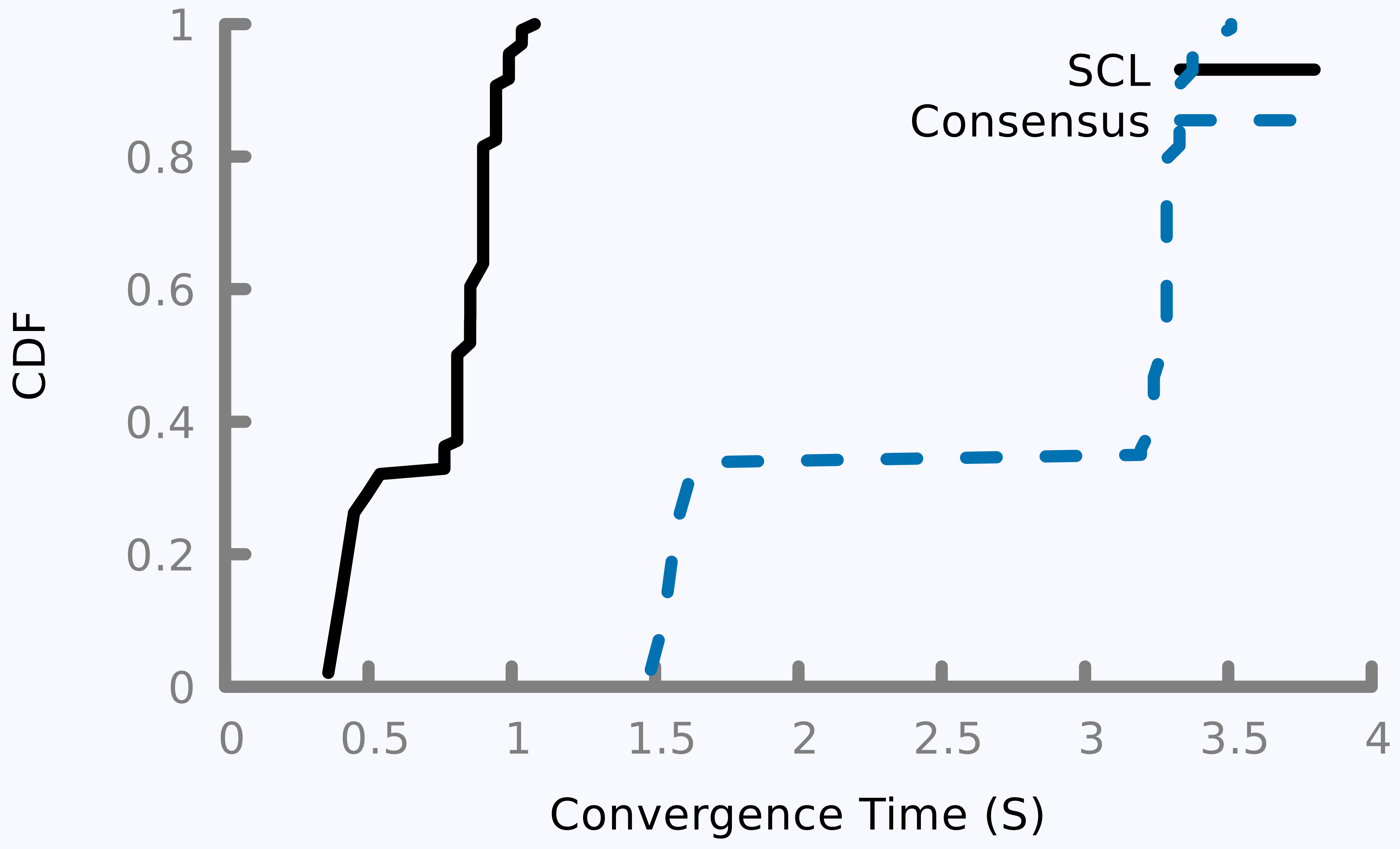


Convergence time in AS1221



