ບົດທີ 7

ການຈັດການໜ່ວຍຄວາມຈຳຫຼັກ
(Real Memory Management)

ເນື້ອໃນຫຍໍ້

- ສະເໜີເບື້ອງຕົ້ນ
- ການຈັດການໜ່ວຍຄວາມຈຳ
- ຍຸດທະຍາສາດໃນການຈັດການໜ່ວຍຄວາມຈຳ
- ການຈັດສັນໜ່ວຍຄວາມຈຳແບບຕໍ່ເນື່ອງ ແລະ ບໍ່ຕໍ່ເນື່ອງ
- ການໃຫ້ຂະບວນການຄຽວໃຊ້ໜ່ວຍຄວາມຈຳທັງໝົດແບບຕໍ່ເນື່ອງ
- ການແບ່ງໜ່ວຍຄວາມຈຳໃຫ້ຫຼາຍຂະບວນການທີ່ມີຂະໜາດບໍ່ປ່ຽນແປງ
- ການແບ່ງໜ່ວຍຄວາມຈຳໃຫ້ຫຼາຍຂະບວນການທີ່ມີຂະໜາດປ່ຽນແປງ
- ການໃຫ້ຫຼາຍຂະບວນການສະຫຼັບກັນໃຊ້ໜ່ວຍຄວາມຈຳບ່ອນໃດໜຶ່ງ

ສະເໜີເບື້ອງຕົ້ນ

- ໜ່ວຍຄວາມຈຳໄດ້ແບ່ງອອກເປັນລະດັບ
  - ໜ່ວຍຄວາມຈຳຫຼັກ
    - ລາຄາຂ້ອນຂ້າງແພງ
    - ຂະໜາດຄວາມຈຸນ້ອຍ
    - ເຮັດວຽກໄດ້ໄວ
  - ໜ່ວຍຄວາມຈຳສຳຮອງ
    - ລາຄາຖືກ
    - ຂະໜາດຄວາມຈຸໃຫຍ່
    - ເຮັດວຽກຊ້າ
  - ໜ່ວຍຄວາມຈຳຫຼັກຈະຕ້ອງໄດ້ມີການບໍລິຫານຈັດການຢ່າງລະເອົາໃຈໃສ່

ສະເໜີເບື້ອງຕົ້ນ

- ໜ່ວຍຄວາມຈຳຫລັກເປັນສູນກາງການດຳເນີນງານຂອງລະບົບຄອມພິວເຕີໃນຍຸກໃໝ່
- ໜ່ວຍຄວາມຈຳຫລັກປະກອບດ້ວຍຈຳນວນແຖວທີ່ໃຊ້ເກັບຂໍ້ມູນຫລາຍໆແຖວ ແຕ່ລະແຖວເອີ້ນວ່າ Word ຫຼື Byte ຊຶ່ງມີທີ່ຢູ່ (Address) ເປັນຂອງຕົວເອງ
- ການໃຊ້ໜ່ວຍຄວາມຈຳຫລັກເຮັດໄດ້ໂດຍການອ່ານ ຫຼື ຂຽນຂໍ້ມູນລົງໃນແຕ່ລະ Word ລວມທັງການດຶງເອົາຄຳສັ່ງຈາກໜ່ວຍຄວາມຈຳເພື່ອສົ່ງໃຫ້ Program Counter
- ໜ່ວຍຄວາມຈຳຫຼັກຈະເກັບສະເພາະຂໍ້ມູນຕ່າງໆເທົ່ານັ້ນ ໂດຍບໍ່ຮູ້ວ່າພວກເປັນຂໍ້ມູນແນວໃດ, ມີຄວາມສຳພັນແນວໃດ, ໃຊ້ເຮັດຫຍັງ

ການຈັດການໜ່ວຍຄວາມຈຳ

- ສາມາດເຮັດໄດ້ຫຼາຍວິທີ
  - ໃຫ້ຂະບວນການດຽວໃຊ້ໜ່ວຍຄວາມຈຳຫຼັກທັງໝົດ
  - ແບ່ງໜ່ວຍຄວາມຈຳອອກເປັນຫຼາຍສ່ວນ ແຕ່ລະສ່ວນໃຫ້ແຕ່ລະຂະບວນການໃຊ້
    - ອາດຈະເປັນການຈັດສັນແບບປ່ຽນແປງໄດ້ ຫຼື ແບບບໍ່ປ່ຽນແປງໄດ້
  - ແນວໂນ້ມໃນອະນາຄົດ:
    - ເຫັນວ່າບັນດາໂປຣແກຣມຕ່າງໆມີຄວາມຕ້ອງການໃຊ້ໜ່ວຍຄວາມຈຳຫຼາຍຂຶ້ນເລື້ອຍໆ
    - ຕົວຢ່າງ: ລະບົບປະຕິບັດການ Windows ຂອງບໍລິສັດໄມໂຄຣຊອບ ຕັ້ງແຕ່ລຸ້ນທຳອິດຈົນເຖິງປະຈຸບັນເຫັນວ່າໃຊ້ໜ່ວຍຄວາມຈຳຫຼາຍຂຶ້ນເປັນລຳດັບ

ภาพจัดภาพพ่อยถอาพจำ

Operating System	Release Date	Minimum Memory Requirement	Recommended Memory
Windows 1.0	November 1985	256KB	
Windows 2.03	November 1987	320KB	
Windows 3.0	March 1990	896KB	1MB
Windows 3.1	April 1992	2.6MB	4MB
Windows 95	August 1995	8MB	16MB
Windows NT 4.0	August 1996	32MB	96MB
Windows 98	June 1998	24MB	64MB
Windows ME	September 2000	32MB	128MB
Windows 2000 Professional	February 2000	64MB	128MB
Windows XP Home	October 2001	64MB	128MB
Windows XP Professional	October 2001	128MB	256MB

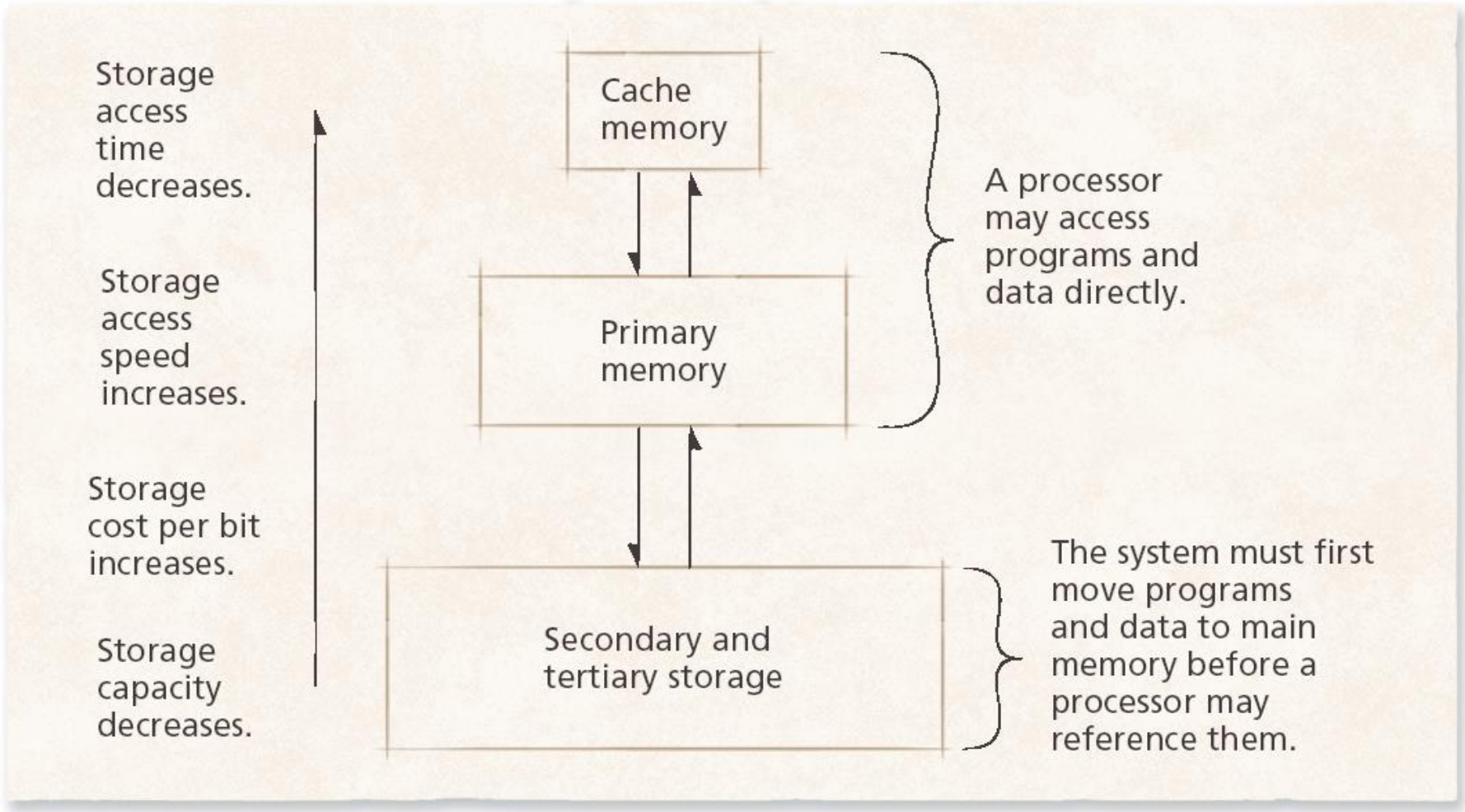
ການບໍລິຫານຈັດການໜ່ວຍຄວາມຈຳຫຼັກ

- ເປັນວິທີການ ຫຼື ຍຸດທະສາດເພື່ອເຮັດໃຫ້ການເຮັດວຽກຂອງໜ່ວຍຄວາມຈຳໄວທີ່ສຸດ, ບັນຈຸຂໍ້ມູນໄດ້ຫຼາຍທີ່ສຸດ
- ເຮັດໂດຍໂປຣແກຣມບໍຫານໜ່ວຍຄວາມຈຳ (Memory Manager) ຊຶ່ງເປັນຜູ້ກຳໜົດວ່າ
  - ຂະບວນການໃດຈະໄດ້ຢູ່ໃນໜ່ວຍຄວາມຈຳຫຼັກ
  - ແຕ່ລະຂະບວນການຈະເຂົ້າໃຊ້ໜ່ວຍຄວາມຈຳໄດ້ຫຼາຍເທົ່າໃດ
  - ແຕ່ລະຂະບວນການສາມາດເຂົ້າໄປໃຊ້ໜ່ວຍຄວາມຈຳບ່ອນໃດແດ່

ລຳດັບຊັ້ນຂອງໜ່ວຍຄວາມຈຳ

- ໜ່ວຍຄວາມຈຳຫຼັກ (Main memory)
  - ຈະເກັບຄຳສັ່ງ ແລະ ຂໍ້ມູນທີ່ໂປຣແກຣມຕ້ອງການໃນປະຈຸບັນເທົ່ານັ້ນ
- ໜ່ວຍຄວາມຈຳສຳຮອງ (Secondary storage)
  - ເກັບຂໍ້ມູນທີ່ບໍ່ທັນຖືກໃຊ້ ແລະ ໂປຣແກຣມທີ່ບໍ່ໄດ້ເຮັດວຽກໃນປະຈຸບັນ
- ໜ່ວຍຄວາມຈຳຊ່ວຄາວ (Cache memory)
  - ມີຄວາມໄວສູງຫຼາຍ
  - ປົກກະຕິມັນຈະຢູ່ໃນໜ່ວຍປະມວນຜົນກາງ
  - ສຳເນົາຂໍ້ມູນທີ່ໃຊ້ເປັນປະຈຳໄປໄວ້ໃນ cache ເພື່ອໃຫ້ເຂົ້າຫາໄດ້ໄວ
  - ເຖິງຈະມີຂະໜາດນ້ອຍແຕ່ກໍ່ຊ່ວຍໃຫ້ປະສິດທິພາບຂອງລະບົບໄດ້ຫຼາຍ
    - ເນື່ອງຈາກວ່າຢູ່ໃກ້ກັບໜ່ວຍປະມວນຜົນ ແລະ ມີຄວາມໄວສູງ

