Isabelle Tutorial:
System, HOL and Proofs

Burkhart Wolff, Makarius Wenzel

Université Paris-Sud

What we will talk about

What we will talk about

Isabelle with:

- its System Framework
- the Logical Framework
- the Isabelle/HOL Environment
- Proof Contexts and Structured Proof
- Tactic Proofs ("apply style")

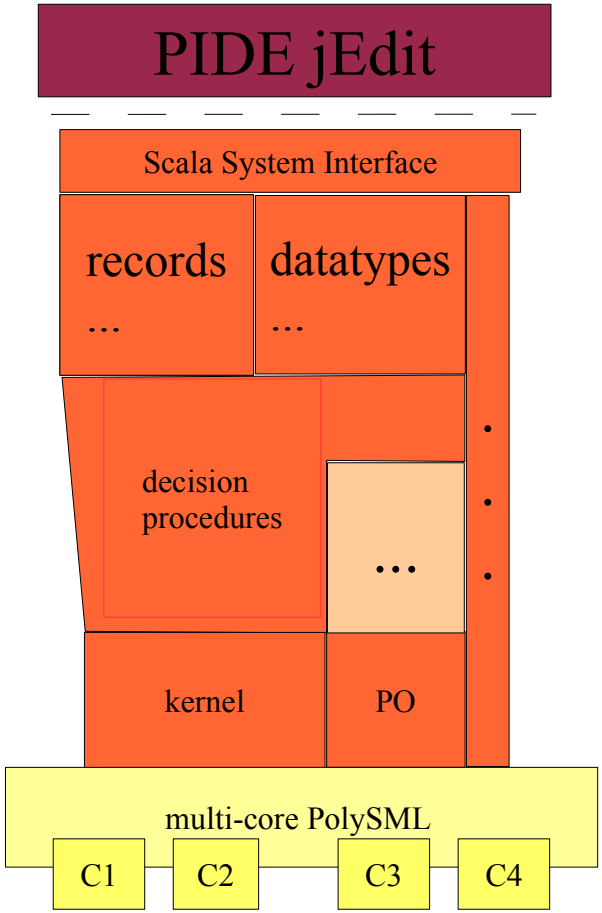
The Isabelle System Framework

Isabelle Architecture

- Modern Isabelle Architecture consists of 5 identifyable layers
  - SML layer
  - Kernel & Proof Object Layer
  - Tactic Layer and decision procedures
  - Isar Engine
  - PIDE Framework and Interface Layer

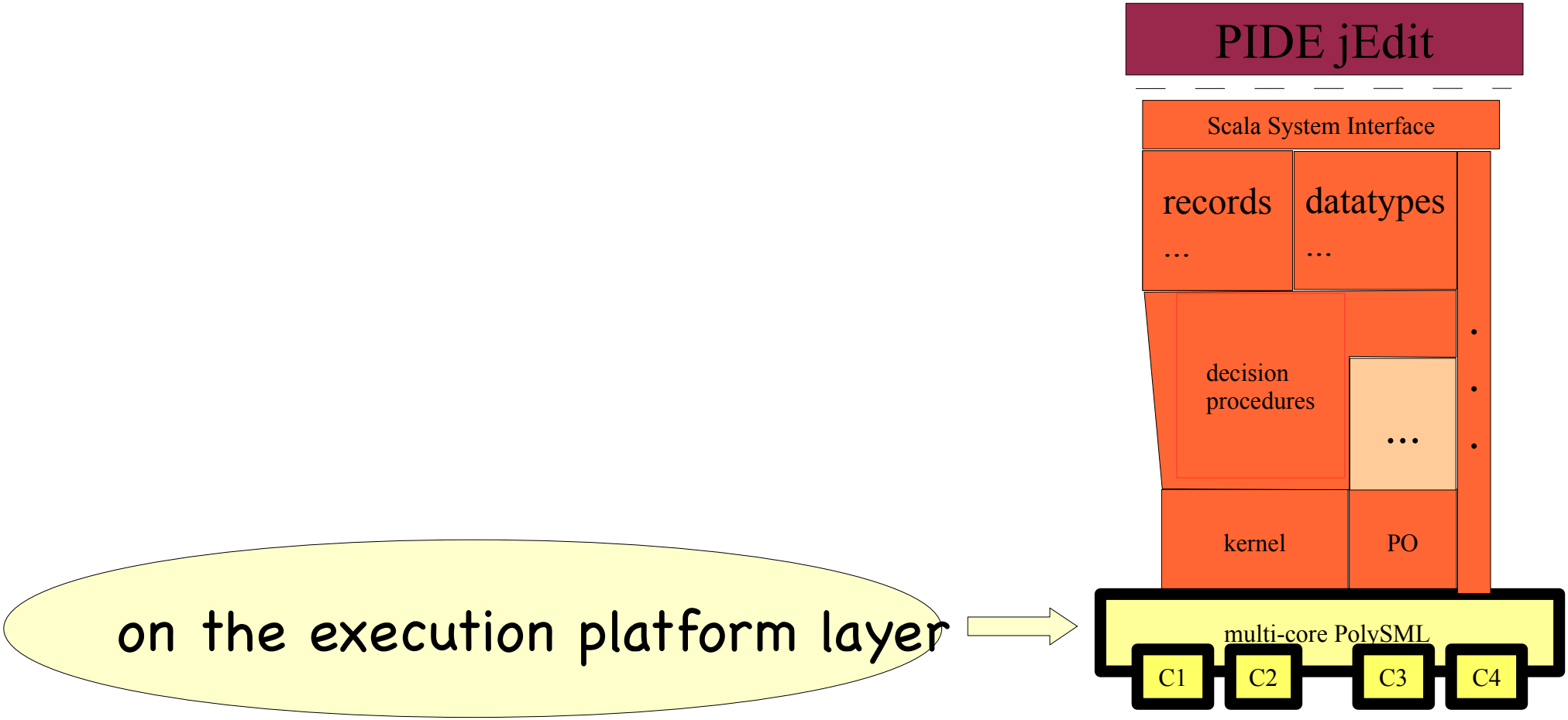
Isabelle Architecture

- Observation:
  Effective parallelization is a PERVASIVE PROBLEM, that must be addressed



