今回は、OSに対して指示を与えるコマンドや画面の使われ方、日本語処理についてです。
教科書では、第2章のオペレーティングシステムのユーザインタフェースと第16章のオペレーティングシステムと標準化が対応します。

オペレーティングシステム
(2024年 第12回)

OSのユーザインタフェースについて

前回の課題について

詳細は「第10回小課題について.pdf」をみてください

（1）AさんがBさんの公開鍵で暗号化した電子メールを、BさんとCさんに送信した結果のうち、適切なものはどれか。
ここで、Aさん、Bさん、Cさんのそれぞれの公開鍵は3人全員がもち、それぞれの秘密鍵は本人だけがもっているものとする。

ア　暗号化された電子メールを、Bさんだけが、Aさんの公開鍵で復号できる
イ　暗号化された電子メールを、Bさんだけが、自身の秘密鍵で復号できる
ウ　暗号化された電子メールを、Bさんも、Cさんも、Bさんの公開鍵で復号できる
エ　暗号化された電子メールを、Bさんも、Cさんも、自身の秘密鍵で復号できる

公開鍵暗号方式では暗号化するための公開鍵と復号するための秘密鍵のペアで暗号化と復号を行います。Bさんの公開鍵を使って暗号化したので、それを復号できる唯一の鍵はBさんの秘密鍵です。したがって「イ」が適切であると考えられます。

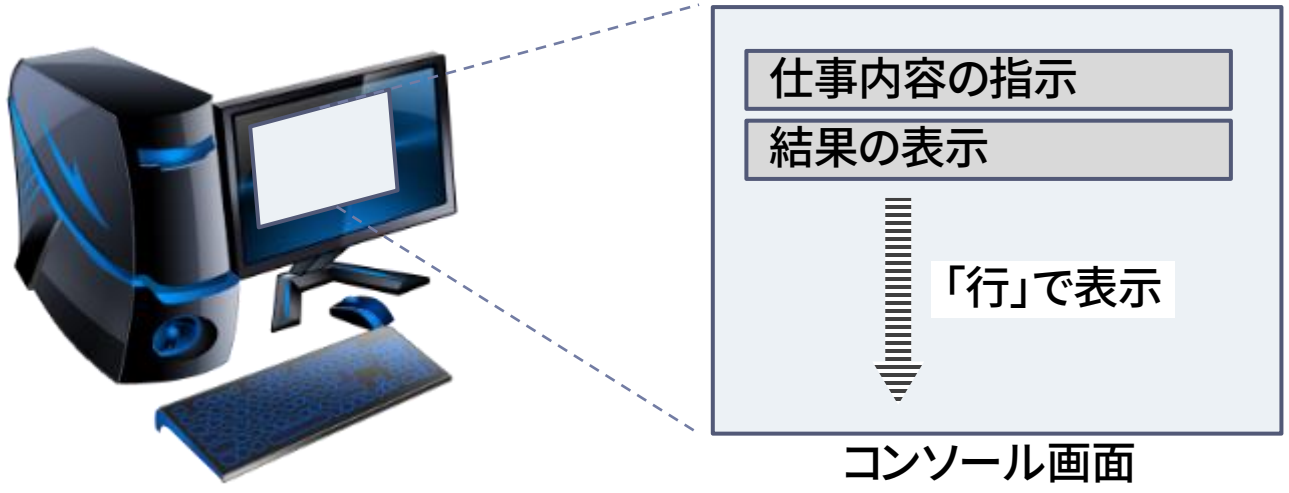
（2）次のpp.11-12 に示すアクセス制御方式をもつファイルシステムにおいて、ファイルに対して以下のアクセス権の条件すべてを満足する設定（アクセス権の設定状況）を示せ

- 全ての利用者が実行できる
- 所有者、および所有グループの利用者だけが読み出しできる
- 所有者だけが書き込みできる

所有者は 読出し、書込み、実行 できるので rwx、所有グループの利用者は 読み出し、実行できるので r-x、
その他の利用者は実行だけできるので --x となり、これらをつなげて rwxr-x--x となります。

オペレーティングシステムのユーザインタフェース

- ユーザから見たオペレーティングシステム



- オペレーティングシステムに仕事を依頼する
  - 仕事・操作の内容（指令、命令）：コマンド
  - 操作の対象やバリエーションの指定：パラメータ

```
$ date
Tue Jul  5 11:07:26 JST 2023
$
```

```
$ echo Hello
Hello
$
```

これまでは、ユーザのプログラムとOSとのインタフェースについてでした。今回は、ユーザとOSとのインタフェースの話になります。

現在のように画面に自由に描画して操作し、指示を与えるようになる前は、画面は「文字」だけからなる行で表示されて、指示を入力すると、処理結果が次々と行で表示されます。

ここでは、文字によるインタフェースを主として説明します。

dateやechoといったコマンドで指令を出し、右側の例ではHelloがechoの操作対象となって、この文字列がディスプレイ出力されることになります。（この画面例はLinuxのものです）

コマンドの例

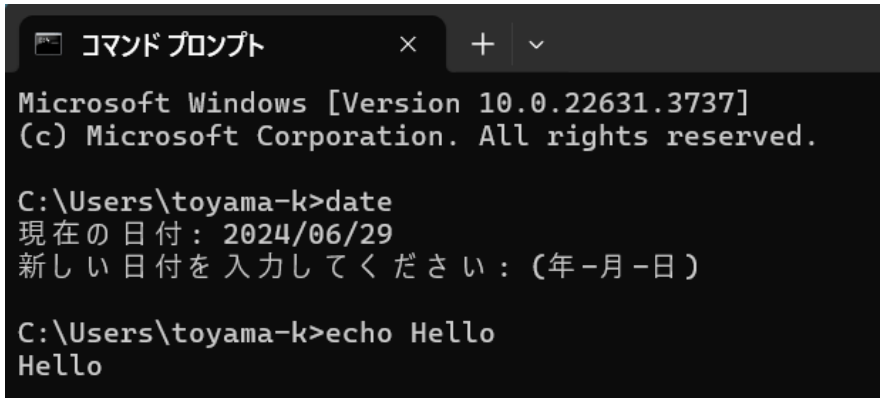
- ファイルを扱うもの
  - 複写、名前の一覧、内容の表示 など
- 装置を扱うもの
  - 初期化、使用方法の指定 など
- プログラムの実行に関するもの
- プログラムの管理に関するもの
- システムの情報を扱うもの
  - 日付、時刻表示 など
- その他

機　能	UNIX系	MS-DOS
ファイルをコピー	cp	copy
ファイルを移動	mv	move
ファイルの内容を表示	cat	type
ディレクトリの内容を表示	ls	dir
日付を表示	date	date
ファイルを削除	rm	del
画面に文字列を表示	echo	echo
ファイルの内容を比較	diff	fc
現在のディレクトリを変更	cd	cd

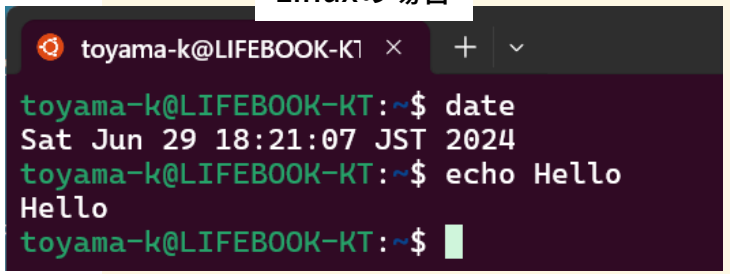
このようにコマンドには、ファイルを操作したり、プログラム(プロセス)を実行したり、日付を表示したりといったものがあります。そうでないのもありますが、dateやmoveなど想像の付くものもあります。

コマンドを使うのは、UNIX系のOSで多く見られることですが、Windowsでコマンド操作を行う画面はこのようになります。(コマンドの出力などもLinuxのものと同じだったり、少し違っていたりします)