ລຳດັບຊັ້ນຂອງໜ່ວຍຄວາມຈຳ



ຍຸດທະສາດໃນການຈັດການໜ່ວຍຄວາມຈຳຫຼັກ

- ມີຫລາຍປະເພດ
  - ຍຸດທະສາດໃນການເອົາຂໍ້ມູນເຂົ້າມາ (Fetch strategies)
    - ເປັນວິທີໃນການຕັດສິນໃຈວ່າແມ່ນຂໍ້ມູນໃດທີ່ຈະ load ໃນຄັ້ງຕໍ່ໄປ
    - ມີ 2 ວິທີ: Demand ຫຼື anticipatory
  - ຍຸດທະສາດໃນການວາງຂໍ້ມູນ (Placement strategies)
    - ເປັນວິທີກຳໜົດວ່າຈະວາງຂໍ້ມູນທີ່ load ເຂົ້າມາໄວ້ບ່ອນໃດໃນໜ່ວຍຄວາມຈຳ
    - ມີຫຼາຍວິທີ: Best Fit, First Fit, Worst Fit, Next Fit
  - ຍຸດທະສາດໃນການປ່ຽນແທນ (Replacement strategies)
    - ເປັນວິທີກຳໜົດວ່າຈະເອົາຂໍ້ມູນໃດອອກໄປ ເພື່ອເອົາຂໍ້ມູນໃໝ່ເຂົ້າມາ ໃນກໍລະນີໜ່ວຍຄວາມຈຳເຕັມ ຫຼື ຕ້ອງການໃຫ້ມີເນື້ອທີ່ຫວ່າງໃນໜ່ວຍຄວາມຈຳ

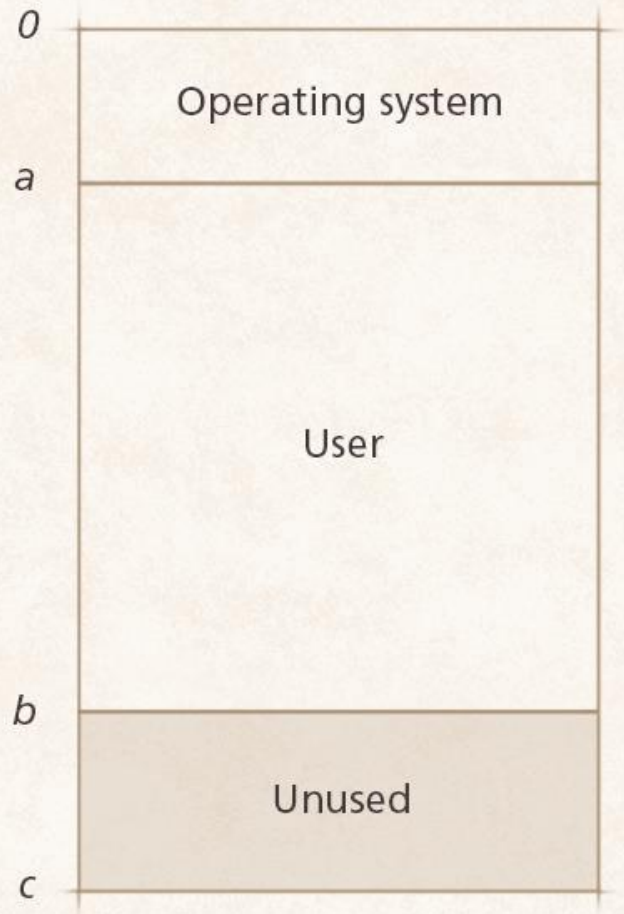
ການຈັດສັນໜ່ວຍຄວາມຈຳແບບຕໍ່ເນື່ອງ ແລະ ບໍ່ຕໍ່ເນື່ອງ

- ການຈັດສັນແບບຕໍ່ເນື່ອງ
  - ເປັນໃຫ້ໜ່ວຍຄວາມຈຳແກ່ໂປຣແກຣມໃດໜຶ່ງແບບຕໍ່ເນື່ອງໄປເປັນຊຸດດຽວ
  - ບາງຄັ້ງເປັນການຍາກທີ່ຈະຊອກຫາເຂດຫວ່າງຂອງໜ່ວຍຄວາມຈຳທີ່ພຽງໃຫ້
  - ເປັນວິທີທີ່ບໍ່ສິ້ນເປືອງ
- ການຈັດສັນແບບບໍ່ຕໍ່ເນື່ອງ
  - ແບ່ງໂປຣແກຣມອອກເປັນສ່ວນຍ່ອຍໆ ເອີ້ນວ່າ Segment
  - ແຕ່ລະສ່ວນສາມາດເກັບໄວ້ແຕ່ລະສ່ວນຂອງໜ່ວຍຄວາມຈຳ
  - ຊອກຫາສ່ວນຂອງໜ່ວຍຄວາມຈຳທີ່ຫວ່າງເພື່ອເກັບໄດ້ງ່າຍ
  - ເປັນວິທີທີ່ຂ້ອນຂ້າງສິ້ນເປືອງ

ການໃຫ້ຂະບວນການຄຽວໃຊ້ໜ່ວຍຄວາມຈຳທັງໝົດແບບຕໍ່ເນື່ອງ

- ແມ່ນການໃຫ້ຂະບວນການຄຽວຄວບຄຸມທັງໝົດລະບົບ
  - ຊັບພະຍາກອນບໍ່ຈຳເປັນຕ້ອງໄດ້ໃຊ້ຮ່ວມກັນ
  - ເບື້ອງຕົ້ນ ແມ່ນບໍ່ມີລະບົບປະຕິບັດການໃນຄອມພິວເຕີ
    - ນັກຂຽນໂປຣແກຣມຂຽນໂຄດເພື່ອບໍລິຫານຊັບພະຍາກອນທັງໝົດ
  - ລະບົບຄວບຄຸມ I/O (IOCS)
    - ມີຄັງຂອງໂຄດທີ່ໄດ້ສ້າງໄວ້ເພື່ອບໍລິຫານອຸປະກອນ I/O
    - ຖືກໃຊ້ກ່ອນມີລະບົບປະຕິບັດການ

ການໃຫ້ຂະບວນການຽວໃຊ້ໜ່ວຍຄວາມຈຳທັງໝົດແບບຕໍ່ເນື່ອງ



ການໃຫ້ຂະບວນການດຽວໃຊ້ໜ່ວຍຄວາມຈຳທັງໝົດແບບຕໍ່ເນື່ອງ

- ການຊ້ອນທັບກັນ (Overlay)
  - ເປັນວິທີຈັດສັນໜ່ວຍຄວາມຈຳໃຫ້ກັບໂປຣແກຣມທີ່ມີຂະໜາດໃຫຍ່ກ່ວາຂະໜາດໜ່ວຍຄວາມຈຳທີ່ມີ
  - ເປັນການເກັບຄຳສັ່ງແລະຂໍ້ມູນທີ່ຈຳເປັນຕ້ອງໃຊ້ວຽກໃນຂະນະນັ້ນໄວ້ໃນໜ່ວຍຄວາມຈຳ ສ່ວນທີ່ເຫຼືອເກັບໄວ້ບ່ອນອື່ນກ່ອນ ເມື່ອຈຳເປັນຕ້ອງໃຊ້ຈຶ່ງເອົາເຂົ້າມາຊ້ອນທັບໃສ່ໜ່ວຍຄວາມຈຳບ່ອນເກົ່າອີກ
  - ຜົນເສຍ
    - ຍາກໃນການຈັດການການຊ້ອນທັບໃຫ້ໃຊ້ໜ່ວຍຄວາມຈຳຫຼັກຢ່າງມີປະສິດທິພາບ
    - ຍຸ້ງຍາກໃນການດັດແປງໂປຣແກຣມ
  - ໃຊ້ໜ່ວຍຄວາມຈຳຈຳລອງ(Virtual memory) ເພື່ອແກ້ໄຂບັນຫາໄດ້ເຊັ່ນດຽວກັນ

ການໃຫ້ຂະບວນການຽກໃຊ້ໜ່ວຍຄວາມຈຳທັງໝົດແບບຕໍ່ເນື່ອງ

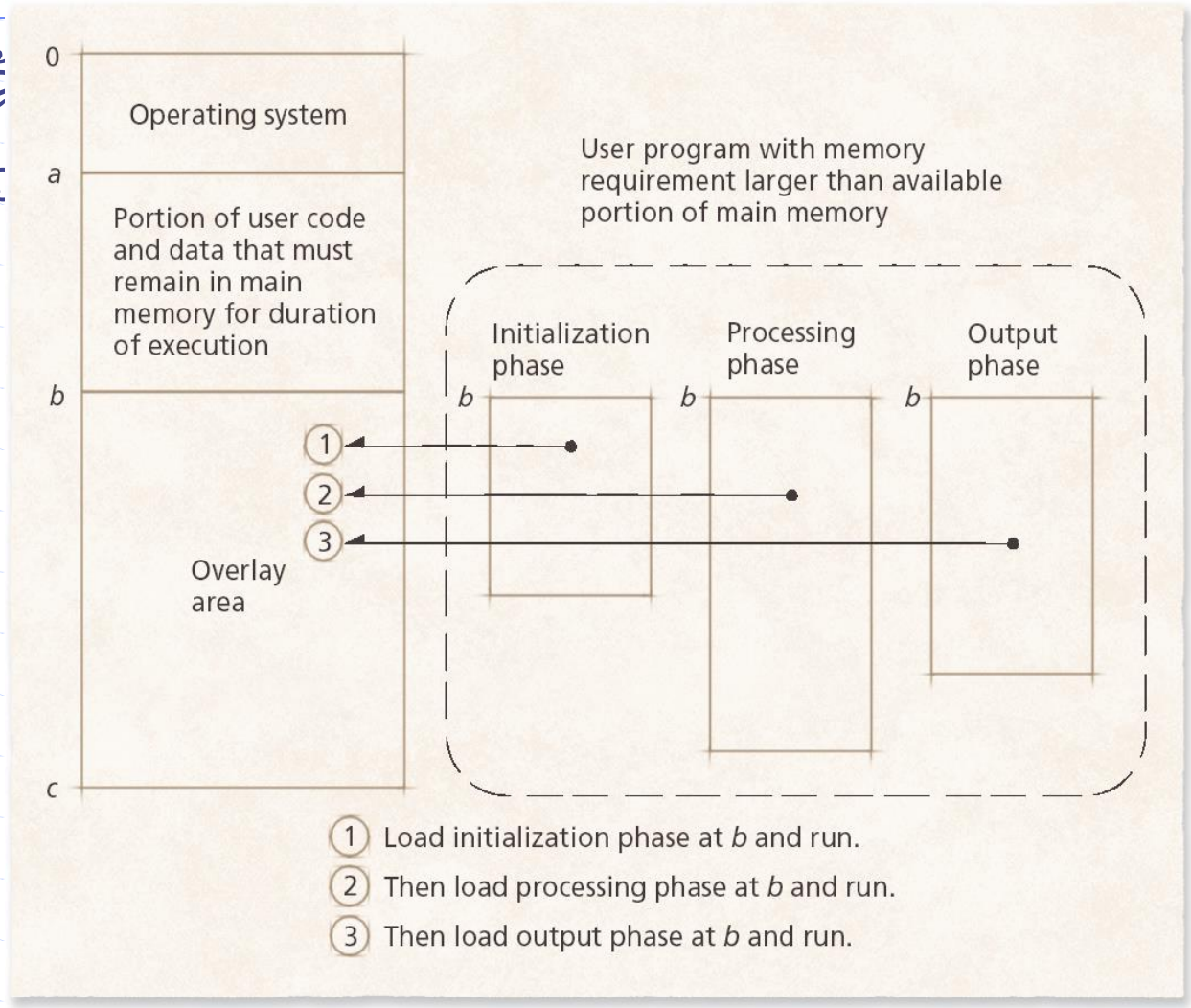
- ການຊ້ອນທັບກັນ (Overlay)
  - ຕົວຢ່າງ: ການແປພາສາ assembly ສົມມຸດວ່າ ມີການແປ 2 ຮອບ
  - ຮອບທຳອິດ 70 KB
  - ຮອບທີສອງ 80 KB
  - ຕາຕະລາງສັນຍາລັກ 20 KB
  - Routine ຮ່ວມກັນ 10 KB
  - ມີໜ່ວຍຄວາມຈຳຫລັກເຫຼືອ 150 KB

