Executable Formal Semantics for the POSIX Shell

Michael Greenberg *and* Austin J. Blatt

Pomona College *Pomona '18, now Puppet Labs*

i’m interested in

powerful

programming languages

you want power?

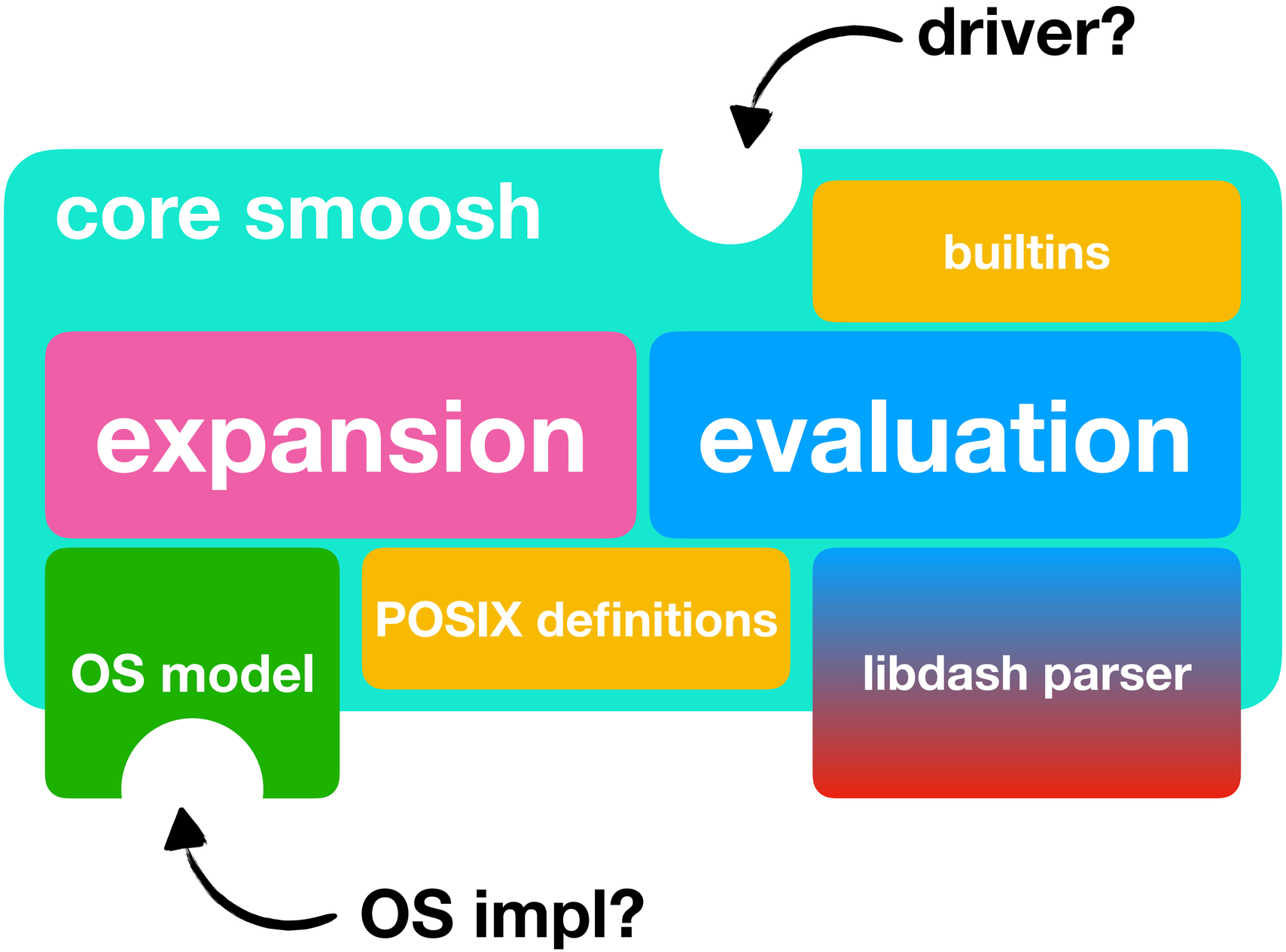
you want the

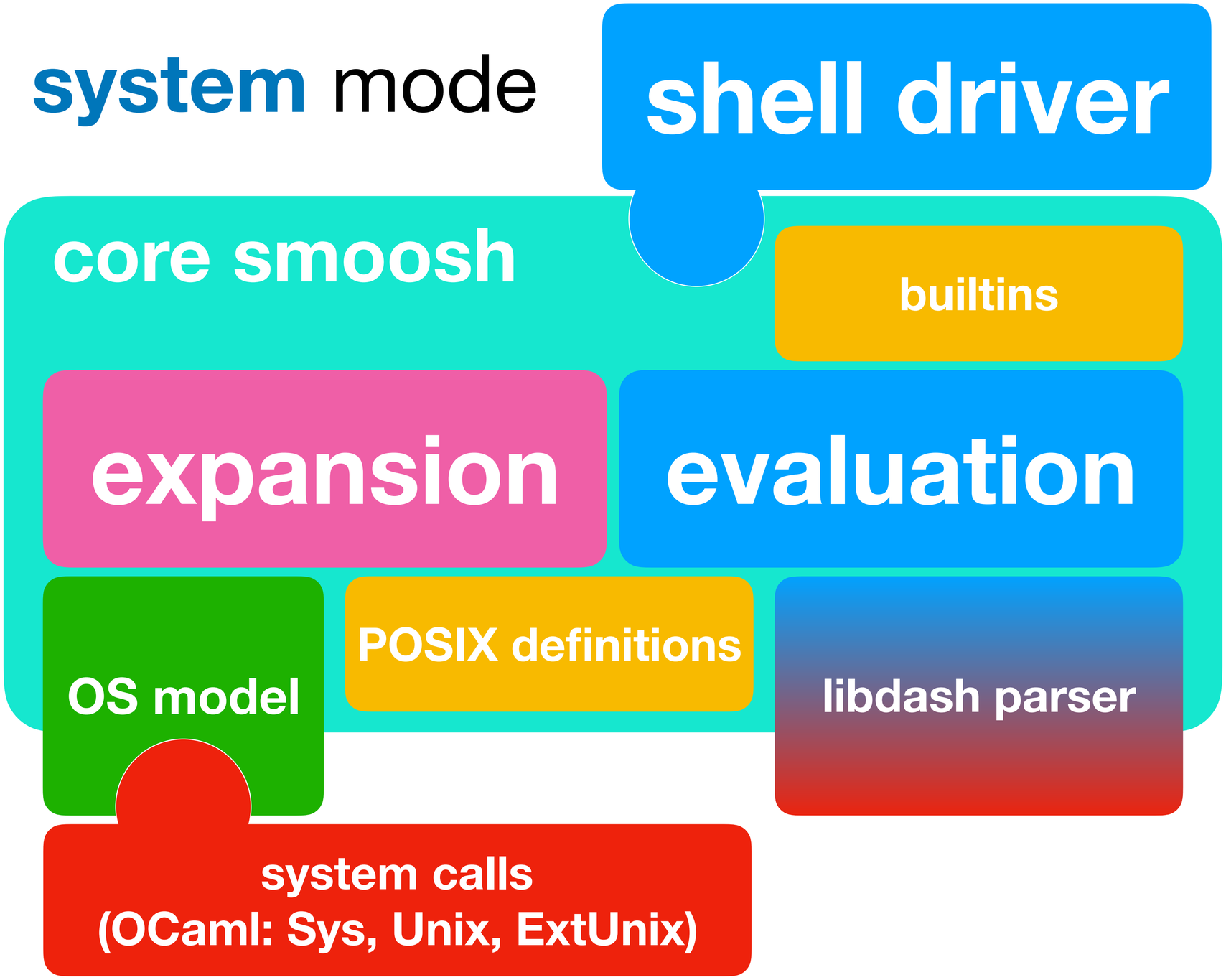
shell

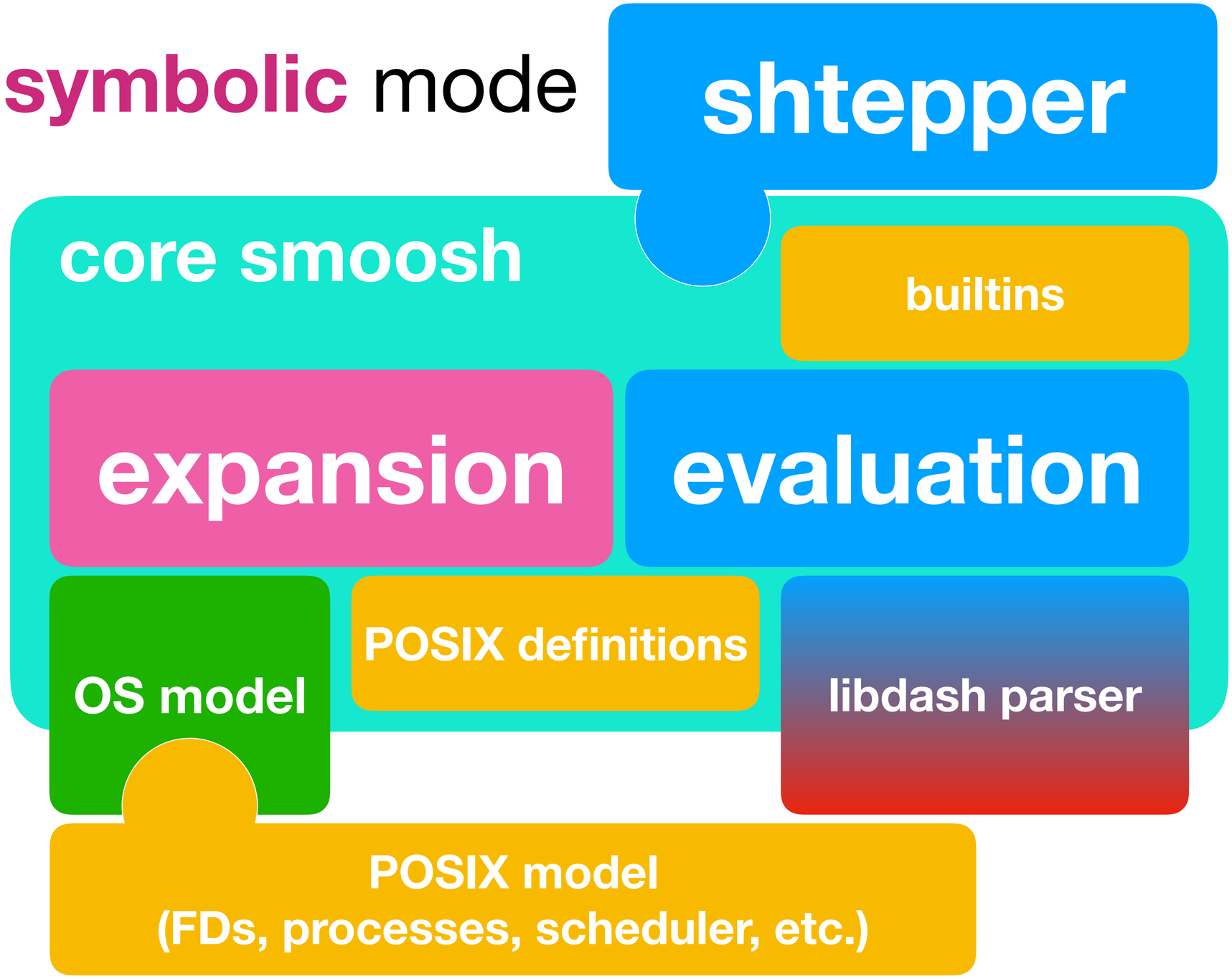
