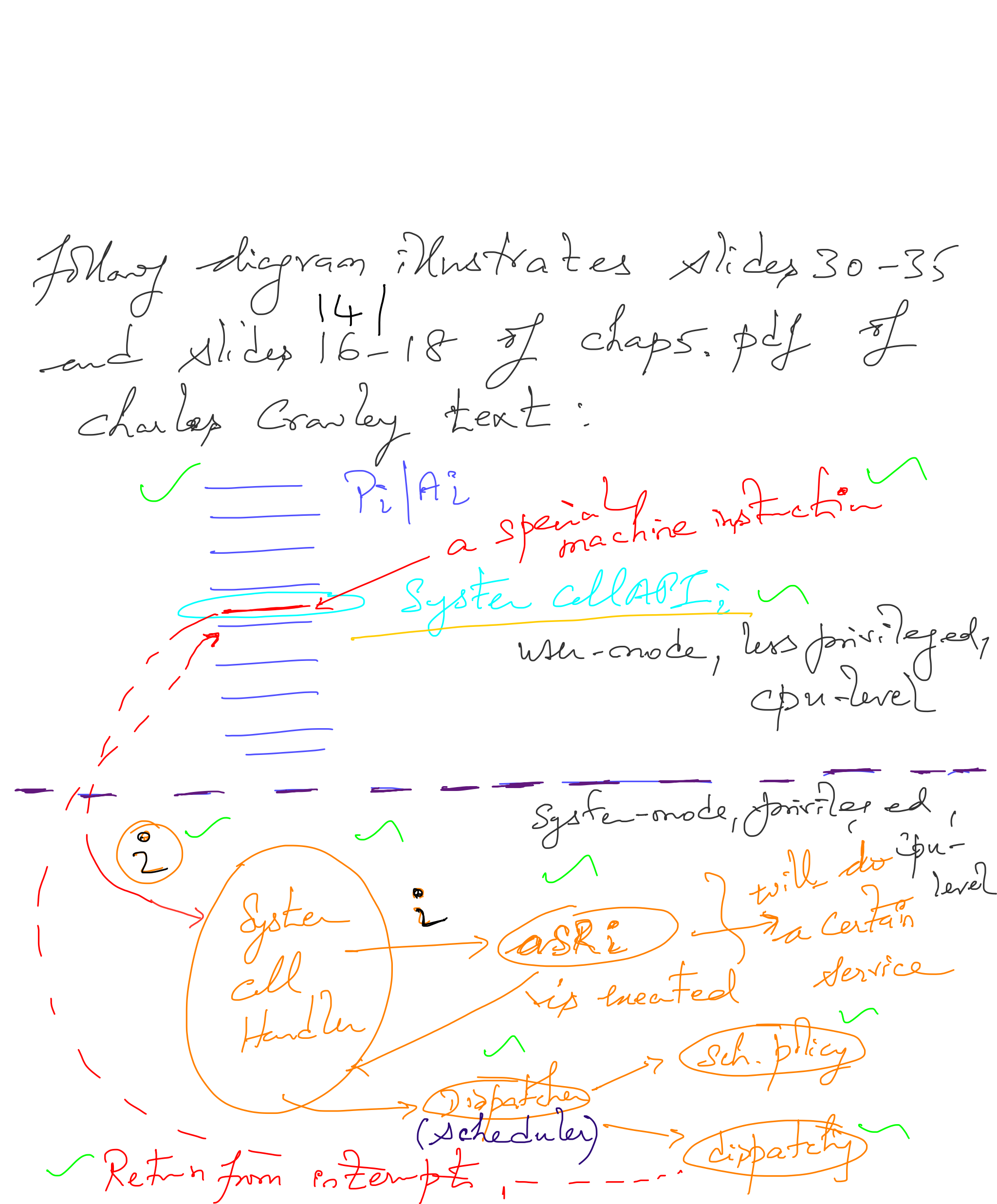
Following diagram illustrates slides 30-35 and slides 16-18 of chap5.pdf of charles Crowley text:

