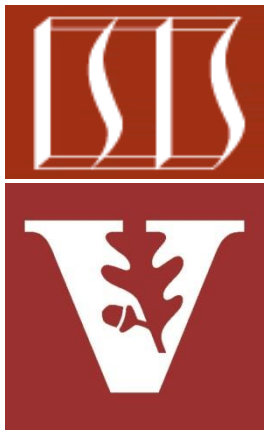
Visualizing a "Fair" Semaphore Via the Specific Notification Pattern

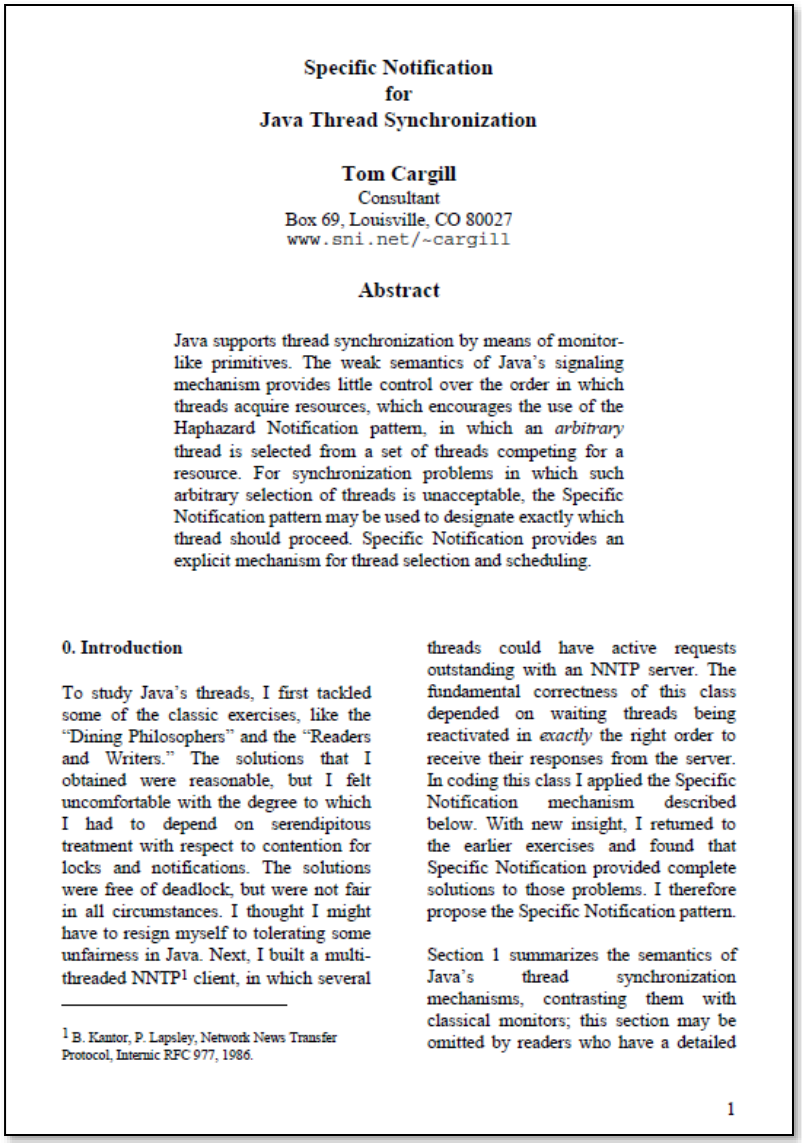


Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand the *Specific Notification* pattern
- Be aware of the semantics of "fair" semaphores
- Recognize how to visualize the implementation of a "fair" semaphore using the *Specific Notification* pattern



Specific Notification
for
Java Thread Synchronization

Tom Cargill
Consultant
Box 69, Louisville, CO 80027
www.sni.net/~cargill

Abstract

Java supports thread synchronization by means of monitor-like primitives. The weak semantics of Java's signaling mechanism provides little control over the order in which threads acquire resources, which encourages the use of the Haphazard Notification pattern, in which an *arbitrary* thread is selected from a set of threads competing for a resource. For synchronization problems in which such arbitrary selection of threads is unacceptable, the Specific Notification pattern may be used to designate exactly which thread should proceed. Specific Notification provides an explicit mechanism for thread selection and scheduling.

0. Introduction

To study Java's threads, I first tackled some of the classic exercises, like the "Dining Philosophers" and the "Readers and Writers." The solutions that I obtained were reasonable, but I felt uncomfortable with the degree to which I had to depend on serendipitous treatment with respect to contention for locks and notifications. The solutions were free of deadlock, but were not fair in all circumstances. I thought I might have to resign myself to tolerating some unfairness in Java. Next, I built a multi-threaded NNTP[1] client, in which several threads could have active requests outstanding with an NNTP server. The fundamental correctness of this class depended on waiting threads being reactivated in *exactly* the right order to receive their responses from the server. In coding this class I applied the Specific Notification mechanism described below. With new insight, I returned to the earlier exercises and found that Specific Notification provided complete solutions to those problems. I therefore propose the Specific Notification pattern.

Section 1 summarizes the semantics of Java's thread synchronization mechanisms, contrasting them with classical monitors; this section may be omitted by readers who have a detailed

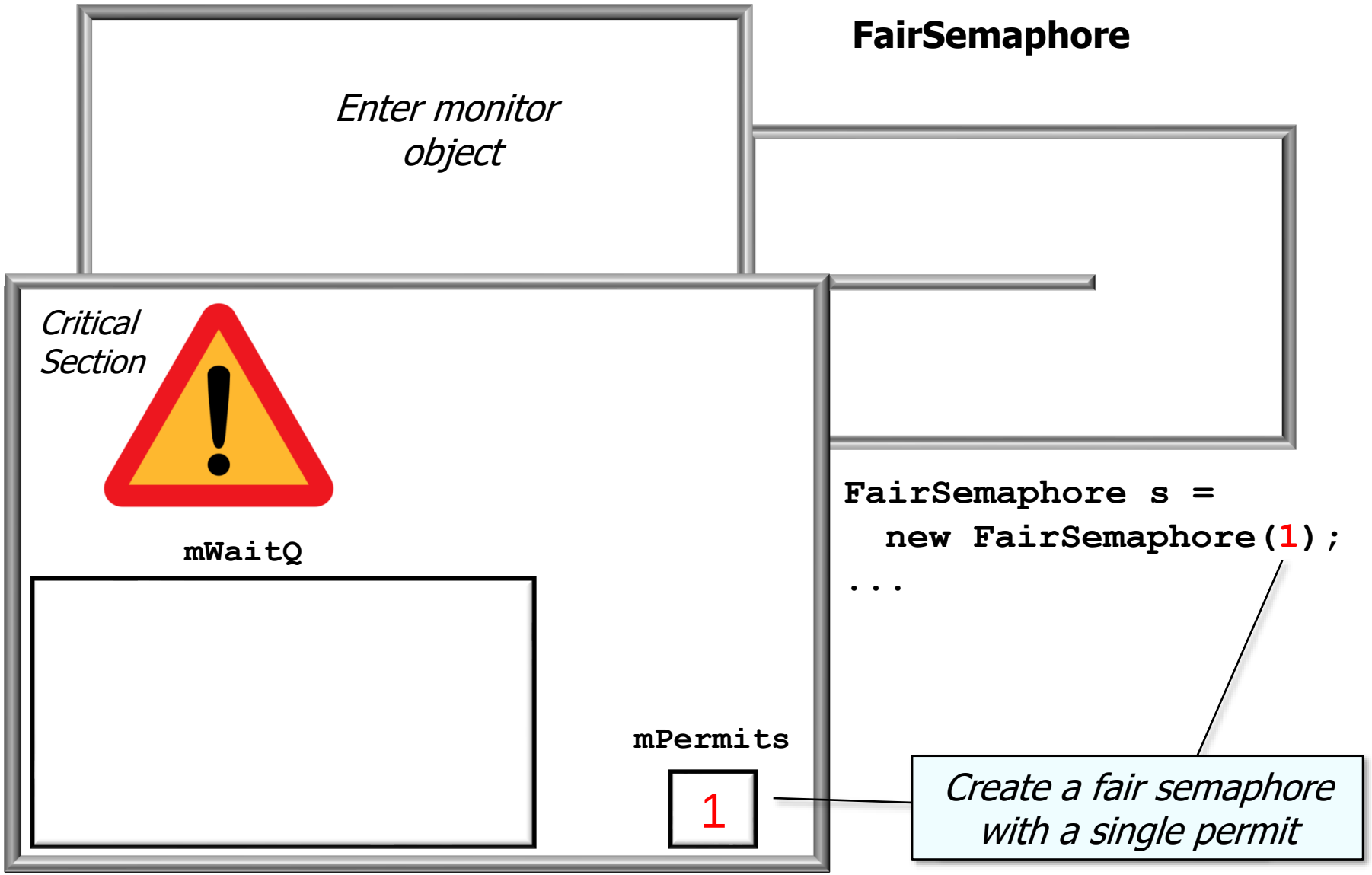
[1] B. Kantor, P. Lapsley, Network News Transfer Protocol, Internic RFC 977, 1986.

1

See www.dre.vanderbilt.edu/~schmidt/PDF/specific-notification.pdf (especially Listing 3)

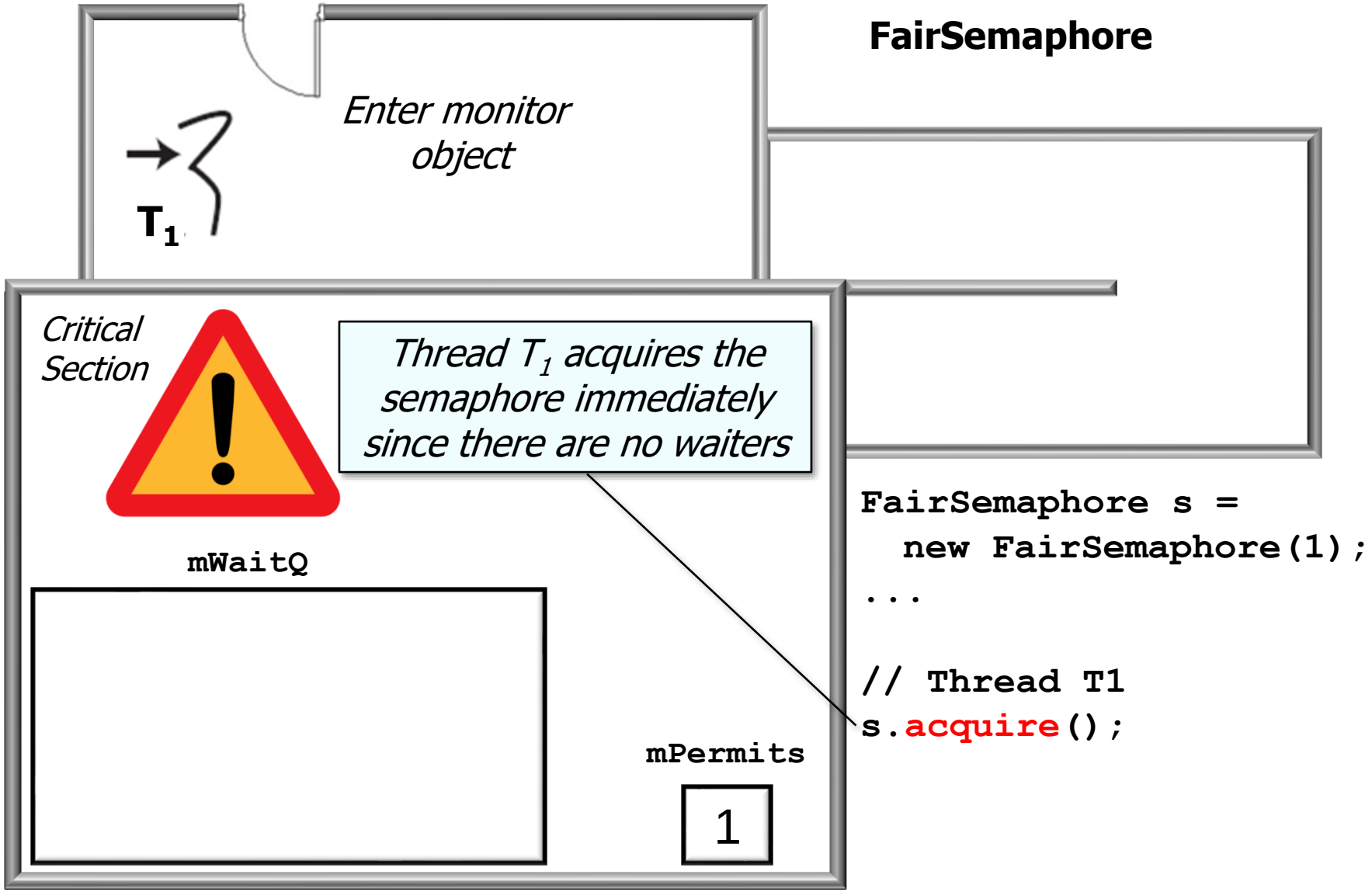
Visual Analysis of Fair Semaphore Acquire (T_1 & T_2)

Visual Analysis of Fair Semaphore Acquire (T_1 & T_2)