i wrote a new POSIX shell

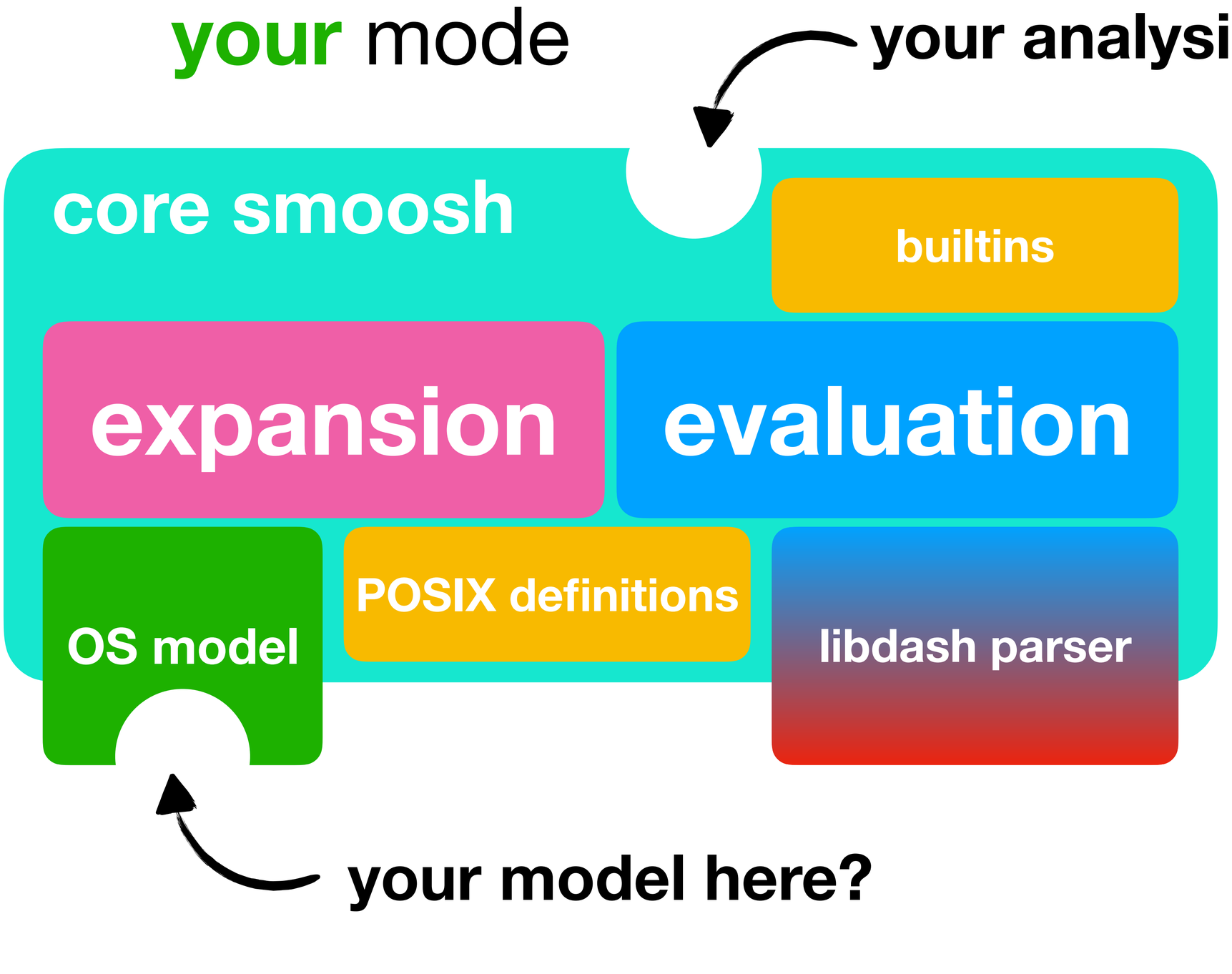
smoosh

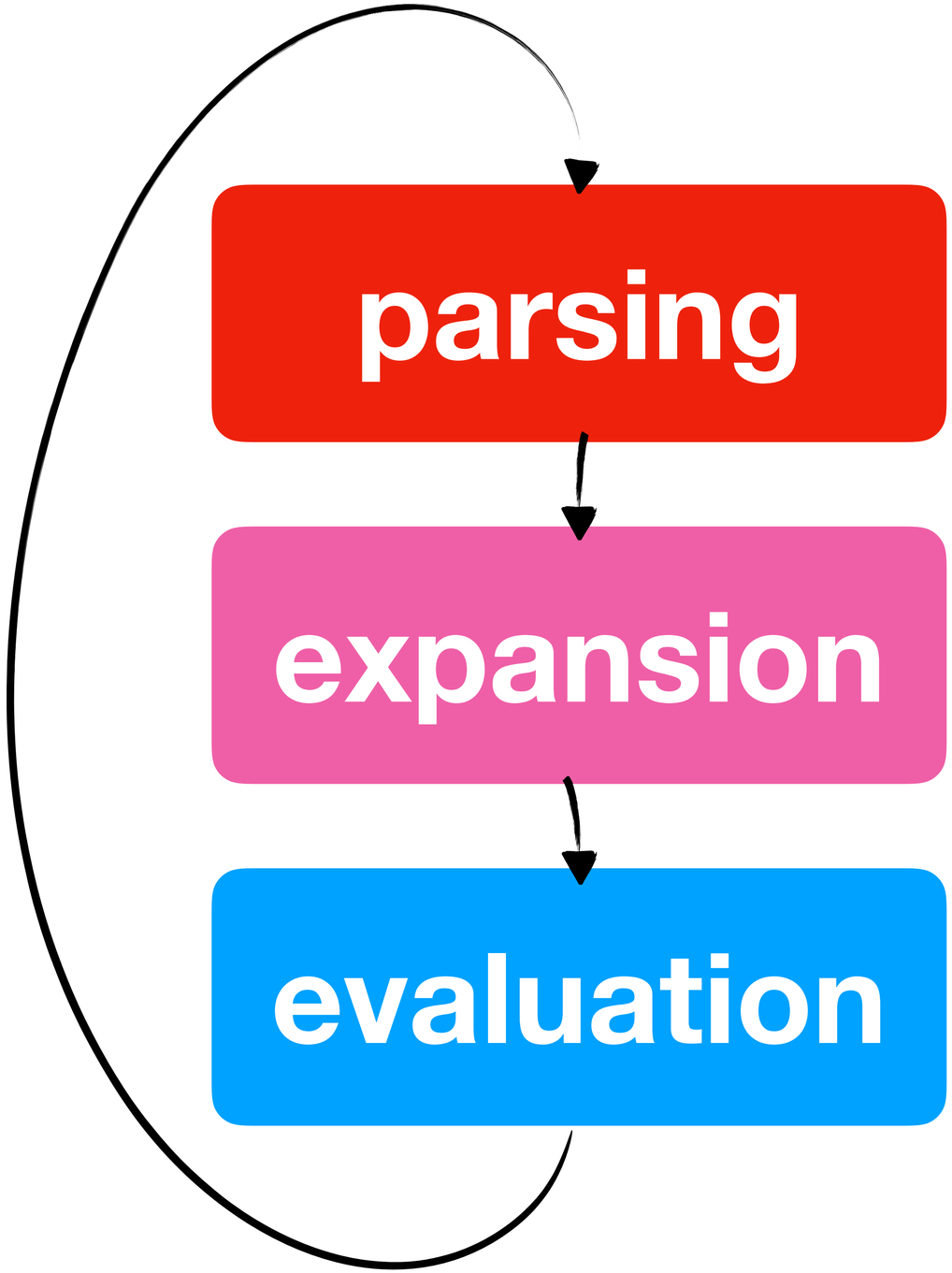
the **S**ymbolic, **M**echanized, **O**bservable, **O**perational **SH**ell


driver?
core smoosh
builtins
expansion
evaluation
OS model
POSIX definitions
libdash parser
OS impl?