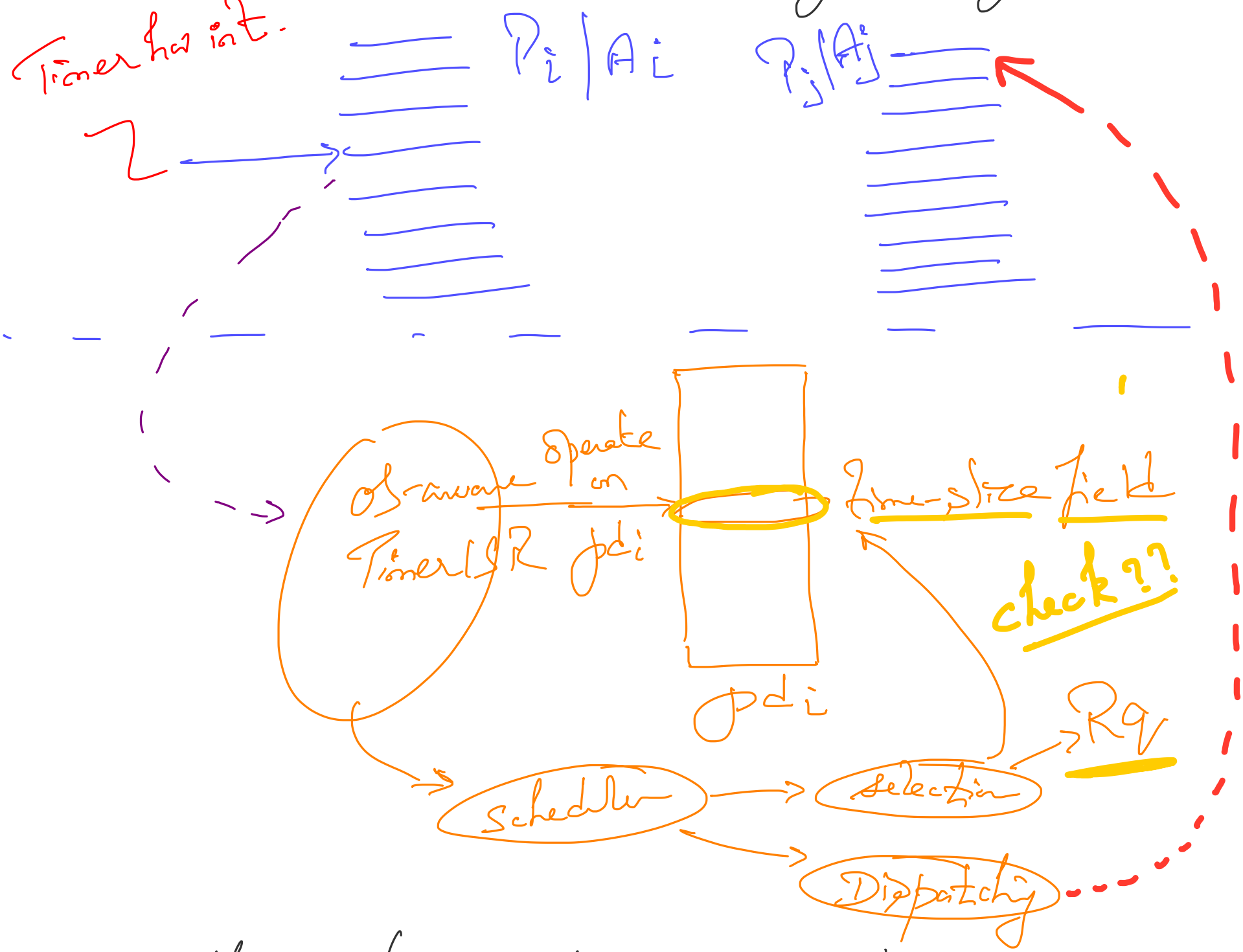


→ in the above diagram, (i)
will be a system call no ??

