SNIA DEVELOPER CONFERENCE

SDC 24

BY Developers FOR Developers

September 16-18, 2024
Santa Clara, CA

NVMeoF: Recovering from Loss of Communication

ECN117(ratified), TP4129 (ratified), TP8028, TP8032

Presented by Randy Jennings, Pure Storage

Agenda

- Examples of Problems to Prevent
- Time-based Recovery, including
  - Keep Alive Timeout (KATO)
  - Command Quiesce Time (CQT)
- Reconnect
- Lost Host Communication
- Cross-Controller Reset

- Not in scope: PCIe®
  - PCIe has tighter connectivity and other reset mechanisms.
  - Additionally, static controllers clear processing of commands before Controller Reset completes.
  - However, Time-based recovery is available to detect if Host processing is interrupted.

As explained in NVMe® Base 2.1 section 9.6.3.2

Common Terms

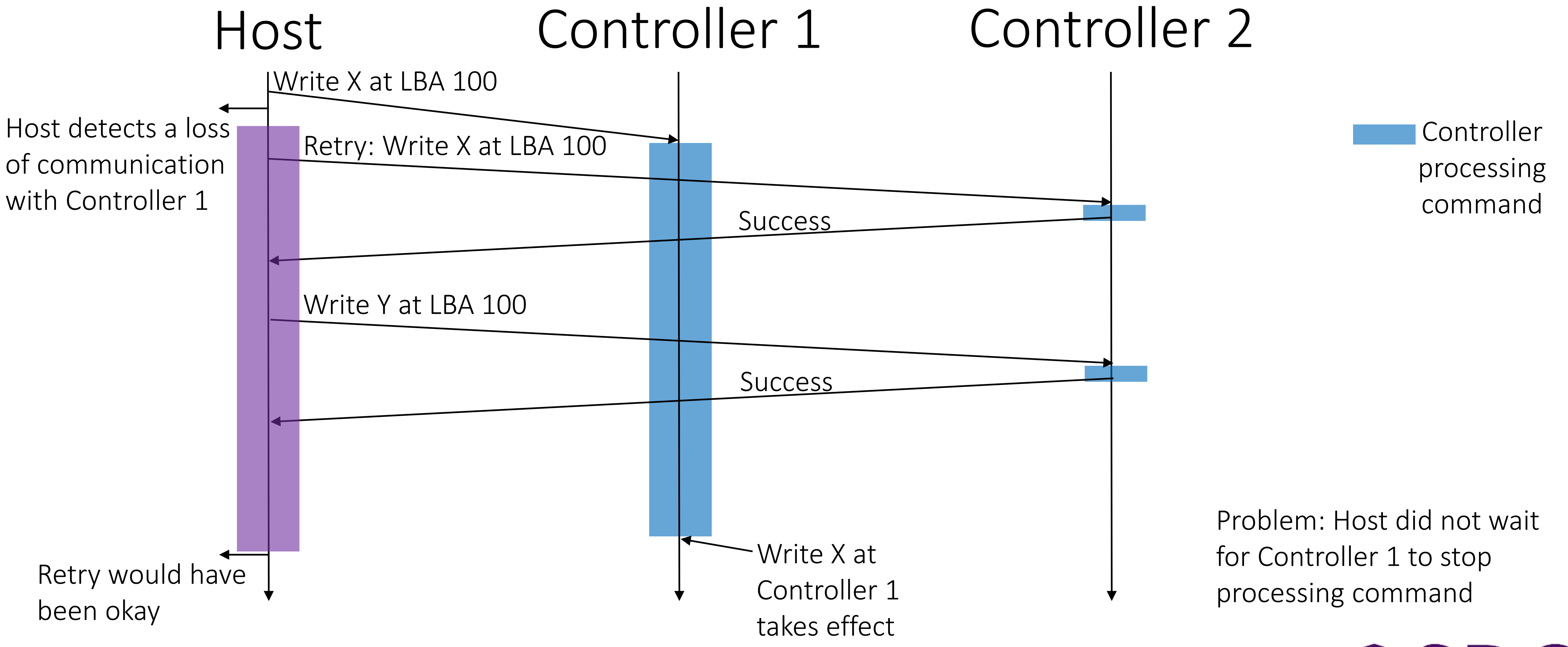
- Host
- NVM Subsystem
- Controller
  - Dynamic vs. Static
- Association
- Loss of Communication
  - (as used here) is the whole association, not just an IO queue
- Asynchronous Event Notificaiton (AEN)
- Log Page

