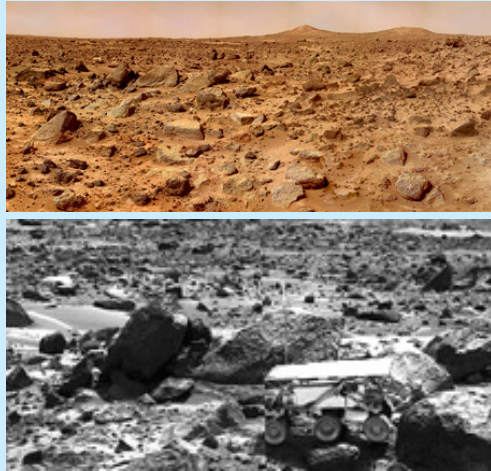
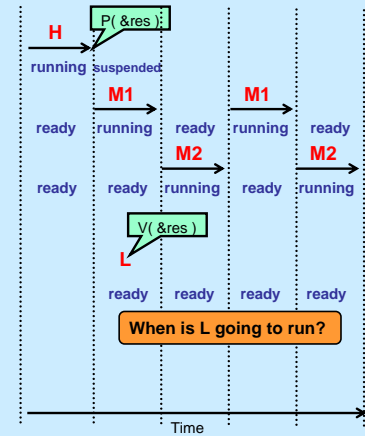


Assessed Tutorial Exercise



Priority Inversion Problem

- Assume a system has processes with 3 priority levels: high, medium and low.
- Let a high-priority process H share a resource with a low-priority process L, for example L producing data that H consumes.
- What happens when H has to wait for a semaphore that will be released by L?
- It all depends on what the medium-priority processes do!



Notes on Solution to the Priority Inversion Problem

Theory

- Let the low-priority process *inherit* the priority of the process that is waiting for it.
- To keep things simple, we will implement a special type of semaphore ("resource") that has exactly 1 producer and 1 consumer.
- We need to register producer and consumer with "resource" semaphores when they are created.
- The idea is then that when a producer blocks on a resource semaphore, the priority of the consumer is set to that of the producer.

In Simple Kernel...

- Add a mechanism for PIDs (process identifiers).
- You can copy the existing implementation of semaphores and add to it.


```
typedef struct Semaphore {
    int count;
    int producer, consumer;
    Queue waiting;
} Semaphore;
```
- Make new versions of `initSema`, `P` and `V` for resource-type semaphores.

Changing the Priority of a Process

- Do we want to change the priority of the producer process in all states?
- In running, what do we have to do?
- In delayed?
- In suspended?
- In ready? (This is the only tedious one.)
- Add a mechanism for keeping track of process state.

