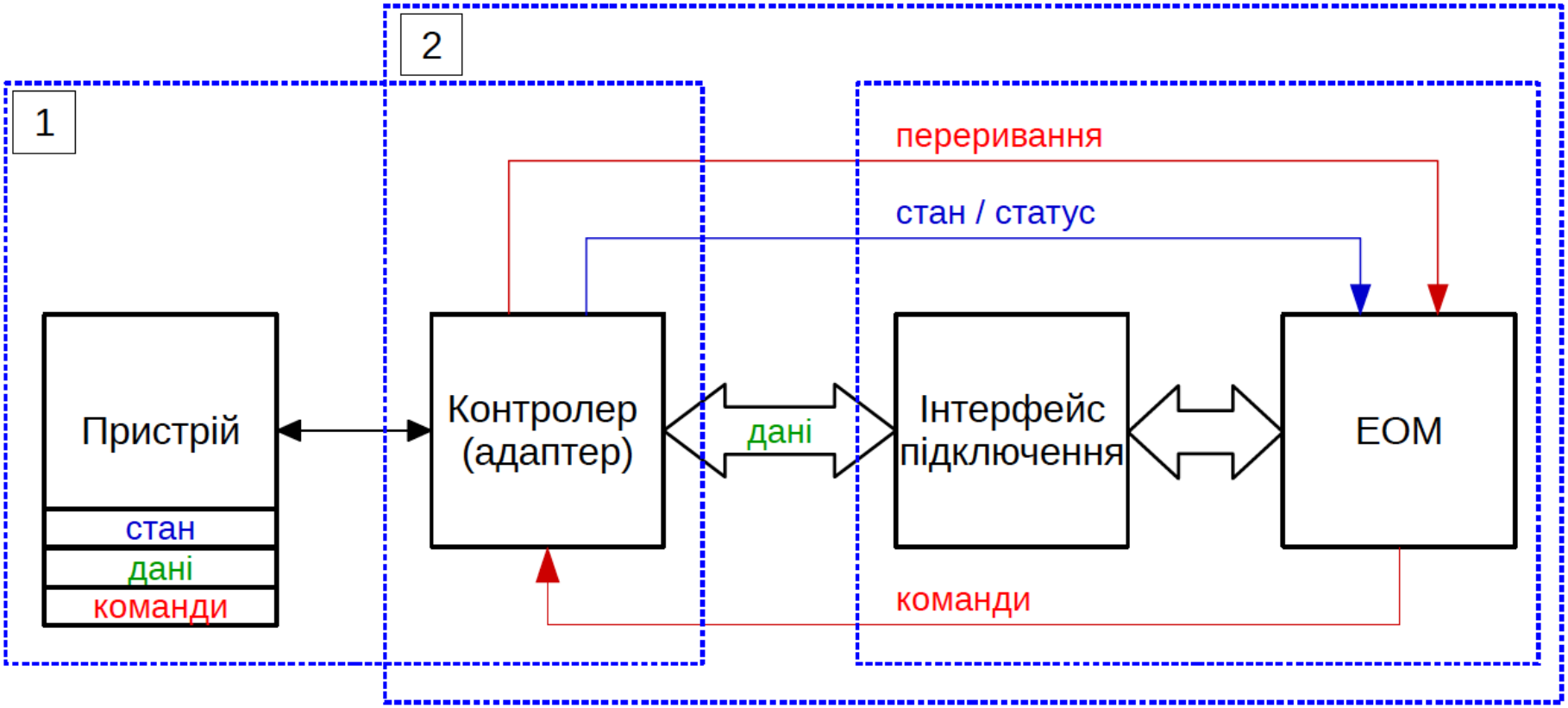
01.1. Фізична організація пристроїв вводу/виводу



Узагальнена схема фізичної організації пристрою вводу/виводу (ПВВ) та його взаємодії з ЕОМ

01.2. Фізична організація пристроїв вводу/виводу

Основні типи пристроїв вводу/виводу з точки зору способу обміну командами та данними з ЕОМ:

1) символьні (байт-орієнтовані); приклади: клавіатура, термінал, сенсорна система та ін.

2) блочні (блок-орієнтовані); приклади: накопичувачі на жорсткому диску (HDD), оптичні приводи, накопичувачі USB;

3) мережні пристрої; приклад: мережний адаптер;

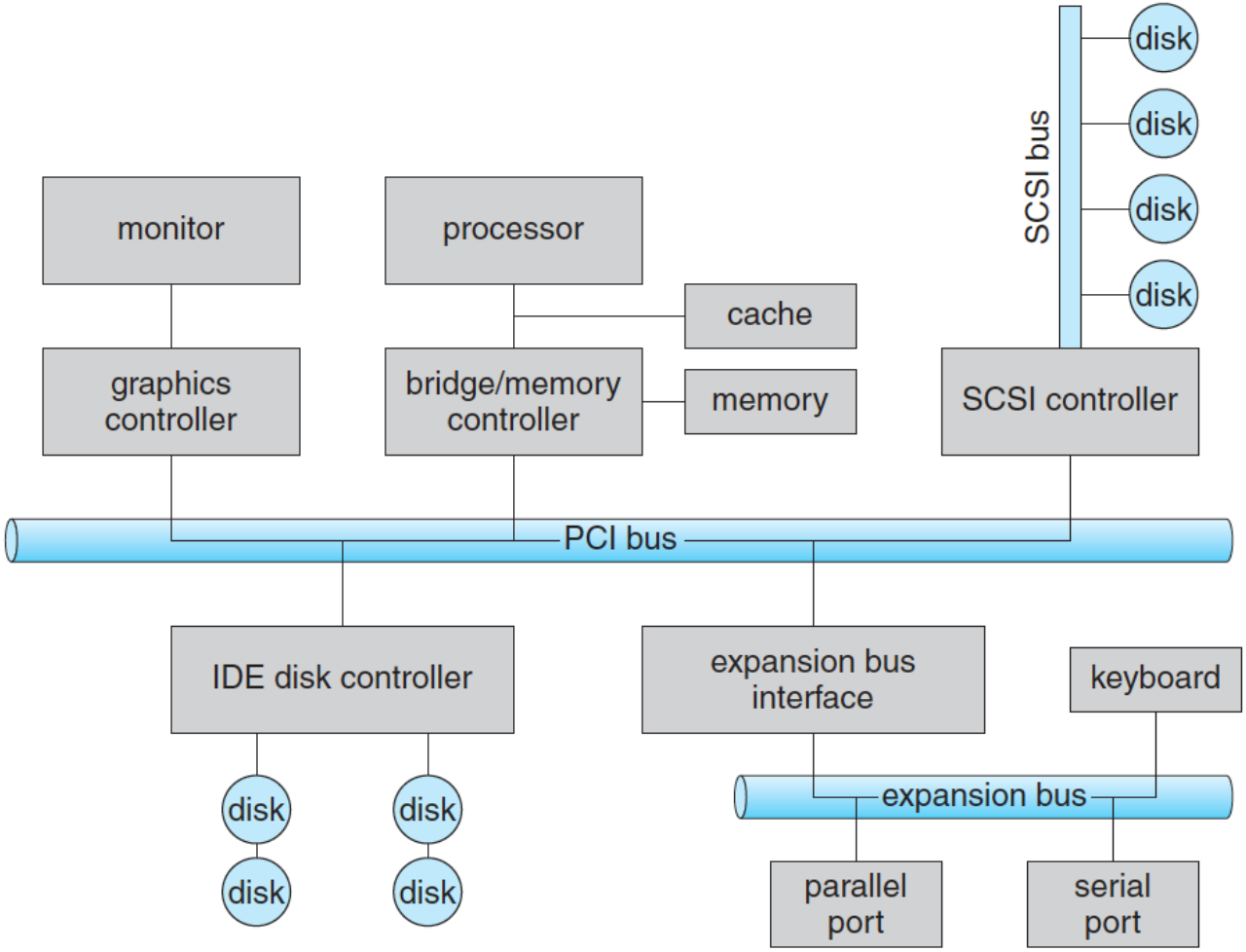
4) аудіо/відео пристрої;