As explained in NVMe® Base 2.1 section 9.6.3.2

Examples of Problems to Prevent

NVMe Base 2.1

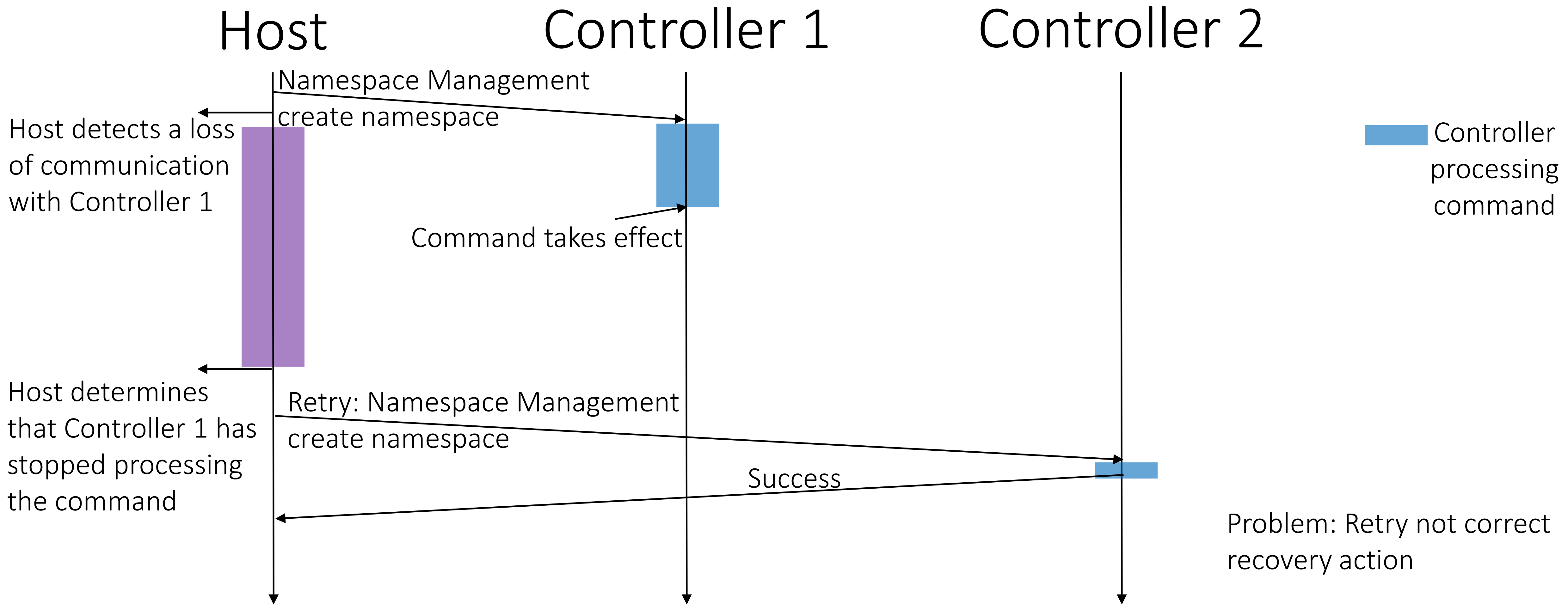
Bad Example 1: Ghost Write -> Corruption


Host
Controller 1
Controller 2
Write X at LBA 100
Host detects a loss of communication with Controller 1
Retry: Write X at LBA 100
Success
Write Y at LBA 100
Success
Retry would have been okay
Write X at Controller 1 takes effect
Controller processing command

Problem: Host did not wait for Controller 1 to stop processing command

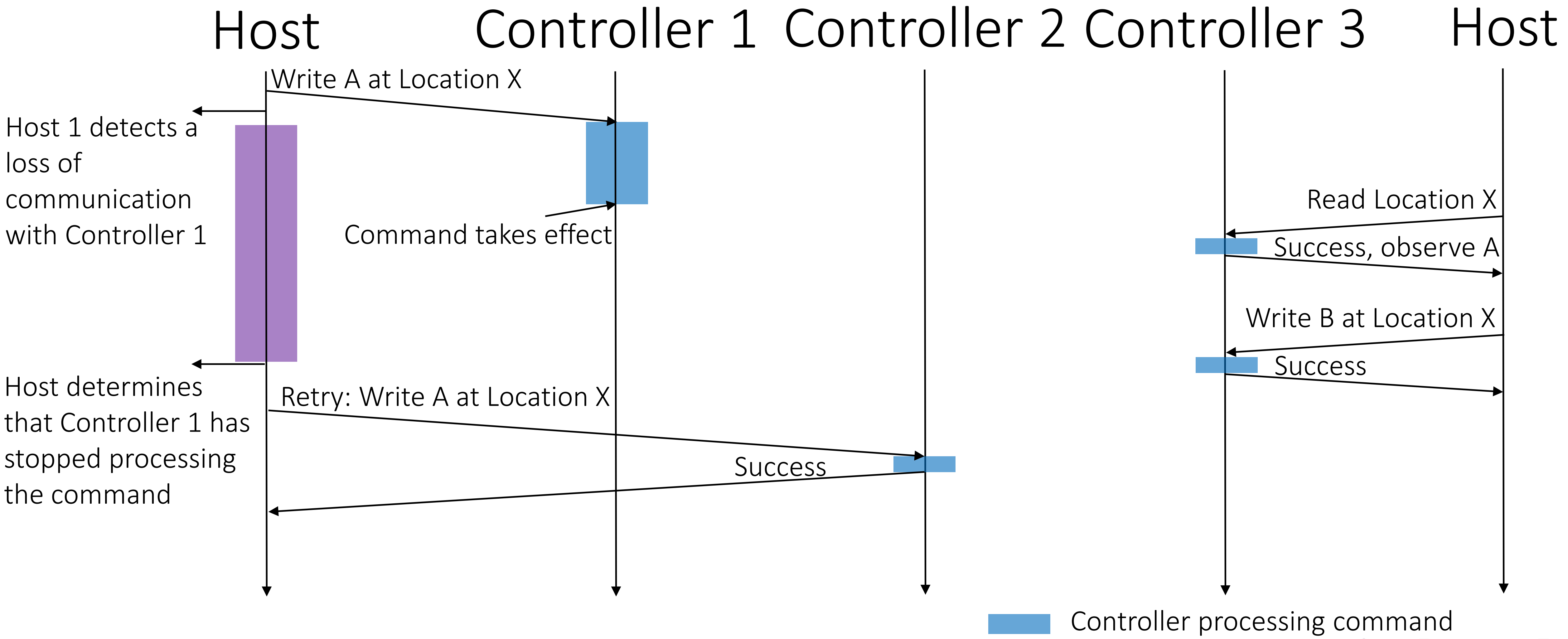
LBA 100 is X not Y!