Windowsでのコマンドの利用



Linuxの場合

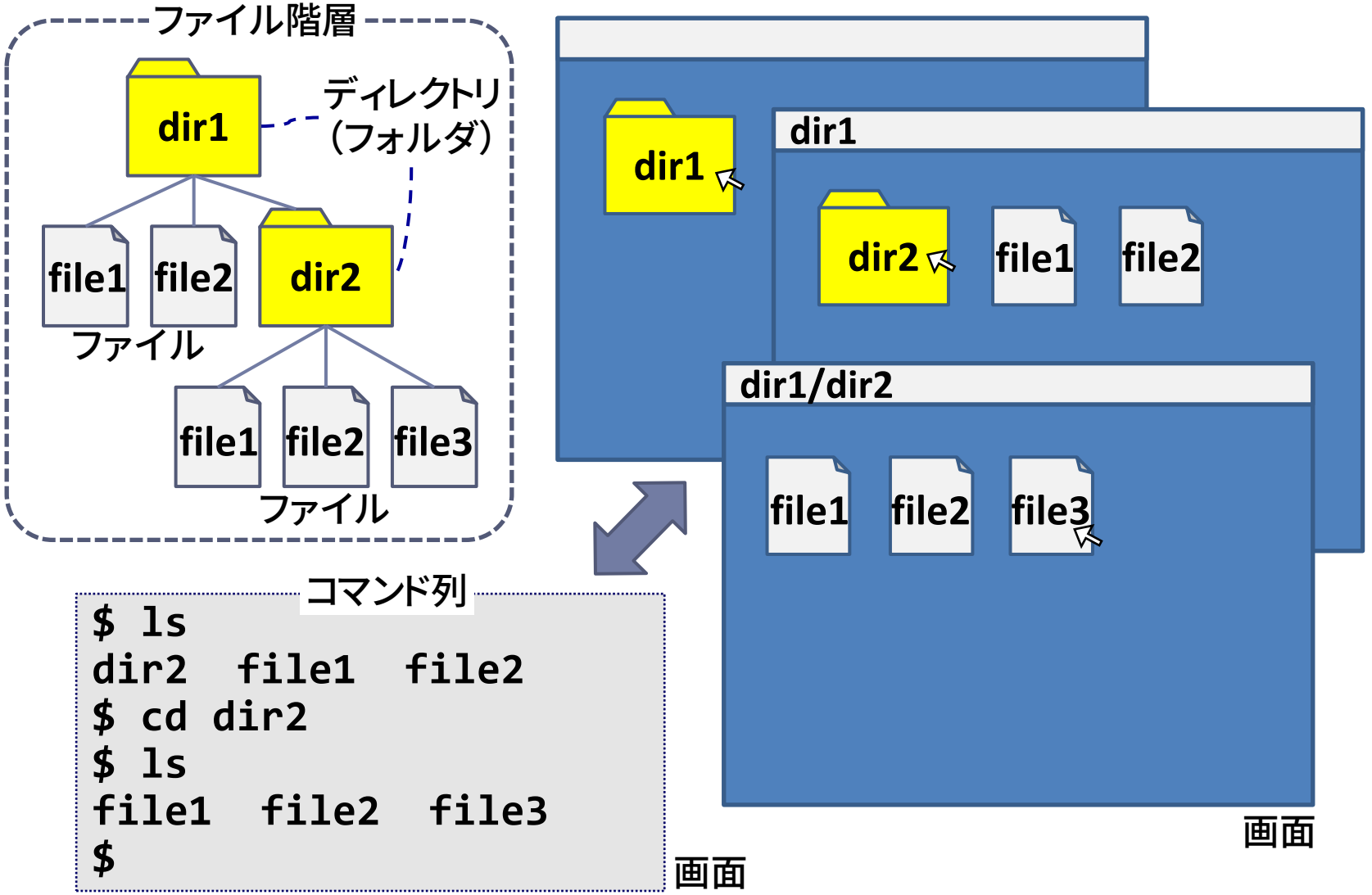


CUIとGUI

- 文字ベースのCUIとグラフィックスベースのGUIがある
  CUIを採用した代表的なOSは初期のUNIXやMS-DOS
- キャラクタユーザインタフェース（CUI）
  キーボードからコマンドを入力、ディスプレイに文字を表示することで出力
  - コマンドを知らないと何もできない。操作の結果が即座に画面に反映されない
  - 複雑な操作や連続した処理も比較的簡単に操作できる
  - UNIX系OSは元々CUIのシステム（現在はGUIも標準的）
    CUIの設計思想に基づいたソフトウェアが多く動いている
- グラフィカルユーザインタフェース（GUI）
  入力としてキーボードやマウスなどを用い、ディスプレイ上にグラフィカルな表示を出力として提示
  - 直観的で分かりやすい操作
  - 大量の処理や複雑な処理は苦手

ユーザインタフェースには、文字を入力して文字が出力されるようなCUIと、アイコンのような図形による表示や動作によって指示や出力を示すGUIとがあります。

GUIでは直感的で創造しやすい操作ができますが、CUIではコマンドを知っていないと操作できません。また、操作結果がすぐには画面に現れないこともあります。
その反面、コマンドの実行結果を画面に反映させることなく連続した処理を行うことができます。



コマンド列

```
$ ls
dir2  file1  file2
$ cd dir2
$ ls
file1  file2  file3
$
```

ファイルの操作をGUIとCUIで比較したものです。
（第9回のp.14「ディレクトリとパス」のファイル階層の例です）

GUIでは、フォルダのアイコン（たとえばdir2）をダブルクリックするとdir2に対応する画面が開いてその中が見れるという、操作者の直感にあったものですが、CUIではcd dir2などとしても画面上は何も変わらず、次にその中をリストするコマンドlsを入力して、初めて中にあるものが分かります。

ウィンドウシステム

- マルチウィンドウを実現し、それを用いたアプリケーション作成を支援するシステム
  - ウィンドウは一般に矩形の領域
  - ウィンドウの描画、移動、マウスやキーボードを使ったやり取りなど
- ユーザは複数のプログラム（プロセス）を同時に実行させることができ、各プログラムにはそれぞれにウィンドウが対応
- ウィンドウシステムの実現方法
  - オペレーティングシステムに組み込む
    - Windows や Mac OS（OS 9以前のもの）など、ウィンドウシステムがOS内に一体化して組み込まれている
  - プロセスとして実現し、複数のアプリケーションにサービスを提供するもの
    クライアントサーバモデル
    - ウィンドウサーバと呼ばれるプロセスが画面出力を制御
    - 他のアプリケーションはサーバにリクエストを出す

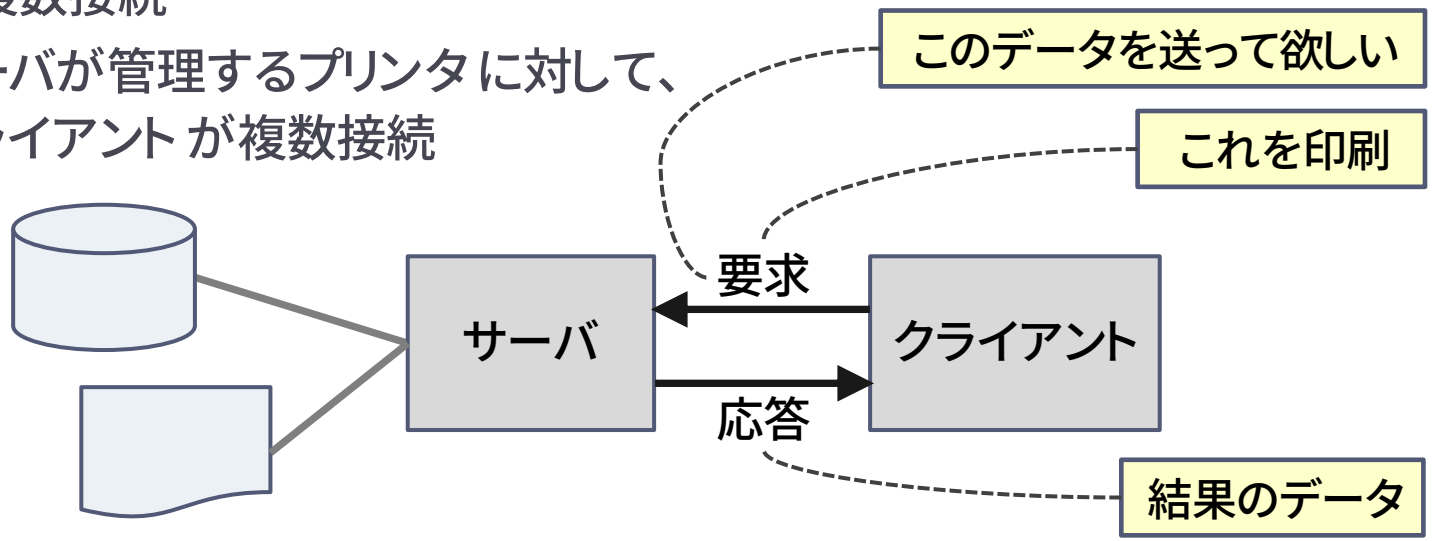
前ページのGUIの説明で、dir1のアイコンをダブルクリックすると、別の画面（ウィンドウ）が開くようなインターフェースを見ました。GUIでは、このようなウィンドウシステムが一般的に使用されます。