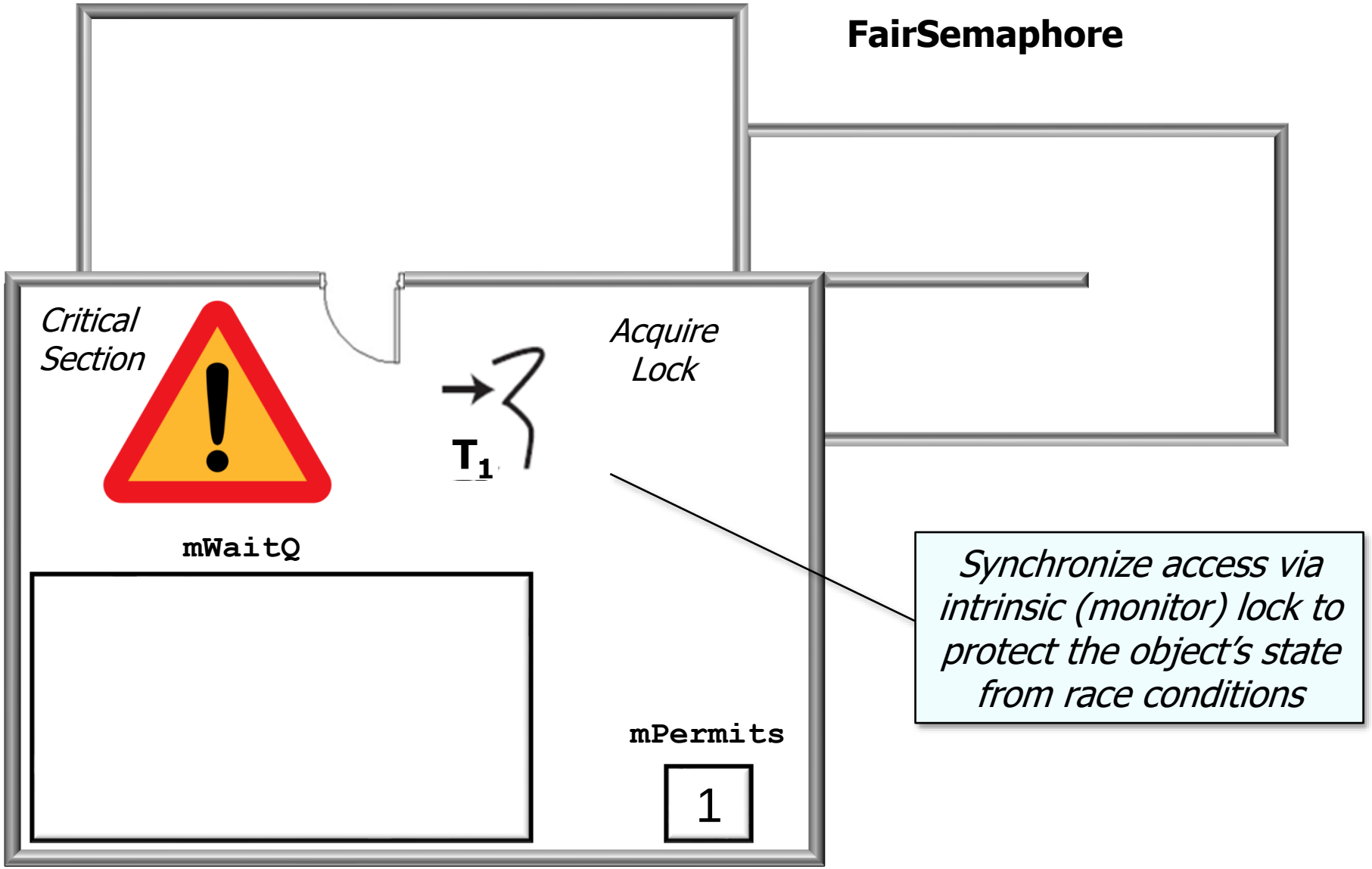


The wait queue is initially empty

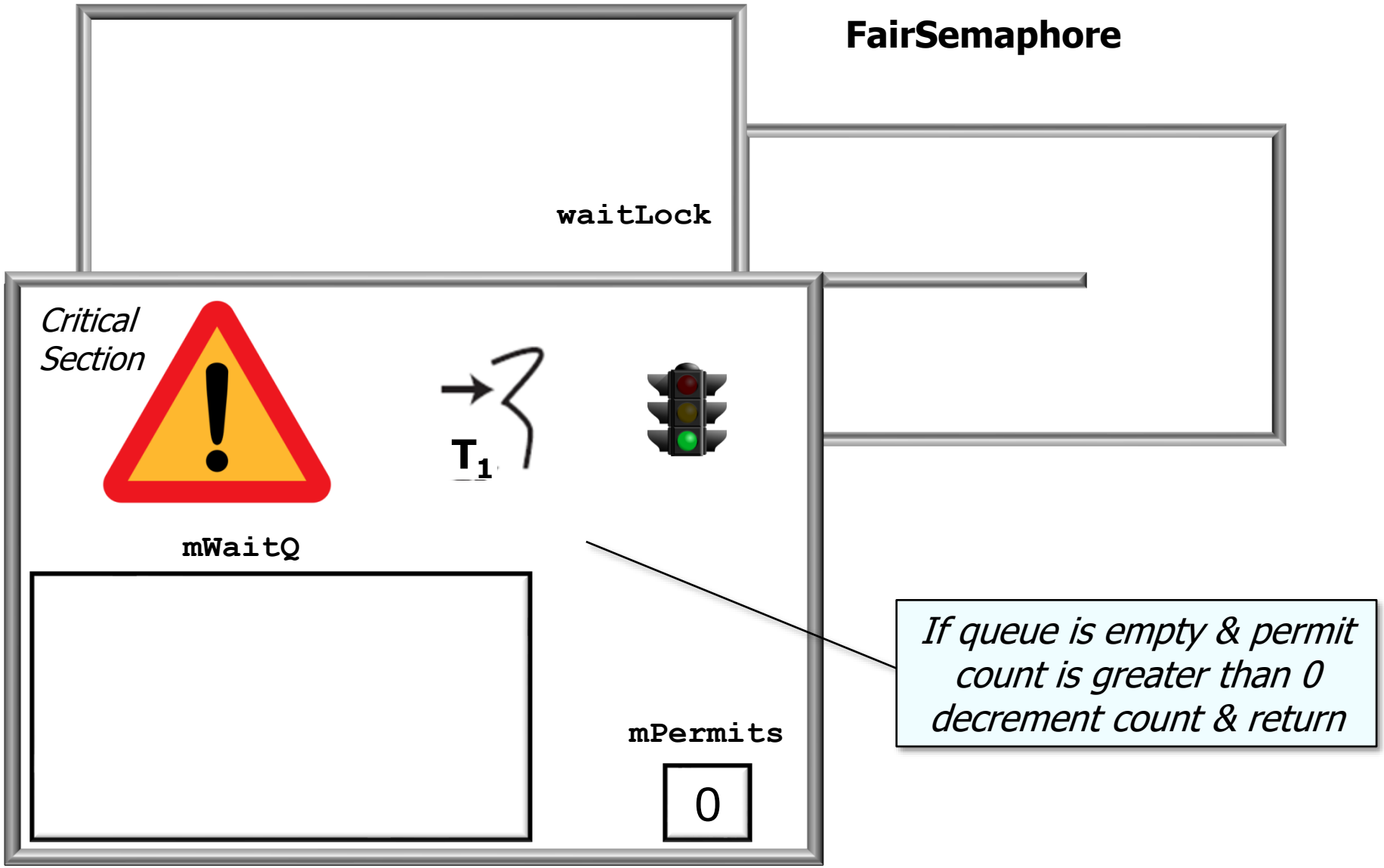
Visual Analysis of Fair Semaphore Acquire (T_1 & T_2)



Visual Analysis of Fair Semaphore Acquire (T_1 & T_2)

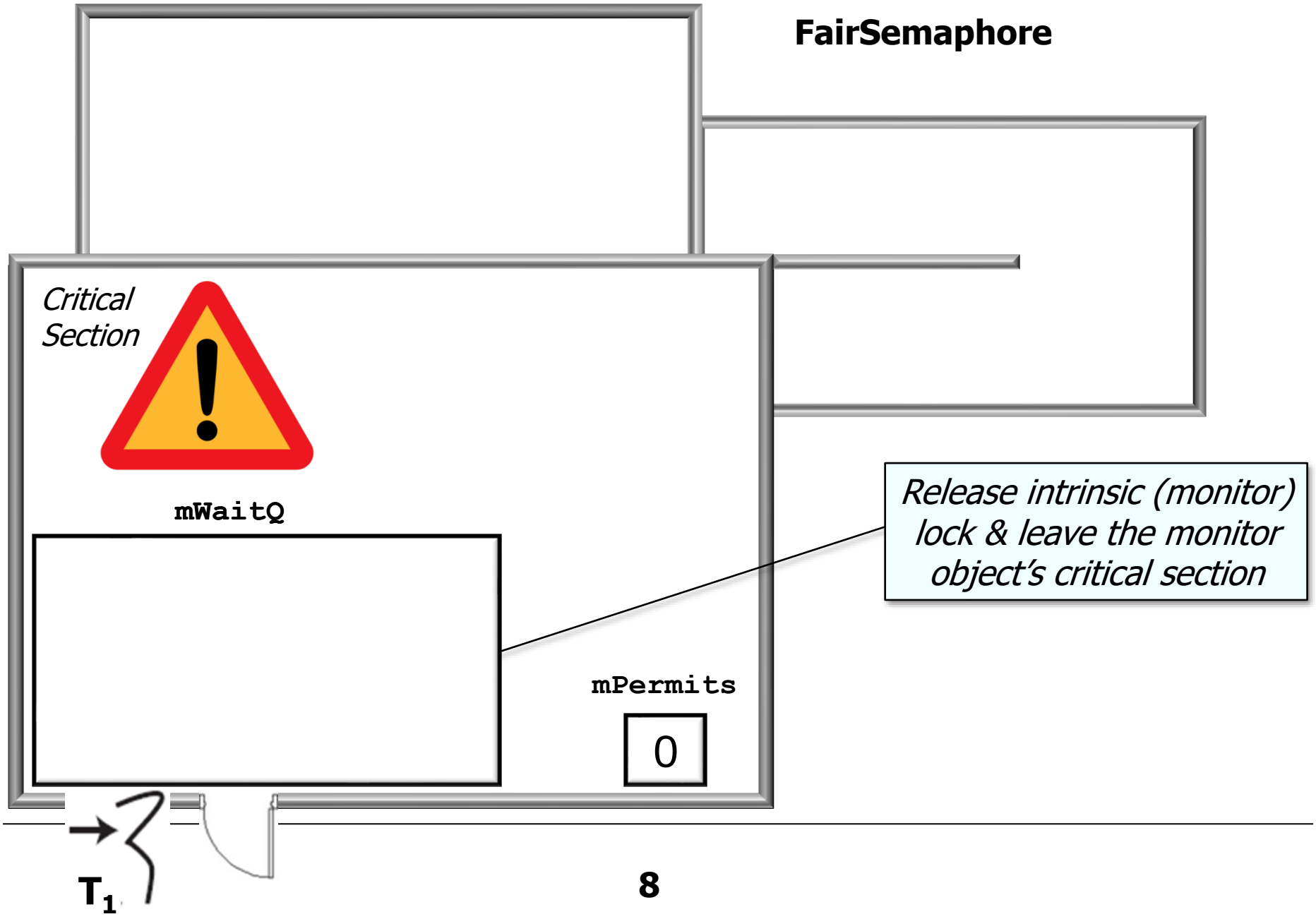


Visual Analysis of Fair Semaphore Acquire (T_1 & T_2)

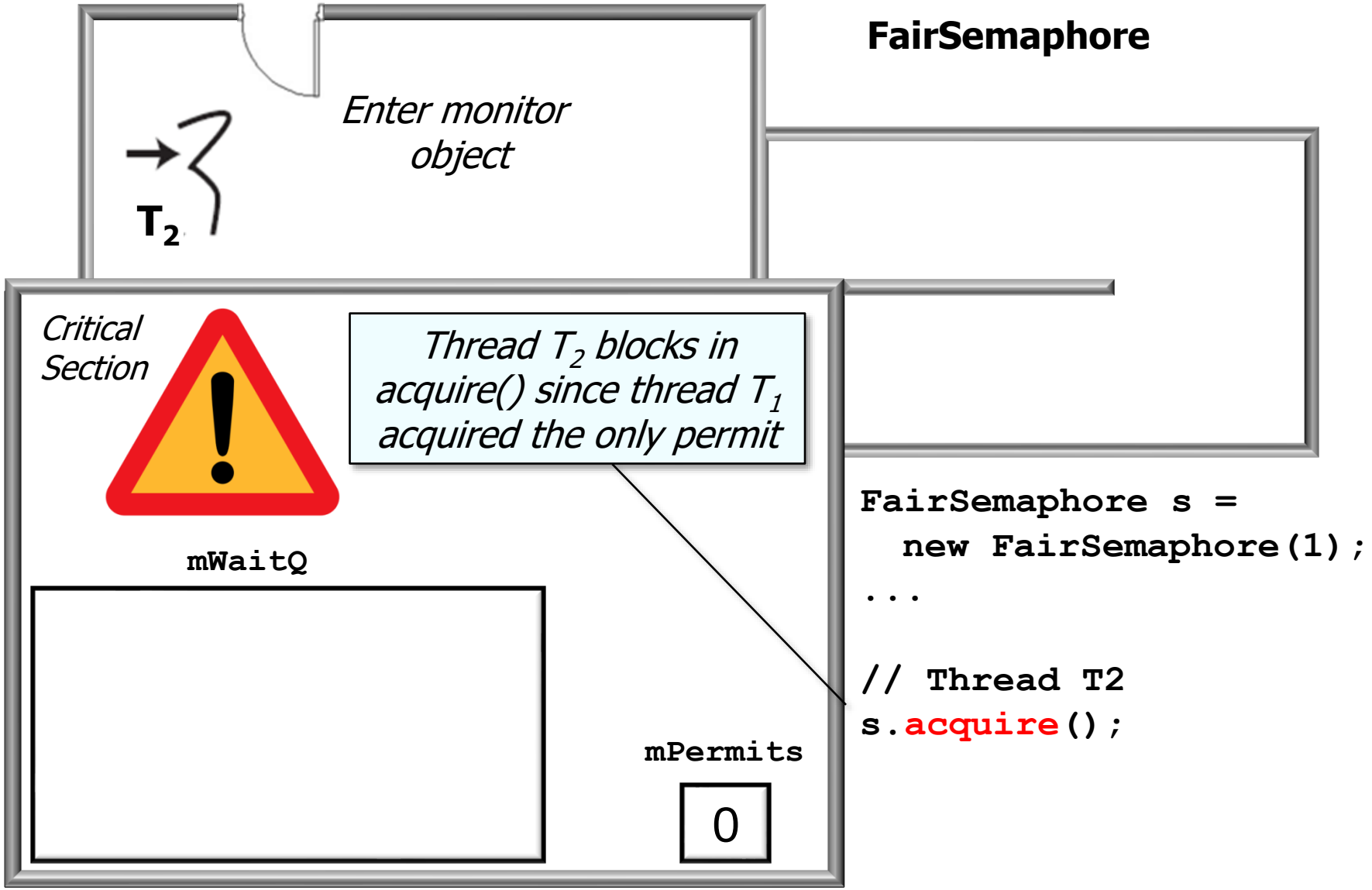


This is the "fast path"

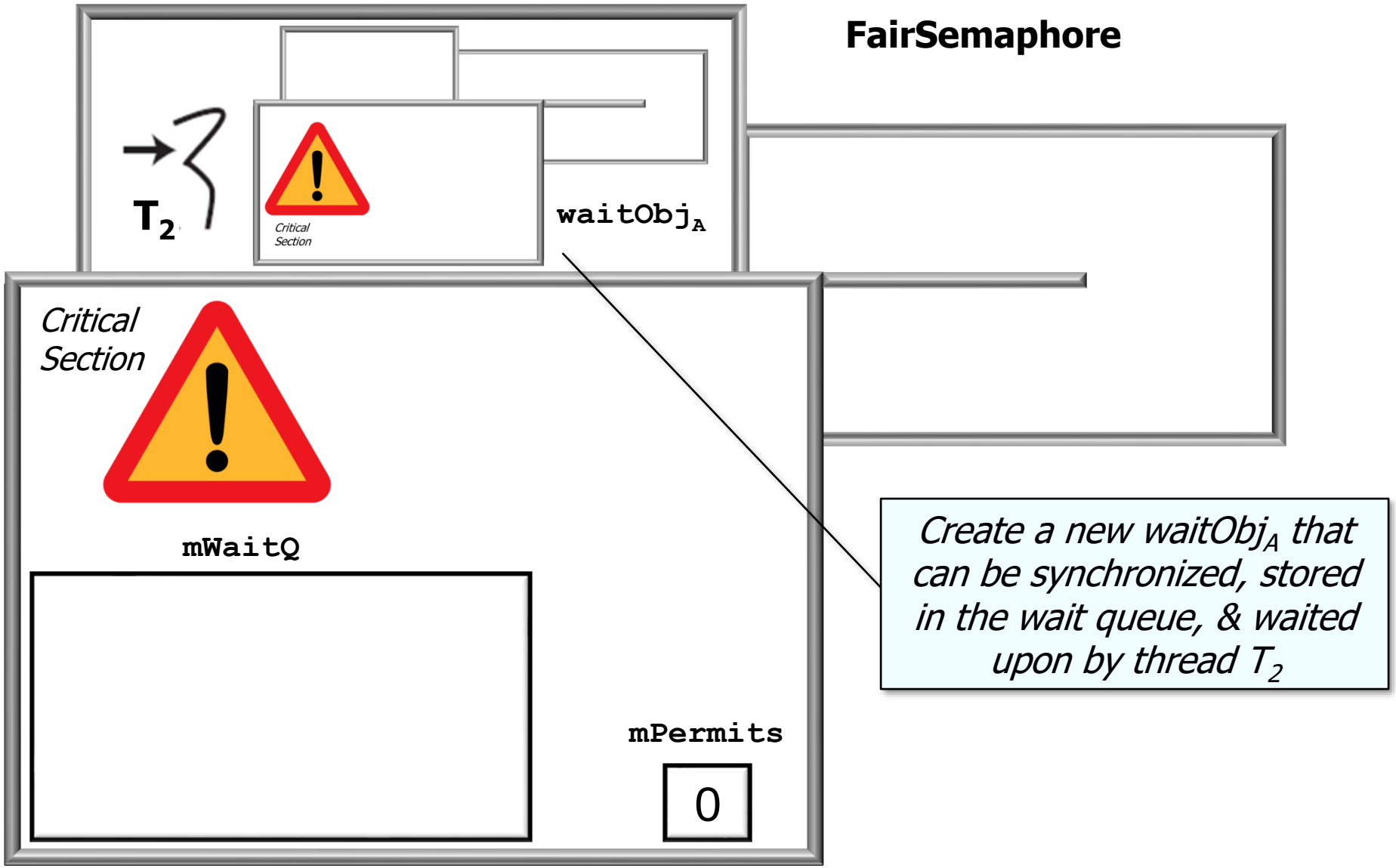
Visual Analysis of Fair Semaphore Acquire (T_1 & T_2)



Visual Analysis of Fair Semaphore Acquire (T_1 & T_2)