Bad Example 2: Non-idempotent command


Host
Controller 1
Controller 2
Namespace Management create namespace
Host detects a loss of communication with Controller 1
Command takes effect
Host determines that Controller 1 has stopped processing the command
Retry: Namespace Management create namespace
Success
🟦 Controller processing command
Problem: Retry not correct recovery action

2 Namespaces are created

Bad Example 3: Idempotent command, but with other hosts


Host
Controller 1
Controller 2
Controller 3
Host
Write A at Location X
Host 1 detects a loss of communication with Controller 1
Command takes effect
Read Location X
Success, observe A
Write B at Location X
Success
Host determines that Controller 1 has stopped processing the command
Retry: Write A at Location X
Success
Controller processing command

Location X is A not B!

General Command Retry Categories

- Unrestricted Retry
  - An idempotent command that has no effect on user data or NVM subsystem state. The command is able to be retried without restrictions. (NVM Read)
- Delayed Retry
  - An idempotent command that modifies user data or NVM subsystem state. Any retry and/or reporting of the result of that command is required to be delayed until no further controller processing is possible of that command. (NVM Write, maybe)
- State-Dependent Retry
  - A non-idempotent command or an idempotent command that is able to affect the behavior of other hosts. The procedures for recovery depend on the extent, if any, to which that command has been processed by the controller. (Namespace Management)

As explained in NVMe Base 2.1 section 9.6.3.2

Time-based Recovery

NVMe Base 2.1

(ECN117, TP4129)

Before Host can recover specific commands

- Controller must detect Loss of Communication, examples:
  - Keep Alive Timeout (KATO): Controller does not receive traffic frequent enough to keep connection alive, establishes an upper bound for time to detect
  - Cable Pull: May not be detected on both sides
  - Host notified of loss of target by Fabric Controller (in Fibre Channel environment): Probably detected at nearly the same time
- Controller must cleanup processing commands
  - Command Quiesce Time (CQT): Controller specifies how long the Controller needs in order to stop processing commands

Time before Host can recover =

Time for Controller to detect + Time for Controller to stop processing

KATO: Command-based Keep Alive (CBKA)

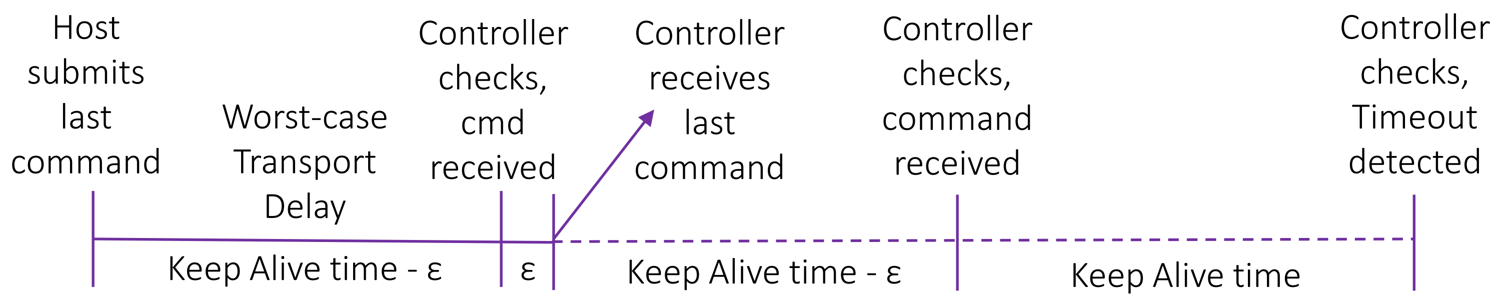
- Basic algorithm
  - Restart the Keep Alive Timer every time the Controller gets a:
    - Keep Alive command
    - Set Features for the Keep Alive Timer feature
  - If the Keep Alive Timer expires, you have a KATO
- Important Characteristics
  - Only involves Admin Queue
  - But, you can have active IO command processing, and still get a KATO
    - Host not sending Keep Alive command with the needed priority
  - Time to detect KATO is very predictable
    - Assume Keep Alive command delayed in the network by at most 1 KATT (Keep Alive Timer Time)
    - Detection happens at: command delay + timer expiration = 2 * KATT