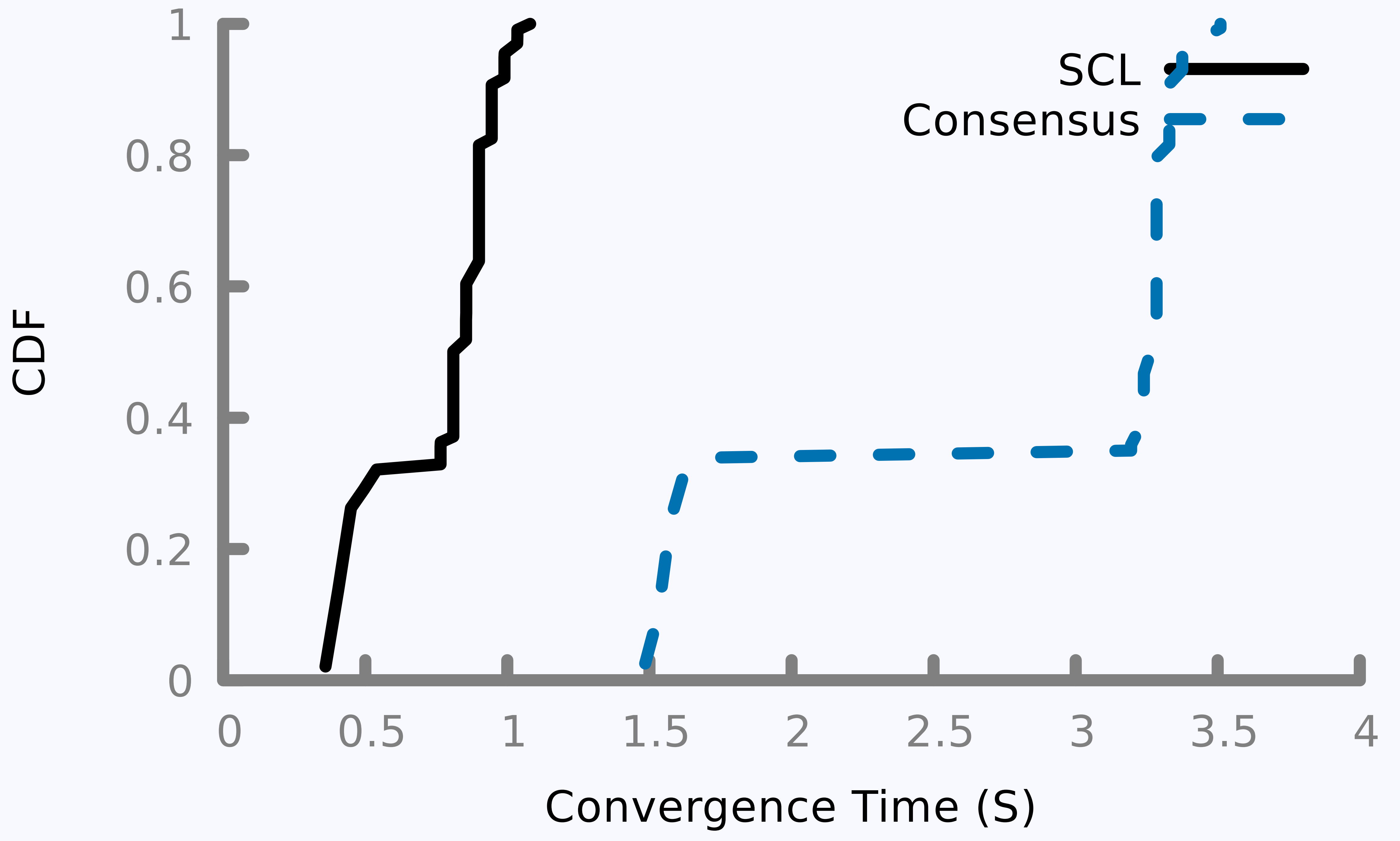
Convergence time for fat tree

When Does Everyone Agree?

