

GRAU D'ENGINYERIA INFORMÀTICA
Curs acadèmic 2018-19

PROGRAMACIÓ II

Problema 1:

**Descomposició funcional
descendent i orientada a objectes**



Continguts

- **Objectius**
- Exercici
- Solucions
- Conclusions

Objectius

- 1) **Introduir diferències entre paradigmes:**
 - Programació procedimental (funcions)
 - Programació orientada a objectes (dades)
- 2) **Elements importants** en aquest canvi de paradigma:
 - Més enllà del 'funciona': ens interessa la **qualitat**
 - **Bona abstracció**: del problema i no del món
 - Estratègia: centrar-se en les **dades** / objectes

Per assolir aquests objectius

- Realitzarem **entre tots un exercici*** de manera guiada
- L'exercici consisteix en dissenyar un **sistema de reserves per a una companyia aèria**
- Construirem i analitzarem **dos solucions, i el pas** de la primera a la segona
 - 1) descomposició funcional descendent
 - 2) descomposició orientada a objectes

* Exemple del capítol 20 del llibre: "Construcción de software orientado a objetos", Bertrand Meyer. Prentice Hall, 1998, amb algunes modificacions / ampliacions introduïdes en el context de l'assignatura de Programació II

Continguts

- Objectius
- **Exercici**
- Solucions
- Conclusions

Exercici - enunciat

- Una companyia aèria ens ha contactat per a que li programem un **sistema de reserves basat en panells**
- L'empresa ens ha proporcionat un exemple de panell i alguns detalls del sistema que vol

Exercici – exemple de panell

– Consulta de vols –

Vol des de: Destí:

Sortida prevista: Arribada:

Companyia aèria:

Requisits especials:

VOLS DISPONIBLES: 1

Vol: AA 42 Sortida 8:25 Arribada 10:05 Escala: -

Escolir una opció:

