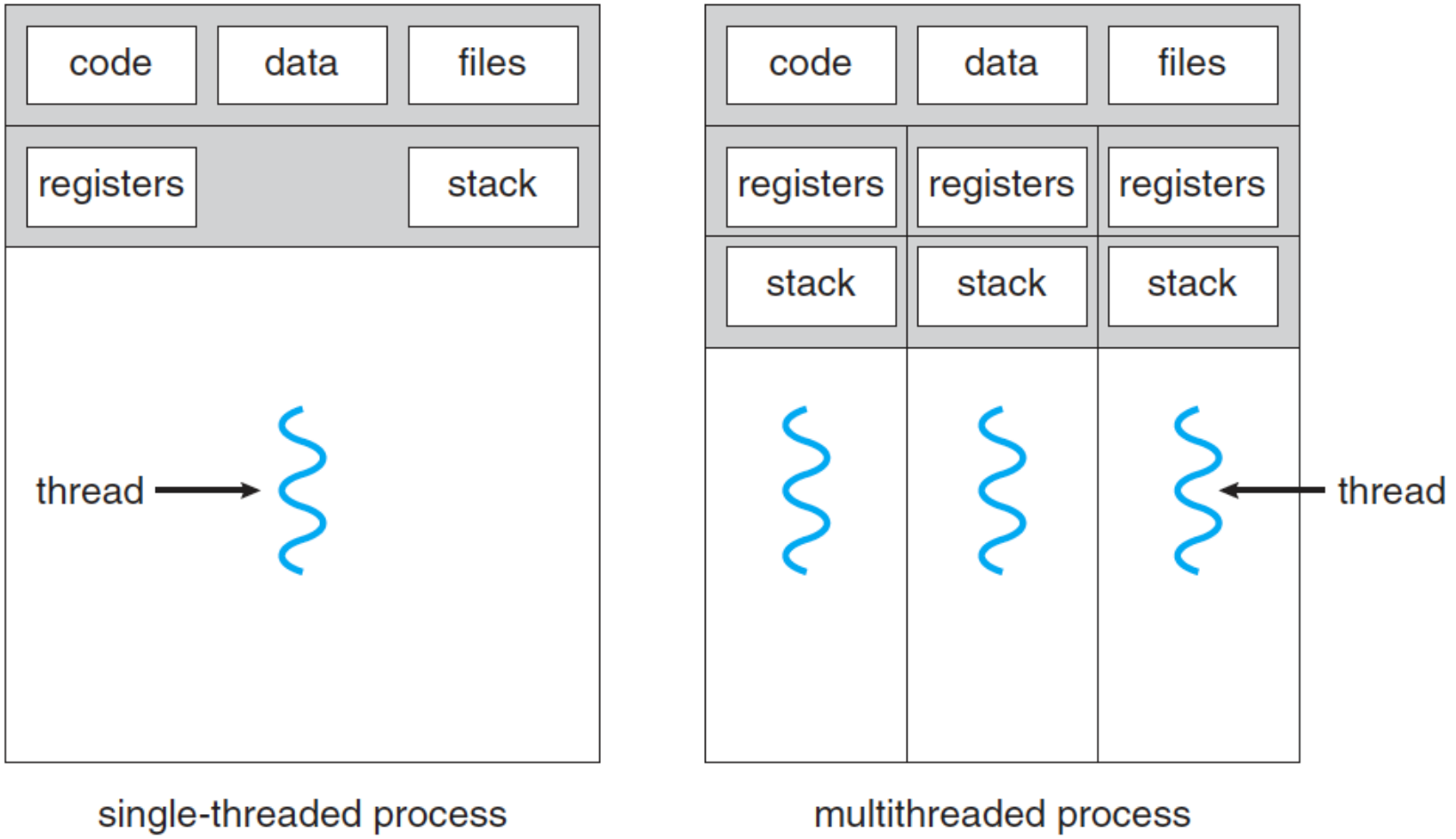
01.1. Програмні потоки (threads)

1. Назви:
- Light-Weight Process (LWP)
- Thread of control
- потік управління обчисленнями
- потік виконання обчислень
- програмний потік

2. Програмний потік (thread) — це одиниця розпаралелення обчислень в межах одного обчислювального процеса.

01.2. Програмні потоки (threads)



single-threaded process

multithreaded process

01.3. Програмні потоки (threads)

В кожного програмного потоку є власні:
1) Thread ID;
2) регістровий контекст (лічильник команд + регістри загального призначення);
3) стек (знаходиться в сегменті стеку відповідного процесу).

У потоків одного процесу спільні:
1) сегменти коду та даних обчислювального процесу;
2) всі ресурси обчислювального процесу (об’єкти IPC, мережні сокети, відкриті файли і т.д.).

01.4. Програмні потоки (threads)

Причини використання програмних потоків:

1. Зменшення часу на переключення контексту
(“в системі Solaris переключення контексту процесу в 5-ть разів повільніше переключення контексту потоків” [Silberschatz]).

2. Зменшення часу створення в порівнянні з процесом
(“в системі Solaris створення процесу в 30 разів довше за створення потоку” [Silberschatz]).

3. Зручність організації взаємодії потоків (в т.ч. обміну даними) в порівнянні з процесами (не потрібні засоби IPC, оскільки у потоків спільний адресний простір = віртуальна пам’ять процесу).

+ зберігаються усі переваги розпаралелення як такого

01.5. Програмні потоки (threads)

- user level threads vs. kernel level threads
- user level threads: програмні потоки, які виконуються на рівні користувача без участі ядра ОС (VM)
- kernel level threads: програмні потоки, які підтримуються та управляються безпосередньо ядром ОС (Windows, Linux, Solaris,...)

01.6. Програмні потоки (threads)

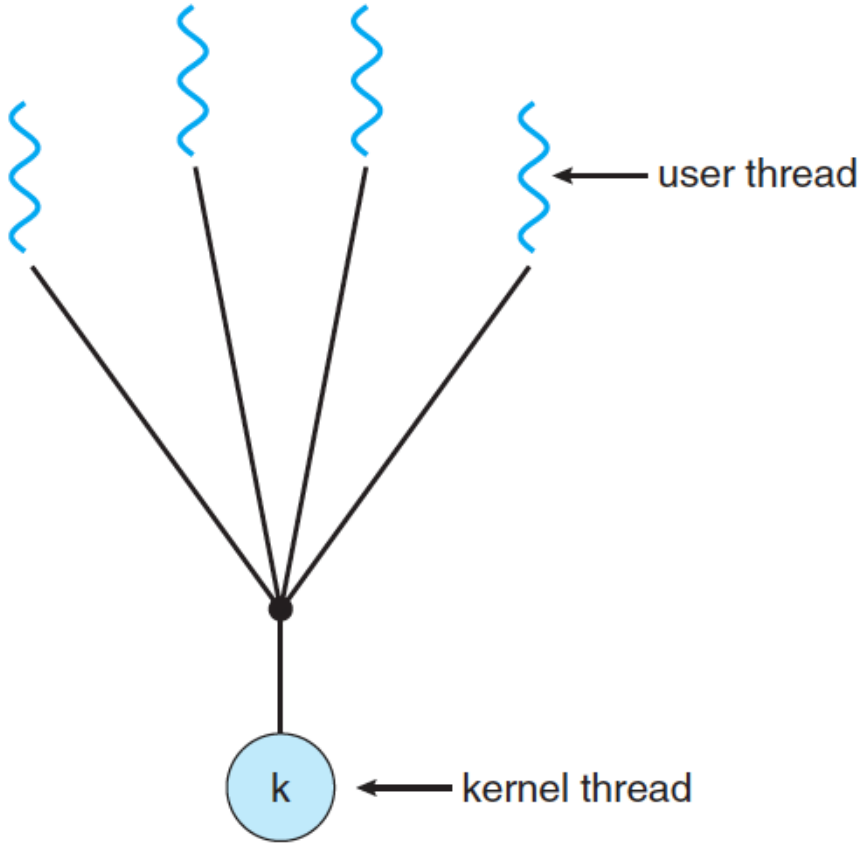
Моделі відображення потоків рівня користувача у потоки рівня ядра:

1. Many-to-One Model (=User level threads)

2. One-to-One Model (=Kernel level theads)

3. Many-to-Many Model (=Hybrid theads)

01.7. Програмні потоки (threads)



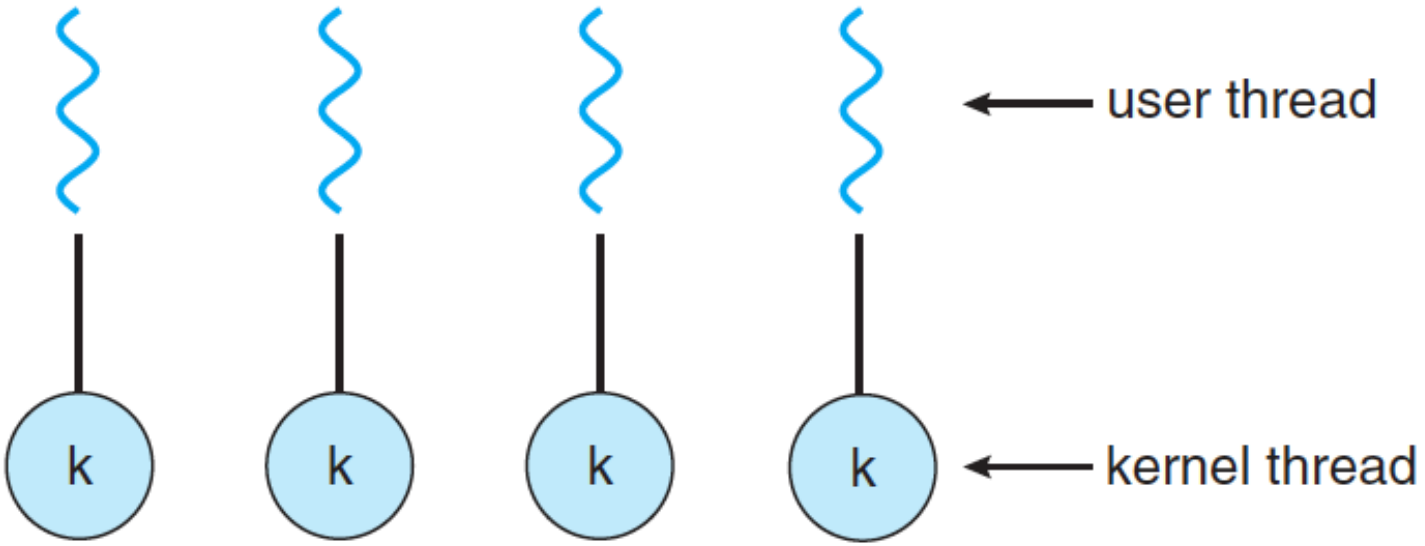
1. Many-to-One Model (=User level threads)