ການໃຫ້ຂະບວນການຽກໃຊ້ໜ່ວຍຄວາມຈຳທັງໝົດແບບຕໍ່ເນື່ອງ

- ການຊ້ອນທັບກັນ (Overlay)
  - ຕົວຢ່າງ: ວິທີການປະຕິບັດ
    - ແບ່ງສ່ວນທີ່ຊ້ອນທັບກັນເປັນ 2 ສ່ວນ
    - ສ່ວນທຳອິດປະກອບດ້ວຍຕາຕະລາງສັນຍາລັກ, routine ຮ່ວມ ແລະ ການແປ ຮອບທີ 1
    - ສ່ວນທີສອງປະກອບດ້ວຍຕາຕະລາງສັນຍາລັກ, routine ຮ່ວມ ແລະ ການແປ ຮອບທີ 2
    - ເພີ່ມ overlay driver ຂະໜາດ 10 KB ເຂົ້າໄປໃນສ່ວນທຳອິດ ແລ້ວເອົາໄປ ເກັບໄວ້ໃນໜ່ວຍຄວາມຈຳຫລັກ
    - ຫລັງຈາກແປຮອບທຳອິດແລ້ວ overlay driver ຈະ load ເອົາສ່ວນທີ່ສອງເຂົ້າ ມາແທນໃນເນື້ອທີ່ສ່ວນທຳອິດ ແລະສົ່ງການຄວບຄຸມໄປໃຫ້ການແປໃນຮອບທີ ສອງ

ການໃຫ້ຂະບວນການຽວໃຊ້ໜ່ວຍຄວາມຈຳທັງໝົດແບບຕໍ່ເນື່ອງ

- ການຂໍ
  - ຕິ

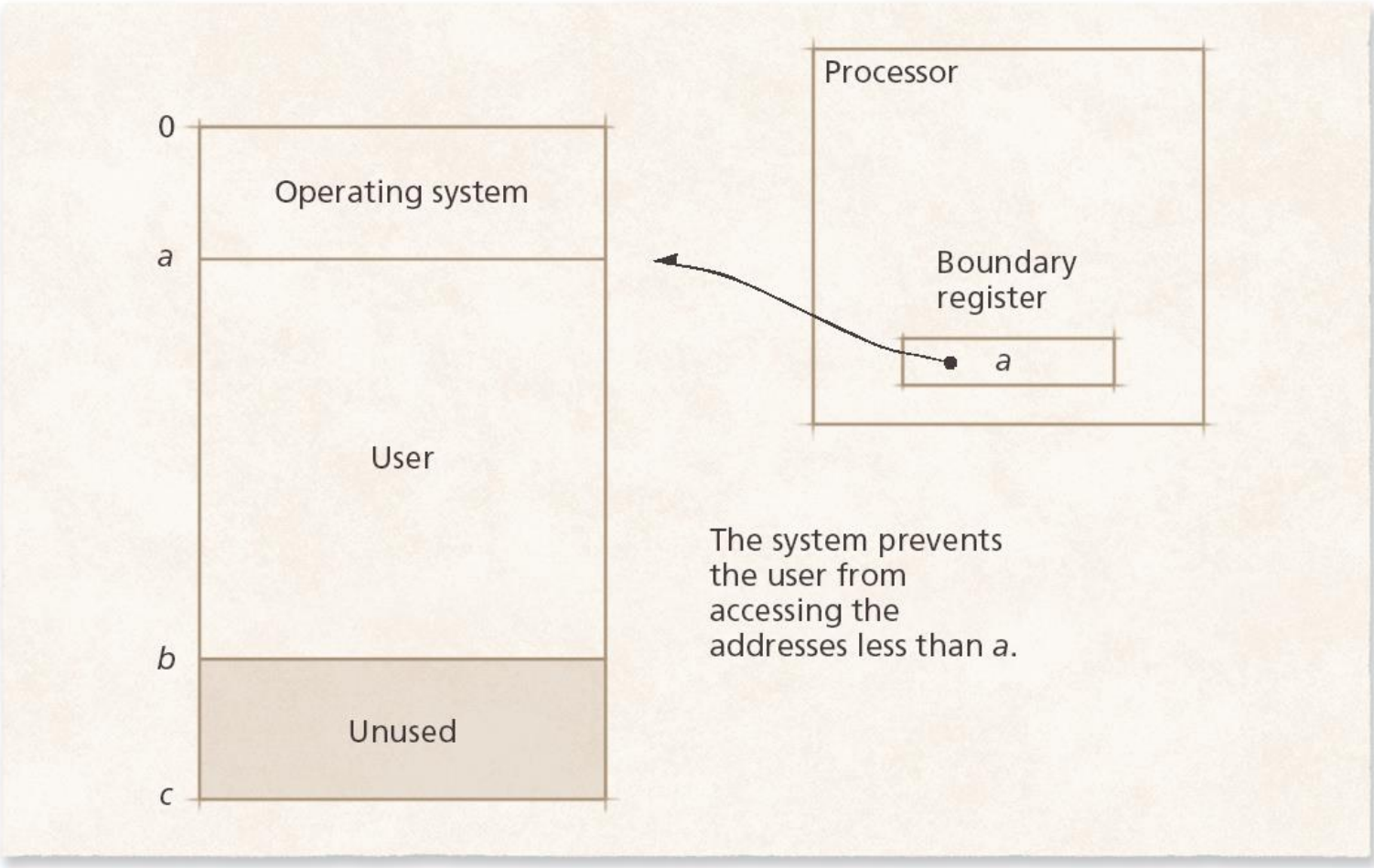


ການໃຫ້ຂະບວນການຽວໃຊ້ໜ່ວຍຄວາມຈຳທັງໝົດແບບຕໍ່ເນື່ອງ

- ການປ້ອງກັນໃນລະບົບທີ່ມີຂະບວນການຽວ
  - ລະບົບປະຕິບັດການຈະຕ້ອງບໍ່ຖືກເຮັດໃຫ້ເປ່ເພເສຍຫາຍໂດຍໂປຣແກຣມຂອງຜູ້ໃຊ້
  - ລະບົບຈະບໍ່ສາມາດເຮັດວຽກໄດ້ຖ້າລະບົບປະຕິບັດການຖືກຂໍ້ມູນອື່ນຂຽນທັບ
  - ປ້ອງກັນໂດຍໃຊ້ register ຂອບເຂດ (Boundary register)
    - ບັນຈຸທີ່ຢູ່ເລີ່ມຕົ້ນຂອງໜ່ວຍຄວາມຈຳທີ່ໂປຣແກຣມໃຊ້
    - ບໍ່ອະນຸຍາດໃຫ້ໂປຣແກຣມໄປໃຊ້ໜ່ວຍຄວາມຈຳທີ່ຢູ່ນອກຂອບເຂດ
    - ສາມາດກຳໜົດຄ່າໂດຍໃຊ້ຄຳສັ່ງສິດທິພິເສດເທົ່ານັ້ນ
    - ໂປຣແກຣມສາມາດເຂົ້າໄປໃຊ້ໜ່ວຍຄວາມຈຳຂອງລະບົບປະຕິບັດການເພື່ອໃຊ້ຂະບວນການໂດຍໃຊ້ system calls

ການໃຫ້ຂະບວນການຄຽວໃຊ້ໜ່ວຍຄວາມຈຳທັງໝົດແບບຕໍ່ເນື່ອງ

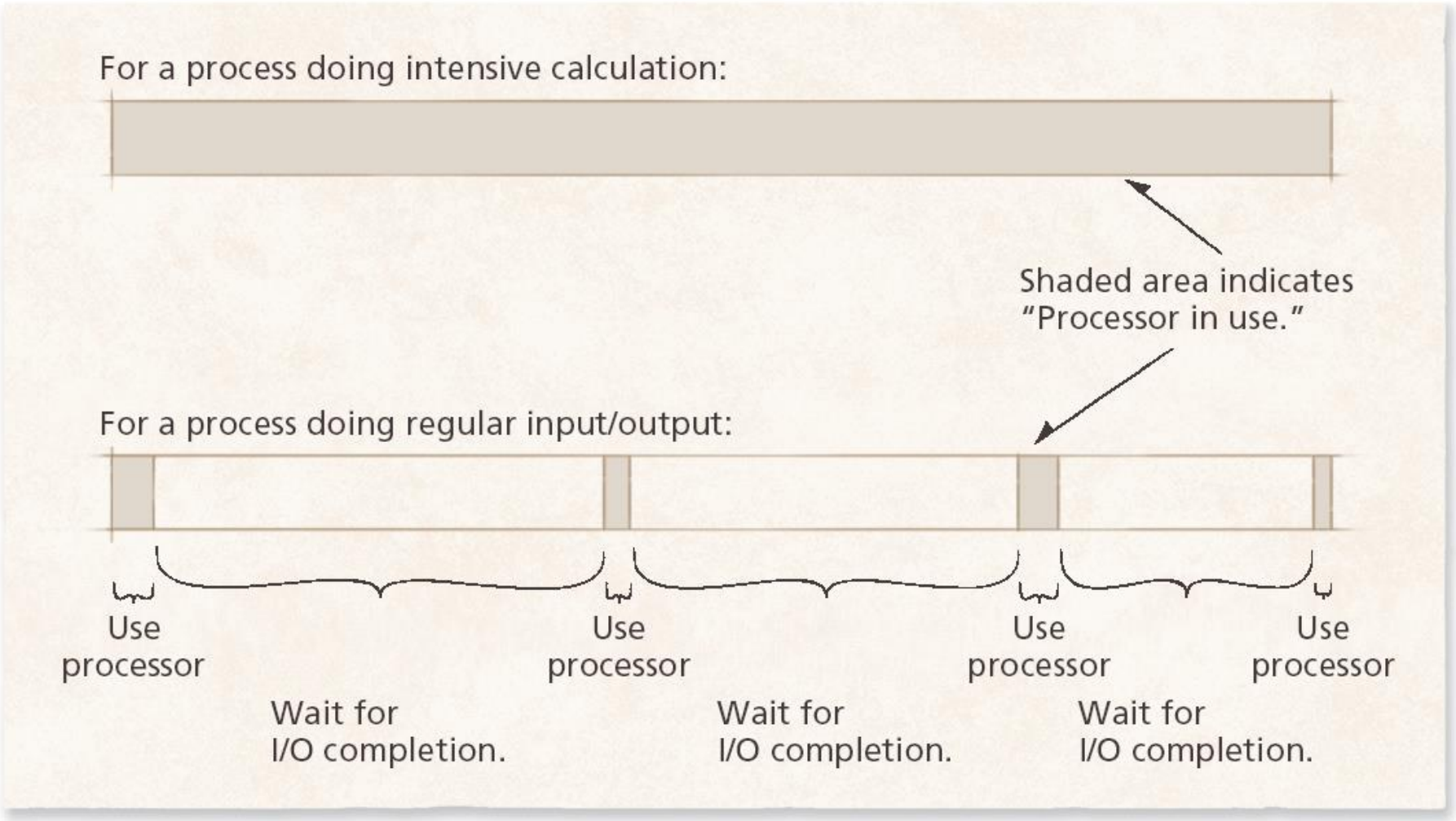
- ການປ້ອງກັນໃນລະບົບທີ່ມີຂະບວນການຄຽວ



ການແບ່ງໜ່ວຍຄວາມຈຳໃຫ້ຫຼາຍຂະບວນການທີ່ມີຂະໜາດບໍ່ປ່ຽນແປງ

- ການຂໍໃຊ້ງານ I/O ເຮັດໃຫ້ເສຍເວລາ CPU
- ລະບົບ Muntiprogramming ເຮັດໃຫ້ການໃຊ້ງານ CPU ເຕັມປະສິທິພາບ
  - ຂະບວນການທີ່ໃຊ້ I/O ຈະຕ້ອງສະລະ CPU ໃຫ້ຂະບວນການອື່ນ
  - ສາມາດໃຫ້ຫລາຍຂະບວນການເຂົ້າໃຊ້ໜ່ວຍຄວາມຈຳ ແລະ ເຮັດວຽກພ້ອມກັນໄດ້