Isabelle Architecture

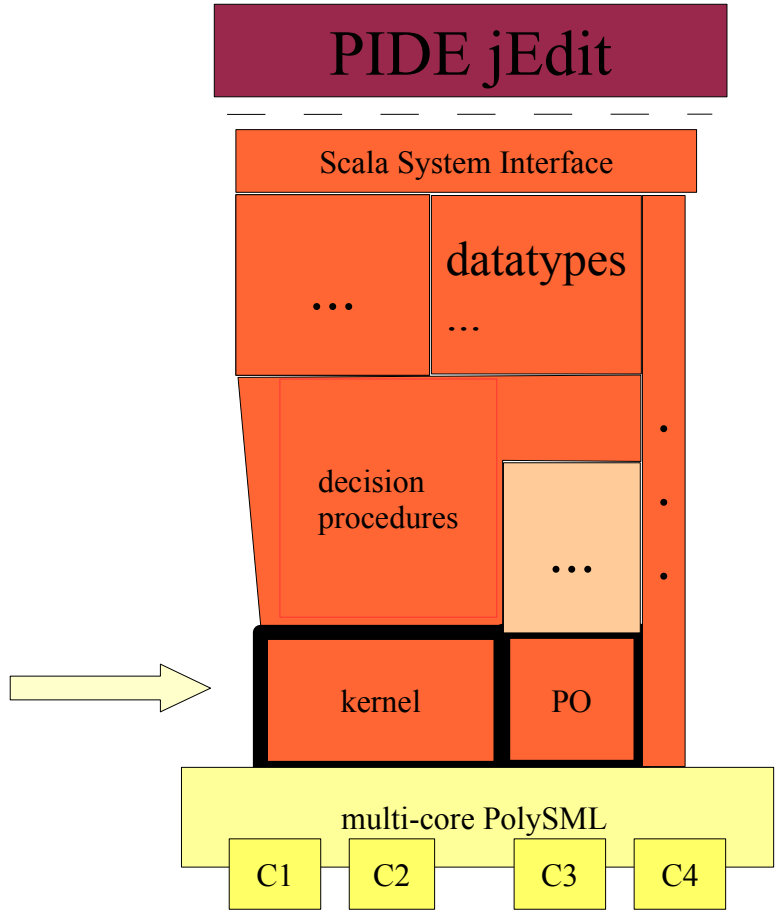
- In detail:



Isabelle Architecture

- In detail:

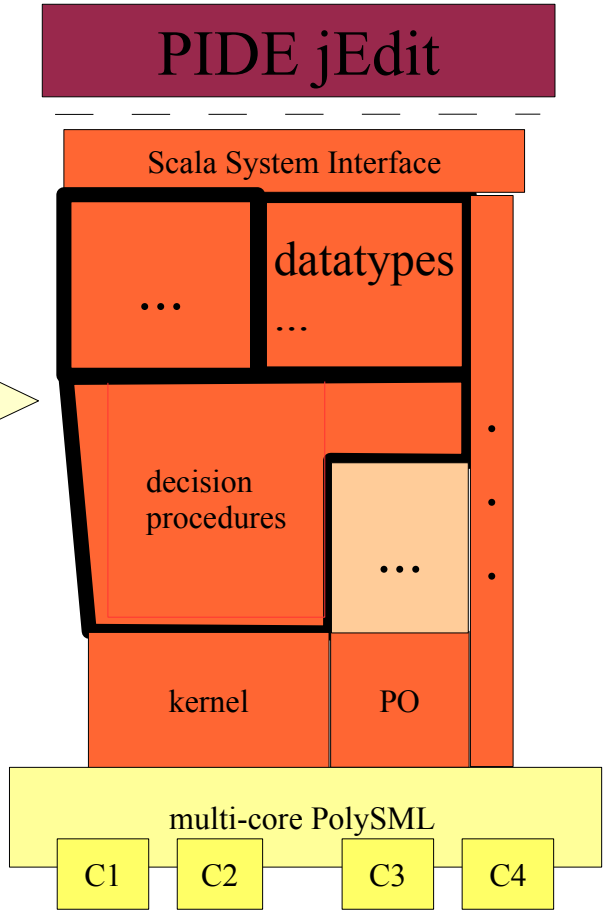
on the kernel layer



Isabelle Architecture

- In detail:

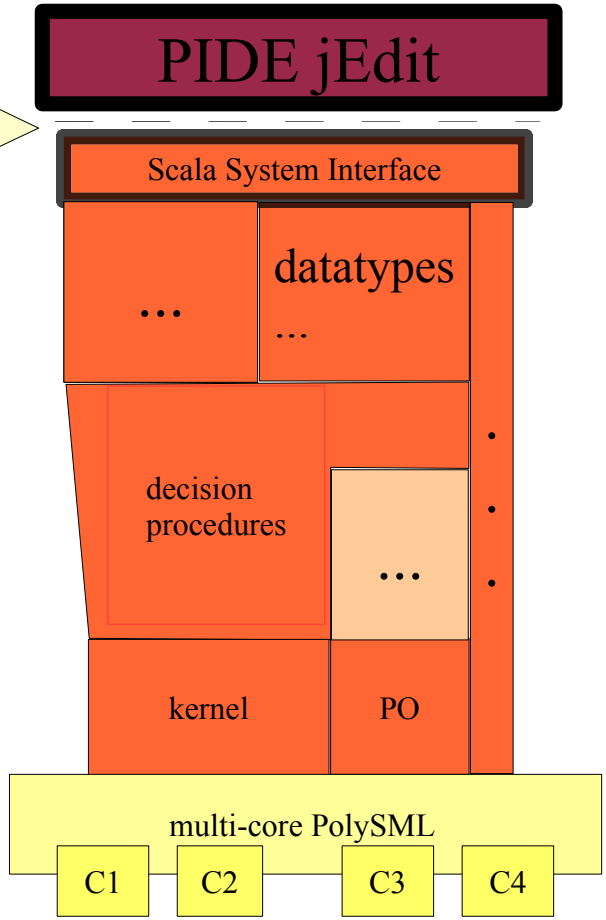
on layer of procedures and packages



Isabelle Architecture

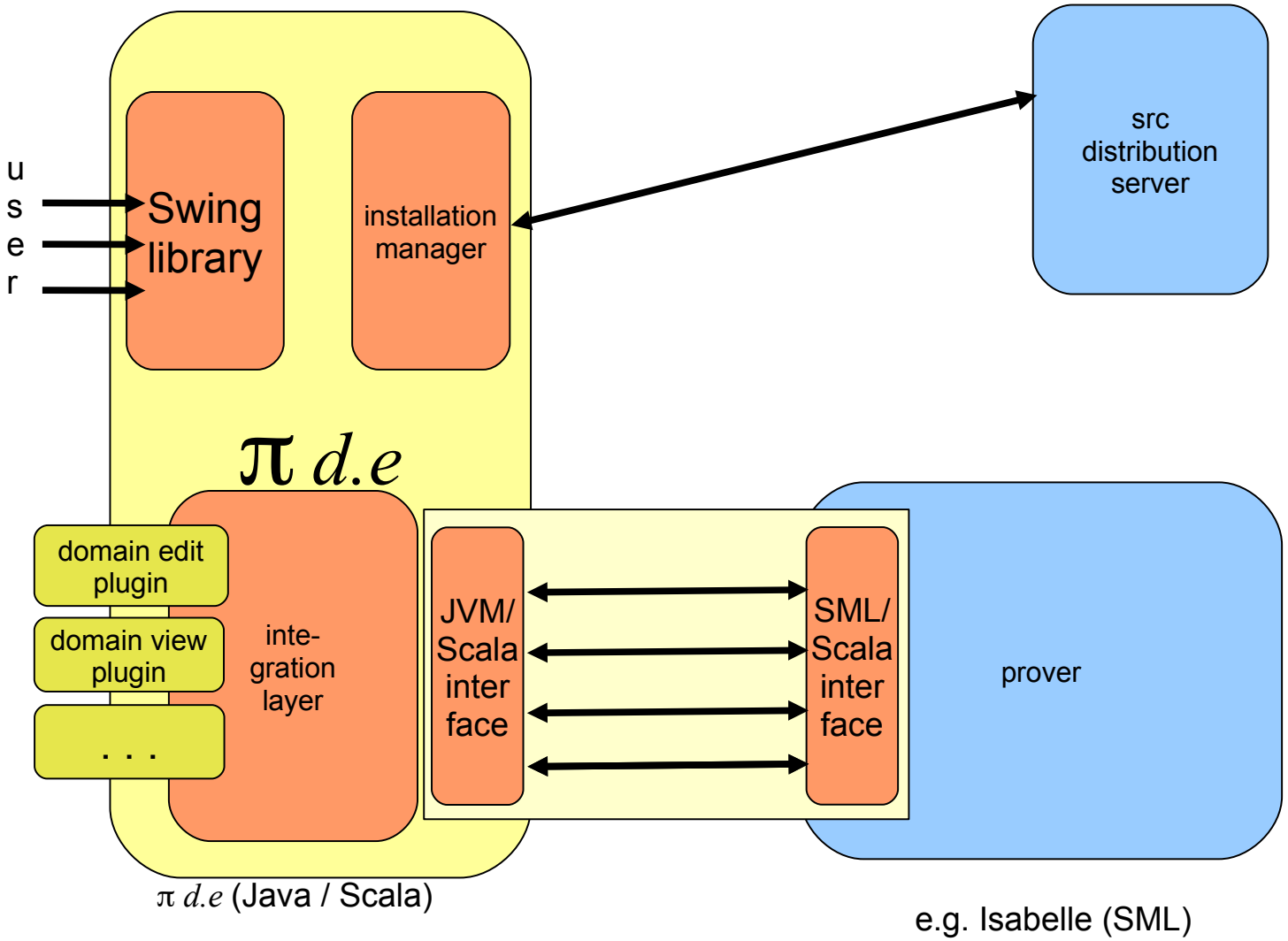
- In detail:

on the interface layer
PIDE framework + Editor



PIDE - GUI - Architecture

(see PIDE - Project: http://bitbucket.org/pide/pide/wiki/Manifesto)