複数のウィンドウが同時に1つの画面上に表示されて、それぞれ別々に扱えるものをマルチウィンドウと呼びます。ウィンドウにはプログラム（プロセス）が対応し、ウィンドウの中でプログラムが動作しているイメージです。

ウィンドウをインタフェースとして利用するには、ウィンドウシステムをOSに一体化して組み込んで（それしか使わないで）いるWindowsのようなものと、Linuxなどのように、ウィンドウシステムがプロセスとして実装されるものがあります。
このようなものはクライアントサーバシステムになります。

クライアントサーバモデル

- 「サーバ」と「クライアント」に役割分担して運用する仕組み
  サーバはクライアントからの処理要求を受けて実際の処理を行い、その結果をクライアントに返すというモデル
  - 例 -
  - 共有データを置くサーバがあり、クライアントが複数接続
  - サーバが管理するプリンタに対して、クライアントが複数接続

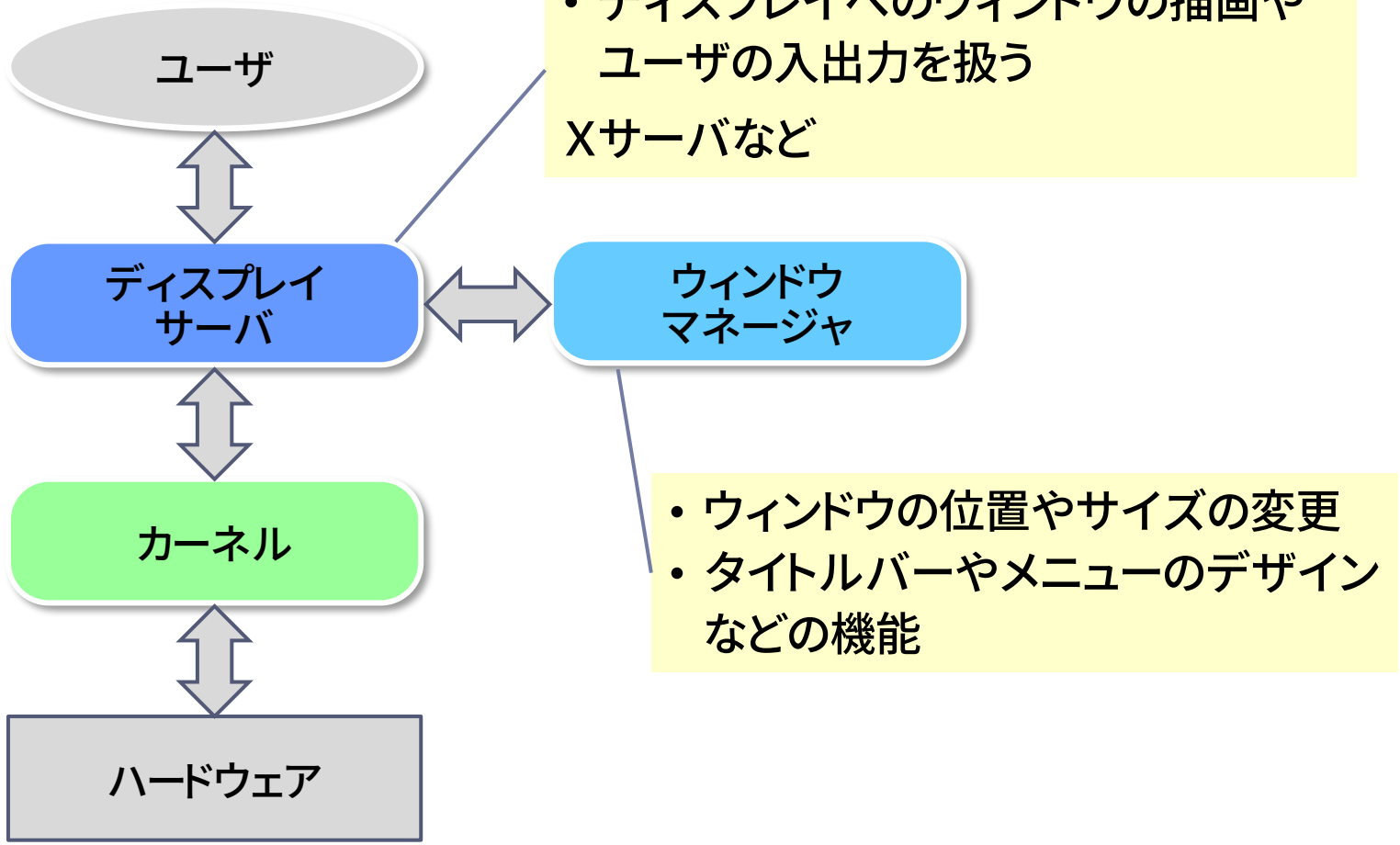


  - 多くの場合、サーバとクライアントは別のコンピュータ
  - 同じコンピュータ内で動作することもある

クライアントサーバシステムは、クライアントとサーバに役割を分けて、動作させる仕組みです。

クライアントからの処理要求（プリント依頼、データベース問合せなど）を受けて、サーバが処理結果を返すというものです。
多くの場合、サーバとクライアントはネットワークでつながれた別のコンピュータですが、同じコンピュータ内でサーバもクライアントも動作することもあります。ウィンドウシステムはそのような例です。