→ based on i, a specific
Routine i is processed

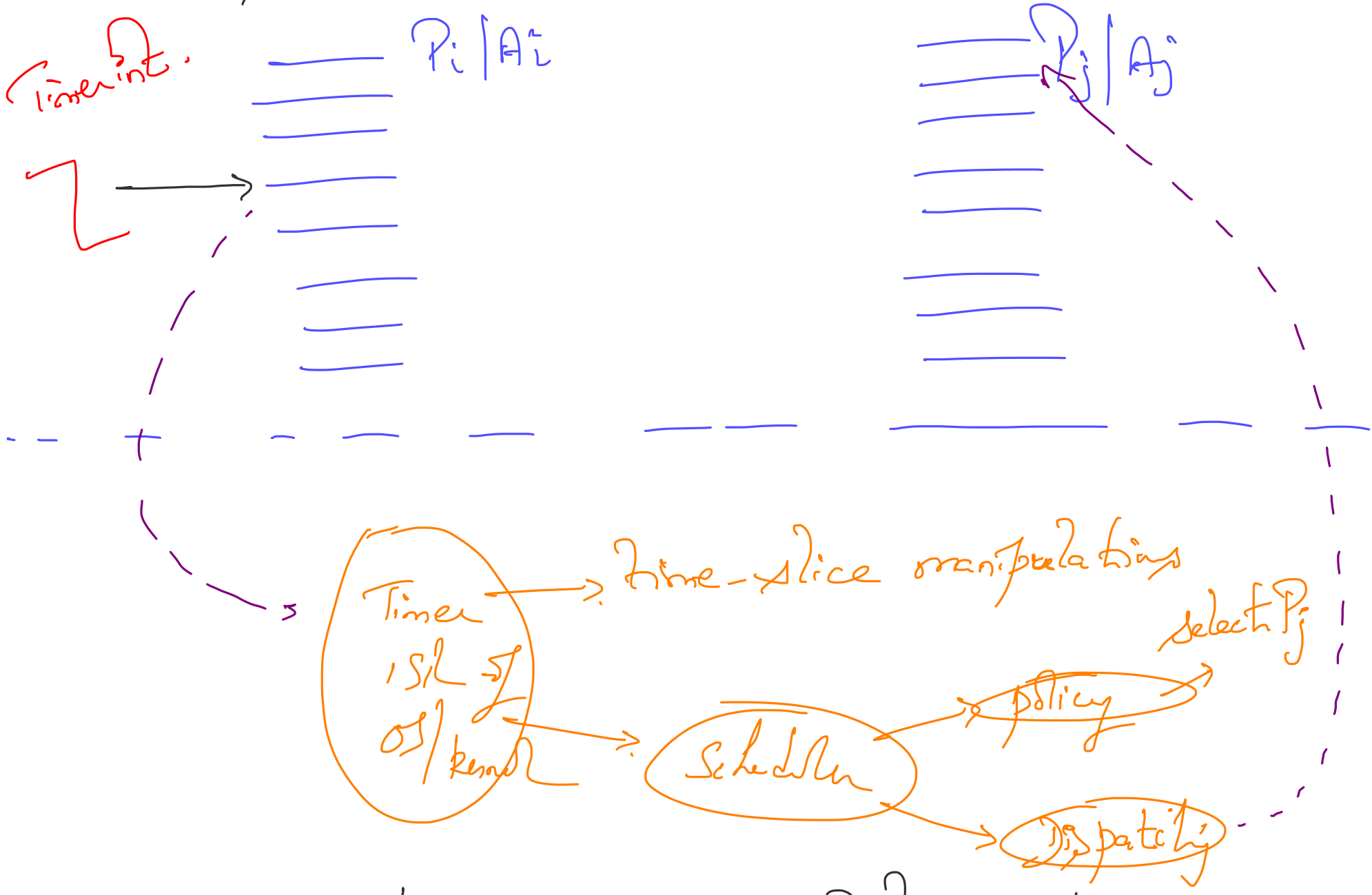
→ refer to cc_2.txt and
chap5.pdf, for further
details

→ refer to slide 20 of chap5.pdf and connect the following diagram:



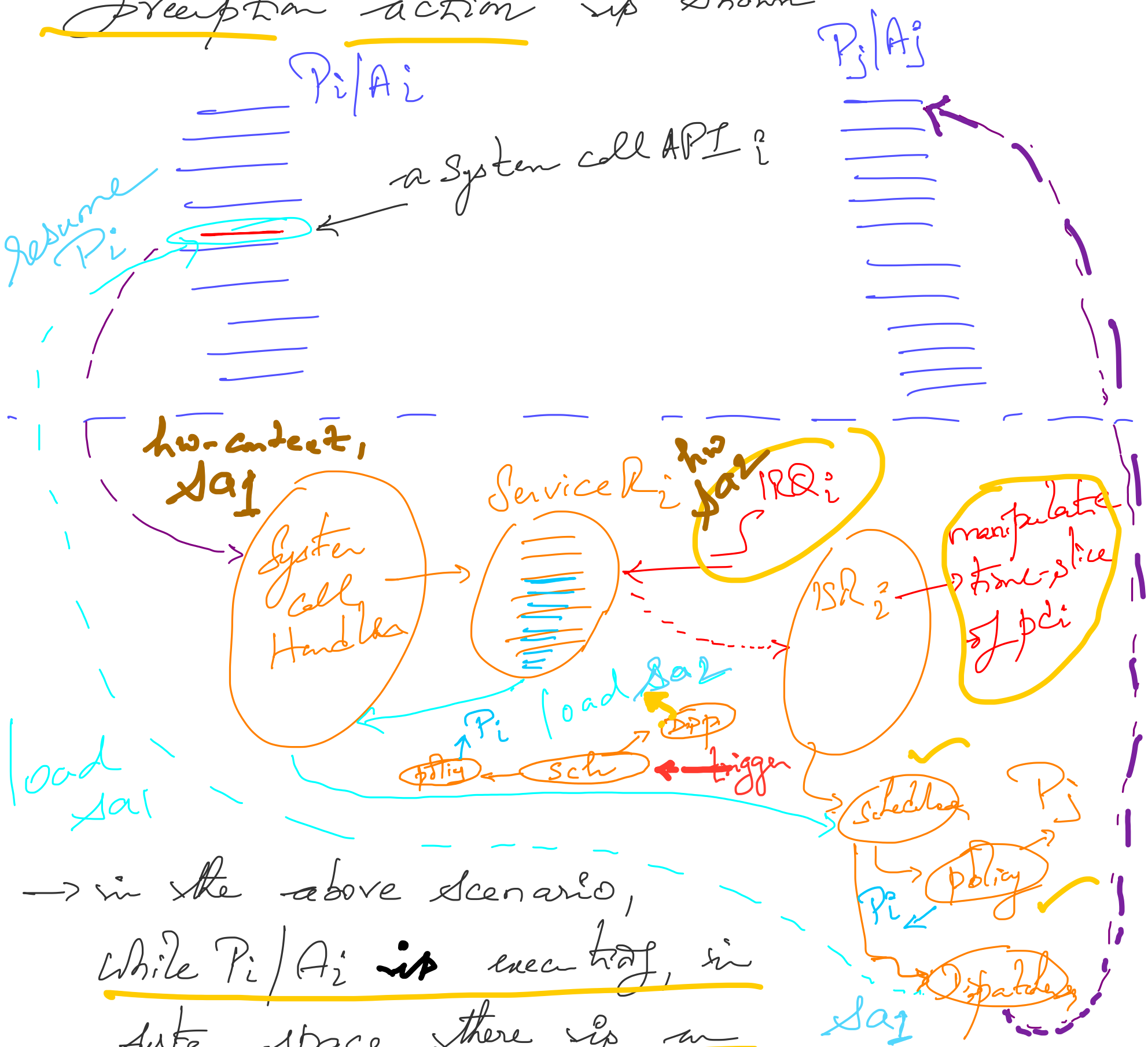
→ in the above scenario, there is a time-slice expiry, for pdi/process and there is a preemption of Pi — further, Pi is scheduled/dispatched

→ in the following illustration, a typical user-space preemption is shown:



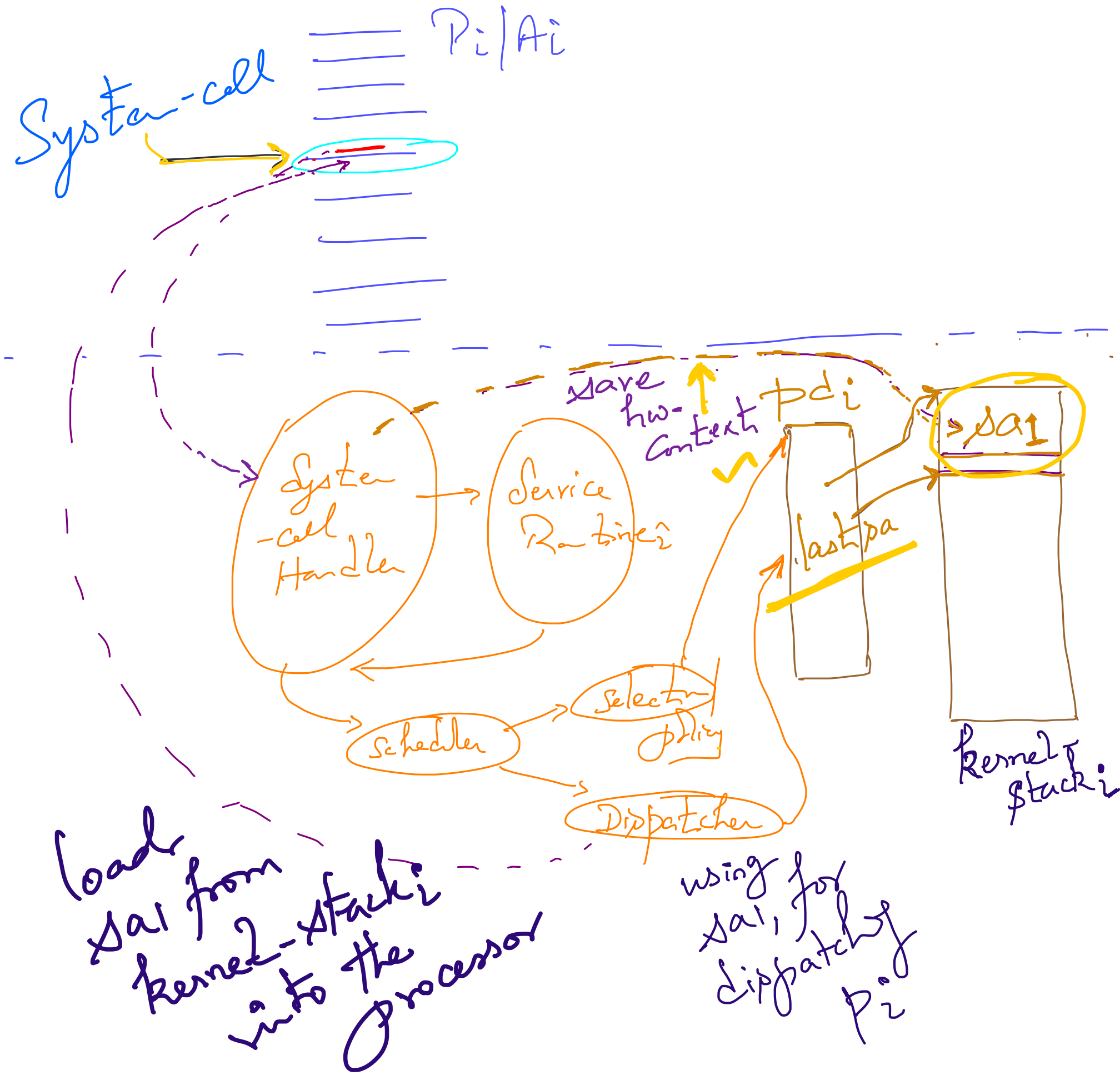
→ in the above scenario, while P_i | App_i are executing in user-space, there is an interrupt event, followed by a preemption action — so, known as an user-space preemption action