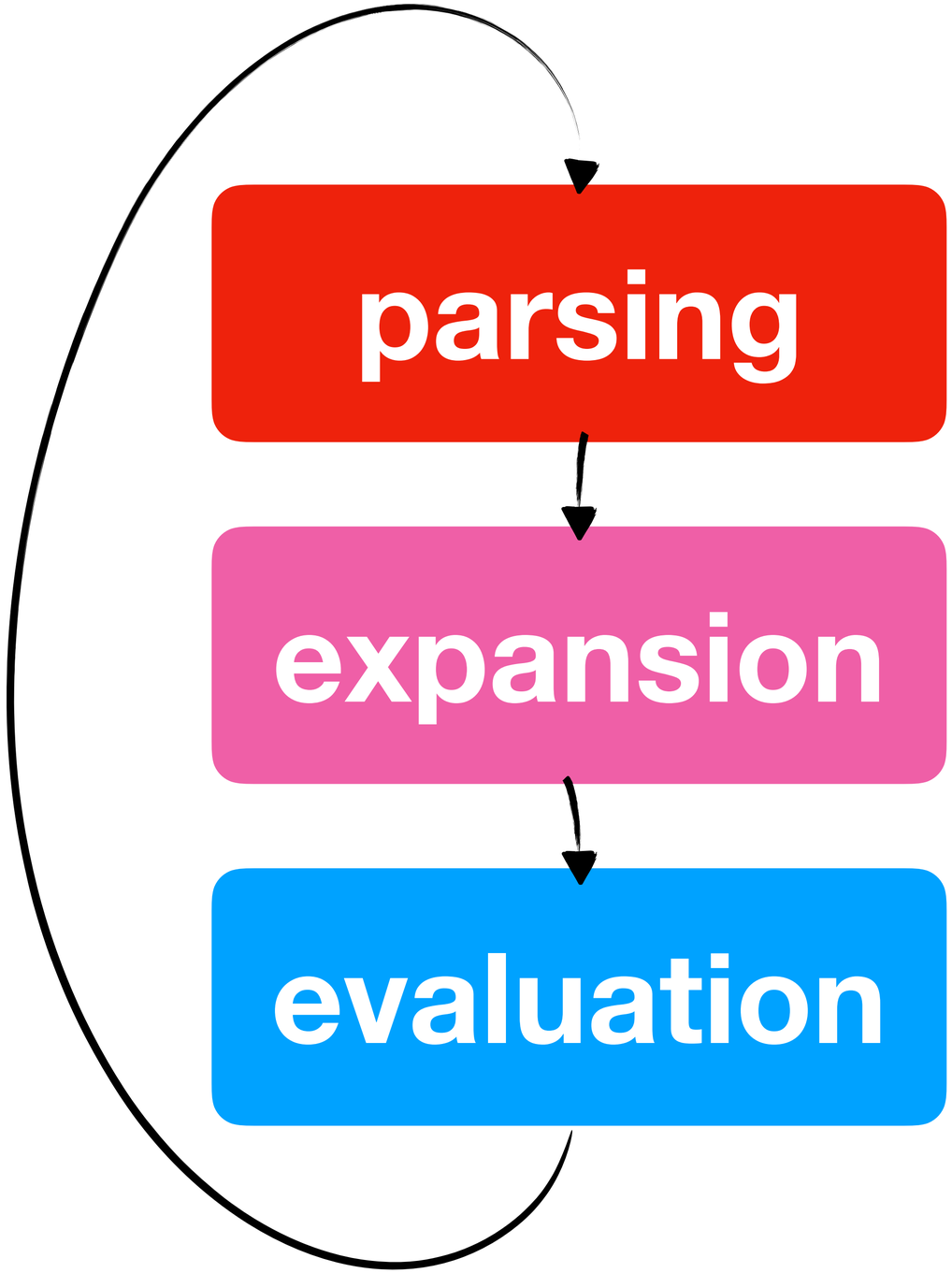

system mode
shell driver
core smoosh
builtins
expansion
evaluation
OS model
POSIX definitions
libdash parser
system calls
(OCaml: Sys, Unix, ExtUnix)


symbolic mode
shtepper
core smoosh
builtins
expansion
evaluation
OS model
POSIX definitions
libdash parser
POSIX model
(FDs, processes, scheduler, etc.)