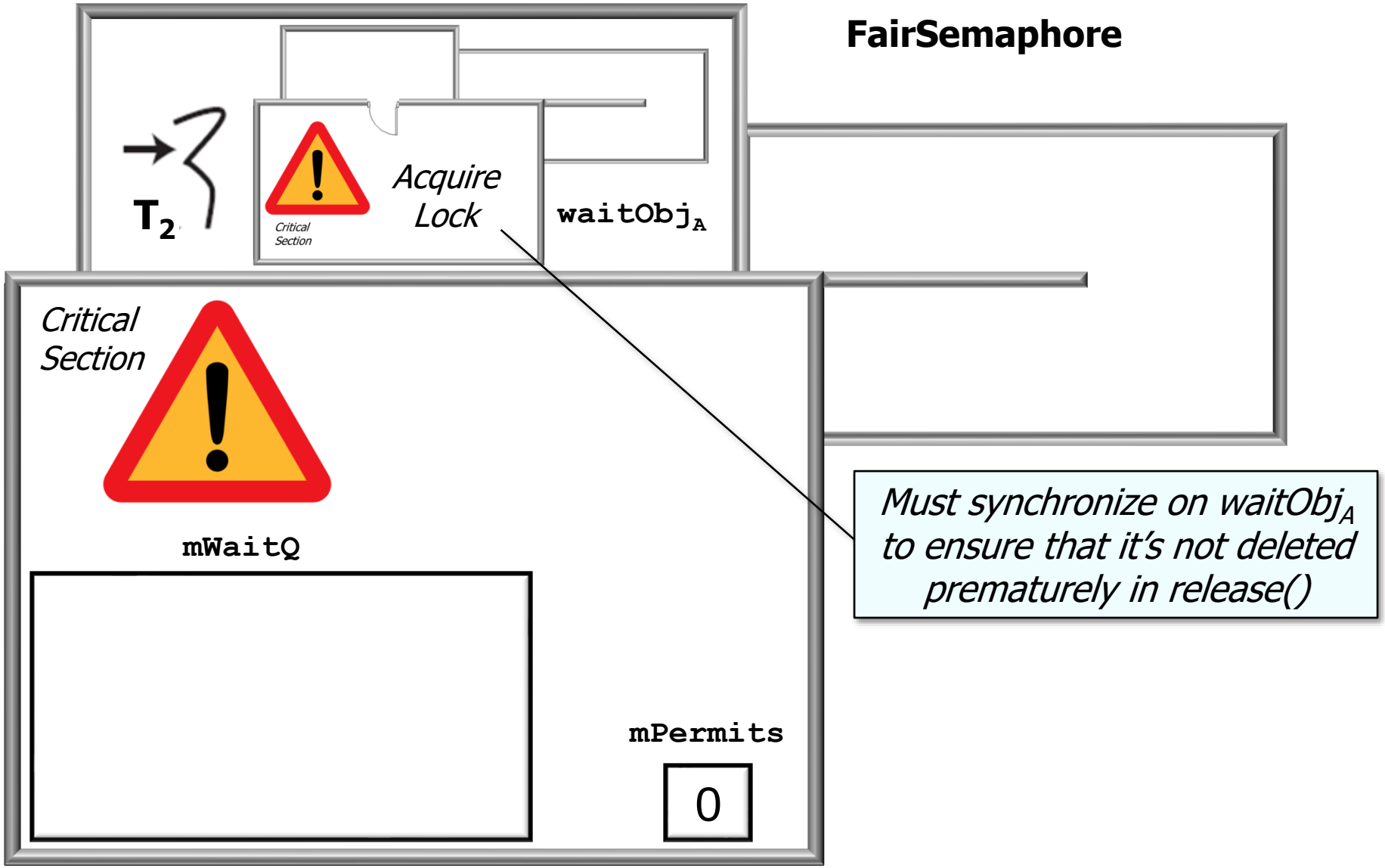


Visual Analysis of Fair Semaphore Acquire (T_1 & T_2)

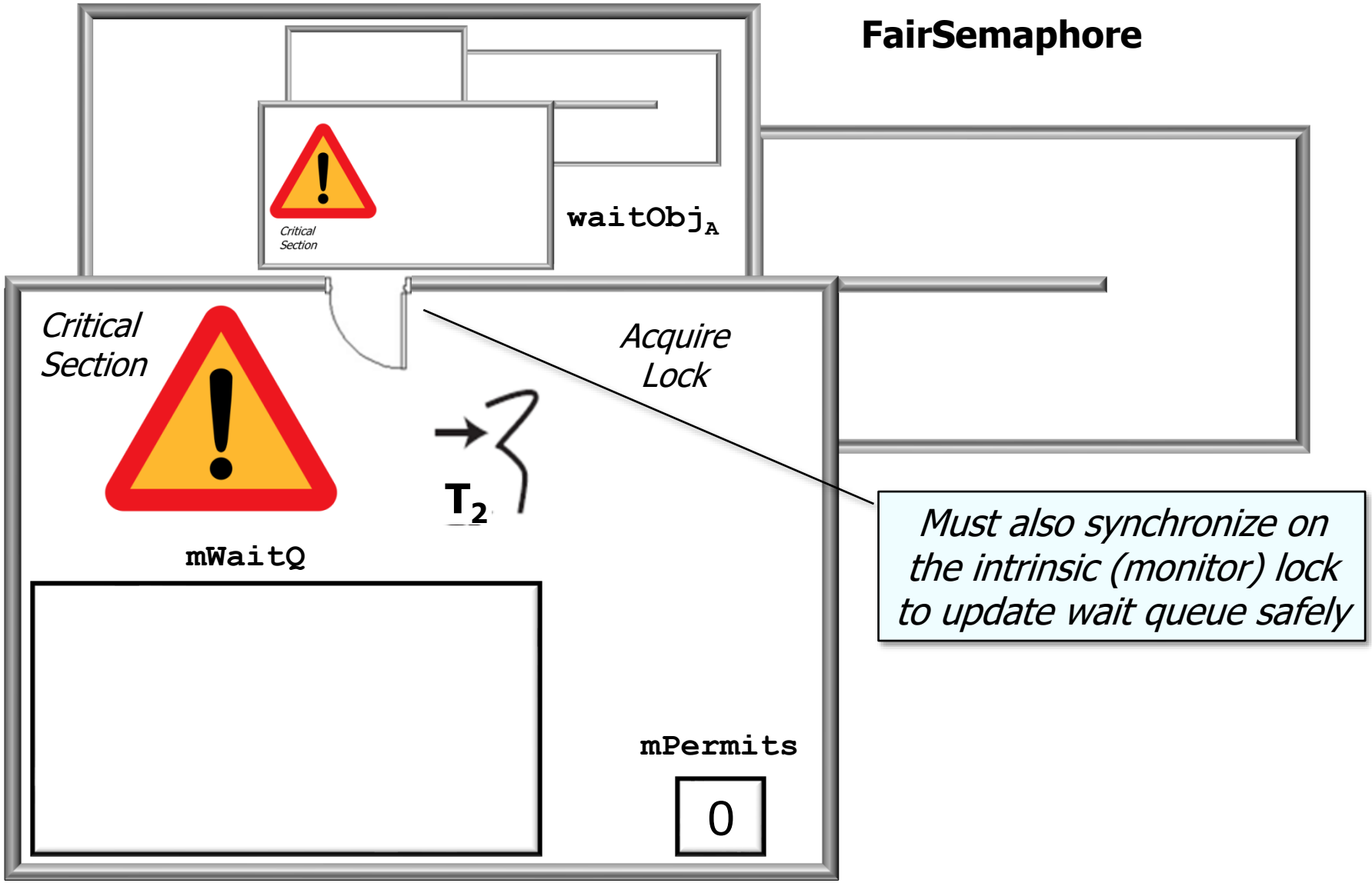


This waiting happens outside of the FairSemaphore's critical section

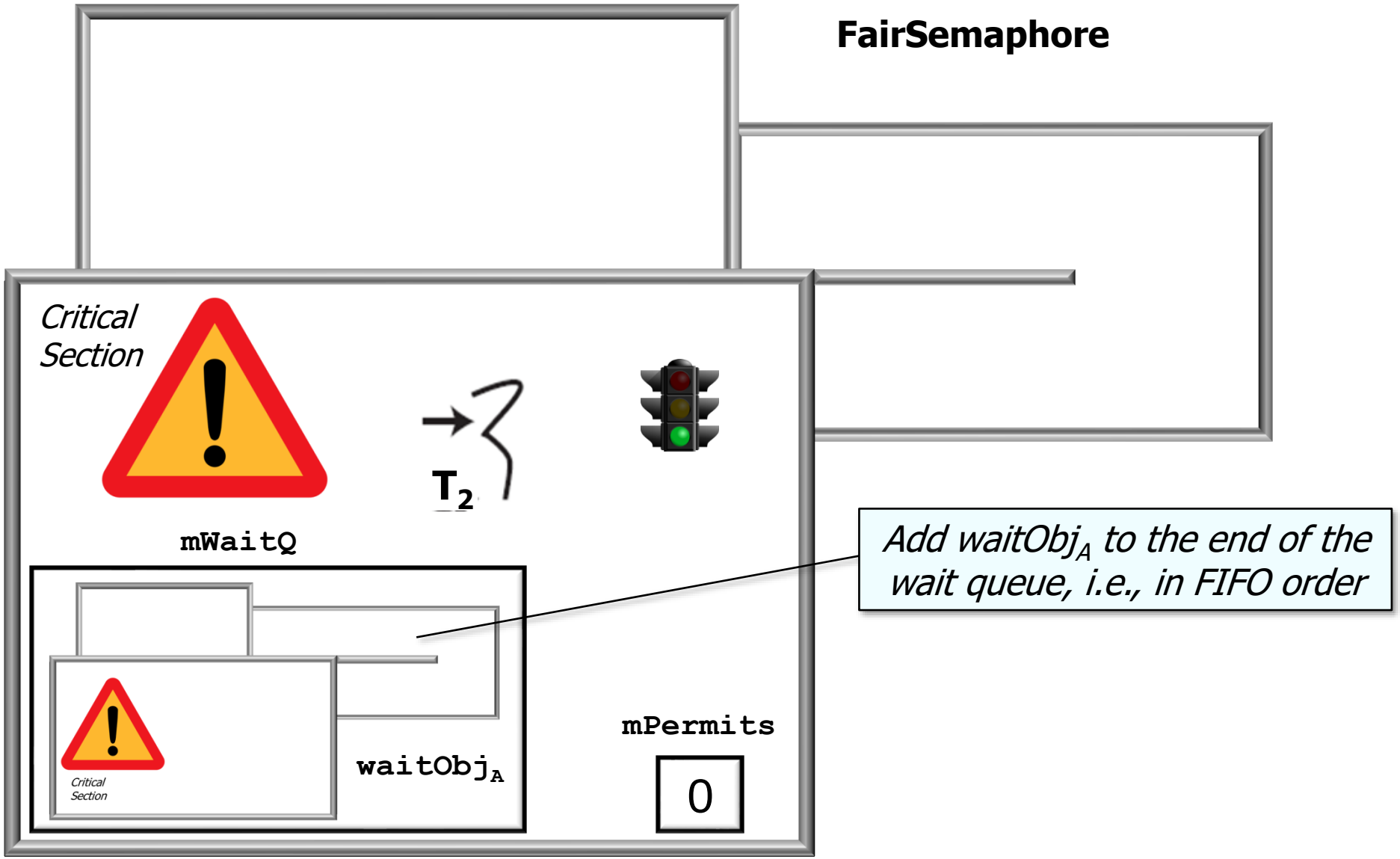
Visual Analysis of Fair Semaphore Acquire (T_1 & T_2)



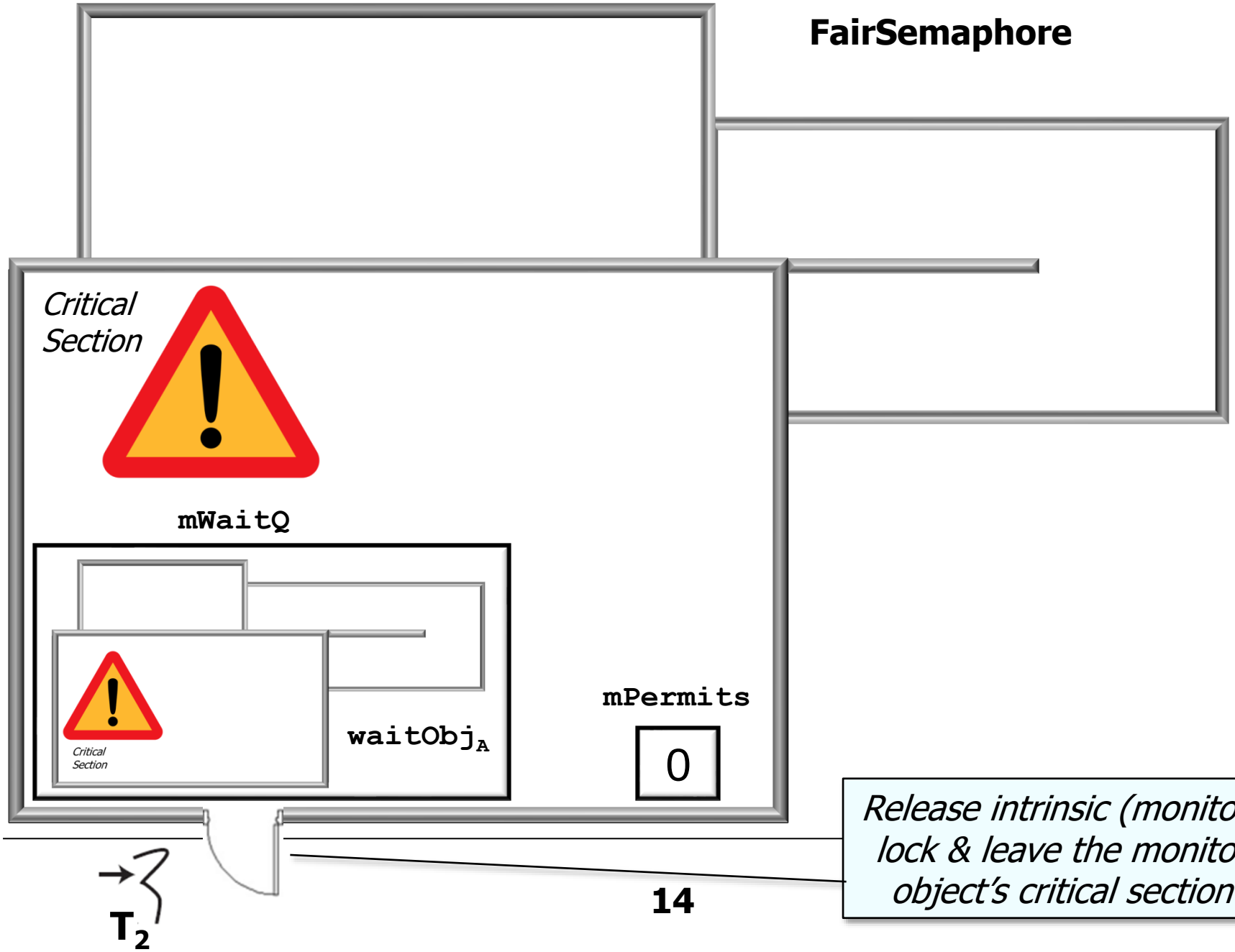
Visual Analysis of Fair Semaphore Acquire (T_1 & T_2)



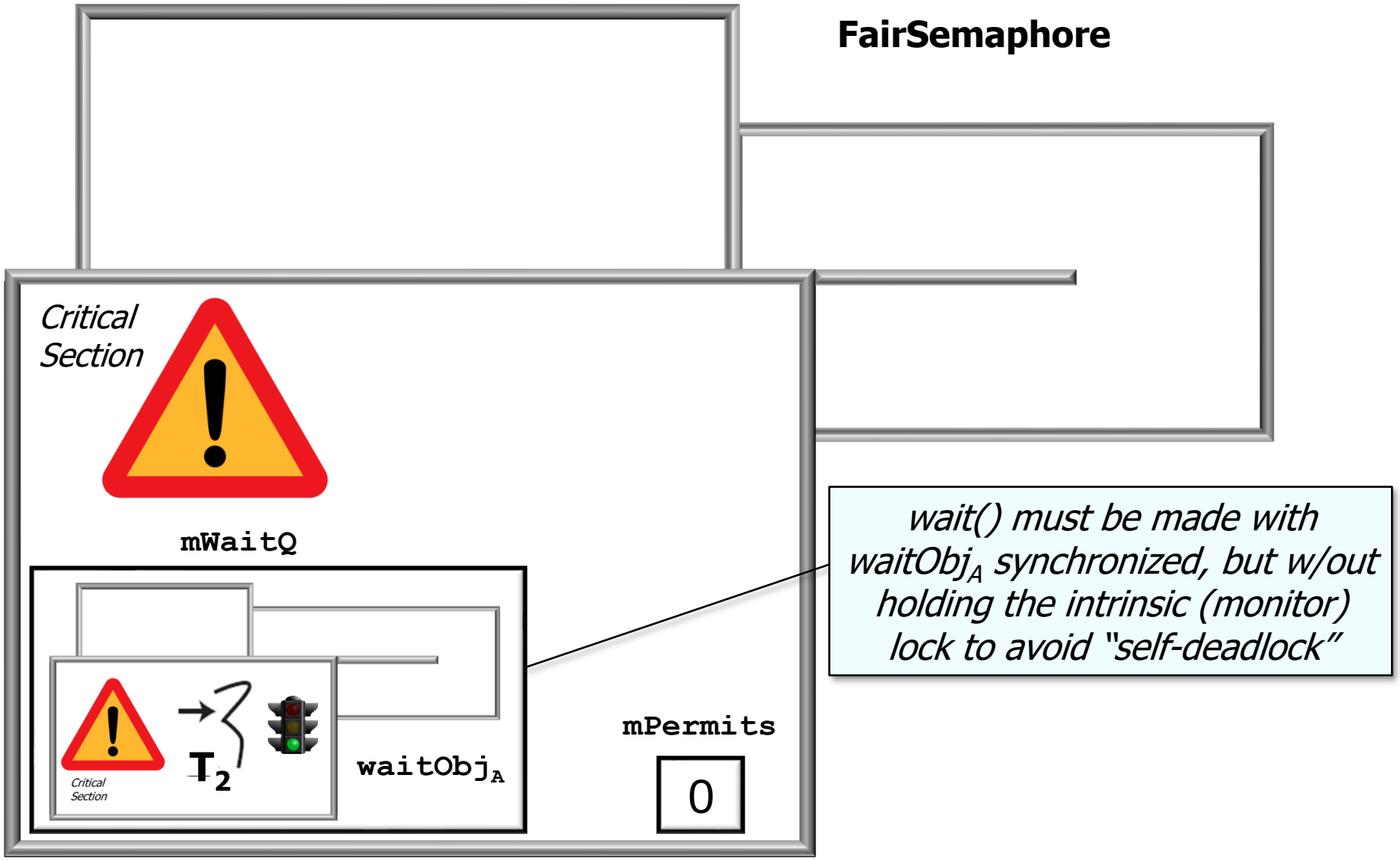
Visual Analysis of Fair Semaphore Acquire (T_1 & T_2)