KATO: Traffic-based Keep Alive (TBKA)

- Basic algorithm
  - Restart the Keep Alive Timer every time:
    - the Controller gets a Keep Alive command
    - the Controller gets a Set Features command for the Keep Alive Timer feature
    - the Keep Alive Timer expires without a KATO
  - If the Controller receives an IO or a different Admin command, set a flag.
  - If the Keep Alive Timer expires, and the flag is not set, you have a KATO.
- Important Characteristics
  - Arrival of any command on any queue keeps the controller alive.
    - Even if queues are bound to specific CPUs, traffic flag can be set by multiple CPUs.
  - But, as long as the Controller is receiving traffic the Controller stays alive.
  - How long does the Controller take to detect KATO?

KATO: Traffic-based Keep Alive, Time to detect KATO

- Assumption: Transport (e.g., fabric, Host and Controller processing) can delay a command by at most 1 KATT (Keep Alive Timer Time).
  - Longest time where an association with a host can stay alive
- The Controller can receive a command arbitrarily close to when the Controller last checked. That means the next check will not detect KATO, the one after will.
- So, detection in 3 * KATT for Traffic-based Keep Alive.



Host submits last command
Worst-case Transport Delay
Controller checks, cmd received
Controller receives last command
Controller checks, command received
Controller checks, Timeout detected
Keep Alive time - ε
ε
Keep Alive time - ε
Keep Alive time

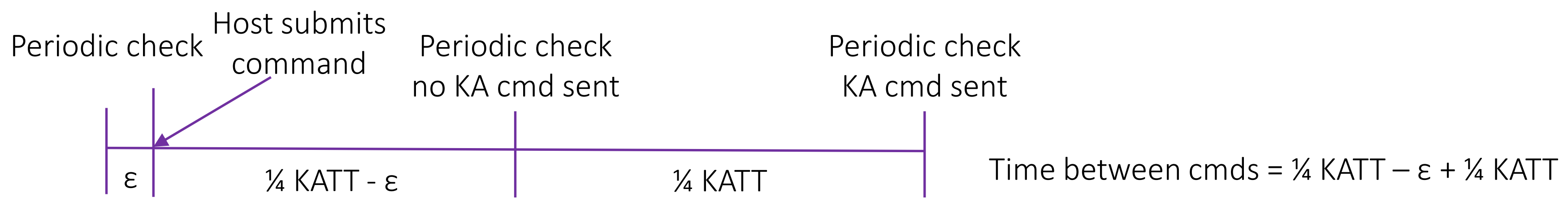
KATO: Host keeps Controller alive

- Host is free to implement any algorithm that keeps the controller alive.
- Controller picks between Command Based Keep Alive (CBKA) & Traffic Based Keep Alive (TBKA).
  - Host algorithm should be similar for both.
- Host does not know exactly when the Controller receives a command, only when the Host sent the command and when the Host receives the response.
  - Only consider commands where the Host sent & received a response within an interval to see if a condition was satisfied (e.g., for TBKA).
- Set Features completed for the Keep Alive Timer feature restarts the timer on the Controller.
  - The value for the interval could be less than what was set previously.
  - Note: The host does not know when the Set Features is received and if the Set Features will succeed when the command is sent. Consider expiration time carefully and/or conservatively.
- Host should send Keep Alive commands more frequently than the Controller checks for KATO.
  - The command takes time to get to the Controller. Suggested interval is ½ KATT..
- The example algorithm in the NVMe Base 2.1 spec satisfies these considerations.

KATO: Host, Why is the TBKA interval ¼, not ½?