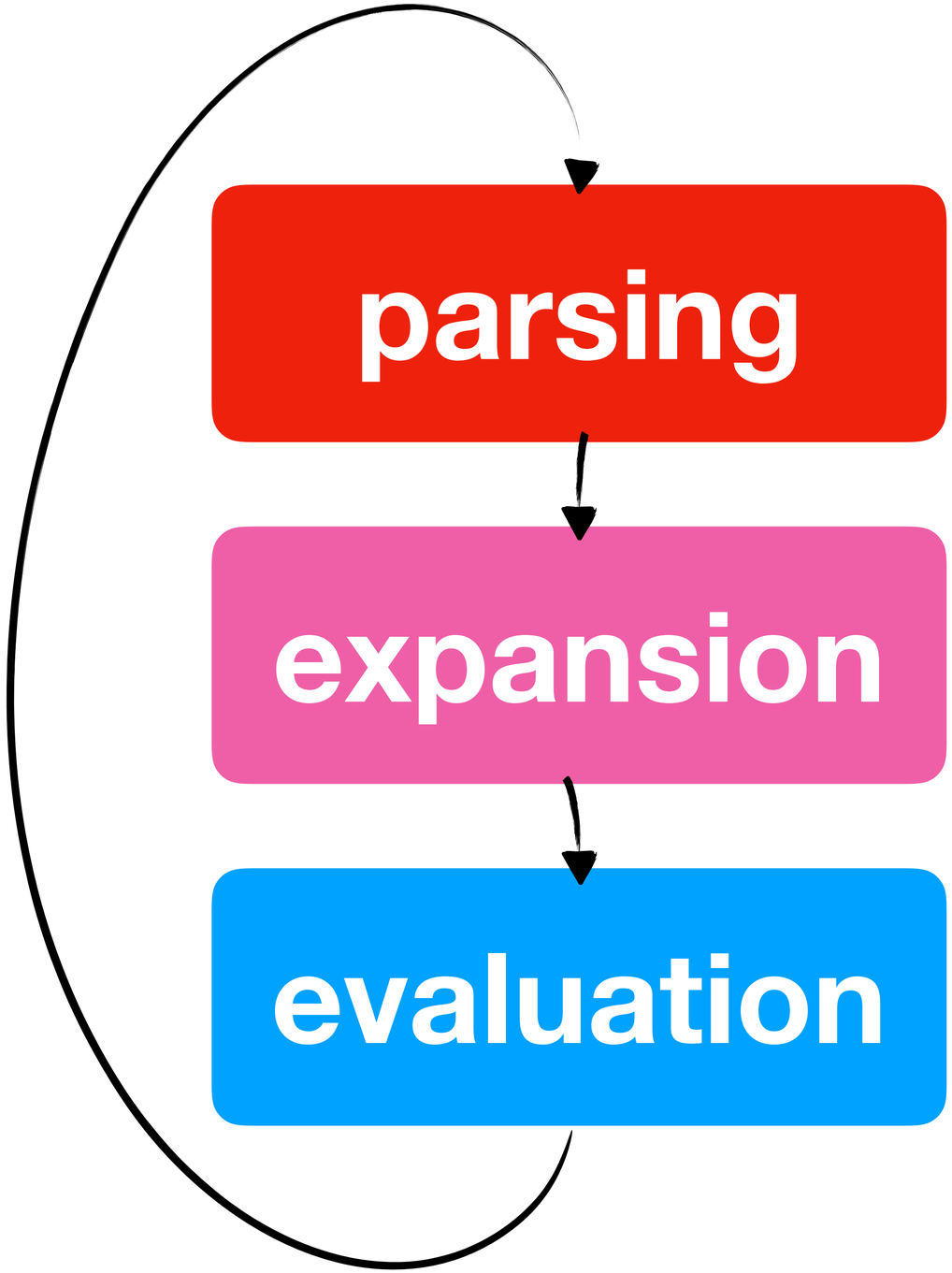

your mode
your analysis here?
core smoosh
builtins
expansion
evaluation
OS model
POSIX definitions
libdash parser
your model here?



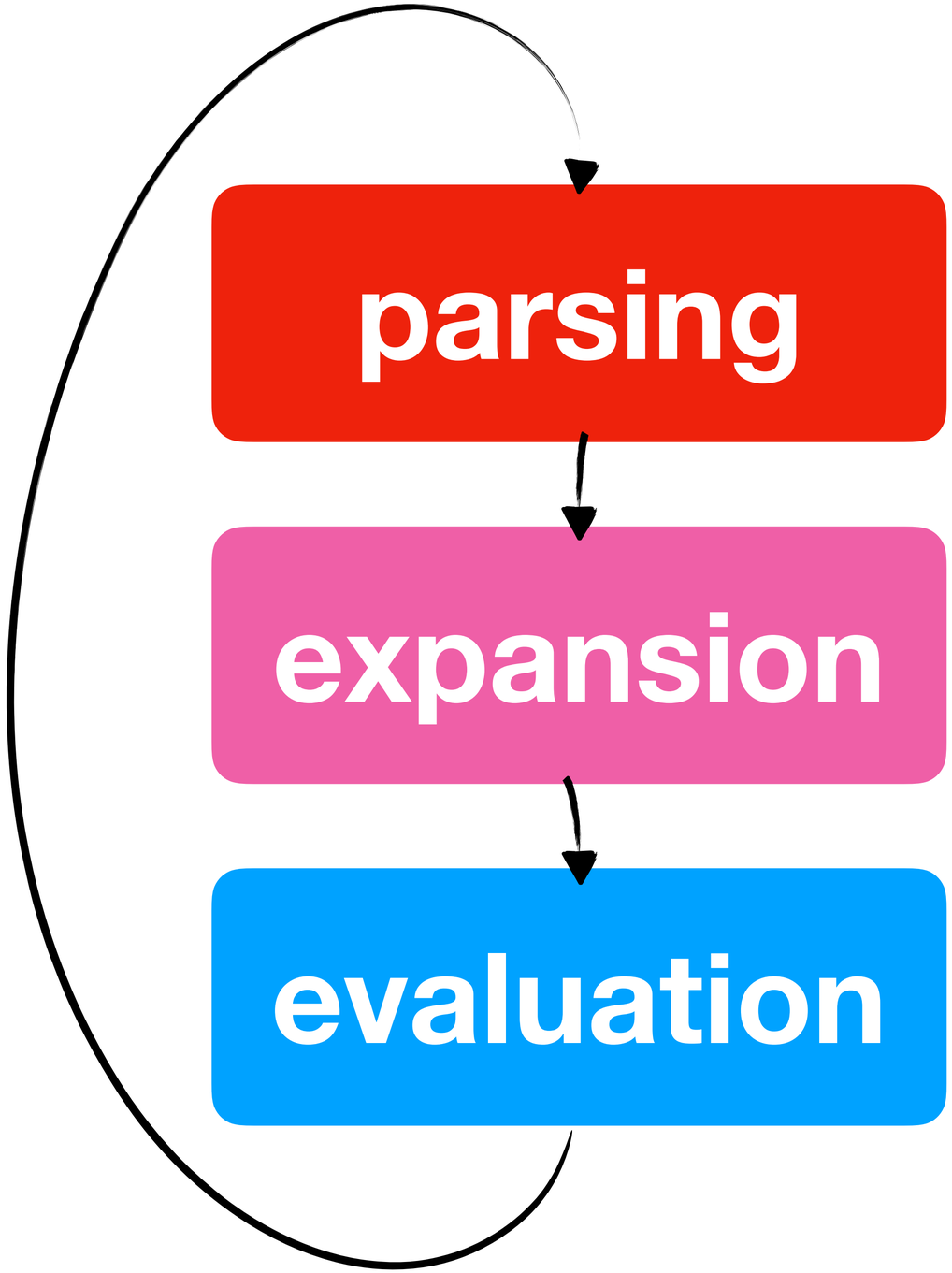
echo ~	→	/Users/mgree
echo ${PWD}	→	/Users/mgree/talks/smoosh
basename `pwd`	→	smoosh
echo $((1+1))	→	2
IFS=“” cat `echo some file`	→	[shows contents of ‘some file’]
echo *	→	abstract.txt posix.key some file
echo you can “” me	→	you can me



echo ~	→	/Users/mgree
echo ${PWD}	→	/Users/mgree/talks/smoosh
basename `pwd`	→	**smoosh**
echo $((1+1))	→	2
IFS=“” cat `echo some file`	→	[shows contents of ‘some file’]
echo *	→	abstract.txt posix.key some file
echo you can “” me	→	you can me