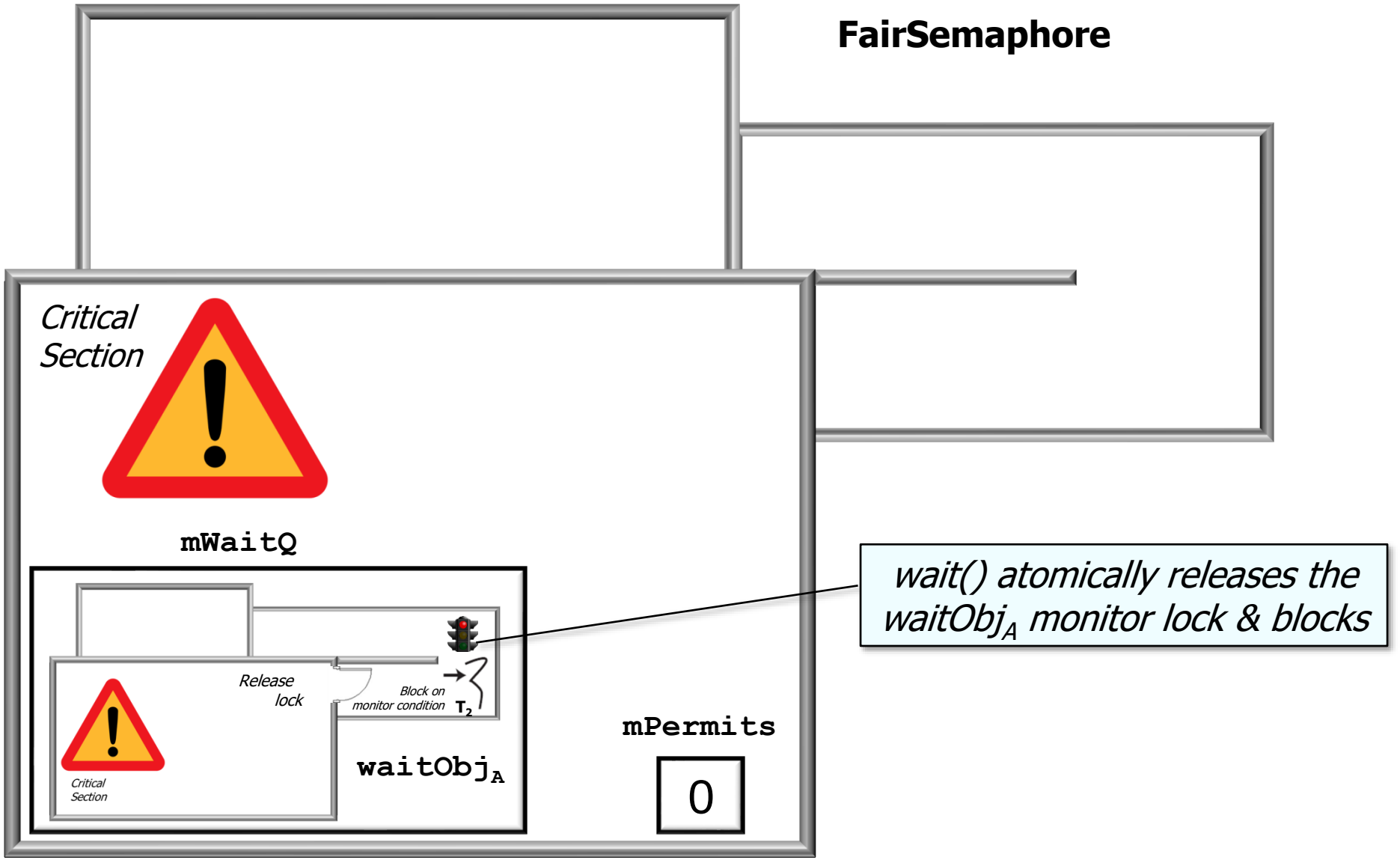
Visual Analysis of Fair Semaphore Acquire (T_1 & T_2)



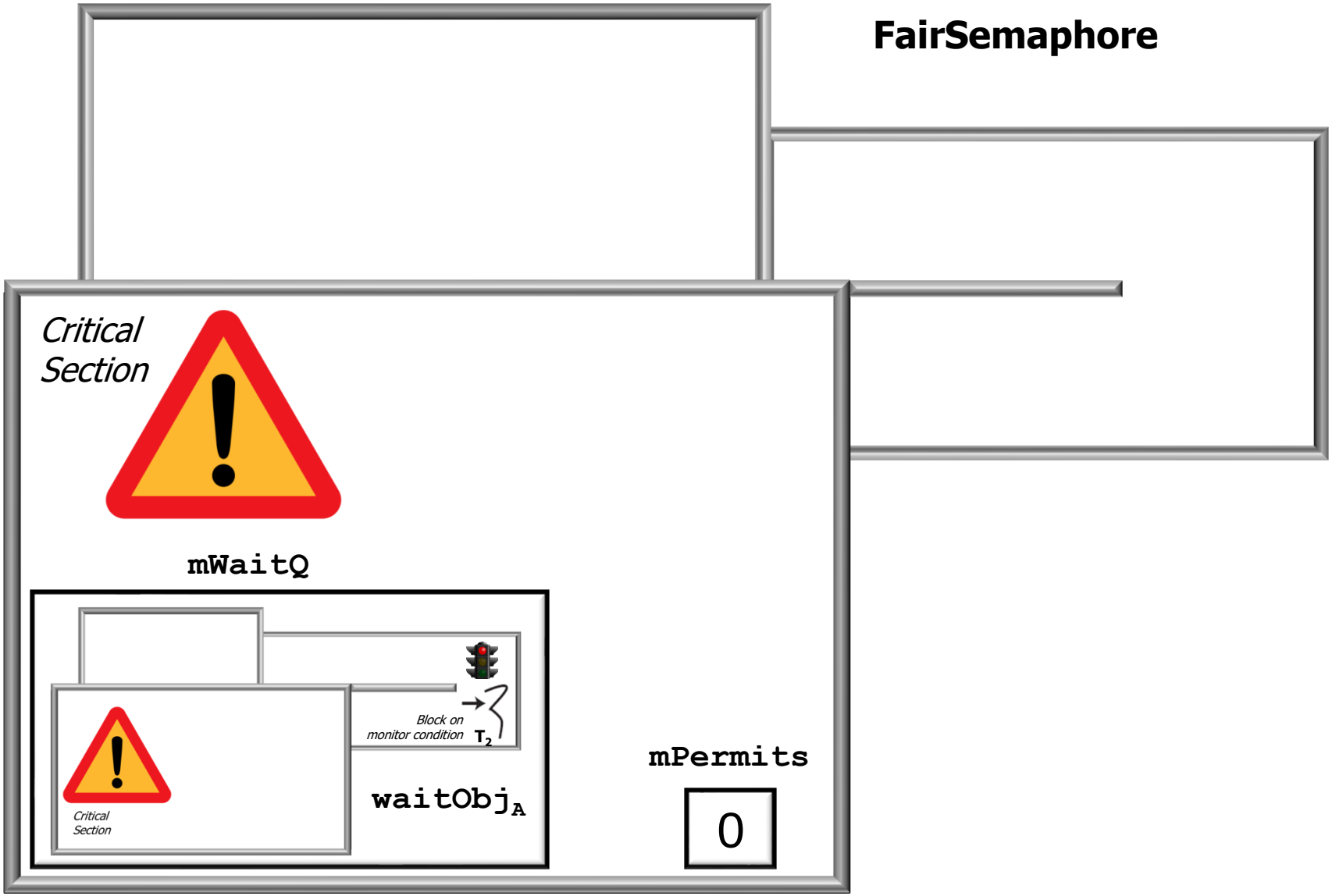
Visual Analysis of Fair Semaphore Acquire (T_1 & T_2)



Visual Analysis of Fair Semaphore Acquire (T_1 & T_2)

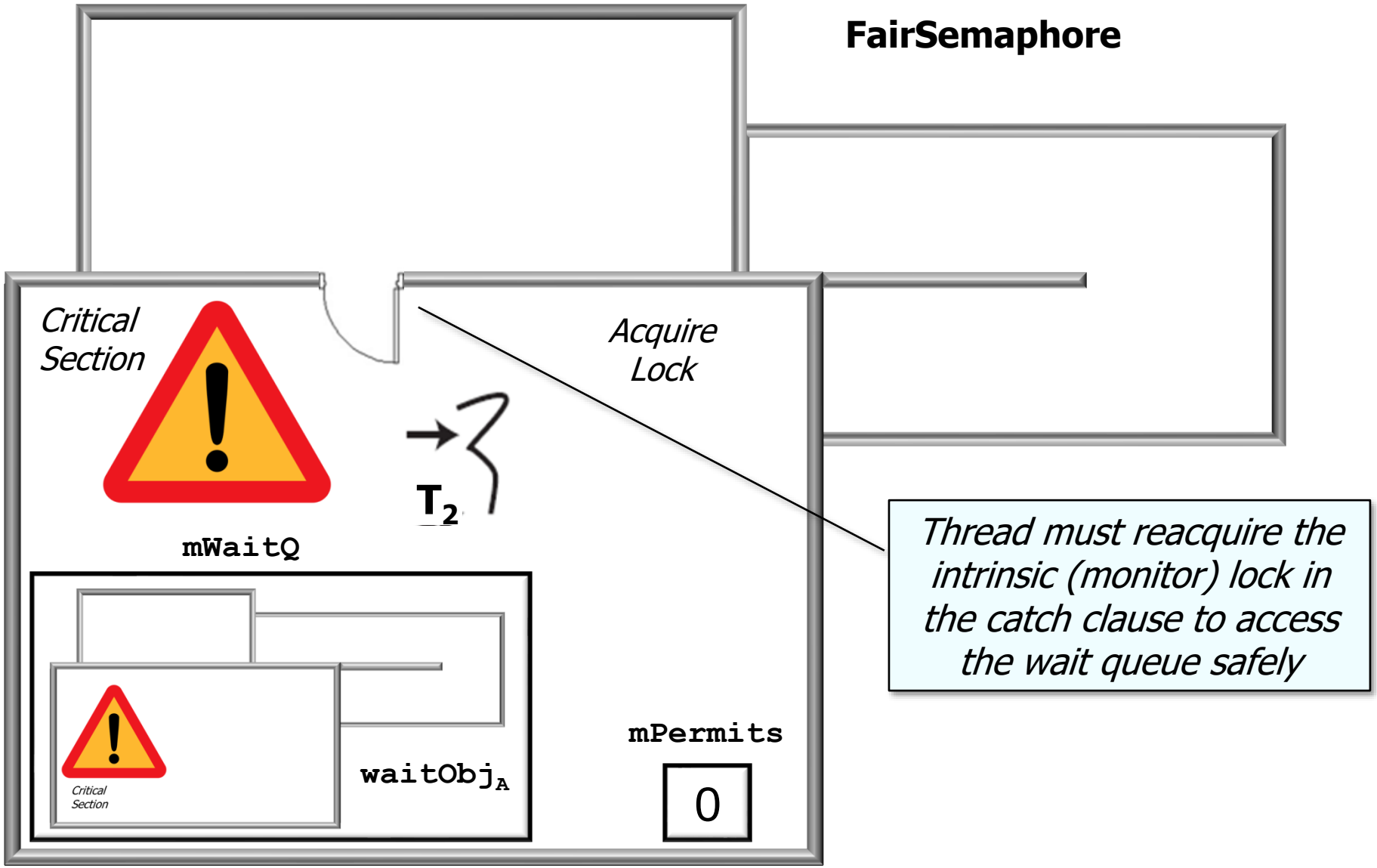


Visual Analysis of Fair Semaphore Acquire (T_1 & T_2)

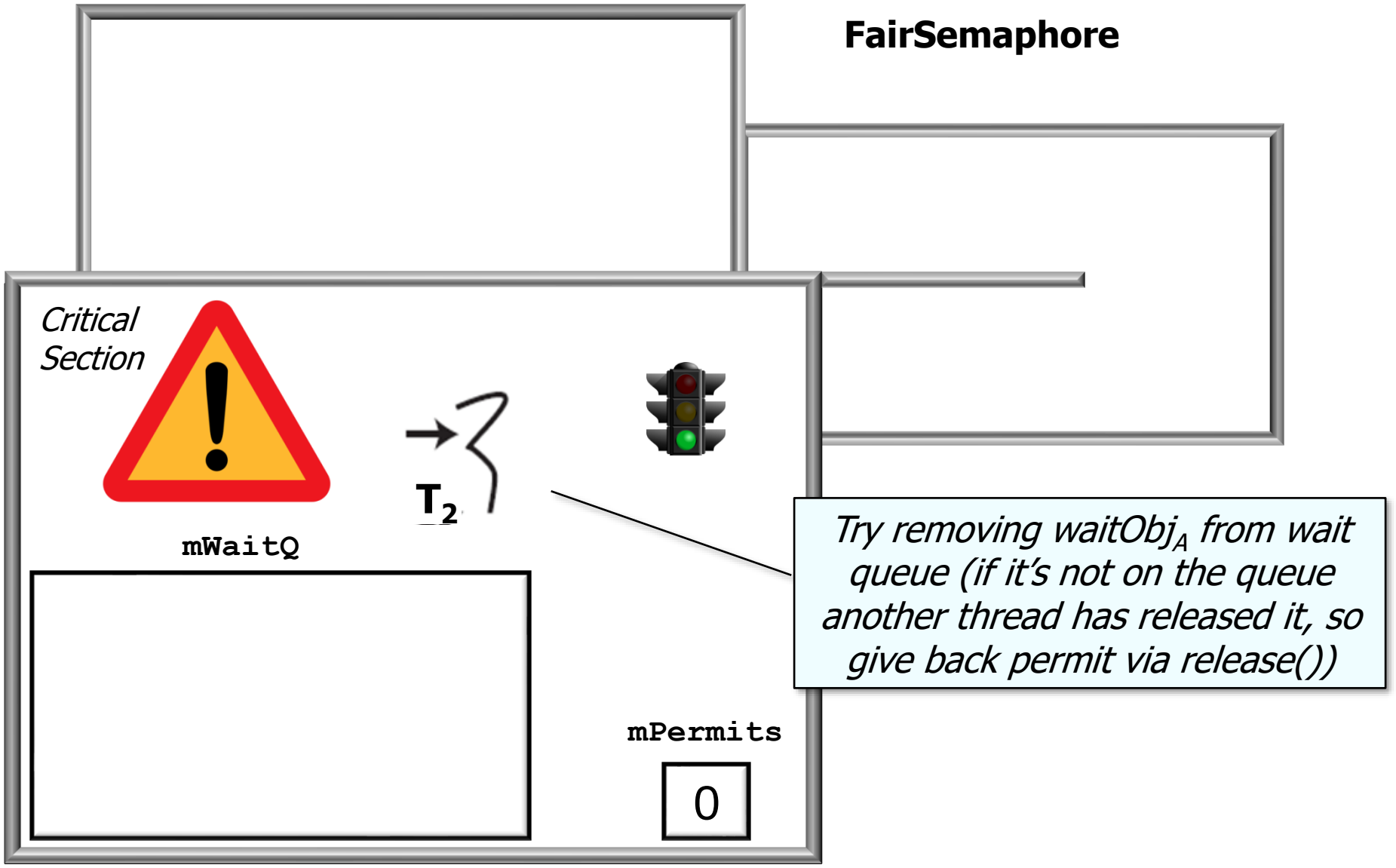


Here's what happens when a Java InterruptedException (IE) is thrown in the acquire() method during a blocking call to wait() on a waitObj