- 0 – Sortida
- 1 – Ajuda
- 2 – Consulta reserves
- 3 – Reserva plaça

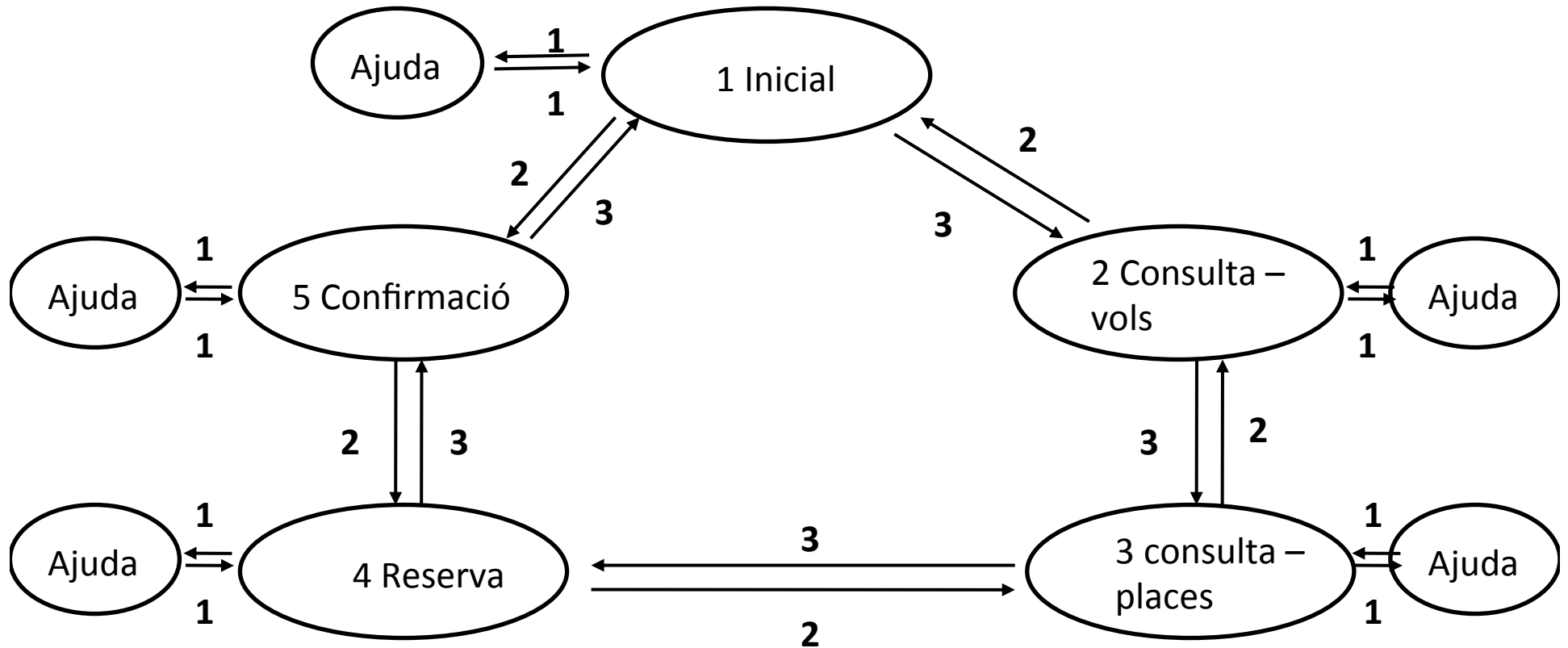
Exercici – detalls del sistema

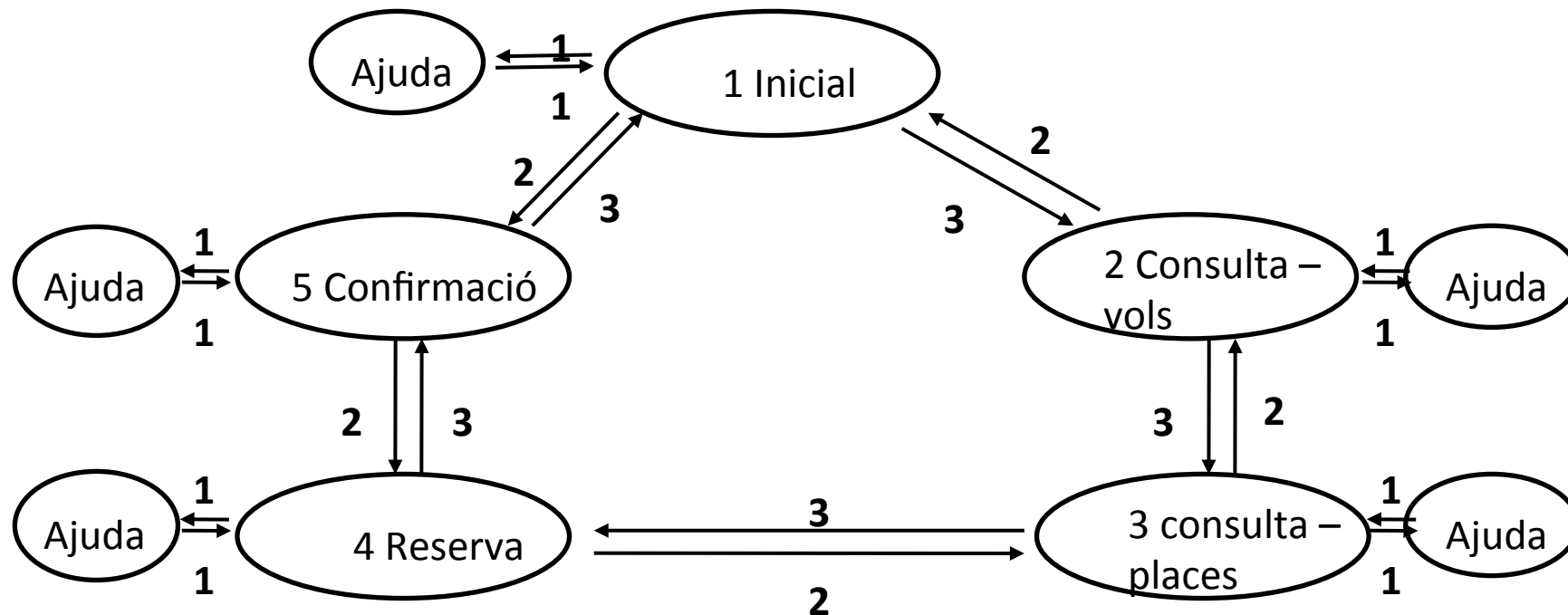
- Les reserves són sobre **vols** i **places** (seients) als vols
- Les converses amb l'empresa han revelat que les reserves es realitzen mitjançant una **sèrie de passos** o estats:
 - a l'**inici**, es mostren totes les opcions del sistema;
 - a l'estat de **consulta de vols**, l'usuari introdueix origen, destí, i dades, i el sistema li mostra els vols disponibles;
 - a l'estat de **consulta de places**, l'usuari selecciona les places, i reserva;
 - a l'estat de **reserva**, l'usuari revisa les dades, introdueix les dades de pagament, i confirma;
 - a l'estat de **confirmació**, es confirma la reserva, i també es pot modificar

Exercici – més detalls del sistema

- L'empresa ens ha demanat que l'ordre de realització dels passos **no sempre sigui endavant**
 - Un usuari, a l'estat de consulta de places, pot tornar, per exemple, a l'estat de consulta de vols
- D'altra banda, per **reservar**, l'usuari ha de passar necessàriament per consulta de vols i places **en aquest ordre**
- A l'estat d'inici l'usuari **s'identifica** per començar la reserva, consultar l'estat d'una reserva i/o modificar-la (si és possible)
- A cada estat es pot demanar mostrar l'**ajuda**

Exercici: diagrama de transició





Taula de transició:

Panell / Opció	0	1	2	3
1 (Inicial)	-1	0	5	2
2 (Vols)		0	1	3
3 (Places)		0	2	4
4 (Reserves)		0	3	5
5 (Confirm.)		0	4	1
0 (Ajuda)		Tornar		
-1 (Final)				