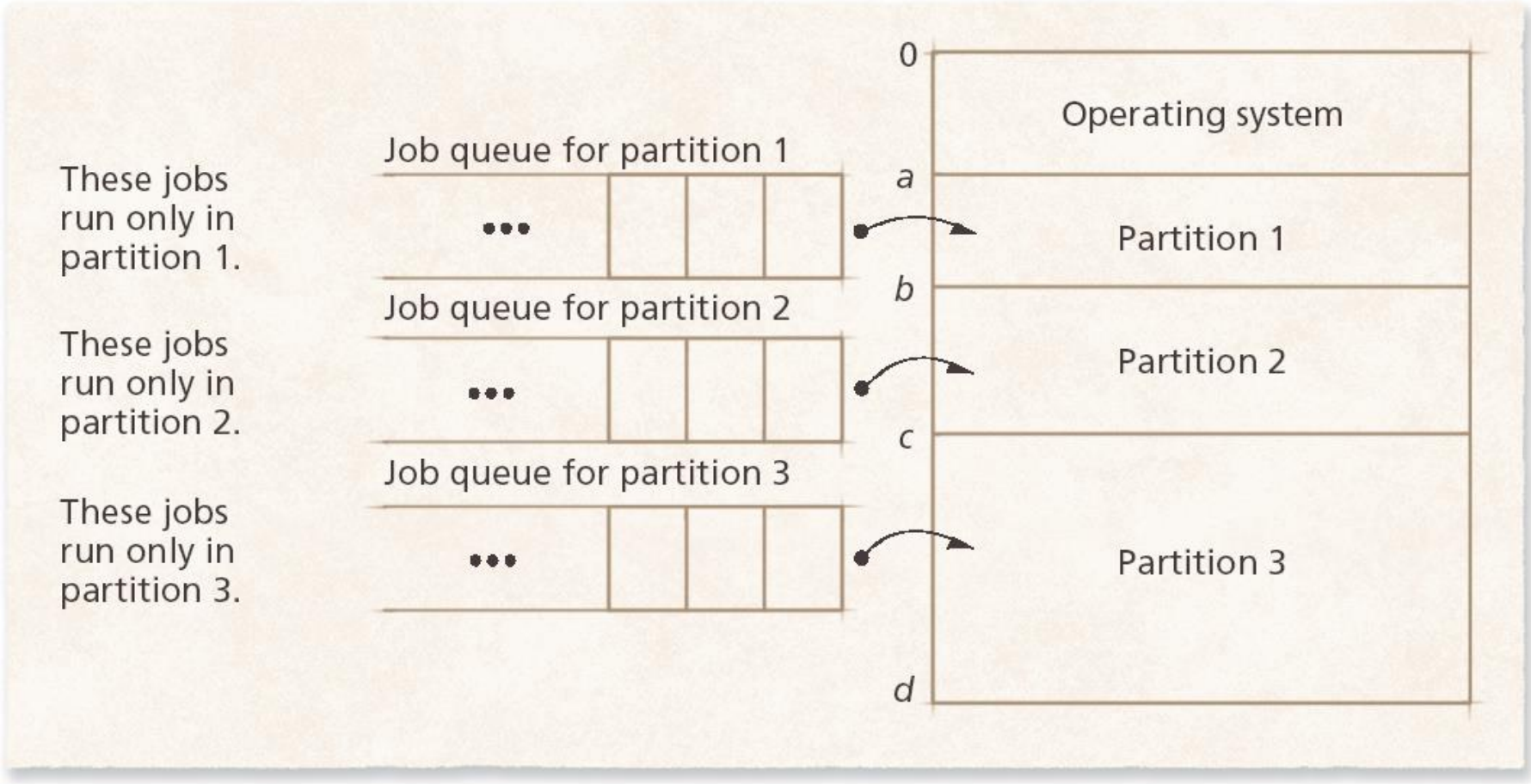
ການແບ່ງໜ່ວຍຄວາມຈຳໃຫ້ຫຼາຍຂະບວນການທີ່ມີຂະໜາດບໍ່ປ່ຽນແປງ



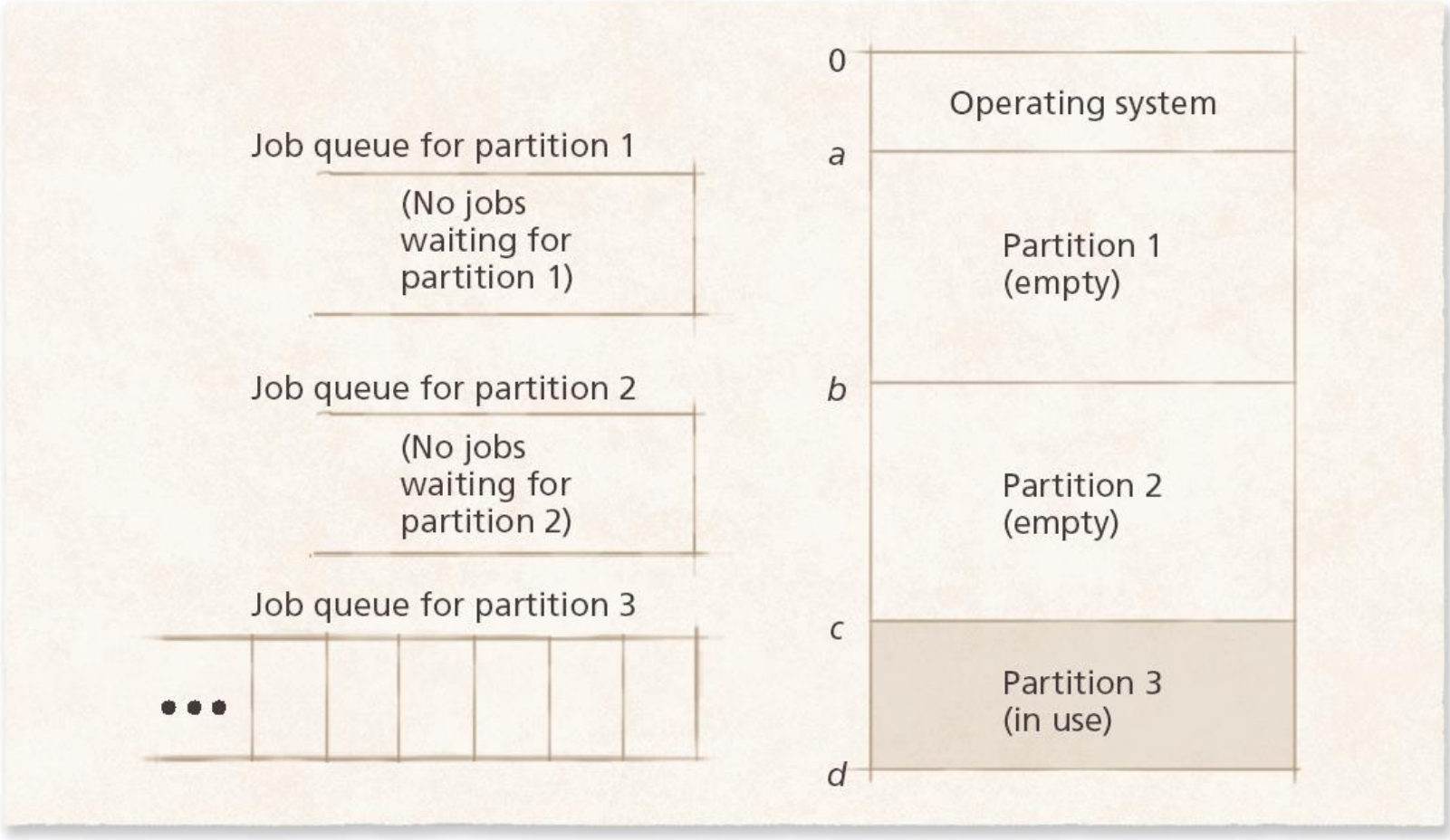
ການແບ່ງໜ່ວຍຄວາມຈຳໃຫ້ຫຼາຍຂະບວນການທີ່ມີຂະໜາດບໍ່ປ່ຽນແປງ

- ເປັນການແບ່ງໜ່ວຍຄວາມຈຳອອກເປັນຫລາຍສ່ວນຍ່ອຍທີ່ມີຂະໜາດຄົງທີ່
- ແຕ່ລະສ່ວນຍ່ອຍຈະເກັບຂະບວນການໄດ້ພຽງ 1 ຂະບວນການ
- ເມື່ອຂະບວນການເຮັດວຽກແລ້ວ ມັນສົ່ງຄືນໜ່ວຍຄວາມຈຳທີ່ມັນໄດ້ຮັບໃຫ້ກັບລະບົບເພື່ອຈັດສັນໃຫ້ຂະບວນການອື່ນ
- ວິທີດັ່ງກ່າວມີຜົນເສຍດັ່ງນີ້
  - ໃນການໃຊ້ທີ່ຢູ່ແບບຕາຍຕົວແບບເກົ່າ ຖ້າວ່າສ່ວນຍ່ອຍດັ່ງກ່າວເຕັມມັນຈະບໍ່ສາມາດເອົາ code ເຂົ້າມາໄດ້ຕື່ມອີກ
  - ມີໜ່ວຍຄວາມຈຳບາງສ່ວນເຫຼືອຢູ່ພາຍໃນ ແຕ່ບໍ່ສາມາດໃຊ້ໄດ້
  - ສິ້ນເປືອງຊັບພະຍາກອນ