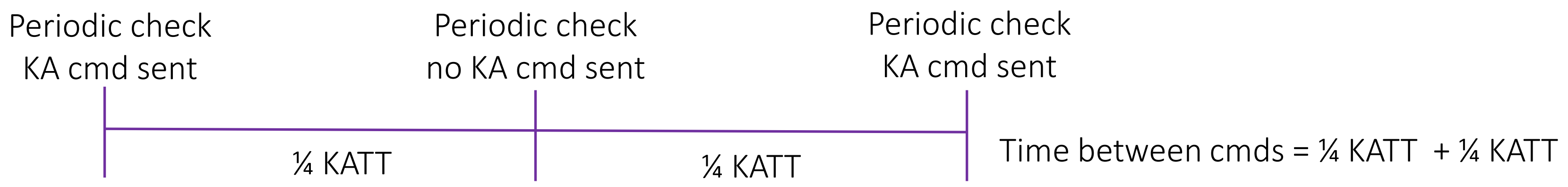
If we want to send Keep Alive commands every ½ KATT, why does the host need to check if it should send a Keep Alive command every ¼ KATT for TBKA?

KA cmd = Keep Alive Command



Periodic check
Host submits command
Periodic check no KA cmd sent
Periodic check KA cmd sent
ε
¼ KATT - ε
¼ KATT
Time between cmds = ¼ KATT – ε + ¼ KATT

Max gap between cmds is ½ Keep Alive Timer Time (KATT).



Periodic check KA cmd sent
Periodic check no KA cmd sent
Periodic check KA cmd sent
¼ KATT
¼ KATT
Time between cmds = ¼ KATT + ¼ KATT

Idle connection sends Keep Alive Command only every ½ Keep Alive time.

Making Recovery Faster

Time-based recovery, especially with KATO is loooong

Keep Alive Timeout Time (KATT) & Command Quiesce Time (CQT) must be sized for Worse-case situations. Most recoveries could end much faster, especially if the controller was prompted to begin recovery.*

Time before Host can recover =
Time for Controller to detect + Time for Controller to stop processing

- Given KATT = 10 seconds, CQT = 15 seconds
- Command Based Keep Alive:
  2 * KATT + CQT = 2 * 10 + 15 = 35 seconds
- Traffic Based Keep Alive:
  - 3 * KATT + CQT = 3 * 10 + 15 = 45 second

* Which is why any delay helps, even if it is not as long as it theoretically should be.

Fabrics-only mechanisms

- Reconnect (TP8032)
  - Sometimes the host, subsystem or network recovers quickly from a hiccup.
- Lost Host Communication (TP8028)
  - Helps Host detect loss of communication.
  - Host learns Controller has detected loss of communication.
- Cross-Controller Reset (TP8028)
  - Fabrics generally has multiple paths to the subsystem, let’s use them.
- Time-based Recovery (KATO/CQT) should still be running in the background.
  - The optimized mechanisms may fail.

Reconnect

TP 8032

Reconnect (TP8032)

- Reconnect to the same Controller over the same physical ports.
- Reconnect ensures that commands are no longer being processed.
- Specifics are rough; the Technical Proposal is early in the process.