5) інші пристрої; приклад: таймери (генератори переривань) та ін.

01.3. Фізична організація пристроїв вводу/виводу

- Контролер (controller) — це пристрій який забезпечує підключення та упраління зв’язком ПВВ з ЕОМ на апаратному рівні.
- Складність та спосіб апаратної реалізації контролера залежать від складності протоколу взаємодії з ПВВ.
- Способи фізичного з’єднання ПВВ з ЕОМ:
  1) окремий порт вводу/виводу (I/O port) → забезпечує під’єднання одного ПВВ;
  2) шина вводу/виводу (I/O bus) → забезпечує під’єднання декількох ПВВ.

01.4. Фізична організація пристроїв вводу/виводу



Типова схема інтерфейсів підключення ПВВ (портів та шин)

01.5. Фізична організація пристроїв вводу/виводу

PCI → Peripheral Component Interconnect

PCIe → PCI Express (пропускна здатність до 16GB в сек.)

SCSI → Small Computer System Interface

IDE → Integrated Drive Electronics

SATA → Serial Advanced Technology Attachment

01.6. Фізична організація пристроїв вводу/виводу

- Для програмного управління обміном даними з ПВВ використовується програмна модель ПВВ.
- Програмна модель ПВВ: набір регістрів різного функціонального призначення.
- Типовий набір регістрів: дані, стан пристрою, команди управління.

01.7. Фізична організація пристроїв вводу/виводу

Способи програмного управління обміном даними з ПВВ (способи доступу до регістрів програмної моделі ПВВ):

1) використання спеціальних машинних команд запису/читання в ПВВ з вказанням адреси порту вводу/виводу → простір портів вводу/виводу;

2) відображення вводу/виводу в адресний простір пам’яті (memory-mapped I/O) → використання стандартних машинних команд запису/читання в пам’ять.

01.8. Фізична організація пристроїв вводу/виводу

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

Приклад адресування портів вводу/виводу в ПК

01.9. Фізична організація пристроїв вводу/виводу

Порт вводу/виводу (I/O port) → типова програмна модель містить 4-ри регістра:

1) регістр для отримання даних від ПВВ (data-in register);

2) регістр для передачі даних у ПВВ (data-out register);

3) регістр стану ПВВ (status register), окремі біти якого вказують на стан готовності ПВВ до читання/запису, статус виконання команд, виникнення помилок;

4) регістр управління (control register) для передачі ПВВ команди на виконання або зміни режиму його роботи.

01.10. Фізична організація пристроїв вводу/виводу

Способи встановлення та підтвердження зв'язку з ПВВ (handshaking) для виконання операцій вводу/виводу:

1. Опитування ПВВ (polling): готовність ПВВ до роботи визначається шляхом періодичної перевірки біту зайнятості (busy bit) в регістрі стану ПВВ → цикл опитування (polling loop): 1) з зупинками на виконання інших обчислень; 2) постійне опитування (busy-wait polling).

2. Переривання (interrupts) від контролера ПВВ: контролер ПВВ повідомляє про готовність ПВВ до роботи шляхом генерування відповідного переривання → цикл вводу/виводу керований перериванням (interrupt-driven I/O cycle).

01.11. Фізична організація пристроїв вводу/виводу

Два способа взаємодії з ПВВ з точки зору участі CPU:

1. Програмований ввід/вивід (programmed input/output) → процесор задіяний як елемент «маршруту» даних між ПВВ і оперативною пам’яттю;

2. Ввід/вивід без участі процесора: Direct Memory Access (DMA) → прямий доступ до пам’яті, яким керує контролер DMA.

02.1. Організація програмного забезпечення вводу/виводу

Основна ідея: розбиття на декілька рівнів, кожний з яких приховує «зайві» деталі організації вводу/виводу нижніх рівнів.

Мета:

1) Забезпечити взаємну незалежність ПЗ від типу пристроя і навпаки пристрою від ПЗ (plug-and-play: принцип взаємозамінних частин).

2) Забезпечити «зручність» програмування (в рамках подолання семантичного розриву).

02.2. Організація програмного забезпечення вводу/виводу

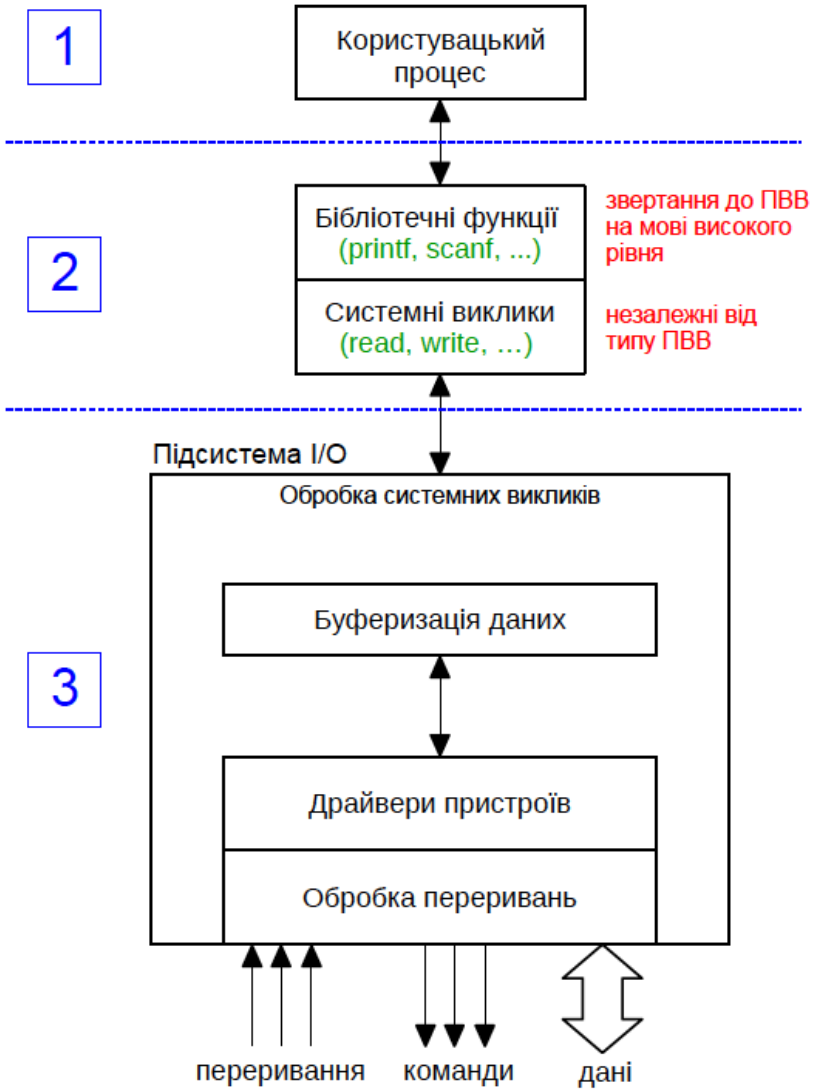
aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read–write	CD-ROM graphics controller disk