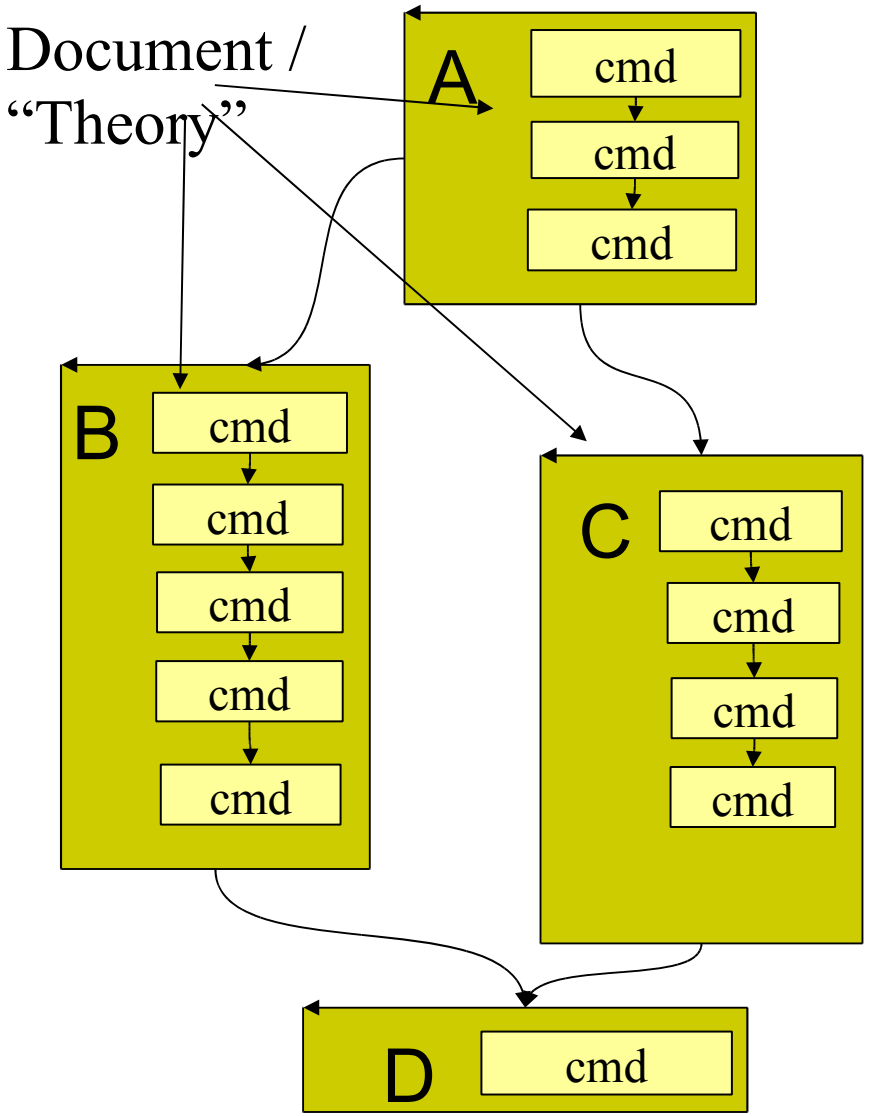


What is Isabelle as a System ?

- A Document Processor
  - … where documents have a unique name
  - … may acyclicly import documents
  - … and consists of an command sequence
  - … where new commands may be intro-duced on the fly (i.e. the system framework is extensible).
  - A session (a collection of documents organized in a hierachy) may be "frozen" to a session (or configuration)

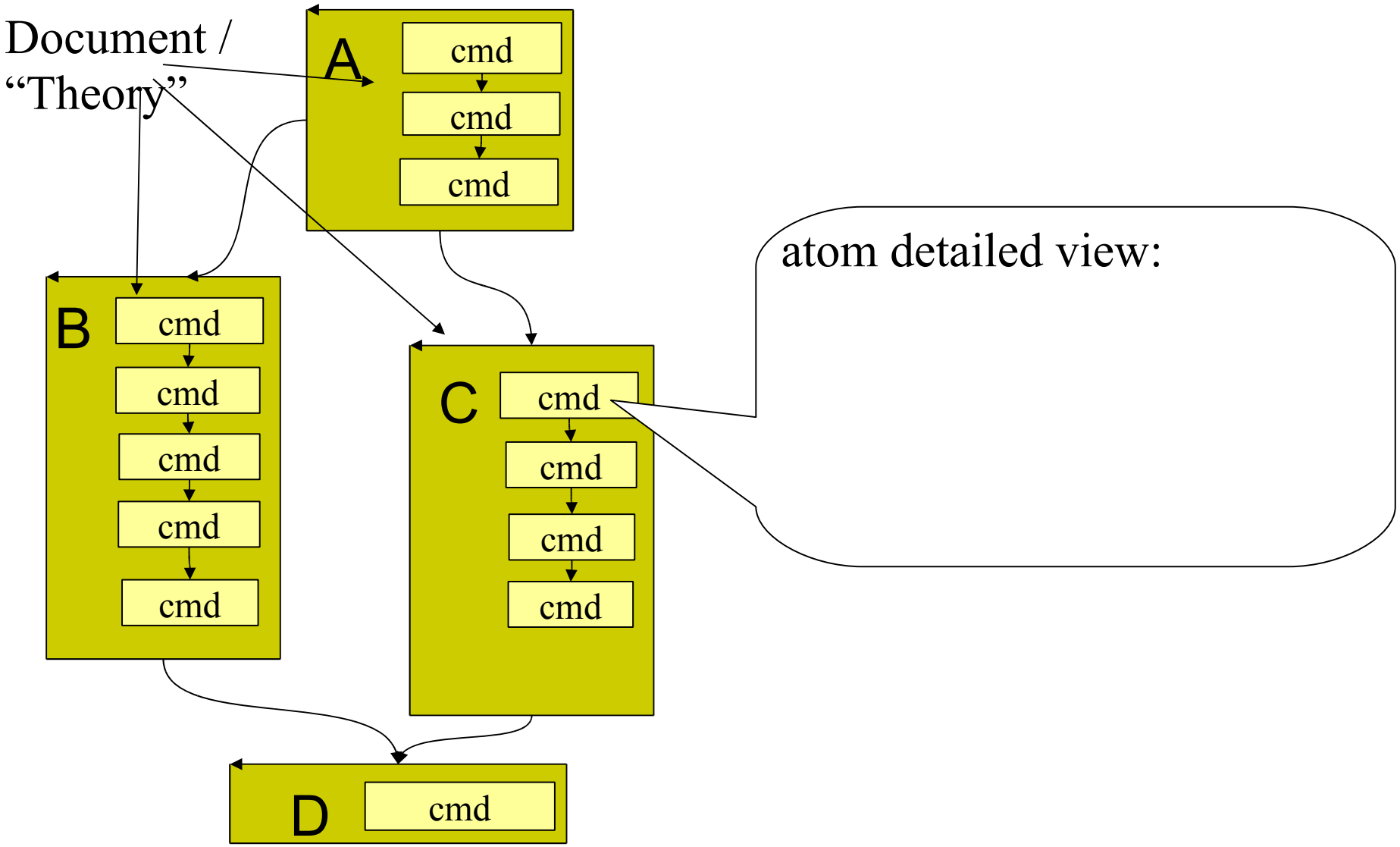
What is Isabelle as a System ?