ການແບ່ງໜ່ວຍຄວາມຈຳໃຫ້ຫຼາຍຂະບວນການທີ່ມີຂະໜາດບໍ່ປ່ຽນແປງ

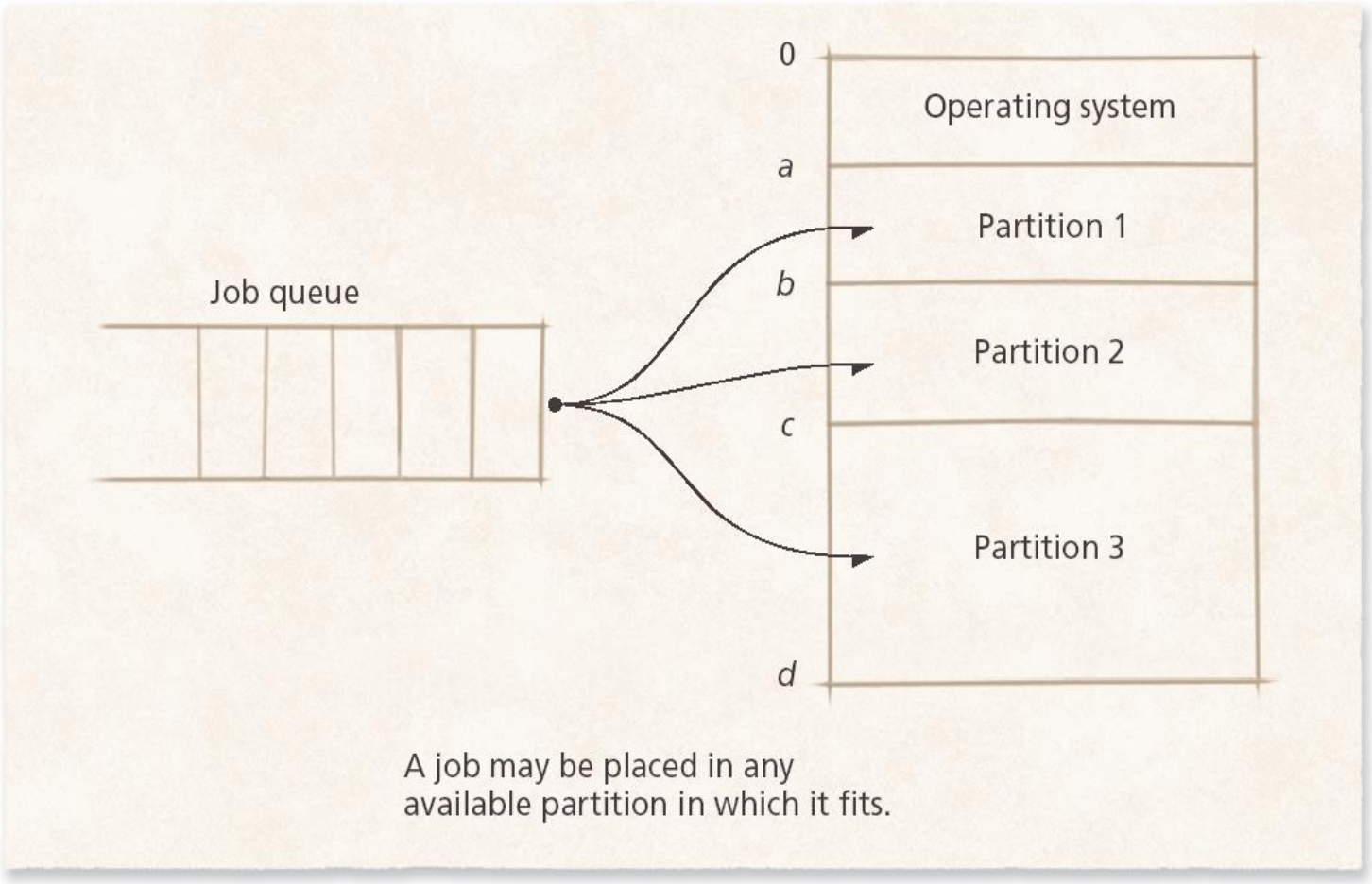


ການແບ່ງໜ່ວຍຄວາມຈຳໃຫ້ຫຼາຍຂະບວນການທີ່ມີຂະໜາດບໍ່ປ່ຽນແປງ



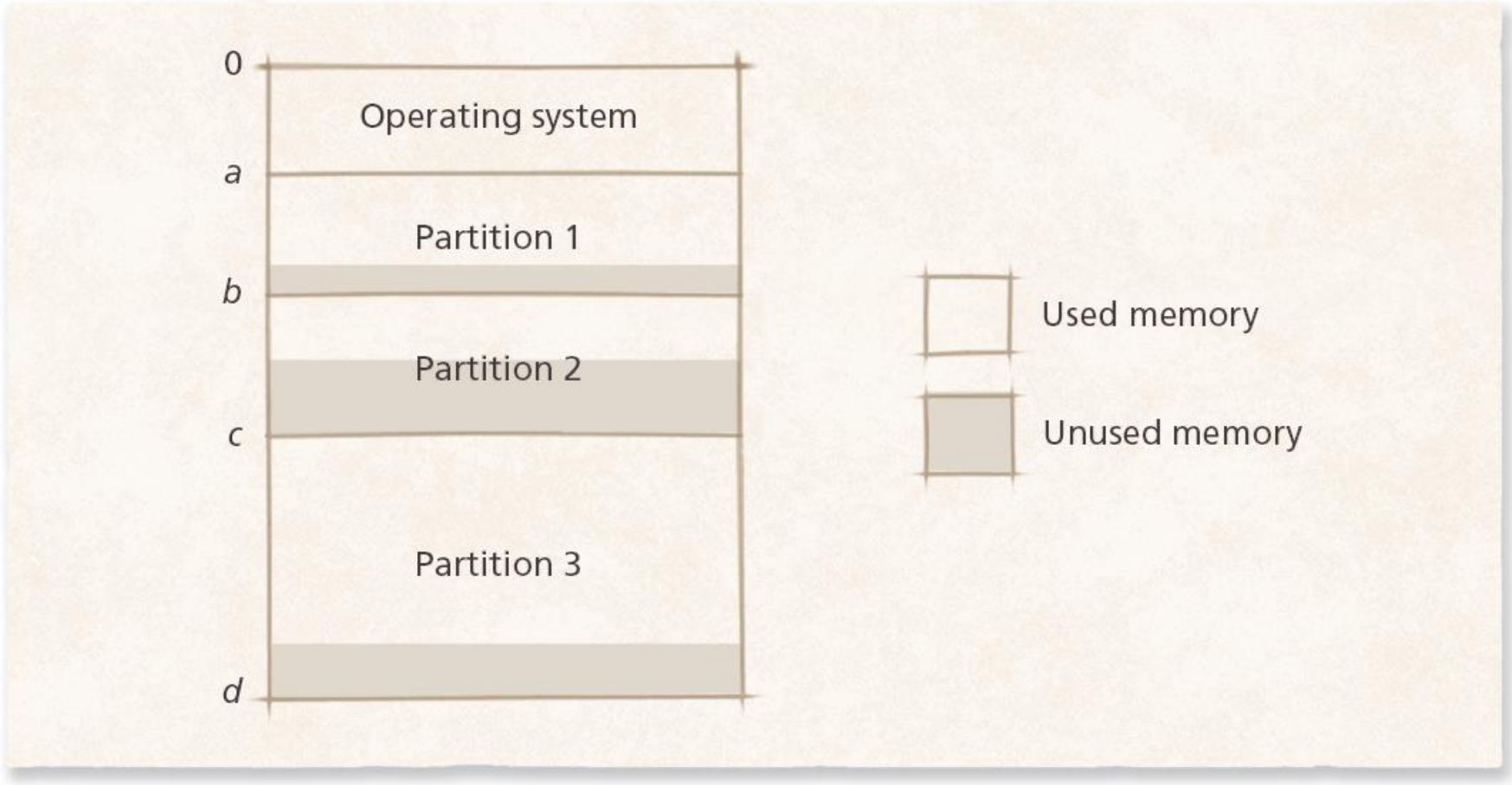
ຜົນເສຍຈາກການກຳໜົດທີ່ຢູ່ແບບຕາຍຕົວ

ການແບ່ງໜ່ວຍຄວາມຈຳໃຫ້ຫຼາຍຂະບວນການທີ່ມີຂະໜາດບໍ່ປ່ຽນແປງ



ແກ້ໄຂບັນຫາດ້ວຍຫລັກການ ປັບທີ່ຢູ່ ແລະ ໂລດຄືນໃໝ່

ການແບ່ງໜ່ວຍຄວາມຈຳໃຫ້ຫຼາຍຂະບວນການທີ່ມີຂະໜາດບໍ່ປ່ຽນແປງ



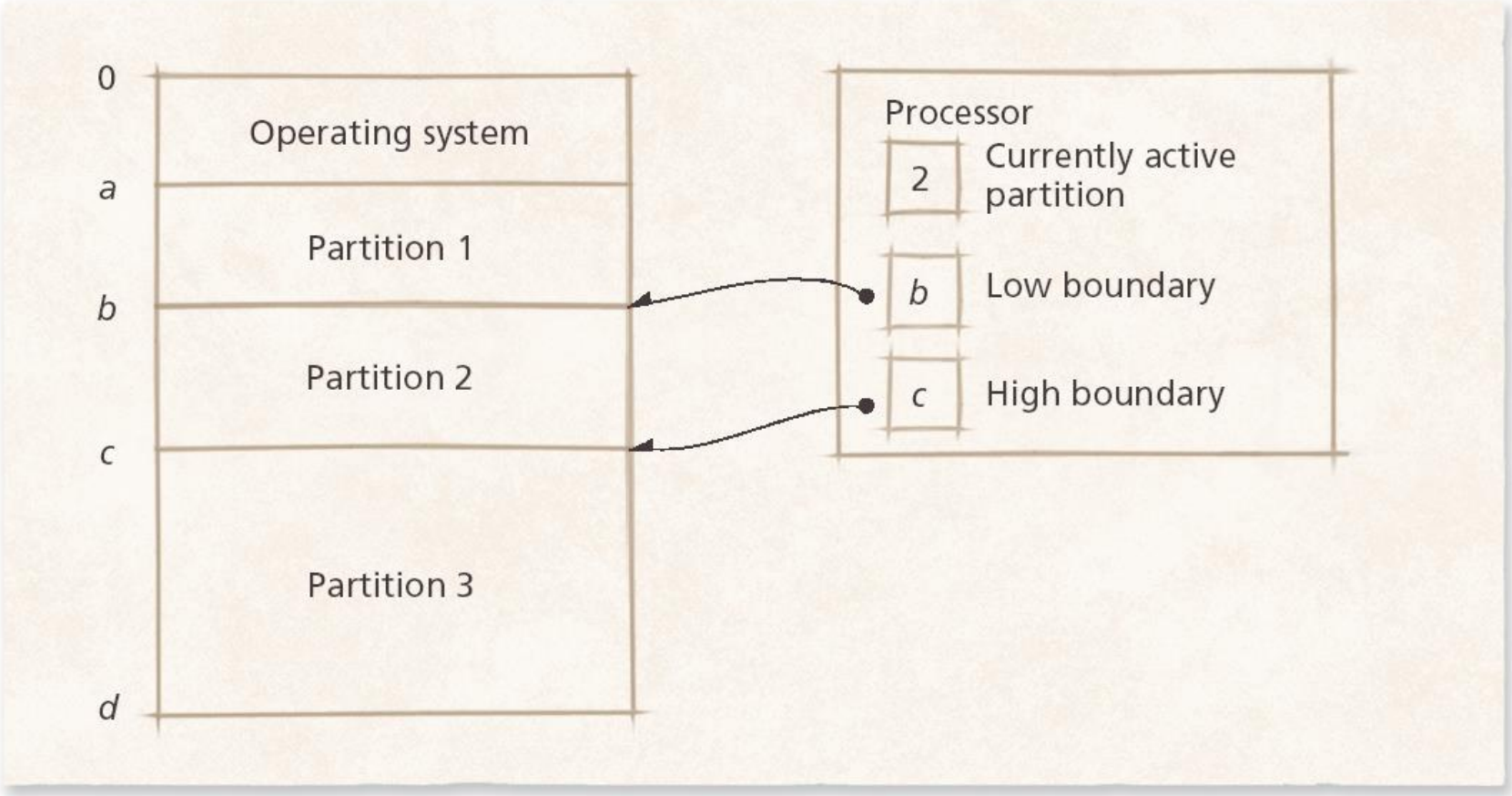
ຜົນເສຍຈາກການມີເນື້ອທີ່ຫວ່າງຢູ່ພາຍໃນສ່ວນຍ່ອຍ

ການແບ່ງໜ່ວຍຄວາມຈຳໃຫ້ຫຼາຍຂະບວນການທີ່ມີຂະໜາດບໍ່ປ່ຽນແປງ

- ການປ້ອງກັນໜ່ວຍຄວາມຈຳ
  - ສາມາດເຮັດໄດ້ໂດຍການໃຊ້ registers ຂອບເຂດທີ່ຊື່ວ່າ base ແລະ limit (ບາງຄັ້ງເອີ້ນວ່າ low ແລະ high)
  - ເປັນການປ້ອງກັນຂະບວນການຕ່າງໆບໍ່ໃຫ້ໃຊ້ໜ່ວຍຄວາມຈຳນອກເຂດທີ່ຈັດສັນໃຫ້ ໂດຍໃຊ້ register ເລີ່ມຕົ້ນ ແລະ register ສຸດທ້າຍ
  - ທີ່ຢູ່ທາງຕັກກະຈະຕ້ອງມີຄ່ານ້ອຍກ່ວາຄ່າໃນ register ສຸດທ້າຍກ່ອນ ຈະເອົາໄປບວກກັບຄ່າ register ເລີ່ມຕົ້ນເພື່ອຊອກຫາຄ່າທີ່ຢູ່ຕົວຈິງ

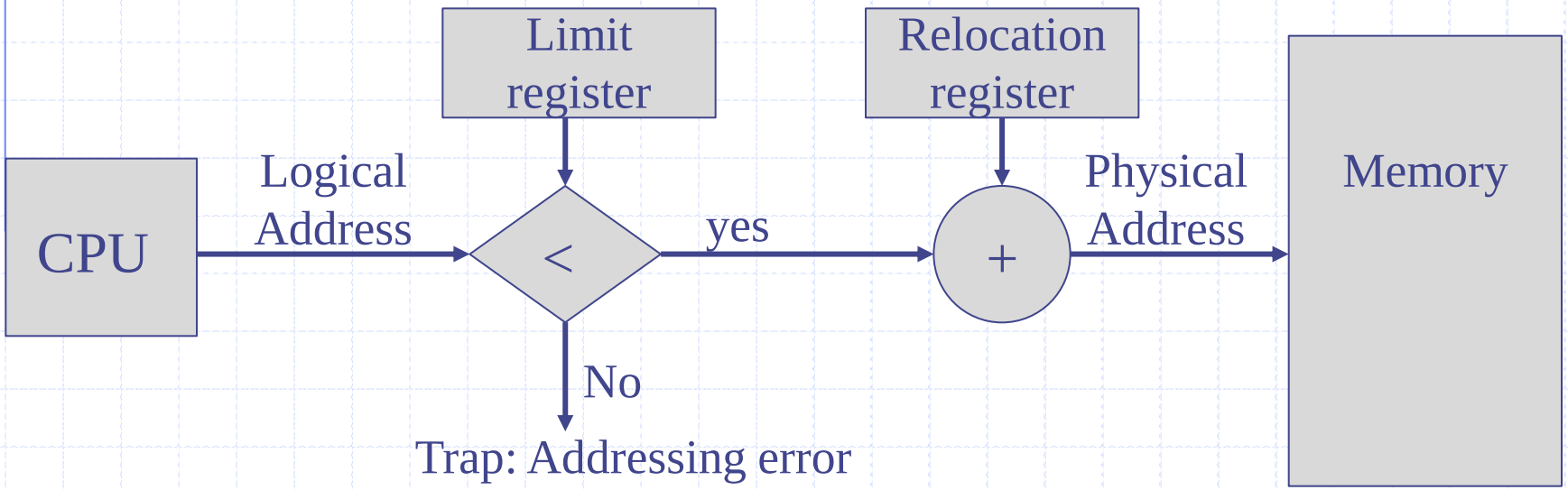
ການແບ່ງໜ່ວຍຄວາມຈຳໃຫ້ຫຼາຍຂະບວນການທີ່ມີຂະໜາດບໍ່ປ່ຽນແປງ