Основні характеристики ПВВ, які породжують розмаїття способів програмного управління

02.3. Організація програмного забезпечення вводу/виводу

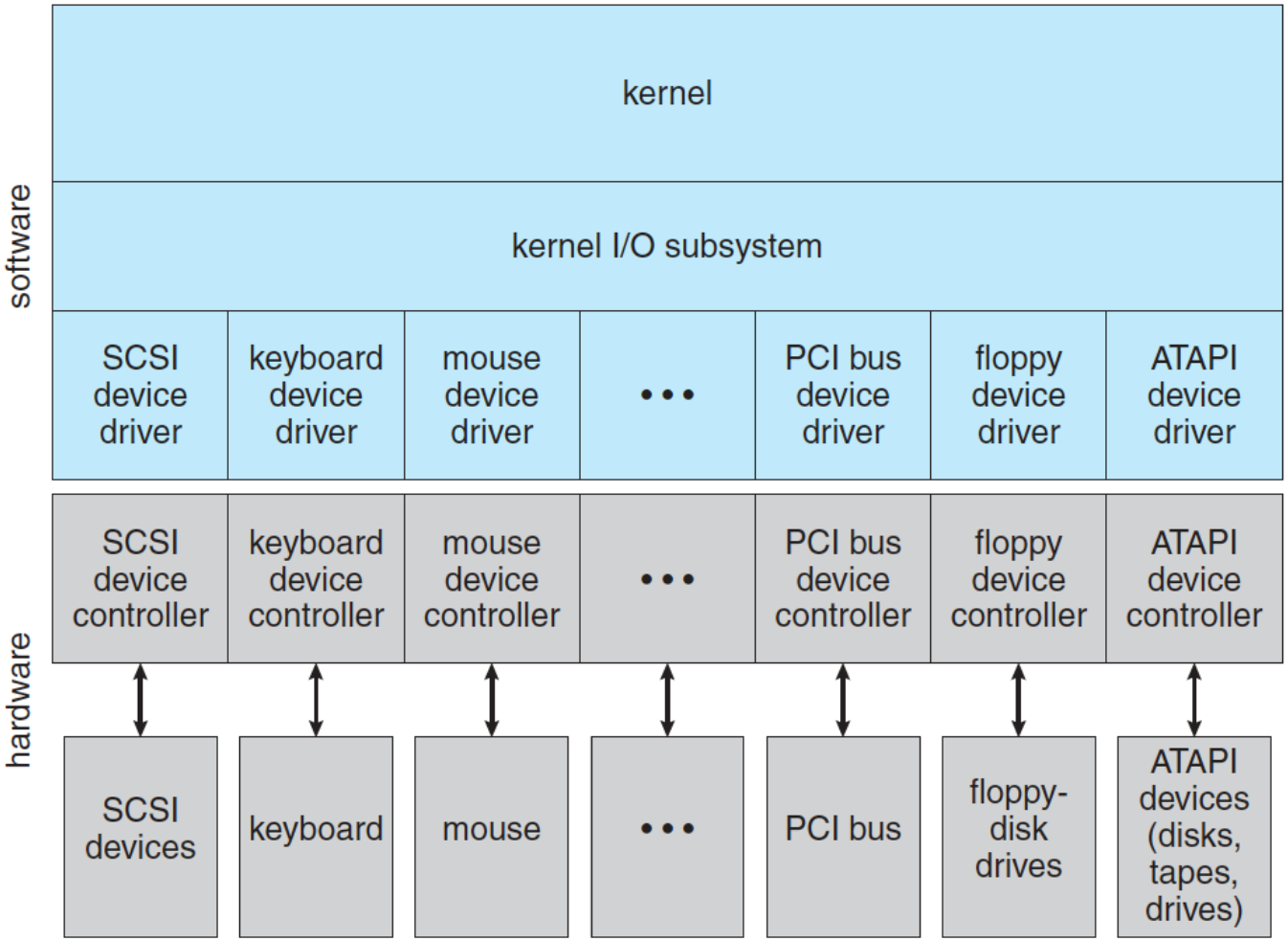
- Обробка помилок → втрати часу → як правило, обробка помилок покладається на контролер ПВВ.
- Способи обміну даними з ПВВ:
  1) передача даних з блокуванням (синхронна);
  2) передача даних без блокування (асинхронна).
- Більшість операцій вводу/виводу на фізичному рівні є асинхронними. Якщо це потрібно, ОС представляє їх для користувацького процесу як синхронні.