ウィンドウシステムの構成



- ディスプレイへのウィンドウの描画やユーザの入出力を扱う

Xサーバなど

- ウィンドウの位置やサイズの変更
- タイトルバーやメニューのデザインなどの機能

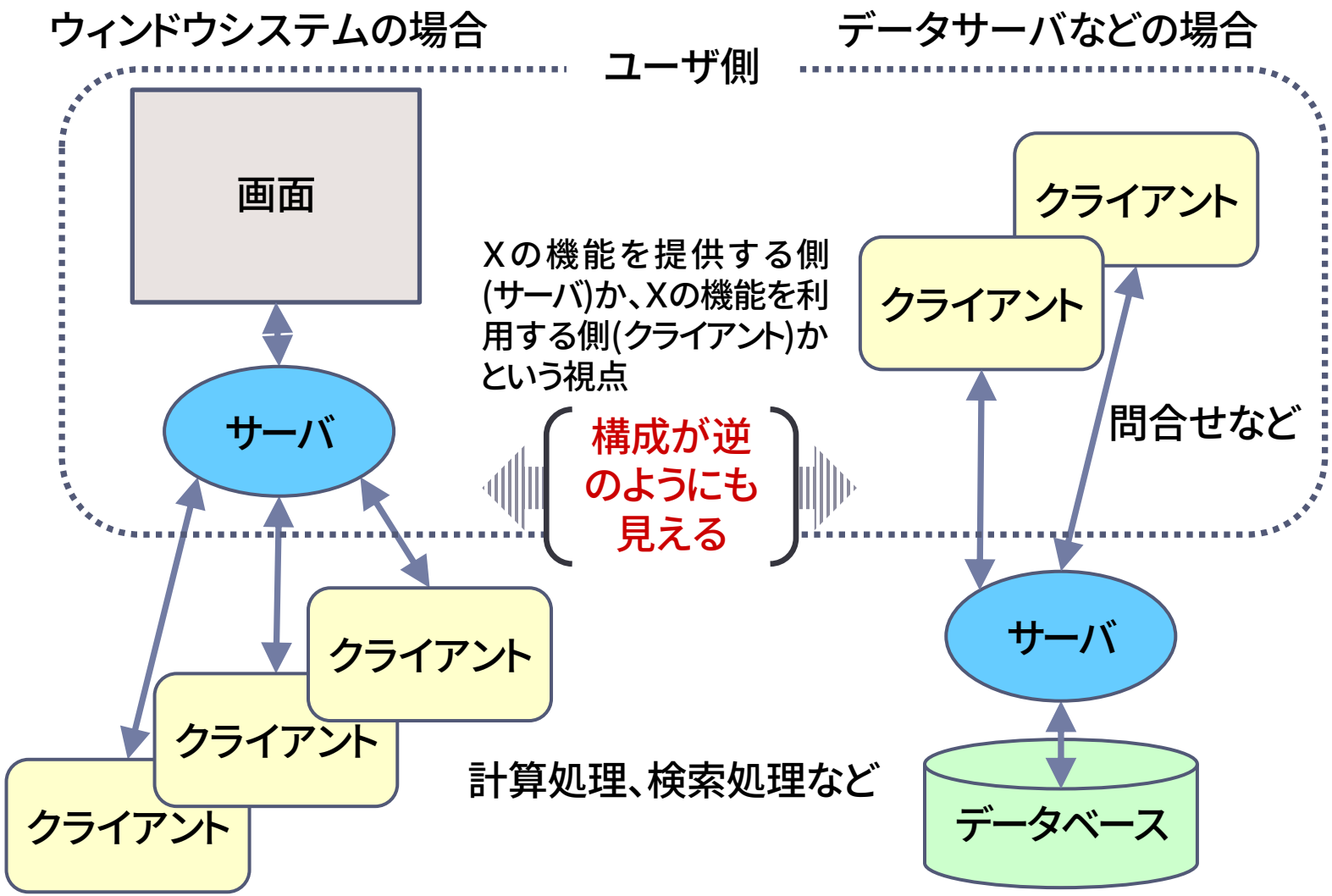
ウィンドウシステムでは、ディスプレイサーバがOSとユーザの間にあって、ディスプレイにウィンドウ描いて、入力や出力を扱います。
一方、ウィンドウのサイズを変える、画面のどの位置に出すか、どのような外観にするかなどは、ウィンドウマネージャが受け持つ構成になります。

Xサーバについては後で触れます。

ウィンドウマネージャはウィンドウの配置、 メニュー、 タイトルバーなどウィンドウの追加的な部品の描画を行います。ウィンドウの配置は、タイル型(画面を互いにオーバーラップしない領域に分割してウィンドウを表示)と、オーバーラッピング型(デスクトップ上のウィンドウを「重ねて」配置して表示)、とに大きく分かれます。
オーバーラッピング型では、さらに、スタック型(ウィンドウを1つずつ、下(画面の奥)から上(画面の手前)へと順番に描画)とコンポジット型(個々のウィンドウのイメージを合成することでスクリーンのイメージを生成して表示用メモリにその結果を書き込む)のタイプがあります。

Windows Vista以降やMac OS Xのウィンドウマネージャはコンポジット型で、3Dやアニメーションなどの視覚効果が可能なものです。

注意

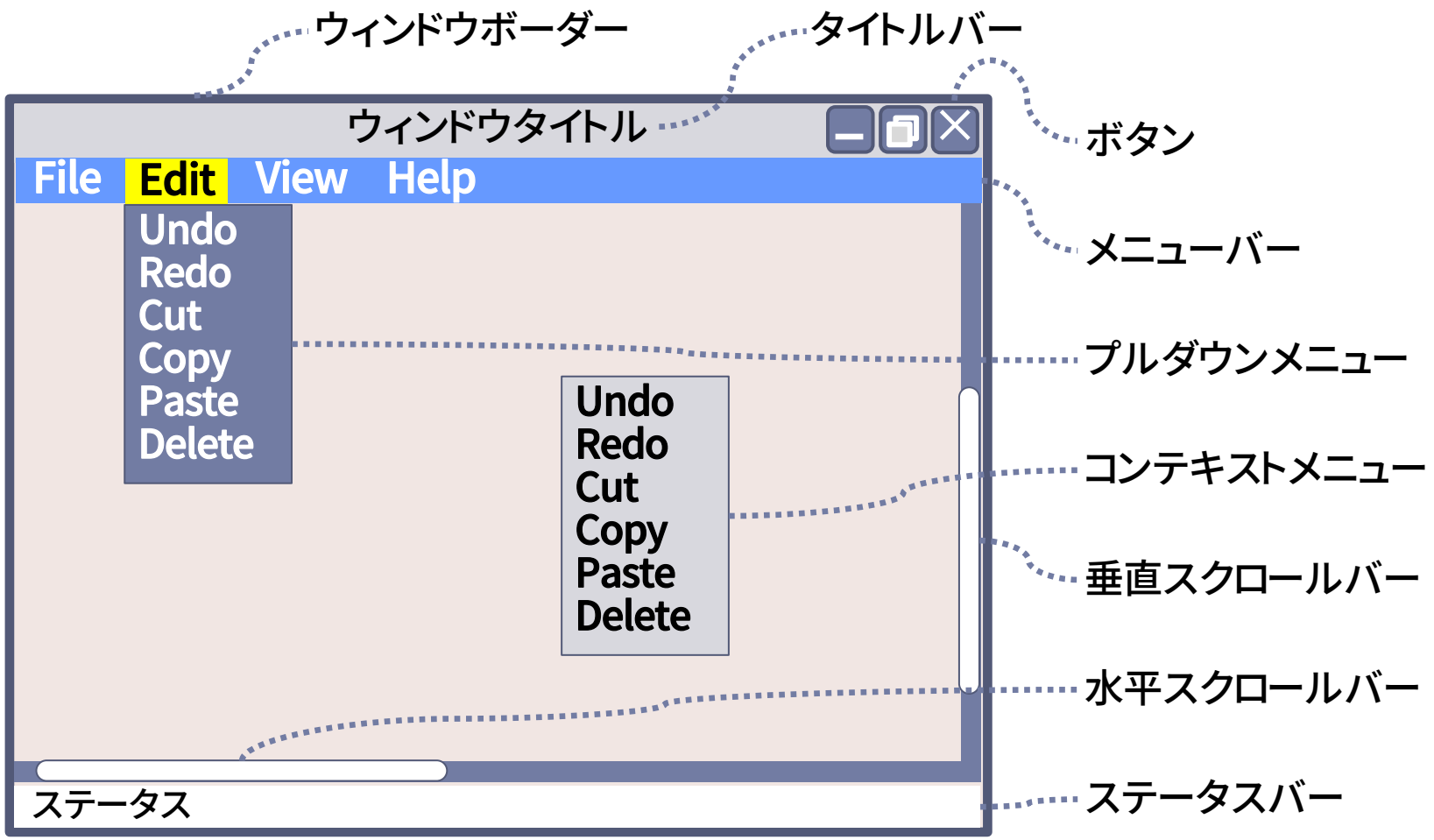


ここで、ウィンドウシステムの場合、ユーザが入出力で使用するコンピュータが「サーバ」になることに注意してください。

データベースに処理を依頼するような場合は、ユーザ側のコンピュータがクライアントでデータベースサーバに処理を依頼してサーブしてもらいます。

ウィンドウの場合は、計算や処理を行った側がクライアントとなって、その出力を画面に出してもらうようにディスプレイサーバに依頼する形式です。

ウィンドウの構成要素



ウィンドウの形式は、バリエーションは様々ですが、大体このようなもので、タイトルや各種のボタン、メニューが配置され、その領域をクリックなどするとメニューが現れる形式です。

メニューは、現れる動作のスタイルからプルダウンメニューとかポップアップメニューとか呼ばれることもあります。図にあるコンテキストメニューは、選択されている操作対象など（コンテキスト：文脈）に応じて、表示されるメニュー項目が変化するもので、図では画面の何もないところで操作したものですが、たとえばファイルのアイコンを選択した場合は、「開く」や「移動」、「名前の変更」などといったメニュー項目が現れます。