01.8. Програмні потоки (threads)

1. Many-to-One Model (=User level threads):

- всі потоки рівня користувача відображаються в один потік рівня ядра;
- управління потоками виконує стороння бібліотека в режимі користувача, яка реалізує тільки багатозадачність з розділенням у часі;
- відсутні обменження на кількість потоків;
- переключення контексту потоків займає менше часу в порівнянні з іншими моделями;
- приклади: Green theads (scheduled by VM), GNU Portable theads.

01.9. Програмні потоки (threads)



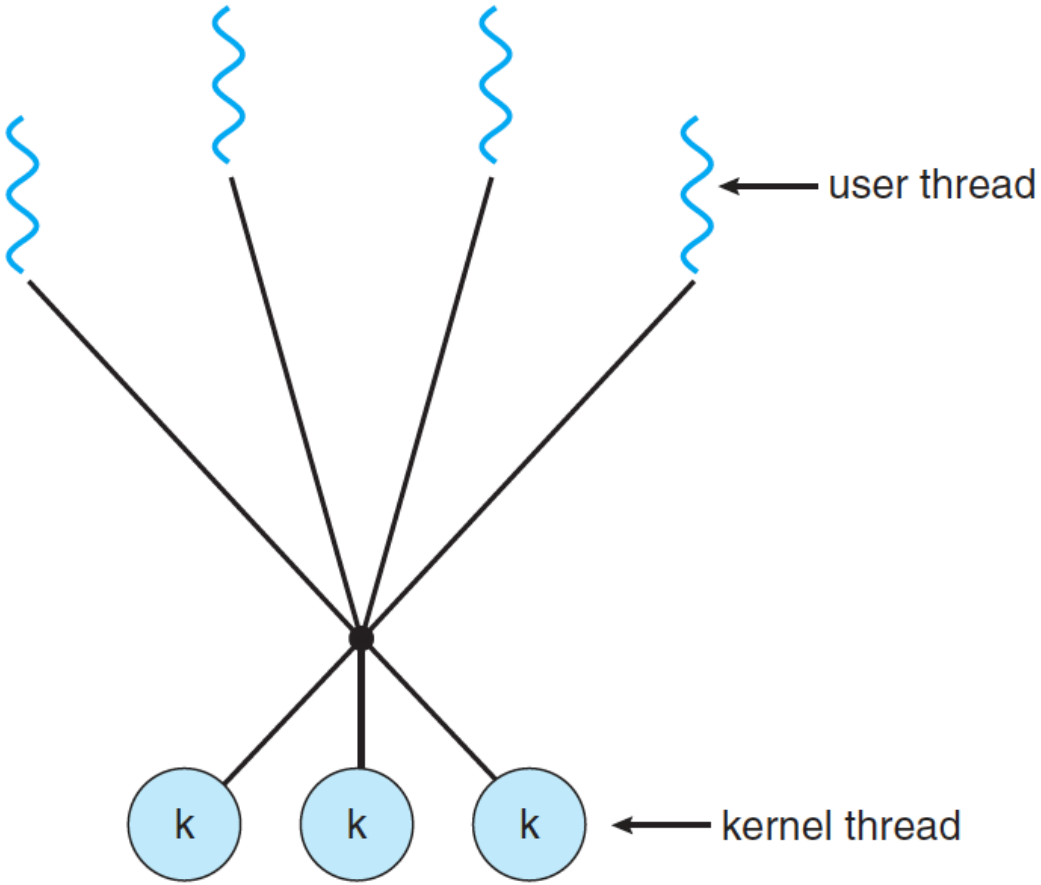
2. One-to-One Model (=Kernel level theads)

01.10. Програмні потоки (threads)

2. One-to-One Model (=Kernel level theads):

- кожний потік рівня користувача відображається в один потік рівня ядра;
- ОС накладає обмеження на максимальну кількість потоків в одному процесі;
- переключення контексту потоків займає більше часу ніж в Many-to-One Model;
- ця модель реалізована в Linux та в ОС сімейства Windows.

01.11. Програмні потоки (threads)



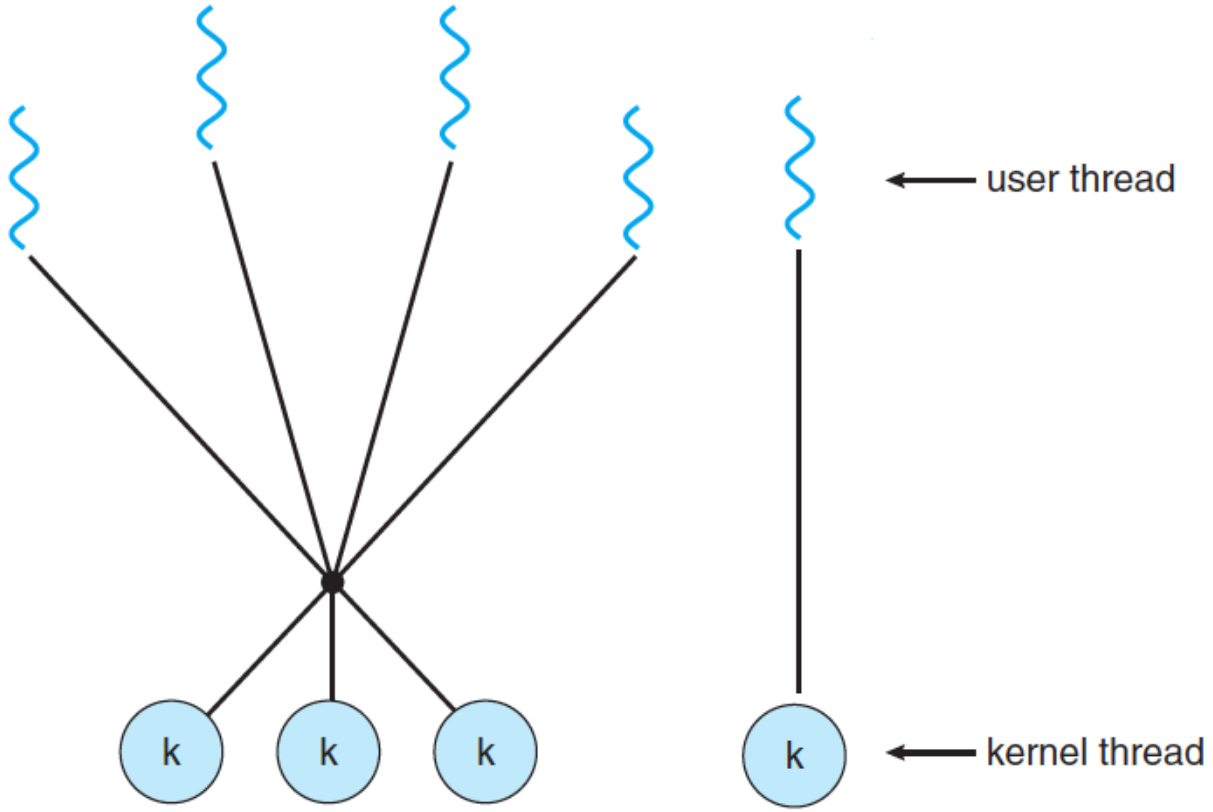
3. Many-to-Many Model (=Hybrid theads)

01.12. Програмні потоки (threads)

3. Many-to-Many Model (=Hybrid theads):

- N потоків рівня користувача відображаються в X ≤ N потоків рівня ядра;
- мета 1: знайти компроміс між перевагами над-швидкого переключення контексту (user threads) та перевагами фізичного («просторового») розпаралелення (kernel threads);
- мета 2: зберегти перевагу необмеженої кількості потоків (необмежена кількість user threads відображається у обмежену кількість kernel threads).

01.13. Програмні потоки (threads)



Варіант Many-to-Many Model: two-level model

01.14. Програмні потоки (threads)

Варіант Many-to-Many Model:

- Двохрівнева модель (two-level model)
- В двохрівневій моделі реалізована додаткова можливість прив’язати окремий потік рівня користувача до одного потоку рівня ядра.
- Двохрівнева модель підтримується в ОС HP-UX, ОС Solaris (до версії 9).