02.4. Організація програмного забезпечення вводу/виводу



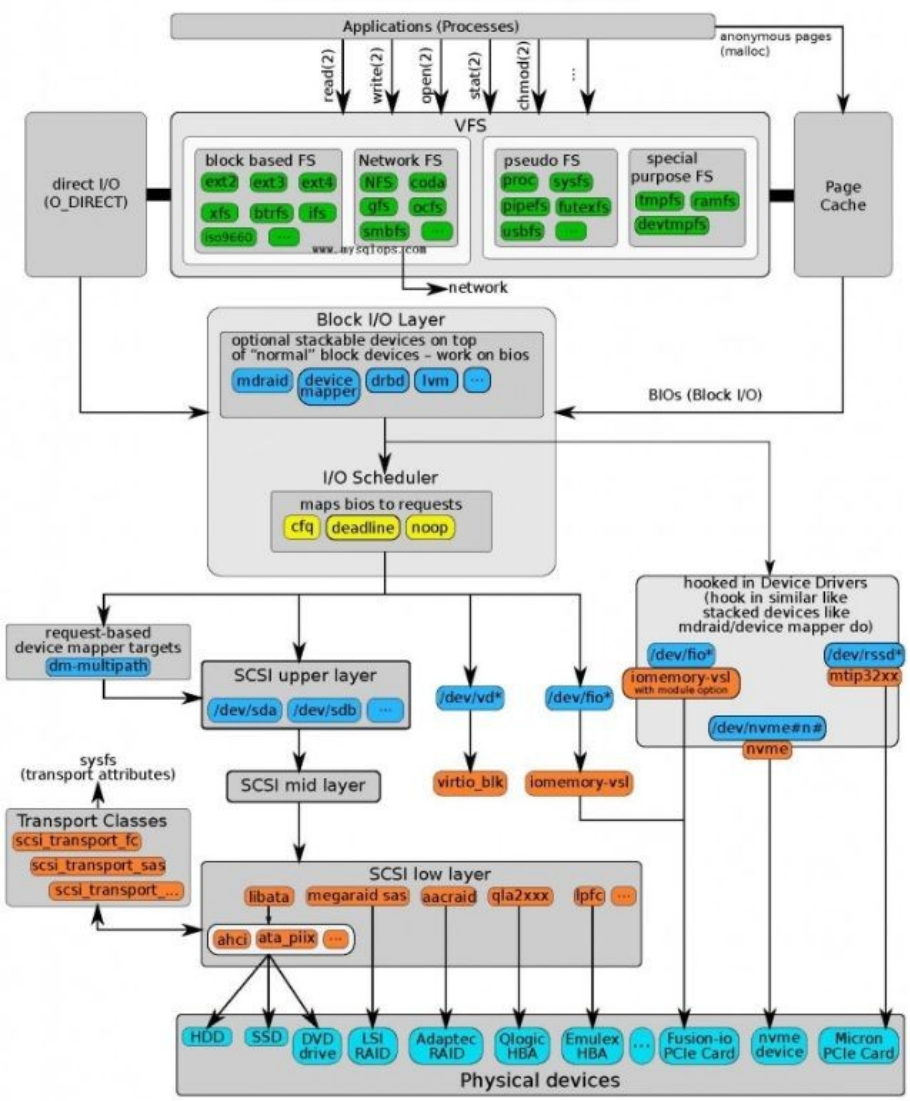
Узагальнена схема багаторівневої організації програмного забезпечення вводу/виводу

02.5. Організація програмного забезпечення вводу/виводу



Приклад структури підсистеми вводу/виводу (I/O subsystem)

02.6. Організація програмного забезпечення вводу/виводу



Підсистема вводу/виводу ОС Linux

02.7. Організація програмного забезпечення вводу/виводу

- Абстракції «незалежності»:
  1) Концепція потоків (streams) вводу/виводу.
  2) Відображення пристроїв вводу/виводу на файлову систему → файл пристрою (UNIX).

- Завдання: охопити усе «розмаїття» ПВВ за допомогою ієрархії узагальнених характеристик та приховування відмінностей між ПВВ на відповідних рівнях ієрархії.

02.8. Організація програмного забезпечення вводу/виводу

Сервіси ядра ОС, які входять до підсистеми вводу/виводу:

1. Планування операцій вводу/виводу (I/O scheduling) → визначення оптимального порядку виконання операцій вводу/виводу.

2. Буферизація (Buffering) → узгодження швидкостей передачі даних, форматів даних, забезпечення безпеки інформації → double buffering.

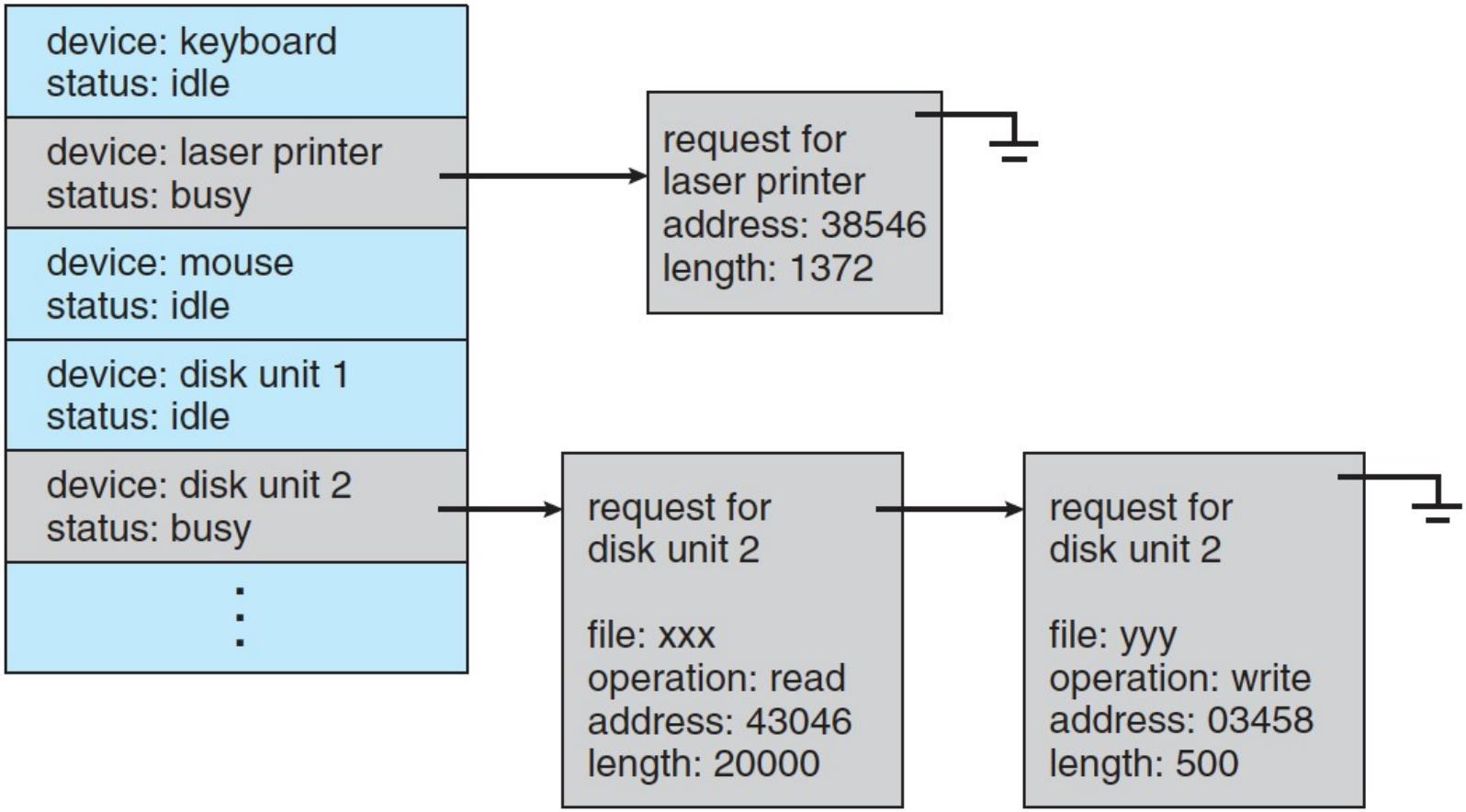
02.9. Організація програмного забезпечення вводу/виводу

3. Кешування (Caching) → підтримка ієрархії пристроїв пам'яті в складі підсистеми вводу/виводу.

4. Спулінг → Spooling [and Device Reservation].