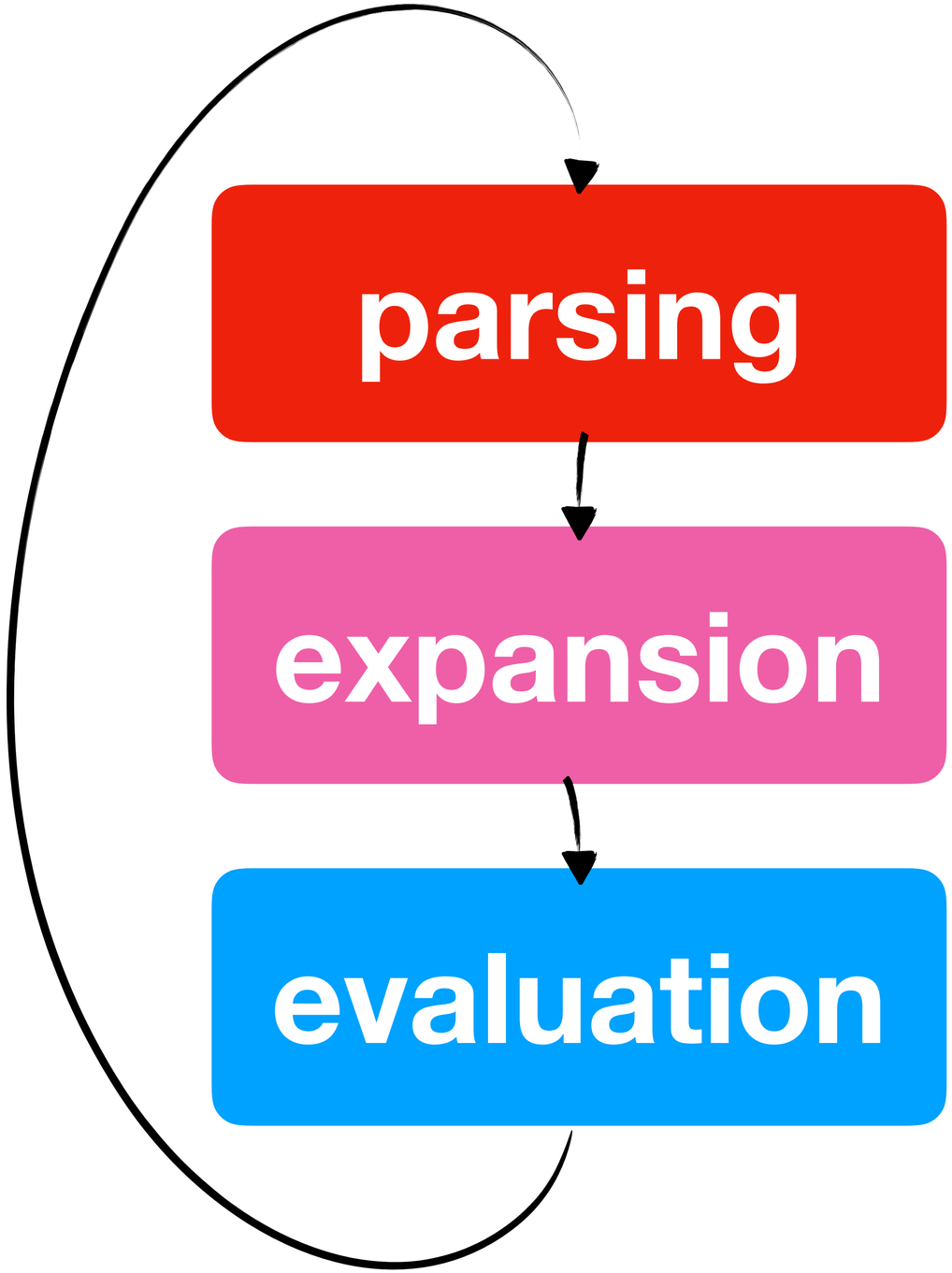


basename `pwd`



basename `pwd`

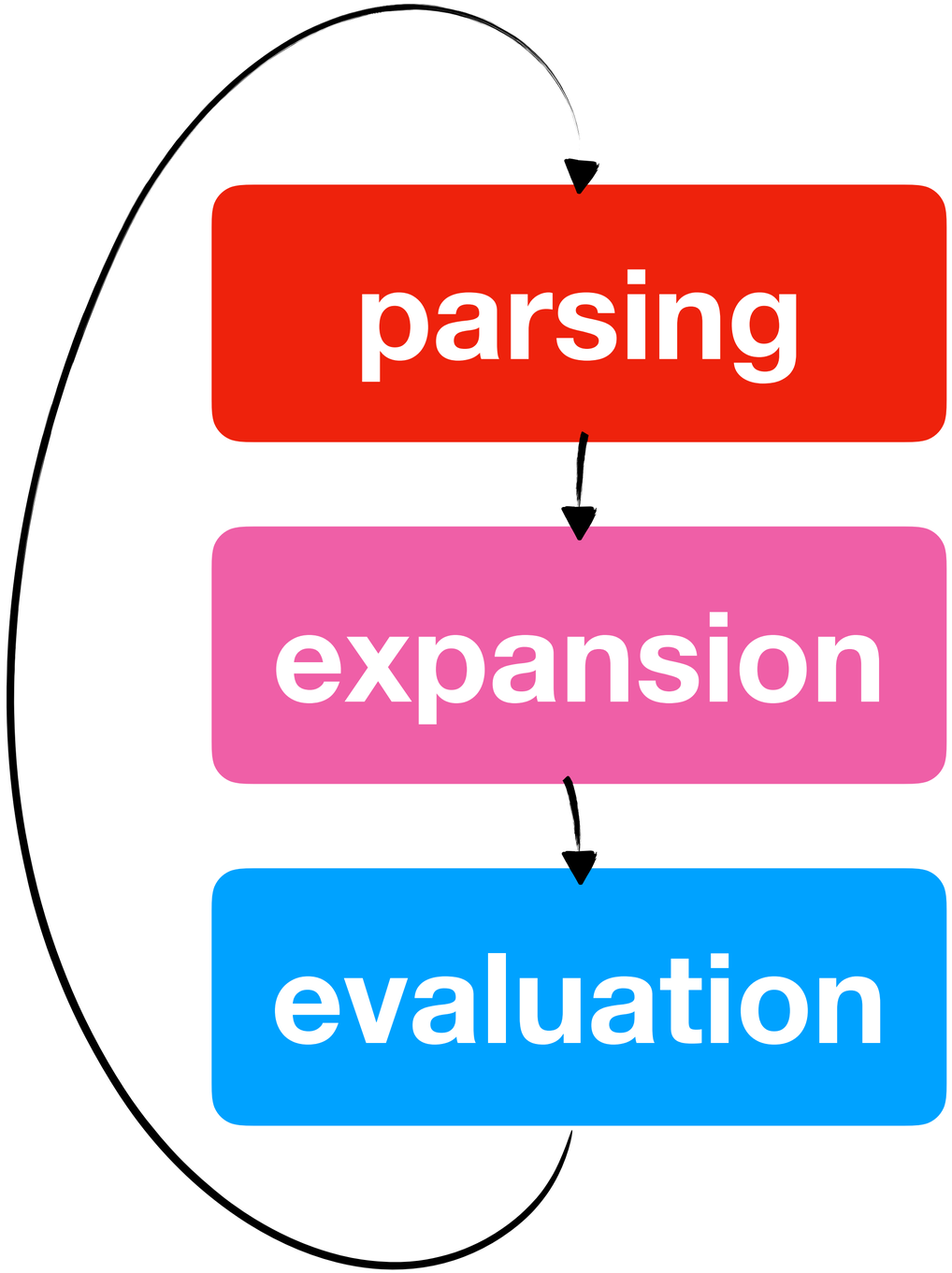
basename /Users/mgree/talks/smoosh



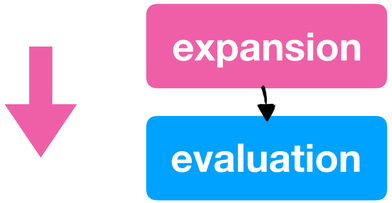
basename `pwd`

basename /Users/mgree/talks/smoosh

smoosh



basename `pwd`

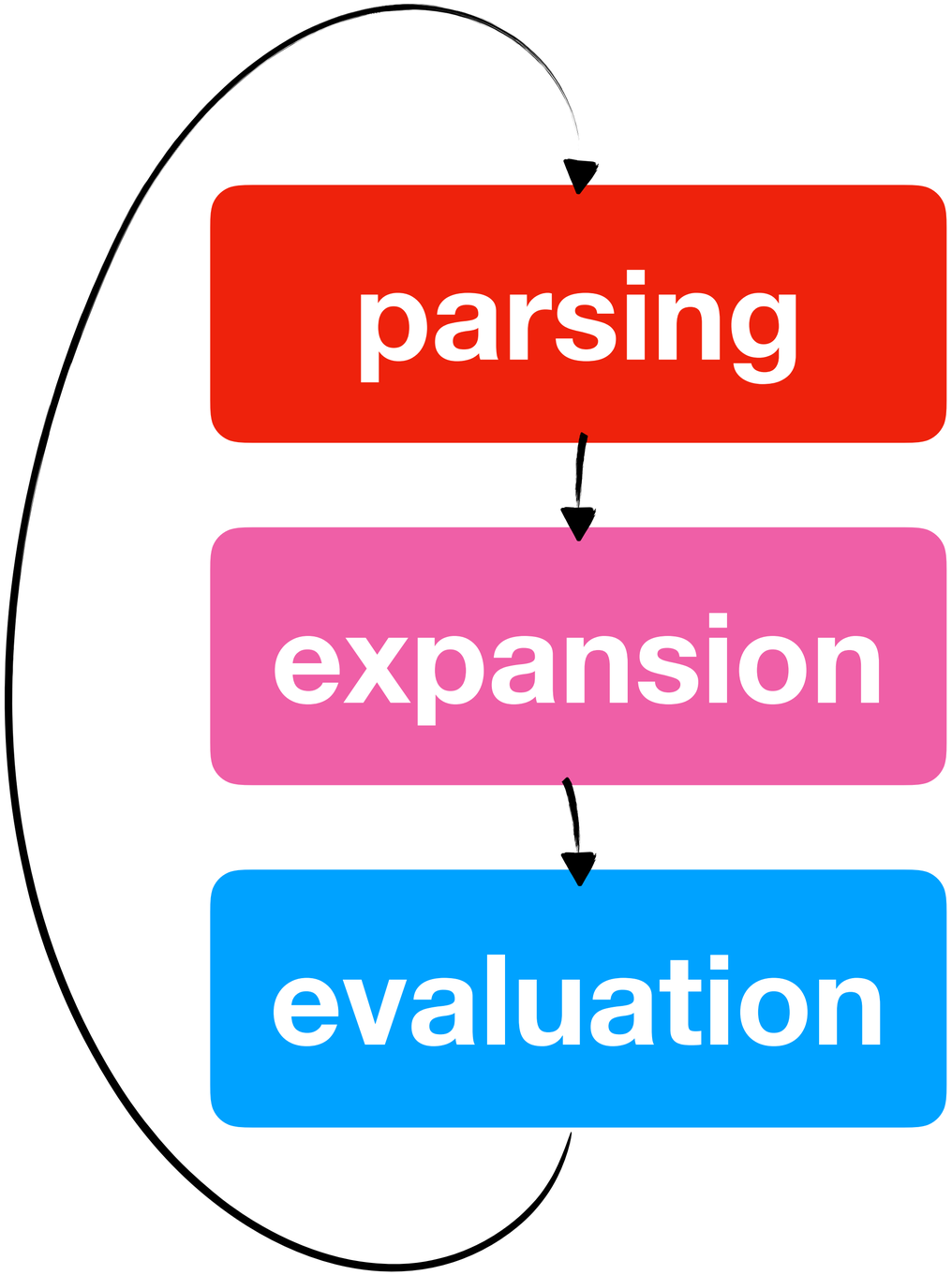


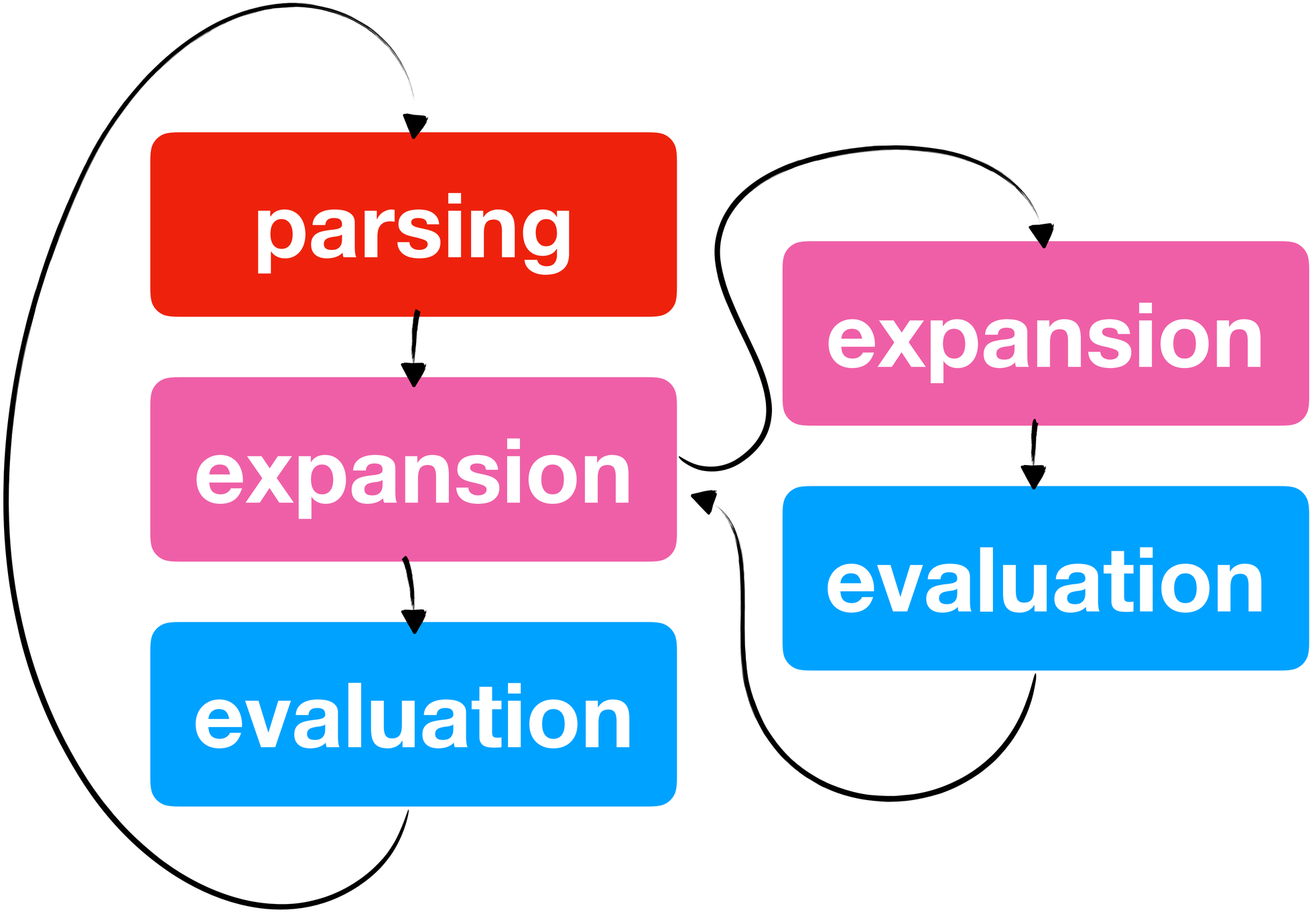
pwd

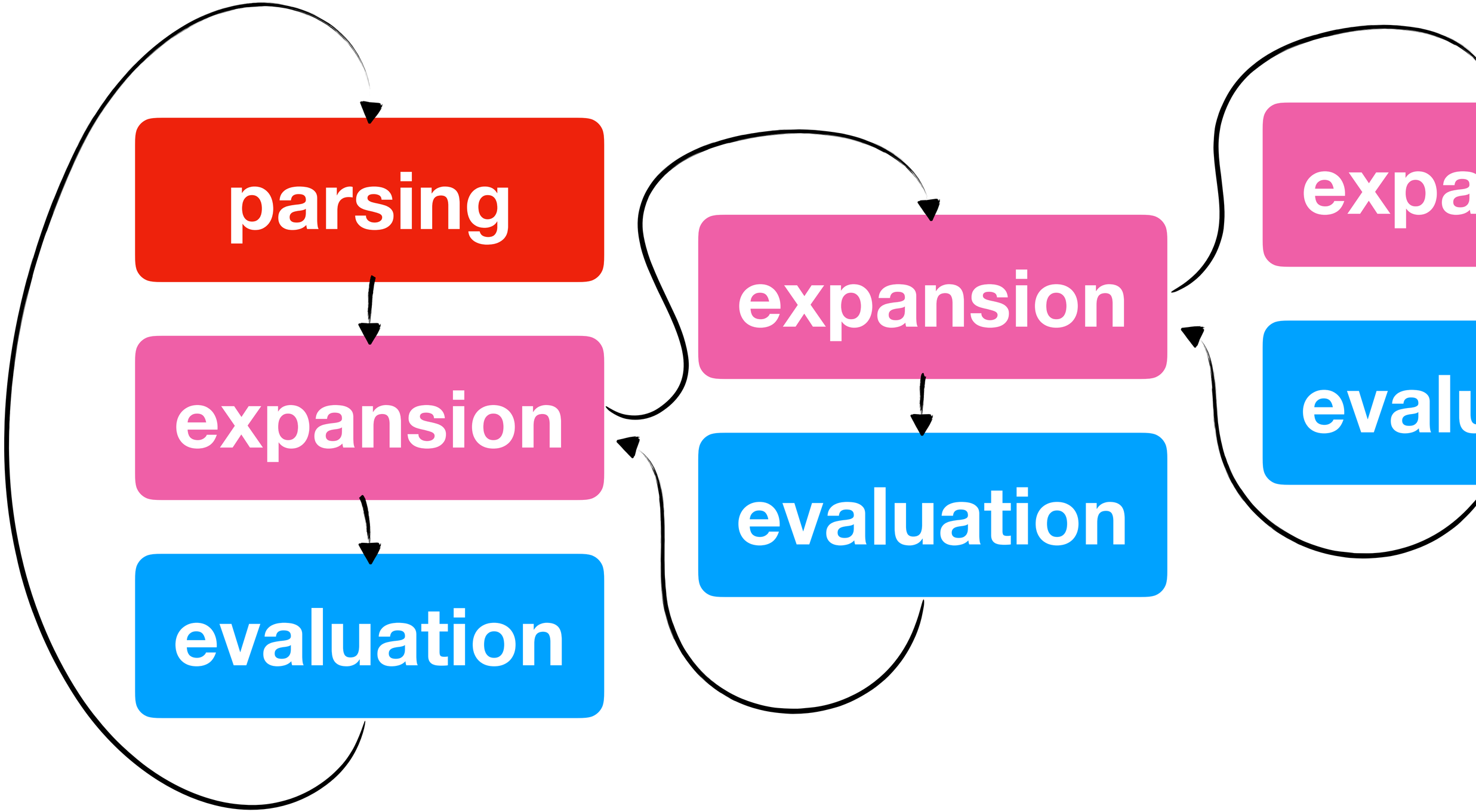
/Users/mgree/talks/smoosh

basename /Users/mgree/talks/smoosh

smoosh


parsing
expansion
evaluation


parsing
expansion
evaluation
expansion
evaluation


parsing
expansion
evaluation
expansion
evaluation
expa
evalu

the shtepper

http://shell.cs.pomona.edu/shtepper

c ::= **v=w ... w ...** | **c r**

| **c_1|c_2|c_3|...|c_n** | **c &** | **(c)**

| **c_1 && c_2** | **c_1 || c_2**

| **! c** | **c_1 ; c_2** | **if c_1 c_2 c_3**

| **switch a ... { case w...) c } ...**

| **while c_1 c_2** | **for x in w ... c**

| **defun v c**

c ::= **v=w ... w ...** | **c r**

| c_1|c_2|c_3|...|c_n | **c &** | **(c)**

| c_1 **&&** c_2 | c_1 **||** c_2