- ການປ້ອງກັນໜ່ວຍຄວາມຈຳ



ການແບ່ງໜ່ວຍຄວາມຈຳໃຫ້ຫຼາຍຂະບວນການທີ່ມີຂະໜາດບໍ່ປ່ຽນແປງ

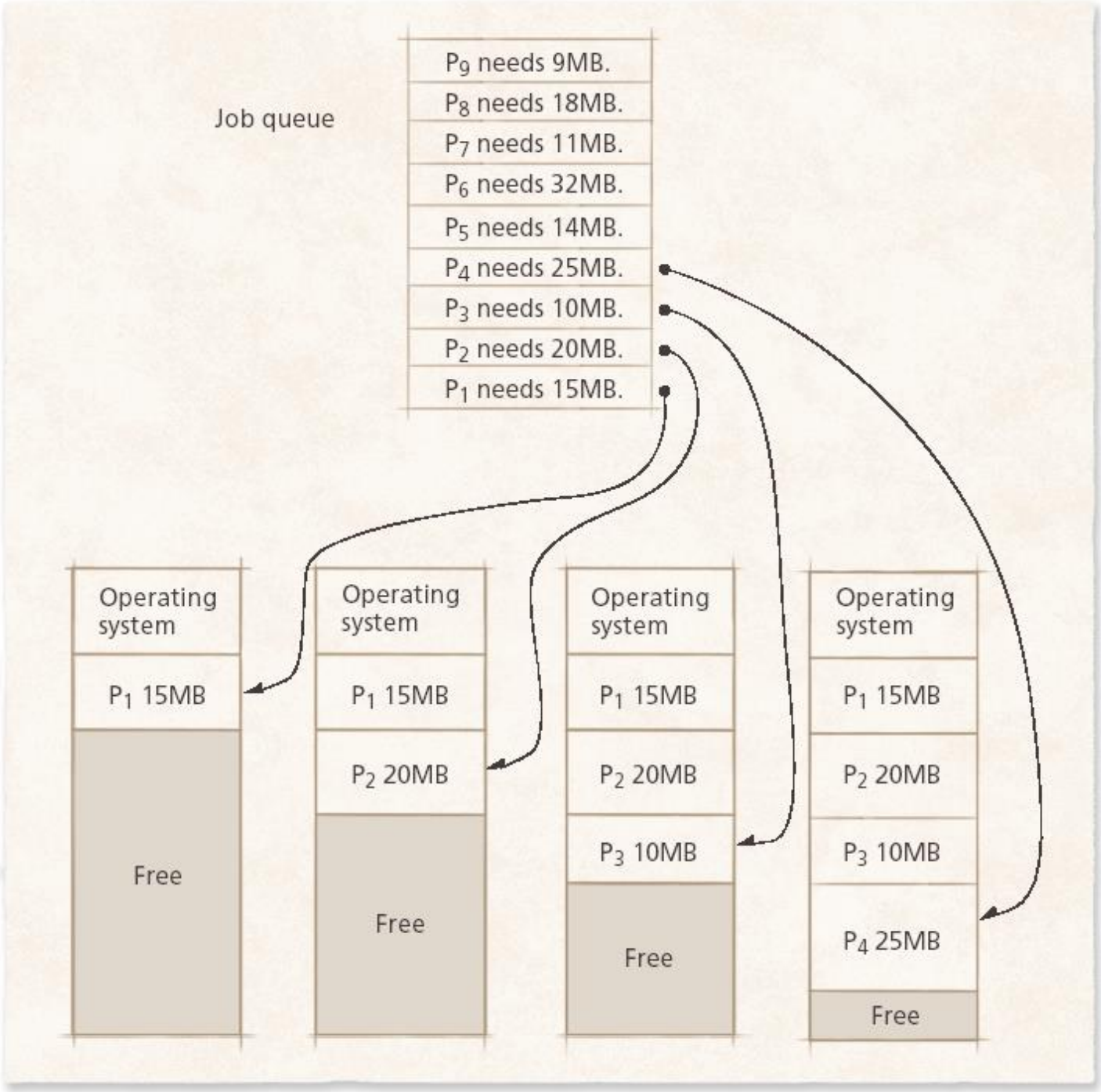
- ການປ້ອງກັນໜ່ວຍຄວາມຈຳ



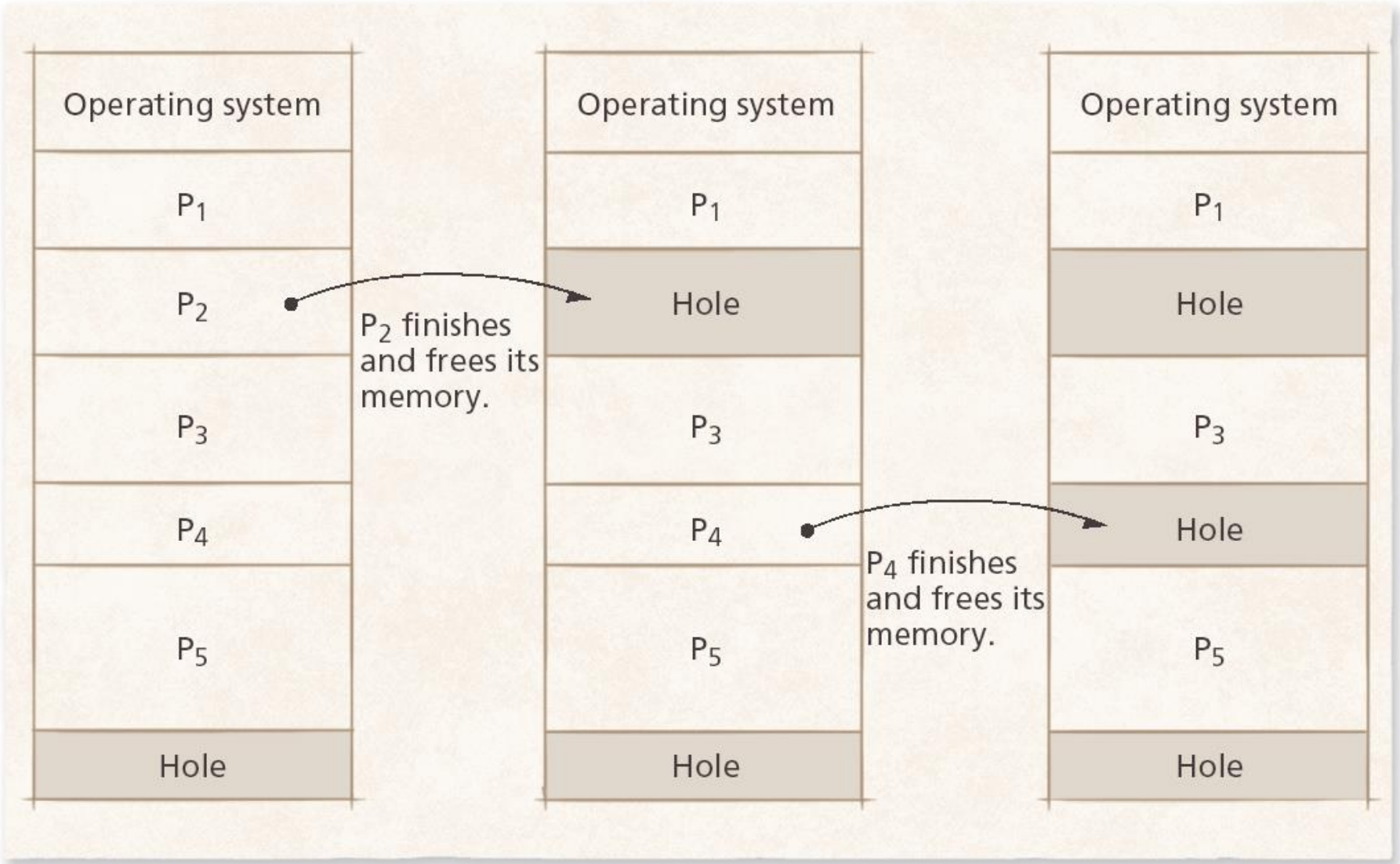
ການແບ່ງໜ່ວຍຄວາມຈຳໃຫ້ຫຼາຍຂະບວນການທີ່ມີຂະໜາດປ່ຽນແປງ

- ເປັນການແບ່ງໜ່ວຍຄວາມຈຳອອກເປັນຫລາຍສ່ວນຍ່ອຍທີ່ມີຂະໜາດບໍ່ຄົງທີ່ ທີ່ມີຄຸນລັກສະນະດັ່ງນີ້
  - ຂະບວນການຈະຖືກເອົາໄປເກັບໄວ້ໃນສ່ວນເໝາະສົມພໍດີກັບມັນ
  - ບໍ່ມີການສູນເສຍເນື້ອທີ່ແບບບໍ່ມີປະໂຫຍດໃນຕອນຕົ້ນ
  - ບໍ່ເກີດເນື້ອທີ່ຫວ່າງຢູ່ທາງໃນ (Internal Fragmentation) ເພາະວ່າ ຂະບວນການຈະຖືກເອົາໄປໄວ້ໃນສ່ວນທີ່ພໍດີກັບມັນ
  - ອາດຈະເຮັດໃຫ້ເກີດເນື້ອທີ່ຫວ່າງຢູ່ງທາງນອກ (External Fragmentation) ເພາະວ່າ ອາດຈະເຮັດໃຫ້ມີຊ່ອງວ່າງທີ່ບໍ່ພໍເກັບຂະບວນການໃໝ່ເຫຼືອຢູ່

ภา
ขม


Job queue
P9 needs 9MB.
P8 needs 18MB.
P7 needs 11MB.
P6 needs 32MB.
P5 needs 14MB.
P4 needs 25MB.
P3 needs 10MB.
P2 needs 20MB.
P1 needs 15MB.
Operating system
P1 15MB
Free
Operating system
P1 15MB
P2 20MB
Free
Operating system
P1 15MB
P2 20MB
P3 10MB
Free
Operating system
P1 15MB
P2 20MB
P3 10MB
P4 25MB
Free