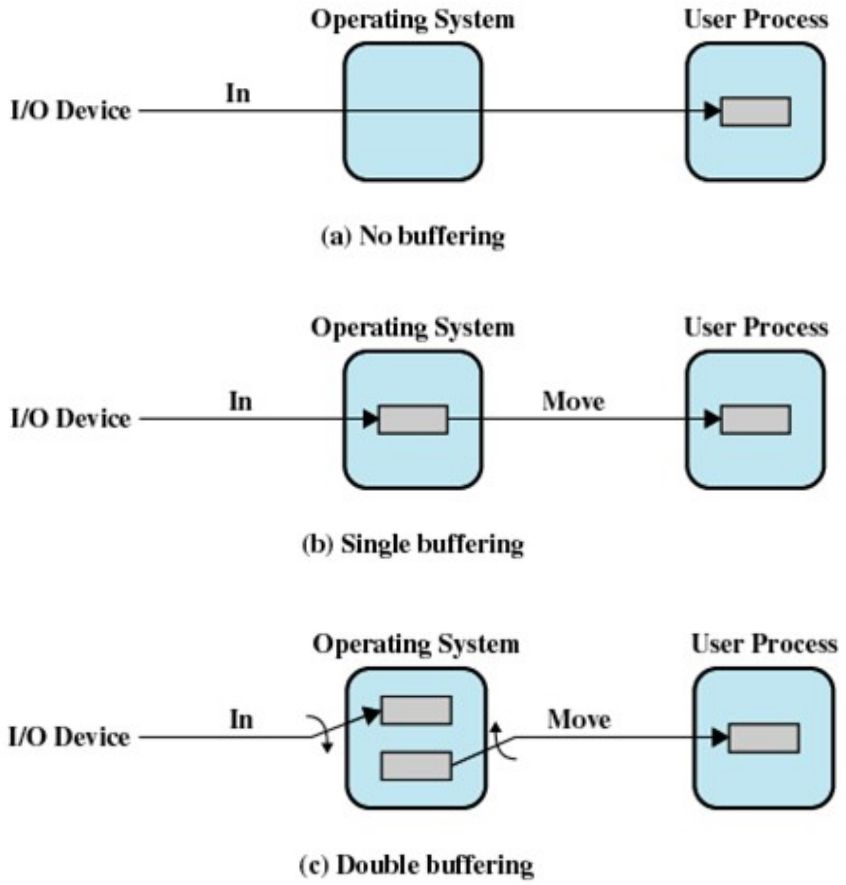
5. Обробка помилок (Error Handling), захист від помилкових операцій вводу/виводу (I/O Protection), моніторинг стану ПВВ (Device-status monitoring).

02.10. Організація програмного забезпечення вводу/виводу



Приклад таблиці станів ПВВ (device-status table), що використовуються для планування операцій вводу/виводу

02.11. Організація програмного забезпечення вводу/виводу



Порівняння різних способів буферизації даних

02.12. Організація програмного забезпечення вводу/виводу

Проблема «приховування» у світлі багатозадачності: для ПВВ з одночасним доступом виникає проблема некоректного «змагання» паралельних процесів за доступ до ПВВ.

Варіанти рішення:

1. Механізм монопольного захоплення ПВВ процесом. Недолік: координація доступу до ПВВ має бути реалізована в коді користувацьких процесів, які до нього доступаються.

02.13. Організація програмного забезпечення вводу/виводу

2. Спулінг (spooling) → використовується спеціальний процес-монітор ПВВ, який вирішує проблему одночасного доступу за допомогою
1) черги (=каталог спулінгу) звертань процесів до пристрою та
2) управління доступом до ПВВ процесів згідно розташування їх звертань у черзі.

Перевага: від користувацьких процесів приховується факт «змагання» за доступ до ПВВ.

Назва: SPOOL → Simultaneous Peripheral Operations On-Line [Tanenbaum].

Приклад: черга файлів на роздрук → служба spoolsv (MS Windows).

03.1. Обробка переривань

Дві ідеології побудови систем:

1) Офісні/лабораторні системи → максимальна передбачуваність → намагання зменшити кількість виникаючих переривань від ПВВ та локалізувати їх в «надрах» ОС.

2) Системи реального часу → непередбачуваність → обробка переривань від ПВВ (=об'єкти управління та контролю) є центральним моментом роботи системи (взаємодія з зовнішнім світом: робототехніка, технологічні процеси і т.п.).