| **!** c | c_1 **;** c_2 | **if** c_1 c_2 c_3

| **switch a ... { case w...) c } ...**

| **while** c_1 c_2 | **for x in w ... c**

| **defun v c**

c ::= **v=w ... w ...** | **c r**

| **c_1|c_2|c_3|...|c_n** | **c &** | **(c)**

| **c_1 && c_2** | **c_1 || c_2**

| **! c** | **c_1 ; c_2** | **if c_1 c_2 c_3**

| **switch a ... { case w...) c } ...**

| **while c_1 c_2** | **for x in w ... c**

| **defun v c**

Words	w	::=	(s | k | ␣)*
Control codes	k	::=	~ | ~s | ${$s$\|$\phi$} | $($c$) | $(($w$)) | “w”
Parameter formats	ϕ	::=	normal | length | default w | …
Strings	s	∈	ASCII

$$c ::= \mathtt{v=w\ldots\ w\ \ldots} \mid \mathtt{c\ r}$$

$$\mid \mathtt{c_1 | c_2 | c_3 | \ldots | c_n} \mid \mathtt{c\ \&} \mid \mathtt{(\ c\ )}$$

$$\mid \mathtt{c_1\ \&\&\ c_2} \mid \mathtt{c_1\ ||\ c_2}$$

$$\mid \mathtt{!\ c} \mid \mathtt{c_1\ ;\ c_2} \mid \mathtt{if\ c_1\ c_2\ c_3}$$

$$\mid \mathtt{switch\ a\ \ldots\ \{\ case\ w\ldots)\ c\ \}\ \ldots}$$

$$\mid \mathtt{while\ c_1\ c_2} \mid \mathtt{for\ x\ in\ w\ \ldots\ c}$$

$$\mid \mathtt{defun\ v\ c}$$

Words	w	::=	(s | k | ␣)*