ウィンドウのサイズは適宜変更できますが、あるサイズの中で見えない部分に行くためにスクロールできるような仕組みがあります。

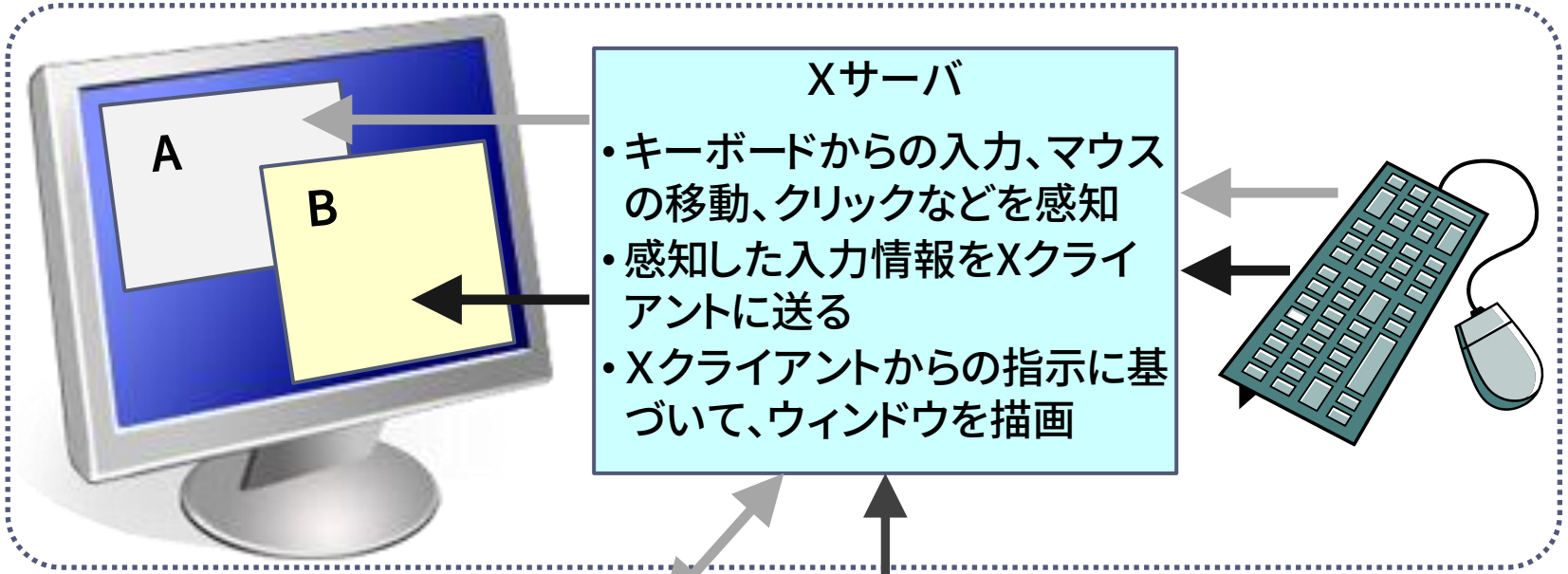
Xウィンドウシステム

- X Window System（X11、X）
- 1984年MITで開発
  現バージョンX11は1987年9月に登場
  - 当初から複数ウィンドウ、アイコン、メニュー、ポインタ（WIMP）を具備
    Window　Icon　Menu　Pointer
  - 最新バージョン：X11R7.7（バージョン11, リリース7.7　2012年6月）
- クライアントサーバの形式で動作
- 「Xサーバ」はユーザのコンピュータ上で稼働
  - キーボードやマウスの入力と画面への描画（ウィンドウや文字の表示）を担当
- 「Xクライアント」はXサーバと通信しながらユーザの目的とする処理を行い、結果に応じた描画指示を送る
- 多くのUNIXではXが標準のウィンドウシステム
  他のOS上でXウィンドウシステムを実行することも可能

ウィンドウシステムで代表的なものは、MITで開発されたXウィンドウシステムです。単にXとか、現バージョン番号を含めてX11と呼ばれることもあります。

ユーザのコンピュータ上のXサーバとユーザの目的とする処理を行って結果に応じた描画の指示を出すXクライアントからなるクライアントサーバ方式で動作します。

動作の仕組み



Xサーバ

- キーボードからの入力、マウスの移動、クリックなどを感知
- 感知した入力情報をXクライアントに送る
- Xクライアントからの指示に基づいて、ウィンドウを描画

XクライアントA

- Xサーバからの入力情報に基づいて処理を行い、結果に応じた描画指示をXサーバに送る

XクライアントB

- Xサーバからの入力情報に基づいて処理を行い、結果に応じた描画指示をXサーバに送る

XサーバとXクライアントは同じPC上で
稼働していても、ネットワーク接続の別のコンピュータで稼働してもよい

Xウィンドウの動作の仕組みは、Xサーバがキーボードなどからの入力をXクライアントに送ります（プロセス間通信です）。
クライアントはプロセスで、ここではA、B2つあり、Xサーバは、マウスポインタの位置から選択されているウィンドウを知って、正しいクライアントに入力情報を送ります。

クライアントではそれぞれ、送られた入力情報に基づいて処理を行って結果に応じた描画指示をXサーバに送ります。
なお、XサーバとXクライアントは同じPC上にあっても、それぞれ別のPCにあってネットワーク接続されているものでも構いません。（p.8）

デスクトップ環境

- デスクトップ環境（デスクトップマネージャ）
  GUIを提供するオペレーティングシステムの重要でよく使う機能をアクセス・設定変更する手段を提供する
  （すべての機能へのアクセスを提供するわけではない）
- デスクトップメタファーの実現
  - コンピュータのディスプレイ上でユーザの机上（デスクトップ）を表現し、そこに文書やフォルダを置く（ゴミ箱などもある）
  - 文書を開くとウィンドウが開き、それが机上に置かれた紙の文書を表現
  - デスクアクセサリと呼ばれる小さなアプリケーション群が机上の様々な道具（電卓など）に対応
- デスクトップ環境は一般にアイコン、ウィンドウ、ツールバー、フォルダ、背景画像、デスクトップウィジェット（ガジェット）などで構成される
- デスクトップの操作
  ファイルの操作、ウィンドウの操作、ワークスペース

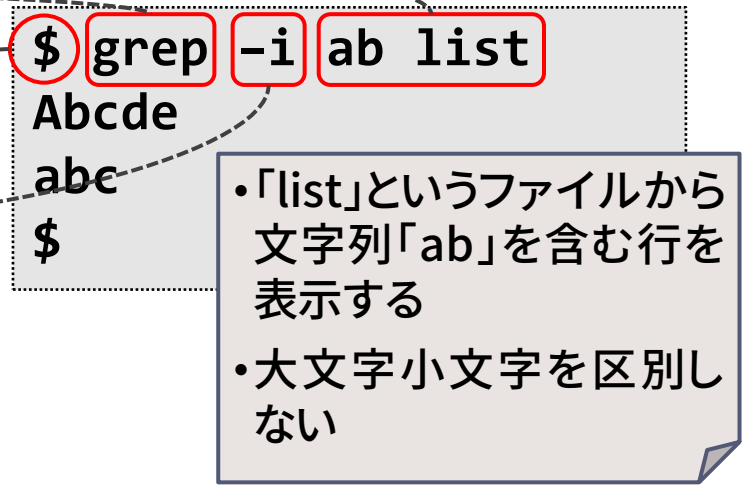
GUIによって、よく使う機能、すなわちアプリケーションを起動したりするやり方（メニューや既定のアプリなど）、ツールのようなものの表示、背景などの設定、などを使い易くしたものがデスクトップ環境です。

ユーザのデスク周りを模擬したようになっていて、ファルダを開いて対象の書類を開くということなどをクリックやタッチで直感的に実現しているものです。
アイコンで引き出しや書類フォルダを表し、開くとウィンドウがその中身を表すなどになります。

デスクトップの操作は、主にファイルを操作（作成・削除や名称変更、情報表示など）したり、ウィンドウを操作したりするものです。
複数のデスクトップを作成して、切り替えて使い分けるワークスペース（仮想デスクトップ）のような機能もあります。

コマンドの実行

- コマンド
  操作の指令
- パラメータ
  操作対象とオプション
  （付加的な機能の指定）
- （コマンド）プロンプト
  オペレーティングシステムが
  「コマンド入力待ち状態」であることを表す記号
  利用者にコマンド入力を促すもの