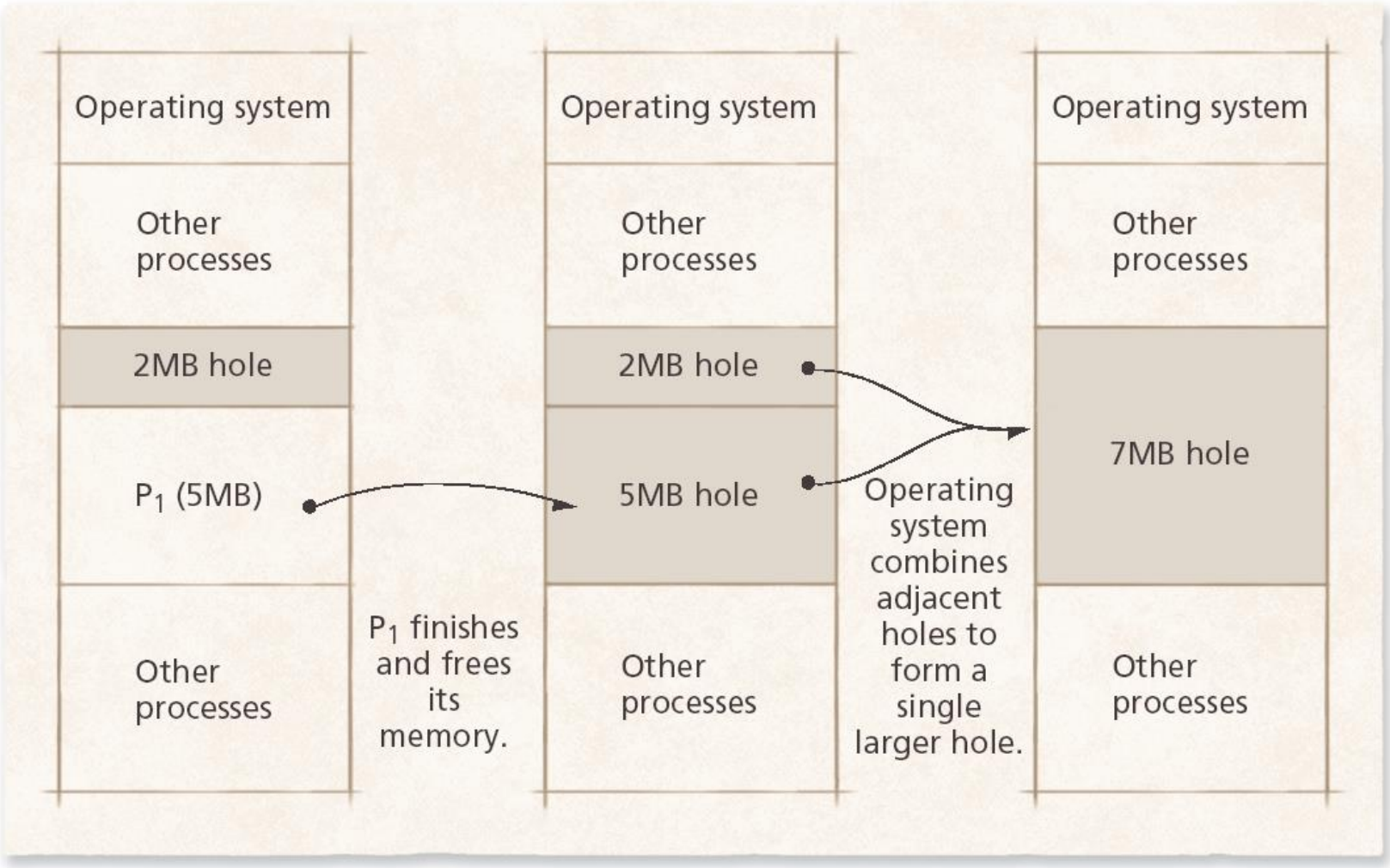
ການແບ່ງໜ່ວຍຄວາມຈຳໃຫ້ຫຼາຍຂະບວນການທີ່ມີຂະໜາດປ່ຽນແປງ



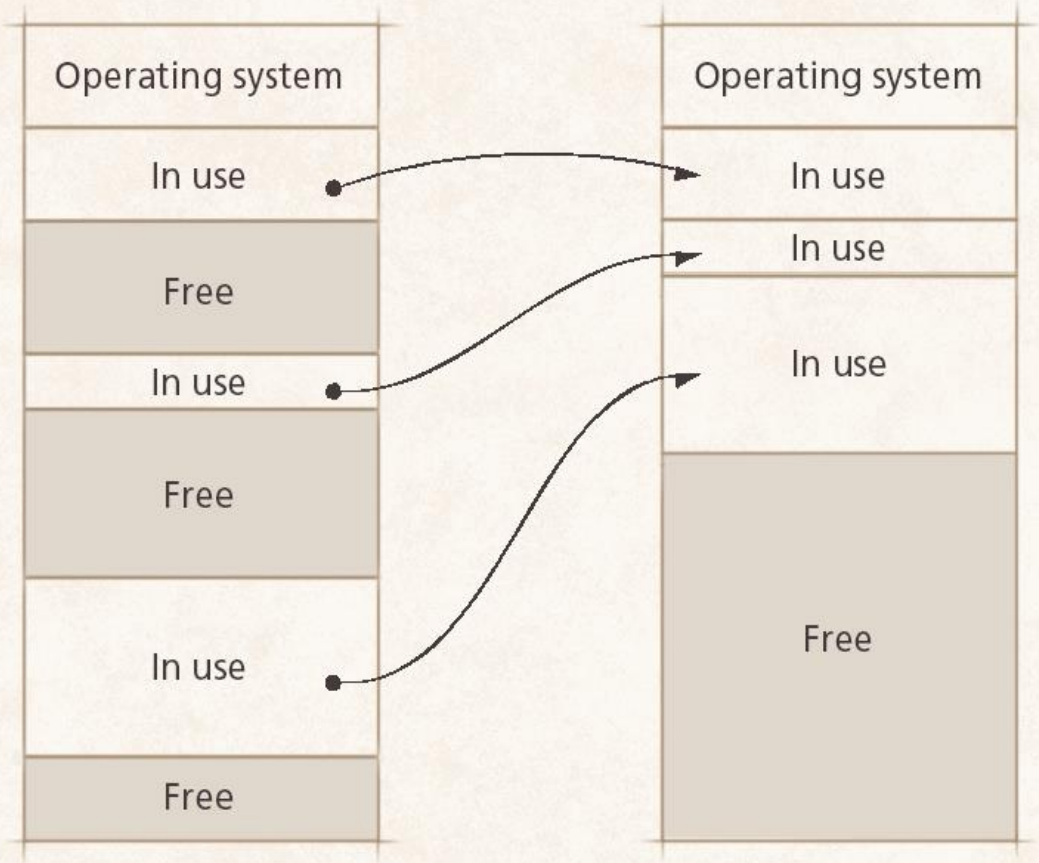
ການແບ່ງໜ່ວຍຄວາມຈຳໃຫ້ຫຼາຍຂະບວນການທີ່ມີຂະໜາດປ່ຽນແປງ

- ວິທີຈັດການກັບ External Fragmentation
  - ລວມເນື້ອທີ່ຫວ່າງທີ່ຢູ່ຕິດກັນເຂົ້າດ້ວຍກັນ (Coalescing)
    - ສ່ວນຫຼາຍແມ່ນຊອກຫາເນື້ອທີ່ຕິດກັນໄດ້ຍາກ ແລະ ເມື່ອລວມເຂົ້າກັນແລ້ວກໍ່ຍັງບໍ່ພຽງພໍ
  - ເອົາເນື້ອທີ່ໃຊ້ງານໄປໄວ້ເຂດໜຶ່ງ ແລະ ເນື້ອທີ່ທີ່ຍັງຫວ່າງໄວ້ເຂັດໜຶ່ງທີ່ຕໍ່ເນື່ອງກັນ (Compaction)
    - ບາງຄັ້ງເອີ້ນວ່າ garbage collection (ບໍ່ແມ່ນ garbage collection ໃນ object-oriented languages)
    - ລວມເອົາເນື້ອທີ່ຫວ່າງທັງໝົດມາໄວ້ບ່ອນດຽວກັນ
    - ວິທີດັ່ງກ່າວແມ່ນສິ້ນເປືອງຊັບພະກອນຫລາຍ

ການແບ່ງໜ່ວຍຄວາມຈຳໃຫ້ຫຼາຍຂະບວນການທີ່ມີຂະໜາດປ່ຽນແປງ



ການແບ່ງໜ່ວຍຄວາມຈຳໃຫ້ຫຼາຍຂະບວນການທີ່ມີຂະໜາດປ່ຽນແປງ



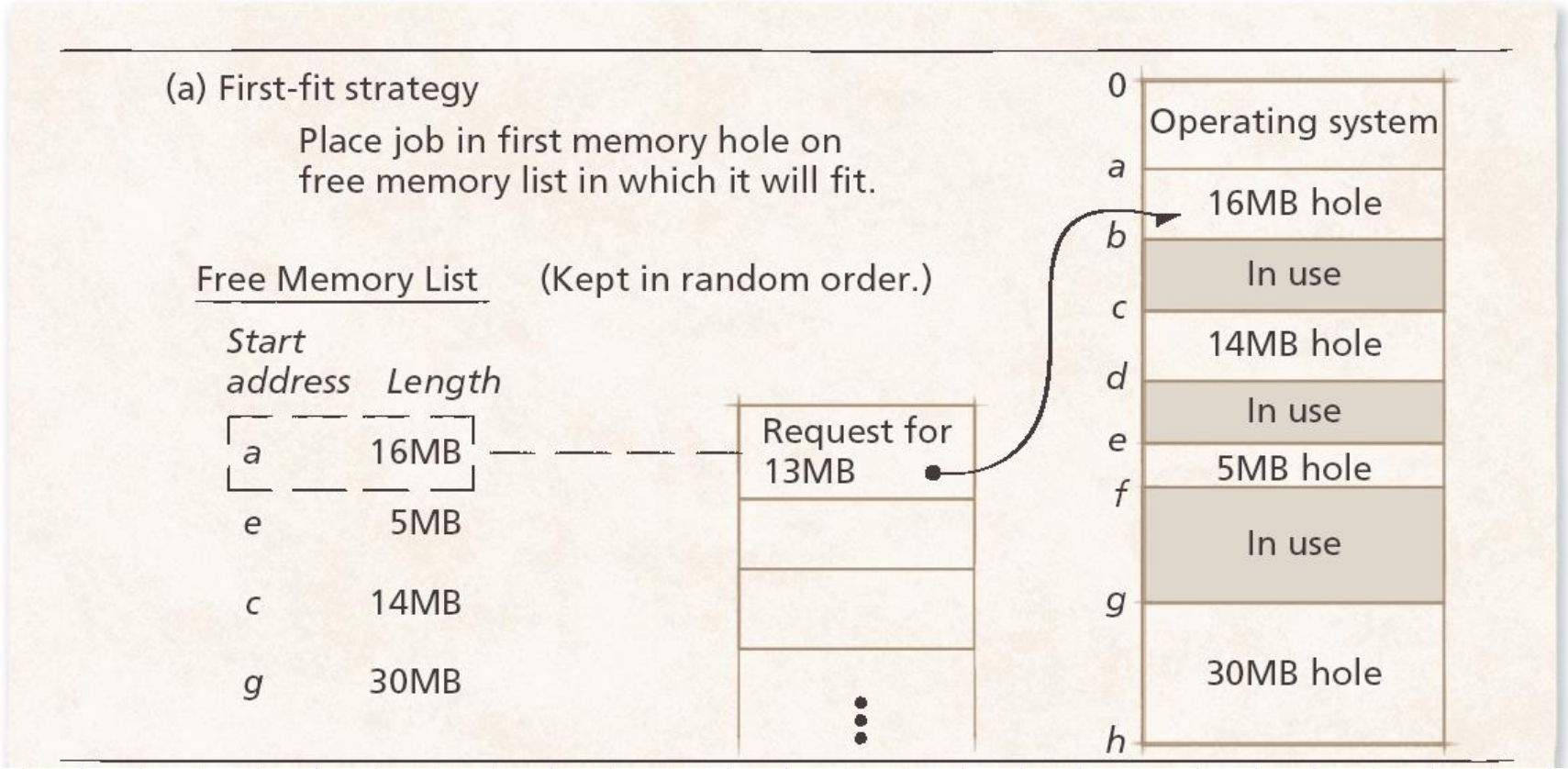
Operating system places all "in use" blocks together leaving free memory as a single large hole.