Control codes	k	::=	~ | ~s | ${$$s$\|$\phi$$}$ | $($$c$$)$ | $(($$w$$))$ | “w”
Parameter formats	ϕ	::=	normal | length | default w | …
Strings	s	∈	ASCII

a bunch of spaces

```
ls   ~/${x}/*.txt
```

c ::= **v=w ... w ... | c r**

| c_1|c_2|c_3|...|c_n | c & | (c)

| c_1 && c_2 | c_1 || c_2

| ! c | c_1 ; c_2 | if c_1 c_2 c_3

| switch a ... { case w...) c } ...

| while c_1 c_2 | for x in w ... c

| defun v c

Words	w ::=	(s | k | ␣)*
Control codes	k ::=	~ | ~s | ${$s$\|$\phi$} | $($c$) | $(($w$)) | “w”
Parameter formats	ϕ ::=	normal | length | default w | …
Strings	s ∈	ASCII

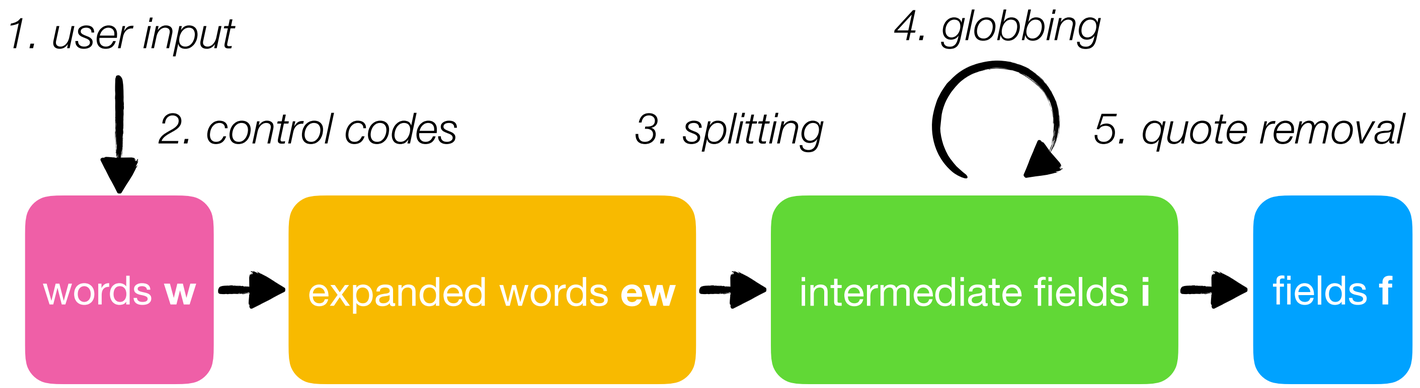
a bunch of spaces

ls ~/${x}/*.txt

s k s k s

ls ␣ ~ / ${x|normal} /*.txt

field #1 *field #2*



Words	*w*	::=	(*s* | *k* | ␣)*
Control codes	*k*	::=	~ | ~*s* | ${*s*|*ϕ*} | $(*c*) | $((*w*)) | “w”
Parameter formats	*ϕ*	::=	normal | length | default *w* | …
Strings	*s*	∈	ASCII/locale
Expanded words	*ew*	::=	(usr s | exp s | ␣ | @ f | “s”)*
Intermediate fields	*i*	::=	(ws ␣ | ␣ | *s* | “s”)*
Fields	*f*	::=	$s_1 \dots s_n$