→ in the following illustration, a syste-space preemption action is shown

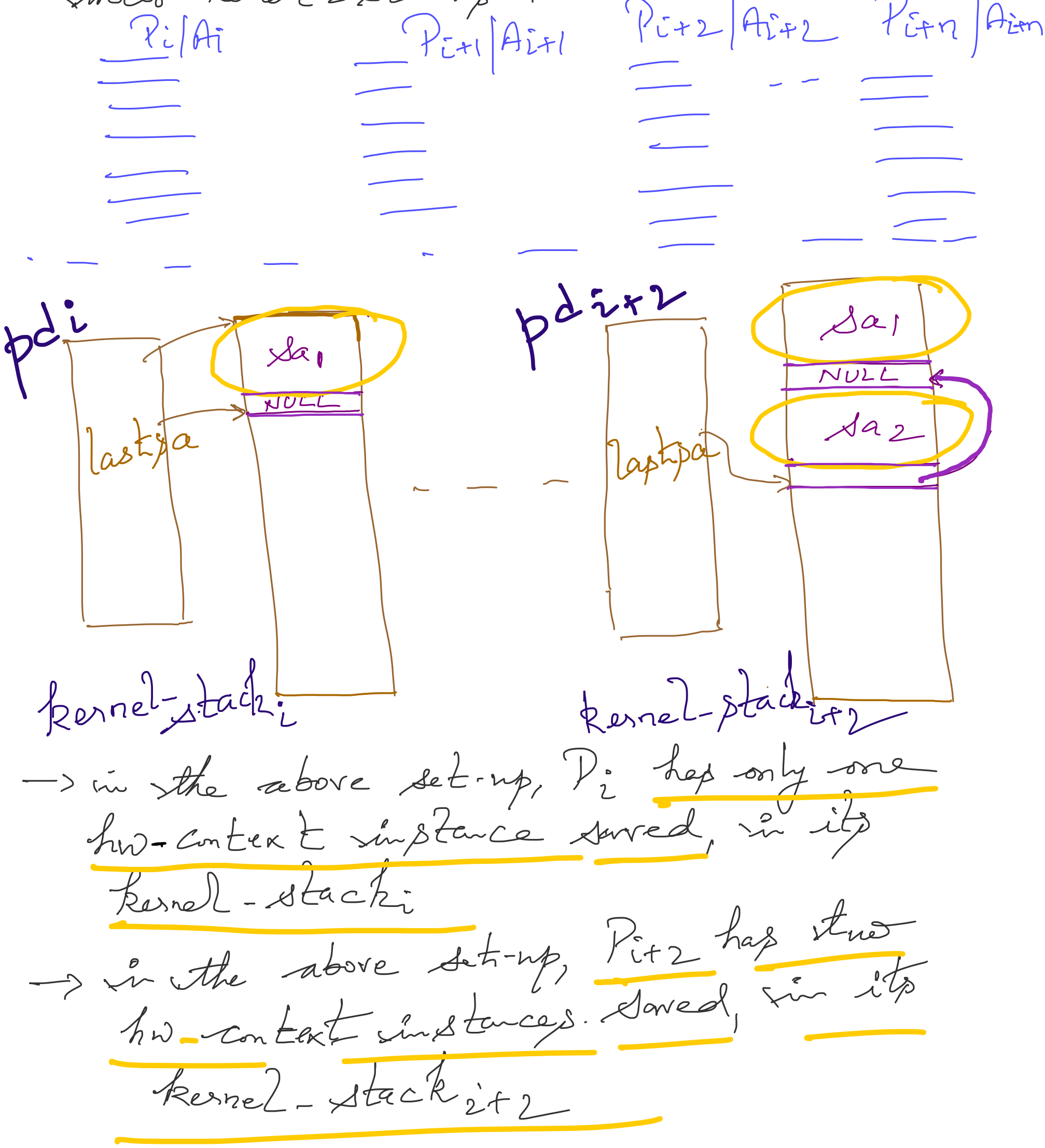


→ in the above scenario, while Pi/Ai is executing, in syste-space, there is an int. event, followed by a preemption action, which is known as a syste space preemption action

→ following diagram illustrates slides 47–58 of chap6.pdf.



→ This digram illustrates different pds & their kernel-stacks !