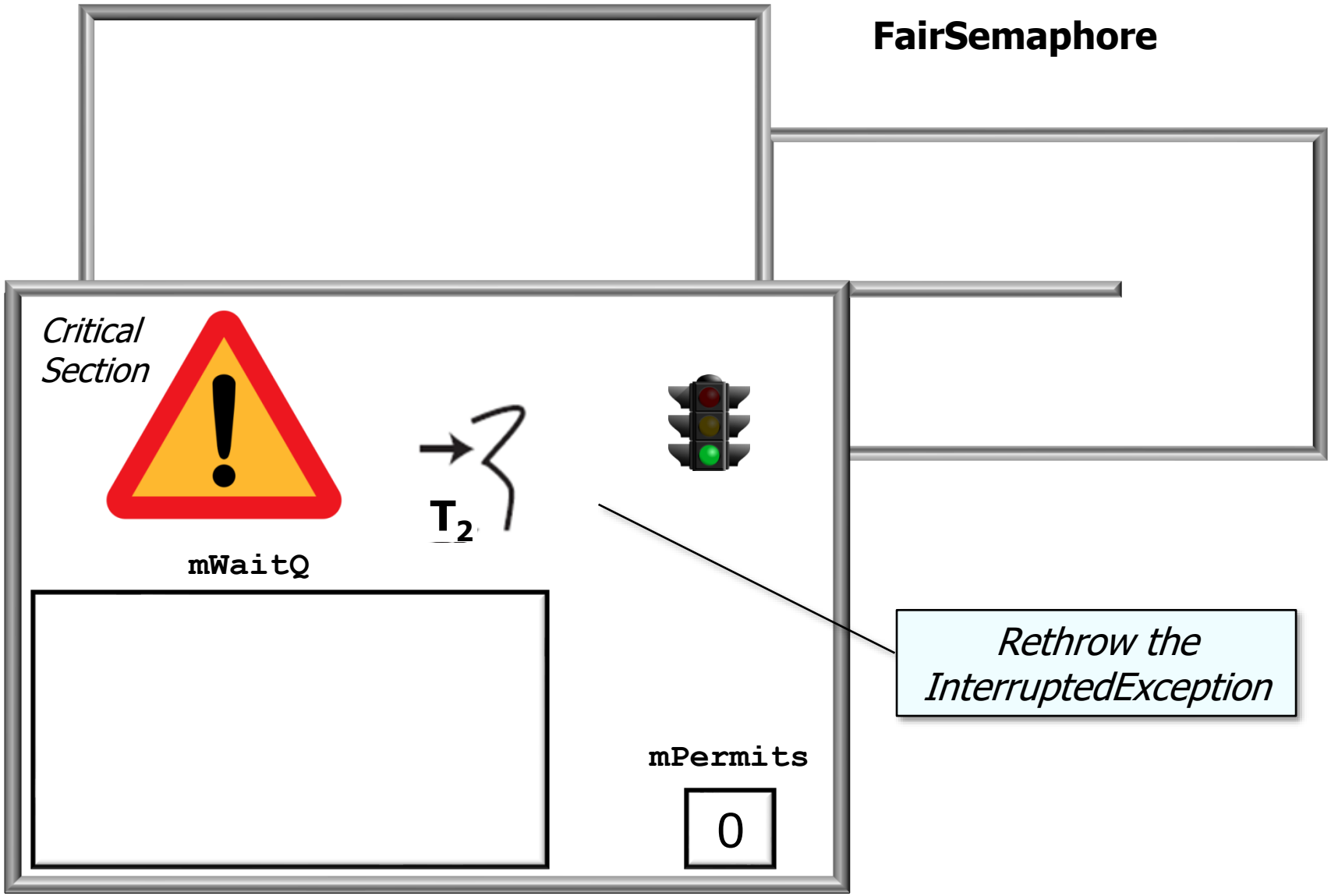
Visual Analysis of Fair Semaphore Acquire (T_1 & T_2)



Visual Analysis of Fair Semaphore Acquire (T_1 & T_2)

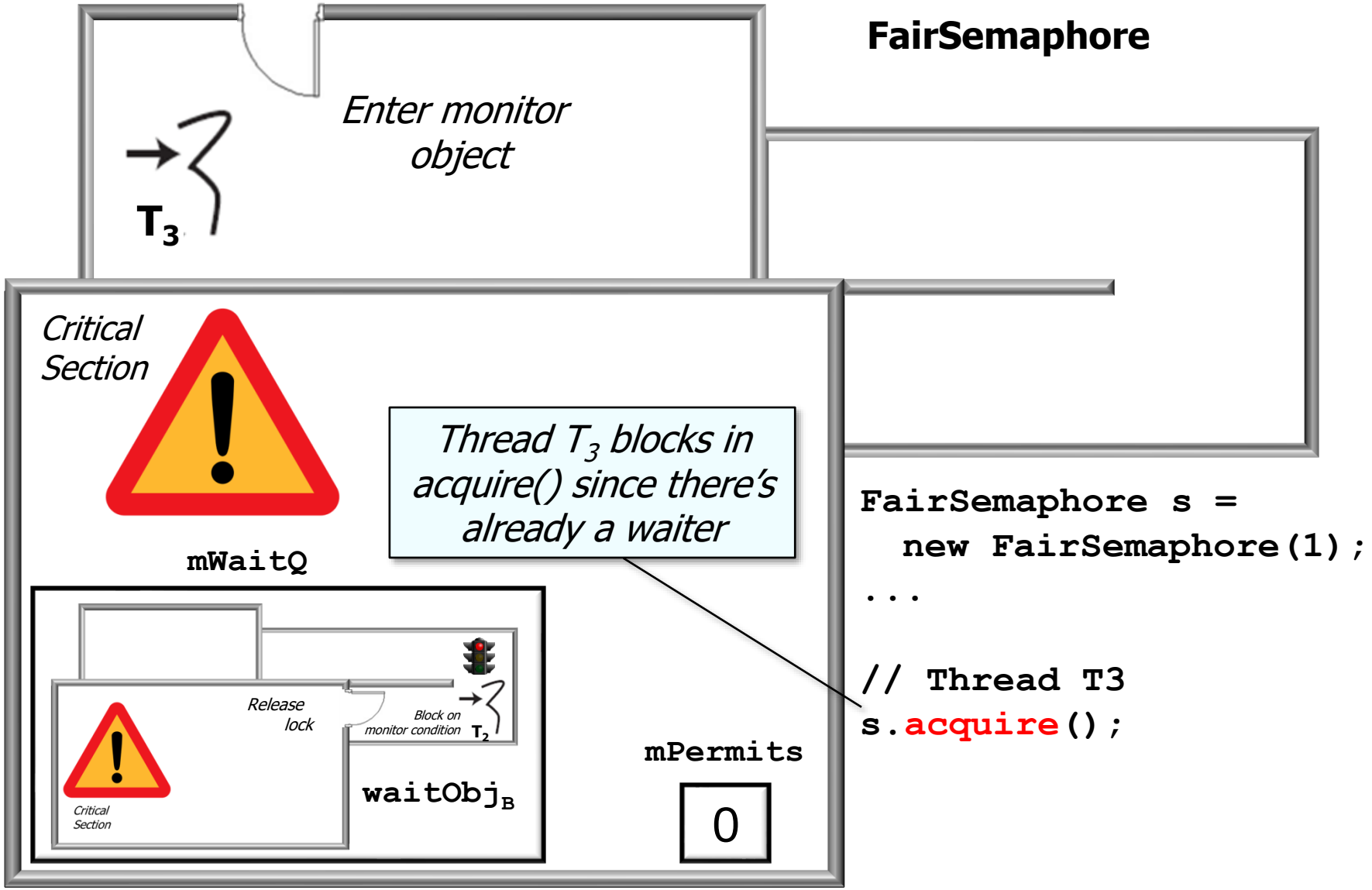


Visual Analysis of Fair Semaphore Acquire (T_1 & T_2)

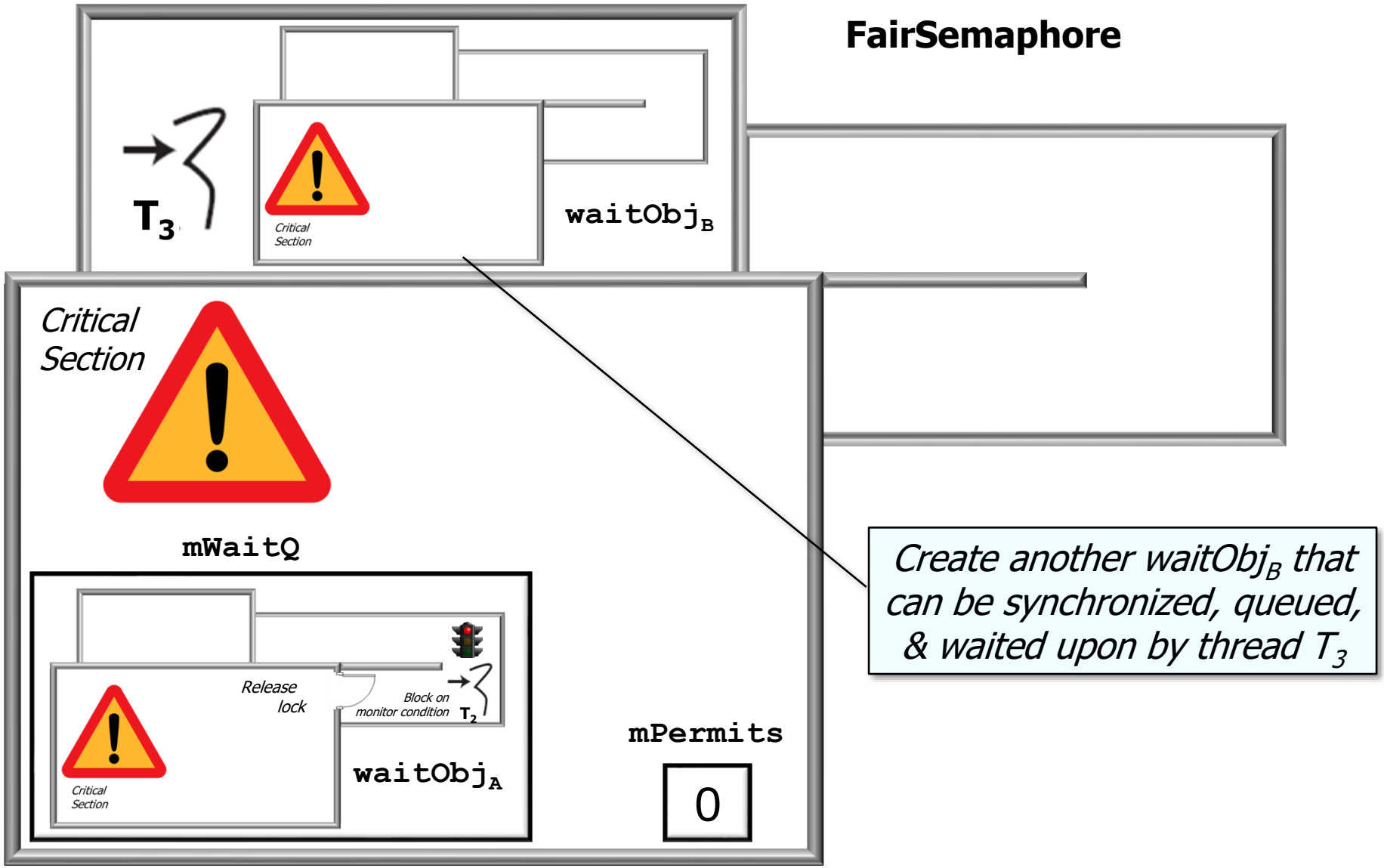


Visual Analysis of Fair Semaphore Acquire (T_3)

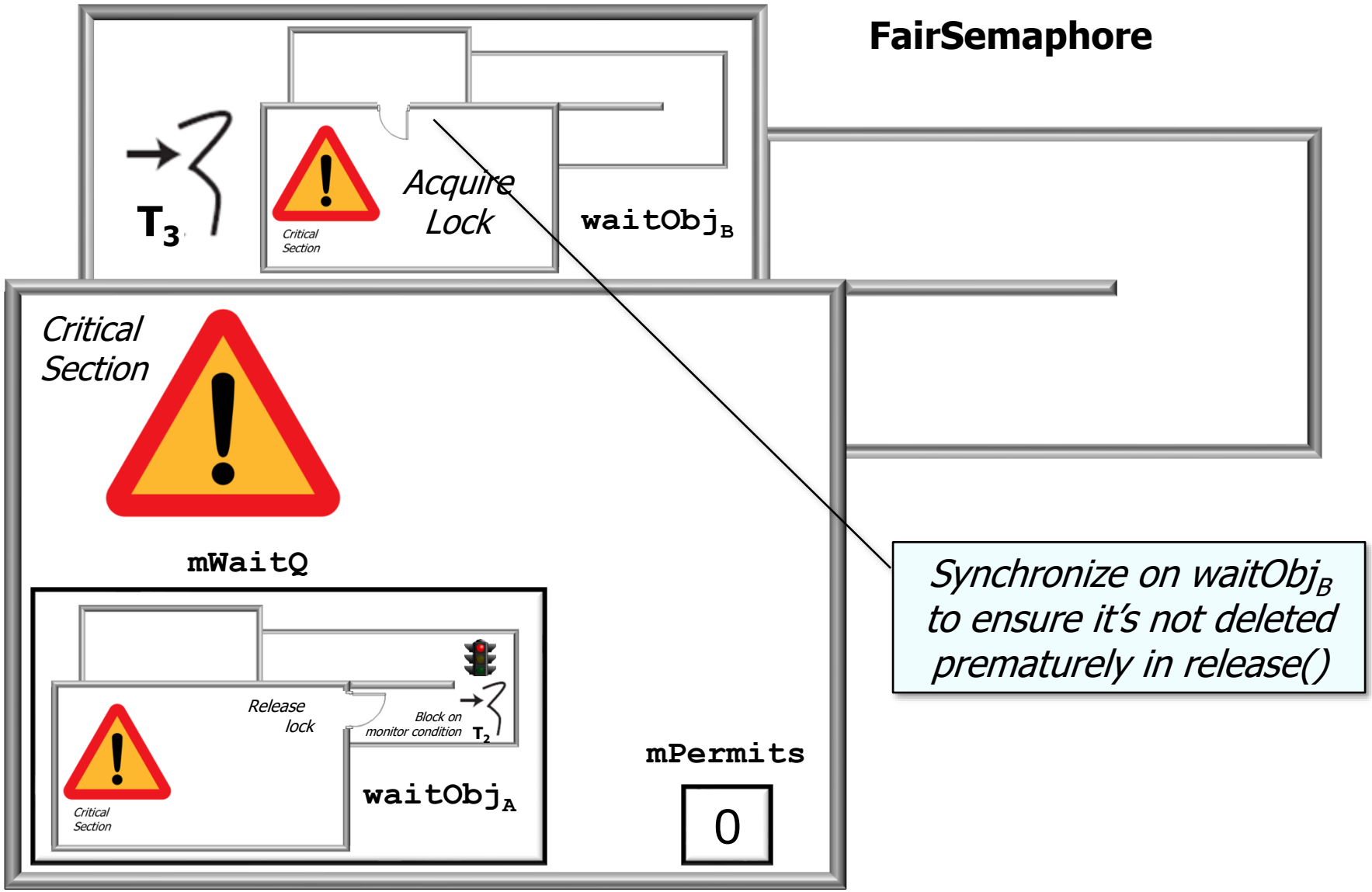
Visual Analysis of Fair Semaphore Acquire (T_3)