- Global View of a “session“



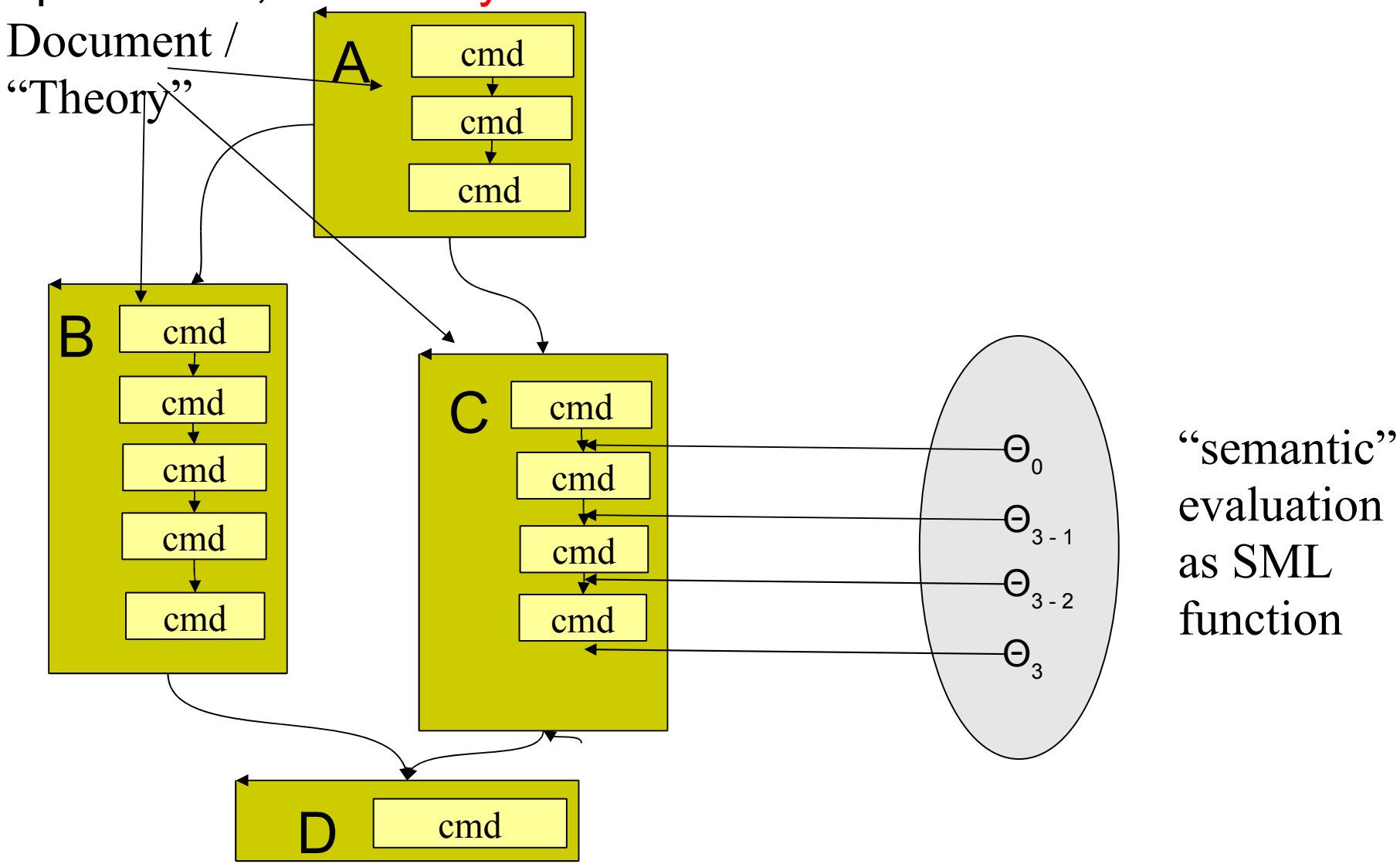
What is Isabelle as a System ?

- Global View



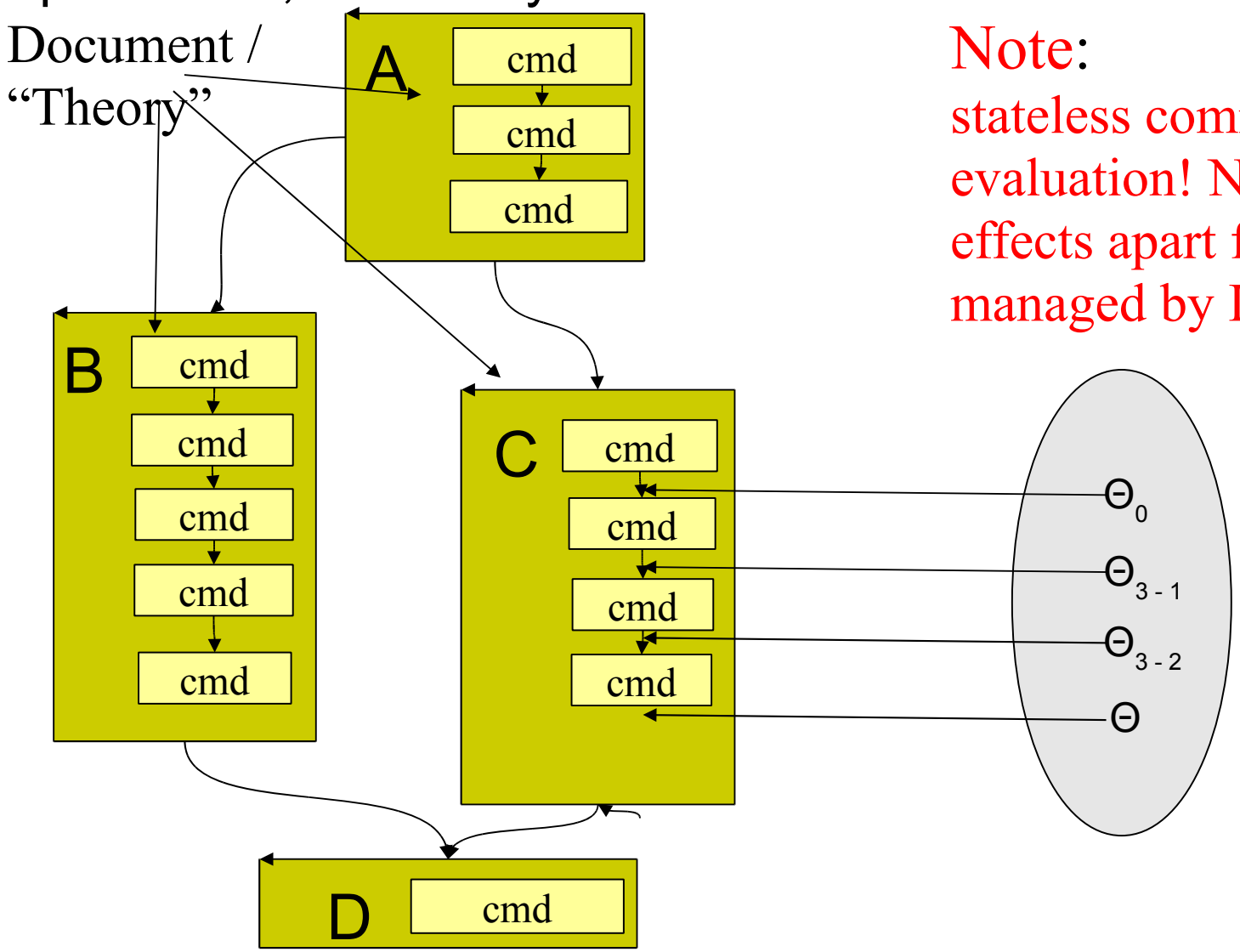
What is Isabelle as a System ?