Continguts

- Objectius
- Exercici
- **Solucions**
- Conclusions

SOLUCIONS

- **Primer intent** (simple)
- Solució funcional descendent
- Solució orientada a objectes

Solucions: primer intent

- P_{Vols}
"Mostra el panel de Consultes de Vols"
repetir
"Llegir la resposta de l'usuari i escollir C
com a següent pas"
si "Hi ha un error a la resposta" llavors
"Mostrar el misstage d'error apropiat" **final**
fins no "error a la resposta" **final**
"Processar la resposta"
cas C en
C₁ : **goto** P_{Ajuda}
C₂ : **goto** P_{Inici}
C₃ : **goto** P_{Places}

Solucions: primer intent

- **Punts forts d'aquesta solució**
 1. És una solució lògica
 2. És una solució estructurada
- **Problemes d'aquesta solució**
 1. És una traducció literal del problema
 2. Si el problema canvia (nous estats, eliminació d'estats, noves transicions entre estats...), la solució també
 3. *Go to*: “spaghetti code”

(primer intent: go to statement)

```
public static void Main()  
{
```

```
    labelA; ←  
    if( ... )  
        goto labelC;  
    if ( ... )  
        goto labelB;
```

```
    labelD; ←
```

```
    if ( ... )  
        goto labelE;  
    labelC ←
```

```
    labelE; ←
```

```
    if ( ... )  
        goto labelA;  
    if ( ... )  
        goto labelD;  
    labelB; ←
```



Hmm... what's
going on here?

<http://www.linuxdigest.org/blog/2012/06/14/why-goto-statement-is-evil/>

- Raons a favor i en contra d'utilitzar instruccions 'go to'
- **En contra:** llegibilitat de programes, pila del sistema i seguiment de l'estat del programa
- **A favor:** s'utilitza (per exemple, tractament d'excepcions), eficiència...
- **Conclusió:** saber quan (no) utilitzar-lo; en el nostre exercici, no

A decorative graphic on the left side of the slide, consisting of a vertical line with ten horizontal bars, each ending in a circle, resembling a spiral binding.