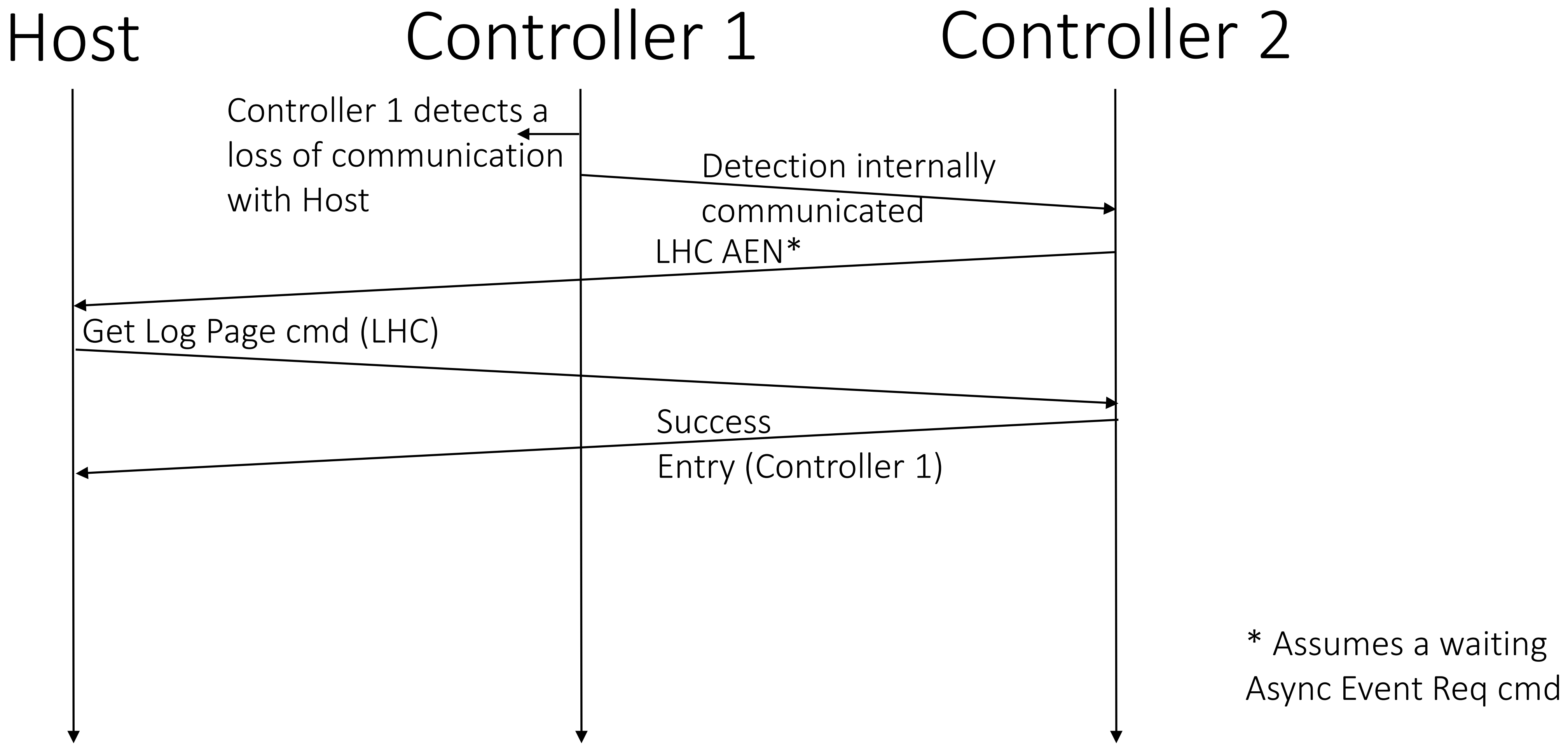
Lost Host Communication

TP8028

Lost Host Communication (LHC) AEN & Log Page

- Lets the host know about communication failure detected by the controller.
- Restricted to come from controllers connected to the same host as determined by NQN.
- Short-circuits waiting for x*KATO (not CQT).
- Sets up the host to do Cross-Controller Reset Recovery.
  - Source controller of AEN is a good candidate to receive Cross-Controller Reset command
- Technical Proposal in progress

Lost Host Communication (LHC) example


Host
Controller 1
Controller 2
Controller 1 detects a loss of communication with Host
Detection internally communicated
LHC AEN*
Get Log Page cmd (LHC)
Success
Entry (Controller 1)
* Assumes a waiting Async Event Req cmd