- Document "positions" were evaluated to an implicit state, the theory context Θ



What is Isabelle as a System ?

- Document "positions" were evaluated to an implicit state, the theory context Θ

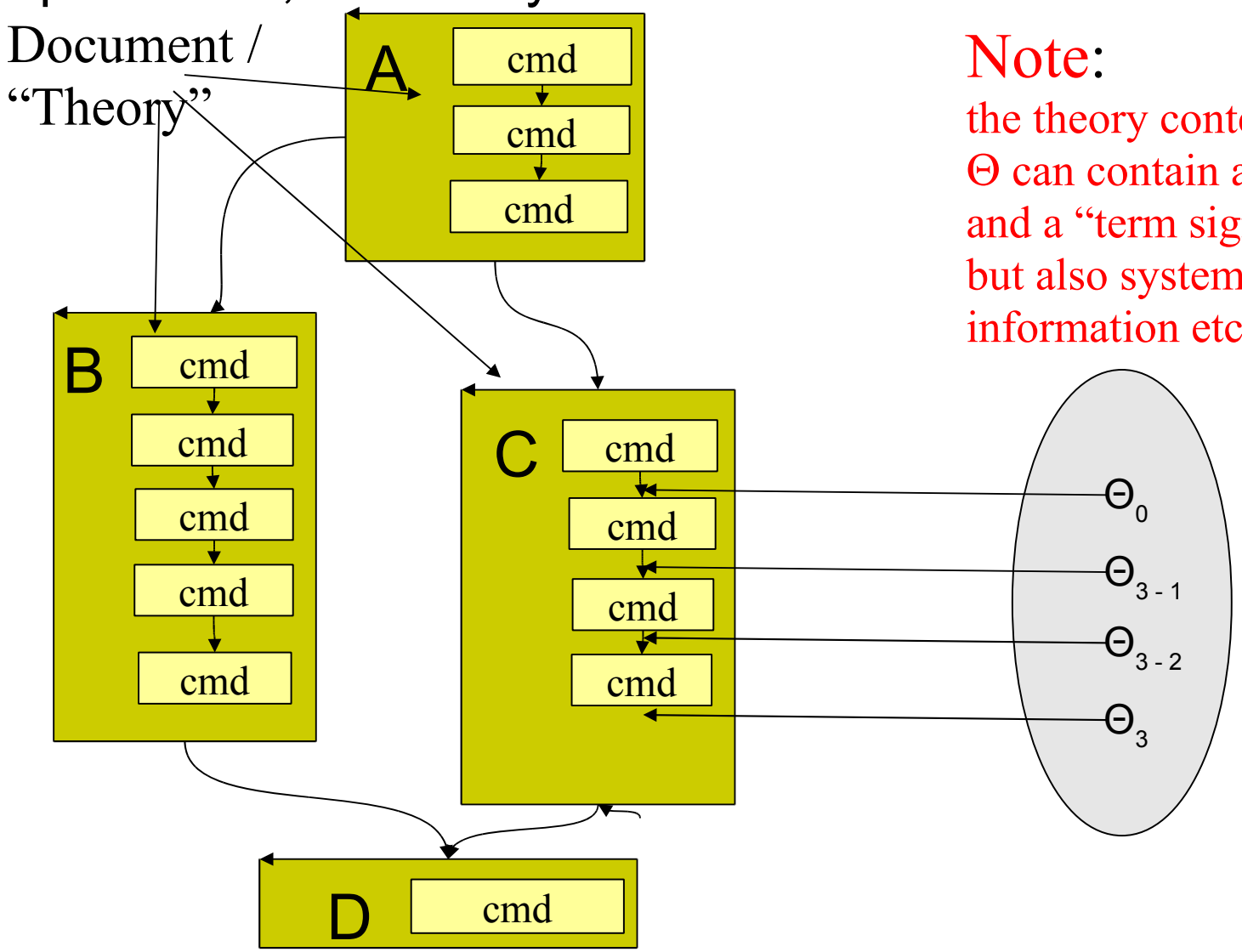


Note:
stateless command evaluation! No-side-effects apart from "state" managed by Isabelle !!!

"semantic" evaluation as SML function

What is Isabelle as a System ?