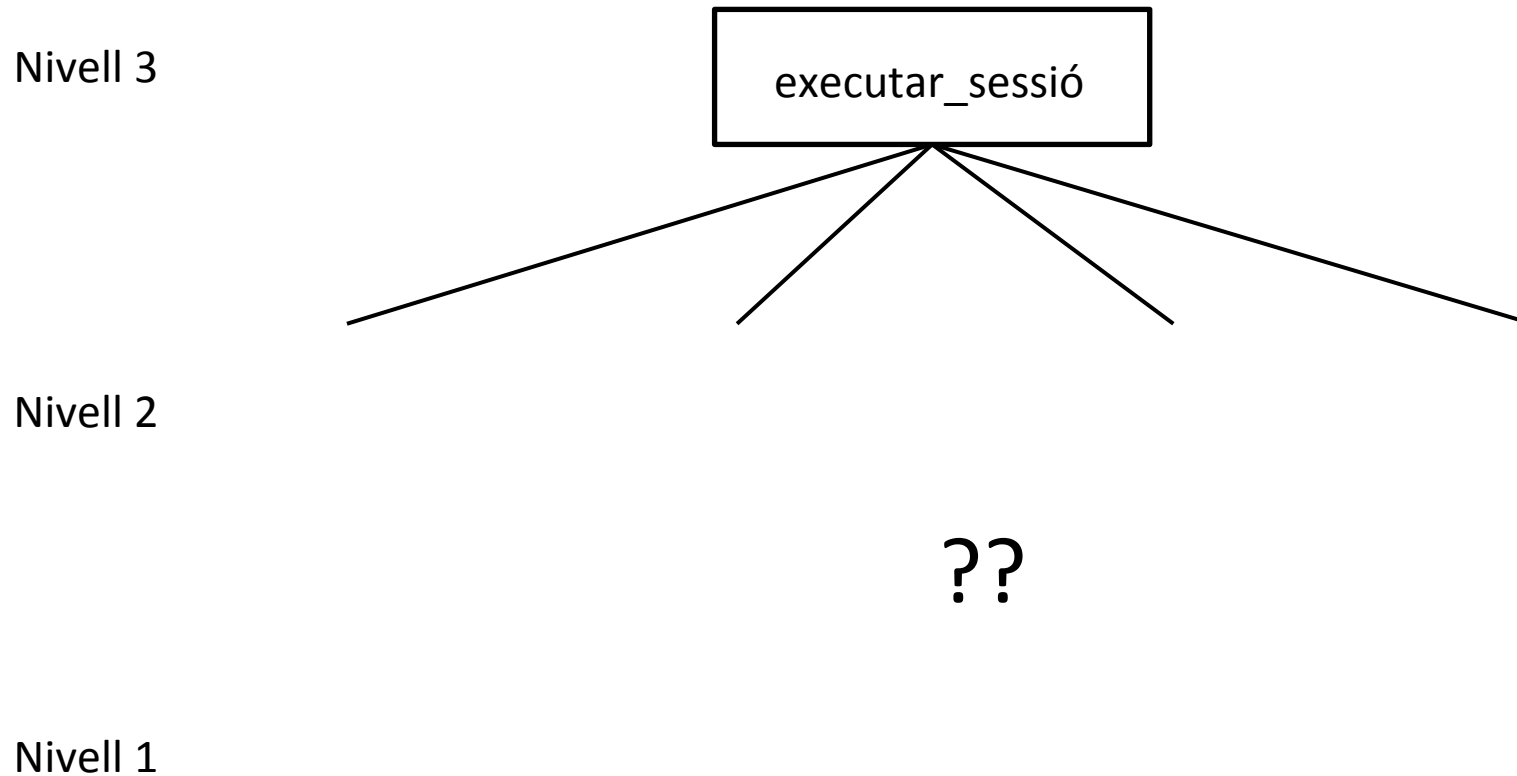
Solucions

Descomposició funcional descendent

Quina és la vostra proposta?

Solucions:

Descomposició funcional descendent

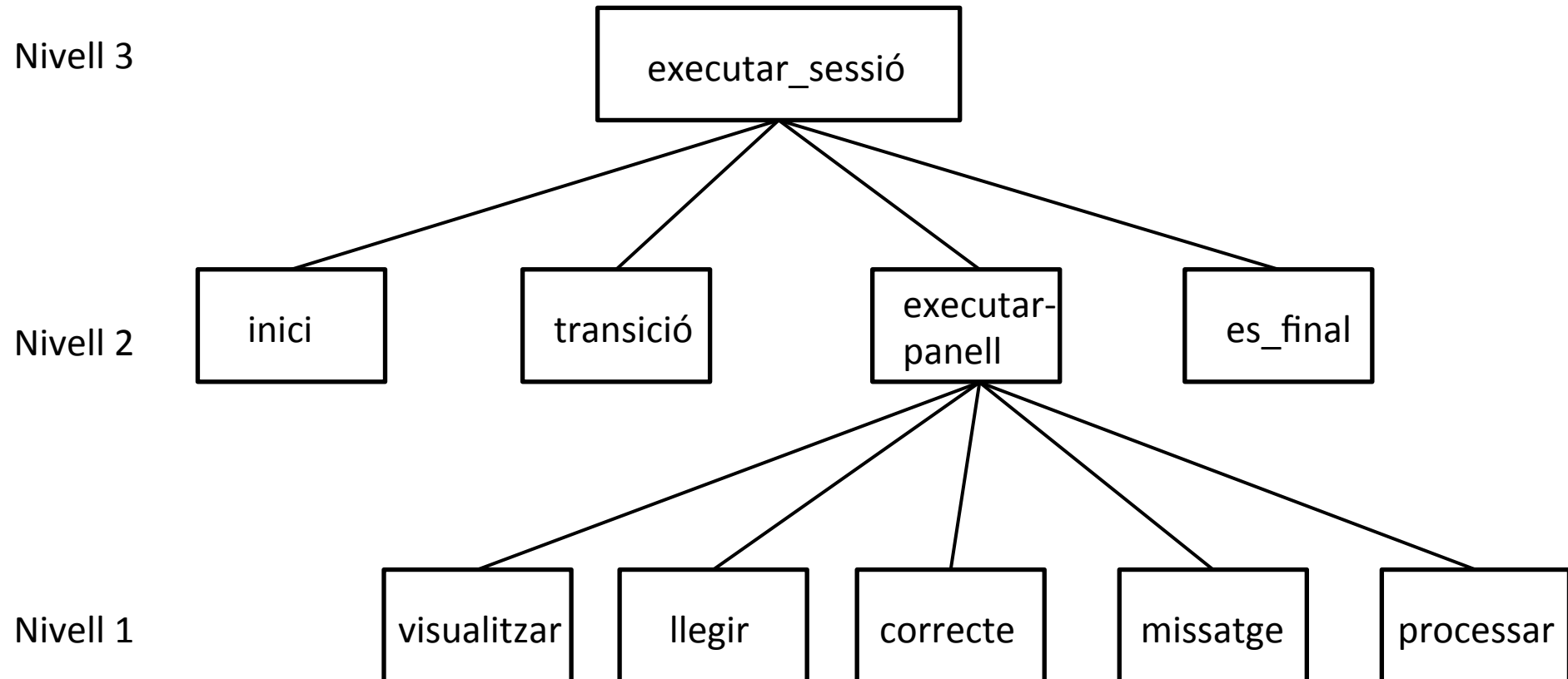


Descomposició funcional descendent

- **Objectius:**
 - Evitar que el nostre programa sigui una “còpia” del diagrama de transicions
 - Intentar que sigui modular
- **Estratègia:**
 - Pensar en les funcions del programa
 - Dividir les funcions (generals) en més funcions (més específiques)

Solucions:

Descomposició funcional descendent



Descomposició funcional

descendent: algunes rutines

```
executar_sessio es
  -- Executa una sessió completa del sistema
  interactiu

  local
    panell, seguent: INTEGER
  fer
    panell := inici
    repetir
      executar_panell(panell, → seguent)
      -- La rutina executar_panell actualitza
      el valor de seguent, a més d'executar
      les accions associades al panell.
      panell := transicio(panell, seguent)
    fins es_final(panell) final
final
```