01.15. Програмні потоки (threads)

Програмні інтерфейси для роботи з потоками (thread libraries):

1. POSIX threads (Pthreads) (UNIX, Linux, Solaris та ін.): робота з потоками на рівні ядра та на рівні користувача;

2. Windows threads (Win SDK, .NET): робота з потоками на рівні ядра;

3. Java threads: «обгортка» Pthreads або Windows theads.

01.16. Програмні потоки (threads)

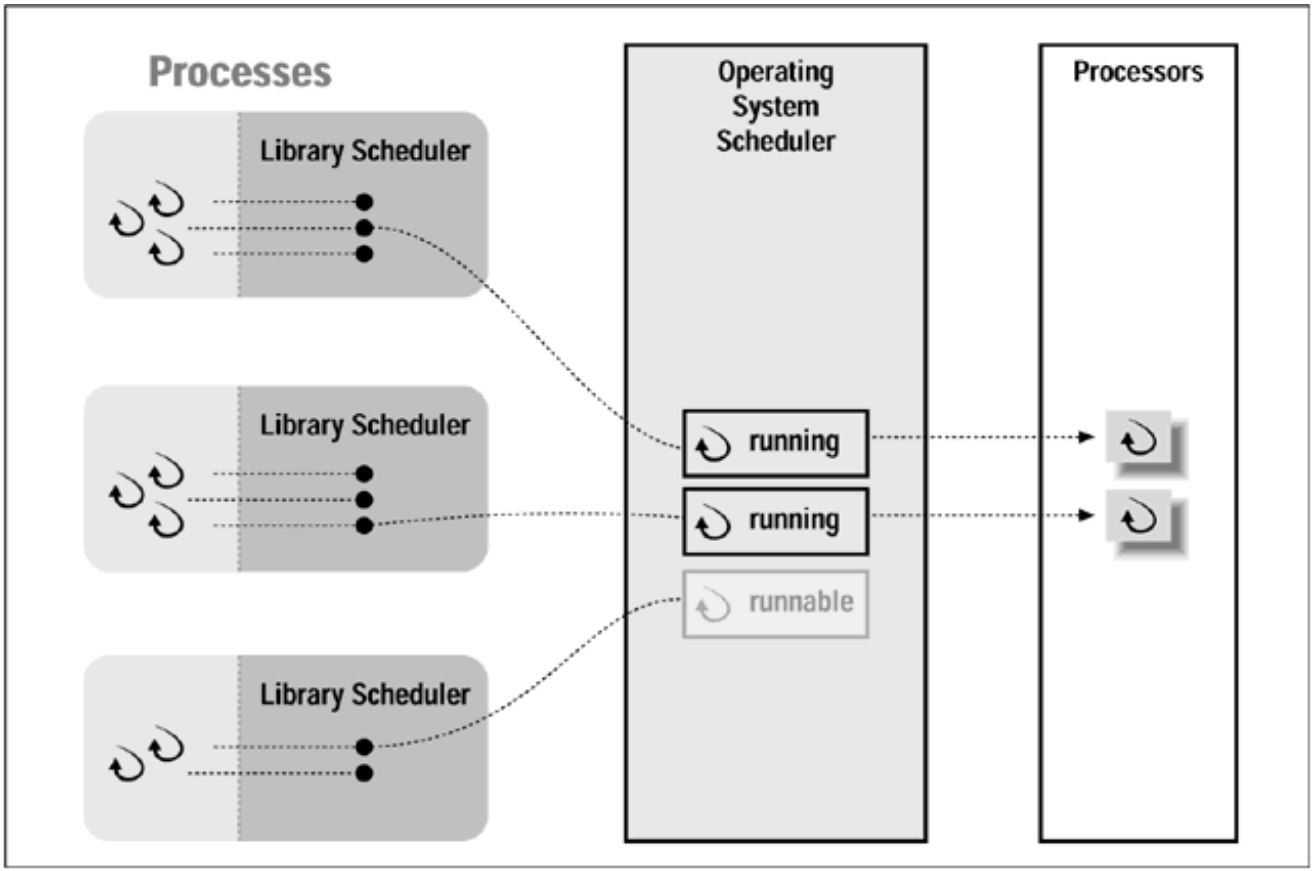


Схема реалізації багатопоточності за допомогою Pthreads: Many-to-One Model

01.17. Програмні потоки (threads)

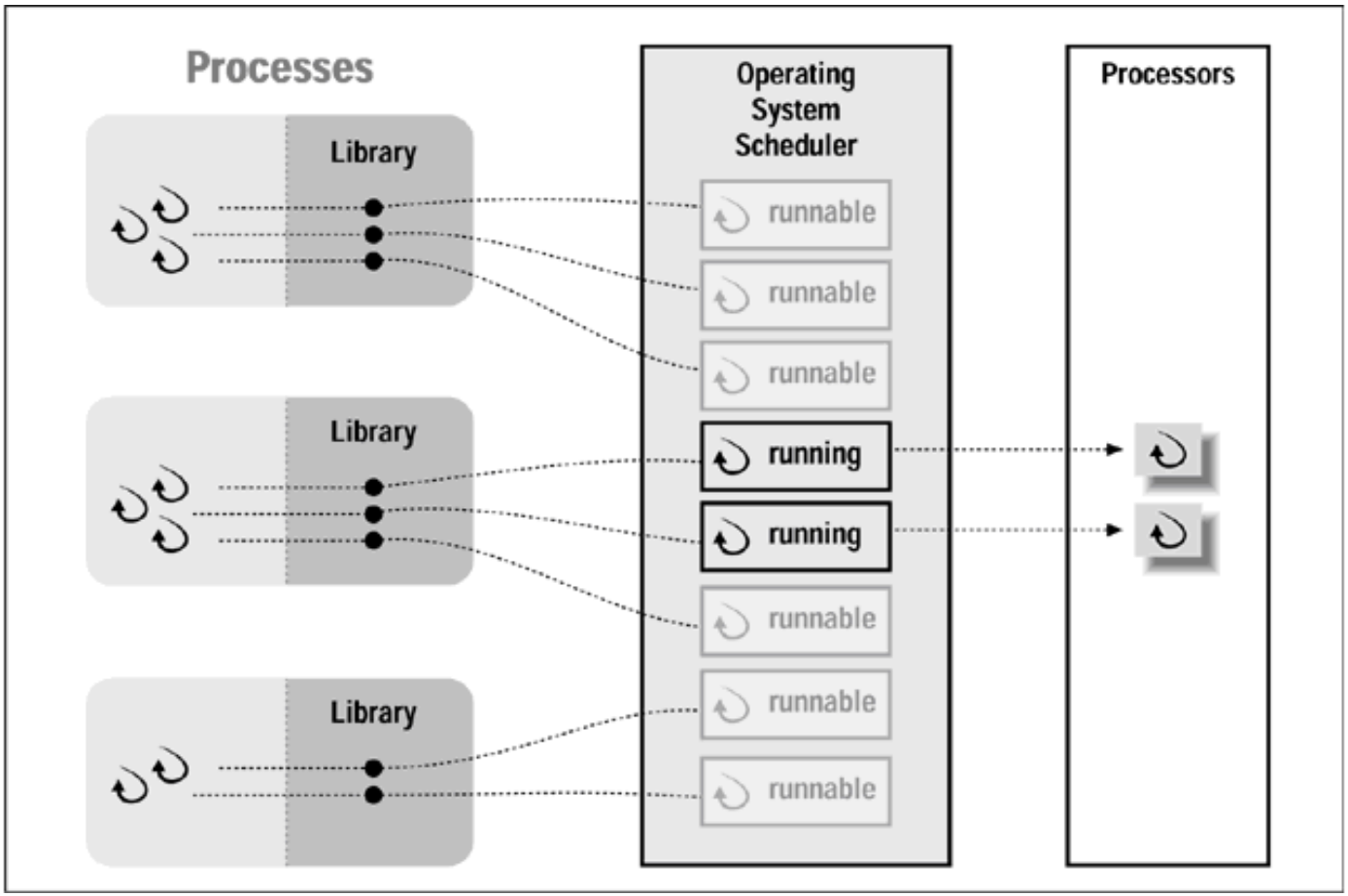


Схема реалізації багатопоточності за допомогою Pthreads: One-to-One Model

01.18. Програмні потоки (threads)