  - 操作対象（引数）
    … ファイル名やデータなど（を指定する文字列）
  - オプション
    … コマンドの機能を調整したりする指示を与えるための文字列
    通常、「-」と組み合わせて指定する

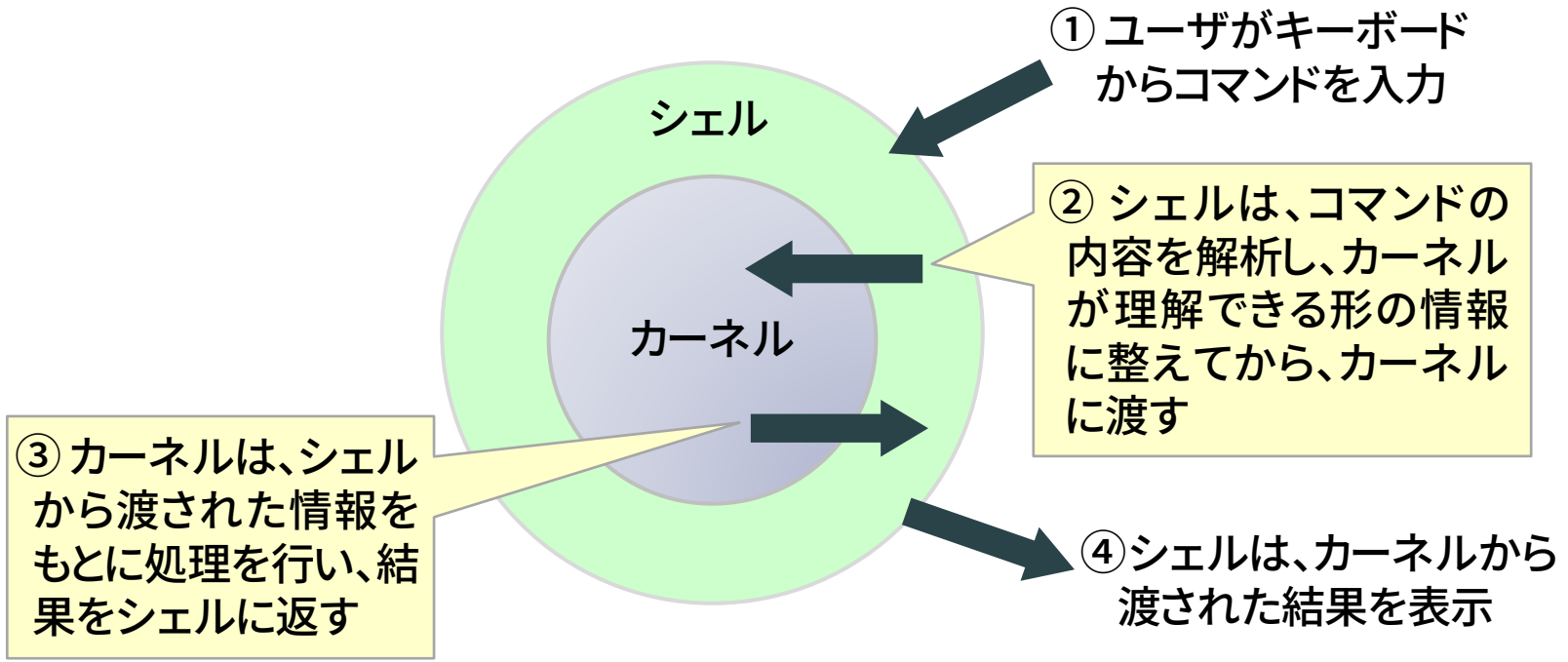
p.3で述べたように、コマンドは、コマンド自体（コマンド名）とパラメータとからなっています。

（Linuxなどで）コマンドを入力するときの画面はこのようになります。ここでは＄で表されていますが、画面上にOSからの文字列が表示され、その直後にカーソルがあって入力可能な状態となっており、そこにコマンドを入力して行きます。
この＄（他にもいろいろありますが）は、入力を待っているものということで、プロンプト（コマンドプロンプト）と呼ばれます。

コマンドに対するオプションは、UNIX系では「-」と組合わせますが、Windowsでは「/」と組合わせます。

コマンドラインインタプリタ（コマンドインタプリタ）

- コマンドラインインタプリタ（Command Line Interpreter：CLI）（UNIXではシェル）
  - ユーザが打ち込むコマンドを読み込み、解釈してそれに応じた動作を実行させてくれるプログラム



シェル（殻）という名称は、カーネル（核）の外層として動作することからきている

コマンドを解釈するプログラムがコマンドラインインタプリタで、これがユーザとOS（カーネル）とのやり取りの仲立ちを行います。

コマンドを読んで解釈（前ページのようにして入力した文字列を区切りで分けて、コマンド名やオプション、パラメータを認識する）し、そのコマンドに対応するプログラムを実行するなど、それに応じた動作をしてくれます。

各種のコマンドラインインタプリタ

- 通常、CLIはキャラクタユーザインタフェースを提供する
  （グラフィカルユーザインタフェースを提供するものをグラフィカルシェルと呼ぶ）
  - 「シェル」はUNIX系オペレーティングシステムで使用される言葉
  - UNIX系ではシェルがユーザプログラムとして実装
    別のシェルに差し替えて使用することができる
  - シェルの例（CUIベースのもの）
    bash … 多くのLinuxシステムで標準シェルとして採用されている
    他に、csh や sh など
  - WindowsではコマンドプロセッサやWindows PowerShellと呼ばれるコマンドラインインタプリタがある
- シェルの主な機能
  プログラムの起動・終了、ジョブの制御、リダイレクト・パイプ、
  コマンドライン中のワイルドカードの展開、入力の補完、
  コマンドの繰り返し実行・条件実行、
  まとまったコマンドをスクリプト（シェルスクリプト）として実行（バッチ処理）

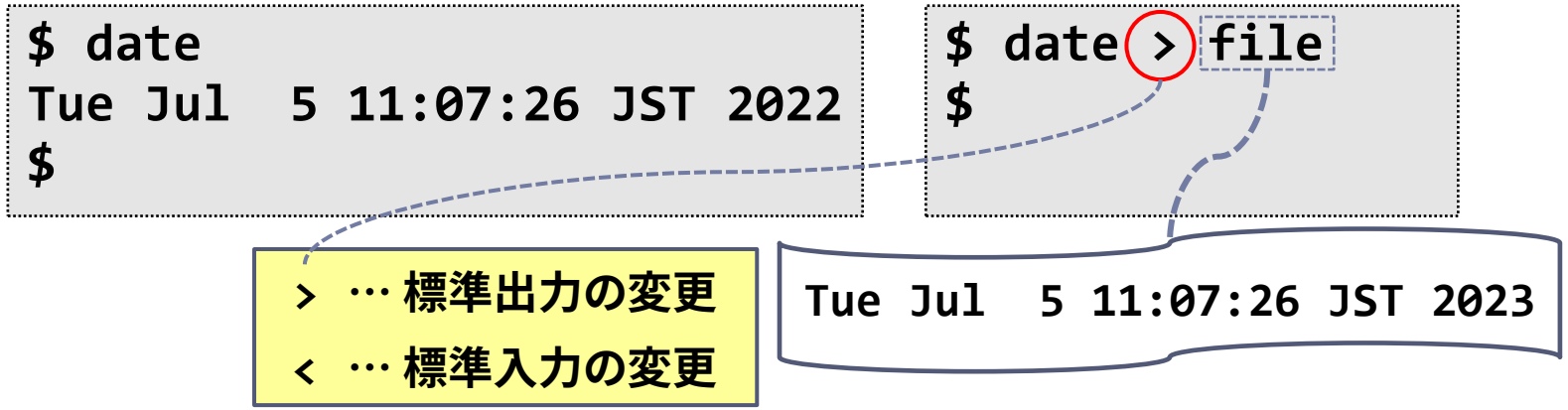
コマンドラインインタプリタは、UNIX系OSではシェルと呼ばれます。Windowsではコマンドプロセッサやコマンドプロンプト（本来の意味からの転用です）と呼ばれます。またシェル相当のWindows PowerShellもあります。