Descomposició funcional

descendent: algunes rutines

```
executar_panell (in p: INTEGER; out op: INTEGER) es  
  -- Executa les accions associades al panell p,  
  -- tornant en op l'opció escollida per l'usuari per  
  -- al següent panell.  
local  
  r: RESPOSTA; ok: BOOLEAN  
fer  
  repetir  
    visualitzar(p)  
    llegir(p, → r)  
    ok := correcte(p, r)  
    si no ok llavors missatge(p, r) final  
  fins que ok final  
  processar(p, r)  
  op := seguent_opcio(r)  
final
```

Descomposició funcional descendent: punts forts

- Estructura:
 - no és un diagrama de transicions
- Modular:
 - cada funció / rutina s'encarrega de realitzar una acció determinada

Descomposició funcional descendent: limitacions (I)

Totes les rutines del **nivell 1 són molt grans i molt poc reutilitzables**, perquè han de realitzar accions diferents depenent del panell p. Per tant, efectuaran una discriminació de la forma:

```
inspeccionar
```

```
    p
```

```
quan Inicial llavors
```

```
    ...
```

```
quan Consulta_sobre_vols llavors
```

```
    ...
```

```
    ...
```

```
final
```

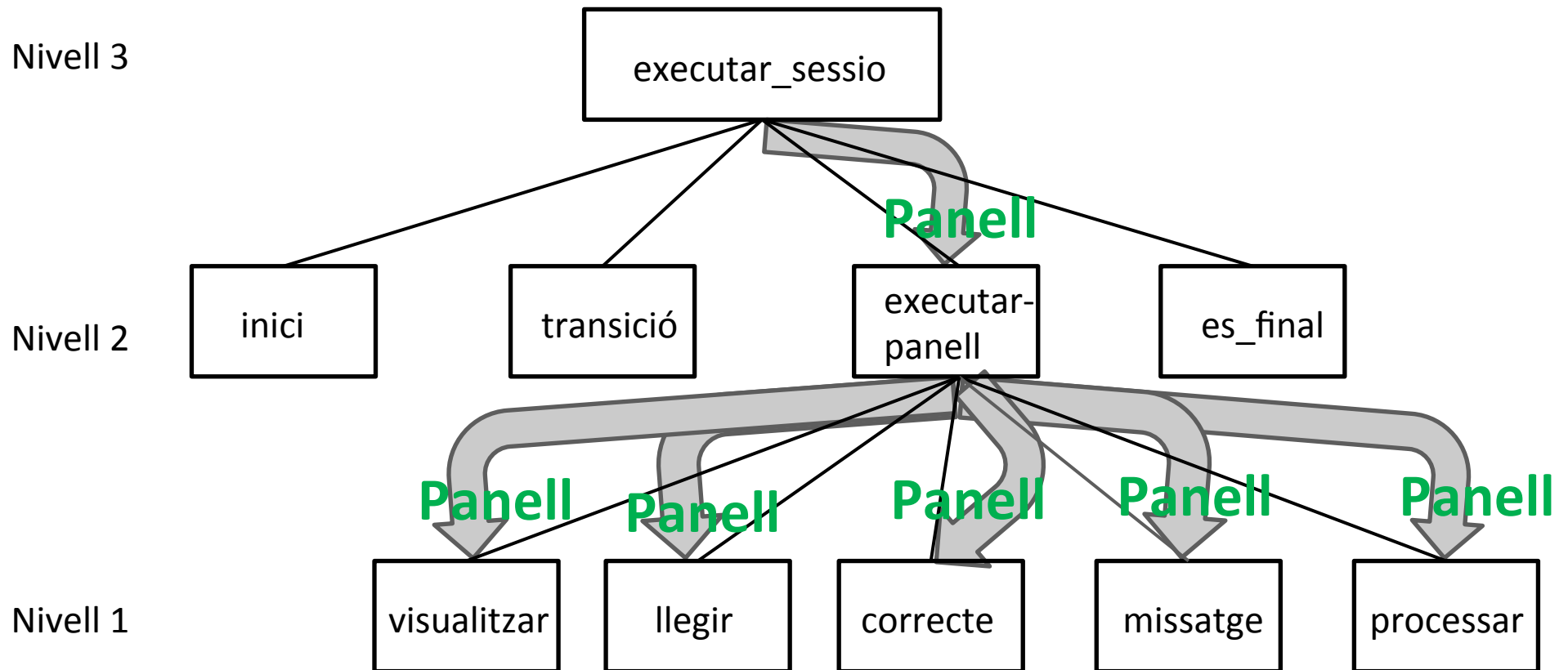

Descomposició funcional descendent: limitacions (II)

- Si considerem les signatures de les rutines, **totes depenen del panell (estat)**

```
executar_panell(in p: PANELL; out p: ELECCIÓ)
visualitzar    (in p: PANELL)
llegir         (in p: PANELL; out r: RESPOSTA)
correcte       (in p: PANELL; r: RESPOSTA): BOOLEAN
missatge      (in p: PANELL; r: RESPOSTA)
proces        (in p: PANELL; r: RESPOSTA)
```

Intervenció de
PANELL

Descomposició funcional descendent: limitacions (III)



Els estats determinen el sistema de reserves, però en aquesta solució, hem perdut la seva importància – el focus està a les funcions

A decorative graphic on the left side of the slide, consisting of a vertical line with a series of horizontal bars and circles, resembling a spiral binding.