ການແບ່ງໜ່ວຍຄວາມຈຳໃຫ້ຫຼາຍຂະບວນການທີ່ມີຂະໜາດປ່ຽນແປງ

- ຍຸດທະສາດໃນການວາງຂໍ້ມູນໃນໜ່ວຍຄວາມຈຳ
  - First-fit strategy
    - ຂະບວນການຈະຖືກເອົາລົງເກັບໄວ້ໃນຊ່ອງທຳອິດທີ່ພົບທີ່ມີຂະໜາດທີ່ສາມາດເກັບໄດ້
    - ງ່າຍ ແລະ ບໍ່ເປືອງເວລາໃນການປະຕິບັດການ
  - Best-fit strategy
    - ຂະບວນການຈະຖືກເອົາລົງເກັບໄວ້ໃນຊ່ອງທີ່ພໍດີກັບຂໍ້ມູນຊຶ່ງມີເນື້ອທີ່ຫວ່າງເຫລືອຢູ່ໜ້ອຍທີ່ສຸດ
    - ຂ້ອນຂ້າງສິ້ນເປືອງເວລາໃນການປະຕິບັດການ
  - Worst-fit strategy
    - ຂະບວນການຈະຖືກເອົາລົງເກັບໄວ້ໃນຊ່ອງທີ່ມີເນື້ອທີ່ຫວ່າງຢູ່ຫລາຍທີ່ສຸດ
    - ເຮັດໃຫ້ມີເນື້ອທີ່ຫວ່າງເຫລືອໄວ້ທີ່ອາດໃຊ້ເກັບຂະບວນການອື່ນໄດ້

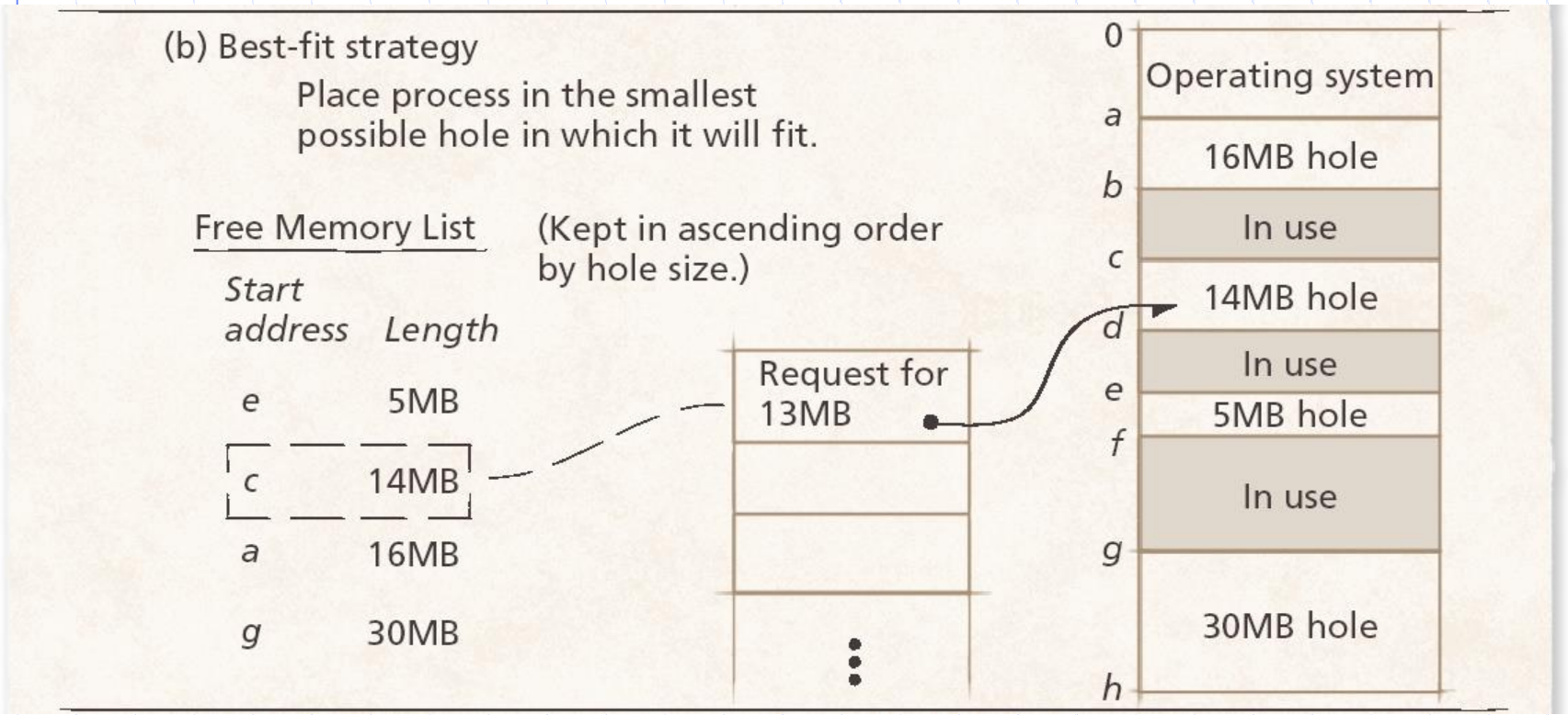
ການແບ່ງໜ່ວຍຄວາມຈຳໃຫ້ຫຼາຍຂະບວນການທີ່ມີຂະໜາດປ່ຽນແປງ

- ຍຸດທະສາດໃນການວາງຂໍ້ມູນໃນໜ່ວຍຄວາມຈຳ



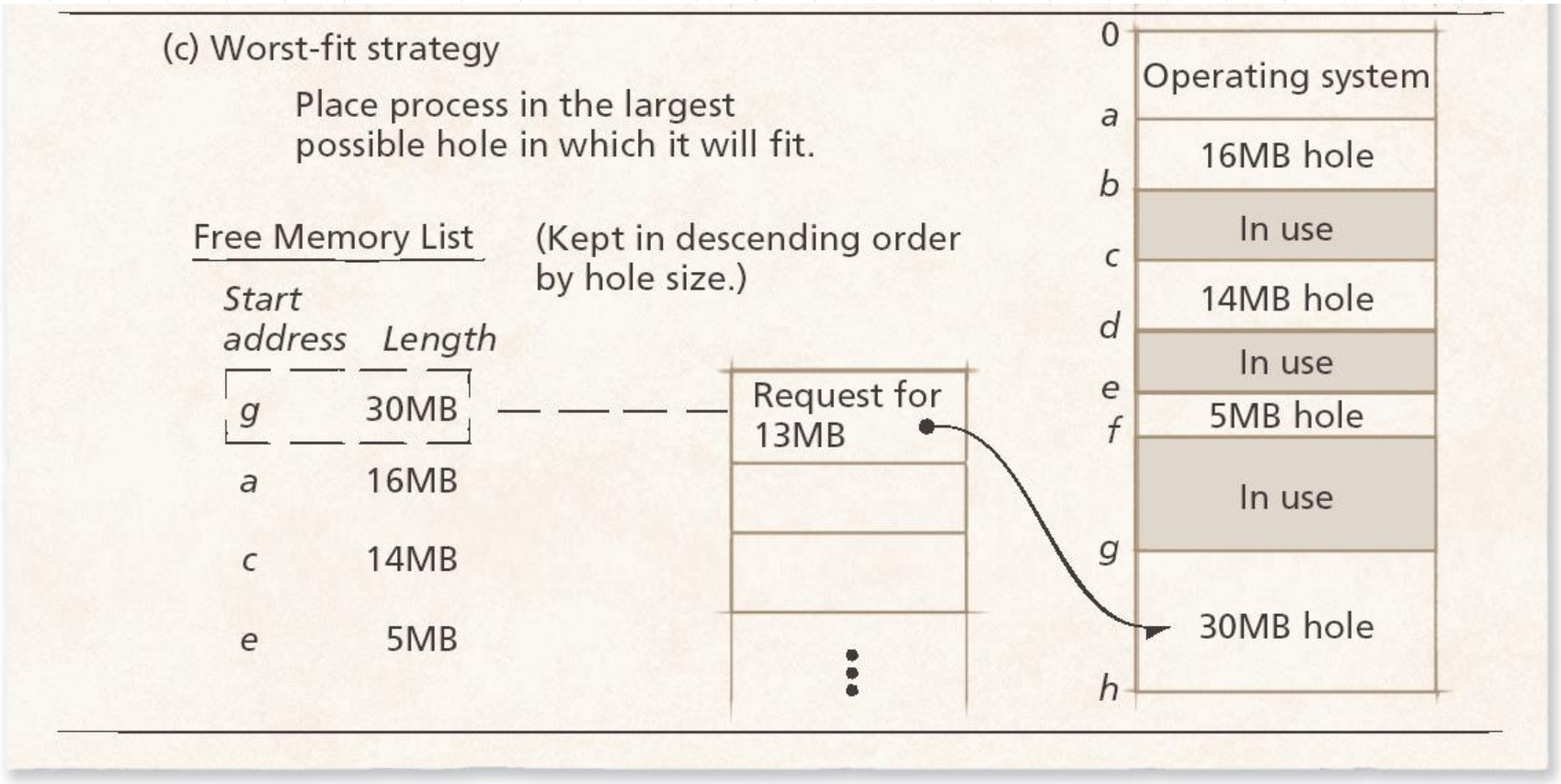
ການແບ່ງໜ່ວຍຄວາມຈຳໃຫ້ຫຼາຍຂະບວນການທີ່ມີຂະໜາດປ່ຽນແປງ

- ຍຸດທະສາດໃນການວາງຂໍ້ມູນໃນໜ່ວຍຄວາມຈຳ



ການແບ່ງໜ່ວຍຄວາມຈຳໃຫ້ຫຼາຍຂະບວນການທີ່ມີຂະໜາດປ່ຽນແປງ

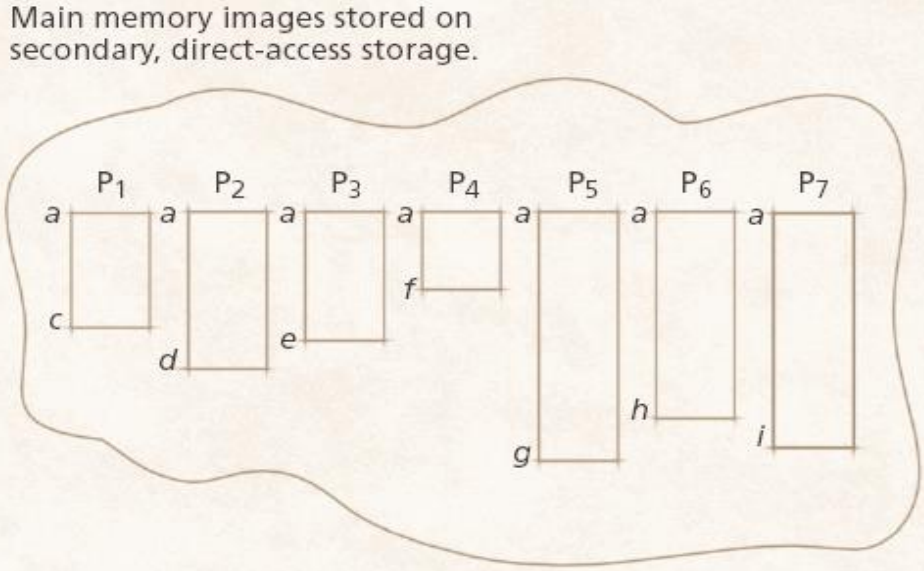
- ຍຸດທະສາດໃນການວາງຂໍ້ມູນໃນໜ່ວຍຄວາມຈຳ



ການໃຫ້ຫຼາຍຂະບວນການສະຫຼັບກັນໃຊ້ໜ່ວຍຄວາມຈຳບ່ອນໃດໜຶ່ງ

- ບໍ່ມີຄວາມຈຳເປັນຕ້ອງເອົາຂະບວນການທີ່ບໍ່ໄດ້ເຮັດເກັບໄວ້ໃນໜ່ວຍຄວາມຈຳ
- Swapping ເປັນການສັບປ່ຽນຂະບວນການເຂົ້າ ແລະ ອອກ
  - ເອົາສະເພາະຂະບວນການທີ່ກຳເຮັດວຽກຢູ່ໄວ້ໃນໜ່ວຍຄວາມຈຳຫຼັກ ສ່ວນຂະບວນການອື່ນໃຫ້ເອົາອອກໄປເກັບໄວ້ໃນໜ່ວຍຄວາມຈຳສຳຮອງ
  - ເຮັດໃຫ້ມີເນື້ອທີ່ໜ່ວຍຄວາມຈຳເຫຼືອຫຼາຍທີ່ສຸດ
  - ຂ້ອນຂ້າງຈະເສຍເວລາໃນການສັບປ່ຽນ
  - ຈະດີກ່ວາ: ຖ້າສາມາດເອົາຂະບວນການທັງໝົດໄວ້ໃນໜ່ວຍຄວາມຈຳພ້ອມກັນ
    - ເຮັດໃຫ້ເປືອງໜ່ວຍຄວາມຈຳ ແຕ່ຄວາມໄວໃນການຕອບສະໜອງດີ
    - ໂດຍໃຊ້ຫຼັກການແບ່ງໜ້າ

ການໃຫ້ຫຼາຍຂະບວນການສະຫຼັບກັນໃຊ້ໜ່ວຍຄວາມຈຳບ່ອນໃດໜຶ່ງ



1. Only one process at a time resides in main memory.

2. That process runs until
   a) I/O is issued,
   b) timer runs out or
   c) voluntary termination.

3. System then swaps out the process by copying the swapping area (main memory) to secondary storage.

4. System swaps in next process by reading that process's main memory image into the swapping area. The new process runs until it is eventually swapped out and the next user is swapped in, and so on.