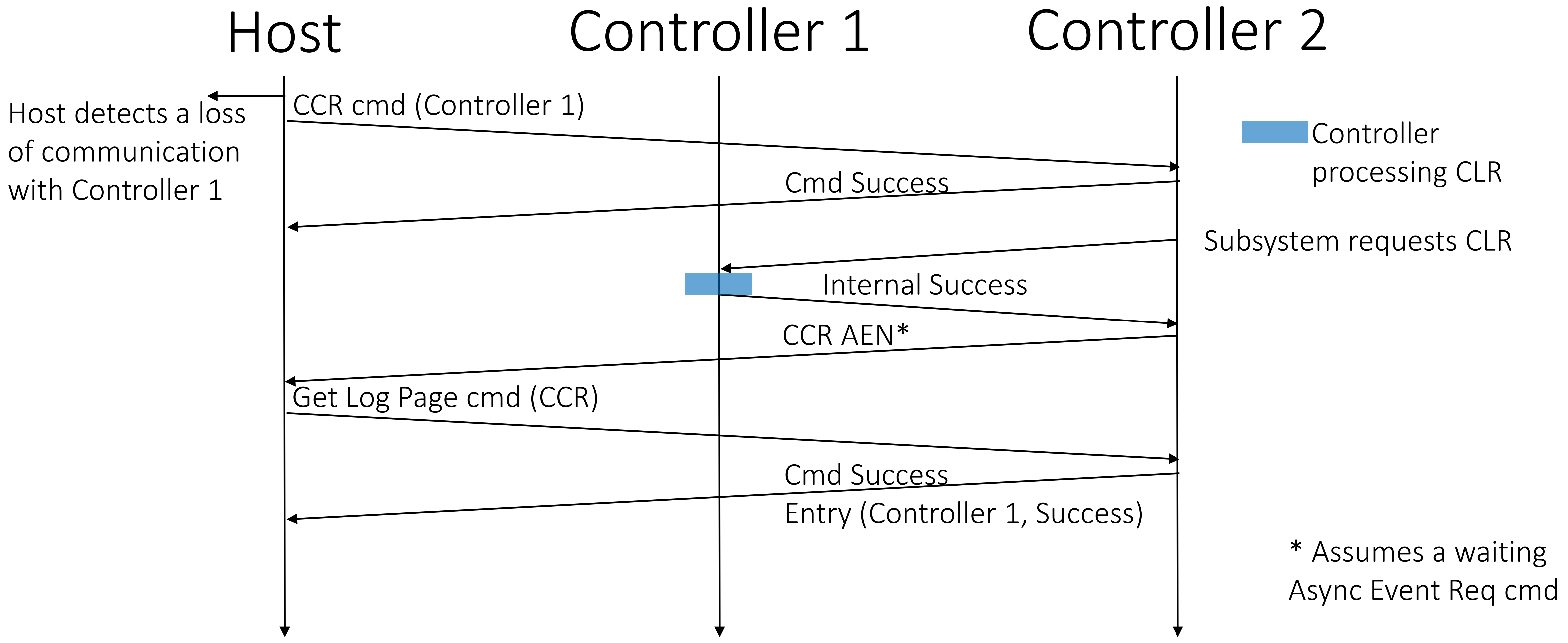
Cross Controller Reset Recovery

TP8028

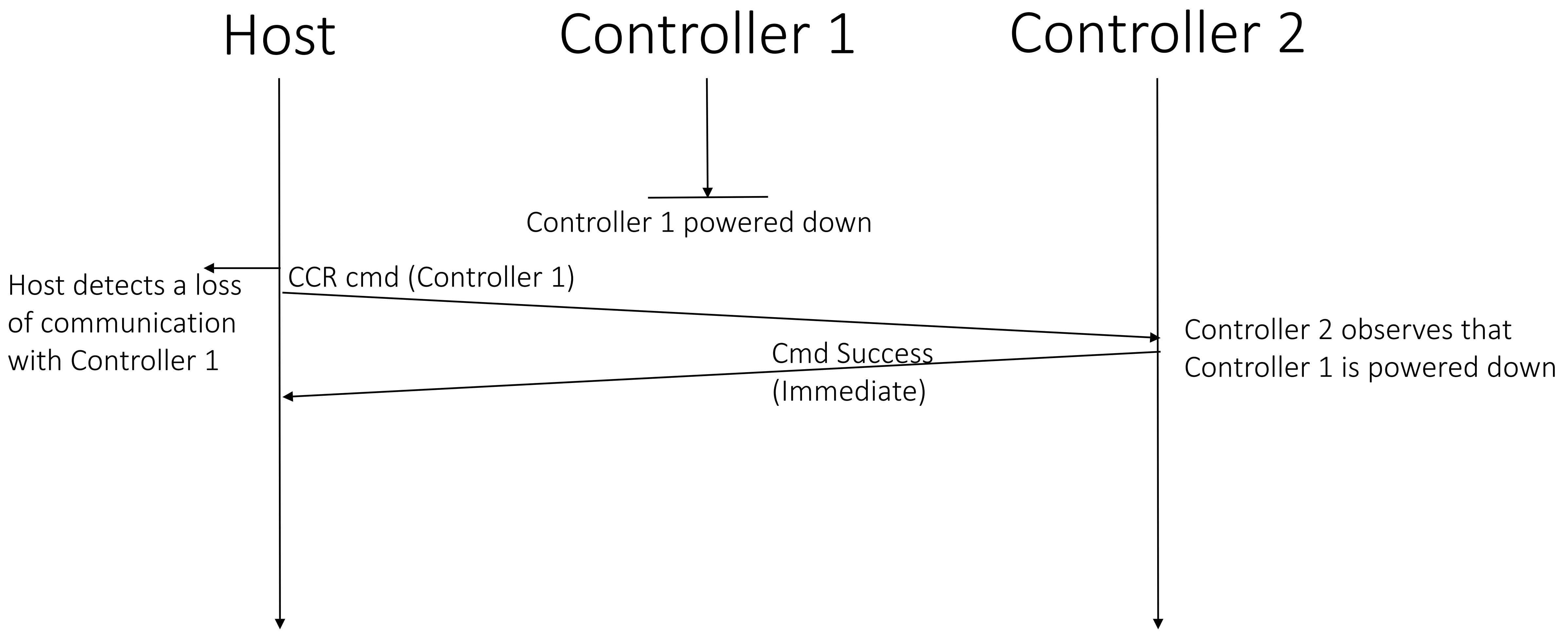
Cross-Controller Reset (CCR) Recovery

- The host triggers Controller Level Reset (CLR) on one controller through a different controller.
  - Both controllers must be connected to the same host.
- New CCR command starts a CCR operation running in the background.
- The host receives an AEN when the operation finishes.
- The host determines result by reading the CCR log page.
- To avoid spurious resets:
  - Dynamic Controllers: Subsystem should not aggressively reuse Controller ID.
  - Static Controllers: New state which changes on Connect or Authentication and Controller becoming Ready (CSTS.RDY ‘0’ -> ‘1’).
- Technical Proposal in progress