→ in the above set-up, P_i has only one hw-context instance saved, in its kernel-stack$_i$

→ in the above set-up, P_{i+2} has two hw-context instances saved, in its kernel-stack$_{i+2}$

→ in the above set-up, scenario 1 (left diagram) is, due to a system-call trigger or an interrupt trigger, in user-space

→ in the above set-up, scenario 2 (right diagram)

✓ is, due to a system call trigger, followed by an interrupt trigger

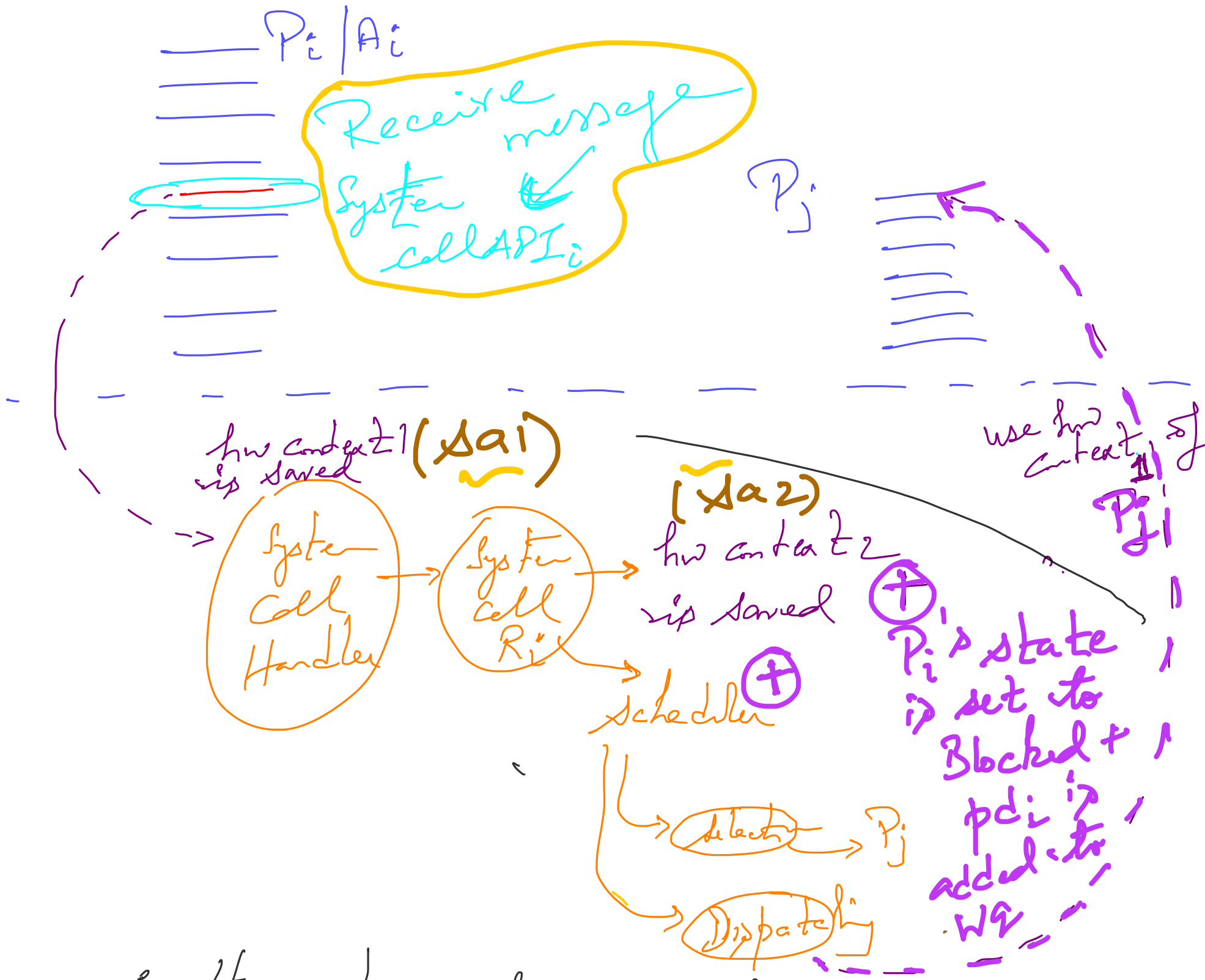
→ in scenario 3, it may be, due to a system-call trigger, followed by a blocking operation,

↳ in the system call

↳ in this scenario-3, Pi is blocked & its kernel-execution context is saved, in SA2