03.2. Обробка переривань

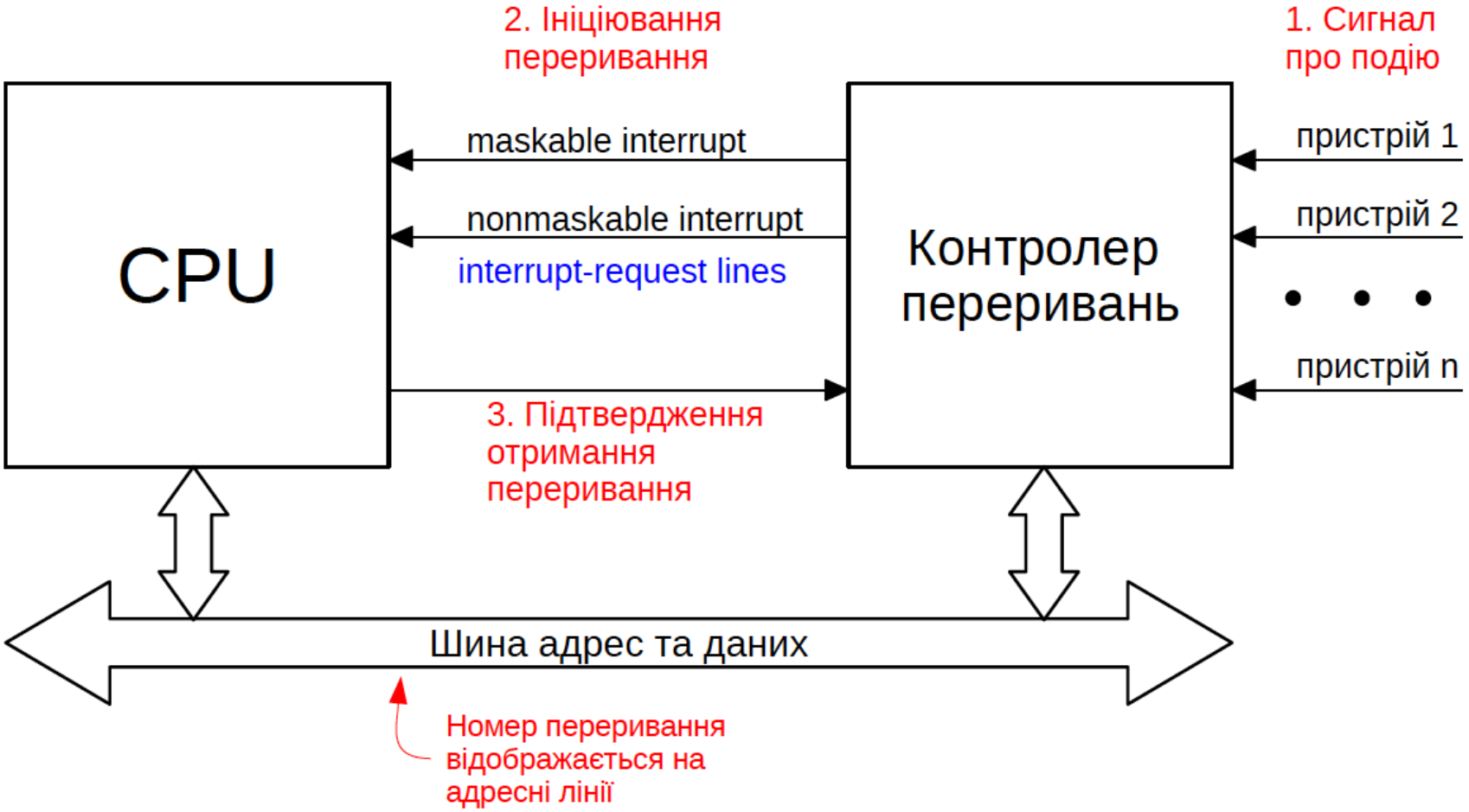


Схема обробки переривань від зовнішніх пристроїв

03.3. Обробка переривань

1. Управління процесом генерування та обробки переривання на апратному рівні здійснює контролер переривань (interrupt-controller hardware): «посередник» між CPU і зовнішніми пристроями, які генерують сигнали про свій стан та результати виконання команд.

2. Interrupt-request lines:
1) nonmaskable — не можуть бути «проігноровані» CPU (наприклад, збій у роботі оперативної пам'яті),
2) maskable — CPU може відкласти обробку переривання до моменту завершення виконання «важливої» послідовності команд.

03.4. Обробка переривань

3. Обробник переривання (interrupt handler → interrupt-handler routine).

4. Вектор переривань (interrupts vector) → таблиця адрес обробників переривань.

5. Рівні пріоритетів переривань (interrupt priority levels) дозволяють класифікувати всі переривання за їх «важливістю». Це, наприклад, дозволяє CPU переривати роботу обробника менш важливого переривання обробкою більш важливого переривання.

5. Складна система пріоритетів переривань в сучасних системах → черга переривань на обробку.

03.5. Обробка переривань

Типи переривань:

1) апаратні: переривання від зовнішніх пристроїв (device interrupts);

2) «виключні» невідкладні ситуації (exceptions), для опрацювання яких використовується механізм переривань (наприклад, ділення на 0 або помилка доступу до пам'яті).

3) програмні переривання (software interrupt = trap): спосіб реалізації обробки системних викликів за допомогою механізму переривань.

03.6. Обробка переривань

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

Приклад таблиці переривань (event-vector table) Intel Pentium

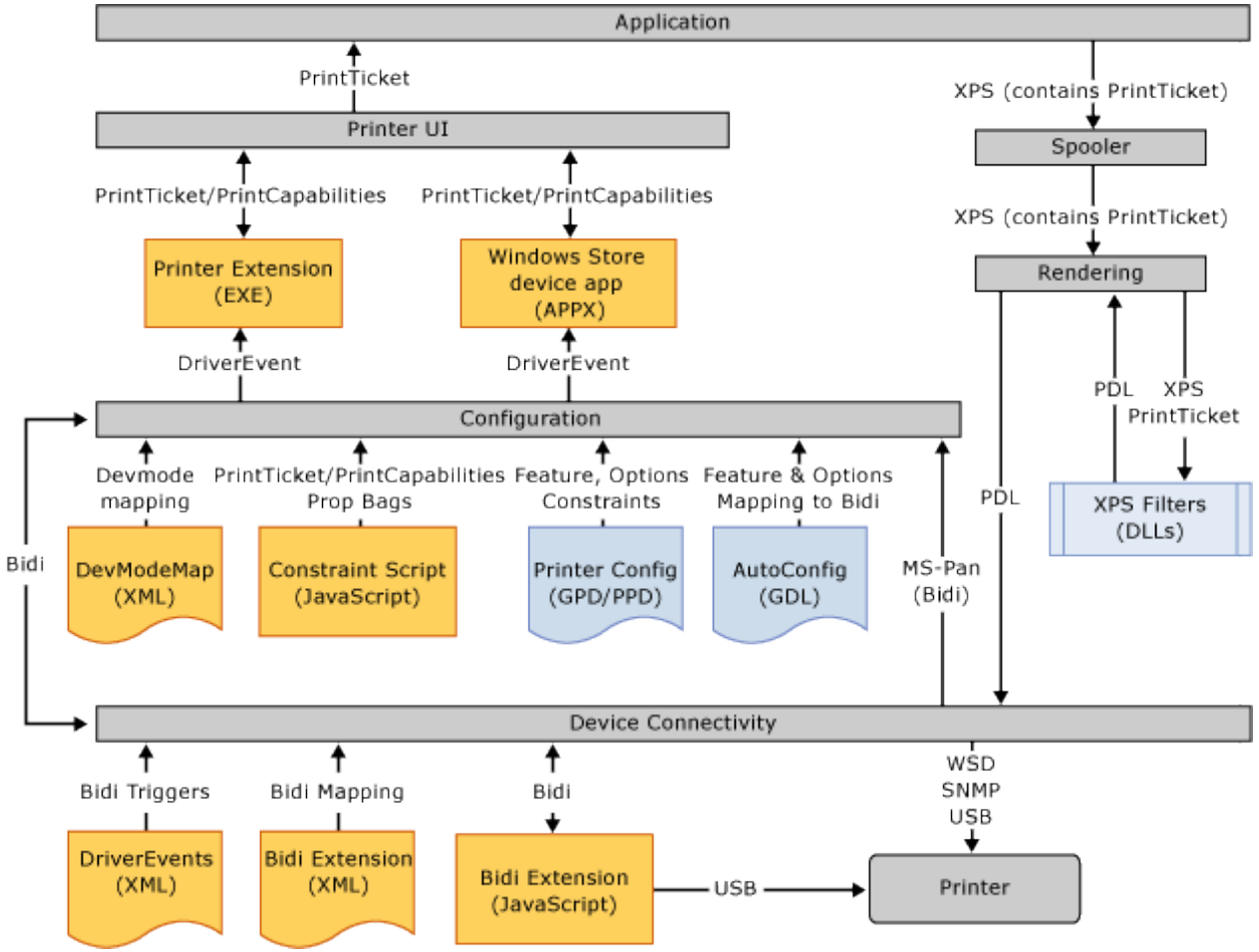
04.1. Драйвер пристрою

- Драйвер пристрою — це програма, яка реалізує програмну модель ПВВ та інкапсулює низькорівневі особливості його роботи.
- Драйвер пристрою містить весь залежний від пристрою код: таким чином драйвер приховує «технічні» деталі організації вводу/виводу.
- Драйвер виконує 1) перетворення форматів даних; 2) необхідні операції по роботі з контролером, згідно функціонального призначення та логіки роботи ПВВ (в тому числі операції обміну даними з блокуванням або без блокування).

04.2. Драйвер пристрою

- Розробка драйверів: в рамках кожної ОС або сімейства ОС пропонується свій підхід та програмні інструменти для рорзробки драйвера, які відображають архітектурні особливості відповідних ОС.
- Приклади: Linux device model (Linux), Hardware Abstraction Layer (HAL) (Android), I/O Kit (Mac OS X, iOS), Windows Driver Kit (WDK).
- WDK - набір програмних інструментів для розробки драйверів під Widnows. Windows Driver Frameworks (WDF) - набір інструментів, бібліотек та шаблонів для розробки драйверів. Більш базовий фреймворк для розробки драйверів - Windows Driver Model (WDM).

