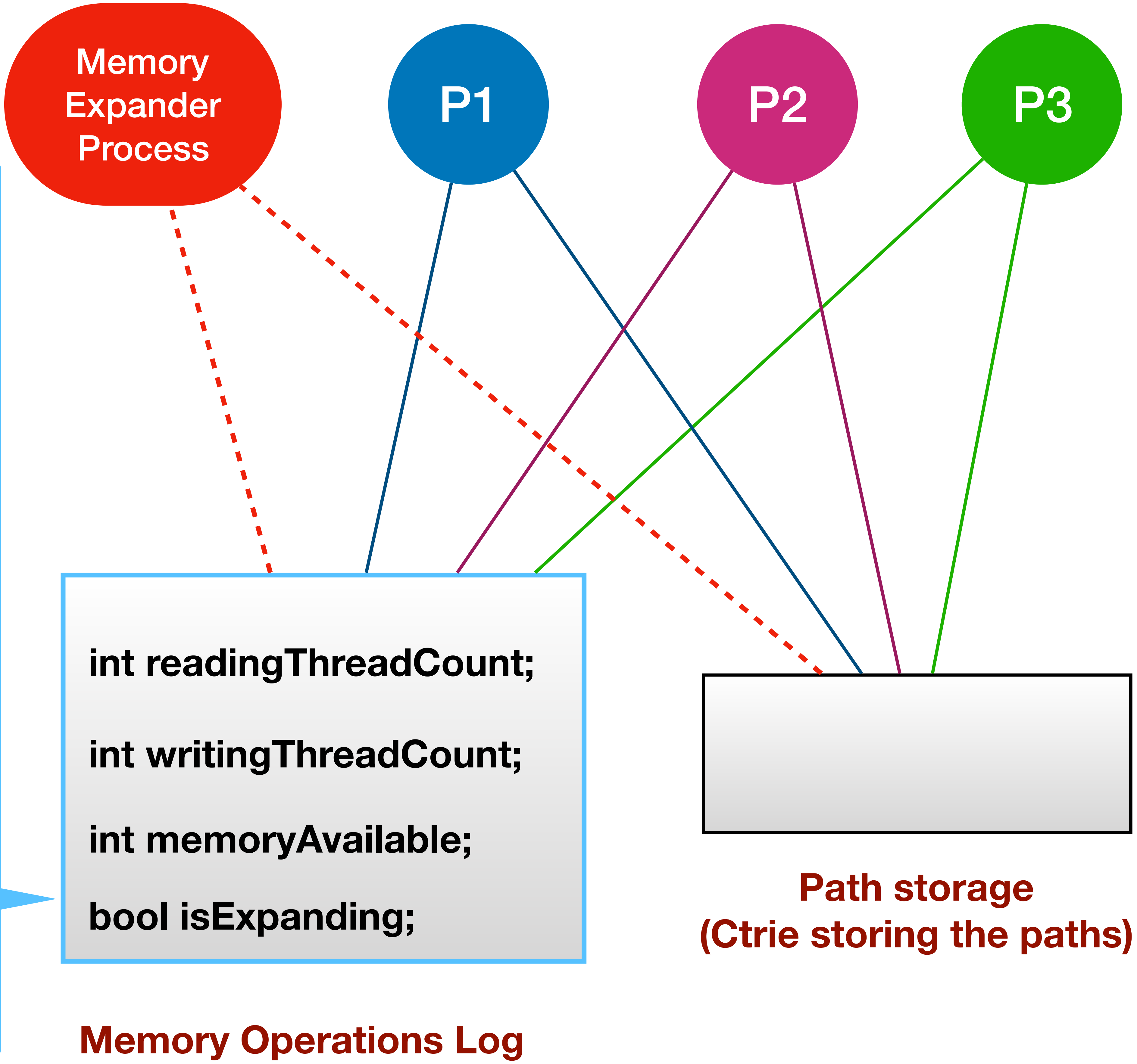
# All processes are made to share two portions of memory:-

- *readingThreadCount* detects number of threads reading
- memoryAvailable is used before mapping the shared memory to get the size which has to be mapped.
- If isExpanding is true, all processes have to loop until it is false before mmap.



```
int readingThreadCount;
int writingThreadCount;
int memoryAvailable;
bool isExpanding;
```

## Lock Free?

The two operations that can be carried out on path storage are:

- Read
- Write

## Lock Free?

## Read Operation

- To read the path storage, the thread needs to mmap memory first. It is allowed to mmap the shared memory iff *isExpanding* is false. If it is true, it keeps looping until it becomes false.
- It maps the shared memory in PROT_READ as per the *memoryAvailable* variable.
- If a thread wants to read the path storage, it first increments *readingThreadCount* by atomic CAS.
- When thread is done reading, it decrements *readingThreadCount** by atomic CAS.

*A local variable isReading resides in every thread local storage which is set as true whenever thread starts reading.

If the thread gets killed in middle of reading, a signal handler would check value of isReading and if it is true, it will decrement readingThreadCount.

## Lock Free?

## Write Operation

- To write data to the path storage, the thread needs to mmap memory first. It is allowed to mmap the shared memory iff *isExpanding* is false. If it is true, it keeps looping until it becomes false.
- It maps the shared memory in PROT_WRITE as per the *memoryAvailable* variable.
- If a thread wants to write the path storage, it first increments *writingThreadCount* by atomic CAS.
- When thread is done reading, it decrements *writingThreadCount* by atomic CAS.

*A local variable isWriting resides in every thread local storage which is set as true whenever thread starts writing.

If the thread gets killed in middle of reading, a signal handler would check value of isWriting and if it is true, it will decrement writingThreadCount.

Memory Expander Process

## Working Of Memory Expander Process In Steps

- This process is created by us by a simple C file which gets executed just before the software build starts.
- It always keeps a temporary file open with ‘X’ memory.
- Purpose of this process is to monitor the shared memory. If It detects the shared memory left to be less than a certain limit, it sets *isExpanding* variable true and waits until *readingThreadCount* and *wirtingThreadCount* variable are found to be 0.
- After that it sets the *memoryAvailable* to a new value {‘old memory size’ + ‘X’ } and appends the new temporary file with the main file to which the path data is getting stored.
- After this isExpanding is set to false.
- It again creates a new temporary file, and continues to check for limit.