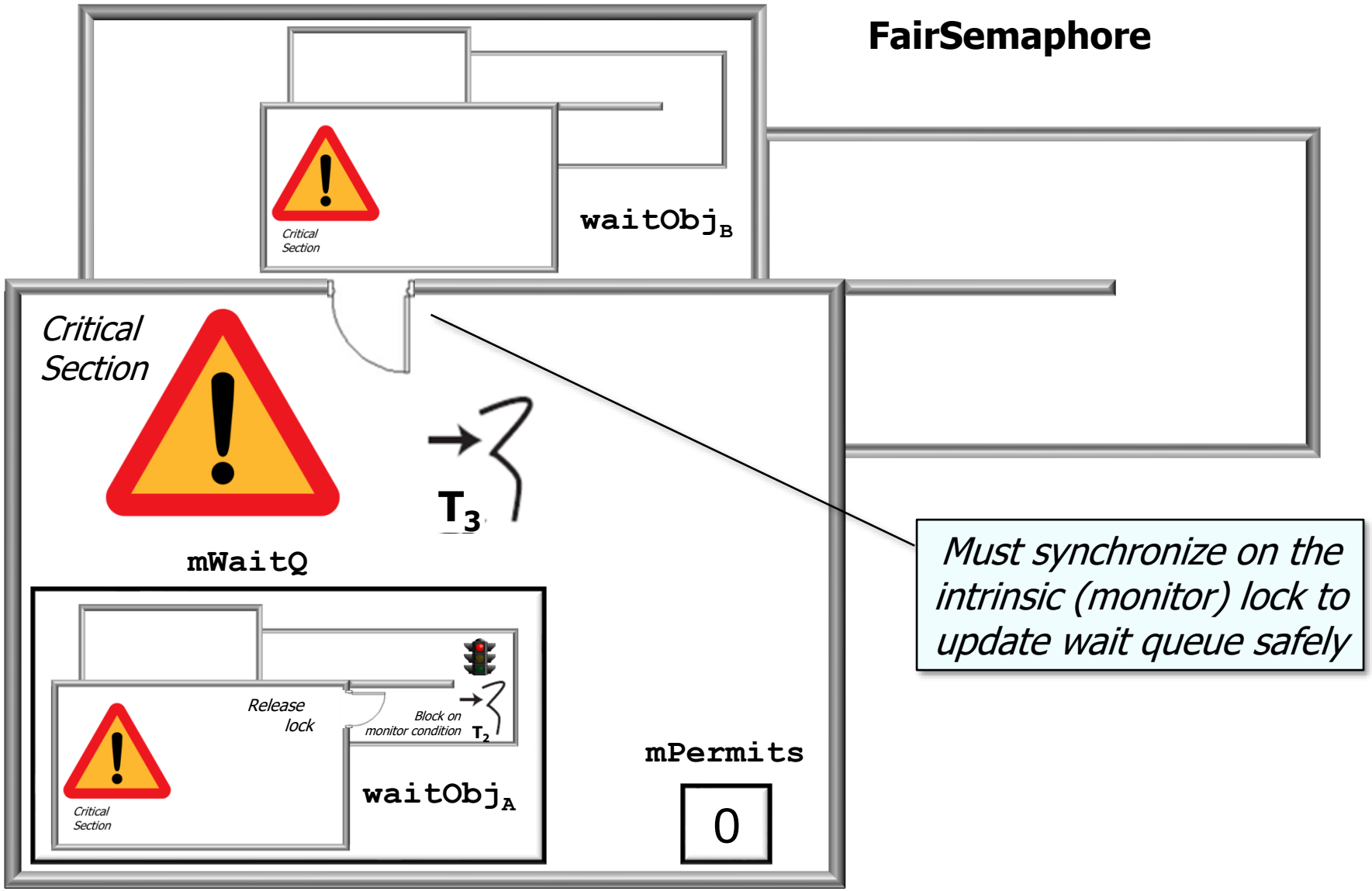
Visual Analysis of Fair Semaphore Acquire (T_3)



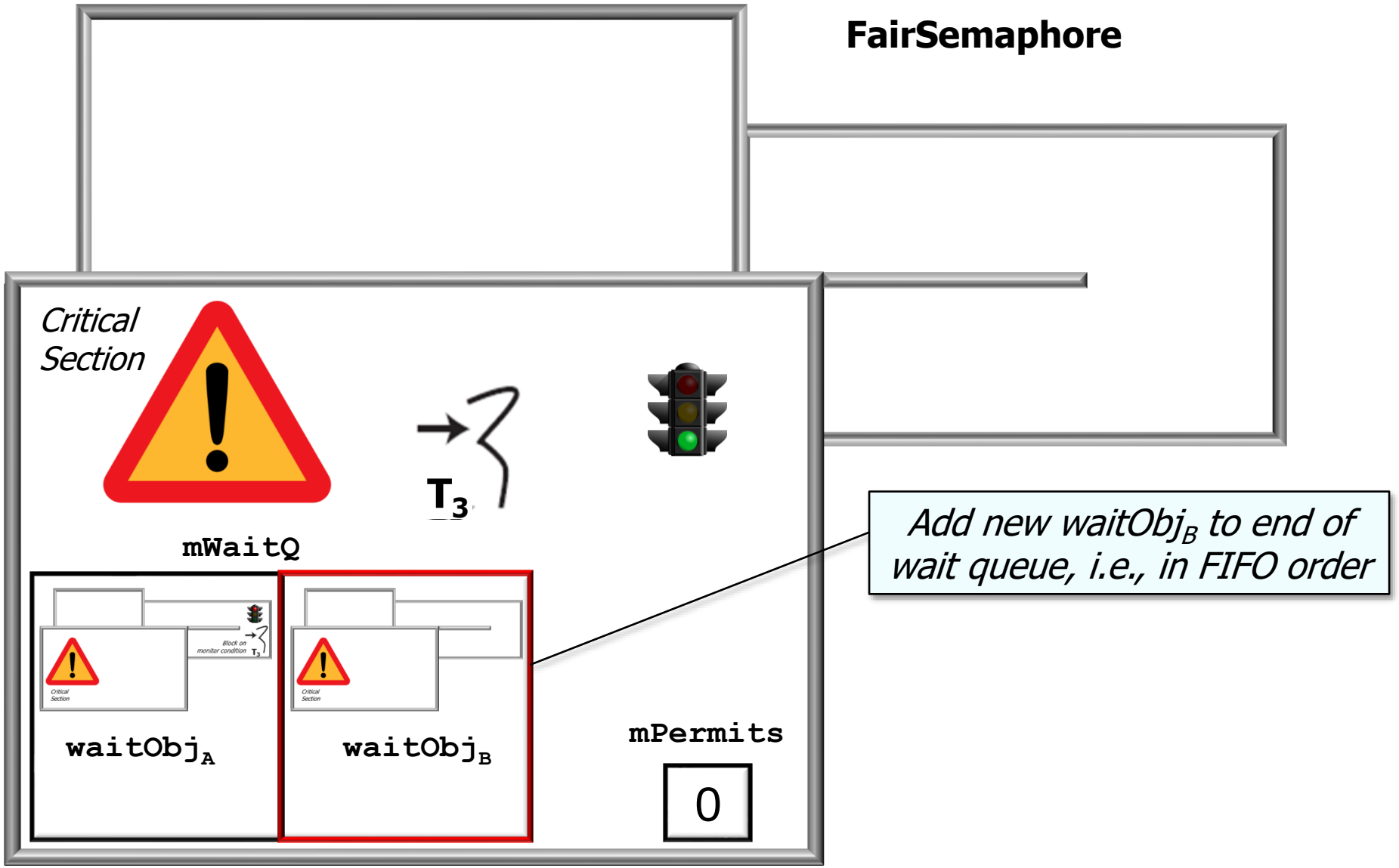
Visual Analysis of Fair Semaphore Acquire (T_3)



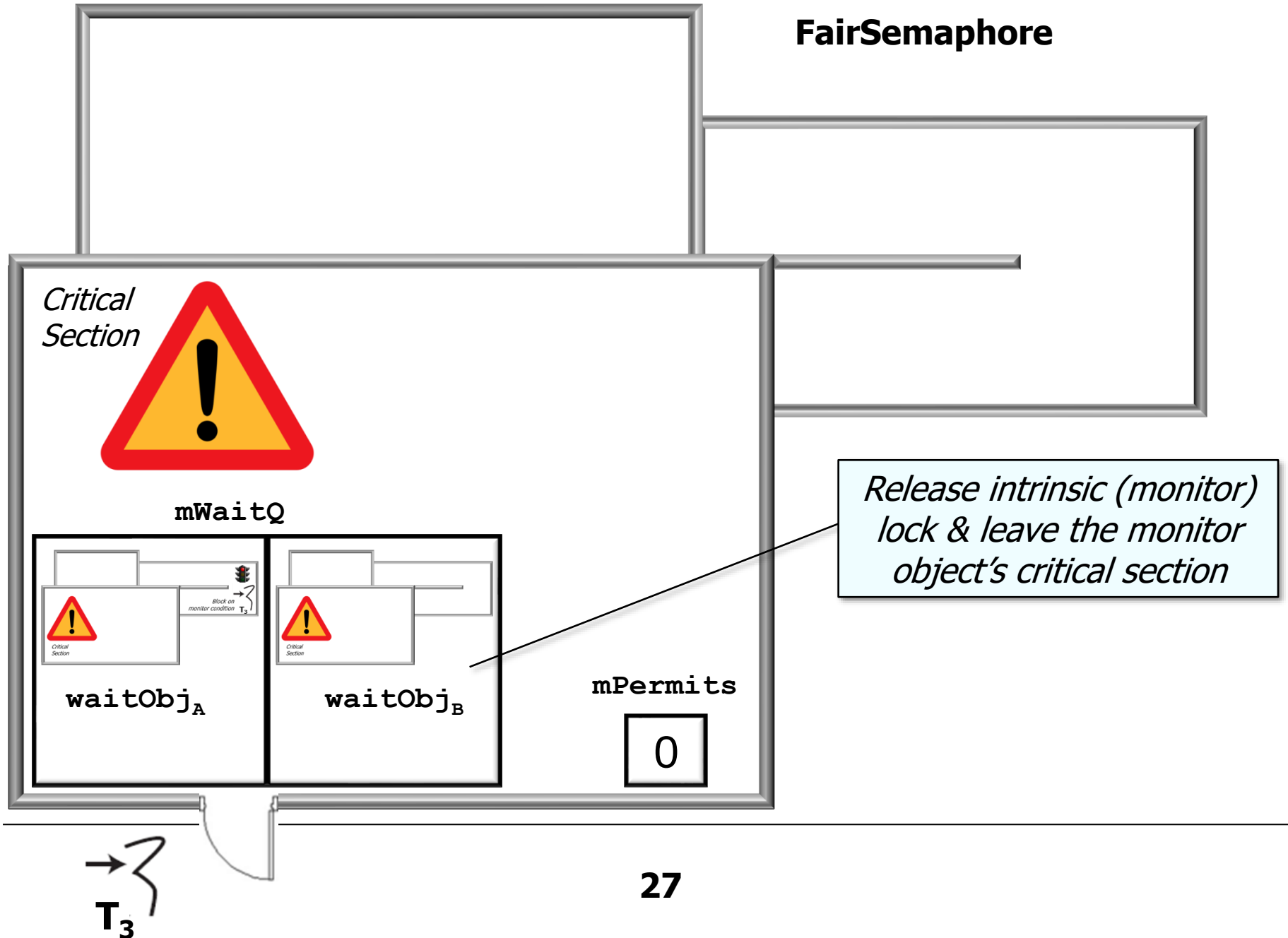
Visual Analysis of Fair Semaphore Acquire (T_3)



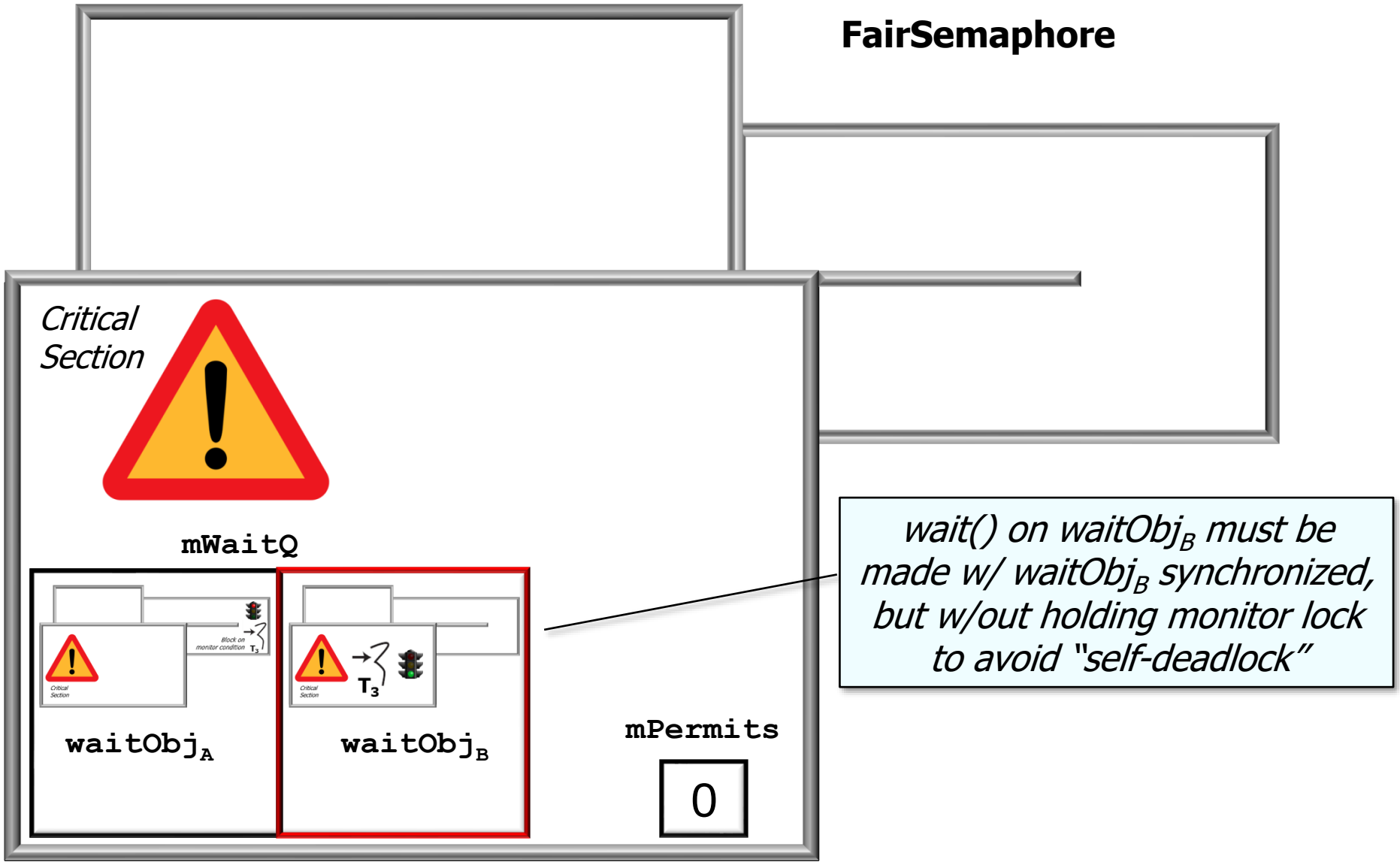
Visual Analysis of Fair Semaphore Acquire (T_3)