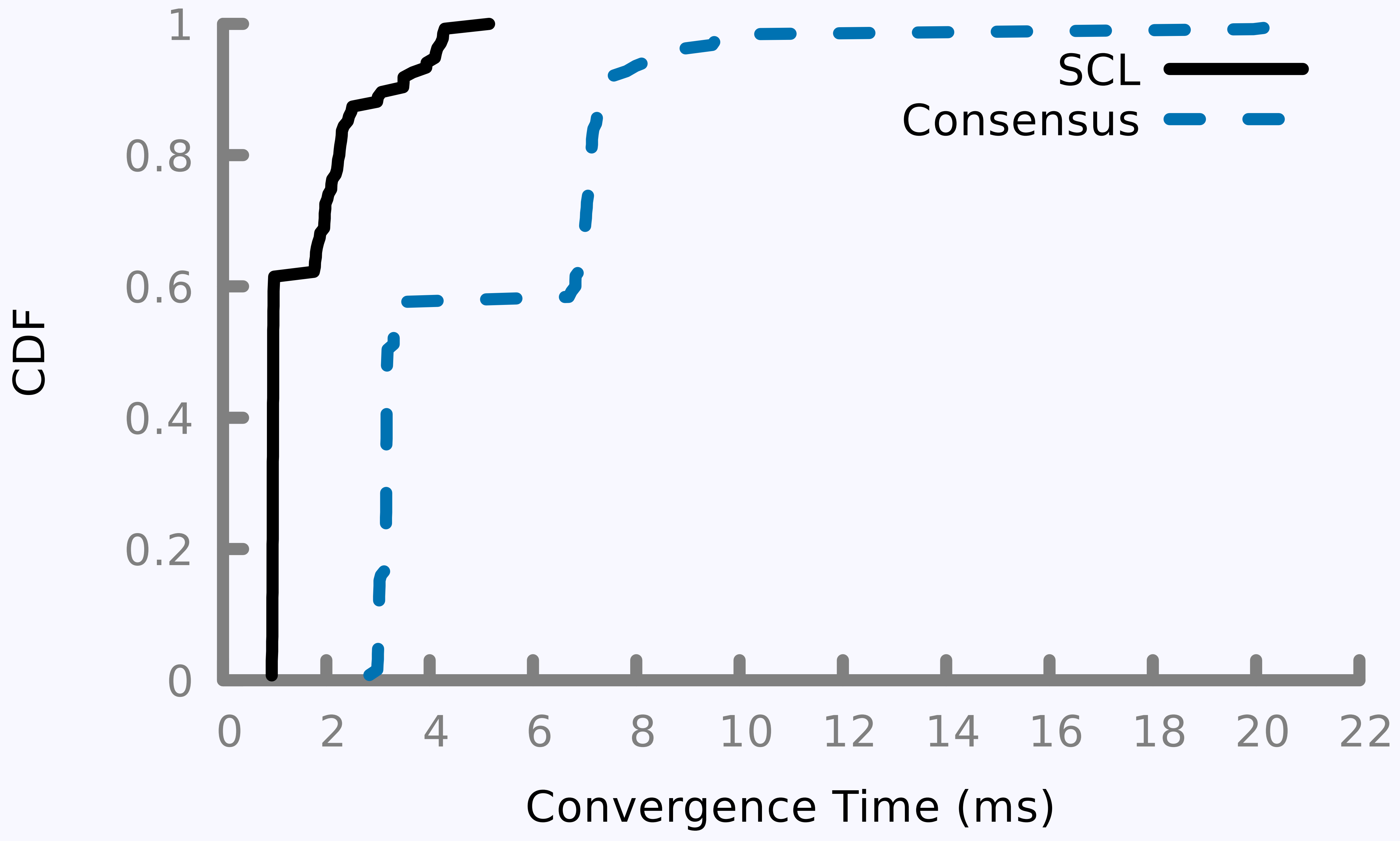


Convergence time in AS1221

When Does Everyone Agree?



Convergence time in AS1221



Convergence time in Fat Tree

In the Paper

- Proof that gossip and periodic update are sufficient to guarantee convergence.
- Broadcast based in-band control channels.
- Mechanisms for policy update.
- Interaction with other types of policies.
- Other performance results.

Related Work

Control Plane Consistency

Data Plane Consistency (Orthogonal)

Related Work

Control Plane Consistency

Serializability

Consensus: ONIX (OSDI'10), ONOS

Data Plane Consistency (Orthogonal)

Related Work

Control Plane Consistency

Serializability

Consensus: ONIX (OSDI'10), ONOS

Atomic registers: Schiff et al (CCR'16)

Data Plane Consistency (Orthogonal)

Related Work

Control Plane Consistency

Serializability

Consensus: ONIX (OSDI'10), ONOS

Atomic registers: Schiff et al (CCR'16)

Stronger Semantics

Exactly-Once: Ravana (SOSR'15)

Data Plane Consistency (Orthogonal)

Related Work

Control Plane Consistency

Serializability

Consensus: ONIX (OSDI'10), ONOS

Atomic registers: Schiff et al (CCR'16)

Stronger Semantics

Exactly-Once: Ravana (SOSR'15)

Data Plane Consistency (Orthogonal)

Labels: Reitblatt et al. (SIGCOMM '12)

Ordered Updates:
- Mahajan et al. (HotNets '13)
- McClurg et al. (PLDI '15)

Synchronized Clocks:
- Mirzahi et al. (SOSR '15)

Conclusion

- **Conventional wisdom**: Distributed SDN controllers need consensus.

Conclusion

- **Conventional wisdom**: Distributed SDN controllers need consensus.
- **This talk**: no consensus required.

Conclusion

- **Conventional wisdom**: Distributed SDN controllers need consensus.
- **This talk**: no consensus required.
    - Can use existing single image controllers with SCL.

Conclusion

- **Conventional wisdom**: Distributed SDN controllers need consensus.
- **This talk**: no consensus required.
  - Can use existing single image controllers with SCL.
- **Implication**

Conclusion

- **Conventional wisdom**: Distributed SDN controllers need consensus.
- **This talk**: no consensus required.
  - Can use existing single image controllers with SCL.
- **Implication**
  - Simplifies controllers.

Conclusion

- **Conventional wisdom**: Distributed SDN controllers need consensus.
- **This talk**: no consensus required.
  - Can use existing single image controllers with SCL.
- **Implication**
  - Simplifies controllers.
  - Improves convergence time, responsiveness, robustness.