- Document “positions” were evaluated to an implicit state, the theory context Θ

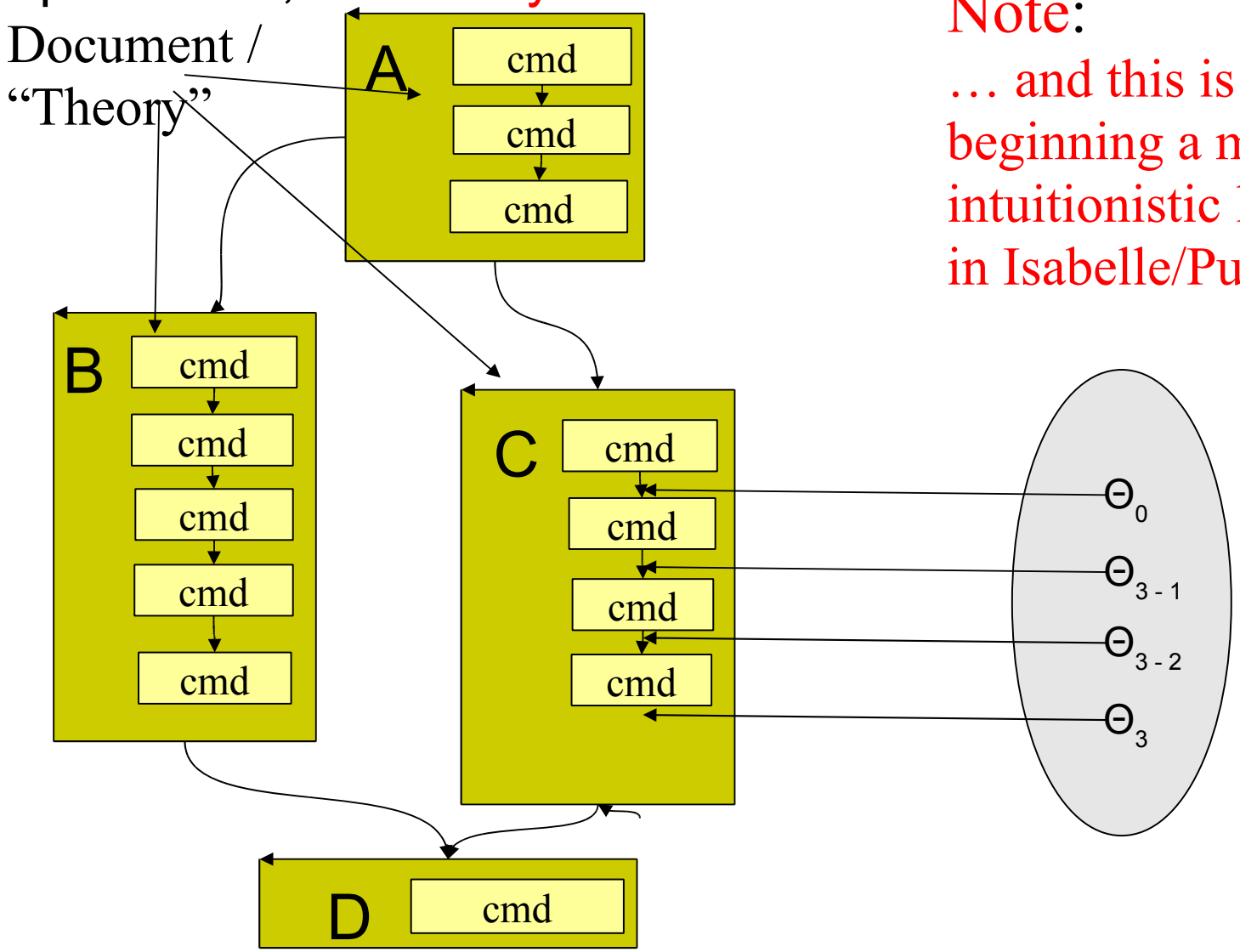


Note:
the theory context
Θ can contain a “type signature” and a “term signature”, “axioms”, but also system configuration information etc.

“semantic” evaluation as SML function

What is Isabelle as a System ?

- Document "positions" were evaluated to an implicit state, the theory context Θ



Note:
… and this is at the beginning a minimal intuitionistic logic called in Isabelle/Pure

"semantic" evaluation as SML function

What is Isabelle as a System ?

- Example

```
theory D
imports B C
begin
section{* First Section *}
text{* Some mathematical text: @{text \<alpha>}.*}
ML{* fun fac x = if x = 0 then 1 else x*fac(x-1) *}

ML{* fac 10 *}
end
```

What is Isabelle as a System ?

- Example

```
theory D
imports B C
begin
section{* First Section *}
text{* Some mathematical text: @{text \<alpha>}.*}
ML{*  fun fac x = if x = 0 then 1 else x*fac(x-1) *}
ML{*  fac 10  *}
end
```

What is Isabelle as a System ?

- Example

```
theory D
imports B C
begin
section{* First Section *}
text{* Some mathematical text: @{text \<alpha>}.*}
ML{* fun fac x = if x = 0 then 1 else x*fac(x-1) *}
ML{* fac 10 *}
end
```

"fac" visible here because the ML environment is part of Θ !!

Demo I

- Start Isabelle (via the PIDE jEdit)
- Browse „demo1.thy“
- Commands:
    - text, section, subsection
    - ML
    - value
    - a browser for theorems: find_theorems
- Capabilities:
    - hovering, jump-link,

Demo I

demo1.thy (modified)

demo1.thy (~/u-psud/fortesse/pub/presentations/2014-14-9-isabelle-tutorial/bu_sol...

```
transcription, so \alpha is just equal to \<alpha>  but
can also be written α.

Only in few cases one has to memorize. For them,
ASCII - oriented shortcuts like ⇒ can be given for =>.

*}

subsection{* Apotheosis *}

text{* It may be necessary to get used to the PIDE - Paradigm:
always checking whenever typing. After a while, however,
one gets used to it. Don't forget to save from time to time !!! *}

subsection{* "The Function" in SML *}

ML{* fun fac n = if n=0 then 1 else n * fac(n-1) *}
ML{* fac 50*}

subsection{* Using the code-generator to SML *}

value "(2::nat) + 2"
```

Auto update | Update | Detach | 100%

```
val it =
  30414093201713378043612608166064768844377641568960512000000000000: int
```

Find | Output | Sledgehammer | Symbols

isabelle

Filter:

```
demo1.thy
▼ demo1
    theory demo1
  ▼ section{* My very first experiments *}
    ▶ subsection{* Thesis *}
    ▶ subsection{* Apotheosis *}
    ▼ subsection{* "The Function" in SML *}
        ML{* fun fac n = if n=0 then 1 else n * fac(n-
        ML{* fac 50*}
    ▶ subsection{* Using the code-generator to SML *}
```

```
subsection{* Using the code-generator to SML *}
```