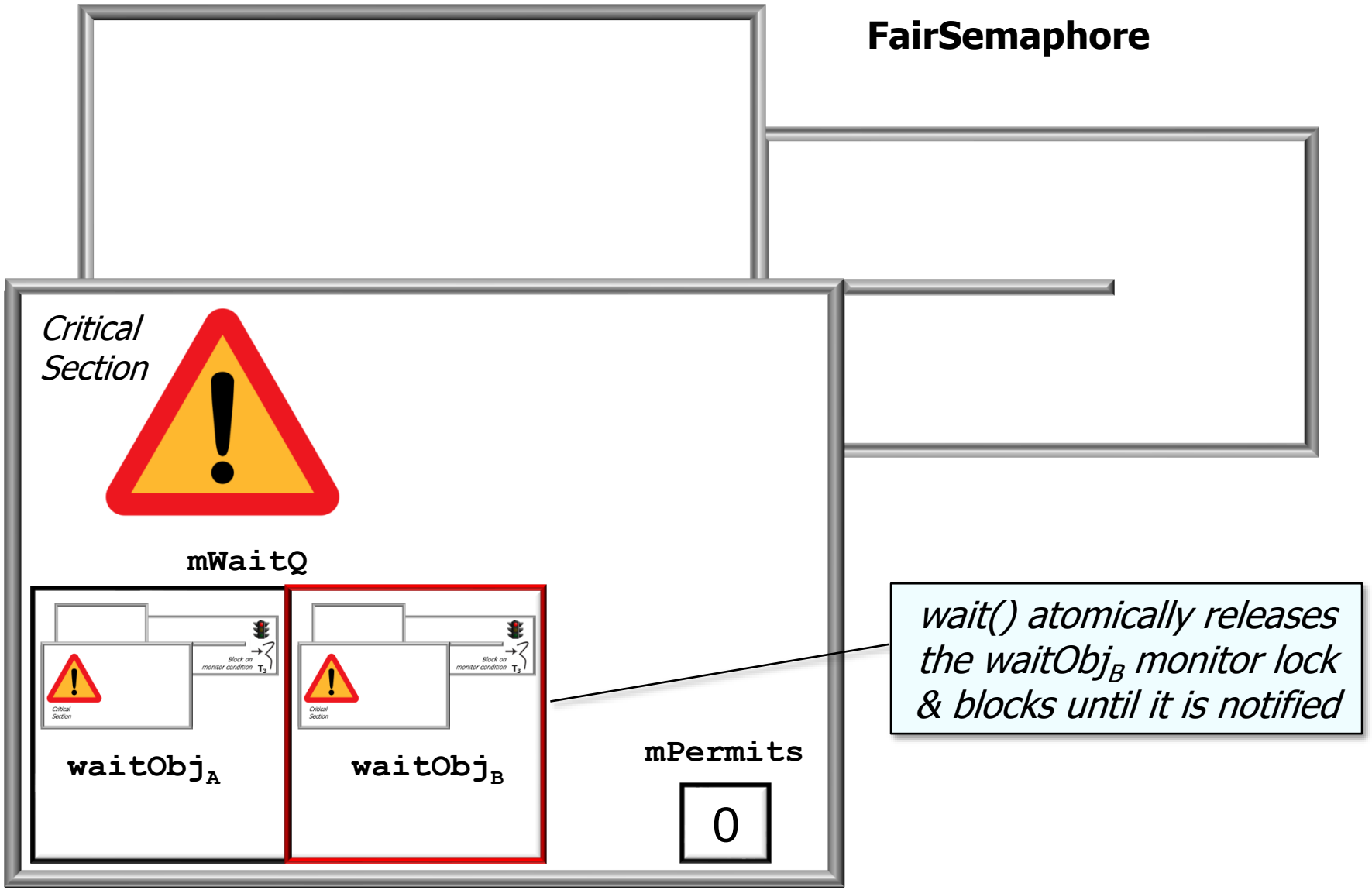
Visual Analysis of Fair Semaphore Acquire (T_3)



Visual Analysis of Fair Semaphore Acquire (T_3)

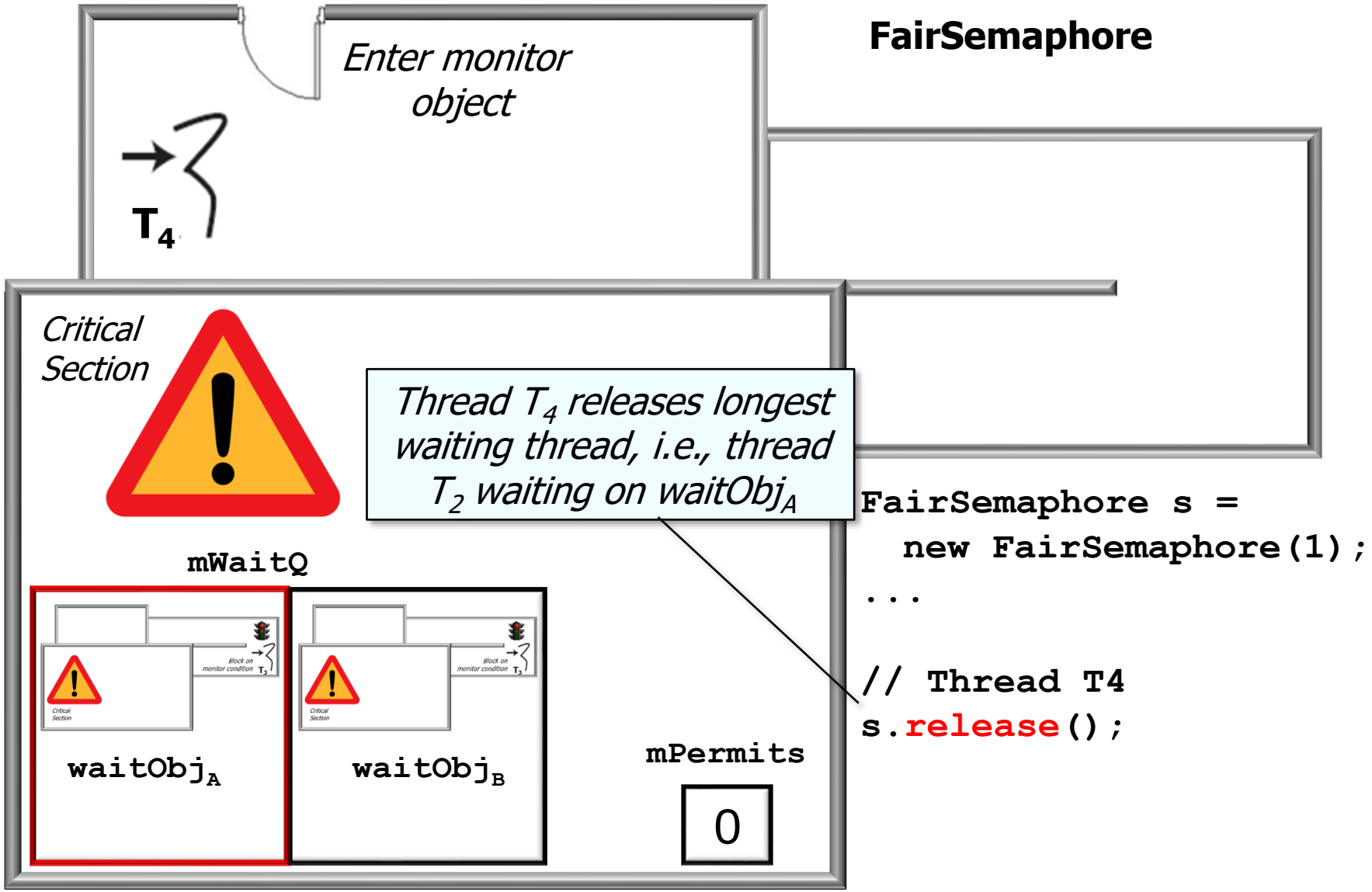


Visual Analysis of Fair Semaphore Acquire (T_3)



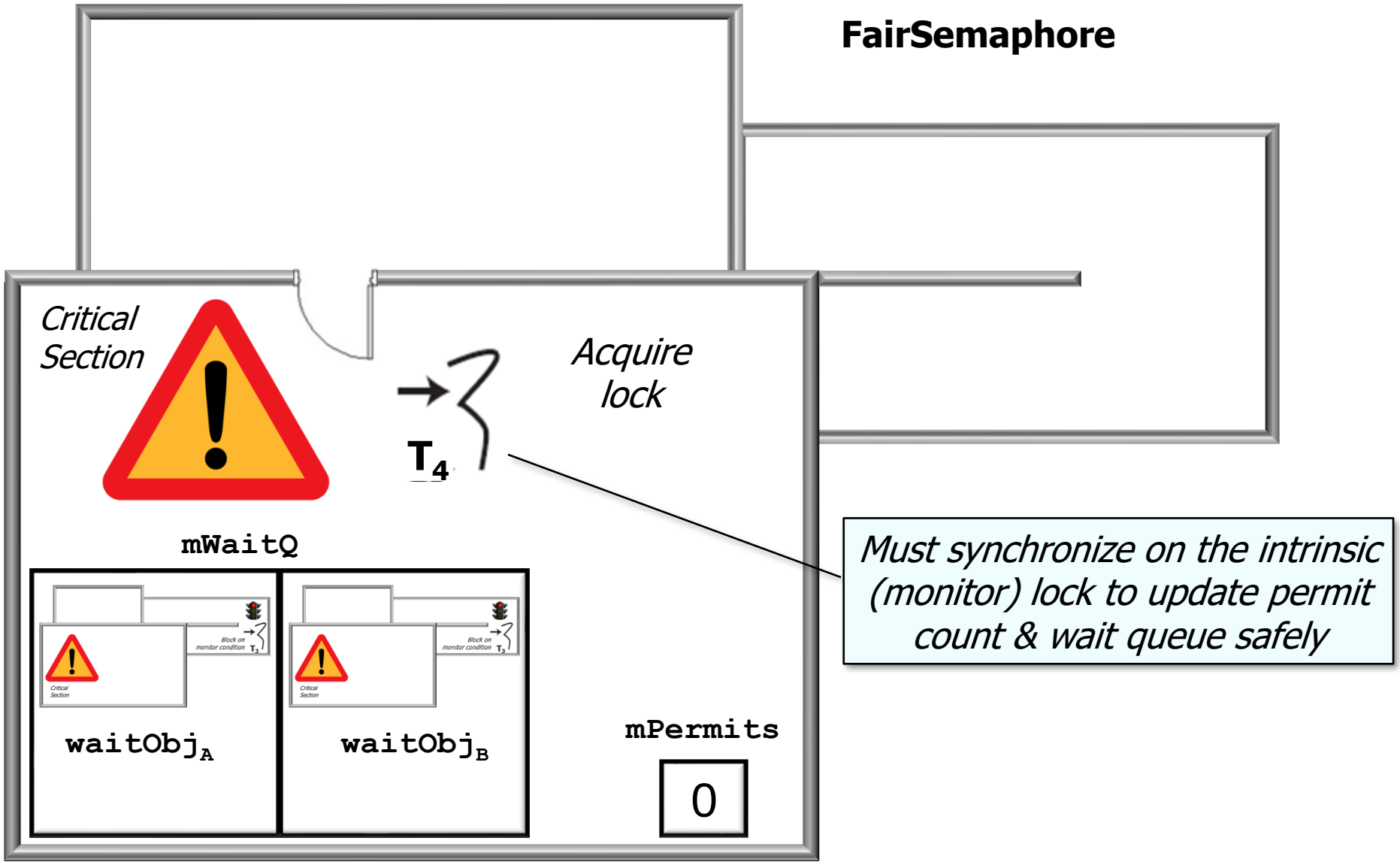
Visual Analysis of Fair Semaphore Release (T_4)

Visual Analysis of Fair Semaphore Release (T_4)

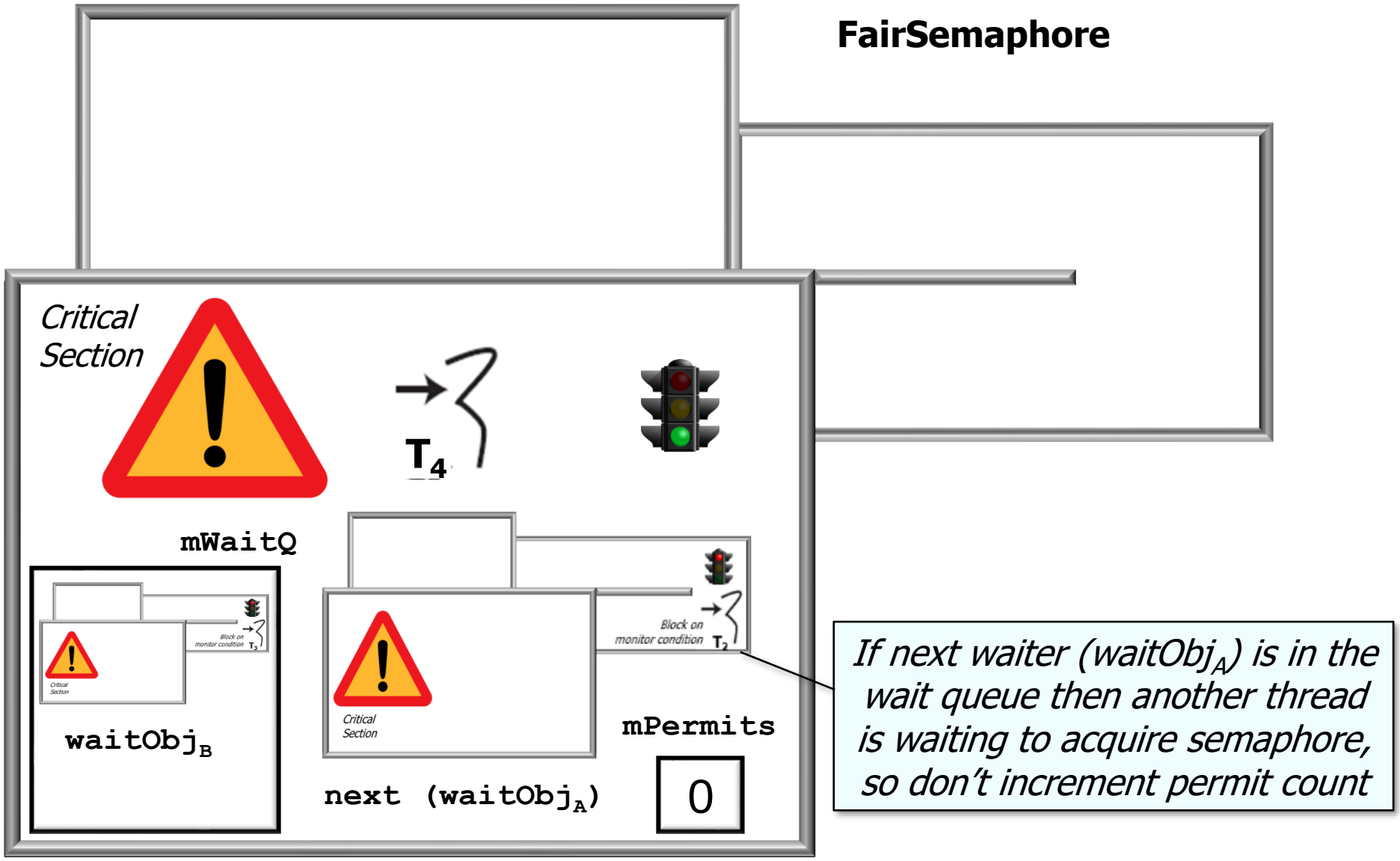


Thread T_4 could be thread T_1 or a different thread altogether

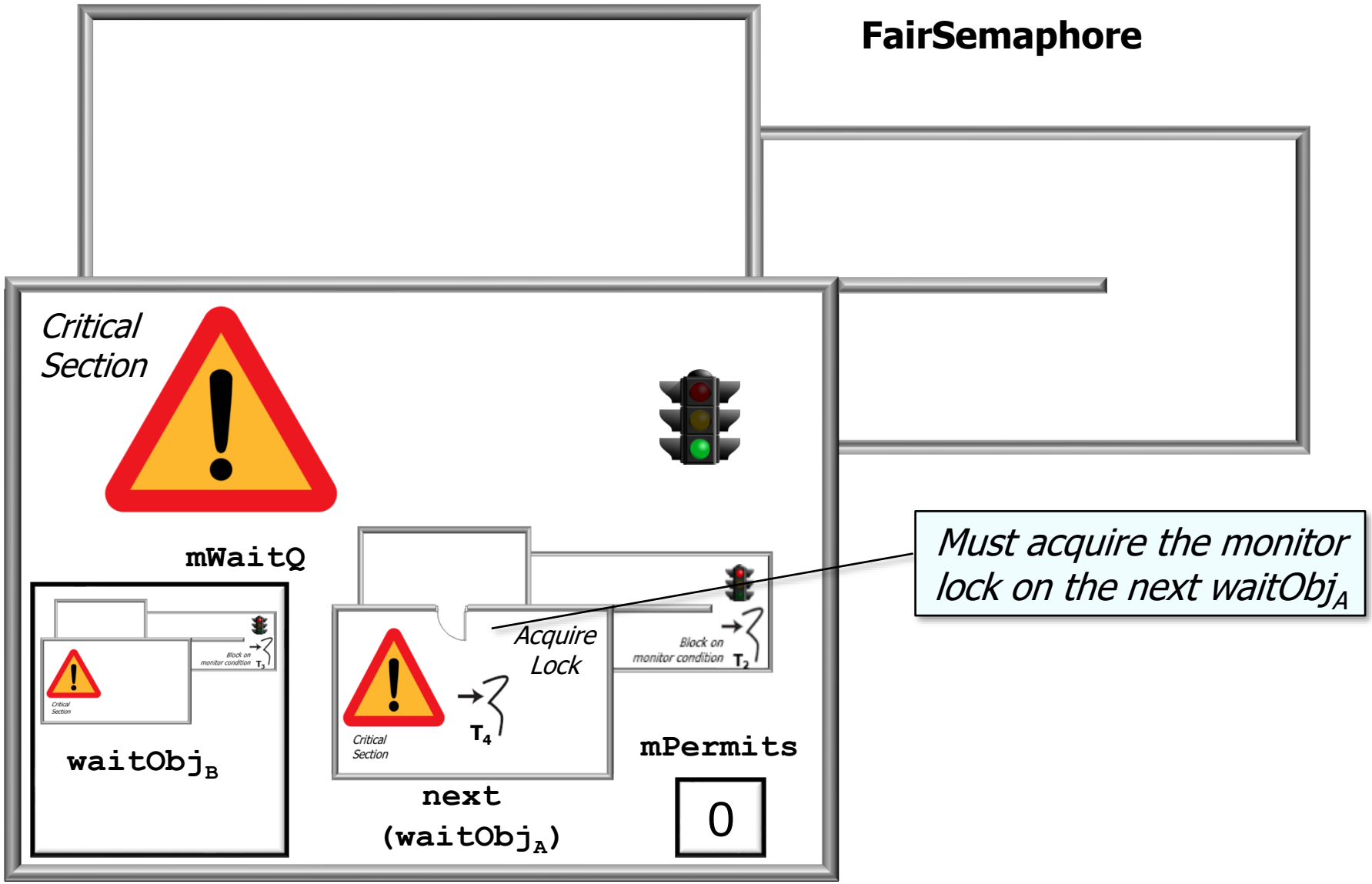
Visual Analysis of Fair Semaphore Release (T_4)