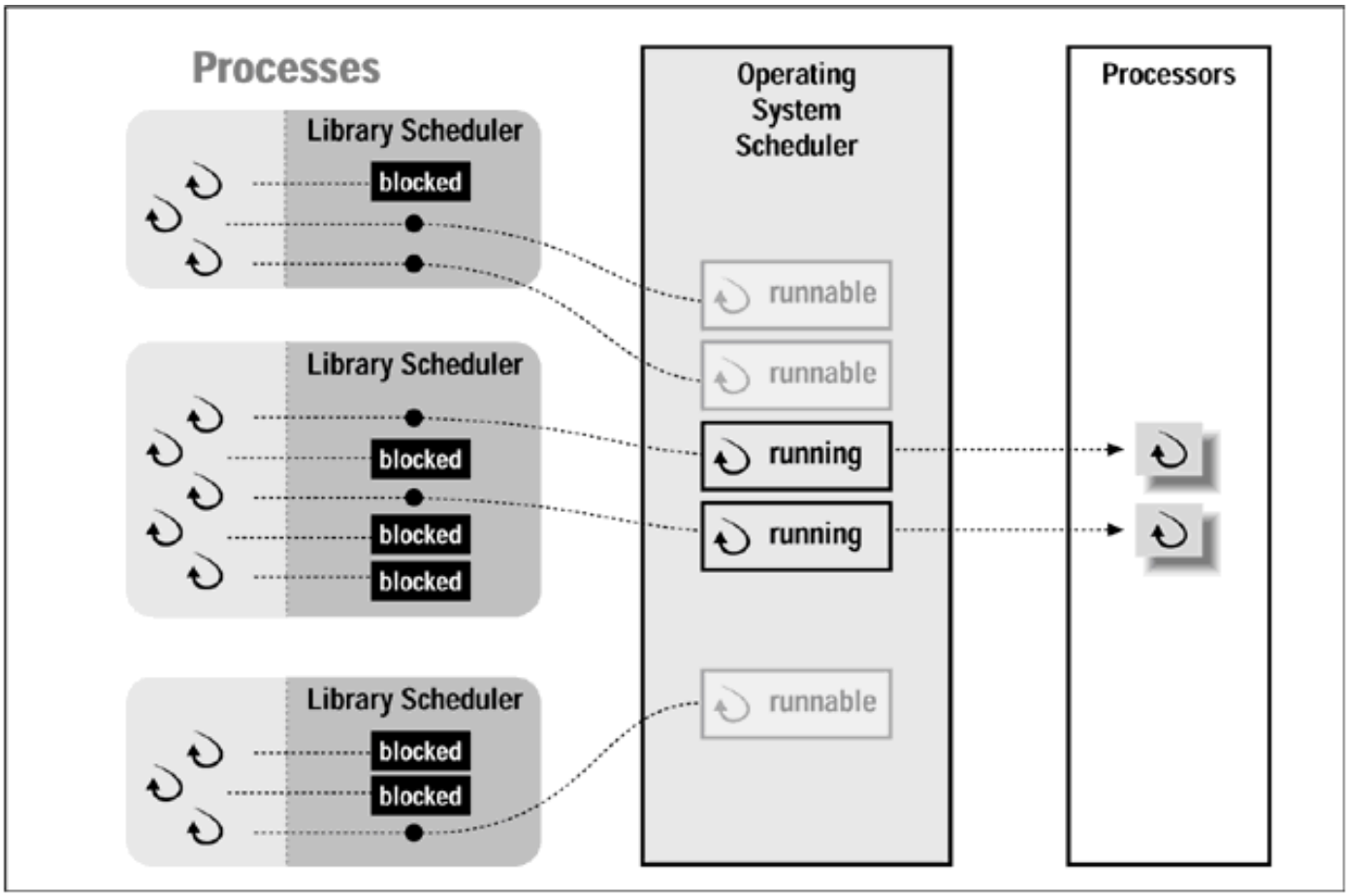
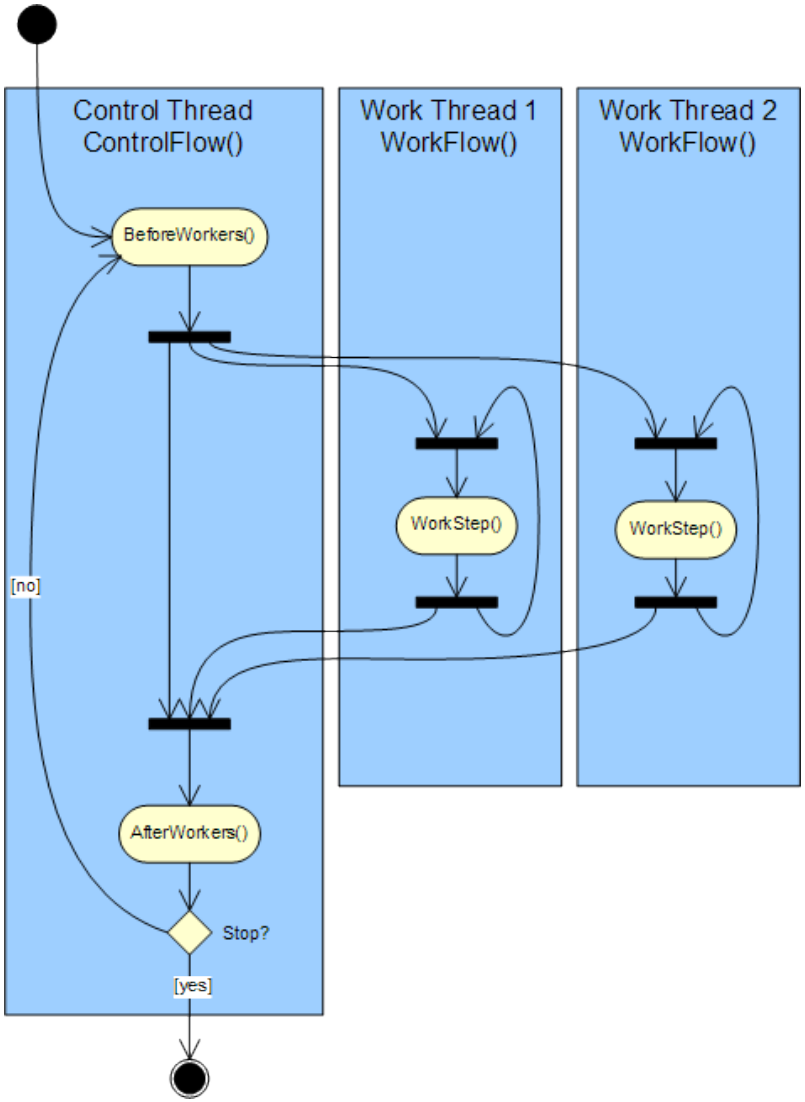


Схема реалізації багатопоточності за допомогою Pthreads: Many-to-Many Model

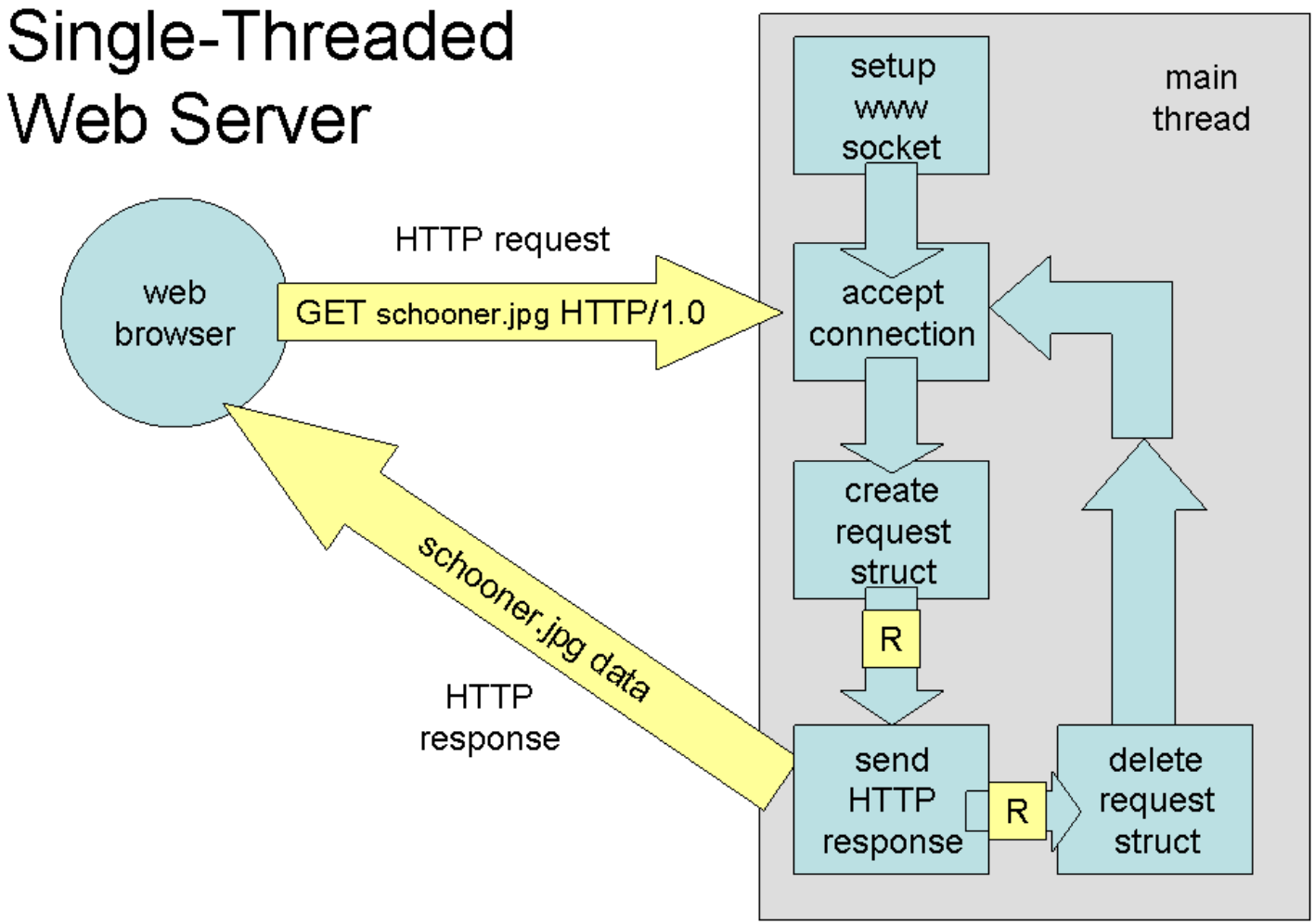
01.19. Програмні потоки (threads)