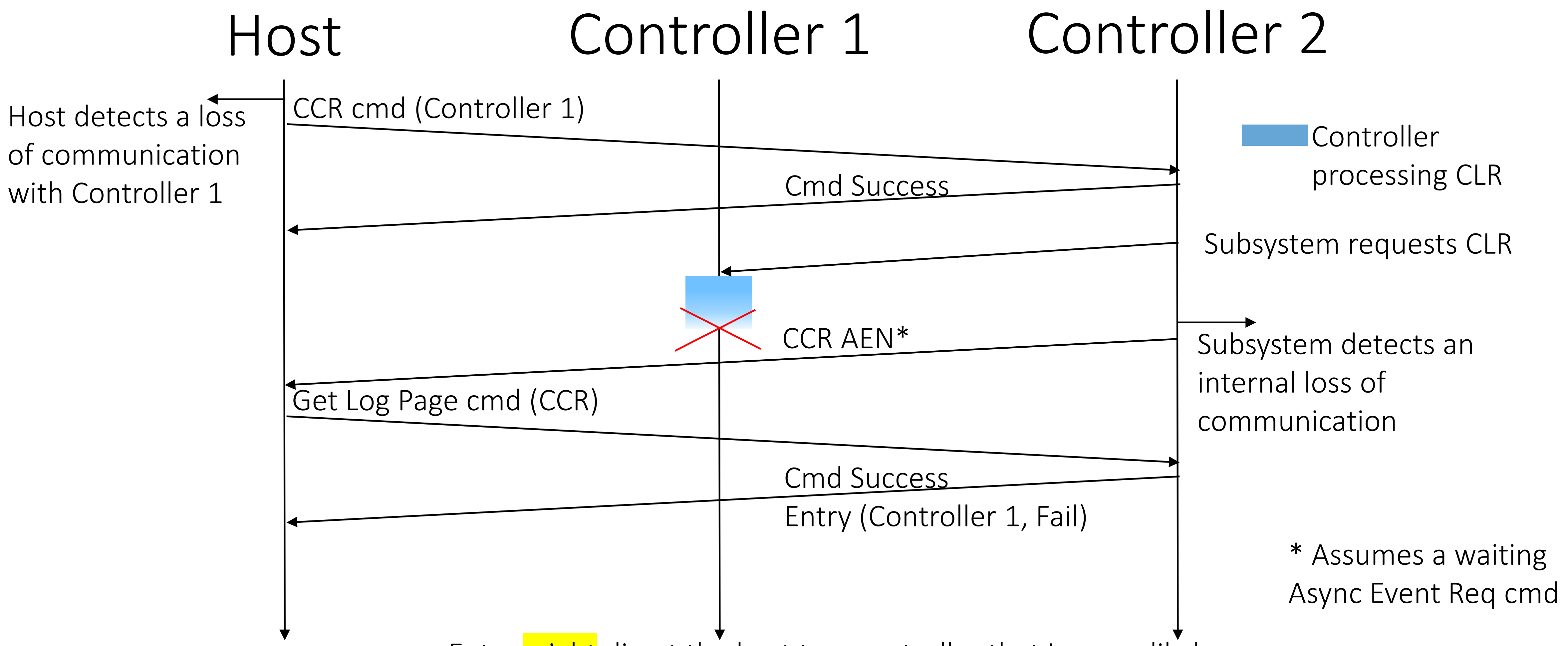
CCR Recovery example, Success


Host
Controller 1
Controller 2
Host detects a loss of communication with Controller 1
CCR cmd (Controller 1)
Controller processing CLR
Cmd Success
Subsystem requests CLR
Internal Success
CCR AEN*
Get Log Page cmd (CCR)
Cmd Success
Entry (Controller 1, Success)
* Assumes a waiting Async Event Req cmd

CCR Recovery example, Immediate Success


Host
Controller 1
Controller 2
Controller 1 powered down
Host detects a loss of communication with Controller 1
CCR cmd (Controller 1)
Controller 2 observes that Controller 1 is powered down
Cmd Success (Immediate)

CCR Recovery example, Failure



Host
Controller 1
Controller 2
Host detects a loss of communication with Controller 1
CCR cmd (Controller 1)
Controller processing CLR
Cmd Success
Subsystem requests CLR
CCR AEN*
Subsystem detects an internal loss of communication
Get Log Page cmd (CCR)
Cmd Success
Entry (Controller 1, Fail)
* Assumes a waiting Async Event Req cmd

Entry might direct the host to a controller that is more likely to perform a CCR operation successfully for Controller 1. (Alternative is reverting to time-base recovery)

Summary

- Problems we are preventing:
  - Data corruption
  - Repeated non-idempotent commands
- Time-based Recovery, including
  - Keep Alive Timeout (KATO)
  - Command Quiesce Time (CQT)

Time before Host can recover =
Time for Controller to detect + Time for Controller to stop processing

- Reconnect
- Lost Host Communication
- Cross-Controller Reset

As explained in NVMe® Base 2.1 section 9.6.3.2

Please take a moment to rate this session.

Your feedback is important to us.