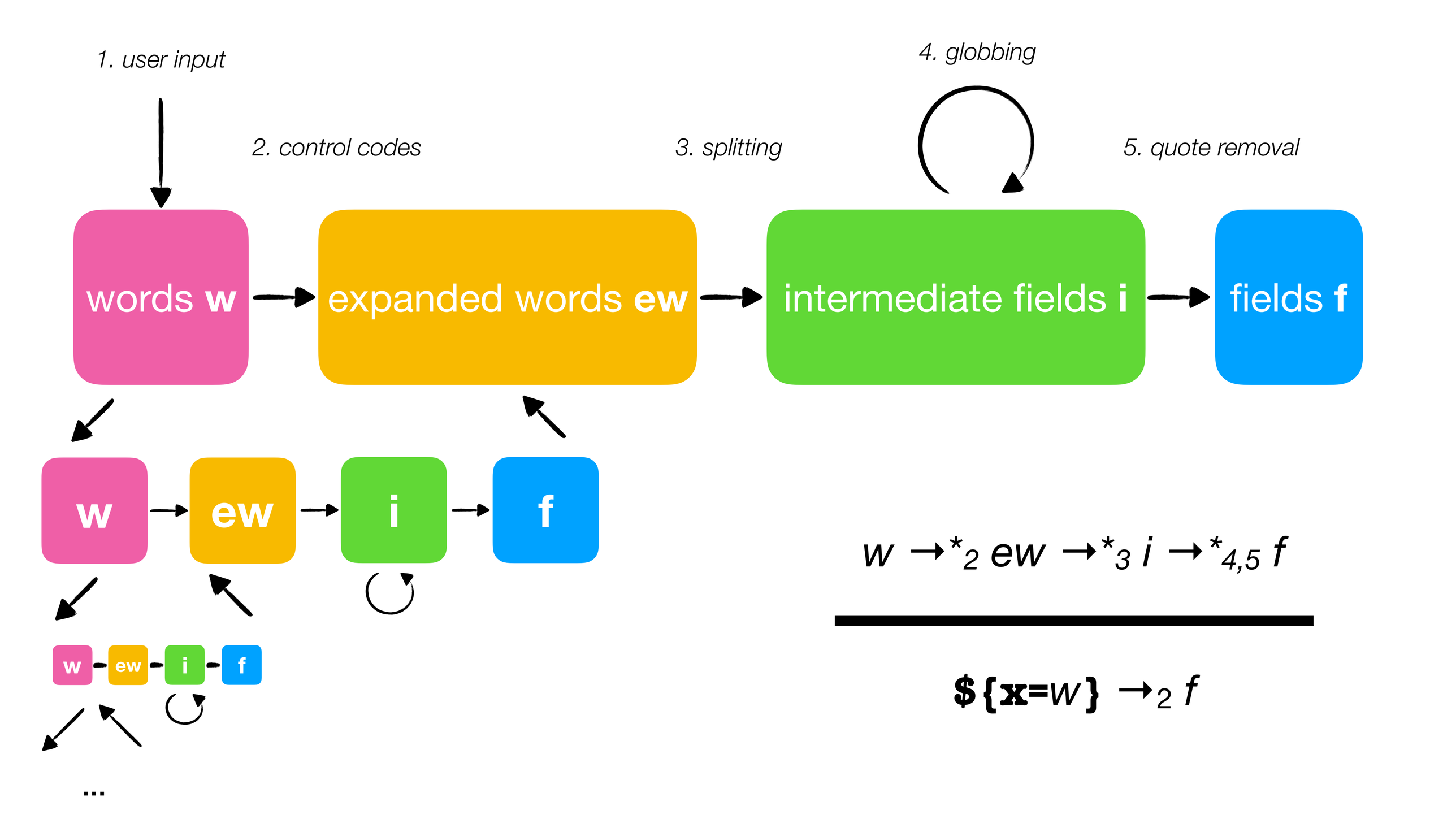
user input

*expanded, no splitting or **

*expanded, split and **

fully expanded

informative, not in the spec!



$$w \rightarrow^*_2 ew \rightarrow^*_3 i \rightarrow^*_{4,5} f$$

$$\texttt{\$\{x=}w\texttt{\}} \rightarrow_2 f$$

what's unique about the shell's semantics

- Modal: **expansion** and **evaluation**
- Lots of **mutual recursion**
- Language semantics uses **system calls**
- Runtime AST includes nested **command loops**
  e.g., `source` a/k/a `.` and `eval`; top level and `-i`

who cares about semantics?

- **lingua franca** for PL folks
- **symbolic execution**
  - stepper
  - bug finding, static analysis
  - support DevOps tools like Rehearsal
- **correctness** baseline for compilers, other tools
- insights into the **shell** and other **interactive languages**

comparing smoosh

	Smoosh	bash*	dash	zsh†	mksh	ksh	yash*
	0.1	4.4-12(1)	0.5.8-2.4	5.3.1-4+b2	54-2+b4	93u+	2.43-1
POSIX test suite (418 tests)							
Failing tests	0	4 (8)	20	×	35	23	22 (23)
Time to run	12m41s	2m43s	2m43s	×	2m52s	3m24s	2m45
Modernish's shell diagnosis (91 potential bugs, 22 potential quirks) and test suite (312 tests)							
Bugs	1	16	2	3	3	14	1
Quirks	0	4	2	5	2	3	8
Failing tests	1	20	3	3	3	17	1
Time to run	5.5s	4.8s	1.4s	1.2s†	3.2s	2.2s	2.4s
Smoosh's test suite (161 tests)							
Failing tests	0	30 (35)	42	52	34	41	43 (39)
Time to run	23s	28s	1m13s	42s	21s	28s	29s

comparing smoosh

	Smoosh	bash*	dash	zsh†	mksh	ksh	yash*
	0.1	4.4-12(1)	0.5.8-2.4	5.3.1-4+b2	54-2+b4	93u+	2.43-1
POSIX test suite (418 tests)							
Failing tests	0	4 (8)	20	×	35	23	22 (23)
Time to run	12m41s	2m43s	2m43s	×	2m52s	3m24s	2m45
Modernish's shell diagnosis (91 potential bugs, 22 potential quirks) and test suite (312 tests)							
Bugs	1	16	2	3	3	14	1
Quirks	0	4	2	5	2	3	8
Failing tests	1	20	3	3	3	17	1
Time to run	5.5s	4.8s	1.4s	1.2s†	3.2s	2.2s	2.4s
Smoosh's test suite (161 tests)							
Failing tests	0	30 (35)	42	52	34	41	43 (39)
Time to run	23s	28s	1m13s	42s	21s	28s	29s

comparing smoosh

	Smoosh	bash*	dash	zsh†	mksh	ksh	yash*
	0.1	4.4-12(1)	0.5.8-2.4	5.3.1-4+b2	54-2+b4	93u+	2.43-1
POSIX test suite (418 tests)							
Failing tests	0	4 (8)	20	×	35	23	22 (23)
Time to run	12m41s	2m43s	2m43s	×	2m52s	3m24s	2m45
Modernish's shell diagnosis (91 potential bugs, 22 potential quirks) and test suite (312 tests)							
Bugs	1	16	2	3	3	14	1
Quirks	0	4	2	5	2	3	8
Failing tests	1	20	3	3	3	17	1
Time to run	5.5s	4.8s	1.4s	1.2s†	3.2s	2.2s	2.4s
Smoosh's test suite (161 tests)							
Failing tests	0	30 (35)	42	52	34	41	43 (39)
Time to run	23s	28s	1m13s	42s	21s	28s	29s

who cares about semantics?

- 10 **bugs** in the POSIX test suite
- 3 **patches** for dash
- dozens of other, **uninvestigated** bugs

POSIX test suite bug: impatient pipes

```
(trap "" PIPE
  sleep 2
  echo “it’s alive” >out
) | true
[ $(cat out) = “it's alive” ]
```

POSIX test suite bug: impatient pipes

```
(trap "" PIPE
  sleep 2
  echo “it’s alive” >out
) | true
[ $(cat out) = “it's alive” ]
```

If the pipeline is not in the background (see Asynchronous Lists), the shell shall wait for the last command specified in the pipeline to complete, and **may** also wait for all commands to complete.

IEEE Std 1003.1-2017 §2.9.2

dash bug: unset arithmetic

```
unset x

[ $((x + 2)) -eq 2 ]
```

dash bug: unset arithmetic

```
unset x

[ $((x + 2)) -eq 2 ]
```

dash: 1: Illegal number:

my motivation

“So, what do you do?”

my motivation

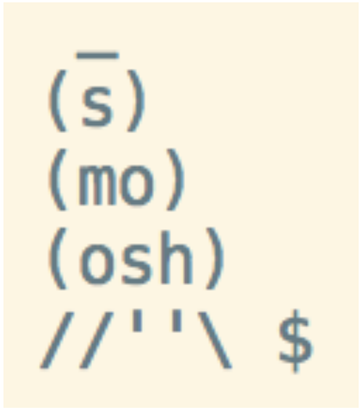
What happened?

smoosh

~11k loc	**shell semantics**	Lem, OCaml
~1k loc	**testing**	OCaml
~0.5 loc	**dash bindings**	C, OCaml

~5k loc	**AST, OS model**
~3k loc	**expansion and evaluation**
~2.5k loc	**builtin commands**

- **small-step** operational semantics
  - **expansion** and **evaluation** are mutually recursive
- semantics uses an **abstract OS interface**
  - **system** (real, working shell) or **symbolic** (fake OS)
- 1**.5 person years** of effort



Lem. Owens, Zappa Nardelli, Boehm, and Sewell 2016