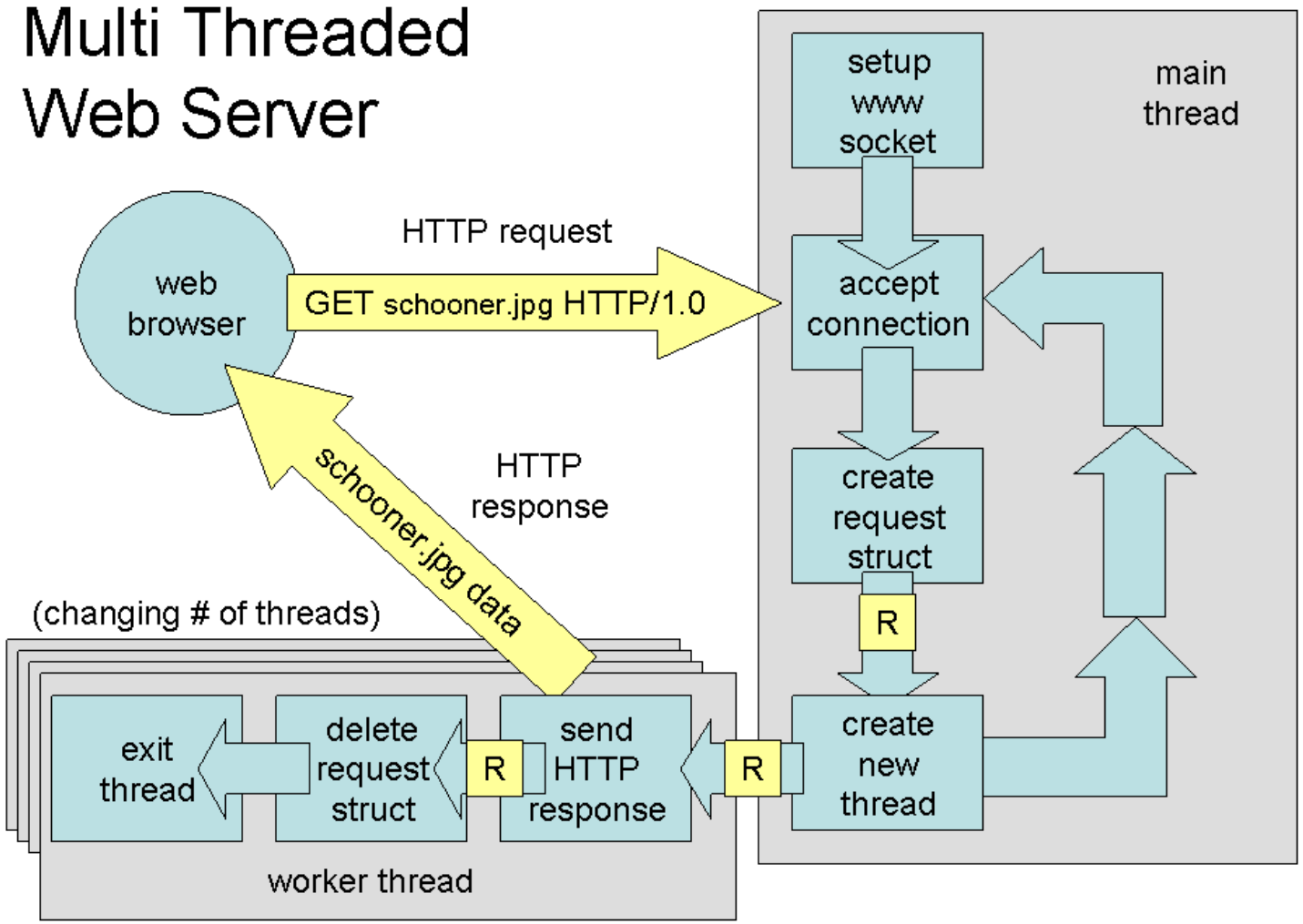
Приклад схеми розпаралелення потоків в межах процесу.

- process={1...n threads}
- main thread

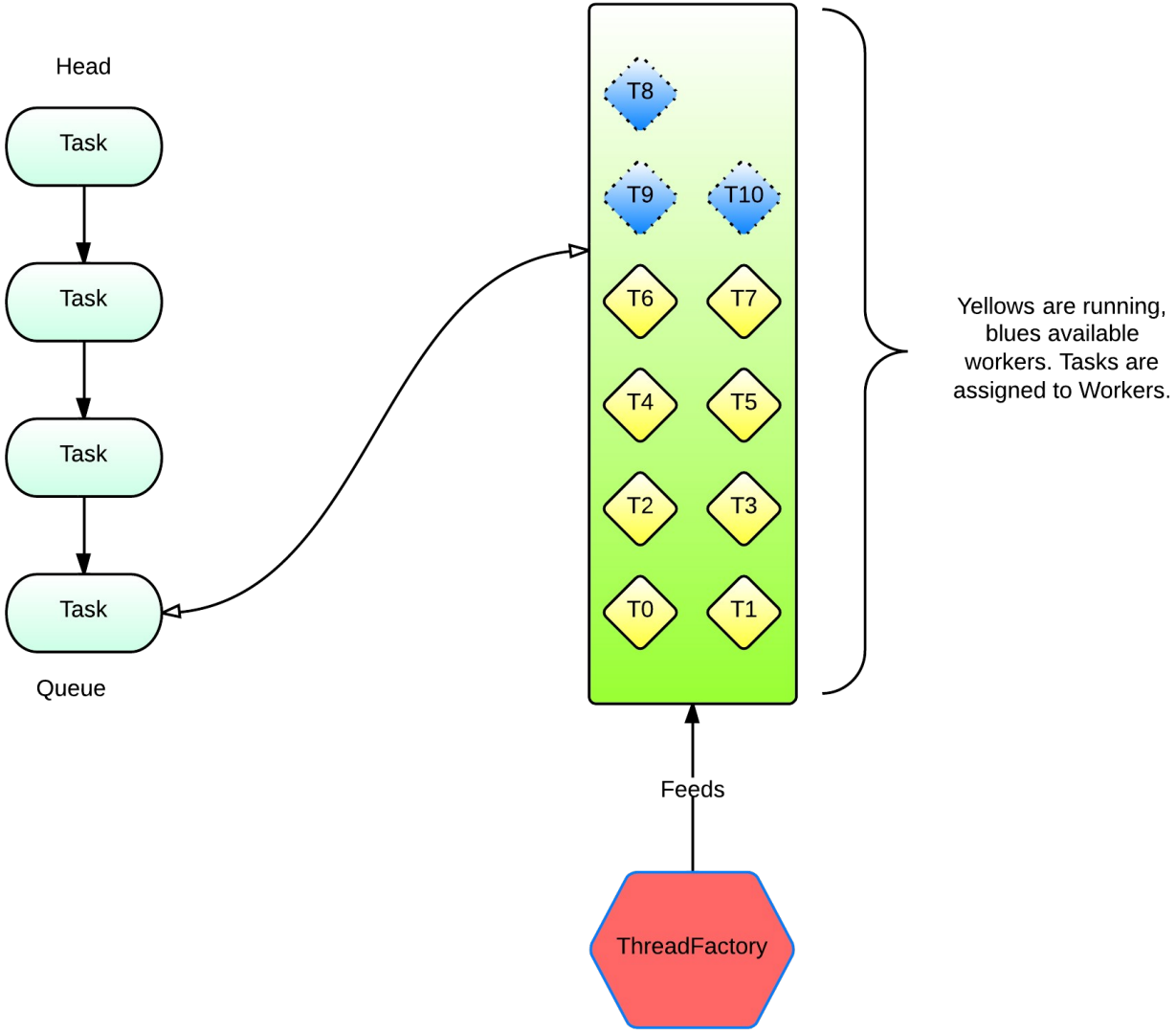
01.20. Програмні потоки (threads)



01.21. Програмні потоки (threads)