シェルは、ユーザプログラムとして実装されるので、ユーザが好みのものを使用することができ、Linuxではbash（バッシュ - Bourne-again Shell）が標準ですが、csh（Cシェル）や sh（ボーンシェル - Bourne Shell）などがあります。

シェルの主な機能の一部について、以降で説明します。

標準入出力とリダイレクト

- UNIX系オペレーティングシステムのコマンドの多くは「標準入力」からデータを取り込み、処理結果を「標準出力」へ送り出す
  - 通常、キーボードからの入力を受付け、処理結果がディスプレイに表示
    標準入力＝キーボード、
    標準出力＝ディスプレイ　と割当てられている
- 「リダイレクト」という機能により、この割当てを変更できる
  - コマンドに対して、キーボード入力の代わりにファイルからデータを入力したり、ディスプレイ出力の代わりにファイルにデータを出力させるように指示できる



ファイルシステム（第9回、p.9）で標準入力と標準出力が出てきましたが、UNIX系のOS（Windowsでも）ではコマンドの多くは、標準入力からデータを得てコマンド処理を行い、結果を標準出力に出すようになっています。
下のdateコマンドでは、dateだけで、出力先を指定しなくてもディスプレイに出るようになっています。

出力先が標準のディスプレイでないときは、標準出力をコマンド指定時に変更することができ、この機能をリダイレクトと呼んでいます。図では、＞ file で「file」という名称のファイルにdateの結果を書き出しています。

パイプ

- UNIX系OSでのパイプ
  - あるコマンドの標準出力を別のコマンドの標準入力として直接利用する

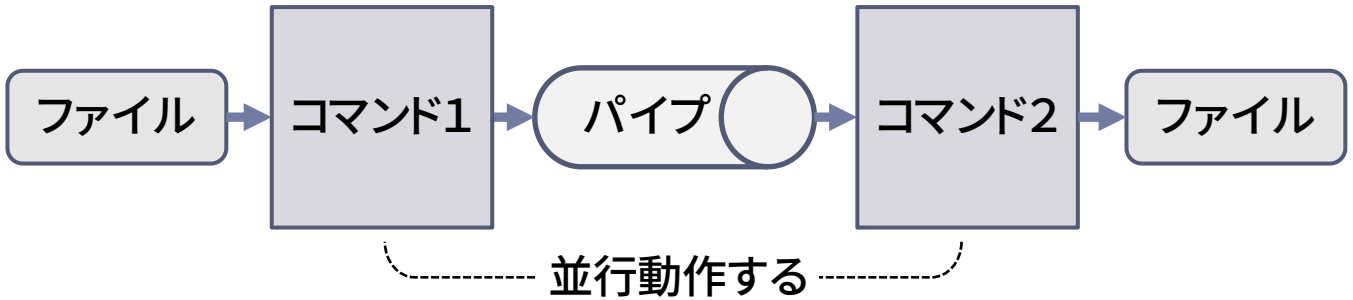
```
$ head -5 list > temp
$ sort < temp
```

- 「list」の最初の5行を「temp」に一時保存
- 「temp」をソート

```
$ head -5 list | sort
```

その前のコマンドの標準出力を次のコマンドの標準入力にリダイレクトする

  - 処理の途中で一時的なファイルを作成しなくても一度に最終結果を得ることができる



パイプ(pipe)という言葉は、プロセス管理(2)(第9回、p.24)のプロセス間通信で出てきました。
図で、コマンド2行で表示されているものは、headは引数のファイルの先頭から何行か(指定しないと10行、ここではオプションで5行を指定)を出力するもので、前ページのリダイレクトによってファイルtempに出力します。

次のsortは引数のファイルを行ごとに(辞書順などの)順序で並べ替えるもので、標準入力をtempというファイルにリダイレクトしたものです。すなわち、tempを一時ファイルとしてlistファイルの内容の先頭5行をとりだして格納し、それをソートするものです。

この一時ファイルをパイプで置き換えて、2つのコマンドをつなげて実行することができます。|によって、2つのコマンドをパイプでつなぐことを指定します。

その他の機能

- ワイルドカード、メタキャラクタ
  任意の文字列を指定するための特殊な文字
  複数のファイルを総称的に指定して効率的に操作できる

?	任意の1文字	[…]	カッコ内の文字のうちのどれか1文字
*	0文字以上の任意の文字列	{…, …}	コンマで区切られた文字列のうちのどれか

  - 例えば ac abc abd というファイルがある場合、「cat a*c」と入力すると、「cat abc ac」のように展開され、catプログラムが起動される（UNIX系の場合）

- 補完
  入力中に [Tab] キーで、シェルが残りの文字列を入力する
  （コマンド名、ファイル名の補完が行われる）

```
$ ls
abc1   abc2   def
$ cat a[Tab]
$ cat abc
```

選択地点（abc1かabc2か）まで表示

コマンドを入力するときに、操作を少なくして便利にする機能があります。

コマンドを入力するとき、引数は長い文字列になったりします。しかもそのほとんどが同じで、最後の1文字だけが1、2、3、などと替わるだけなどということが多かったりします。そのような、ある性質をもった文字列を総称して指定することができます。

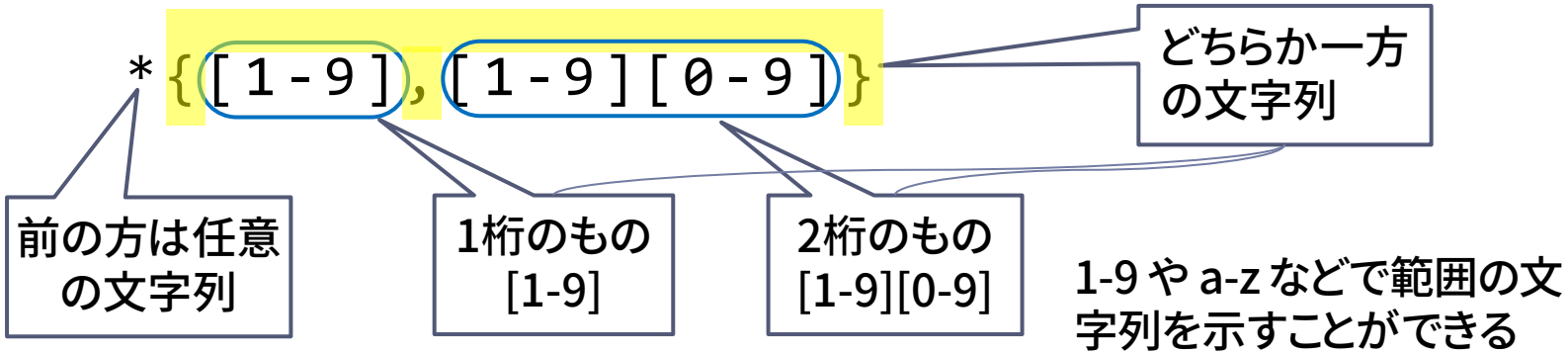
特別な意味をもつ文字を与えます。文字列を総称的に指定するものがワイルドカードで、任意の1文字と0文字以上の任意の文字列を表すものがあります。a?cだと3文字の2文字目がいろいろに変化するということで、aac、abc、acc、a1c、a2c、･･･ が得られる文字列です。a* は、aだけ、ab、a123、a12345、などaで始まる任意の文字列を表します。

その他、[ ]で囲まれた文字列では、その中の1文字とマッチしたもの、{ }の中に , で区切られた文字列があると、そのうちのどれか1つとマッチしたものとなります。これら?や＊[ ] { }のように、本来の文字の意味ではなく、特別な意味をもつ文字をメタキャラクタと呼びます。