04.3. Драйвер пристрою



V4 Printer Driver model: модель драйвера принтера (Microsoft Windows)

04.4. Драйвер пристрою

- Проблема розробки та стандартизації драйверів пристроїв (уніфікація та стандартизація програмного інтерфейсу: ОС-драйвер).
- Ініціатива Uniform Driver Interface: загальний переносимий програмний інтерфейс для драйверів пристроїв (переносимість між різними ОС без зміни коду драйвера).
- Згідно ініціативи Uniform Driver Interface для драйверів забезпечується інкапсулююче програмне середовище з добре продуманим інтерфейсом, яке «ізолює» драйвери від специфіки роботи ОС, апаратної платформи ЕОМ та особливостей шини вводу/виводу (I/O bus). Такий підхід зокрема дозволяє зробити розробку драйвера повністю незалежною від розробки ОС.

05.1. Концепція контекстно-залежних обчислень (context-aware computing)

1. Bill N. Schilit, Norman Adams, Roy Want, Context-Aware Computing Applications, in Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications, December 8-9, 1994.

2. Ідея: програмне забезпечення персонального мобільного обчислювального пристрою відслідковує стан оточення (context) користувача та реагує на зміни в ньому.

3. Концепцію контекстно-залежних обчислень можна розглядати як наступний етап еволюції способів взаємодії користувача з обчислювачем за допомогою ПВВ, який потребує розробки нових технологій управління вводом/виводом.

05.2. Концепція контекстно-залежних обчислень (context-aware computing)

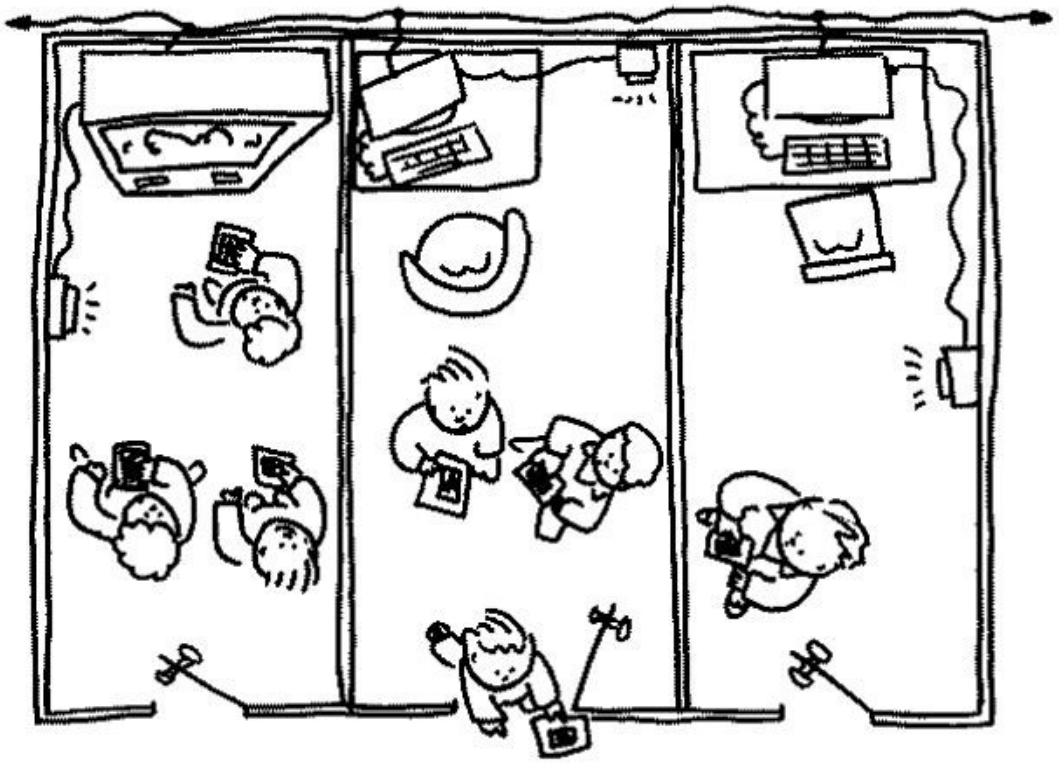


Figure 1: A Context-Aware Computing System (PARCTAB)