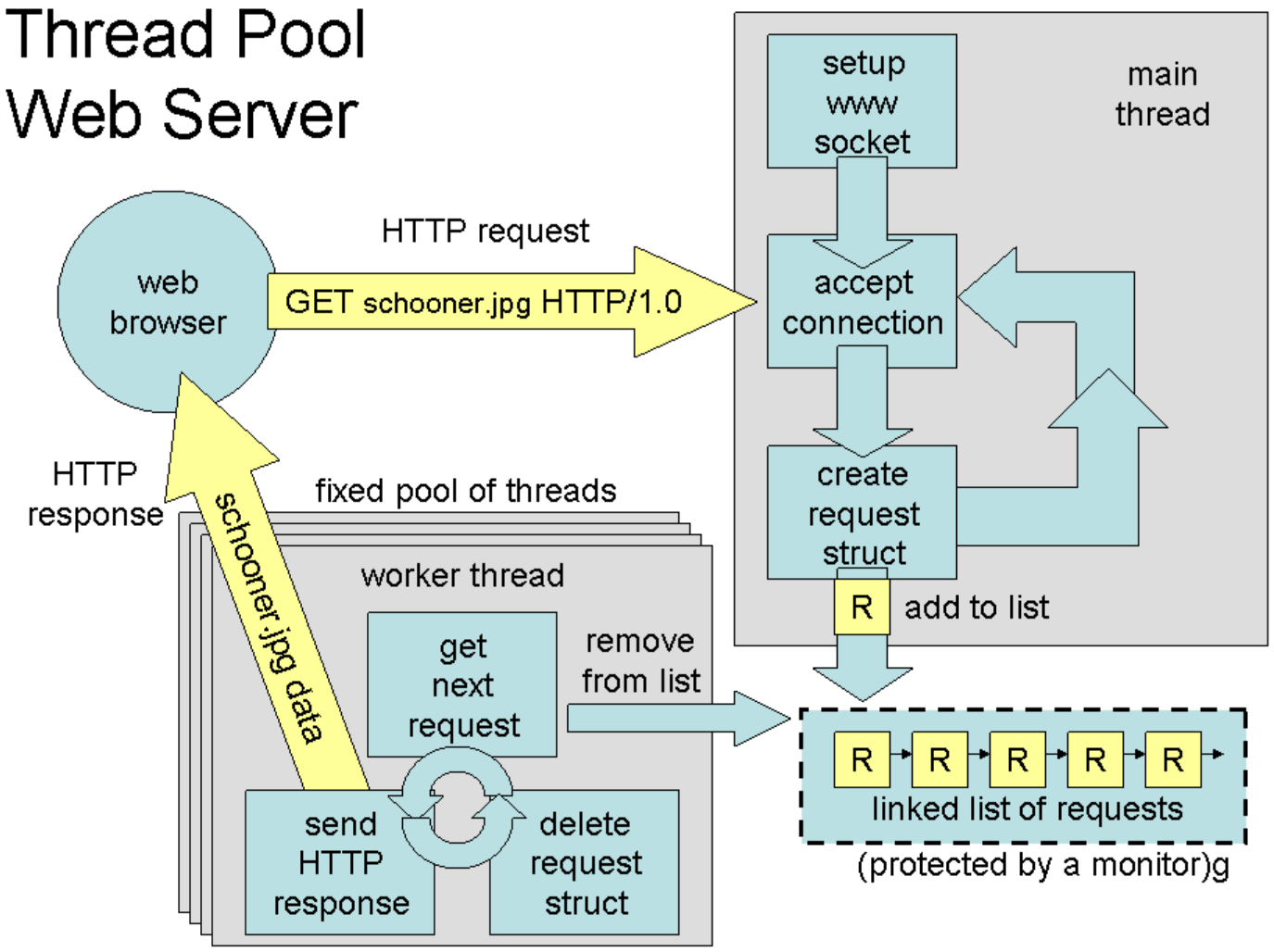
01.22. Програмні потоки: thread pool



01.23. Програмні потоки: thread pool

1. Обмежений набір заздалегідь створених [однакових] потоків, яким передаються нові завдання у міру їх надходження.
2. Ідея: зменшення витрат на породження нових потоків під час роботи програми.
3. Типовий сценарій використання: ситуації, в яких максимально можливе обчислювальне навантаження є відомим (наприклад, максимальна кількість одночасних запитів від користувачів до web-сервера).
4. Коли максимально можливе навантаження невідомо, вибір кількості потоків у пулі стає складною проблемою.
5. Оптимізаційні задачі: наприклад, балансування розмірів пулів потоків різного функціонального призначення.

01.24. Програмні потоки (threads)



01.25. Програмні потоки (threads)

Диспетчеризація потоків:

1. В сучасних ОС з точки зору планування паралельного виконання потоки на рівні ядра прирівняні до процесів (термін: task).

2. В моделях Many-to-One та Many-to-Many, як правило, присутні додаткові можливості по конфігурації режимів планування паралельного виконання потоків (реалізовані викликами функцій/методів відповідного API).

01.26. Програмні потоки (threads)

Недоліки використання потоків:

1. Зменшення надійності роботи внаслідок використання спільної для всіх потоків пам’яті.

2. Необхідність додаткових «зусиль» для забезпечення синхронізації потоків (в першу чергу, це організація одночасного доступу потоків до спільних структур даних).

3. Критична помилка в одному потоці зупиняє весь процес.

01.27. Програмні потоки (threads)

- Перелік TID потоків, які виконуються в межах процесу з PID=1183:
  $ ls /proc/1183/task/
- Визначити кількість програмних потоків процесу з PID=1183:
  $ ls /proc/1183/task/ | wc -w
  $ cat /proc/1183/status | grep -i Threads
  $ ps hH p 1183 | wc -l
- Подивитись ділянку дерева процесів для процесу з PID=1183 та його потоків:
  $ pstree -pau -l -G -s 1183

02.1. Взаємодія обчислювальних процесів

- Процеси можна поділити на незалежні (independent) та взаємодіючі (cooperating).
- Взаємодія процесів обумовлена
  1) спільним використанням даних/інформації;
  2) прискоренням обчислень за рахунок декомпозиції даних;
  3) прискоренням обчислень за рахунок функціональної декомпозиції.
- Відтак потрібний механізм взаємодії процесів (Inter-Process Communication, IPC), який би дозволив їм обмінюватись даними та службовою інформацією.

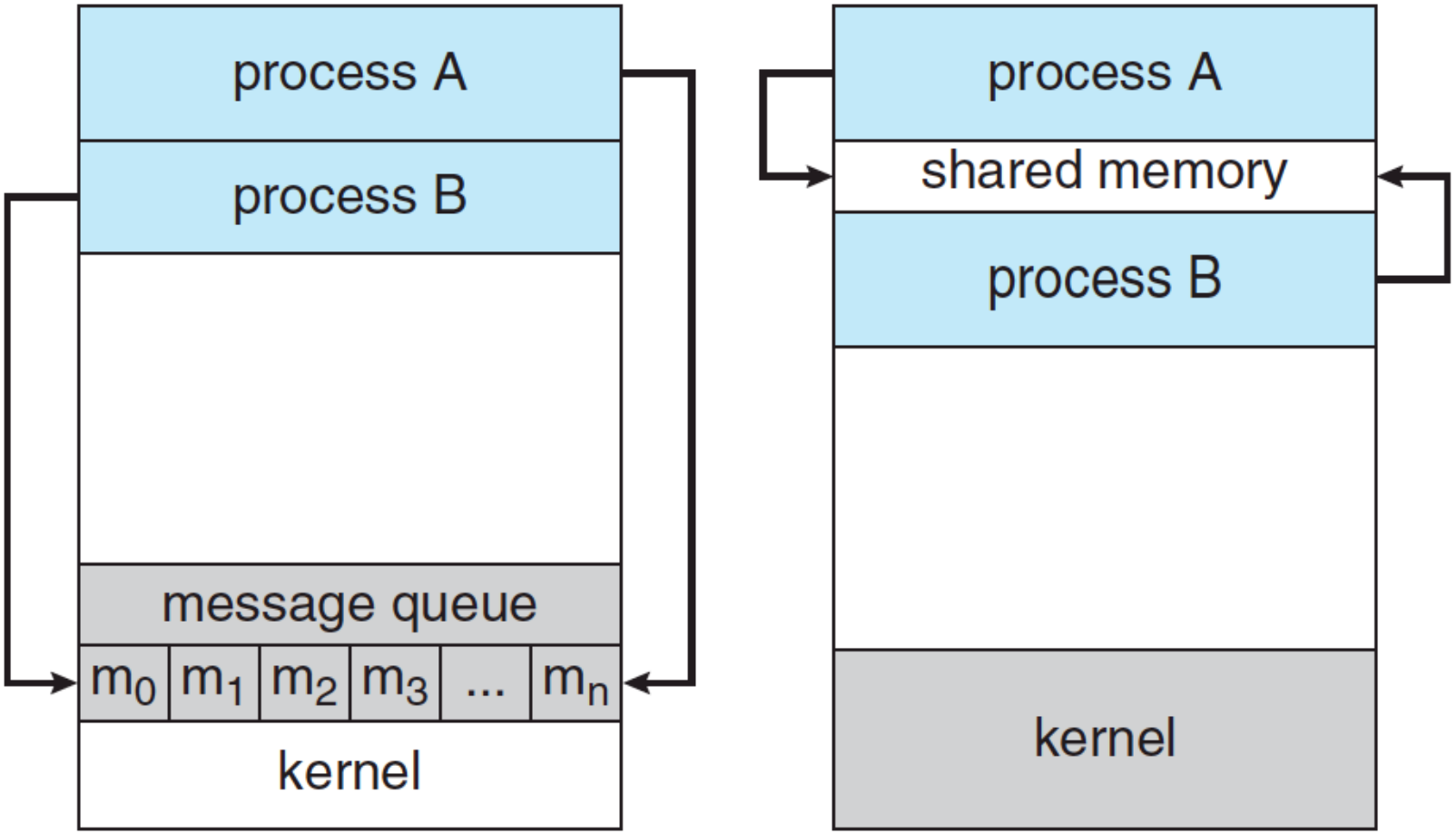
02.2. Взаємодія обчислювальних процесів

Дві фундаментальні моделі IPC:

1. Shared memory (спільна пам’ять; пам’ять, що розділяється): процеси обмінюються даними за допомогою операцій запису та читання даних зі спільної ділянки пам’яті.