Soluciones

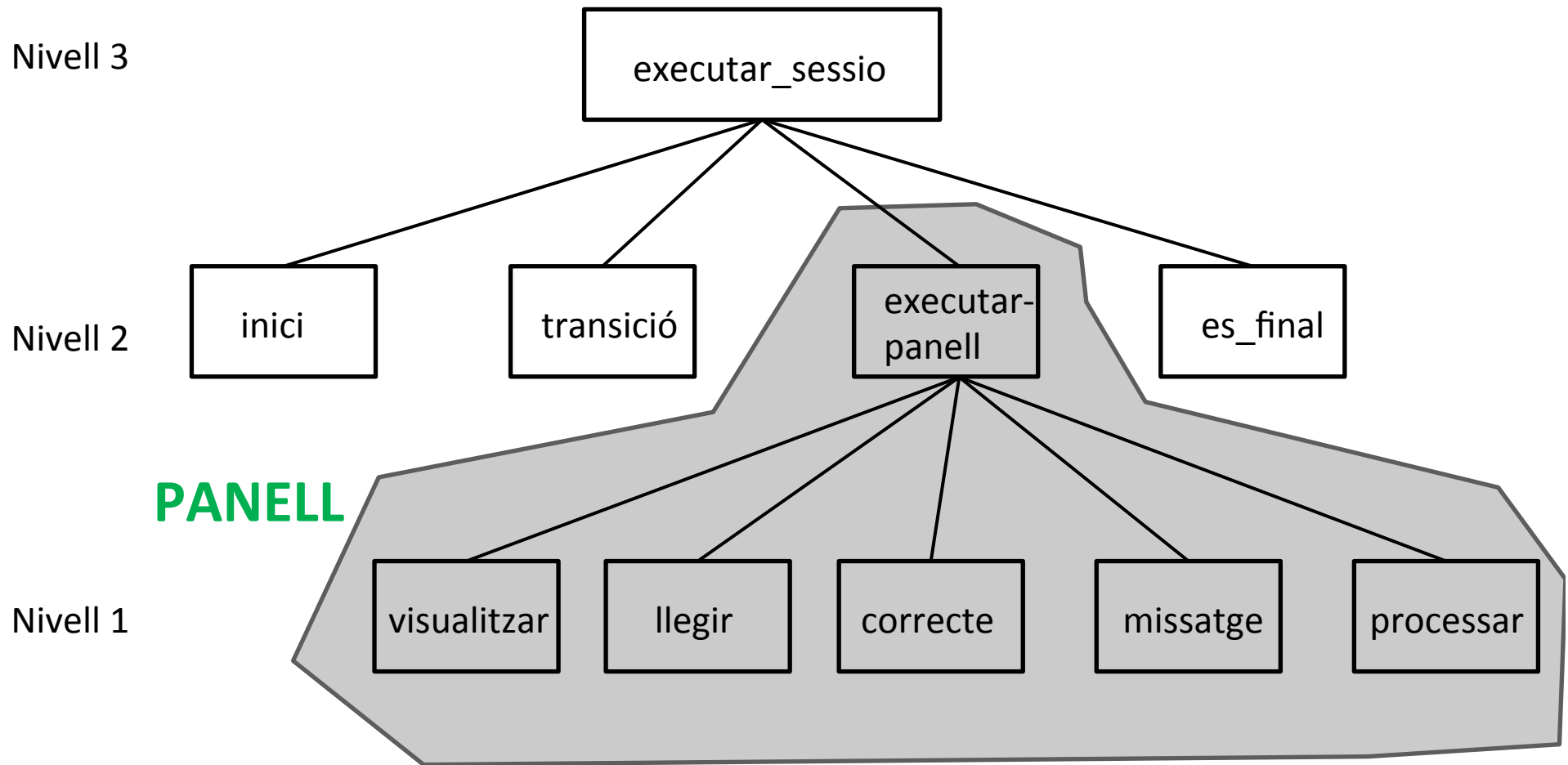
Descomposició orientada a objectes

Descomposició orientada a objectes

Llei d'inversió:

“Si les rutines intercanvien massa dades, posar les rutines en les dades.”

Descomposició orientada a objectes: panell (estat)



Descomposició orientada a objectes: classe panell

classe PANELL **característica**

entrada: RESPOSTA

opció: INTEGER

executar **es fer** ... **final**

visualitzar **es** ...

llegir **es** ...

correcte: BOOLEAN **es** ...

missatge **es** ...

processar **es** ...