Documentation | Sidekick | Theories

31,12 (798/909) (isabelle,sidekick,UTF-8-Isabelle)Nm r o UG 257/333MB 14:11

Demo I

Main (Editing) Panel

demo1.thy (modified)

demo1.thy (~/u-psud/fortesse/pub/presentations/2014-14-9-isabelle-tutorial/bu_sol...

```
transcription, so \alpha is [illegible] alpha> but
can also be written [illegible].

Only in few cases one has to memorize. For them,
ASCII oriented shortcuts like ⇒ can be given for =>.

*)

subsection{* Apotheosis *}

text{* It may be necessary to get used to the PIDE - Paradigm:
always checking whenever typing. After a while, however,
one gets used to it. Don't forget to save from time to time !!! *}

subsection{* "The Function" in SML *}

ML{* fun fac n = if n=0 then 1 else n * fac(n-1) *}
ML{* fac 50*}

subsection{* Using the code-generator to SML *}

value "(2::nat) + 2"
```

Auto update | Update | Detach | 100%

```
val it =
  30414093201713378043612608166064768844377641568960512000000000000: int
```

Find | Output | Sledgehammer | Symbols

31,12 (798/909)

isabelle

Filter:

```
demo1.thy
▼ demo1
    theory demo1
  ▼ section{* My very first experiments *}
    ▶ subsection{* Thesis *}
    ▶ subsection{* Apotheosis *}
    ▼ subsection{* "The Function" in SML *}
        ML{* fun fac n = if n=0 then 1 else n * fac(n-
        ML{* fac 50*}
    ▶ subsection{* Using the code-generator to SML *}
```

Documentation | Sidekick | Theories

```
subsection{* Using the code-generator to SML *}
```

(isabelle,sidekick,UTF-8-Isabelle) UG 257/333MB 14:11

Demo I

demo1.thy (modified)

demo1.thy (~/u-psud/fortesse/pub/presentations/2014-14-9-isabelle-tutorial/bu_sol...

```
transcription, so \alpha is just equal to \<alpha>  but
can also be written α.

Only in few cases one has to memorize. For them,
ASCII - oriented shortcuts like ⇒ can be given for =>.

*}

subsection{* Apotheosis *}

text{* It may be necessary to get used to the PIDE - Paradigm:
always checking whenever typing. After a while, however,
one gets used to it. Don't forget to save from time to time !!! *}

subsection{* "The Function" in SML *}

ML{* fun fac n = if n=0 then 1 else n * fac(n-1) *}
ML{* fac 50*}

subsection{* Using the code-generator to SML *}

value "(2::nat) + 2"
```

Auto update | Update | Detach | 100%

```
val it =
  30414093201713378043612608166064768844377641568960512000000000000: int
```

Find | Output | Sledgehammer | Symbols

31,12 (798/909)

isabelle

Filter:

```
demo1.thy
▼ demo1
    theory demo1
  ▼ section{* My very first experiments *}
    ▶ subsection{* Thesis *}
    ▶ subsection{* Apotheosis *}
    ▼ subsection{* "The Function" in SML *}
        ML{* fun fac n = if n=0 then 1 else n * fac(n-
        ML{* fac 50*}
    ▶ subsection{* Using the code-generator to SML *}
```

Documentation | Sidekick | Theories

```
subsection{* Using the code-generator to SML *}
```

(isabelle,sidekick,UTF-8-Isabelle) UG 257/333MB 14:11

Output Panel

Demo I

Sidekick Panel/
[Documentation
Panel |
Theories Panel]

demo1.thy (modified)

demo1.thy (~/u-psud/fortesse/pub/presentations/2014-14-9-isabelle-tutorial/bu_sol...

```
transcription, so \alpha is just equal to \<alpha>  but
can also be written α.

Only in few cases one has to memorize. For them,
ASCII - oriented shortcuts like ⇒ can be given for =>.

*}

subsection{* Apotheosis *}

text{* It may be necessary to get used to the PIDE - Paradigm:
always checking whenever typing. After a while, however,
one gets used to it. Don't forget to save from time to time !!! *}

subsection{* "The Function" in SML *}

ML{* fun fac n = if n=0 then 1 else n * fac(n-1) *}
ML{* fac 50*}

subsection{* Using the code-generator to SML *}

value "(2::nat) + 2"
```

Auto update | Update | Detach | 100%

```
val it =
  30414093201713378043612608166064768844377641568960512000000000000: int
```

Find | Output | Sledgehammer | Symbols

isabelle

Filter:

```
demo1.thy
  demo1
    theory demo1
    section{* My very first experiments *}
      subsection{* Thesis *}
      subsection{* Apotheosis *}
      subsection{* "The Function" in SML *}
        ML{* fun fac n = if n=0 then 1 else n * fac(n-1
        ML{* fac 50*}
      subsection{* Using the code-generator to SML *
```

Documentation | Sidekick | Theories

```
subsection{* Using the code-generator to SML *}
```

31,12 (798/909) (isabelle,sidekick,UTF-8-Isabelle) 257/333MB 14:11

Exercises

- Start Isabelle
  (via the PIDE/jEdit: isabelle jedit demo1.thy)
- Explore Demo1.thy: Set output window, modify texts and value-computations, ML-code.
- Browse „Editor.thy";
- Edit a (brief) document with mathematical notation.
- Edit and evaluate a small SML program
  (see http://en.wikipedia.org/wiki/Standard_ML
  as primer)

Parallel Nano-Kernel LCF-Archi-tecture

in the

jEdit - GUI (PIDE)

fine-grained, asynchronous parallelism (Isabelle2009-2)

Example.thy (modified)

Example.thy (~/tmp/)

```
theory Example
imports Main
begin

inductive path for rel :: "'a ⇒ 'a ⇒ bool" where
  base: "path rel x x"
| step: "rel x y ⟹ path rel y z ⟹ path rel x z"

theorem example:
  fixes x z :: 'a assumes "path rel x z" shows "P x z"
  using assms
proof induct
  case (base x)
  show "P x x" by auto
next
  case (step x y z)
  note `rel x y` and `path rel y z`
  moreover note `P y z`
  ultimately show "P x z" by auto
qed

end
```

16,20 (318/422) (isabelle,none,UTF-8-Isabelle)- --- UG 68/554Mb 1:41 PM