2. Message passing (обмін повідомленнями): процеси обмінюються даними за допомогою відправки та прийому повідомлень.

02.3. Взаємодія обчислювальних процесів



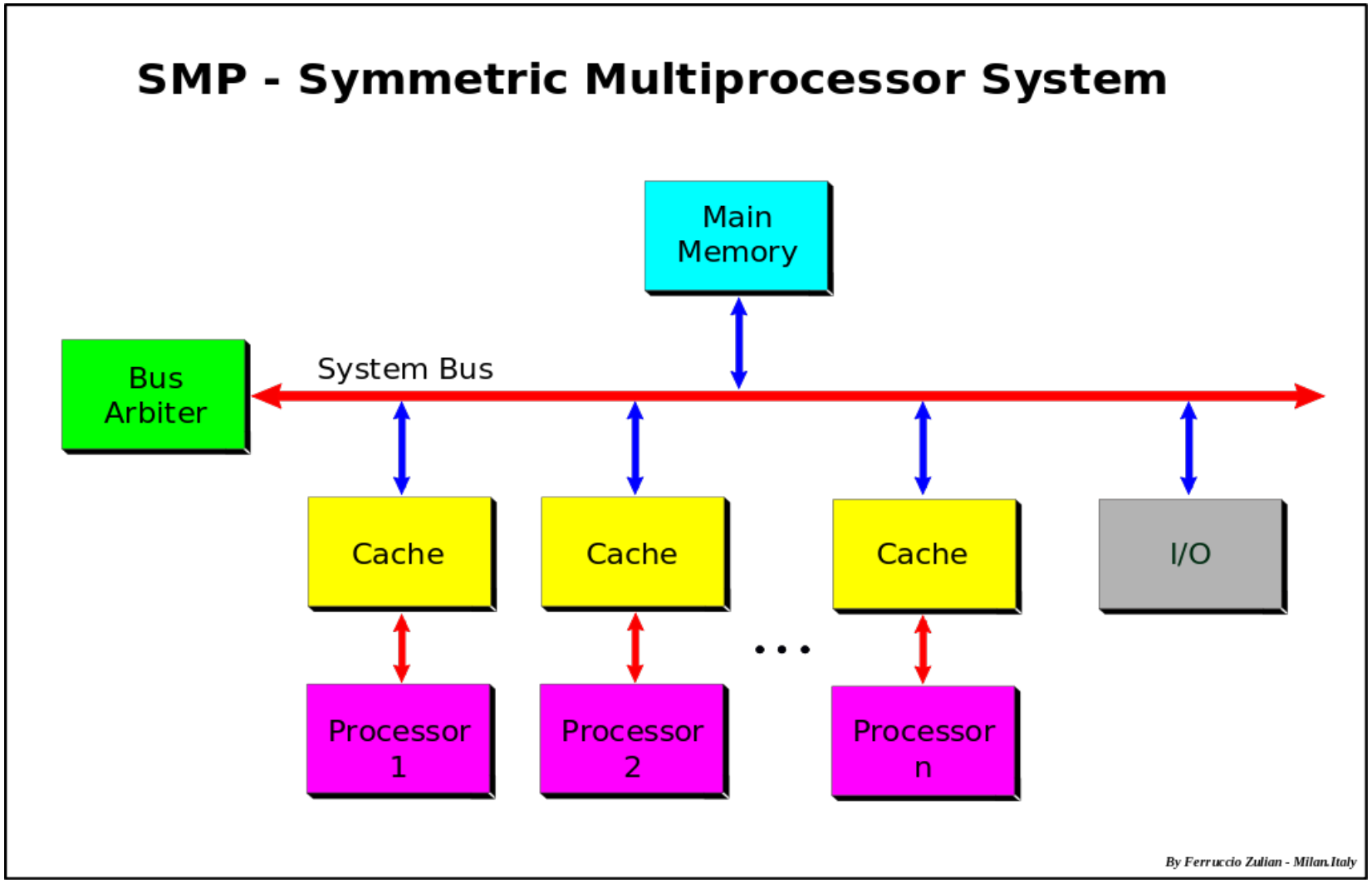
02.4. Взаємодія обчислювальних процесів

- Вибір моделі залежить від архітектури паралельного обчислювача та моделі розпаралелення, яка реалізується в програмі.

- В сильно-зв’язаних мультипроцесорних системах (зокрема в SMP-системах) перевагу віддають моделі shared memory.

- SMP (Symmetric MultiProcessing) - “Симетрична мультипроцесорність - це архітектура багатопроцесорних комп'ютерів, в якій два або більше однакових процесорів підключаються до загальної пам'яті. Більшість багатопроцесорних систем сьогодні використовують архітектуру SMP.”

02.5. Взаємодія обчислювальних процесів



02.6. Взаємодія обчислювальних процесів

- В розподілених системах перевагу віддають моделі message passing.
- В багатоядерних системах вигідно застосовувати гібридні моделі, які поєднують елементи shared memory та message passing.
- Засоби взаємодії процесів в моделі message passing одночасно є засобами синхронізації процесів (за рахунок блокування на операціях send/receive).

02.7. Взаємодія обчислювальних процесів: Засоби IPC

- неіменовані канали (pipes),
- іменовані канали (named pipes),
- черги повідомлень (message queues),
- сокети (sockets),
- віддалений виклик процедур (RPC, remote procedure call),
- віддалений виклик методів (RMI, remote method invocation),

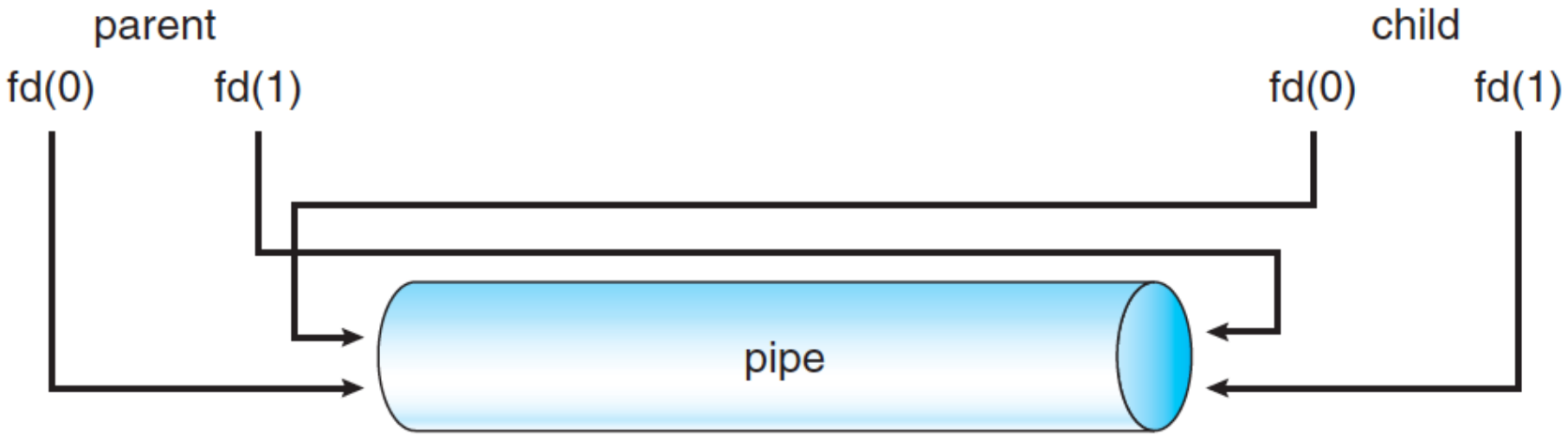
+

- спільна пам’ять (shared memory),
- файл, відкритий одночасно двома або більше процесами,
- файли відображені в пам’ять (memory mapped files),

+

- сигнали (signals),
- [семафори (semaphores)].

02.8. Взаємодія обчислювальних процесів: неіменований канал (pipe)




```
int fd[2];

pipe(fd);
write(fd[1], buf, BUF_SIZE);
read(fd[0], buf, BUF_SIZE);
```

03.1. Синхронізація обчислювальних процесів та потоків

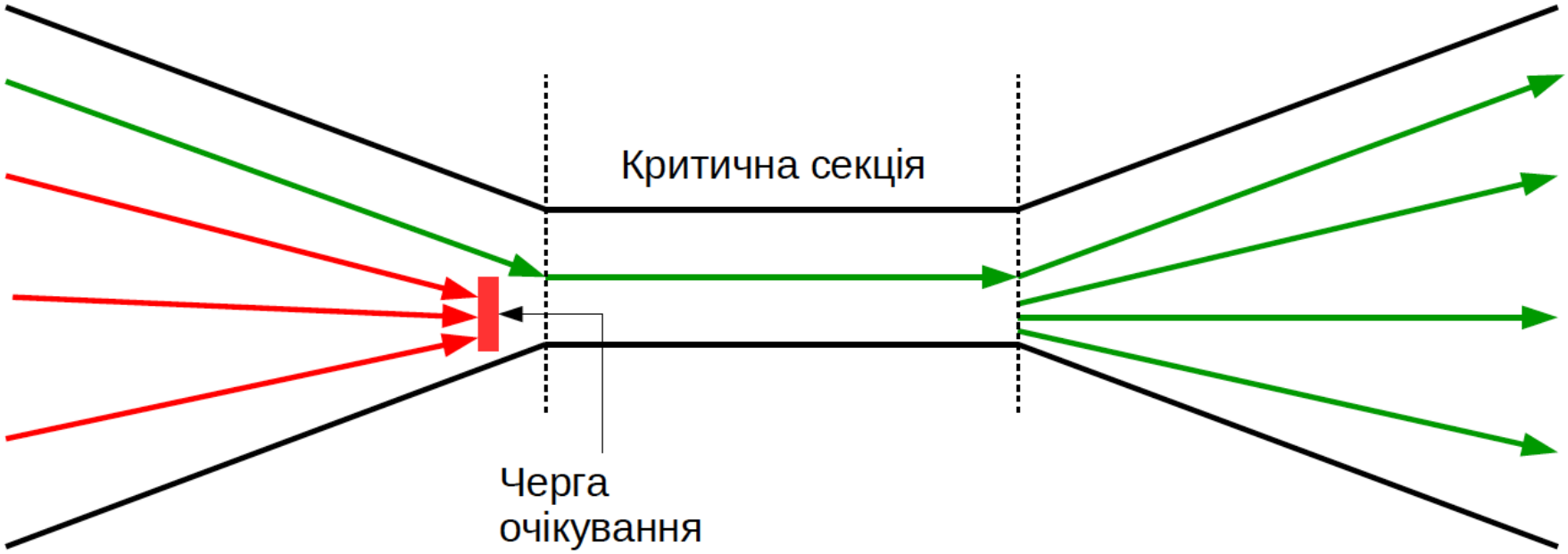
- Взаємодіючі процеси та потоки потребують координації своїх спільних дій для
  1) запобігання помилкового виконання (порушення логіки роботи паралельної програми);
  2) забезпечення цілісності (consistency) спільних даних (впорядкування операцій читання/запису в часі).
- Для цього використовуються засоби синхронізації процесів та потоків, які надають або ядро ОС, або сторонні бібліотеки (наприклад, Pthreads).
- При вирішенні проблем синхронізації розрізняють фізичний та логічний час.