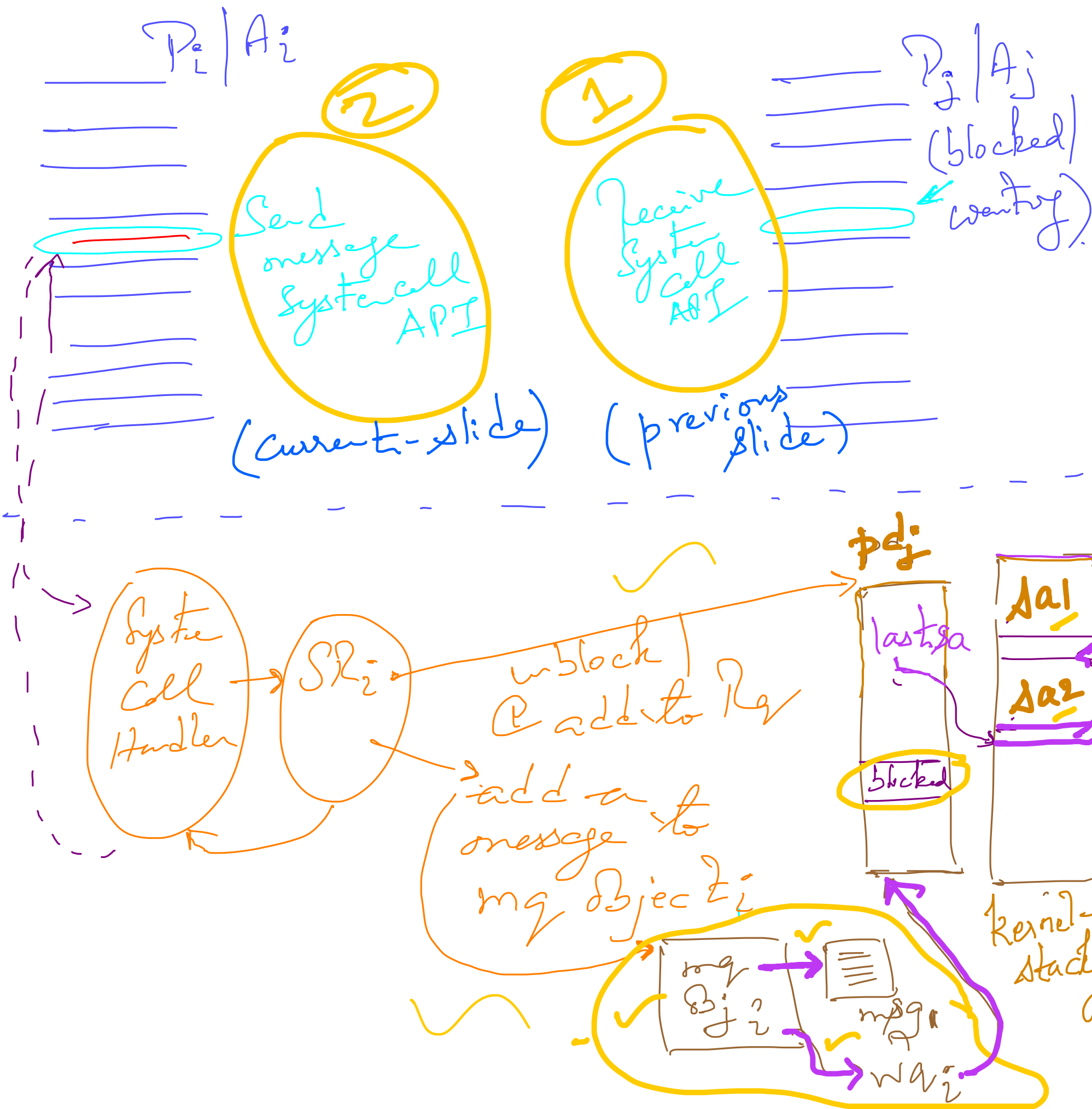
↳ Basically, here is a blocking-operation, in the system-call SRi

→ Following diagram illustrates a system call API_i and its blocking operation;



→ in the above scenario, three hw context instances are saved, in kernel stacks of P_i — refer to slides 7 – 8 above !!

→ in the following illustration, a process P_i invokes a send message system call API — related actions are illustrated



→ based on the above send message system call API and working of Pj, following actions are done, when Pj is scheduled and dispatched