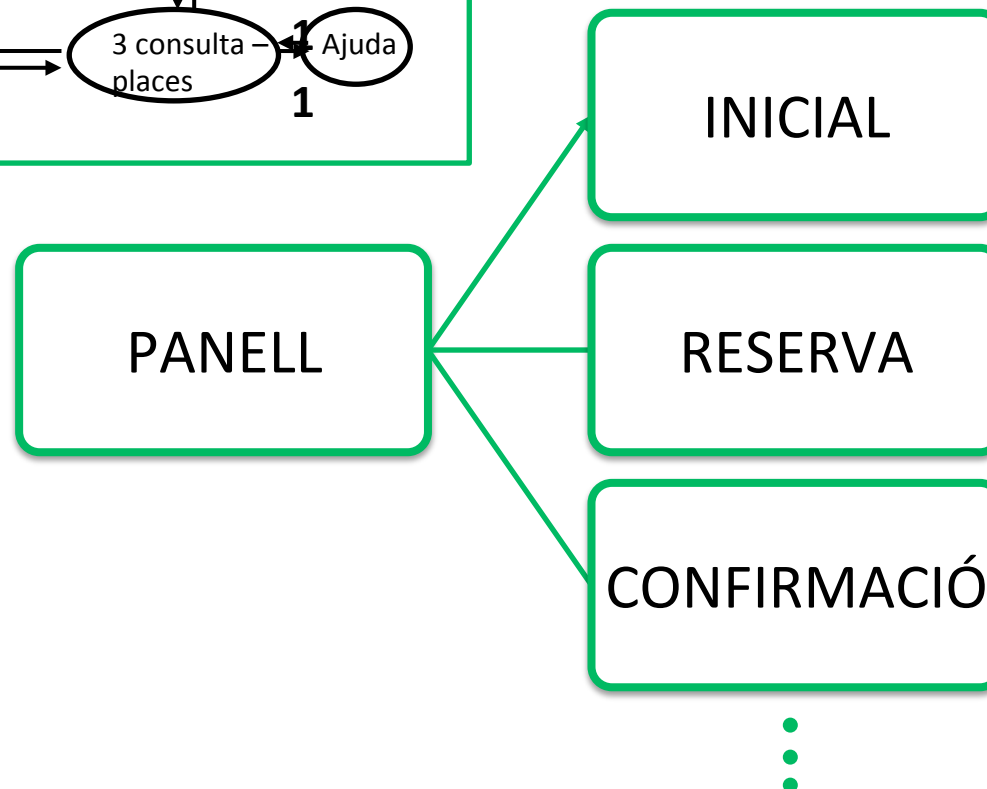
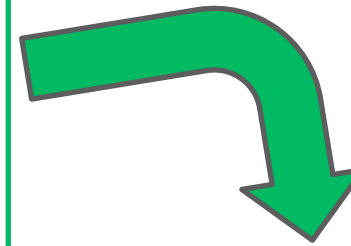
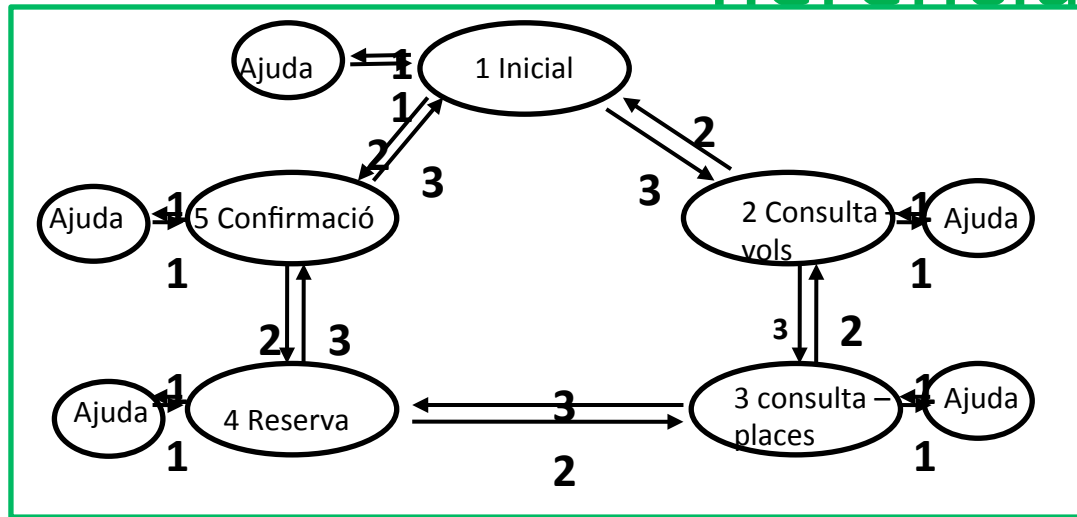
final