03.2. Синхронізація обчислювальних процесів та потоків

Приклади засобів синхронізації:

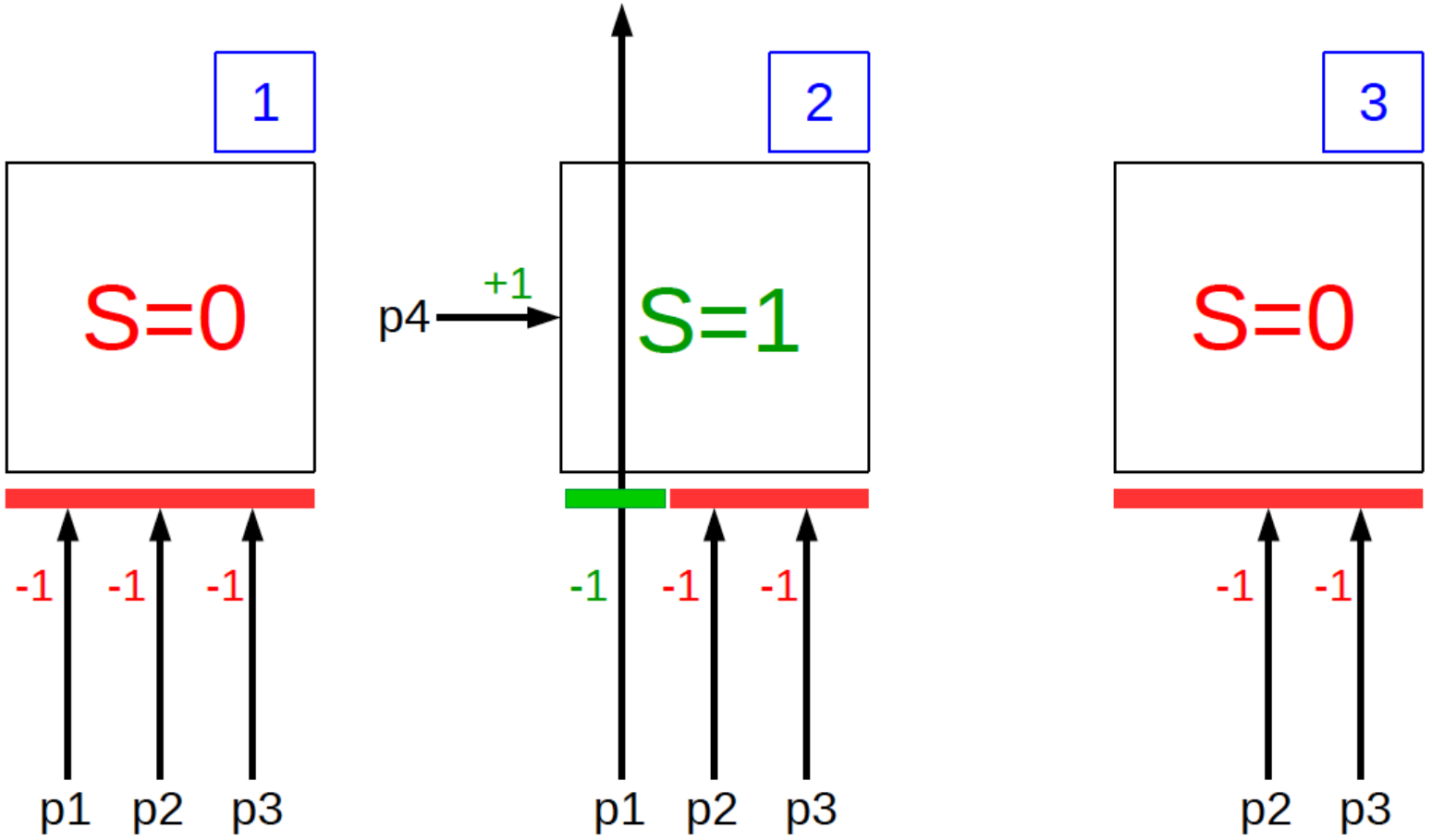
1. Блокуючі операції send/receive (message passing).
2. Критична секція коду (critical section).
3. Семафор (semaphore).
4. М’ютекс (lock, mutex, binary semaphore).
5. Монітор (monitor).
6. Умовна змінна (conditional variable).
7. Пам’ять з підтримкою транзакцій (transactional memory).

03.3. Синхронізація обчислювальних процесів та потоків



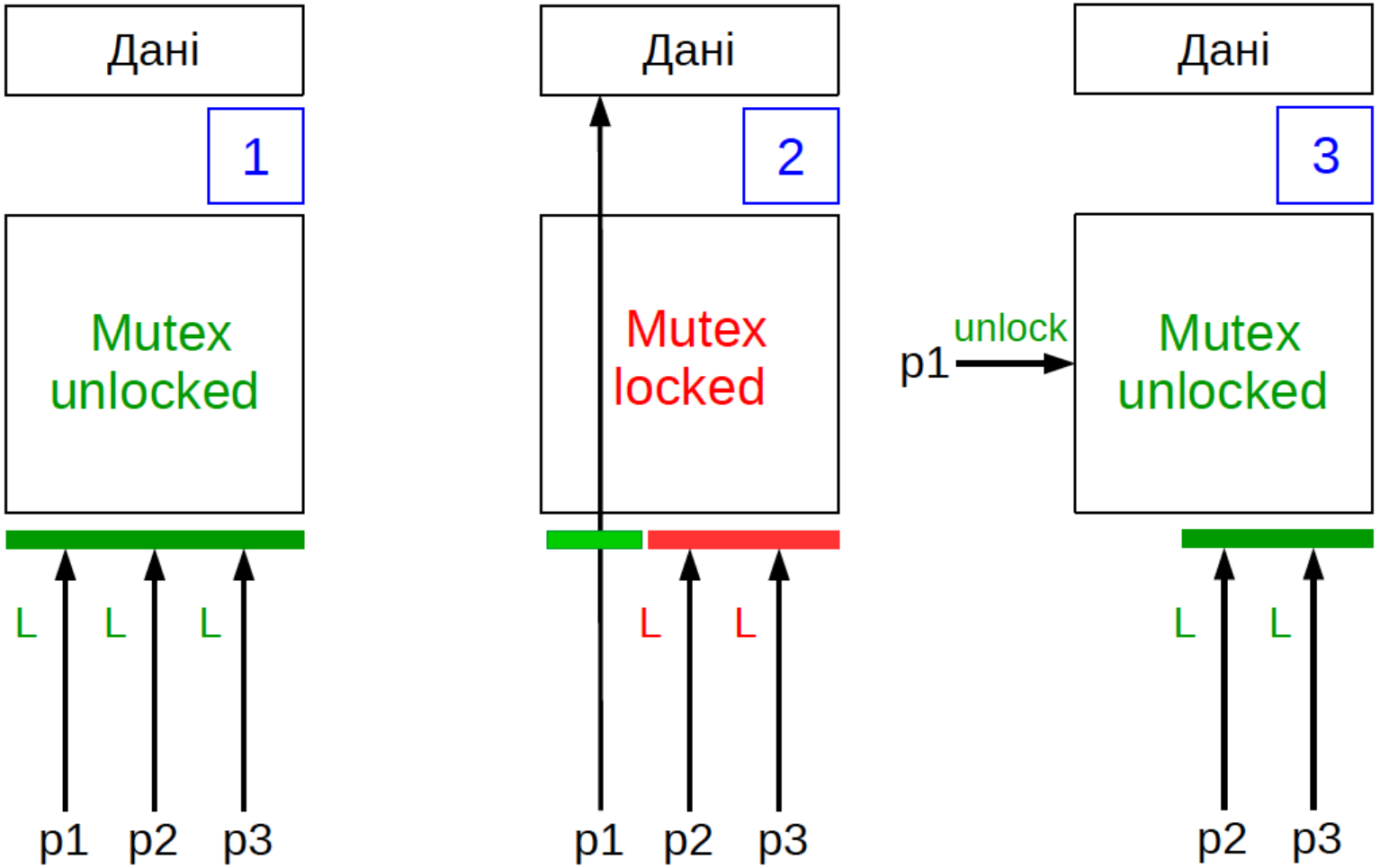
Критична секція коду (critical section)

03.4. Синхронізація обчислювальних процесів та потоків



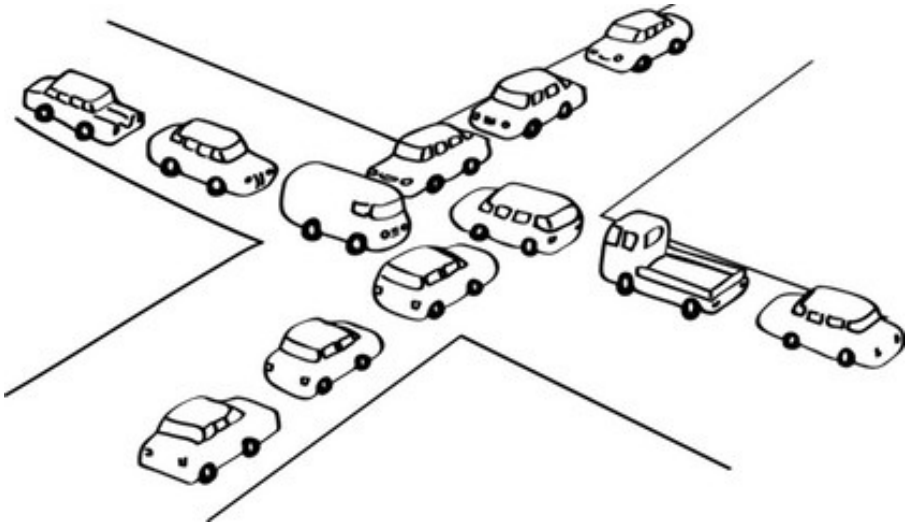
Семафор (semaphore)

03.5. Синхронізація обчислювальних процесів та потоків



М’ютекс (mutex)

03.6. Синхронізація обчислювальних процесів та потоків



Deadlock: ситуація, коли в множині процесів, що знаходяться в стані очікування на подію, знаходиться процес, який один здатний згенерувати цю подію.

Проблема 1: аналіз програм (в тому числі автоматичний) на наявність deadlocks.

Проблема 2: формальний доказ того, що розроблена паралельна програма ніколи не опиниться в ситуації deadlock → числення процесів (process calculi).