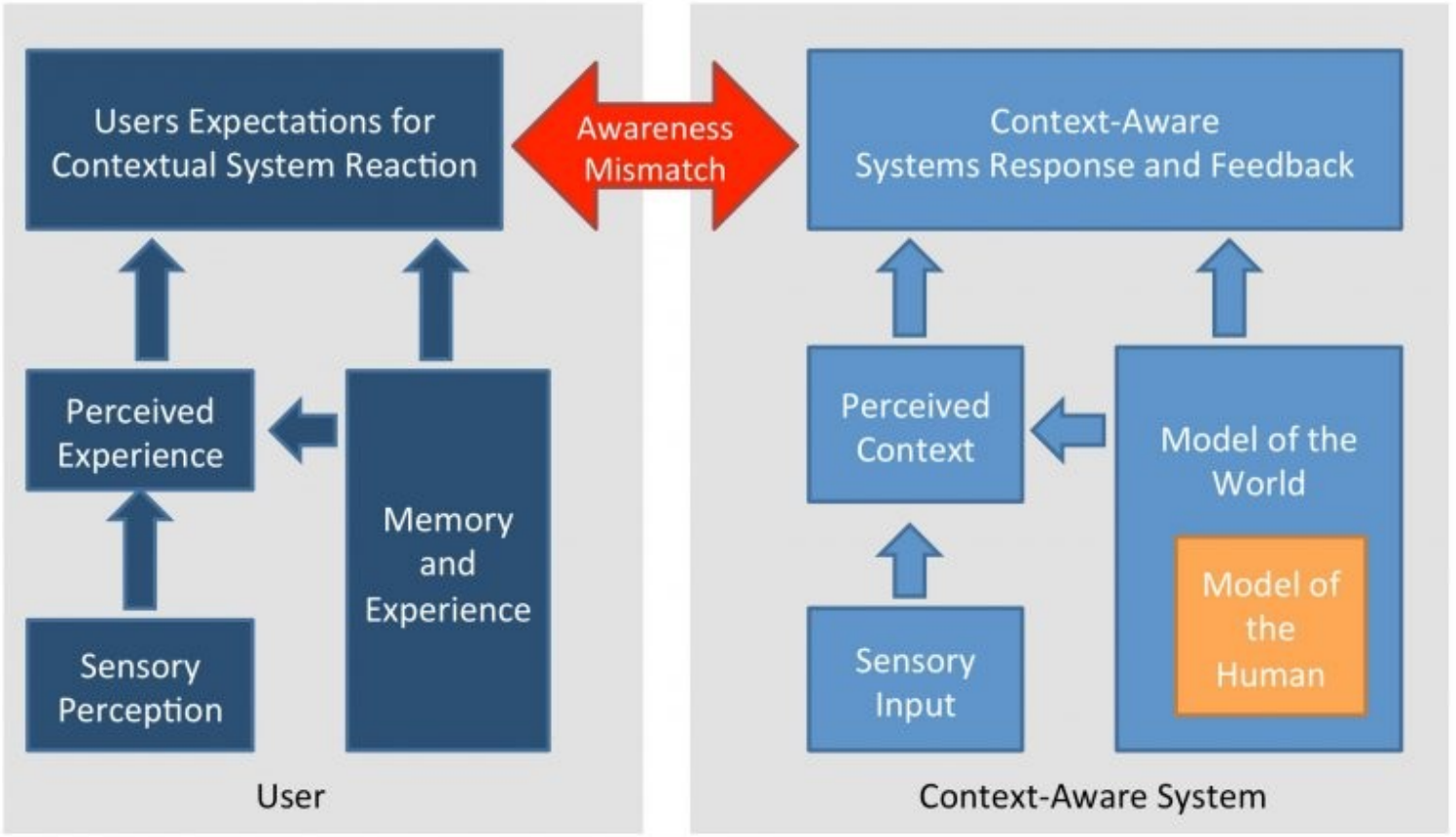
Приклад різних контекстів користувача

05.3. Концепція контекстно-залежних обчислень (context-aware computing)

4. Що входить у контекст користувача? → проблема моделювання контексту та наповнення моделі відповідними даними.

5. Приклад структури контексту користувача [Schilit et al.]:
1) де знаходиться користувач? → user location
2) хто знаходиться поруч з користувачем? → social situation
3) які апаратні ресурси присутні в оточенні користувача?

05.4. Концепція контекстно-залежних обчислень (context-aware computing)



Модель формування контексту, який відповідає сподіванням користувача

05.5. Концепція контекстно-залежних обчислень (context-aware computing)

6. Проблеми:

1) точність визначення контексту;

2) повнота визначення контексту;

3) правильність розпізнавання контексту;

4) визначення пріоритетів змін (які зміни контексту більш важливі за інші?);

5) врахування швидкості зміни контексту.

05.6. Концепція контекстно-залежних обчислень (context-aware computing)

7. Класифікація способів реагування на зміни у контексті [Schilit et al.]:

	Реагування на зміну контексту за участю користувача (manual)	Реагування на зміну контексту без участі користувача (automatic)
«Інформаційно-програмне» реагування	proximate selection & contextual information	automatic contextual reconfiguration
«Командне» реагування	contextual commands	context-triggered actions

05.7. Концепція контекстно-залежних обчислень (context-aware computing)

1) Proximate selection → для взаємодії з користувачем обираються пристрої, які знаходяться на зручній відстані від нього (користувач обирає один з них).

2) Contextual information → видача інформації користувачу з врахуванням контексту (наприклад, відповідь на пошуковий запит може залежати від поточного місцезнаходження користувача).

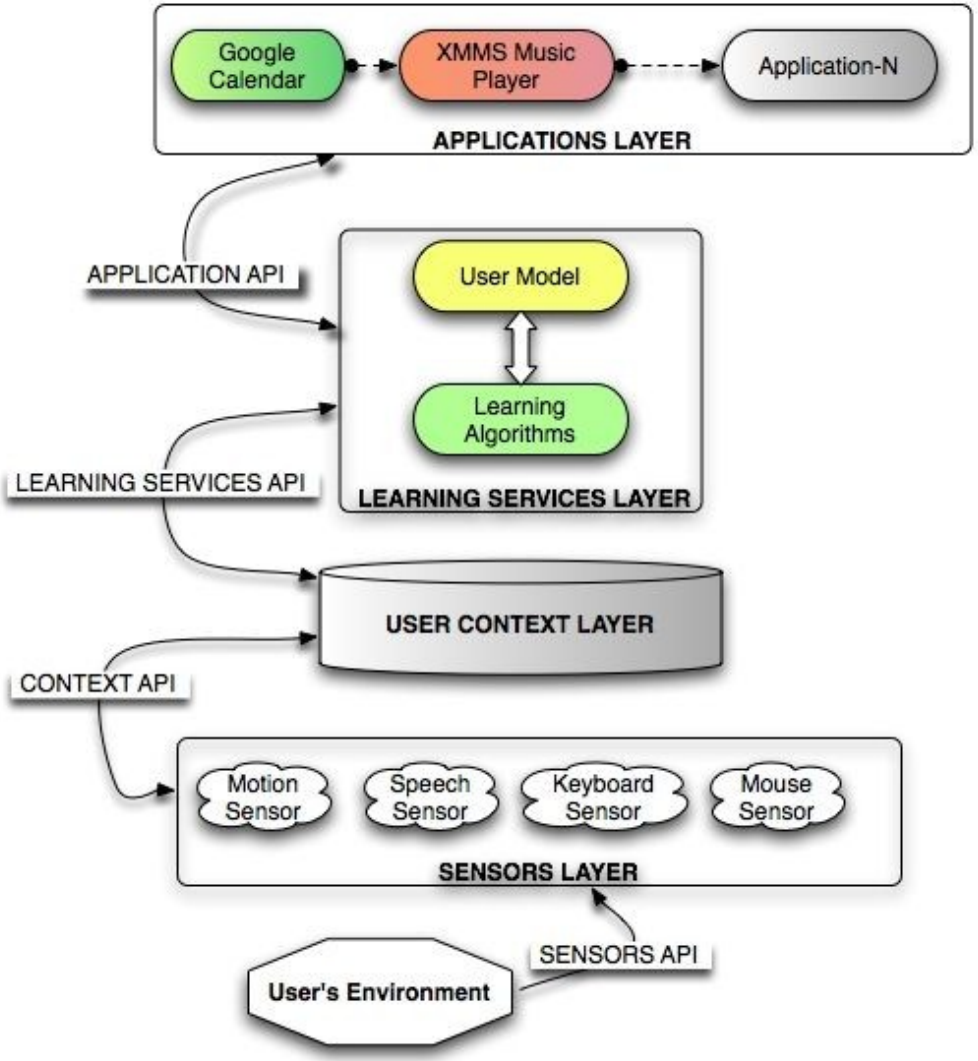
3) Contextual commands → при виконанні команд користувача враховується контекст (наприклад, в разі команди «друк» обирається найближчий до користувача принтер).

05.8. Концепція контекстно-залежних обчислень (context-aware computing)

4) Automatic Contextual Reconfiguration → програмне забезпечення змінює свій режим роботи та/або свою конфігурацію в залежності від контексту (наприклад, у випадку зборів учасників проекту автоматично завантажується віртуальна whiteboard зі збереженою історією попередніх зборів).

5) Context-Triggered Actions → автоматичне виконання команд в разі детектування в контексті відповідних вхідних умов (на основі заданих правил IF-THEN).

05.9. Концепція контекстно-залежних обчислень (context-aware computing)



Приклад архітектури контекстно-залежних обчислень з використанням алгоритмів машинного навчання (machine learning)