atributs

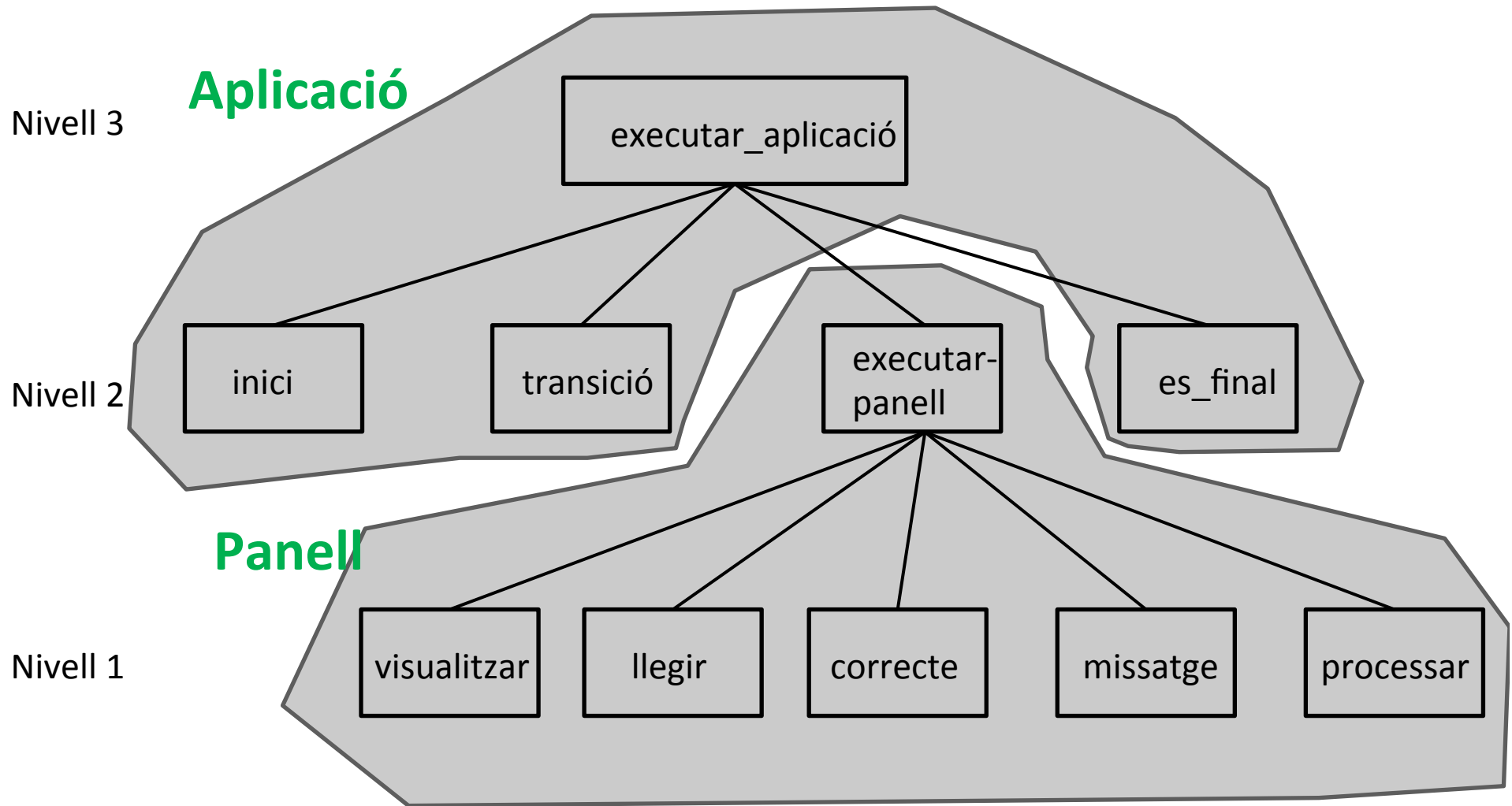
Rutina igual per a tots els panells

Rutines específiques de cada panell

Descomposició orientada a objectes: herència



Descomposició orientada a objectes: aplicació



Descomposició orientada a objectes: classe aplicació

```
classe APLICACIO caracteristica
    inicial: PANELL
    final: PANELL
    transicions: INTEGER [][]
    panells: PANELL []

    //funcions
    void setPanellInicial (PANELL p)
    PANELL getPanellInicial()
    boolean isFinal (PANELL p)
    void buildMatrixOfTransicions()
    ...
Final

    panells[inicial].executar();
```

Continguts

- Objectius
- Exercici
- Solucions
- **Conclusions**

Conclusions

- **Canvi de paradigma**
 - *From*: programació procedimental
 - Ens centrem en les funcions
 - Les funcions són l'element que més canvia en un programa (afegim, eliminem, canviem...)
 - *To*: programació orientada a objectes
 - Els programes es centren en les dades
 - Abstracció, reutilització, escalabilitat...

Conclusions

- **Elements importants** en aquest canvi de paradigma
 - Més enllà del 'funciona': ens interessa la **qualitat**
 - **Bona abstracció**: del problema i no del món
 - Estratègia: centrar-se en les **dades** / objectes

Per si voleu continuar pensant...

- Com es podria programar una interfície gràfica d'usuari (GUI) seguint el paradigma de la programació procedimental / descomposició funcional descendent?
 - GUI: finestra, botons, icones, imatges, vídeos...