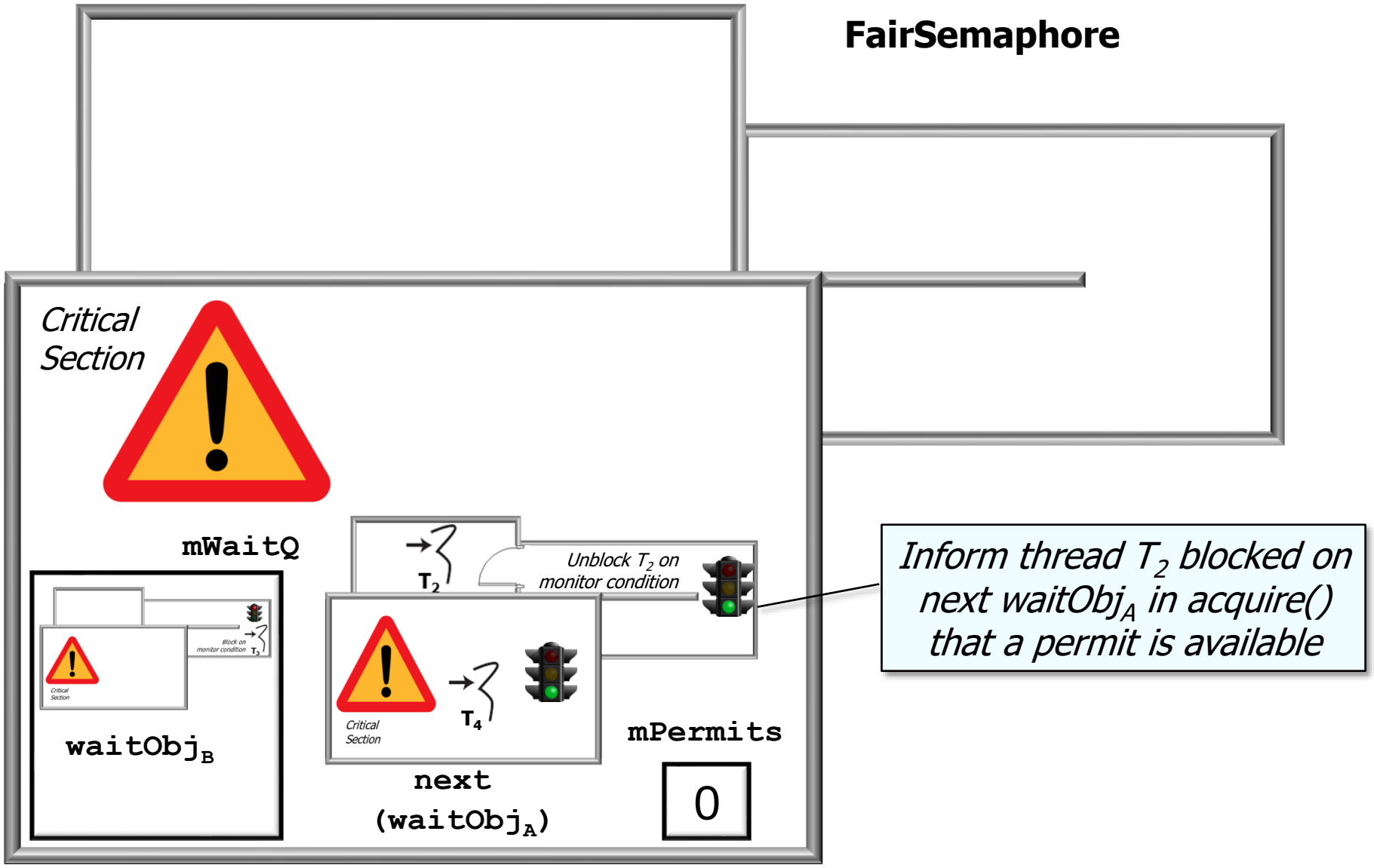
Visual Analysis of Fair Semaphore Release (T_4)



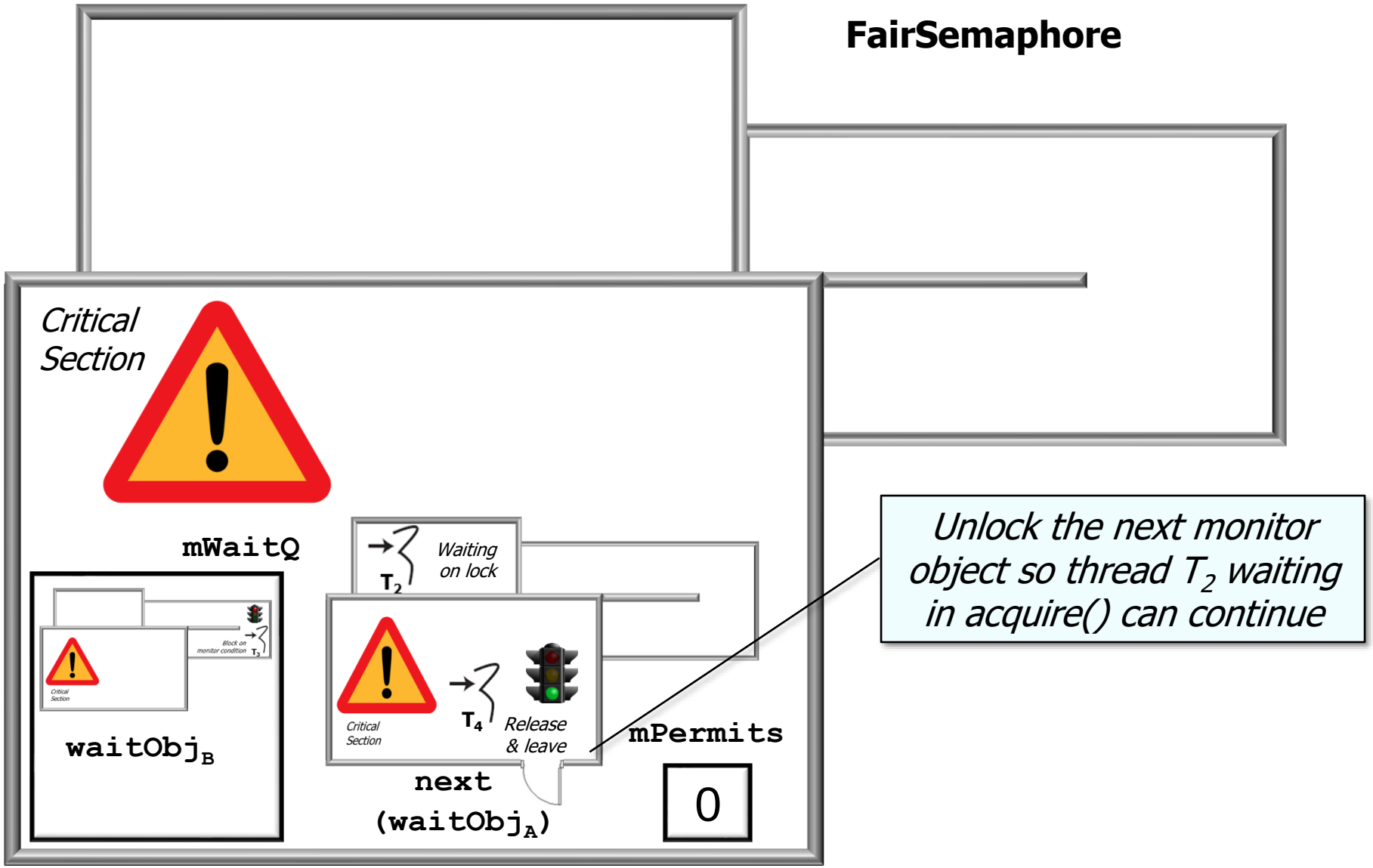
Visual Analysis of Fair Semaphore Release (T_4)



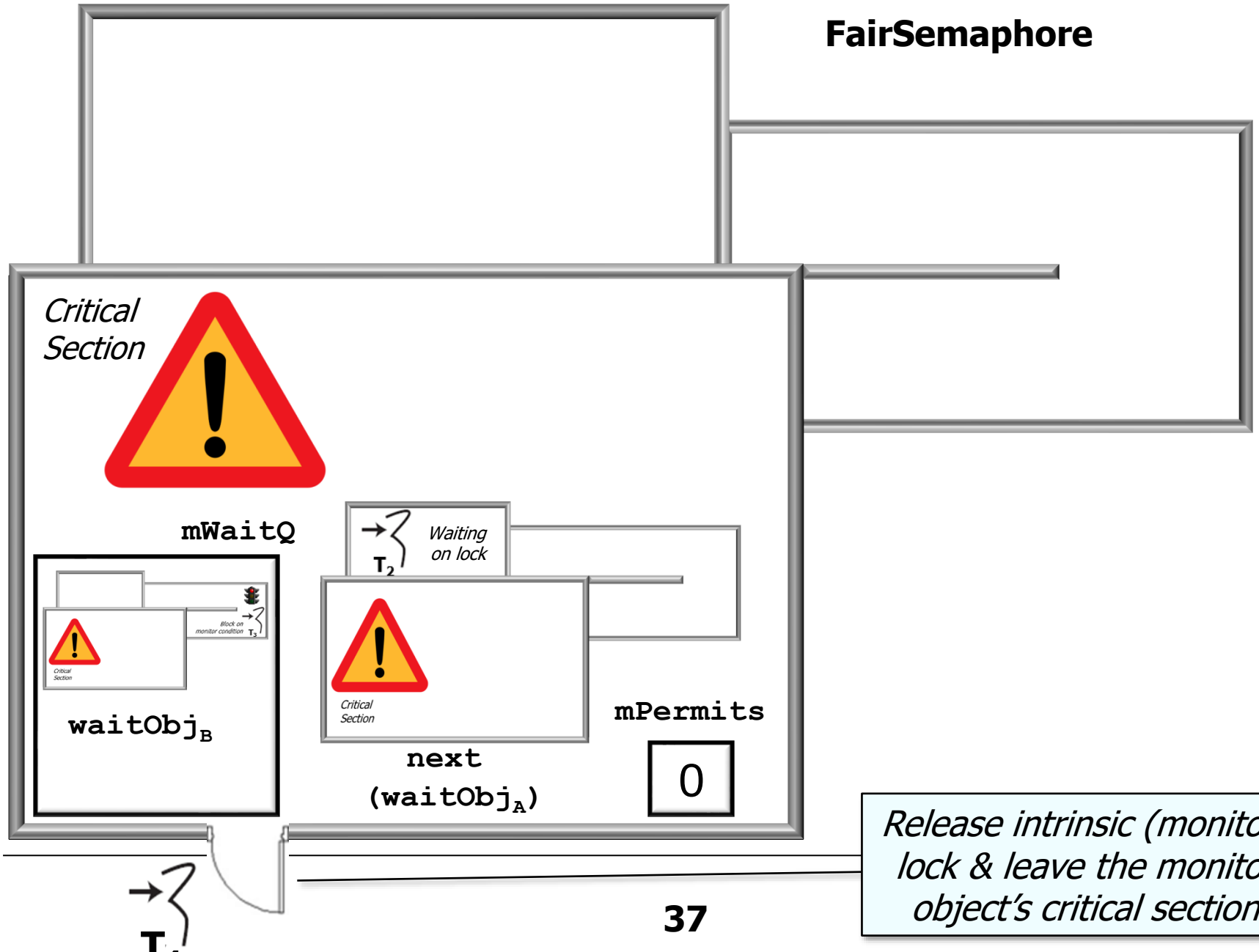
Visual Analysis of Fair Semaphore Release (T_4)



Visual Analysis of Fair Semaphore Release (T_4)

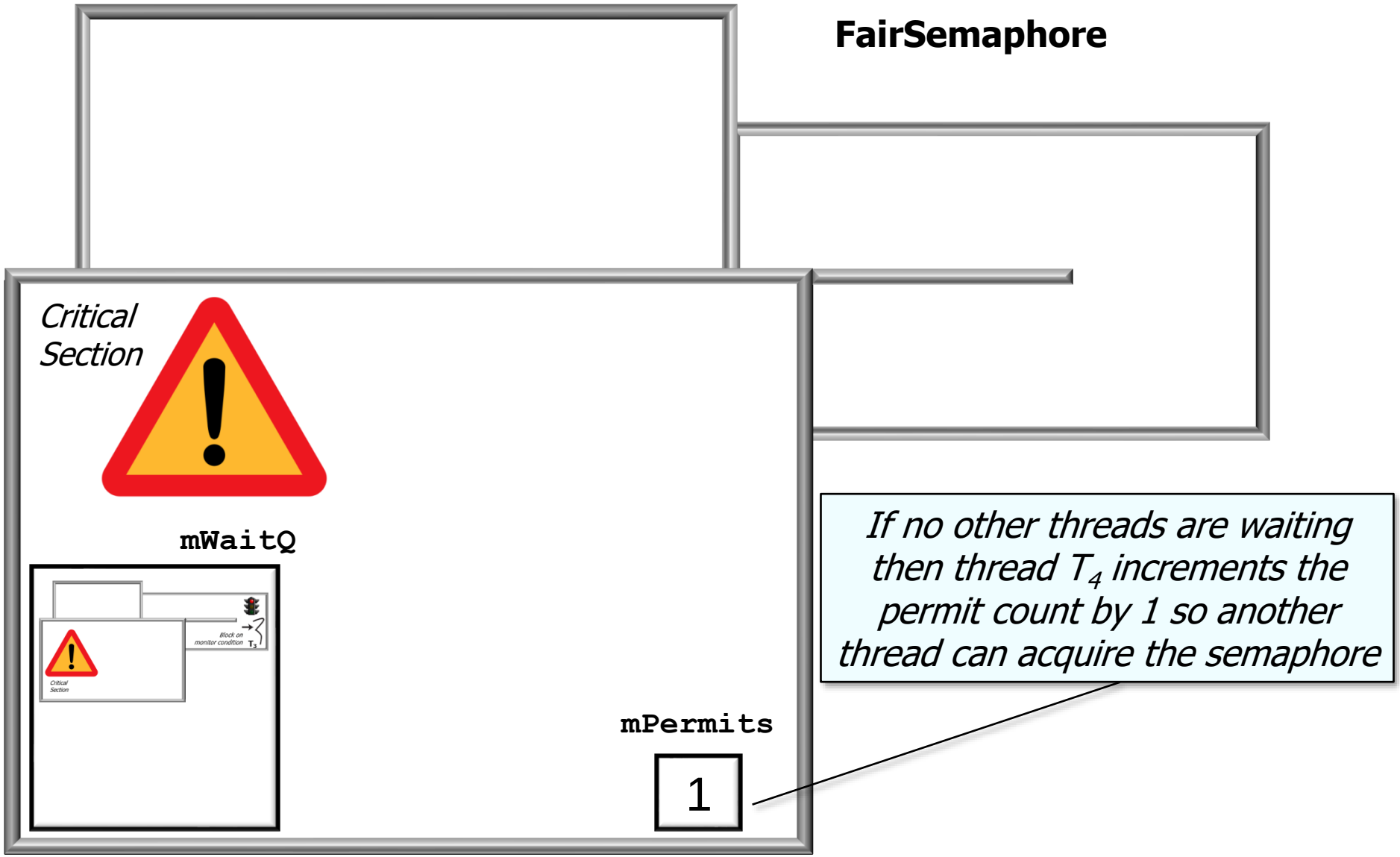


Visual Analysis of Fair Semaphore Release (T_4)



Release intrinsic (monitor) lock & leave the monitor object's critical section

Visual Analysis of Fair Semaphore Release (T_4)



End of Visualizing a Fair Semaphore Via the Specific Notification Pattern