入力中にtabキーを押すと、確定できる場合、残りの文字列が自動的に得られます。複数ある場合は次に選択する地点までを自動で表示します。

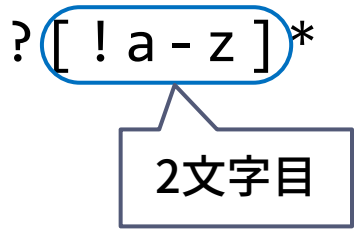
ワイルドカードとメタキャラクタの使用例

- 末尾が 1 ～ 99 のいずれかである文字列



1-9 や a-z などで範囲の文字列を示すことができる

- 先頭から2文字目にアルファベットの小文字を含まないもののみ



[!文字リスト] … 指定した文字リスト以外にある文字と一致

- *abc?[2-5] とマッチする文字列

(1) abcd4　(2) xabc5　(3) 01abc23
(4) abc.1　(5) aaabc?5　(6) zabc?-

ワイルドカードとメタキャラクタの使用例を示しています。

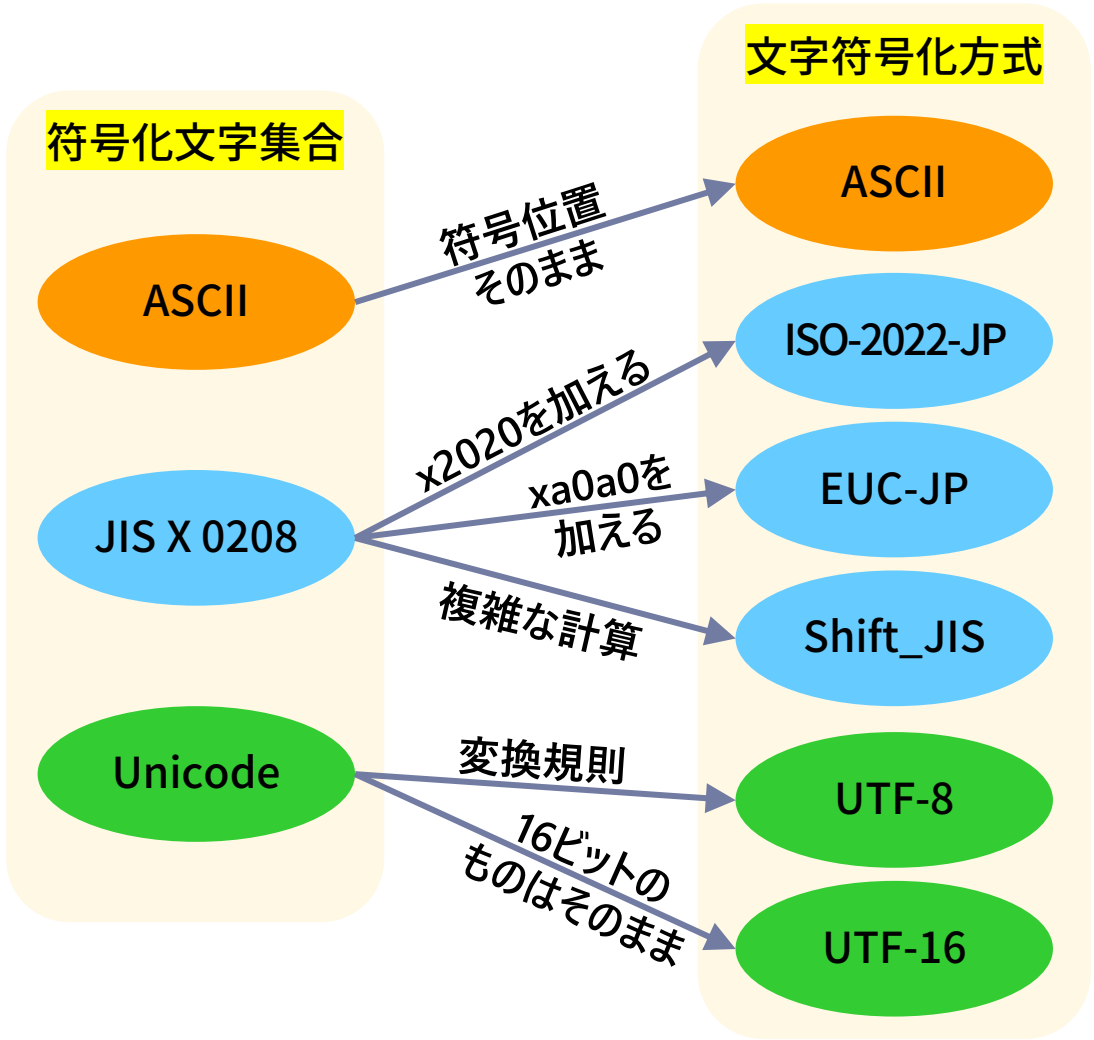
[123456789] は、「-」を使って[1-9]とも書けます。同様に、[a-z]は小文字のアルファベットa～zの範囲の1文字を示します。
「!」はリスト内にある文字以外の1文字を意味するものです。

3例目で、(2)ではcのあとに2文字必要です。

国際化・地域化

- 国際化 … 種々の国や地域での多様性を吸収して共通的な枠組みで対応できるようにすること
- 地域化 … 国や言語、文化等に対応すること
- ロケール … 文字、日付、単位、通貨をはじめとする国や地域ごとに異なる基本設定の集合
- 符号化文字集合（文字コード） … 文字の集合の各要素に一意な表現（バイトやバイトの並び）を割り当てたもの
  - ASCII　7ビットコード。大小のラテン文字や数字、英文でよく使われる約物など
  - JIS X 0208　2バイトコード。非漢字（記号、ラテン文字、平仮名など）524文字、第一・第二水準漢字 6355文字
  - JIS X 0213　JIS X 0208を拡張し、第三・第四水準漢字などを加えた上位集合 (11233文字)
  - Unicode　世界中の様々な言語の文字を収録し、通し番号を割り当てて1つのコード体系のもとで使用できるように設計
    登録文字それぞれについて「コードポイント」と呼ぶ一意の通し番号を与える（日本語の「ア」には12450番 …「U+30A2」のように表記）
- 文字符号化方式 … 符号化文字集合で定義された文字をコンピュータで使えるように変換する方式
  - ISO-2022-JP（JISコード）　国際的な文字コード規格の一つであるISO/IEC 2022の枠組みに沿って定義された
    文字を7ビット単位で符号化する方式を定めたもの。エスケープシーケンスを挿入することで複数の文字集合の切り替えを行う
  - EUC-JP（日本語EUC/拡張UNIXコード）　規格自体は日本語だけでなく、多バイト文字（マルチバイト文字）で用いられる
    1バイト文字（ASCII、「半角カナ」）であればそのまま、そうでなければマルチバイト文字とする
  - Shift_JIS　MS-DOSやWindowsが標準の日本語文字コードとして採用したことから広く普及
    文字集合を分割し、それぞれ離れた領域へ移動（shift）させる（ISO-2022-JPやEUC-JPなどでは連続したコード領域に文字を収録）
    1バイト目に1バイト文字と重ならないようにコードを配置することでエスケープシーケンス不要
  - UTF-8　1～4バイトの可変長で表現。ひらがなや漢字は3バイトで表現される

符号化



JIS X 0208
⇓
EUC-JP

あ
0402
(区点)

0000 0100 0000 0010
0 4 0 2

xa0a0を加える

1010 0100 1010 0010
a 4 a 2

Unicode
⇓
UTF-8

あ
U+3042

0011 0000 0100 0010
3 0 4 2

11100011 10000001 10000010
e3 81 82

Unicode文字の範囲	UTF-8でのビット列
U+0000～U+007F（0xxx xxxx）	そのまま
U+0080～U+07FF（0000 0yyy xxxx xxxx）	110yyyxx 10xxxxxx
U+0800～U+FFFF（yyyy yyyy xxxx xxxx）	1110yyyy 10yyyyxx 10xxxxxx
U+10000～U+10FFFF	略

教科書との対応

- UTF-8 について補足しています
- コマンドの操作とシェルスクリプトについて、説明を追加しています
- クライアント・サーバ方式（教科書 12.4）について、今回触れました

教科書での説明と記載箇所が違うところがありますので、読むときに注意してください。

第12回の課題

（1）文字コードをすべて（英数字、漢字とも）2バイトで統一して扱うことの利点と問題点を述べよ

（2）末尾が 10〜59 のいずれかである文字列（たとえば、data25、file23、files40 など）をワイルドカードとメタキャラクタを使って表せ

今回の課題です。クラスウェブのレポートで提出してください。

期限は、7/6の午前中とします。

事後学習・事前学習

- 今回の講義資料に基づいて内容を振り返り、教科書などの該当箇所を読む
- 教科書第11章（11.1）に目を通しておく

今回の講義内容の振り返りと次回の準備をお願いします。